This article charts the tractability frontier of two classes of relational algebra queries in tuple-independent probabilistic databases. The first class consists of queries with join, projection, selection, and negation, but without repeating relation symbols and union. The second class consists of quantified queries that express the following binary relationships amongst sets of entities: set division, set inclusion, set equivalence, and set incomparability. Quantified queries are expressible in relational algebra using join, projection, nested negation, and repeating relation symbols.

Each query in the two classes has either polynomial-time or \#P-hard data complexity and the tractable queries can be recognised efficiently. Our result for the first query class extends a known dichotomy for conjunctive queries without self-joins to such queries with negation. For quantified queries, their tractability is sensitive to their outermost projection operator: They are tractable if no attribute representing set identifiers is projected away and \#P-hard otherwise.

Categories and Subject Descriptors: H.2.4 [Database Management]: Query Processing; G.2.1 [Combinatorics]: Counting Problems

General Terms: Algorithms, Theory

Additional Key Words and Phrases: complexity dichotomy, knowledge compilation, probabilistic databases

ACM Reference Format:
Robert Fink and Dan Olteanu. 2015. Dichotomies for queries with negation in probabilistic databases. ACM Trans. Datab. Syst. 1, 1, Article 1 (October 2015), 45 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Charting the tractability frontier of query evaluation lies at the foundation of probabilistic databases. The probabilistic database systems MystiQ [Dalvi and Suciu 2007a] and MayBMS/SPROUT [Huang et al. 2009] distinguish between tractable and intractable queries at compile time and provide exact evaluation techniques for tractable queries at the speed of relational databases and approximate techniques for intractable queries. The relevance of such tractability results goes beyond probabilistic databases: The problems of tractable query evaluation in probabilistic databases and of domain-lifted inference for weighted first-order model counting [den Broeck 2011], which is actively investigated by the AI community, essentially coincide [Gribkoff et al. 2014]

Complexity dichotomies have been established for several classes of relational queries in probabilistic databases: Any query in such a class is either tractable or intractable, that is, its data complexity is either polynomial time or \#P-hard. Such dichotomies are known for conjunctive queries without repeating relation symbols [Dalvi and Suciu 2007a] and their extension to ranking [Olteanu and Wen 2012], and for unions of conjunctive queries [Dalvi and Suciu 2012]. This article complements these results with dichotomies for two classes of queries with negation. These dichotomies has been announced previously in extended abstracts [Fink et al. 2013] [Fink and Olteanu 2014].
The two query classes considered in this article are fragments of relational algebra without union. The first class is denoted 1RA and consists of queries with equi-join, projection, selection, and negation, which is expressed using the difference operator, but without repeating relation symbols. Examples of 1RA queries are given in Figs. 1 and 3. The second class consists of the set-division, set-inclusion, set-equivalence, set-incomparability quantified queries. In contrast to 1RA queries, quantified queries can only be expressed in relational algebra using repeating relation symbols, cf. Figs. 5 and 15. The two classes can also be mixed by allowing 1RA queries in place of relations in quantified queries.

The tractable queries from both classes admit efficient syntactic recognition. This is convenient for practical reasons: By just inspecting the query, the query optimiser of a probabilistic database management system can quickly decide whether to employ efficient exact inference in case of tractable queries, or approximate inference otherwise.

The probabilistic database model considered in this article is that of tuple-independent databases, where each tuple in the database is an independent probabilistic event. Prime examples of tuple-independent probabilistic databases are the knowledge bases from Google Knowledge Vault [Dong et al. 2014] and NELL [Carlson et al. 2010]. For more complex probabilistic models, query tractability is quickly lost: for block-independent disjoint tables consisting of independent groups of mutually exclusive tuples, tractability essentially falls back to that for tuple-independent databases by restricting joins to attributes representing group keys, while for the general model of probabilistic c-tables, simple selection and projection queries can be #P-hard [Suciu et al. 2011].

The following theorem states our first dichotomy result:

**Theorem 1.1.** 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 for a 1RA query $Q$. 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 $X(A_1)$ and $Y(A_2)$, both the join $X \bowtie_{A_1=A_2} Y$ and the difference $X - A_1 \leftrightarrow A_2 Y$ under the attribute mapping $A_1 \leftrightarrow A_2$ enforce that $[A_1] = [A_2]$.

**Definition 1.2.** A 1RA query $Q$ is hierarchical if for every pair of attribute classes $[A]$ and $[B]$ without attributes in $Q$’s result or in selections with equality conditions, 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 $[A]$.

**Example 1.3.** Figure 1 depicts the non-hierarchical query $\pi_0[X \bowtie (R - \pi_A(T \bowtie S))]$ over schema $(X(A), R(A), T(B), S(A, B))$. The query $\pi_0(R \bowtie S \bowtie T)$ over the same schema is the classical example of a non-hierarchical query without negation [Suciu et al. 2011]. Non-Boolean versions of these queries are hierarchical. Figure 3 depicts the hierarchical query $\pi_0[R \times T - U \times V]$ over schema $(R(A), U(A), T(B), V(B))$.

Hierarchical 1RA queries can be recognised in LOGSPACE. The hierarchical property plays a central role in studies with seemingly disparate focus. All join queries that admit parallel evaluation with one broadcast step in the Massively Parallel Communication model are hierarchical [Koutris and Suciu 2011]. The results of any hierarchical conjunctive query in relational databases admit lossless factorised representations that are at most linear in the size of the input databases [Olteanu and Závodný 2015]. In the finite cursor model, the hierarchical queries are exactly those semi-join algebra queries with one-step streaming evaluation [Grohe et al. 2009]. In provenance databases, the hierarchical queries are exactly those conjunctive queries with provenance polynomials of bounded readability [Olteanu and Závodný 2012]. The hierarchical queries are exactly those non-repeating conjunctive queries that are tractable in probabilistic databases [Dalvi and Suciu 2007a]. A key contribution of

ACM Transactions on Database Systems, Vol. 1, No. 1, Article 1, Publication date: October 2015.
this article is to understand the connection between the hierarchical property and negation in probabilistic databases. Theorem 1.1 states that the hierarchical property partitions the query language 1RA− into tractable and hard queries, thereby lifting the dichotomy for non-repeating conjunctive queries [Dalvi and Suciu 2007a] to queries with negation. In Section 7, we discuss difficulties of extending this result to non-repeating relational calculus with negation and to non-repeating relational algebra with union.

The tractability and hardness proofs for 1RA− are non-trivial generalisations of those for queries without negation. 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 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 defined by the interaction between difference and join operators. In contrast, for queries without negation there is a single minimal hard pattern and it requires two probabilistic relations. We next exemplify techniques used in the hardness and tractability proofs.

**#P-hardness of non-hierarchical queries**

We prove that every non-hierarchical 1RA− query $Q$ has #P-hard data complexity by reduction from the #P-hard model-counting problem for positive bipartite DNF formulas: Given such a formula $\Psi$ and the query $Q$, for most reductions used in this article we construct an input database whose input tuples are annotated with variables in $\Psi$ such that the result of $Q$ becomes annotated with $\Psi$ or $\neg\Psi$. 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 of $\Psi$ is then $2^n P_Q$ or $2^n(1 - P_Q)$, where $n$ is the number of variables in $\Psi$. The query evaluation problem is not technically in #P since it is not a counting problem, cf. [Suciu et al. 2011] (page 47) for a detailed discussion.

The starting point of our analysis is an alternative characterisation of the hierarchical property via minimal hard patterns: A query is not hierarchical exactly when it matches such a pattern [Fink and Olteanu 2014]. There is a pattern for each possible binary tree with leaves $A$, $AB$, and $B$, and with inner nodes join and difference operators (48 in total).
A query matches a pattern if there is a total mapping of the nodes of the pattern to nodes in the parse tree of the query such that: (1) The join and difference operators in the pattern are mapped to join and respectively difference operators in the query; (2) the leaves $A$, $AB$, and $B$ are mapped to relations $R$, $S$, and respectively $T$ as in Definition 1.2 and (3) parent-child edges in the pattern are mapped to ancestor-descendant edges in the query.

Example 1.4. For each query, we craft a specific reduction depending on which pattern is matched. For example, the query $Q_{\text{hard}}$ in Fig. 1 (right) is not hierarchical as it matches the pattern in Fig. 1 (left). We exemplify the reduction for formula $\Psi = x_1 y_1 \lor x_1 y_2$, where we consider each variable random and with probability $1/2$.

We populate the relations $R$, $S$, $T$, and $X$ as shown in Fig. 2. The relations $R$ and $X$ consist of tuples representing the indices 1 and 2 of the clauses in $\Psi$ and annotated by the Boolean constant true ($\top$). The relation $S$ lists all pairs of the index of a clause and variable in that clause; these tuples are also annotated by $\top$. Finally, the relation $T$ lists all variables occurring in $\Psi$, where each tuple for variable $z$ is annotated by $\neg z$. In our encoding, we may use variable names as constants, e.g., the values of the attribute $B$ in relations $R$ and $T$.

The probabilistic database $(R, X, S, T)$ represents a finite set of possible database instances, with each instance defined by a total assignment of the variables in the annotation columns $\Phi$ [Suciu et al. 2011]. The instance defined by a variable assignment consists of those tuples whose annotations are satisfied by the assignment. For instance, the assignment mapping all variables occurring in $\Psi$, where each tuple for variable $z$ is annotated by $\neg z$. In our encoding, we may use constant values as constants, e.g., the values of the attribute $B$ in relations $R$ and $T$.

Figure 2 also depicts the intermediate results during the computation of $Q_{\text{hard}}$. Whereas the input relations are tuple-independent, the intermediate results exhibit correlated annotations. These annotations are Boolean formulas over the annotations of the input relations [Green et al. 2007]: a join (projection) of tuples is annotated by the conjunction (respectively, disjunction) of their annotations, and a difference of two tuples is the conjunction of the annotation of the first tuple and the negation of the second tuple. The query result is the projection on the empty set of the bottom-right relation; the annotation associated with this nullary result tuple is $\Psi$.

Example 1.4 shows the power of negation: Our query $Q_{\text{hard}}$ can compute $\#\Psi$ for any positive 2DNF formula $\Psi$ and is thus #P-hard already when one of its relations is probabilistic (here, $T$) while all other relations are deterministic. In contrast, hardness can only be achieved for queries without negation when at least two input relations are probabilistic.

The key challenge in the hardness reductions from Section 5 is to identify three relations $(R, S, T)$ that establish the match of a non-hierarchical query with one of the minimal hard patterns, and to populate them such that the annotation of the query result is the input positive 2DNF formula $\Psi$. The remaining relations ($X$ in Example 1.4) are populated such that they do not influence the interaction between the annotations of $R$, $S$, and $T$. The reductions put forward in this article vary substantially and there is no one unifying reduction for all the minimal hard patterns. Indeed, some reductions require a single relation to be probabilistic, while others require two. Furthermore, some reductions populate a single probabilistic database (such as in the above example), while others require to populate a number of databases linear in the size of the input formula $\Psi$.

**Efficient evaluation algorithm for hierarchical queries**

We evaluate a hierarchical 1RA$^-$ query $Q$ in five steps: (1) We translate $Q$ into an equivalent relational calculus expression $Q_{RC}$ that is further rewritten into a disjunction of disjunction-free existential relational calculus expressions by pushing down negation and existential quantifiers; (2) we compute the formulas representing the annotations of the results of $Q_{RC}$'s disjuncts; (3) we compile each such formula into an ordered binary decision diagram
(OBDD); (4) we compute an OBDD representing the disjunction of the OBDDs from step (3); (5) and finally, we compute the probability of the OBDD from step (4). Resorting to OBDDs for query evaluation in probabilistic databases is not new [Olteanu and Huang 2008; Jha and Suciu 2013]. While for arbitrary queries the OBDDs may be exponential in the size of their annotations and thus of the input database, we show in Section 5 that those from step (5) can be computed in time polynomial in the input database. Since OBDDs admit linear-time probability computation [Wegener 2004], we obtain an overall query evaluation algorithm with polynomial-time data complexity. While the OBDD sizes are independent of the query size and linear in the database size for hierarchical non-repeating conjunctive queries [Olteanu and Huang 2008], they remain linear in the database size, but may depend exponentially on the query size, for hierarchical IRA queries. The exponential dependency on the query size arises due to the query rewriting and the OBDD construction steps.

Example 1.5. Consider the hierarchical query \( Q_{\text{easy}} \) and the database \( D \) from Fig. 3.

The formula annotating \( Q_{\text{easy}} \)'s result is

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

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, that is, a formula where each variable appears once; 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 \( Q_{\text{easy}} \) on \( D \) admits an OBDD of size \( O(|D| \cdot f(Q_{\text{easy}})) \), where \( f(Q_{\text{easy}}) \) is the OBDD width (cf. Section 2.3) and only depends on the query size \( |Q_{\text{easy}}| \). The underlying idea is to translate \( Q_{\text{easy}} \) into an equivalent disjunction of disjunction-free existential relational calculi 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 \( \text{RC}^3 \). For \( Q_{\text{easy}} \), this translation yields the query

\[
Q_{\text{RC}} = \exists_A (R(A) \land \neg U(A)) \land \exists_B T(B) \lor \exists_A R(A) \land \exists_B T(B) \land \neg V(B).
\]

The formulas annotating the results of \( Q_1 \) and \( Q_2 \) on the database \( D \) from Fig. 3 are

\[
\Psi_1 = (r_1 \land \neg u_1 \lor r_2 \land \neg u_2) \land (t_1 \lor t_2) \quad \Psi_2 = (r_1 \lor t_2) \land (t_1 \land \neg v_1 \lor t_2 \land \neg v_2),
\]

and clearly \( \Psi_1 \lor \Psi_2 = \Psi \). The queries \( Q_1 \) and \( Q_2 \) can be written such that (i) for each quantifier \( \exists_X(Q) \) every relation symbol in \( Q \) contains the 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 of Fig. 4. Property (ii) implies that these OBDDs have the same global variable order, which enables efficient computation of their conjunctions, disjunctions, and negation [Wegener 2004]. □
Quantified queries for reasoning about sets

The study of tractability for queries with repeating relation symbols raises additional challenges due to the interaction between copies of the same relation and to query containment. The language of unions of conjunctive queries (with repeating relation symbols) admits a complexity dichotomy, though there is no syntactic characterisation of tractable queries in this language: instead, there is an algorithm that runs in polynomial time for all tractable unions of conjunctive queries [Dalvi and Suciu 2012].

In Section 6 we investigate the class of so-called quantified queries that are expressible in an extension of 1RA queries with repeating relation symbols. They express binary relationships amongst sets of items encoded in relations. We consider set division, set inclusion, set equality, set difference, and set incomparability. Their definitions in relational algebra are given in Figs. 5 and 15. Each quantified query stands for a set of queries if we allow the input relations to be replaced by hierarchical 1RA queries whose results are tuple-independent probabilistic relations.

Given a relation $S(sid, item)$ encoding an arbitrary number of sets and their items, the set inclusion query $S \subseteq$ returns the pairs of set identifiers $sid_1$ and $sid_2$ such that all items of $sid_1$ are also items of $sid_2$; the result of the set equality query $S = $ consists of those pairs of sets that consist of the same items; the result of the set incomparability query $S \!<\!>$ consists of those pairs of sets such that none is contained in the other. Given the above relation $S(sid, item)$ and a second relation $I(item)$ consisting of items, the set division query $S \div I$ returns those sets that contain all items in $I$. Such queries are used in decision support applications, such as insurance and healthcare [Rao et al. 1996] or data mining applications [Rantzau 2004].

Similar to 1RA queries, the tractable quantified queries admit efficient syntactic recognition. In particular, the queries are tractable if they retain in the result the attributes for set identifiers. If some of these attributes are projected away, then they become #P-hard.

Example 1.6. Consider a tuple-independent database consisting of a supplier relation $S$ with two columns $sid$ for supplier key and $pid$ for product key, and a product relation $I$ with only a product item key $pid$ that contains keys of all product items of a given brand (cf. Fig. 5). Then the set division query $S \div P$ returns suppliers that supply all products of this brand. The query can be expressed as shown in Fig. 5. In a deterministic setting, the result would only consist of the tuple with $sid 1$, since this is the only $sid$ paired in $S$ with
We explain the annotation for sid 1 by following the structure of the query. The projection \(\pi_{sid}(S)\) corresponds to the formula \(x_1 \lor \cdots \lor x_4\): Indeed, sid 1 can be part of the result if at least one of these variables is true. The subsequent cross product with I generates the tuple \((1, i)\) with annotation \((x_1 \lor \cdots \lor x_4)y_i\) for \(1 \leq i \leq 3\). The difference with S keeps all tuples in the product and their annotations are extended by the negation of the annotation of the corresponding tuples in S: In case of tuple \((1, i)\), its annotation becomes \((x_1 \lor \cdots \lor x_4)y_i \neg x_i\). The next projection yields the disjunction of all annotations for tuples \((1, 1), (1, 2), \) and \((1, 3): (x_1 \lor \cdots \lor x_4)[y_1 \neg x_1 \lor y_2 \neg x_2 \lor y_3 \neg x_3]\). The outermost difference produces the annotation in Fig. 5. The final annotation reads as follows: sid 1 is in the result provided at least one of the tuples with sid 1 is present in S, and if pid = i is present in I, then the tuple \((1, i)\) must be present in S. We show in Section 6 that the probability of this annotation can be computed in time linear in the size of the input database. However, the Boolean version of the division query, namely \(\pi_S(S \div I)\) is \#P-hard.

Fig. 5. Supplier S, Item I, Set division \(S \div I = \pi_{sid}(S) - \pi_{sid}(\pi_{sid}(S) \bowtie I - S)\).

all pids in I. In a probabilistic setting, the third tuple from I can be absent with probability 
\(1 - P_{y_3}\), where \(P_{y_3}\) is the probability that this tuple is present. In that case, sid 2 can be 
in the result of \(S \div I\), since sid 2 is paired in S with pids 1 and 2 but not with 3. The set 
of instances that witness \(sid \in \{1, 2\}\) in the result is defined by the annotation expressions 
on the input random variables given in column \(\Phi\).

