Dichotomies for Queries with Negation in Probabilistic Databases

ROBERT FINK and DAN OLTEANU, University of Oxford

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 among 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.

4

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. 2016. Dichotomies for queries with negation in probabilistic databases. ACM Trans. Database Syst. 41, 1, Article 4 (February 2016), 47 pages.
DOI: http://dx.doi.org/10.1145/2877203

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 artificial intelligence (AI) community, essentially coincide [Gribkoff et al. 2014b].

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

Authors' addresses: R. Fink and D. Olteanu, Department of Computer Science, University of Oxford, Parks Road, Oxford, OX1 3QD, UK.

DOI: http://dx.doi.org/10.1145/2877203

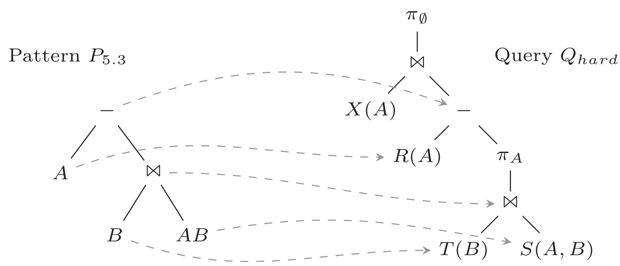


Fig. 1. Query pattern $P_{5.3}$ (left) matched by the $1\mathrm{RA}^-$ query Q_{hard} (right).

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. 2011; Fink and Olteanu 2014].

The two query classes considered in this article are fragments of relational algebra without union. The first class is denoted $1\mathrm{RA}^-$ 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 $1\mathrm{RA}^-$ queries are given in Figures 1 and 3. The second class consists of the set-division, set-inclusion, set-equivalence, and set-incomparability quantified queries. In contrast to $1\mathrm{RA}^-$ queries, quantified queries can only be expressed in relational algebra using repeating relation symbols, cf. Figures 5 and 15. The two classes can also be mixed by allowing $1\mathrm{RA}^-$ 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 Never-Ending Language Learning (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 $1\mathrm{RA}^-$ 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 $1\mathrm{RA}^-$ 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 nonhierarchical query $\pi_\emptyset[X \bowtie (R - \pi_A(T \bowtie S))]$ over schema $(X(A), R(A), T(B), S(A, B))$. The query $\pi_\emptyset(R \bowtie S \bowtie T)$ over the same schema is the classical example of a nonhierarchical query without negation [Suciu et al. 2011]. Non-Boolean versions of these queries are hierarchical. Figure 3 depicts the hierarchical query $\pi_\emptyset[R \times T - U \times V]$ 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 nonrepeating conjunctive queries that are tractable in probabilistic databases [Dalvi and Suciu 2007a]. A key contribution of 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 nonrepeating conjunctive queries [Dalvi and Suciu 2007a] to queries with negation. In Section 7, we discuss difficulties of extending this result to nonrepeating relational calculus with negation and to nonrepeating relational algebra with union.

The tractability and hardness proofs for $1RA^-$ are nontrivial 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 Nonhierarchical Queries

We prove that every nonhierarchical $1RA^-$ 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 Ψ and the query Q, for most reductions used in this article we construct an input database whose input tuples are annotated with variables in Ψ such that the result of Q becomes annotated with Ψ or $\neg\Psi$. To count the models of Ψ, we call an oracle that computes the probability P_Q of the query Q on a tuple-independent database where each variable has probability 1/2. The number of models $\#\Psi$ is then $2^n P_Q$ or $2^n(1-P_Q)$, where n is the number of variables in Ψ. 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

R

A	Φ
1	$\top$
2	$\top$

X

A	Φ
1	$\top$
2	$\top$

T

B	Φ
x$_1$	$\neg x_1$
y$_1$	$\neg y_1$
y$_2$	$\neg y_2$

S

A	B	Φ
1	x$_1$	$\top$
1	y$_1$	$\top$
2	x$_1$	$\top$
2	y$_2$	$\top$

$T \bowtie S$

A	B	Φ
1	x$_1$	$\neg x_1$
1	y$_1$	$\neg y_1$
2	x$_1$	$\neg x_1$
2	y$_2$	$\neg y_2$

$\pi_A(T \bowtie S)$

A	Φ
1	$\neg x_1 \vee \neg y_1$
2	$\neg x_1 \vee \neg y_2$

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

A	Φ
1	$x_1 y_1$
2	$x_1 y_2$

Fig. 2. Hardness reduction for query Q_{hard} in Figure 1 and formula $x_1y_1 \vee x_1y_2$. To avoid clutter (and in contrast to Section 5's naming convention), relations may share attribute names and all joins are natural.