We explain the structure of the query. The relational algebra subset 1RA− consists of queries composed of:

- Non-repeating 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 \( A_1, \ldots, A_n \). We also use \( \pi_{-B_1, \ldots, -B_m} \) as shorthand for discarding the attributes in the class \( [B] = \{B_1, \ldots, B_m\} \);
— Difference: \( Q_1 -_\rho Q_2 \), where \( Q_1 \) and \( Q_2 \)’s results are over schemas \( \{A_1, \ldots, A_n\} \) and \( \{B_1, \ldots, B_n\} \) respectively, and \( \rho \) is the attribute mapping \( (A_1 \leftrightarrow B_1) \land \cdots \land (A_n \leftrightarrow B_n) \);
— Selection: \( \sigma_{A \theta c} \), where \( A \) is an attribute, \( c \) a constant, and \( \theta \) an arithmetic comparison.

Without loss of generality, we only consider in the sequel 1RA\(^-\) queries without selections: Selections can be resolved prior to the development put forward in this article, since their results on tuple-independent relations are also tuple-independent. We recall from Definition 1.2 that in case of an equality condition \( A = c \), we can safely ignore the attribute class \( [A] \) when checking the hierarchical property since \( A \) can only take one value.

In \( Q_1 \triangleright_{\rho} Q_2 \) and \( Q_1 -_{\rho} Q_2 \), we write \( A \in \rho \) to express that \( \rho \) contains an equality or mapping on \( A \), and \( (A=A') \in \rho \) or \( (A\leftrightarrow A') \in \rho \) to express that \( \rho \) contains the equality \( A=A' \) or mapping \( A\leftrightarrow A' \), respectively. When no confusion arises, we choose a schema with suggestive unique attribute names like \( R(A_1), S(A_s, B_s), T(B_1) \) and then write the queries \( R \equiv_{A_s=A, S} S \) and \( (R \bowtie T) - A_s \leftrightarrow A_s \land B_s \leftrightarrow B_s \) more concisely as \( R \equiv S \) 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: \( pol(Q, Op) = 1 \) if \( Op \) has odd polarity in \( Q \), and 0 otherwise. The bottom join operator in \( Q_{hard} \) from Fig. 4 has polarity 1; for query \( Q \) in Fig. 7, the joins on the leftmost path in have polarity 0, the join of \( V \) and \( X \) has polarity 1, and relation \( S \) has polarity 2.

The equivalence class \( [A] \) of an attribute \( A \) in \( Q \) consists of \( A \) and all attributes made equal or mapped to \( A \) in \( Q \).

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

\[
\mathcal{E}(Q_1 \triangleright_{\rho} Q_2) = \mathcal{E}(Q_1) \cup \mathcal{E}(Q_2) \quad \mathcal{E}(Q_1 -_{\rho} Q_2) = \mathcal{E}(Q_1)
\]

\[
\mathcal{E}(\pi_{A_1, \ldots, A_n}(Q)) = \{A_1, \ldots, A_n\} \quad \mathcal{E}(\pi_{-B}(Q)) = \mathcal{E}(Q) - [B] \quad \mathcal{E}(R) = \text{sch}(R).
\]

A query \( Q \) exports \( [A] \) if there exists \( A' \in [A] \) such that \( A' \notin \mathcal{E}(Q) \). Conversely, \( Q \) does not export \( [A] \) if for all \( A' \in [A] \) it holds that \( A' \notin \mathcal{E}(Q) \). By \( Q^{[A]}, Q^{[B]}, \) and \( Q^{[A] \setminus [B]} \) we denote a query \( Q \) that exports \( [A] \), does not export \( [B] \), and respectively exports \( [A] \) and not \( [B] \). Using this notation, the triple of relations used to disprove the hierarchical property in Definition 1.2 is \( (R^{[A] \setminus [B]}, T^{[B] \setminus [A]}, S^{[A] \setminus [B]}) \) for distinct attributes \( A \) and \( B \).

### 2.2. The relational calculus subset \( RC^3 \)

Our query tractability results in Section 3 make use of standard translation between the relational algebra subset 1RA\(^-\) and the relational calculus subset called \( RC^3 \). The latter consists of queries \( \{H \mid F\} \), where the query head \( H \) is the set of query variables that occur unquantified in the query body \( F \), and \( F \) is a formula defined by the following grammar:

\[
F ::= R(X_1, \ldots, X_n) \mid \exists_X (F_1) \mid F_1 \land F_2 \mid F_1 \lor F_2 \mid \neg(F_1),
\]

The size \( |Q| \) of a query \( Q \) is the number of its relation symbols.

### 2.3. Propositional formulas and their compilation into decision diagrams

Propositional formulas are essential to the probabilistic database formalism used in this article. We next review the necessary vocabulary from their syntax to (probabilistic) semantics, and conclude with their compilation to ordered binary decision diagrams.

**Syntax.** Let \( \mathbf{X} \) be a finite set of variable symbols. A literal is a variable or its negation. A clause is a conjunction of literals. A formula can be constructed using variables and
constants $\top$ (true) and $\bot$ (false) using the logical connectives $\vee$ (or), $\wedge$ (and), and $\neg$ (not). We denote by $B(X)$ the set of propositional formulas over variables $X$. A formula is positive if it contains only positive literals. A formula is in *disjunctive normal form* (DNF) if it is a disjunction of clauses. Given two disjoint sets of variables, $X$ and $Y$, a DNF formula is bipartite over $X$ and $Y$ if each clause has the form $x \wedge y$ with variables $x \in Y$ and $y \in Y$. The set of positive bipartite DNF formulas is denoted by $2\text{DNF}$. A convenient way of representing a 2DNF formula is by labelling the variables by natural numbers, that is, $x_1, x_2, \ldots$ and $y_1, y_2, \ldots$, and representing each clause by a pair $(i, j) \in \mathbb{N} \times \mathbb{N}$. A set $E \subseteq \mathbb{N} \times \mathbb{N}$ of such pairs then defines the formula $\Psi = \bigvee_{(i, j) \in E} x_i y_j$.

**Semantics.** Given the set $X$ of variables, we denote by $I$ the set of possible assignments of all variables from $X$ to constants $\top$ and $\bot$. For a formula $\Psi$, its set of assignments is denoted by $I(\Psi) = \{ I : \text{vars}(\Psi) \rightarrow \{ \top, \bot \} \}$. A *model of* $\Psi$ is a satisfying assignment, that is, an assignment $I$ that maps $\Psi$ to $\top$, also denoted by $I \models \Psi$. The set of models of $\Psi$ is denoted by $M(\Psi)$. Counting the number of models (determining the number $\#\Psi = |M(\Psi)|$) is already $\#P$-hard for 2DNF [Provan and Ball 1983].

**Probabilistic interpretation.** Let now $X$ be a set of independent random and variables. For each variable $x \in X$, let $P_x$ be the probability of $x$ being true; we assume $P_x > 0$ without loss of generality. The probability mass function $\Pr(I) = \prod_{x \in X} P_x^{I_x} \cdot \prod_{x \in X} (1 - P_x)^{\neg I_x}$ for each assignment $I \in I$ and the probability measure $\Pr(E) = \sum_{I \in E} \Pr(I)$ for all $E \subseteq I$ define the probability space $(I, 2^I, \Pr)$ that we call the *probability space induced by* $X$.

A formula $\Psi$ over random variables is itself a random variable $\Psi : I \rightarrow \{ \top, \bot \}$ over $(I, 2^I, \Pr)$ by letting $\Psi : I \rightarrow I(\Psi)$ and with probability distribution defined as

$$\Pr(\Psi = \top) = \sum_{I \in I, I(\Psi) = \top} \Pr(I).$$

We write $P_\Psi$ or $P(\Psi)$ for $\Pr(\Psi = \top)$ and $P_{\neg \Psi}$ or $P(\neg \Psi)$ for $\Pr(\Psi = \bot) = 1 - \Pr(\Psi = \top)$. If $P_x = 1/2$ for each variable $x$, the model counting problem reduces to the probability computation problem: $P_\Psi = 2^{-|\text{vars}(\Psi)|} \#\Psi$, and the latter problem is $\#P$-hard for 2DNF.

**Binary decision diagrams (BDDs).** BDDs form a representation system for Boolean propositional formulas. 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 $\top$ (true) and $\bot$ (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 use of the following results on OBDDs:

**Lemma 2.1** [Wegener 2004]. Given an OBDD for a formula $\Psi$, the probability $P_\Psi$ can be computed in time linear in the size of the OBDD.

Let $\Phi_1, \Phi_2$ be two formulas, $\Pi$ a fixed variable order on their variables, and $O_1$ and $O_2$ be $\Pi$-OBDDs of width $w_1$ and $w_2$ for $\Phi_1$ and $\Phi_2$, respectively. Then, $\Pi$-OBDDs for $\Phi_1 \land \Phi_2$ and $\Phi_1 \lor \Phi_2$ can be constructed in time $O(|O_1| \cdot |O_2|)$ and have width at most $w_1 \cdot w_2$.

1 A different notion of BDD width refers to the maximum number of nodes in any level.
Example 2.2. Figure 4 shows three OBDDs with the same variable order \((r_1, u_1, r_2, u_2, t_1, v_1, t_2, v_2)\). The path \(r_1 \rightarrow \neg u_1 \rightarrow t_2 \rightarrow \bot\) encodes that under any truth assignment \(\nu\) with \(\nu(r_1) = \top\) and \(\nu(\neg u_1) = \bot\), the expression \(\Psi_1 = (r_1 \lor \neg u_1) \land (t_1 \lor \neg t_2)\) 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 in Fig. 4 represents the disjunction of the two leftmost OBDDs and has width five. Intuitively, a disjunction of two OBDDs is computed in a top-down OBDD with children computed recursively as the disjunctions of the OBDDs rooted at the children of the input OBDDs accessed by following the solid and respectively dotted edges. We choose the next node for the output OBDD following the common global variable order.

We can compute the probability of a BDD in one bottom-up pass. We exemplify for the OBDD of \(\Psi_1\), where by \(P(\alpha \beta)\) we denote the probability at node \(\alpha\):

\[
\begin{align*}
P(\alpha t_2) &= P_{t_2} \cdot P_{\top} + P_{\neg t_2} \cdot P_{\bot} = P_{t_2} \\
P(\alpha t_1) &= P_{t_1} \cdot P_{\top} + P_{\neg t_1} \cdot P(\alpha t_2) = P_{t_1} + (1 - P_{t_1}) \cdot P(\alpha t_2) \\
P(\alpha u_2) &= P_{u_2} \cdot P(\alpha t_1) + (1 - P_{u_2}) \cdot P_{\bot} = (1 - P_{u_2}) \cdot P(\alpha t_1) \\
P(\alpha r_2) &= P_{r_2} \cdot P(\alpha u_2) + (1 - P_{r_2}) \cdot P_{\bot} = P_{r_2} \cdot P(\alpha u_2) \\
P(\alpha u_1) &= P_{u_1} \cdot P(\alpha t_1) + (1 - P_{u_1}) \cdot P(\alpha r_2) \\
P(\alpha r_1) &= P_{r_1} \cdot P(\alpha u_1) + (1 - P_{r_1}) \cdot P(\alpha r_2).
\end{align*}
\]

The probability of \(\Psi_1\) is the probability of the OBDD, which is \(P(\alpha r_1)\).

2.4. Probabilistic databases

Syntax and semantics. Probabilistic c-tables (pc-tables) are relational databases where each tuple is annotated with a formula over a set \(X\) of independent Boolean random variables [Imielinski and Lipski 1984; Suciu et al. 2011]. In its simplest form, each annotation formula is a distinct variable: This is the tuple-independent model considered in this article.

Under the possible worlds semantics, a pc-table \(D\) represents a finite set of possible worlds: Each total assignment \(I\) of the variables in \(X\) defines a possible world representing a relational database consisting of exactly those tuples in \(D\) whose annotations are satisfied by \(I\). The probability of each world is the product of probabilities of the variable assignments in \(I\), that is, \(Pr(I)\) as defined above. This representation formalism is complete in the sense that it can represent arbitrary probability distributions over any finite set of possible worlds.

Query evaluation. Given a query \(Q\) and a pc-table \(D\), the query evaluation problem is to compute the distinct tuples in the results of \(Q\) in the worlds of \(D\) together with their probabilities. The probability \(P(t \in Q(D))\) of a tuple \(t\) is the probability that \(t\) is in the result of \(Q\) in a world randomly drawn from \(D\). We adopt the intensional approach to query evaluation [Suciu et al. 2011]. For each result tuple \(t\), first construct a propositional formula \(\Phi_{t \in Q(D)}\) that annotates \(t\) such that \(P(t \in Q(D)) = P(\Phi_{t \in Q(D)})\), then compute \(P(\Phi_{t \in Q(D)})\) as per Equation (1). The annotation of a Boolean query \(Q\) is denoted by \(\Phi_{Q(D)}\); when the context is clear, we often omit the explicit reference to the database \(D\) and simply write \(\Phi_{Q}\). We next explain how to annotate the results of relational algebra queries on pc-tables.

The tuples together with their annotation in the result of a relational algebra query \(Q\) can be computed directly from the input pc-table \(D\). This is achieved by rewriting \(Q\) into a query \(Q^n\) such that standard relational evaluation of \(Q^n\) on \(D\) yields a pc-table representing the results of \(Q\) in the worlds of \(D\). Algorithm 1 specifies such a rewriting function \([\cdot]\). It assumes that the formulas annotating the tuples in the input pc-table are stored in a distinguished column called \(\Phi\); for a relation \(R\), we consider that this column
Algorithm 1: Computing annotated result tuples for RA queries on pc-tables.

is not selected by the selector $R \ast$. The rewriting is expressed here in SQL and — besides a straightforward encoding of the relational operators in SQL — it constructs formulas annotating result tuples based on the formulas of input tuples as follows. In case of identity, selection, and renaming operators, the input annotations are just copied to the result. For projection, the formula of each distinct result tuple is constructed as the disjunction of all input tuples with the same restriction to the attributes in the projection list. For the join operator, the formula of a result tuple is the conjunction of the formulas of the contributing input tuples; to avoid cluttering, we slightly abuse notation in stating the attributes of the select clause: $R \ast$ means here the set-union of the attributes in $R$ and $S$. A tuple $t$ in the result of $Q_1 - Q_2$ has annotation $\Phi_1$ if $t$ is in $Q_1$ with annotation $\Phi_1$ and $t$ is not in $Q_2$, and has annotation $\Phi_1 \land \neg \Phi_2$ if $t$ is in $Q_1$ with annotation $\Phi_1$ and in $Q_2$ with annotation $\Phi_2$. These two cases are implemented in \[ \] by a left outer join.

Example 2.3. Figure 2 shows pc-tables, where each tuple is annotated by a distinct Boolean random variable stored in column $\Phi$, and how annotations are propagated through the sub-queries of the depicted relational algebra query. The query result is the empty tuple annotated with the formula $\Phi = x_1 y_1 \lor x_1 y_2$.

3. HIERARCHICAL $1RA^-$ QUERIES ARE TRACTABLE

In this section we show the following result:

**Lemma 3.1.** Any hierarchical $1RA^-$ query $Q$ on tuple-independent databases has polynomial-time data complexity.

**Proof.** We assume without loss of generality that $Q$ is Boolean; if $Q$ is non-Boolean, we define a hierarchical Boolean $1RA^-$ query for each tuple $t$ in the result of $Q$, where the tuple of attributes exported by $Q$ is set to $t$. We prove the lemma via a sequence of steps:

$Q$ is a hierarchical (Boolean) $1RA^-$ query.

**Lemma \[ \] $Q$ is equivalent to a relational calculus query $Q_{RC}$ that is RC-hierarchical (Definition 3.2) and \( \exists \)-consistent (Definition 3.3).

**Lemma \[ \] For any tuple-independent database $D$, we compute an OBDD $o$ for the formula annotating the query result $Q_{RC}(D)$ in time and of size $O(|D| \cdot 2^{|Q_{RC}|})$.

**Corollary \[ \] The probability of $\Phi$ can be computed in one bottom-up pass over the OBDD $o$, so in time $O(|D| \cdot 2^{|Q_{RC}|})$. \]

The reason for translating $1RA^-$ queries to relational calculus in the first step of the proof is that relational calculus is more flexible and allows to unfold negated expressions as per $\neg(Q_1 \land Q_2) \equiv \neg Q_1 \lor \neg Q_2$. The obtained rewritings are not arbitrary relational calculus queries. They are disjunctions of disjunction-free existential relational calculus queries that
are expressible in the language $\text{RC}^3$ and thus have no universal quantifiers. They are safe, that is, every query variable appears in at least one positive relation symbol. They use negation solely to capture the difference operator in relational algebra, which means that in an expression $Q_1 \land \neg Q_2$, the results of $Q_1$ and $Q_2$ have the same arity. They are canonicalised in the sense that every occurrence of a relation symbol $R(Y_1, \ldots, Y_m)$ in a query rewriting has the same query variables $Y_1, \ldots, Y_m$. In contrast to the original $1\text{RA}^-$ queries, the $\text{RC}^3$ rewritings may have repeating relation symbols.

The $\text{RC}^3$ rewritings enjoy two properties that are key to our query evaluation algorithm introduced later in this section. Firstly, they are hierarchical in a more syntactically restricted sense than the $1\text{RA}^-$ queries.

**Definition 3.2.** An $\text{RC}^3$ query $Q$ is $\text{RC}$-hierarchical if for every sub-query $\exists X(F)$ in $Q$, the variable $X$ occurs in every relation symbol in $F$. We say that $X$ is root in $F$.

Secondly, $\text{RC}^3$ rewritings allow for a total nesting order of its existential quantifiers for all their disjuncts.

**Definition 3.3.** A canonicalised $\text{RC}^3$ query $Q$ is $\exists$-consistent if there exists a total order $>_3$ of the variable symbols in $Q$ such that $X >_3 Y$ implies that there is no sub-query $\exists Y F(\exists X)$ in $Q$, where $F(\exists X)$ denotes an expression that contains the quantifier $\exists X$.

Intuitively, $\exists$-consistency for an $\text{RC}^3$ query that is a conjunction or disjunction of sub-queries means that these sub-queries have compatible join orders, that is, non-contradicting $>_3$ orders. This also 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 $\text{RC}$-hierarchical property effectively helps inferring from the order of the existential quantifiers an OBDD variable order under which the OBDD has size 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 3.4.** Consider the following three disjunction-free $\text{RC}^3$ 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 $\text{RC}$-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></th>
<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>$\Phi$</td>
<td>$A$</td>
<td>$\Phi$</td>
<td>$A$</td>
<td>$\Phi$</td>
</tr>
<tr>
<td>$B$</td>
<td>$\Phi$</td>
<td>$\Phi$</td>
<td>$B$</td>
<td>$\Phi$</td>
<td>$B$</td>
<td>$\Phi$</td>
</tr>
<tr>
<td>$1$</td>
<td>$m_1$</td>
<td>$n_1$</td>
<td>$r_1$</td>
<td>$t_1$</td>
<td>$u_1$</td>
<td>$v_1$</td>
</tr>
<tr>
<td>$2$</td>
<td>$m_2$</td>
<td>$n_2$</td>
<td>$r_2$</td>
<td>$t_2$</td>
<td>$u_2$</td>
<td>$v_2$</td>
</tr>
</tbody>
</table>

The annotations of the results for our queries evaluated on $D$ are the read-once formulas

$$\Phi_1 = (m_1 r_1 \lor m_2 r_2) \land (n_1 \lor n_2)$$

$$\Phi_2 = (m_1 \lor m_2) \land (n_1 t_1 \lor n_2 t_2)$$

$$\Phi_3 = (m_1 u_1 \lor m_2 u_2) \land (n_1 v_1 \lor n_2 v_2)$$
and, similarly to the first two OBDDs in Fig. 4, can be represented by OBDDs with one
Dichotomies for Queries with Negation in Probabilistic Databases 1:13
−
ACM Transactions on Database Systems, Vol. 1, No. 1, Article 1, Publication date: October 2015.
Following Lemma 2.1, the disjunction of the OBDDs of Φ
3.1. From 1RA to RC3
Our evaluation algorithm for hierarchical 1RA− queries relies on a translation of 1RA− queries into equivalent RC3 queries. The translation function \[\llbracket\cdot\rrbracket\], which is given in Algorithm 2, is the standard recursive inside-out translation from relational algebra to safe relational calculus (Lemma 5.3.11, Abiteboul et al. 1995), with the addition that after each recursive translation step we “flatten” the resulting RC3 query as follows:

\[
[\Pi_{123}] = \{m_1, r_1, u_1, m_2, r_2, n_2, v_1, n_1, r_2, n_2, t_2, v_2, n_2, t_2, v_2\}
\]

Following Lemma 2.1, the disjunction of the OBDDs of Φ1, Φ2, and Φ3 can be represented by an OBDD of width at most 23 for the annotation Φ1 ∨ Φ2 ∨ Φ3 of query \(Q_{123}\). □
— Every $\exists$ quantifier is pushed as deep as possible in the $\text{RC}^3$ query without pushing it past negation: $\exists_X$ distributes over disjunctions and is pushed past conjuncts in which $X$ does not appear. Lemma 3.5 shows that every $\exists_X$ quantifier can be pushed until $X$ becomes root, that is, $X$ occurs in all relation symbols in its scope.

— Every negation symbol is pushed as deep as possible in the $\text{RC}^3$ query (as per $\neg (A \land B) \rightarrow \neg A \lor \neg B$ and its dual) without pushing it past an existential quantifier.

— Conjunctions of disjunctions are eagerly expanded into disjunctions of conjunctions.

We may also apply the following two equivalence-preserving simplification rules, which are not necessary for the properties described in this article but useful for a practical implementation: Given $\text{RC}^3$ expressions $Q_1$ and $Q_2$,

$$Q_1 \lor Q_1 \land Q_2 \rightarrow Q_1 \quad \text{and} \quad \neg \exists_X (Q_1) \land \neg \exists_X (Q_1 \land Q_2) \rightarrow \neg \exists_X (Q_1)$$

The second rewriting is essentially a special case of the first rewriting, though our translation function can generate instances of left-hand sides of both rules.

Our translation has several desirable properties:

**Lemma 3.5.** For any hierarchical $1\text{RA}^-$ query $Q_{RA}$, the $\text{RC}^3$ rewriting $Q_{RC} = \llbracket Q_{RA} \rrbracket$ satisfies the following properties:

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

(b) $Q_{RC}$ is canonicalised.

(c) $Q_{RC}$ is a disjunction of disjunction-free $\text{RC}^3$ queries.

(d) For every variable $X$ occurring in $Q_{RC}$, $Q_{RC}$ has no sub-query of the form $\exists_X (Q) \land F(\exists_X)$, where $F$ is an expression that contains the quantifier $\exists_X$.

(e) $Q_{RC}$ is $\text{RC}$-hierarchical.

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

**Proof.** Property (a) holds since every rewriting step preserves equivalence.

Property (b). The attributes that are transitively joined in $Q_{RA}$ translate to the same variable name in $Q_{RC}$. This is indeed achieved in $[Q_1 \bowtie Q_2]$ and $[Q_1 - Q_2]$ by renaming all attributes in $Q_2$ to the corresponding variable name in $Q_1$. When a quantifier $\exists_X$ is introduced via $[\pi_{-X}(Q)]$, then all occurrences of $X$ in $Q$ are in the scope of $\exists_X$: furthermore, the $\text{Push}^3_X$ procedure never pushes $\exists_X$ past a relation symbol containing $X$ and hence every occurrence of $X$ is in the scope of a quantifier $\exists_X$.

Property (c). The combination of pushing down negation ($\text{Push}^-$) and expanding conjunctions of disjunctions ($\text{Expand}$) is standard and yields an expression with disjunctions only at the top level.

Property (d). Let us assume that $Q_{RC}$ contains an expression $E = \exists_X (Q) \land F(\exists_X)$. Then the parse tree of $Q_{RC}$ contains a sub-tree of the following form:

$$\exists_X \quad \lor \quad \exists_X \quad \land \quad F$$

By construction using Algorithm 2, $E$ occurs positively in $Q_{RC}$, otherwise the algorithm would push the negation past the conjunction and yield $\neg \exists_X (Q) \lor \neg F$. 

ACM Transactions on Database Systems, Vol. 1, No. 1, Article 1, Publication date: October 2015.
Let us consider the possible transformation sequences that could have led to \( E \). Since \( Q_{RA} \) is non-repeating and since \( Q_{RC} \) is canonicalised, the two \( \exists_X \) quantifiers originate from one \( \exists_X \) quantifier that has been applied to the entire sub-query as a result of the translation of \( \pi_{-X}(Q) \). The \( \exists_X \) quantifier must subsequently have been “duplicated” as a result of distributing it through disjunctions in \( \text{PUSH}^{3X} \). Since \( \text{PUSH}^{3X} \) does not distribute \( \exists_X \) over a conjunction, the conjunction operator on top of this sub-query must have been flipped by a negation operator into a disjunction through which \( \exists_X \) was distributed; in order to get back to a conjunction operator, there must have been an odd number of \( \neg \) operators on top of the \( \exists_X \) quantifier. This is a contradiction to the assumption that \( Q_{RC} \) contains the expression \( E \), where the \( \exists_X \) quantifier appears positively.

**Property 2**. We show this property by induction on the different cases in the translation function \( \llbracket \cdot \rrbracket \) in Algorithm 2.

Base case: \( Q_{RA} = R. \) Since \( \llbracket R \rrbracket = \{(\text{sch}(R)) \mid R(\text{sch}(R))\} \) does not contain existential quantifiers, \( \llbracket R \rrbracket \) is vacuously RC-hierarchical.

Hypothesis: The \( \text{RC}^3 \) expressions obtained by translating sub-queries of \( Q_{RA} \) are RC-hierarchical.

Inductive step: Any step taken by the translation preserves the RC-hierarchical property. The only translation step that may introduce a non-root \( \exists_X \) quantifier is the case for \( \llbracket \pi_{-X}(Q_{RA}) \rrbracket \). By the induction hypothesis, let \( Q_{RC} = \llbracket Q_{RA} \rrbracket \) be the RC-hierarchical \( \text{RC}^3 \) query resulting from the translation of \( Q_{RA} \). Following properties (a) – (d), \( Q_{RC} \) is canonicalised with respect to \( Q_{RA} \), it is a disjunction of disjunction-free \( \text{RC}^3 \) queries, and it does not contain an expression of the form \( \exists_X(Q) \land F(\exists_X) \). We show that if \( \text{PUSH}^{3X}(Q_{RC}) \) fails, then \( Q_{RA} \) is non-hierarchical. Conversely, for any hierarchical query \( Q_{RA} \), \( \text{PUSH}^{3X} \) always succeeds in pushing \( \exists_X \) so that \( X \) becomes a root variable in the scope of \( \exists_X \).

The rule for \( Q_{RC} = \bigvee_j Q_j \) pushes \( \exists_X \) through the top-level disjunction and applies \( \exists_X \) to each of the disjunction-free queries \( Q_j \). Each \( Q_j \) is a conjunction \( \wedge_i q_i \) where each expression \( q_i \) has the form \( \exists_Y(q), \neg \exists_Y(q), R, \) or \( \neg R \). We analyse different cases separately:

1. **Variable X occurs in exactly one expression \( q_i \):**
   (i) \( q_i = \exists_Y(q) \). Then, by induction hypothesis, \( Y \) is root in \( q \) and this remains true after commuting \( \exists_X \) and \( \exists_Y \): \( \exists_Y(\text{PUSH}^{3X}(q)) \). We next analyse \( \text{PUSH}^{3X}(q) \).
   (ii) \( q_i = R \) or \( q_i = \neg R \). Then, \( \text{PUSH}^{3X} \) returns \( \exists_XR \) or \( \exists_X\neg R \), respectively, and \( X \) is root in the scope of \( \exists_X \) in both cases.
   (iii) \( q_i = \neg \exists_Y(q) \). If \( X \) is root in \( q \), then the rule for \( Q_{RC} = \neg Q' \) applies and we return \( \exists_X \neg \exists_Y(q) \), in which \( X \) is root. If \( X \) were not root in \( q \), then \( q \) must contain two relations symbols such that by induction hypothesis \( Y \) occurs in both relation symbols and \( X \) occurs in one but not the other. Without loss of generality, let us assume that these relations are \( T(Y) \) and \( S(X,Y) \). Since \( \llbracket \cdot \rrbracket \) does not commute \( \neg \) and \( \exists \), \( Q_{RA} \) must contain a sub-query of the form \( \pi_{-X}(Q_1 \pi_{-Y}Q_2) \), where \( Q_2 \) refers to relations \( S \) and \( T \) and \( Q_1 \) exports \( [X] \); furthermore, since \( Q_{RC} \) is canonicalised, no variable in \( [Y] \) can occur in any relation symbol in \( Q_1 \). This means that \( Q_{RA} \) contains three relation symbols \( R^X[\neg Y], S^X[Y], T^{[\neg X][Y]} \) and is thus non-hierarchical.

2. **Variable X occurs in more than one of the expressions \( q_i \):**
   (i) \( X \) is root in all expressions \( q_i \). Then, the case \( Q_{RC} = (\bigwedge_i Q_i) \wedge (\bigwedge_j Q_j') \) applies in \( \text{PUSH}^{3X} \) and \( X \) becomes root in the scope of \( \exists_X \).
   (ii) \( X \) is not root in all expressions \( q_i \). Then, \( X \) occurs in one of them, say \( q_i \), and there is a second expression, say \( q_k \), that contains two relation symbols \( S(X,Y) \) and \( T(Y) \) such that \( X \) occurs in one of them but not in the other. Thus \( q_k \) has the form \( \exists_Y Q(S,T) \) or \( \neg \exists_Y Q(S,T) \) where by induction hypothesis \( Y \) occurs in both \( S \)
and \( T \). The latter case is equivalent to the above case (iii). In the former case, if \( q_i \) is a single relation \( R \) (or its negation) in which \( X \) occurs without a quantifier, then this relation cannot have variable \( Y \) because all occurrences of \( Y \) are in the scope of a quantifier, cf. Property (ii). The relations \( R, S, T \) thus render \( Q_{RA} \) non-hierarchical. On the other hand, \( q_i \) may have the form \( \exists Z(Q) \) where \( Z \) is a variable different from \( Y \) and \( Q \) does not contain a quantifier \( \exists Y \), cf. Property (ii). Then the relation in \( q_i \) in which \( X \) occurs, together with \( S \) and \( T \) render \( Q_{RA} \) non-hierarchical.

**Property (ii).** Assume \( Q_{RC} \) contains two sub-queries \( Q_a = \exists_X Q(\exists_Y) \) and \( Q_b = \exists_Y Q(\exists_X) \) in which the order of the quantifiers \( \exists_X \) and \( \exists_Y \) is inconsistent. If one of the two sub-queries has the form \( \exists_X \exists_Y(Q') \), then the order of the quantifiers may be switched to obtain \( \exists \)-consistent queries; conversely, there are two structurally different cases in which the order of the quantifiers cannot be switched: (i) \( \exists_X Q(\neg \exists_Y) \), and (ii) \( \exists_Y [Q(X) \land Q'(\exists_Y)] \). We consider the four combinations of these cases for the sub-queries \( Q_a \) and \( Q_b \):

1. They are of type (i): \( Q_a = \exists_X Q_1(\neg \exists_Y) \) and \( Q_b = \exists_Y Q_2(\neg \exists_X) \). Then, since \( Q_{RC} \) is canonicalized and since the order of \( \exists \) and \( \neg \) is never swapped by the function \( [.] \), the structure of \( Q_a \) implies that \( Q_{RC} \) has a sub-query of the form \( \pi_{\neg X}(Q'_1 - \pi_{X}(Q'_2)) \) and the structure of \( Q_b \) implies that \( Q_{RC} \) has a sub-query of the form \( \pi_{\neg Y}(Q'_2 - \pi_{X}(Q'_1)) \). This is a contradiction.

2. They are of type (ii): \( Q_a = \exists_X [Q_1(X) \land Q'_1(\exists_Y)] \) and \( Q_b = \exists_Y Q_2(Y) \land Q_2(\exists_X) \). Then \( Q_1 \) contains a relation \( R^{X \land [\neg Y]} \), \( Q_2 \) contains a relation \( T^{[\neg X][Y]} \), and, since \( Q_{RC} \) and all its sub-queries are hierarchical, \( Q'_1 \) contains a relation \( S^{[X][Y]} \). \( Q_{RA} \) is thus non-hierarchical since it contains the relations \( R^{X \land [\neg Y]}, S^{[X][Y]}, T^{[\neg X][Y]} \). This is a contradiction.

3. \( Q_a \) is of type (i) and \( Q_b \) is of type (ii): \( Q_a = \exists_X Q_1(\neg \exists_Y) \) and \( Q_b = \exists_Y [Q_2(Y) \land Q_2(\exists_X)] \). Since there is no negation between \( \exists_Y \) and \( \exists_X \) in \( Q_b \), there is also no difference operator between \( \pi_{\neg X} \) and \( \pi_{\neg Y} \) in \( Q_{RA} \). Conversely, the negation between \( \exists_X \) and \( \exists_Y \) in \( Q_a \) requires a difference operator between \( \pi_{\neg X} \) and \( \pi_{\neg Y} \) in \( Q_{RA} \). This is a contradiction.

4. \( Q_a \) is of type (ii) and \( Q_b \) is of type (i): This is symmetric to the previous case. □

**Property (iii) disallows sub-queries of the form \( \exists_X (Q_1) \land \exists_X (Q_2), \exists_X (Q_1) \land \neg \exists_X (Q_2), \exists_X (Q_1) \land \neg \exists_Y (Q_2 \land \neg \exists_X (Q_3)), \) but does not forbid \( \neg \exists_X (Q_1) \land \neg \exists_X (Q_2) \) or \( \exists_X (Q_1) \lor \exists_X (Q_2) \).**

**Example 3.6.** Consider the following two 1RA\(^{-}\) queries over the database schema \((M(A), N(B), R(A_1), T(B_1), U(A_2), V(B_2))\):

\[
Q_a = \exists_\emptyset \left[ M \times N \land \neg A_{\emptyset} \uplus A_1, B_{\emptyset} \uplus B_1 \left[ R \times T \land \neg A_{\emptyset} \uplus A_2, B_1 \uplus B_2 \land U \lor V \right] \right]
\]

\[
Q_b = \exists_\emptyset \left[ \pi_1 (M \times N) \land \neg A_{\emptyset} \uplus A_1 \pi_1 \left[ R \times T \land \neg A_{\emptyset} \uplus A_2, B_1 \uplus B_2 \land U \lor V \right] \right].
\]

Query \( Q_a \) translates to \( Q_{123} \) from Example 3.4 where subsumed sub-queries are removed:

\[
[Q_a] = \exists_4 \exists_B (M(A) \land N(B) \land \neg (R(A) \lor T(B) \lor \neg (U(A) \land V(B))))
= \exists_4 \exists_B (M(A) \land N(B) \land \neg (R(A) \lor T(B) \land \neg (U(A) \lor V(B))))
= \exists_4 \exists_B (M(A) \land N(B) \land \neg (R(A) \lor T(B) \land U(A) \land V(B)))
= \exists_4 \exists_B (M(A) \land N(B) \land \neg (R(A) \lor T(B) \lor U(A) \land V(B)))
= \exists_4 (M(A) \land \neg R(A)) \land \exists_B (N(B) \land \neg T(B)) \lor \exists_4 \exists_B (N(B) \land \neg T(B)) \lor \exists_4 (M(A) \land U(A)) \land \exists_B (N(B) \land V(B)) = Q_{123}.
\]
Query $Q_b$ is similar to $Q_a$, but with additional projections on $A$ on both sides of the top-most difference operator (and hence $B$ is not in the equivalence class of $B_1$ and $B_2$):

$$\left[Q_b\right] = \exists_A \left[ M(A) \land \exists_B (N(B) \land \neg \exists_{B_1} (R(A) \land T(B_1) \land \neg U(A) \land V(B_1))) \right]$$

$$= \exists_A \left[ M(A) \land \exists_B (N(B) \land \neg \exists_{B_1} (R(A) \land T(B_1) \land \neg U(A) \lor R(A) \land T(B_1) \land \neg V(B_1))) \right]$$

$$= \exists_A \left[ M(A) \land \exists_B (N(B) \land (\neg R(A) \lor \neg \exists_{B_1} (T(B_1))) \lor U(A) \right]$$

$$\land (\neg R(A) \lor \neg \exists_{B_1} (T(B_1) \land \neg V(B_1))))$$

$$= \exists_A \left[ M(A) \land \exists_B (N(B) \land (\neg R(A) \lor \neg \exists_{B_1} (T(B_1))) \lor U(A) \land \neg \exists_{B_1} (T(B_1) \land \neg V(B_1))) \right]$$

$$= \exists_A \left[ M(A) \land \neg R(A) \land \exists_B N(B) \lor \exists_A M(A) \land \exists_B N(B) \land \neg \exists_{B_1} T(B_1) \lor \right.$$

$$\exists_A \left[ M(A) \land U(A) \land \exists_B N(B) \land \neg \exists_{B_1} (T(B_1) \land \neg V(B_1)) \right]$$

Both $RC^3$ queries $\left[Q_a\right]$ and $\left[Q_b\right]$ satisfy Lemma 3.5 for every quantifier $\exists_A (\exists_B$ or $\exists_{B_1})$, $A$ ($B$ or $B_1$) is a root variable in its scope (Property (e)), and the nesting orders of these operators are consistent in all sub-queries (Property (f)).