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_{hard} in Figure 1 (right) is not hierarchical as it matches the pattern in Figure 1 (left). We exemplify the reduction for formula $\Psi = x_1y_1 \vee x_1y_2$, where we consider each variable random and with probability 1/2.

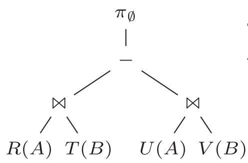
We populate the relations R, S, T, and X as shown in Figure 2. The relations R and X consist of tuples representing the indices 1 and 2 of the clauses in Ψ and annotated by the Boolean constant true ($\top$). The relation S lists all pairs of the index of a clause and variable in that clause; these tuples are also annotated by $\top$. Finally, the relation T lists all variables occurring in Ψ, where each tuple for variable z is annotated by $\neg z$. In our encoding, we may use variable names as constants, for example, 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 Φ [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 x_1, y_1, and y_2 to $\top$ defines the following database instance: R, S, and X retain all their tuples, since their annotation $\top$ is always satisfied; and T becomes empty, since the assignment falsifies the annotation of each tuple in T.

Figure 2 also depicts the intermediate results during the computation of Q_{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 Ψ.

Example 1.4 shows the power of negation: Our query Q_{hard} can compute $\#\Psi$ for any positive 2DNF formula Ψ 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 nonhierarchical 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 Ψ. 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,



R	
A	Φ
1	r_1
2	r_2

T	
B	Φ
1	t_1
2	t_2

U	
A	Φ
1	u_1
2	u_2

V	
B	Φ
1	v_1
2	v_2

$R \bowtie T$		
A	B	Φ
1	1	r_1t_1
1	2	r_1t_2
2	1	r_2t_1
2	2	r_2t_2

$R \bowtie T - U \bowtie V$		
A	B	Φ
1	1	$r_1t_1\neg(u_1v_1)$
1	2	$r_1t_2\neg(u_1v_2)$
2	1	$r_2t_1\neg(u_2v_1)$
2	2	$r_2t_2\neg(u_2v_2)$

Fig. 3. Hierarchical $1RA^-$ Boolean query Q_{easy} and a database $\mathcal{D} = (R, T, U, V)$. The tables $R \bowtie T$ and $R \bowtie T - U \bowtie V$ show how the annotations of R, T, U, V are propagated by Q_{easy}.

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 Ψ.

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 3 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 nonrepeating conjunctive queries [Olteanu and Huang 2008], they remain linear in the database size but may depend exponentially on the query size, for hierarchical $1RA^-$ 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_{easy} and the database $\mathcal{D}$ from Figure 3. The formula annotating Q_{easy}'s result is

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

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

We show in Section 3 that for every tuple-independent database $\mathcal{D}$, the annotation of the result of Q_{easy} on $\mathcal{D}$ admits an OBDD of size $\mathcal{O}(|\mathcal{D}| \cdot f(Q_{easy}))$, where $f(Q_{easy})$ is the OBDD width (cf. Section 2.3) and only depends on the query size $|Q_{easy}|$. The underlying idea is to translate Q_{easy} into an equivalent disjunction of disjunction-free existential relational calculus queries such that each of the disjuncts gives rise to a compact OBDD and all OBDDs have compatible variable orders and can be combined efficiently into a single OBDD. We denote the language of such queries by $RC^\exists$. For

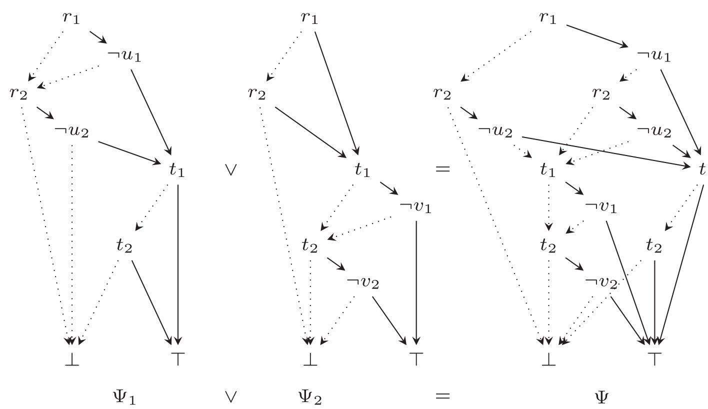


Fig. 4. From left to right: OBDDs for Ψ_1, Ψ_2, and $\Psi = \Psi_1 \vee \Psi_2$ in Example 1.5. The inner nodes of the OBDD are variables in Ψ and the leaves are the Boolean constants $\top$ and $\bot$. For an inner node n, the outgoing dotted edge is for $n = \bot$, while the outgoing solid edge is for $n = \top$. The three OBDDs share the same order of variables on any root-to-leaf path.