The query translation can lead to large $RC^3$ queries: A conservative upper bound on their sizes would be a non-elementary function of the size of the input $IRA^-$ query, explained by the rapid increase in the size and number of disjuncts when pushing down negation, quantifiers, and conjunctions. A singly-exponential upper bound holds for $IRA^-$ 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 $Q$ (that is, $X$ is root in $Q$). The query $Q_a$ in Example 3.6 satisfies this condition trivially, since it has no projection that is a right descendant of a difference operator. This conservative upper bound suffices for the data-complexity argument in Lemma 3.1 since the blowup is only in the query size. A practical implementation of Algorithm 2 would eagerly apply the simplification rules after each expansion step.

3.2. OBDD Construction

The penultimate step in the proof of Lemma 3.1 is the OBDD compilation of the annotation $\Phi$ of the $RC^3$ query $Q_{RC}$, which is the rewriting of an input hierarchical $IRA^-$ query $Q_{RA}$ as per Lemma 3.5. 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 3.7. The Boolean hierarchical $IRA^-$ query $\pi_0[R \bowtie \pi_X(S - T)]$, over a probabilistic database with schema $(R(X), S(X, Y), T(X, Y))$, translates to the $RC^3$ query

$$Q_{RC} = \exists_X \left[ R(X) \land \exists_Y (S(X, Y) \land \neg T(X, Y)) \right].$$

Since $X$ is root in $Q_{RC}$, the OBDDs for $Q_{RC}$’s annotations for different values of $X$ share no Boolean variables (that is, are independent) and can be concatenated. For each value $x$ in the active domain of $X$, we construct an OBDD for the annotation of the query $R(x) \land \exists_Y (S(x, Y) \land \neg T(x, Y))$; a good variable order for this OBDD is the sequence formed by the annotation of $R(x)$ and all annotations of $S(x, y)$ and $T(x, y)$ for all values $y$ in the active domain of $Y$. If we write $R(x_i)$ for the annotation of tuple $(x_i)$ in $R$, and similarly for $S$ and $T$, then the overall variable order is (e.g., for tuples with $X = x_1$ and $X = x_2$):

$$R(x_1), S(x_1, y_1), T(x_1, y_1), S(x_1, y_2), T(x_1, y_2), S(x_1, y_3), T(x_1, y_3), \ldots,$$

$$R(x_2), S(x_2, y_1), T(x_2, y_1), S(x_2, y_2), T(x_2, y_2), S(x_2, y_3), T(x_2, y_3), \ldots.$$
consider all annotations for \( X = x_1 \), then all annotations with \( X = x_2 \), and so on. For all annotations for \( X = x_1 \), we first consider those for \( Y = y_1 \), then those for \( Y = y_2 \), and so on. 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 \( \exists_Y \).

The \( \text{RC}^3 \) query \( Q_1 = \exists_A (R(A) \land \neg U(A)) \land \exists_B T(B) \) from Example 1.5 is the conjunction of two \text{RC}-hierarchical and \( \exists \)-consistent sub-queries: \( Q_A = \exists_A (R(A) \land \neg U(A)) \) and \( Q_B = \exists_B T(B) \). Similarly, \( Q_1 \)'s annotation \( \Psi_1 = (r_1 \land u_1) \lor r_2 \land u_2 \) is the conjunction of two formulas: \( \varphi_A = r_1 \land u_1 \lor r_2 \land u_2 \) annotating \( Q_A \) and \( \varphi_B = t_1 \lor t_2 \) annotating \( Q_B \). The query variables \( A \) and \( B \) are root in \( Q_A \) and respectively \( Q_B \). We independently construct OBDD variable orders for \( \varphi_A \) and \( \varphi_B \) and then concatenate them to obtain the overall variable order for the OBDD of \( \Psi_1 = \varphi_A \land \varphi_B \). Since \( A \) is root, the OBDD variable order for \( \varphi_A \) is a sequence of annotations \( R(a_1), S(a_1), \ldots, R(a_n), S(a_n) \) for the domain \( \{a_1, \ldots, a_n\} \) of \( A \): This is \( r_1, u_1, r_2, u_2 \). Similarly, we obtain the OBDD variable order \( t_1, t_2 \) for \( \varphi_B \). 

This variable order derived from the structure of the \( \text{RC}^3 \) query rewritings leads to poly-size OBDDs for query annotations. In general, however, finding an optimal OBDD variable order, that is, one that minimizes the size of the OBDD, is NP-complete [Wegener 2004].

**Lemma 3.8.** For any \( \text{RC}^3 \) query \( Q_{RC} \) that satisfies the properties of Lemma 3.5, 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 non-Boolean case follows as per discussion in the proof of Lemma 3.5. Let the relation symbols in \( Q_{RC} \) be \( R_1, \ldots, R_n \), the query variables be \( X_1, \ldots, X_m \), and let \( \text{ADom}(X_i) \) be the active domain of variable \( X_i \). The annotation of tuple \( t \) of relation \( R_6 \) is denoted by \( R_6(t) \), e.g., the annotation of tuple \( (a, b) \) in relation \( R_6 \) is \( R_6(a, b) \). We assume without loss of generality that the order of the query variables \( X_1, \ldots, X_m \) is such that \( X_i \succ X_j \Leftrightarrow i < j \) with respect to the nesting order \( \succ \) defined by the \( \exists \)-consistency of \( Q_{RC} \); that is, \( i < j \) allows for the quantifier nesting \( \exists_X Q(\exists_X \cdot) \), but not \( \exists_X Q(\exists_X \cdot) \). Since \( Q_{RC} \) is canonicalised and \( \exists \)-consistent (Lemma 3.5), we can assume without loss of generality that the query variables in each relation symbol \( R \) occur in \( \succ \) order (we can always relabel the query and database schema such that the query variables occur in \( \succ \)-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 \( x_i \in \text{ADom}(X_i) \) and \( x_j \in \text{ADom}(X_j) \), \( x_i < x_j \Leftrightarrow i < j \); 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 \( \Pi \) on the annotations of the tuples in \( D \) as follows. We first associate with every annotation \( R(t) \) the string \( \text{string}(R(t)) = tR \), 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

\[ R(t) \Pi R'(t') \Leftrightarrow \text{string}(R(t)) \text{lex} \text{string}(R(t')) \]

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. The order \( \Pi \) is uniquely defined by the order of the relation symbols and the order on the active domain of \( D \).

We show by structural induction over the annotation \( \Phi \) that \( \Phi \) has a \( \Pi \)-OBDD of width \( 2^{|Q_{RC}|} \), where \( |Q_{RC}| \) denotes the number of relation symbols in \( Q_{RC} \).
The base case is a Boolean variable $R(t)$ which corresponds to a trivial II-OBDD with one variable $R(t)$ and width 2 (there are two edges between the level of the root node $R(t)$ and the next level of leaf nodes $\top$ and $\bot$).

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 $2^{Q_1}$ and $2^{Q_2}$, respectively. Then by Lemma 2.1 the annotation of $Q_{RC}$ has a II-OBDD of width $2^{Q_1} + 2^{Q_2} = 2^{Q_{nc}}$.

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

If $Q_{RC} = \exists x_i Q$, then for every $x_i \in ADom(X_i)$ the annotations $\Phi_t$ of queries $Q[x_i/X_i]$ are over disjoint sets of variables because $Q_{RC}$ is RC-hierarchical by Lemma 3.5 and $X_i$ is root in $Q$. Moreover, each annotation $\Phi_t$ has a II-OBDD of width $2^Q$ by induction hypothesis. Let $ADom(X_i) = \{x_1, \ldots, x_h\}$ such that $x_k <_{\text{lex}} x_l$ if and only if $k < l$. The annotation $\Phi$ of $Q_{RC}$ is the disjunction $\bigvee_{x_i \in ADom(X_i)} \Phi_t$. Since the formulas $\Phi_t$ are over disjoint sets of variables for distinct values of $l$, an OBDD for their disjunction is obtained by their concatenation in which the leaf node $\bot$ of the OBDD for $\Phi_t$ is replaced by the root node of the OBDD for $\Phi_{t+1}$.

It remains to show that this construction yields an OBDD over order $\Pi$. Firstly, the OBDD for each annotation $\Phi_t$ is over order $\Pi$ by induction hypothesis; we show that for any two annotations $R(t_k)$ in $\Phi_k$ and $R'(t_l)$ in $\Phi_l$ with $k < l$, it holds that $R'(t_l) <_{\Pi} R(t_k)$; by the definition of $<_{\Pi}$, this is equivalent to showing $t_l R <_{\text{lex}} t_k R'$. The strings $t_k$ and $t_l$ are identical in the $i$-th places since, by construction, the occurrences of each variable $X_i$ with $j < i$ are set to the same constant. The lexicographic order of $t_l$ and $t_k$ — and hence the II-order of $R(t_k)$ in $\Phi_k$ and of $R'(t_l)$ in $\Phi_l$ — is determined by the values of $X_i$ in $t_k$ and in $t_l$; this value is $x_l$ in $t_l$ and $x_k$ in $t_k$. Since we concatenate the OBDDs in the order $\Phi_1 \to \cdots \to \Phi_h$ and since $x_1 <_{\text{lex}} \cdots <_{\text{lex}} x_h$, it follows that $t_k <_{\text{lex}} t_l$ and thus $R(t_k) <_{\Pi} R'(t_l)$. The constructed OBDD has width $2^{Q_{nc}} = 2^Q$, because the OBDD concatenation leaves the width unchanged.

The OBDD construction in the above proof shows that conjunction, disjunction, negation, and existential quantification of $\text{RC}^3$ queries representing rewritings of hierarchical 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 $\text{RC}^3$ constructs in the proof. A tighter bound can be obtained via a more specific analysis: Any non-repeating RC-hierarchical $\text{RC}^3$ query $Q$ admits an OBDD of width at most $|Q|$ and size linear in the input database size and independent of the query size [Ofteanu and Huang 2008]. 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 $|Q_1| \cdots |Q_n|$, which is smaller than $2^{Q_1} + \cdots + 2^{Q_n}$ as used in the proof.

We can now use both Lemmata 2.1 and 3.8 to obtain the polynomial-time computation of query probability, or equivalently of the formula annotating the query result:

**Corollary 3.9 (Lemmata 2.1 and 3.8).** Let $Q_{RC}$ be a $\text{RC}^3$ query satisfying the properties of Lemma 3.8, $D$ a tuple-independent database, and $\varphi$ the formula annotating the query result $Q_{RC}(D)$. Then, the probability of $\varphi$ can be computed in time $O(|D| \cdot 2^{Q_{nc}})$.

### 4. DATABASE CONSTRUCTION SCHEME USED IN HARDNESS REDUCTIONS

We next present a database construction scheme that prescribes how to populate relations used in a non-hierarchical query such that the query result is annotated with a desired 2DNF formula. It focuses on two distinguished attribute classes $[A]$ and $[B]$ that witness the non-hierarchical property of the query. The construction scheme populates the attributes that
are not in \([A]\) and \([B]\) such that the input values for the attributes in \([A]\) and \([B]\) along with their annotations are propagated through the query operators to the result. This scheme is used in Section 5 to construct hardness reductions for non-hierarchical queries.

We use two finite sets of constants, \(K_A\) and \(K_B\), and a constant \(\Box\) distinct from those in \(K_A\) and \(K_B\). In this section, the projection operator \(\pi^\Phi_Q\) retains both the attribute \(A\) and the annotation column \(\Phi\); in contrast, \(\pi^\Phi_A\) only selects the attribute \(A\) without the annotation column \(\Phi\). The notation \((a_1, \ldots, a_n)\Phi(a_1, \ldots, a_n))\) denotes a tuple \((a_1, \ldots, a_n)\) annotated with formula \(\Phi(a_1, \ldots, a_n)\).

### 4.1. Preserving the data of one attribute

We first consider the case of one distinguished attribute \(A\). Let \(\Phi\) be a total function on \(K_A\). A query \(Q\) (and its particular case of a relation) is \(A\)-reducible to \((K_A, \Phi)\) if the \([A]\)-attributes of \(Q\) are filled with all values from \(K_A\), all attributes not in \([A]\) are filled with \(\Box\), and the annotation of a tuple identified by \(a \in K_A\) is \(\Phi(a)\):

\[
\pi^\Phi_A(Q) = \{ (a, \Phi(a)) \mid a \in K_A \} \quad \text{for any attribute } A \in [A]
\]

\[
\pi^\Box_C(Q) = \{ (\Box) \} \quad \text{for any attribute } C \notin [A]
\]

By \(\text{red}_A(Q) = K_A|\Phi\) we denote that \(Q\) is \(A\)-reducible to \((K_A, \Phi)\). A query \(Q\) that does not export \([A]\) is \(\emptyset\)-reducible to \((\Box|\Phi)\), and we denote it by \(\text{red}_\emptyset(Q) = \Box|\Phi\) where

\[
\pi^\emptyset_A(Q) = \{ (\Phi) \}
\]

\[
\pi^\Box_C(Q) = \{ (\Box) \} \quad \text{for any attribute } C.
\]

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

\[
Q^{[A]} \in Q^A \text{ if } \text{red}_A(Q) = K_A|\Phi \text{ or } \text{red}_A(Q) = K_A|\neg\Phi \quad (2)
\]

\[
Q^{[A]} \in Q_{\text{fill}} \text{ if } \text{red}_A(Q) = K_A|\Phi_T \quad (3)
\]

\[
Q^{[\neg A]} \in Q_{\text{fill}} \text{ if } \text{red}_\emptyset(Q) = \Box|\Phi_T \quad (4)
\]

\[
Q \in Q_\emptyset \text{ if } Q = \emptyset \quad (5)
\]

Queries in \(Q^A\) are relations in which the values of \([A]\)-attributes are populated with values from \(K_A\), and values for attributes not in \([A]\) 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 in \(Q_{\text{fill}}\) are like \(Q^A\)-queries with the difference that every tuple is annotated with \(\top\). Queries in \(Q_\emptyset\) are empty relations.

**Example 4.1.** Given the domain \(K_A = \{a_1, a_2, a_3\}\), 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 \(Q_{\text{fill}}\)-query.

\[
Q^A\text{-relation } X \\
\begin{array}{ccc|c}
A & B & C & \Phi \\
\hline
a_1 & 1 & 1 & x_1 \\
a_2 & 1 & 1 & x_2 \\
a_3 & 1 & 1 & x_3 \\
\end{array}
\]

\[
Q_{\text{fill}}\text{-relation } Y \\
\begin{array}{ccc|c|c}
A & B & C & \Phi \\
\hline
a_1 & 1 & 1 & \top \\
a_2 & 1 & 1 & \top \\
a_3 & 1 & 1 & \top \\
\end{array}
\]

The functional dependency \(A \rightarrow \Phi\) is trivially satisfied by \(\Phi(a_i) = x_i\).

Figure 6 shows how \(Q^A\), \(Q_{\text{fill}}\), and \(Q_\emptyset\)-queries are propagated through query operators: For query classes \(Q_1\) and \(Q_2\), the right-most column in the table shows the query class \(Q_1 Op Q_2\) for an operator \(Op\) that is join or difference.
Dichotomies for Queries with Negation in Probabilistic Databases

Fig. 6. Class membership of queries connecting classes \( Q_A, Q_{\text{fill}}, \) and \( Q_\emptyset \) with operators \( \otimes, - \).

Example 4.2. Continuing Example 4.1, the join \( X \Join Y \) on the corresponding \( A, B, C \) attributes of \( Q_A \)-query \( X \) and \( Q_{\text{fill}} \)-query \( Y \) yields the following relation:

\[
\begin{array}{cccccc}
A_x & A_y & B_x & B_y & C_x & C_y \\
\hline
a_1 & a_1 & a_2 & a_2 & a_3 & a_3 \\
\end{array}
\]

This join satisfies the conditions of a \( Q_A \)-query as suggested by the rule \( Q_A \Join Q_{\text{fill}} \rightarrow Q_A \) in Fig. 6. Similarly, the difference \( Y - X \) is also a \( Q_A \)-query:

\[
\begin{array}{ccc}
A_y & B_y & C_y \\
\hline
x_1 & ~x_1 & ~x_1 \\
\end{array}
\]

For a query containing a \( Q_A \)-relation \( X^{[A]} \), we can populate its relations such that it becomes a \( Q_A \)-query and thus satisfies Equation (2):  

\[ \text{Lemma 4.3.} \quad \text{Given an attribute } A, \text{ a relation } X \text{ exporting } A, \text{ and a query } Q \text{ containing } X \text{ and exporting } A. \text{ If } X \in Q_A, \text{ then the relations in } Q \text{ can be filled such that } Q \in Q_A. \]  

\[ \text{Proof.} \quad \text{Let } Op_\rightarrow \text{ be the set of difference operators in } Q \text{ that do not have } X \text{ as a right descendant. We partition the relations of } Q \text{ into three sets:} \]

\[
\text{rels}_X = \{X\} \\
\text{rels}_\emptyset = \text{relations that are right descendants of a } Op_\rightarrow \text{ operator} \\
\text{rels}_{\text{fill}} = \text{all other relations} \\
\]

We populate every \( \text{rels}_{\text{fill}} \) relation as a \( Q_{\text{fill}} \)-query and every \( \text{rels}_\emptyset \) relation as a \( Q_\emptyset \)-query. 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:

\[
\begin{align*}
&- Q_L \Join Q_R, \text{ where without loss of generality } Q_L \text{ contains } X. \text{ Then, } Q_L \text{ is a } Q_A \text{-query, } Q_R \text{ contains a relation from } \text{rels}_{\text{fill}} \text{ and is a } Q_{\text{fill}} \text{-query. Hence, } Q_L \Join Q_R \text{ is a } Q_A \text{-query.} \\
&- Q_L - Q_R, \text{ where } Q_L \text{ contains } X. \text{ Then the difference operator is in } Op_\rightarrow \text{ and } Q_R \text{ is a } Q_\emptyset \text{-query, } Q_L \text{ is a } Q_A \text{-query. Hence, } Q_L - Q_R \text{ is a } Q_A \text{-query.} \\
&- Q_L - Q_R, \text{ where } Q_R \text{ contains } X. \text{ Then, } Q_R \text{ is a } Q_A \text{-query, } Q_L \text{ contains a relation from } \text{rels}_{\text{fill}} \text{ and is a } Q_{\text{fill}} \text{-query. Hence, } Q_L - Q_R \text{ is a } Q_A \text{-query.} \\
\end{align*}
\]
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 $\Phi_Q(a) = \neg\Phi_X(a)$.

**Example 4.4.** Consider the query in Fig.[7](#) (top). We would like to preserve the attribute $A_x$ in relation $X$, where we use $K_A = \{a_1, a_2, a_3\}$ as domain for $[A]$ and annotations $\Phi(a_i) = x_i$. The bottom-left graph shows the partition of $Q$'s relations into $\text{rels}_X$, $\text{rels}_\emptyset$, and $\text{rels}_{\text{fill}}$ relations (bottom-left), assuming that relation $X$ and its annotations are to be preserved by the filling. The bottom-right figure shows how the query classes of the sub-queries of $Q$ are propagated through query operators for two cases: (left) we preserve values and annotations for attributes in $[A]$; and then all sub-queries rooted in operators on the path from $X$ to the root of $Q$ are $Q^A$-queries; (right) we preserve for attributes in $[A]$ and $[B]$ and then these sub-queries become $Q^{AB}$-queries. Since $X$ has odd polarity in $Q$, the annotations of tuples in $Q$ are the negated annotations of the corresponding tuples in $X$.

![Diagram](#)
Then, the relation represented by \( Q \) is:

\[
\begin{array}{c|c|c|c|c}
A_r & B_r & D_r & \Phi \\
\hline
a_1 & \square & \square & \neg x_1 \\
a_2 & \square & \square & \neg x_2 \\
a_3 & \square & \square & \neg x_3 \\
\end{array}
\]

\( Q \) preserves \([A]\) and the annotations of relation \( X \); the annotations of tuples in \( Q \) are the negation of the corresponding annotations in \( X \) since \( X \) has odd polarity in \( Q \) \( \square \)

4.2. Preserving the data of two attributes

We can extend the previous construction scheme to the case of two attributes \( A \) and \( B \) from distinct classes. We first generalise the notation of \( A \)-reducible queries.

Let \( \Phi^{AB} \) be a total function on \( K_A \times K_B \), and let \( \Phi^A \) be a total function on \( K_A \cup K_A \times K_B \) such that \( \Phi^A(a) = \bigvee_{b \in K_B} \Phi^A(a, b) \) for all \( a \in K_A \); \( \Phi_T \) is the constant function \( \Phi_T(.) = T \). As before, a query (or relation) \( Q \) is \( A \)-reducible to \((K_A, \Phi^A)\), if

\[
\pi_A^\Phi(Q) = \{(a|\Phi^A(a)) \mid a \in K_A\}
\]

\[
\pi_C(Q) = \{\square\}
\]

for any attribute \( A \in [A] \) and attributes \( C \notin [A] \)

Similarly, \( Q \) is \( AB \)-reducible to \((K_A \times K_B, \Phi^{AB})\) if

\[
\pi_A^{AB}(Q) = \{(a, b|\Phi^{AB}(a, b)) \mid a \in K_A, b \in K_B\}
\]

\[
\pi_C(Q) = \{\square\}
\]

for any attribute \( C \notin [A] \cup [B] \)

By \( \text{red}_{AB}(Q) = K_A \times K_B|\Phi^{AB} \) we denote that \( Q \) is \( AB \)-reducible to \((K_A \times K_B, \Phi^{AB})\).

We next define the following classes of queries:

\[
Q^{[A][B]} \in Q^A \text{ if } \text{red}_A(Q) = K_A|\Phi^A \text{ or } \text{red}_A(Q) = K_A|\neg \Phi^A \quad (6)
\]

\[
Q^{[A][B]} \in Q^A \text{ if } \text{red}_{AB}(Q) = K_A \times K_B|\Phi^A \text{ or } \text{red}_{AB}(Q) = K_A \times K_B|\neg \Phi^A \quad (7)
\]

\[
Q^{[A][B]} \in Q^{AB} \text{ if } \text{red}_{AB}(Q) = K_A \times K_B|\Phi^{AB} \text{ or } \text{red}_{AB}(Q) = K_A \times K_B|\neg \Phi^{AB} \quad (8)
\]

\[
Q^{[A][B]} \in Q_{\text{full}} \text{ if } \text{red}_A(Q) = K_A|\Phi_T \quad (9)
\]

\[
Q^{[A][B]} \in Q_{\text{full}} \text{ if } \text{red}_{AB}(Q) = K_A \times K_B|\Phi_T \quad (10)
\]

\[
Q^{[A][B]} \in Q_{\text{full}} \text{ if } \text{red}_B(Q) = \square|\Phi_T \quad (11)
\]

\[
Q \in Q_{\emptyset} \text{ if } Q = \emptyset \quad (12)
\]

Queries from these classes are propagated by query operators as depicted in Fig. 6.

We now extend Lemma 4.3 to the case of one or two attributes:

**Lemma 4.5.** Given a query \( Q \), attributes \( A \) and \( B \) from distinct classes in \( Q \), and a relation \( X \) in \( Q \). If \( X \in Q^A \) and \( \text{exports} \) \( A \) and not \( B \), then the relations in \( Q \) can be filled such that \( Q \in Q^A \) regardless whether \( Q \) \( \text{exports} \) \( A \) and not \( B \), cf. Equation (6), or \( \text{exports} \) both \( A \) and \( B \), cf. Equation (7). If \( X \in Q^{AB} \) and \( Q \) \( \text{exports} \) both \( A \) and \( B \), then the relations in \( Q \) \( \text{can be filled} \) such that \( Q \in Q^{AB} \), cf. Equation (8).

**Proof.** Let \( \mathcal{OP}_{\_} \) be the set of operators that do not have \( X \) as a right descendant. We partition the relations of \( Q \) into three sets:

\[
\text{rels}_X = \{X\}
\]

\[
\text{rels}_0 = \text{set of relations that are right descendants of an operator in } \mathcal{OP}_{\_}
\]

\[
\text{rels}_{\text{full}} = \text{all other relations}
\]
We first show $Q^{[A][B]} \in Q^A$ given an $A$-reducible relation $X \in Q^A$, cf. Equation (7). The relations in $Q$ are populated depending on their types. Every relfill relation is populated as a $Q_{fill}$-query: Each attribute in $[A]$ ([B]) with constants in $K_A$ (respectively, $K_B$), and the other attributes with $\blacksquare$. Each relation in relfill that exports attributes in both $[A]$ and $[B]$ is populated such that: its projection on any attribute in $[A]$ ([B]) is $K_A$ (respectively, $K_B$); its projection on any pair $(A, B)$ of attributes with $A \in [A]$ and $B \in [B]$ is $K_A \times K_B$: all attributes not in $[A]$ and $[B]$ take the value $\blacksquare$. Every rel9 relation is a $Q_9$-query and thus kept empty.

We use a similar inductive argument as in the proof of Lemma 4.3 and show that every operator on the path from $X$ to the root of $Q$ is a $Q^X$-query. Note however, that the lowest $\blacksquare$-operator on the path from $X$ to the root of $Q$ that introduces a $[B]$-attribute marks the transition from a $Q^A$ sub-query of type $Q^{[A][\neg B]}$ (Equation (6)) to $Q^{[A][B]}$ (Equation (7)). That is, the tuples $(a|\Phi(a))$ in $X$ are expanded to tuples $(a,b|\Phi(a))$ by means of the cross product between the $Q^A$-relation that does not export $[B]$ and the $Q_{fill}$-relation that does export $[B]$.

Finally, the case of a query $Q^{[A][B]}$ that contains a $Q^{AB}$-relation $X^{[A][B]}$ follows as above and $Q \in Q^{AB}$, that is, $Q$ is equivalent to $X$ with respect to attributes $[A]$ and $[B]$ and the annotations of the $(a,b)$-tuples.

The remark following Lemma 4.3 regarding the polarity and the sign of the annotations in $X$ and $Q$ carries over to our generalisation in Lemma 4.5.

**Example 4.6.** Referring again to the query in Fig. 7 we would now like to preserve relation $X$ with respect to $[A]$ and $[B]$. We take the domains $K_A = \{a_1, a_2, a_3\}$ and $K_B = \{b_1, b_2\}$ and the relation $X \in Q^{AB}$ as depicted in Fig. 8.

The sets of relations relsX, relsfill and rels9 are identical to those in Example 4.4 and are depicted in the bottom-left graph in Fig. 7. The $Q^{AB}$ and $Q_{fill}$ classes through the query is depicted in the bottom-right graph. However, the relations represented by the different sub-queries may now be of different types.

Let us first consider the sub-query $Q_{RWU}$ consisting of relations $R, W, U$. Attribute $A_r$ is filled with $K_A$, and attributes $C_r, C_w, D_w$ are set to $\blacksquare$; relation $U$ is set to $\emptyset$; all annotations are $\top$. Then, $Q_{RWU} \in Q_{fill}$-query by Equation (9). The relation $Q_{RWU}$ is shown in Fig. 8.

In relation $T$, attribute $B_t$ is filled with $K_B$ and $D_t$ is filled with $\blacksquare$; its annotations are $\top$. The join between $Q_{RWU}$ and $T$ enforces an equality condition $D_w = D_t$ and yields the relation $Q_{RWUT}$ as depicted in Fig. 8. $Q_{RWUT}$ is a $Q_{fill}$-query by virtue of Equation (10).
On the right side of the top most operator in \( Q \), the sub-query \( Q_{VX S} \) consisting of relations \( V, X, S \) is in \( Q^{AB} \); it is depicted in Fig. 8. This leads to the \( Q_{RVUT}^{AC} \)-query \( Q = Q_{RVUT} - Q_{VX S} \) as depicted in the figure.

5. Non-Hierarchical 1RA- Queries Are #P-Hard

In this section we show the following result:

**Lemma 5.1.** 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 \( \mathcal{D} \) such that \( \Psi \) 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 (Definition 5.3) 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 match (Definition 5.7), for which a database construction scheme is possible such that the query result becomes annotated by \( \Psi \).

The proof steps are summarised as follows:

\[
Q \text{ is non-hierarchical } \iff \text{Proposition 5.4 } \iff Q \text{ has a match with a pattern in Fig. 9} \iff \text{Lemma 5.3 } \iff Q \text{ has an annotation-preserving match with a pattern in Fig. 9} \iff \text{Lemma 5.10 } \iff Q \text{ is #P-hard.}
\]

5.1. Patterns and matches

We next define hard minimal query patterns and matches.

**Definition 5.2.** A pattern \( P \) over attributes \( A, B \) and relational operators \( \text{Op}_1, \text{Op}_2 \in \{\times, -\} \) is a binary tree with leaves \( A, B, \) and \( AB \), root node \( \text{Op}_1 \), and inner node \( \text{Op}_2 \).

There are 48 different patterns: There are two distinct unlabeled binary trees with three leaves, the two operators can each be either join (\( \times \)) or difference (\( - \)), and there are 6 possible orders of the labels \( A, AB, \) and \( B \). Figure 9 shows 24 of the 48 patterns and omits for each pattern the symmetric pattern obtained by swapping leaves \( A \) and \( B \). By exploiting symmetries of the join operator in queries and patterns, it suffices to only consider 14 patterns (those shown in dark colour and not the source of directed arrows).

**Definition 5.3.** A 1RA- query \( Q \) matches a pattern \( P \) if there is mapping from the nodes of \( P \) to nodes in the parse tree of \( Q \) that preserves ancestor-descendant relationships:

\[ A \mapsto R[A][\neg B], \ B \mapsto T[\neg A][B], \ AB \mapsto S[A][B], \ \text{Op}_1 \mapsto \text{Op}_1, \text{ and Op}_2 \mapsto \text{Op}_2. \]

We also say that \( Q \) is an \( (R,S,T) \)-match of \( P \) to emphasise which relations establish the match.

Figures 1 and 10 show examples of queries matching patterns. Pattern matching is intimately linked to the non-hierarchical property:

**Proposition 5.4.** A 1RA- query is non-hierarchical if and only if it matches one of the patterns in Fig. 9.

**Proof.** Let \( Q \) be a non-hierarchical query. By Definition 1.2, there are two attributes \( A \) and \( B \) such that \( Q \) contains relations \( R[A][\neg B], S[A][B], \) and \( T[\neg A][B] \). \( Q \) matches exactly the pattern \( P \) whose two operators correspond to the operators \( \text{Op}_1 \) and \( \text{Op}_2 \) in \( Q \). The patterns
are exhaustive in the sense that there is exactly one pattern per possible combination of the operators \( \times, \ldots \).

\( \Leftrightarrow \): Every query \( Q \) that matches a pattern contains three distinct relation symbols \( R, S, T \) that render \( Q \) non-hierarchical by Definition 1.2.

The notion of a match is further specialised to that of an annotation-preserving match. Whereas the database construction scheme detailed in Section 4 does not work for general matches, it does work for annotation-preserving matches. We first define left-deep operators.
Without loss of generality, we assume that \( T \) contains at least one relation that exports \([A]\) or \([B]\). Then the left sub-query of \( o_j \) contains at least one relation that exports \([A]\) or \([B]\). This relation together with a subset of \( R, S, T \) would establish a match of \( Q_j \) with some pattern; this is a contradiction.

**Definition 5.5.** An operator \( \text{Op} \) is left-deep in a 1RA\(^-\) query \( Q \) if \( \text{Op} \) is a left descendant of every difference operator on the path between the root of \( Q \) and \( \text{Op} \).

**Example 5.6.** In Fig. 10, 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 5.7.** A 1RA\(^-\) query \( Q \) is an annotation-preserving \((R, S, T)\)-match of a pattern \( P \) over attributes \( A \) and \( B \) and operators \( \text{Op}_1 \) and \( \text{Op}_2 \) if: (1) \( Q \) is an \((R, S, T)\)-match of \( P \); (2) For every difference operator \( \text{Op}_- \) in \( Q \), if \( \text{Op}_1 \) is a right descendant of \( \text{Op}_- \), then \( \text{Op}_- \) does not export \([A]\) or \([B]\); (3) If \( \text{Op}_2 \) is a left descendant of \( \text{Op}_1 \) in \( Q \), then \( \text{Op}_2 \) is left-deep in the sub-query rooted at \( \text{Op}_1 \). We say that \( Q \) is an annotation-preserving \((R, S, T)\)-match of \( P \) to emphasise the relations establishing the match.

Figure 10 shows examples of annotation-preserving matches. We next look closer at the connection between matches and annotation-preserving matches. Lemma 5.8 establishes that any query that matches a pattern necessarily has an annotation-preserving match with a possibly different pattern. 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 5.8.** 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 \( Q_1 \) is the first sub-query in the above order that matches a pattern in Fig. 9, then \( Q_1 \) is an annotation-preserving match of a pattern.

**Proof.** We show that \( Q \) is an annotation-preserving match of a pattern \( P' \), that is, \( Q \) satisfies the three properties from Definition 5.7. Let \( \text{Op}_1 \) and \( \text{Op}_2 \) be the two operators in \( Q \) as established by a match with a pattern \( P \).

**Property 1.** This is already satisfied by definition: \( Q_1 \) already matches a pattern \( P \). Without loss of generality, we assume that \( Q_1 \) is a \((R[A][\neg B], S[A][B], T[\neg A][B])\)-match of \( P \); the patterns are exhaustive in the sense that any arrangement of three relations \((R[A][\neg B], S[A][B], T[\neg A][B])\) and two operators corresponds to exactly one pattern. Note that \( o_i = \text{Op}_1 \) by construction.

**Property 2.** Proof by contradiction. Assume that there is an operator \( o_j = \neg \) that is an ancestor of \( o_j = \text{Op}_1 \) such that \( \text{Op}_1 \) is a right descendant of \( o_j \). It holds that \( j < i \) due to the in-order of the query operators. Assume that \( o_j \) exports \([A]\) or \([B]\). Then the left sub-query of \( o_j \) contains at least one relation that exports \([A]\) or \([B]\). This relation together with a subset of \( R, S, T \) would establish a match of \( Q_j \) with some pattern; this is a contradiction.
to the assumption that $Q_i$ is the first sub-query (in the in-order sequence of sub-queries) to establish a match. Hence $o_j$ cannot export $[A]$ or $[B]$.

**Property 3.** Proof by contradiction. We are given that $Op_2$ is a left descendant of $Op_1$. Now assume that $Op_2$ is not left-deep in the left sub-query $Q_L$ of $Op_1$. Then, there is a top-most difference operator in $Q_L$, say $Op_-$, such that $Op_2$ is its right descendant. There are the following cases:

- Case 1: $R$ and $T$ are descendants of $Op_2$. Then $S$ is a right descendant of $Op_1$. Furthermore, $Op_-$ exports $[A]$ and $[B]$ since every operator on the path between $R$ ($T$, respectively) must export $[A]$ ($[B]$, respectively) in order for $Op_1$ to establish an equality or mapping for the attributes in $[A]$ and $[B]$ in $R$, $T$, and $S$. Thus the left sub-query of $Op_-$ contains relations $X^{[A][\neg B]}$ and $Z^{[\neg A][B]}$, or it contains a relation $Y^{[A][B]}$. In the former case, the three relations $R, S, T$ establish an annotation-preserving match, with $Op_2 = Op_-$ and $Op_1$ as before. The latter case is a contradiction to the assumption that $Q_i$ is the first sub-query in the in-order sequence of sub-queries of $Q$ that matches a pattern, because the sub-query rooted at $Op_-$ precedes $Q_i$ in $Q$’s in-order and matches a pattern via $R, Y, T$.

- Case 2: $R$ and $S$ are descendants of $Op_2$. Then $T$ is a right descendant of $Op_1$, and $Op_-$ exports $[B]$. Thus the left sub-query of $Op_-$ contains a relation $Y^{[A][B]}$, or it contains a relation $X^{[\neg A][B]}$. In the former case, the three relations $R, Y, T$ establish a lineage-preserving match, with $Op_2 = Op_-$ and $Op_1$ as before. The latter case is a contradiction to the assumption that $Q_i$ is the first sub-query in the in-order sequence of sub-queries of $Q$ that matches a pattern, because the sub-query rooted at $Op_-$ precedes $Q_i$ in $Q$’s in-order and matches a pattern via $X, S, T$.

- Case 3: $T$ and $S$ are descendants of $Op_2$. Symmetric to case 2.

**Example 5.9.** Consider the query $Q_2$ in Fig. 10. 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 $P_{4.3}$.

**5.2. Hardness reductions**

The 24 patterns in Fig. 9 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 5.10.** The data complexity of any $1RA^-$ query that is an annotation-preserving match of one of the patterns in Fig. 9 is $\#P$-hard.

Putting together Proposition 5.4 and Lemmata 5.8 and 5.10 we obtain that the data complexity of all non-hierarchical $1RA^-$ queries is $\#P$-hard.

The proof of Lemma 5.10 goes over each pattern case and shows hardness via a reduction from the $\#2DNF$ problem: we only need to consider 14 distinct patterns, since, as shown in Fig. 9, ten patterns are equivalent to other ones.

Let $Q$ be a query that is an annotation-preserving $(R, S, T)$-match for a pattern $P$, and let $\Psi = \bigvee_{(i,j) \in E} \neg t_i y_j$ be a $2DNF$ formula with $|E|$ clauses over disjoint variable sets $X$ and $Y$. We construct in polynomial time a tuple-independent database $D$ using the database construction scheme in Section 4 such that the annotation of the query result $Q(D)$ is either $\Psi$ and hence $P_{Q(D)} = P_{\Psi} = \#\Psi \cdot 2^{-|\text{vars}(\Psi)|}$, or $\neg \Psi$ and then $P_{Q(D)} = 1 - P_{\Psi}$.

In the following reductions, we use $K(X)$ to denote the set of constants defined by the set $X$ of Boolean variables; similar for $Y$ and the union $X \cup Y$. While these constants are used for attributes in relations, their corresponding variables are used in propositional formulas for the special annotation column $\Phi$.

By Definition 4.7, the query $Q$ contains two distinct operators $Op_1$ and $Op_2$ and relations $R^{[A][\neg B]}$, $S^{[A][B]}$, $T^{[\neg A][B]}$. In the following, sub-queries $Q_R$, $Q_S$, $Q_T$ of $Q$ are defined to be the left or right sub-queries of $Op_1$ or $Op_2$ that contain exactly one of $R$, $S$, or $T$. 

ACM Transactions on Database Systems, Vol. 1, No. 1, Article 1, Publication date: October 2015.
Interestingly, the reduction for some patterns such as those of any difference operator on the path between $\Phi_{\mathit{R}}$ and $\Phi_{\mathit{S}}$ respectively, of the operator $\mathit{Op}_1$, that is, it is the left descendant of any difference operator on the path between $\mathit{Op}_1$ and $\mathit{Op}_2$.

The remainder of the proof treats the case of each pattern separately.

Amongst the patterns, $\mathit{P}_{1.1}$ is the only one needed to show hardness of non-hierarchical $\mathit{1RA}^-$ queries without negation, that is, of non-repeating conjunctive queries studied in prior work [Dalvi and Suciu 2007a]. Interestingly, the reduction for some patterns such as $\mathit{P}_{2.3}$ establishes that a query matching the pattern can be already hard for databases in which one relation is probabilistic and all other relations are certain.

5.2.1. Reductions for patterns $\mathit{P}_{1.1}$, $\mathit{P}_{2.1}$, $\mathit{P}_{3.1}$, and $\mathit{P}_{3.4}$.

Pattern $\mathit{P}_{1.1}$. Let $Q$ be a query that is an annotation-preserving match of $\mathit{P}_{1.1}$. Figure 11 (left) depicts such a query $Q$, where $Q_R, Q_S, Q_T, Q_{RT}, Q_{RS}$ denote sub-queries of $Q$. By Definition 5.7, $Q$ contains operators $\mathit{Op}_1 = \mathit{A}\Phi$ and $\mathit{Op}_2 = \mathit{A}\Phi$ and relations $R[\mathit{A}][\mathit{B}], S[\mathit{A}][\mathit{B}], T[\mathit{A}][\mathit{B}]$. Every operator on the path $R - \mathit{Op}_2 - \mathit{Op}_1 - S$ exports $\mathit{A}$, and every operator on the path $T - \mathit{Op}_2 - \mathit{Op}_1 - S$ exports $\mathit{B}$. Moreover, the operator $\mathit{Op}_1$ expresses a join on both $\mathit{A}$ and $\mathit{B}$.

Since $Q$ is an annotation-preserving match, it satisfies the following additional structural properties: (1) If $\mathit{Op}_1$ is a right descendant of a difference operator, then this operator does not export $\mathit{A}$ or $\mathit{B}$. (2) Operator $\mathit{Op}_2$ is left-deep in $Q_{RT}$, that is, it is the left descendant of any difference operator on the path between $\mathit{Op}_1$ and $\mathit{Op}_2$.

We populate the relations $R, S, T$ as $Q^A$, $Q^B$, and $Q^{AB}$-relations following Equations 6 and 8. We thus fill the relation $R$ with constants from $\mathbf{K}(X)$ for $\mathit{A}$-attributes and annotations $\Phi_R$, the relation $T$ with constants from $\mathbf{K}(Y)$ for $\mathit{B}$-attributes and annotations $\Phi_T$, and the relation $S$ with the Cartesian product of the two sets of constants for attributes $\mathit{A}$.

---

**Fig. 11.** Schematic illustrations of queries that are annotation-preserving matches for pattern $\mathit{P}_{1.1}$ (left) and $\mathit{P}_{2.2}$ (right). A curly edge indicates that further operators may occur on this path. By Definition 5.7, the operator $\mathit{Op}_2$ is left-deep in the left sub-query $Q_{RT}$, or $Q_{RS}$ respectively, of the operator $\mathit{Op}_1$, that is, it is the left descendant of any difference operator on the path between $\mathit{Op}_1$ and $\mathit{Op}_2$.

**Fig. 12.** Relations $R, S, T$ for the hardness reduction of a query with an annotation-preserving match for pattern $\mathit{P}_{1.1}$. The filling of $S$ assumes that the formula $\Psi$ is $x_1 x_2$ and thus tuples $(x_1, y_1)$ and $(x_1, y_2)$ are the only $S$-tuples annotated with $\top$ (assuming even polarity of $S$ in $Q_S$). To avoid clutter, the full polarity function is omitted from the annotation columns; for relation $R$, $\neg \mathit{pol}^1$ stands for $\neg \mathit{pol}^1(Q_R;R)$, for $T$ the polarity is $\neg \mathit{pol}^1(Q_T;T)$, and for $S$ it is $\neg \mathit{pol}^1(Q_S;S)$.
and $[B]$, and annotations $\Phi_S$. All other attributes are set to \textbf{[]} The annotation functions are defined as follows:

$$\Phi_R : K(X) \cup K(X) \times K(Y) \to X \cup Y$$

$$\Phi_S : K(X) \times K(Y) \to X \cup Y$$

$$\Phi_T : K(Y) \cup K(Y) \times K(X) \to X \cup Y$$

This database construction can be given more concisely using the notation from Section 4

- $\text{red}_A(R) = K(X)|_\text{pol}^{R,R} \Phi_R$
- $\text{red}_{AB}(S) = K(X) \times K(Y)|_\text{pol}^{S,S} \Phi_S$
- $\text{red}_B(T) = K(Y)|_\text{pol}^{T,T} \Phi_T$

Recall that the function $\text{pol}(Q,R)$ defines the even (0) and odd (1) polarity of a relation symbol $R$ in the query $Q$. We use the convention $\lnot \Phi = \neg \Phi$ and $\neg \Phi = \lnot \Phi$.

Figure 12 depicts the relations $R$, $S$, and $T$. By applying the results of Section 4 and Lemma 5.2, the remaining relations in $Q_R$, $Q_S$, and $Q_T$ can be filled such that $Q_R \in Q^A$, $Q_T \in Q^B$, and $Q_S \in Q^{AB}$; that is, the values and annotations of $R$, $S$, and $T$ are preserved in $Q_R$, $Q_S$, and respectively $Q_T$. Since the filling of $R,S,T$ accounts for their polarity in their sub-queries $Q_R$, $Q_S$, $Q_T$, the latter relations take the following simple form\footnote{The match of $Q$ with pattern $P_{1,1}$ does not prohibit $O_{P_2}$ from expressing a join on $[B]$. In that case, the sub-queries of $O_{P_2}$ are $Q_R^{A}[B]$ and $Q_S^{A}[B]$ and satisfy $\text{red}_{AB}(Q_R) = K(X) \times K(Y)|_\text{pol}^{R,R} \Phi_R$ and $\text{red}_{AB}(Q_T) = K(Y) \times K(X)|_\text{pol}^{T,T} \Phi_T$.}

$$\text{red}_A(Q_R) = K(X)|_\Phi_R \text{ red}_{AB}(Q_S) = K(X) \times K(Y)|_\Phi_S \text{ red}_B(Q_T) = K(Y)|_\Phi_T$$

Let us now define the following annotations:

$$\Phi_{RT} : K(X) \times K(Y) \to X \cup Y$$

$$\Phi_{RST} : K(X) \times K(Y) \to X \cup Y$$

The sub-query $Q_R \Join Q_T$ is populated as follows:

$$\text{red}_{AB}(Q_R \Join Q_T) = K(X) \times K(Y)|_\Phi_{RT}$$

By applying Lemma 6.5 to $Q_{RT}$, the annotations in the sub-query $Q_R \Join Q_T$ can be preserved by $Q_{RT}$. Since $O_{P_2}$ is left-deep in $Q_{RT}$, $O_{P_2}$ has even polarity in $Q_{RT}$ and hence the annotations of tuples in $Q_{RT}$ carry the same sign as in $Q_R \Join Q_T$. This yields

$$\text{red}_{AB}(Q_{RT}) = K(X) \times K(Y)|_\Phi_{RT} \text{ red}_{AB}(Q_{RT} \Join Q_S) = K(X) \times K(Y)|_\Phi_{RST}.$$

The sub-query $Q_{RST} = Q_{RT} \Join Q_S$ rooted at $O_{P_1}$ thus has exactly one tuple $(x_i,y_j)$ annotated with $x_i y_j$ for each such clause in $\Psi$, and one tuple $(x_i,y_j)$ annotated with $\bot$ for each clause $x_i y_j$ not in $\Psi$.

Since by Definition 5.7 there does not exist a difference operator above $O_{P_1}$ that exports $[A]$ or $[B]$, we can use the techniques of Lemma 5.5 to fill the relations representing sub-queries of $Q$ that are not descendants of $O_{P_1}$ such that the annotations of $Q_{RST}$ are preserved by any operator above $O_{P_1}$. Finally, since by Definition 5.3 $Q$ does not export $[A]$ or $[B]$, those attributes are eventually projected out above $O_{P_1}$, yielding as result a single tuple annotated with the disjunction of the annotations of $Q_{RST}$. If this projection is
followed by difference operators (that do not export \([A]\) or \([B]\)), then each of these difference operators will flip the sign of the annotation, that is, the annotation of \(Q\) is \(\neg^{\text{pol}}(Q,\text{Op}_1)\). The probability \(P_{\text{Q}(D)}\) of query \(Q\) on the database constructed above is then \(P_\Phi\) or \(1 - P_\Phi\).

**Pattern \(P_{2.2}\).** A query that is an annotation-preserving match for \(P_{2.2}\) has the same form as depicted in Fig. 11 (left), except that \(\text{Op}_2\) = \(\neg\). The annotation functions \(\Phi_R, \Phi_S, \Phi_{RT},\) and \(\Phi_{RST}\) are as in the case of pattern \(P_{1.1}\), and the annotation function \(\Phi_T\) carries an additional negation to account for the flipped polarity of \(\text{T}\) (when compared to pattern \(P_{1.1}\)) due to the difference operator \(\text{Op}_2\):

\[
\Phi_T : K(Y) \cup K(Y) \times K(X) \rightarrow X \cup Y \quad \Phi_T(y_1) = \Phi_T(y_1, x_1) = \neg y_1
\]

We fill the relations \(R, S,\) and \(T\) using \(\text{red}_A(R), \text{red}_A(S),\) and \(\text{red}_B(T)\) as for \(P_{1.1}\).

The operator \(\text{Op}_2\) exports \([A]\) and \([B]\) and thus the queries \(Q_R\) and \(Q_T\) export both \([A]\) and \([B]\). These queries represent the following relations:

\[
\text{red}_A(Q_R) = K(X) \times K(Y)|\Phi_R \
\text{red}_A(Q_S) = K(X) \times K(Y)|\Phi_S \
\text{red}_B(Q_T) = K(Y) \times K(X)|\Phi_T
\]

The sub-query \(Q_R - Q_T\) is thus populated following \(\text{red}_A(Q_R - Q_T) = K(X) \times K(Y)|\Phi_{RT}\). The remainder of the reduction is identical to the case of pattern \(P_{1.1}\).

**Patterns \(P_{1.3}\) and \(P_{1.4}\).** The reductions are identical to the cases of patterns \(P_{1.1}\) and respectively \(P_{2.2}\), except for the definition of \(\Phi_S\) which carries an extra negation to account for the swapped polarity of \(S\):

\[
\Phi_S : K(X) \times K(Y) \rightarrow X \cup Y \quad \Phi_S(x_i, y_j) = \begin{cases} 
\bot & \text{if } (i, j) \in E \\
\top & \text{if } (i, j) \notin E
\end{cases}
\]

**5.2.2. Reduction for patterns \(P_{2.2}, P_{2.3},\) and \(P_{2.4}\).** Let \(Q\) be a query that is annotation-preserving match for one of these patterns. Such a query is depicted in Fig. 11 (right) for the case of \(P_{2.2}\). The query \(Q\) satisfies these structural constraints:

- Any operator on the path \(R - \text{Op}_2 - S\) exports \([A]\), and every operator on the path \(S - \text{Op}_2 - \text{Op}_1 - T\) exports \([B]\). The operator \(\text{Op}_2\) expresses an equality condition on \([A]\), and the operator \(\text{Op}_1\) expresses an join condition (for patterns \(P_{2.1}\) and \(P_{2.2}\)) or a difference mapping (for patterns \(P_{2.3}\) and \(P_{2.4}\)) on \([B]\).
- If \(\text{Op}_1\) is a right descendant of a difference operator, then this operator does not export \([A]\) or \([B]\).
- The operator \(\text{Op}_2\) is left-deep in \(Q_{RS}\), that is, it is the left descendant of any difference operator on the path between \(\text{Op}_1\) and \(\text{Op}_2\).

**Pattern \(P_{2.2}\).** Relations \(R, S, T\) are filled using \(\Phi_R, \Phi_S,\) and \(\Phi_T\) exactly as in the case of pattern \(P_{1.3}\). Additionally, define the following annotation functions:

\[
\Phi_{RS} : K(Y) \cup K(Y) \times K(X) \rightarrow X \cup Y \\
\Phi_{RS}(y_1, x_1) = \begin{cases} 
x_i & \text{if } (i, j) \in E \\
\bot & \text{if } (i, j) \notin E
\end{cases}
\]

\[
\Phi_{RS}(y_1) = \bigvee_{(i,j)\in E} x_i
\]

\[
\Phi_{RST} : K(Y) \cup K(Y) \times K(X) \rightarrow X \cup Y \\
\Phi_{RST}(y_1, x_1) = \begin{cases} 
x_i y_j & \text{if } (i, j) \in E \\
\bot & \text{if } (i, j) \notin E
\end{cases}
\]

\[
\Phi_{RST}(y_1) = y_j \bigvee_{(i,j)\in E} x_i
\]
Then \( Q_R - Q_S \) and \( Q_R S \) are \( Q^B \)-queries that satisfy
\[
\text{red}_B(Q_R - Q_S) = K(Y)|\Phi_{RS} \quad \text{red}_B(Q_R S) = K(Y)|\Phi_{RS}
\]
and \( Q_T \) is a \( Q^B \)-query with
\[
\text{red}_B(Q_T) = K(Y)|\Phi_T
\]
Finally, the join \( Q_R S \Join Q_T \) satisfies
\[
\text{red}_B(Q_R S \Join Q_T) = K(Y)|\Phi_{RST}
\]
and the reasoning about operators above \( Op_1 \) is as in the case of pattern \( P_{1.1} \). The eventual projection \( \pi_{[B]} \) yields the nullary relation with one tuple annotated with \( \neg_{\text{pol}}(Q,Op_1)|\Psi \).

**Pattern \( P_{2.3} \).** We fill the relations \( R, S, \) and \( T \) as in the case of pattern \( P_{1.2} \). The analysis is identical to \( P_{2.2} \).

**Pattern \( P_{2.4} \).** This case is identical to \( P_{2.2} \), where the relation \( T \) annotation uses the following function:
\[
\Phi_T : K(Y) \cup (K(Y) \times K(X)) \rightarrow X \cup Y \quad \Phi_T(y_j) = \Phi_T(y_j,x_i) = \neg y_j
\]

### 5.2.3. Reduction for patterns \( P_{3.2} \) and \( P_{4.4} \)

The structure of queries matching any of these patterns is equivalent to those matching a pattern \( P_{2.x} \) with relations \( R \) and \( S \) swapped. The structural constraints remain the same and hence the reductions are very similar.

**Pattern \( P_{3.2} \).** This case is similar to \( P_{2.2} \) where \( \Phi_R \) and \( \Phi_S \) are negated:
\[
\Phi_R : K(X) \cup (K(X) \times K(Y)) \rightarrow X \cup Y \quad \Phi_R(x_i) = \Phi_R(x_i,y_j) = \neg x_i
\]
\[
\Phi_S : K(X) \times K(Y) \rightarrow X \cup Y \quad \Phi_S(x_i,y_j) = \begin{cases} \top & \text{if } (i,j) \in E \\ \bot & \text{if } (i,j) \notin E \end{cases}
\]
Then, with \( \Phi_{RS} \) as for \( P_{2.2} \), sub-queries \( Q_S - Q_R \) and \( Q_R S \) are \( Q^B \)-queries and satisfy
\[
\text{red}_B(Q_S - Q_R) = K(Y)|\Phi_{RS} \quad \text{red}_B(Q_R S) = K(Y)|\Phi_{RS}
\]
The remainder of this reduction is identical to the case of \( P_{2.2} \).

**Pattern \( P_{3.4} \).** This case is identical to the case of \( P_{2.4} \) where we use the annotation functions \( \Phi_R \) and \( \Phi_S \) for the pattern \( P_{3.2} \).

### 5.2.4. Reduction for patterns \( P_{4.3} \) and \( P_{4.4} \)

For queries matching the patterns \( P_{4.3} \) or \( P_{4.4} \), it is only possible to directly encode the 2DNF formula \( \Psi \) as a database \( D \) such that the annotation of \( Q(D) \) is exactly \( \Psi \), if the polarity of \( Op_2 \) is odd in \( Q_{RT} \). In the case of even polarity, we show that we can derive a database \( D \) and another formula \( \Upsilon \) from \( \Psi \) such that \( P_Q(D) = P_T \) and linearly many calls to an oracle for \( P_T \) suffice to determine \( \#\Psi \).

Let \( \Psi = \bigvee_{(i,j) \in E} x_i y_j = \psi_1 \lor \cdots \lor \psi_{|E|} \) be a 2DNF formula with \( |E| \) clauses over disjoint variable sets \( X \) and \( Y \). Let \( \Theta \) be the set of assignments of variables \( X \cup Y \). Then the number of models of \( \Psi \) is defined by \( \#\Psi = \sum_{\theta \in \Theta} \theta = \Psi 1 \). If we partition \( \Theta \) into disjoint sets \( \Theta_0 \cup \cdots \cup \Theta_{|E|} \), such that \( \theta \in \Theta_i \) if and only if \( \theta \) satisfies exactly \( i \) clauses of \( \Psi \), then this sum can be equivalently written as
\[
\#\Psi = \sum_{\theta \in \Theta_1} 1 + \cdots + \sum_{\theta \in \Theta_{|E|}} 1 = |\Theta_1| + \cdots + |\Theta_{|E|}|.
\]
Fig. 13. 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.

We next show how to compute $|\Theta_i|$ using an oracle for $P_T$, 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

$$\Upsilon = \bigvee_{i=1}^{|E|} \neg z_i \land \neg \psi_i,$$

or, equivalently

$$\neg \Upsilon = \bigwedge_{i=1}^{|E|} (z_i \lor \psi_i) \quad (13)$$

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 $1 - P_T = P_{\neg T}$ can be expressed by conditioning on the number of satisfied clauses of $\Psi$:

$$P_{\neg T} = \sum_{k=0}^{|E|} P(\text{exactly } k \text{ clauses of } \Psi \text{ are satisfied}) \cdot P(\text{exactly } k \text{ clauses of } \Psi \text{ are satisfied})$$

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

Intuitively, the first term simplifies to $p_z^{|E| - k}$, because if exactly $k$ clauses $\psi_i$ are satisfied in $\neg \Upsilon$, then in order to satisfy the remaining $|E| - k$ clauses $(z_i \lor \psi_i)$ 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 $|\Theta_0|, \ldots, |\Theta|$. The $|E| + 1$ coefficients can be derived from $|E| + 1$ pairs $(p_z, P_T)$ using Lagrange’s polynomial interpolation formula. We conclude that $|E| + 1$ oracle calls to $P_T$ 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 one of the patterns $P_{4,3}$ and $P_{4,4}$; given this encoding, any algorithm that evaluates $P_{Q(T)}$ constitutes the above oracle. We give encodings for the two patterns $P_{4,3}$ and $P_{4,4}$ separately.

Pattern $P_{4,3}$. We use the illustration of a query matching $P_{4,3}$ in Fig. 13 (left). By Definition 5.7, 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 distinguish two cases depending on the polarity of $Op_2$ in the sub-query $Q_{RT}$: If the polarity is odd, we can use a filling similar to that of pattern $P_{1,1}$; if it is even we fill to obtain formula $Y$ as outlined above.
Case 1: Odd polarity ($\text{pol}(Q_{RT}, Op_2) = 1$). We fill $R, S, T$ like in the case of pattern $P_{1,1}$ 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 the annotation functions are as follows:

$$
\begin{align*}
\Phi_R : K(X) \cup K(X) \times K(Y) &\rightarrow X \cup Y & \Phi_R(x_1) = \Phi_R(x_1, y_j) = x_i \\
\Phi_S : K(X) \times K(Y) &\rightarrow X \cup Y & \Phi_S(x_1, y_j) = \begin{cases} \top & \text{if } (i, j) \in E \\
\bot & \text{if } (i, j) \notin E \end{cases} \\
\Phi_T : K(Y) \cup K(Y) \times K(X) &\rightarrow X \cup Y & \Phi_T(y_j) = \Phi_T(y_j, x_1) = y_j \\
\Phi_{RT} : K(X) \times K(Y) &\rightarrow X \cup Y & \Phi_{RT}(x_1, y_j) = x_i y_j \\
\Phi_{RST} : K(X) \times K(Y) &\rightarrow X \cup Y & \Phi_{RST}(x_1, y_j) = \begin{cases} x_i y_j & \text{if } (i, j) \in E \\
\bot & \text{if } (i, j) \notin E \end{cases}
\end{align*}
$$

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 $y_j$ and annotation $y_j$ for each variable $y_j \in Y$ that occurs in $\Psi$; $S$ consists of a tuple with $(A, B)$-values $(x_i, y_j)$ and annotation $\top$ for each clause $x_iy_j$ in $\Psi$. Recall that we turn variables to constants when used for attributes in relations. 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 attribute in $[A]$ and no attribute in $[B]$ is filled like $R$, but with annotation $\top$. Symmetrically, any relation with an attribute in $[B]$ and no attribute in $[A]$ is filled like $T$, but with annotation $\top$. Relations with attributes in both $[A]$ and $[B]$ become the Cartesian product of $K(X)$ and $Y$ and annotation $\top$. Any attribute that is neither in $[A]$ nor in $[B]$ is filled with the constant $\blacksquare$.

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

$$
\text{red}_{AB}(Q_{RT}) = K(X) \times K(Y) | \neg \Phi_{RT}.
$$

Then, $Q_S - Q_{RT}$ is a $Q^{AB}$ query populated as follows:

$$
\text{red}_{AB}(Q_S - Q_{RT}) = K(X) \times K(Y) | \Phi_{RST}.
$$

$$
\Phi_{RST}(x_1, y_j) = \begin{cases} x_i y_j & \text{if } (i, j) \in E \\
\bot & \text{if } (i, j) \notin E \end{cases}
$$

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

Case 2: Even polarity ($\text{pol}(Q_{RT}, Op_2) = 0$). We encode the formula $\Upsilon$ from Equation (13) using the above annotation functions adjusted as follows:

$$
\begin{align*}
\Phi_S : K(X) \times K(Y) &\rightarrow X \cup Y & \Phi_S(x_1, y_j) = \begin{cases} \neg z_k & \text{if } x_iy_j \text{ is a clause } \psi_k \text{ in } \Psi \\
\bot & \text{else} \end{cases} \\
\Phi_{RST} : K(X) \times K(Y) &\rightarrow X \cup Y & \Phi_{RST}(x_1, y_j) = \begin{cases} \neg z_k \land \neg \psi_k & \text{if } x_iy_j \text{ is a clause } \psi_k \text{ in } \Psi \\
\bot & \text{else} \end{cases}
\end{align*}
$$

The sub-queries $Q_{RT}$ and $Q_S - Q_{RT}$ are then populated as follows:

$$
\begin{align*}
\text{red}_{AB}(Q_{RT}) = K(X) \times K(Y) | \Phi_{RT} \\
\text{red}_{AB}(Q_S - Q_{RT}) = K(X) \times K(Y) | \Phi_{RST}
\end{align*}
$$
since $Op_1$ has even polarity in $Q_{RT}$. As before, the final projection $\pi_{\neg[A]-[B]}$ yields one answer tuple whose annotation is the disjunction of the annotation of $Q_S \bowtie Q_{RT}$ which is exactly $\top$, cf. Equation 13.

Pattern $P_{3.4}$. The reduction is equivalent to the case of pattern $P_{3.3}$ when the sign of the annotation functions $\Phi_T$ is flipped regardless of the polarity $\text{pol}(Q_{RT}, Op_2)$ of $Op_2$ in $Q_{RT}$.

5.2.5. Reduction for patterns $P_{3.3}$ and $P_{3.4}$. By Definition 5.7, a query $Q$ that has an annotation-preserving match with one of $P_{3.3}$ and $P_{3.4}$ if $Op_1$ is a right descendant of a difference operator, then this operator does not export $[A]$ or $[B]$.

Pattern $P_{3.3}$. Figure 13 (right) gives a schematic illustration of a query matching $P_{3.3}$. We distinguish two cases depending on whether $[B]$ is or is not exported by $Op_1$.

$[B]$ is exported by $Op_1$. Without loss of generality, assume that $Op_1$ is the first operator that allows for a match by virtue of Lemma 5.8. Then $Q_R$ contains a relation $X$ that exports $[B]$ and is joined with $R$ in $Q_R$. If this relation is $X^{[A][B]}$, then $Q$ is an annotation-preserving ($R, X, T$)-match of one of the patterns $P_{2.1}$ or $P_{3.4}$; in the latter case, $Q$ is an annotation-preserving ($R, S, X$)-match of one of the patterns $P_{1.3}$. $[B]$ is not exported by $Op_1$. The sub-query $Q_{ST}$ contains a projection operator $Op_\pi = \pi_{[B]}$ such that every operator between $Op_\pi$ and $Op_1$ exports $[A]$ but not $[B]$, and every operator between $Op_\pi$ and $Op_2$ exports $[A]$ and $[B]$. Let $Q_\pi$ be the sub-query rooted at $Op_\pi$. We first show that one may assume without loss of generality that $Op_2$ is left-deep in $Q_\pi$. Assume to the contrary that there is a difference operator $Op_-$ between $Op_\pi$ 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 $P_{3.4}$; in the latter case, $Q$ is an annotation-preserving ($R, Z, T$)-match of pattern $P_{6.4}$. In both cases, the new operator $Op_2$ is left-deep in $Q_\pi$. Within this second case, we analyse two sub-cases depending on the polarity of $Op_\pi$ in $Q_{ST}$:

Case 1: Even polarity ($\text{pol}(Q_{ST}, Op_\pi) = 0$). Let $V = X \cup Y$ and $N = \{1, \ldots, |E|\}$ be the set of indices of $\Psi$’s clauses: $\Psi = \psi_1 \vee \cdots \vee \psi_{|E|}$. We use the following annotation functions:

\[
\begin{align*}
\Phi_R : N \rightarrow V & \quad \Phi_R(i) = \top \\
\Phi_S : N \times K(V) \rightarrow V & \quad \Phi_S(i, v) = \begin{cases} \top & \text{if clause } \psi_i \text{ contains variable } v \\ \bot & \text{else} \end{cases} \\
\Phi_T : K(V) \cup K(V) \times N \rightarrow V & \quad \Phi_T(v) = \Phi_T(v, i) = \neg v \\
\Phi_{ST} : N \times K(V) \rightarrow V & \quad \Phi_{ST}(i, v) = \begin{cases} \neg v & \text{if clause } \psi_i \text{ contains variable } v \\ \bot & \text{else} \end{cases} \\
\Phi_{\pi ST} : N \rightarrow V & \quad \Phi_{\pi ST}(i) = \neg \psi_i \\
\Phi_{RST} : N \rightarrow V & \quad \Phi_{RST}(i) = \psi_i
\end{align*}
\]

That is, we set relation $R$ to contain a tuple $(n)$ annotated with $\top$ for every clause with index $n \in N$. Relation $S$ contains all tuples $(n, v)$ where $n \in N$ is a clause index and $v \in K(V)$ is the constant corresponding to the variable $v \in V$; $(n, v)$ is annotated with $\top$ if clause with index $n$ contains variable $v$, and with $\bot$ otherwise. Relation $T$ has a tuple $(v)$ annotated with $\neg v$ for each variable $v$ in $\Psi$. The annotations of relations $R, S, T$ account for their respective polarity in $Q_R, Q_S, Q_T$. The sub-query $Q_T \bowtie Q_S$ is a $Q^{AB}$-relation:

\[
\text{red}_{AB}(Q_T \bowtie Q_S) = N \times K(V)|\Phi_{ST}
\]
Fig. 14. Relations $R, S, T$ for the hardness reduction of a query that is an annotation-preserving match of pattern $P_{3,3}$ where (1) $Op_1$ does not export $[B]$ and (2) the projection operator $\pi_{-[A]}$ 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 $\Psi = \psi_1 \lor \psi_2 = x_1 y_1 \lor x_1 y_2$.

The operator $Op_\pi$ turns the $Q_{AB}$-relation $Q_T \implies Q_S$ into a $Q^A$-relation $Q_\pi$. Since $Op_\pi$ has even polarity in $Q_{ST}$, this annotation can be preserved for $Q_{ST}$:

$$\text{red}_A(Q_\pi) = N|\Phi_{\pi ST}$$

Finally, the annotations of $R$ can be preserved in $Q_R$. The sub-query $Q_R \setminus Q_{ST}$ flips the sign of the annotations of $Q_{ST}$. This yields

$$\text{red}_A(Q_R) = N|\Phi_R$$

As in the previous cases, the final projection $\pi_{-[A]}$ yields a nullary relation whose only tuple is annotated with $\neg\text{red}_1(Q_{AB})\Psi$.

Example 5.11. Figure 14 shows how $R, S, T$ are filled for the formula $\Psi = x_1 y_1 \lor x_1 y_2$ and a query matching the pattern $P_{3,3}$, and how these annotations are propagated through the query operators.

Case 2: Odd polarity ($pol(Q_{ST}, Op_\pi) = 1$). The number of difference operators between the root of the query and the relations $S$ and $T$ is even. For the annotation of the query, these operators act equivalently to a sequence of join operators: We fill the relations such that $Q_T$ is a $Q^B$-query, $Q_S$ is a $Q^{AB}$-query, $Q_R$ is a $Q^A$-query, and then $Q_{ST}$ is a $Q^A$-query, where the annotation functions for $R, S, T$ are as for the pattern $P_{1,1}$, that is:

$$\Phi_R : K(X) \cup K(X) \times K(Y) \rightarrow X \cup Y$$
$$\Phi_{ST} : K(X) \times K(Y) \rightarrow X \cup Y$$

$$\Phi_{\pi ST} : K(X) \rightarrow X \cup Y$$

$$\Phi_{RST} : K(X) \rightarrow X \cup Y$$
and obtain the following reductions
\[
\begin{align*}
\text{red}_A(Q_R) &= \mathbf{K}(X)|\Phi_R \\
\text{red}_{AB}(Q_S) &= \mathbf{K}(X) \times \mathbf{K}(Y)|\Phi_S \\
\text{red}_{B}(Q_T) &= \mathbf{K}(Y)|\Phi_T
\end{align*}
\]
where the sign of the annotations of \(\text{red}_A(Q_S)\) and \(\text{red}_{AB}(Q_S)\) is flipped because \(Op_x\) has odd polarity in \(Q_{ST}\). This yields
\[
\text{red}_A(Q_{RST}) = \mathbf{K}(X)|\Phi_{RST}
\]
for the sub-query rooted at \(Op_1 = -\) and the annotation \(\neg^{po1(Q,Op_1)}\Phi\) for the query \(Q\).

Pattern \(P_{b.4}\). The analysis of the pattern \(P_{b.4}\) is analogous to the case of the pattern \(P_{b.3}\), where the sign of the annotation function \(\Phi_S\) is flipped.

5.2.6. Reduction for the pattern \(P_{b.4}\). The analysis of pattern \(P_{b.4}\) is analogous to the case of pattern \(P_{b.3}\), where the sign of the annotation function \(\Phi_T\) is flipped.

6. THE TRACTABILITY FRONTIER FOR QUANTIFIED QUERIES

This section investigates the data complexity of the probabilistic query evaluation problem for quantified queries that express binary relationships amongst sets of entities: set division, set inclusion, set equality, set difference, and set incomparability. For example, a set-inclusion query could find non-critical overseas suppliers, that is, overseas suppliers for parts that also have domestic suppliers. A set-division query could find all suppliers for a given set of items. These queries can be expressed in relational algebra using nested negation and repeated relation symbols, as shown in Figs. 15 and 5. We analyse the data complexity of their exact computation on tuple-independent databases: For tractable queries, we give an explicit \(O(|D|)\)-algorithm for computing the probability of the query annotation based on Shannon expansion and the inclusion-exclusion principle; for intractable queries, we give a hardness reduction from \(\#2\text{DNF}\).

Although we only discuss a handful of quantified queries, each of them can in fact represent an entire class by taking as input relations independent hierarchical IRA\(^-\) queries, such that for each such query \(Q\) all of its exported attributes are root in \(Q\) (that is, for a root attribute \(A\), each relation in \(Q\) has an attribute in the class \(\{A\}\)). This holds since the result of \(Q\) on a tuple-independent database is again a tuple-independent database.

6.1. Tractable quantified queries

Assume we are given a tuple-independent relation \(S\) with schema \(\text{sid, item}\) that specifies pairs of set identifiers and items in these sets. We would like to compute the pairs of set identifiers \((s_1, s_2)\) and their probabilities, such that \(s_1\) is included/strictly included in/equal to/incomparable with \(s_2\). The corresponding queries are denoted by \(S_{\leq}, S_{\leq_2}\), and \(S_{\leq_2}\), respectively, and defined in Fig. 15 the queries \(S_{\leq_2}\) and \(S_{\leq_2}\) are the restrictions of \(S_{\leq_2}\) and \(S_{\leq_2}\) to pairs of different set identifiers. The set-division quantified query is given in Fig. 5.

Example 6.1. Relation \(S\) from Fig. 15 defines three uncertain subsets of \(\{a, b, c\}\): \(s_1 = \{a, b, c\}\), \(s_2 = \{a, b\}\), and \(s_3 = \{a, c\}\). In the absence of uncertainty, we have that \(s_2 \subseteq s_1\) and \(s_3 \subseteq s_1\), and \(s_2\) and \(s_3\) are incomparable. Under the possible worlds semantics, however, further relationships may hold between the sets. For instance, the set \(s_1\) is included in \(s_2\) for those assignments of the random variables that satisfy the annotation associated with \((1, 2)\) in \(S_{\leq_2}\). This annotation reads as follows. If \(a\) or \(b\) are in \(s_1\), then they must also be in \(s_2\); this is expressed by the term \(\neg(x_1 \lor y_1 \lor x_2 \lor y_2)\). If \(c\) is in \(s_1\), then \((1, 2)\) may not be in \(S_{\leq_2}\); this is expressed by \(\neg x_3\). The disjunctions \(x_1 \lor x_2 \lor x_3\) and \(y_1 \lor y_2\) ensure that the two sets have at least one item and are thus recorded in \(S\). The direct application of
where $k$ ple 6.1) have the general form

The annotations associated with tuples in $S$ defined in Fig. 15. Any tuple in the answer to these queries has an annotation of size $O(1)$.

**Remark 6.2.** Since a set is equal to itself, one would expect that $(i, i)$ occurs in $S_\text{=}$. However, as shown in Fig. 15, the annotation of $(1, 1) \in S_\text{=}^*$ is $(x_1 \vee x_2 \vee x_3)$, whose probability is not always 1. This is correct due to the closed world assumption in relational databases: In the worlds in which the annotation is false, there is no set 1 and hence the set-inclusion query cannot produce pairs involving this set.

The probability that a pair of set identifiers is in the answer to any of our quantified queries can be computed efficiently.

**Theorem 6.3.** Let $S$ be a tuple-independent relation over schema $S(\text{sid}, \text{item})$ that defines sets and their items, and $S_\subseteq, S_{\delta\subseteq}, S_{\subseteq}, S_{\delta\subseteq}, S_{\subseteq}, S_{\delta\subseteq}$ be their quantified queries defined in Fig. 15. Any tuple in the answer to these queries has an annotation of size $O(|S|)$ and its probability can be computed in time $O(|S|)$.

**Proof.** We compute the probabilities of the annotations for the query $S_\subseteq$ using recurrences and Shannon expansion. The annotations associated with tuples in $S_\subseteq$ (cf. Example 6.1) have the general form

$$(x_1 \lor \ldots \lor x_m) (y_1 \lor \ldots \lor y_n) \neg (x_1 \neg y_1 \lor \ldots \lor x_k \neg y_k \lor x_{k+1} \lor \ldots \lor x_m),$$

where $k$ represents the number of items the two sets have in common, and $m - k$ is the number of items in the first set and not in the second. This is equivalent to $(x_1 \lor \ldots \lor x_k) (y_1 \lor \ldots \lor y_n) \neg (x_1 \neg y_1 \lor \ldots \lor x_k \neg y_k \lor x_{k+1} \lor \ldots \lor x_m)$, and since the variables in the last negated disjunction occur only once, we can compute the probability of this disjunction efficiently and separately from the rest. We are thus left with $(x_1 \lor \ldots \lor x_k) (y_1 \lor \ldots \lor y_n) \neg (x_1 \neg y_1 \lor \ldots \lor x_k \neg y_k \lor x_{k+1} \lor \ldots \lor x_m)$.
Using Shannon expansion, we can decompose them as follows:

\[
\begin{align*}
\Sigma^x_i &= (x_i \lor \ldots \lor x_k), \\
\Sigma^{x,y}_i &= (x_i \lor \ldots \lor x_k)\Sigma^y_i, \\
\Sigma^{x,y}_i &= (y_i \lor \ldots \lor y_n)\Sigma^{x,y}_i.
\end{align*}
\]

We then have the following for any \(i\) with \(1 \leq i < k\):

\[
\begin{align*}
\Sigma^x_i &= (x_i y_i \lor \neg x_i)\Sigma^{x,y}_{i+1}, \\
\Sigma^{x,y}_i &= x_i y_i \Sigma^{x,y}_{i+1} \lor \neg x_i \Sigma^{x,y}_{i+1}, \\
\Sigma^{x,y}_i &= x_i y_i \Sigma^{x,y}_{i+1} \lor \neg x_i[y_i \Sigma^{x,y}_{i+1} \lor \neg y_i \Sigma^{x,y}_{i+1}].
\end{align*}
\]

Each of the above three formulas has a constant number of variables and refers recursively to at most three sub-formulas where one pair of variables \((x_i, y_i)\) is removed. The recursion depth is thus bounded by the number of variables in \(S\). Given the probabilities for the referred formulas, the probability of each referring formula can be computed efficiently, since all terms in the sums are pairwise mutually exclusive. We thus have \(O(|S|)\) time complexity for probability computation of \(\Sigma^{x,y}_i\) and of annotations in \(S_{\subseteq}\). Similar recurrences can be obtained for \(S_n\) and \(S_{\supset}\) under the same variable order elimination.

We next discuss the case of relational division. In the TPC-H scenario, a useful query with division would find the most likely suppliers for all parts of a given brand, cf. Fig. 3 for a query evaluation example. Similar to set-division queries, we can use recurrence formulas to obtain a linear-time algorithm for computing the probabilities of tuples in the result of a set-division query.

**Theorem 6.4.** Let \(T = R \div S\), where \(R\) and \(S\) are any tuple-independent relations. Then, any tuple in \(T\) has an annotation of size \(O(|R| + |S|)\) and its probability can be computed in time \(O(|R| + |S|)\).

**Proof.** Let \(R(\bar{A}, \bar{B})\) and \(S(\bar{B})\) be the schemas of \(R\) and \(S\), respectively. The schema of \(T\) is thus \(T(\bar{A})\). The following analysis applies separately to each value \(\bar{a}\) in \(R\).

Let \(\{y_1, \ldots, y_n\}\) and \(\{x_1, \ldots, x_m\}\) be the variables associated with tuples in \(S\) and with tuples \((\bar{a}, \bar{b})\) in \(R\) for a value \(\bar{a}\), respectively. The annotation of \(\bar{a}\) in \(T\) has the form \((x_1 \lor \ldots \lor x_m)\neg(y_1 \neg x_1 \lor \ldots \lor y_k \neg x_k \lor y_{k+1} \lor \ldots \lor y_n)\), where \(k \leq m\) is such that \(x_1, \ldots, x_k\) are those variables associated with tuples \((\bar{a}, \bar{b})\) in \(R\) where \(\bar{b}\) is in \(S\). The term \(\neg(y_{k+1} \lor \ldots \lor y_n)\) can be factored out and its probability computed efficiently since it is a sum of independent variables that do not occur elsewhere in the annotation. We are left with \((x_1 \lor \ldots \lor x_m)\neg(y_1 \neg x_1 \lor \ldots \lor y_k \neg x_k)\). Let

\[
\begin{align*}
\Sigma^x_i &= (x_i \lor \ldots \lor x_m), \\
\Sigma^y_i &= (y_i \lor \ldots \lor y_n), \\
\Sigma^{x,y}_i &= \Sigma^x_i \Sigma^y_i.
\end{align*}
\]

Using Shannon expansion, we can decompose them as follows:

\[
\begin{align*}
\Sigma^x_i &= x_i \lor \neg x_i \Sigma^x_{i+1}, \\
\Sigma^y_i &= [x_i \lor \neg x_i y_i] \Sigma^y_{i+1}, \\
\Sigma^{x,y}_i &= x_i \Sigma^{x,y}_{i+1} \lor \neg x_i y_i \Sigma^{x,y}_{i+1}.
\end{align*}
\]

These recurrence formulas share the properties of those for set-division queries: Given the probabilities for the referred formulas, the probability of each referring formula can be computed efficiently, since all terms in the sums are pairwise mutually exclusive. Moreover, the referred formulas have at least one variable less than the referring one. \(\square\)
All recurrence formulas for our quantified queries use the same variable order for Shannon expansion: $x_1, y_1, \ldots, x_k, y_k$.

### 6.2. Intractable quantified queries

Some of the queries discussed in Section 6.1 become \#P-hard when one or more of their attributes are projected out.

**Theorem 6.5.** For any $x \subset \{s_1, s_2\}$, the data complexity of the queries $\pi_\theta(S \div I)$, $\pi_x(S_{d=})$, $\pi_x(S_{d\leq})$, $\pi_x(S_{d<})$, and $\pi_x(S_{d>})$ is \#P-hard.

**Proof.** The proof is by direct reduction from the model counting problem for 2DNF formulas. We detail the reduction for the case of $\pi_\theta(S \div I)$, cf. Fig. 5 for its definition and an example; the reductions for the remaining queries are analogous.

Let $\Phi = c_1 \lor \cdots \lor c_n$ be an input 2DNF formula with $n$ clauses. Without loss of generality, we assume that the relation $I$, which specifies set items, is unary and the relation $S$, which specifies sets and their items, is binary. We construct the relation $I$ such that for each variable $v$ in $\Phi$ there is exactly one tuple, or item, $v$ in $I$ with annotation $\neg v$. We construct the relation $S$ such that there is one distinct set $i$ for each clause $c_i$ in $\Phi$ and this set consists of the items corresponding in $I$ to the variables not in $c_i$. That is, for each clause $c_i$, $S$ consists of as many tuples as variables that are in $\Phi$ and not in $c_i$; one such tuple is a pair of the set $i$ and a constant representing a variable in $\Phi$ but not in $c_i$. All tuples in $S$ are annotated with $\top$, that is, the relation $S$ is deterministic. By construction, the annotation of the query result becomes $\Phi$.

Although Boolean relational division $\pi_\theta(S \div I)$ is hard in general, its tractability depends on the input probability distribution in case each item value in $I$ is paired with each set in $S$, as we discuss next. Assume without loss of generality there are $n$ item values $1, \ldots, n$ in $I$ annotated with distinct variables $y_1, \ldots, y_n$, and there are $m$ sets $1, \ldots, m$ in $S$, such that each set $i$ has $n + k_i$ possible tuples $(i, 1), \ldots, (i, n + k_i)$ annotated by $\neg x_{1i}^1, \ldots, x_{ni}^i$. Following the annotation pattern in Fig. 5, the annotation of the query becomes:

$$\Phi = \bigvee_{i=1}^m (\neg x_{1i}^1 \lor \cdots \lor \neg x_{ni}^i) \neg (y_1 x_{11}^1 \lor \cdots \lor y_n x_{ni}^i)$$

By negating $\Phi$ and removing redundant terms, we obtain:

$$\neg \Phi = \left[ \bigwedge_{i=1}^m \bigwedge_{j=1}^{n+k_i} x_{ji}^i \right] \lor \left[ \bigwedge_{i=1}^m y_1 x_{1i}^1 \lor \cdots \lor y_n x_{ni}^i \right]$$

By applying the inclusion-exclusion principle and simplifying, the probability of $\neg \Phi$ is then:

$$P(\neg \Phi) = P\left[ \bigwedge_{i=1}^m \bigwedge_{j=1}^{n+k_i} x_{ji}^i \right] + P\left[ \bigwedge_{i=1}^m x_{1i}^1 \lor \cdots \lor x_{ni}^i \right] - P\left[ \bigwedge_{i=1}^m \bigwedge_{j=1}^{n+k_i} x_{ji}^i \right] \cdot P\left[ y_1 \lor \cdots \lor y_n \right]$$

The first and the third terms in the sum can be computed trivially regardless of the probability distribution. The second term can be computed efficiently in case of uniform distribution:

**Proposition 6.6.** The number of models of the propositional formula

$$\bigwedge_{i=1}^m y_1 x_{1i}^1 \lor \cdots \lor y_n x_{ni}^i$$

is

$$\sum_{j=1}^n \binom{n}{j} (2^n - 2^{n-j})^m.$$
The proof exploits the combinatorial structure of the formula. The formula of Proposition 6.6 admits efficient model counting — and thus probability computation under uniform probability distribution for the variables — due to its symmetry: For any choice of \( k \) out of \( n \) variables \( y_j \) set to true, the number of satisfying assignments is the same and only depends on \( k \), \( n \), and \( m \). In case of arbitrary input probability distributions, however, the formula is no longer symmetric and setting different \( k \) variables \( y_j \) to true can lead to different probabilities. In fact, arbitrary positive bipartite 2CNF formulas can be obtained by appropriately setting variables \( x_j \) to true or false.

7. 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 cannot have an effective procedure that tells us that their union is a tractable query. Restrictions on the use of negation, e.g., guarded negation [Bárání et al. 2012], 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.

7.1. 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 \( Q = \pi_{\emptyset}[S - (R \Join S_1 \cup T \Join S_2)] \) over database schema \( (S(A,B), R(A), S_1(A,B), T(B), S_2(A,B)) \) has two hierarchical union-free sub-queries under \( \pi_{\emptyset} : \pi_{\emptyset}(S - (R \Join S_1)) \) and \( \pi_{\emptyset}(S - (T \Join S_2)) \). 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 RC3-hierarchical property of the RC3 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 3-consistent and RC-hierarchical. Checking these properties can be done efficiently in the size of the input RC3 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 RC-hierarchical 3-consistency, which incurs an exponential blowup in the size of the query.

7.2. Non-repeating RC3

There are subtle differences between non-repeating relational algebra 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 non-repeating relational algebra. Whereas the former query is tractable, the latter is \#P-hard: This means that non-repeating relational algebra cannot express both tractable and hard non-repeating RC3 queries.

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 Fig. 3 that is, \( Q = \exists A \exists B [R(A) \land S(B) \land \neg (U(A) \land V(B))] \), 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∃ query $Q$ into an RC∃ query $Q_r$ as outlined in Section 3.1: $Q_r = \exists_A [R(A) \land \neg U(A)] \land \exists_B S(B) \lor \exists_A R(A) \land \exists_B [S(B) \land \neg V(B)]$ for the above query $Q$, and then again $Q$ is tractable when $Q_r$ is RC-hierarchical and $\exists$-consistent.

8. RELATED WORK

Negation is a source of complexity already for databases with incomplete information and without probabilities [Aliteboul et al. 1991]. In probabilistic databases, the MystiQ system supports a limited class of NOT EXISTS queries [Wang et al. 2008]. A framework for the exact and approximate evaluation of full relational algebra queries (thus including negation) in probabilistic databases is part of SPROUT [Fink et al. 2011; Fink et al. 2013]. Further work looks at approximating queries with negation [Khanna et al. 2011].

The dichotomy results of this article contribute to a succession of complexity results for queries on probabilistic databases: Starting from a first example of a #P-hard query [Grädel et al. 1998], polynomial-time/#P-hard dichotomies have been established for non-repeating conjunctive queries [Dalvi and Suciu 2004] and their ranking versions [Olteanu and Wen 2012], and for unions of conjunctive queries (UCQs) [Dalvi and Suciu 2012]. Our result for 1RA− strictly generalises the dichotomy for non-repeating conjunctive queries. Whereas tractable 1RA− queries can be recognised efficiently via the hierarchical syntactic property, no such syntactic characterisation of tractable UCQs is known. Further tractability results are known for inequality joins [Olteanu and Huang 2008; Olteanu and Huang 2009; Jha and Suciu 2012], and queries with aggregates and group-by clauses [Re and Suciu 2009; Fink et al. 2012].

The closest in spirit to the proof techniques in this article are those connecting OBDDs with query tractability [Olteanu and Huang 2008; Jha and Suciu 2013] and for the UCQ dichotomy result [Dalvi and Suciu 2012]. 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. Similar to root variables in our algorithm, the existence of separator variables for UCQs 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.

Further related work, which has been developed independently of this work, is a dichotomy for a class of so-called Type-1 relational calculus queries with negation, that is, CNF formulas where each clause has at most two variables and each relational symbol is unary or binary [Gribkoff et al. 2014a]. The query languages considered in this article are incomparable with the Type-1 class. This work builds upon the UCQ dichotomy [Dalvi and Suciu 2012] and as such it does not provide a syntactic characterisation of tractable queries. The first connection between polysize OBDDs and tractable queries has been shown for hierarchical non-repeating conjunctive queries [Olteanu and Huang 2008]. For UCQ queries, the inversion-freeness property corresponds to polysize OBDDs [Jha and Suciu 2013]. Queries with inequalities have been characterised in terms of their corresponding OBDDs [Olteanu and Huang 2008; Olteanu and Huang 2009; Jha and Suciu 2012].

The problems of tractable query evaluation in probabilistic databases and of domain-lifted inference for weighted first-order model counting [den Broeck 2011] essentially coincide [Gribkoff et al. 2014b]. A common assumption in much existing work in probabilistic databases is that the probabilities of two tuples of a same relation may differ; this is referred to as the asymmetric probability case. The symmetric case, where all tuples of a relation have the same probability, is more common in lifted inference in AI. A number of complexity results have been recently shown for symmetric first-order model counting [Beame et al. 2015]. A promising direction of future research is combining the asymmetric and symmetric cases.
The vast majority of hardness reductions in the above works are from the \#P-hard model-counting problem for positive (2)DNF formulas \cite{Valiant1979,Provan1983}. The complexity class \#P was originally defined by Valiant \cite{Valiant1979}. An overview of various topics in probabilistic databases has been compiled recently \cite{Suciu2011}.

9. CONCLUSION

This paper discusses a fundamental computational aspect of query processing in probabilistic databases, namely the classification of non-repeating conjunctive queries with negation and of quantified queries into tractable (polynomial-time) and intractable (\#P-hard) ones. The existence of an efficient recognition procedure for tractable queries allows a probabilistic query engine to switch between exact evaluation for tractable queries and approximate evaluation for intractable queries. A future challenge is understanding which extensions of the considered languages, e.g., with restricted union or repeating relation symbols, would still admit an efficient characterisation of tractable queries.

Acknowledgements. The authors are indebted to Dan Suciu for insightful discussions and to the anonymous reviewers for useful feedback on earlier drafts.

REFERENCES

A. RECOGNITION ALGORITHM FOR THE HIERARCHICAL PROPERTY

The hierarchical property for 1RA− is given in Definition 1.2. The membership problem

Problem IsHierarchical: Input: 1RA− query Q
Output: “Yes” if Q is hierarchical, “No” else

is the same problem as for conjunctive queries without negation [Dalvi and Suciu 2007b], with the exception that IsHierarchical requires the computation of transitivejoinsn attributes, while they are explicitly given in the case of conjunctive queries. Since the hierarchical property can be decided in AC0 [Dalvi and Suciu 2007b] and since deciding whether two attributes are tranisitively joined is in LOGSPACE, we may expect that IsHierarchical is in LOGSPACE.

Algorithm 3 gives an alternative, explicit LOGSPACE algorithm for deciding IsHierarchical. For each pair of variables A, B, the algorithm iterates over the relation symbols in
Algorithm 3: Algorithm to decide the hierarchical property for $1\text{RA}^-$ queries.

$Q$ and indicates by three Boolean flags whenever one of the relations $R[\neg A][B]$, $S[A][B]$, or $T[\neg A][B]$ has been found. This amounts to checking whether two attributes are transitively joined in $Q$, i.e. whether $A' \in [A]$. The LOGSPACE complexity is due to the following argument. It uses a constant number of Boolean flags and a constant number of iterators over $Q$. Moreover, the transitive join condition $A' \in [A]$ can be cast as the LOGSPACE-problem [Reingold 2008] of checking whether $A$ and $A'$ are connected in the undirected graph whose vertices are the attributes of $Q$ and which has an edge between $X$ and $Y$ if and only if $Q$ contains an operator $\text{b}_\rho$ with $(X=Y) \in \rho$ or an operator $\text{neg}_\rho$ with $(X \leftrightarrow Y) \in \rho$.

**Corollary A.1** ([Dalvi and Suciu 2007b, Reingold 2008]). The decision problem IsHierarchical is in LOGSPACE. □