Q_{easy}, this translation yields the query

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

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

$$\Psi_1 = (r_1\neg u_1 \vee r_2\neg u_2) \wedge (t_1 \vee t_2) \qquad \Psi_2 = (r_1 \vee r_2) \wedge (t_1\neg v_1 \vee t_2\neg v_2),$$

and clearly $\Psi_1 \vee \Psi_2 \equiv \Psi$. 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 Ψ_1 and Ψ_2 admit OBDDs of size $\mathcal{O}(|\mathcal{D}|)$, as exemplified in the diagrams of Figure 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 among 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 Figures 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 $S(\text{sid}, \text{item})$ 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

Supplier S			Item I		Set division $S \div I$	
sid	pid	**Φ**	pid	**Φ**	sid	**Φ**
1	1	x_1	1	y_1	1	$(x_1 \vee x_2 \vee x_3 \vee x_4)\neg[(x_1 \vee x_2 \vee x_3 \vee x_4)(y_1\neg x_1 \vee y_2\neg x_2 \vee y_3\neg x_3)]$
1	2	x_2	2	y_2		$= (x_1 \vee x_2 \vee x_3 \vee x_4)(y_1 \to x_1)(y_2 \to x_2)(y_3 \to x_3)$
1	3	x_3	3	y_3	2	$(x_5 \vee x_6 \vee x_7)\neg[(x_5 \vee x_6 \vee x_7)(y_1\neg x_5 \vee y_2\neg x_6 \vee y_3)]$
1	7	x_4				$= (x_5 \vee x_6 \vee x_7)(y_1 \to x_5)(y_2 \to x_6)(\neg y_3)$
2	1	x_5				
2	2	x_6				
2	5	x_7				

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

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].

Similarly 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. Figure 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 Figure 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 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 over the input random variables given in column Φ.

We explain the annotation for sid 1 by following the structure of the query. The projection $\pi_{\text{sid}}(S)$ corresponds to the formula $x_1 \vee \cdots \vee x_4$: 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 $(x_1 \vee \cdots \vee x_4)y_i$ 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 $(x_1 \vee \cdots \vee x_4)y_i\neg x_i$. The next projection yields the disjunction of all annotations for tuples (1, 1), (1, 2), and (1, 3): $(x_1 \vee \cdots \vee x_4)[y_1\neg x_1 \vee y_2\neg x_2 \vee y_3\neg x_3]$. The outermost difference produces the annotation in Figure 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 $\pi_\emptyset(S \div I)$, is #P-hard.

Organisation of the Article

The remainder of the article is organised as follows. Section 2 recalls background on propositional formulas and probabilistic databases. Section 3 presents the polynomial-time procedure for computing probabilities of hierarchical 1RA$^-$ queries. Section 4 introduces techniques necessary in the hardness proofs for nonhierarchical queries, and Section 5 details the #P-hardness reductions. The case of quantified queries is discussed in Section 6. The article closes with a discussion of related work and open research directions in Section 7.

2. PRELIMINARIES

We introduce necessary vocabulary for the $1RA^-$ query language, the $RC^\exists$ relational calculus to which we translate $1RA^-$ queries for query evaluation, probabilistic databases, propositional formulas annotating results of queries in probabilistic databases, and their compilation into ordered binary decision diagrams as used for efficient query evaluation.

2.1. The Relational Algebra Subset $1RA^-$

We assume database schemas with unique attribute names. The set of attributes of a relation R is sch(R). A query Q is *nonrepeating* if each relation symbol occurs at most once in Q.

The relational algebra subset $1RA^-$ consists of queries composed of:

—Nonrepeating *relation symbols*;
—*Equi-join*: $Q_1 \bowtie_\rho Q_2$, where ρ is a conjunction of equality conditions $\rho = (A_1 = B_1) \wedge \cdots \wedge (A_n = B_n)$ such that all A_i are attributes of Q_1 and all B_i are attributes of Q_2;
—*Projection*: $\pi_{A_1,\ldots,A_n}$ for attributes $A_1, \ldots, A_n$. We also use $\pi_{-B_1,\ldots,-B_m}(Q) = \pi_{-[B]}(Q)$ 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 ρ is the attribute mapping $(A_1 \leftrightarrow B_1) \wedge \cdots \wedge (A_n \leftrightarrow B_n)$;
—*Selection*: $\sigma_{A\theta c}$, where A is an attribute, c a constant, and θ 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 $Q_1 \bowtie_\rho Q_2$ and $Q_1 -_\rho Q_2$, we write $A \in \rho$ to express that ρ contains an equality or mapping on A, and $(A = A') \in \rho$ or $(A \leftrightarrow A') \in \rho$ to express that ρ 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 $R(A_r)$, $S(A_s, B_s)$, $T(B_t)$ and then write the queries $R \bowtie_{A_r = A_s} S$ and $(R \bowtie T) -_{A_r \leftrightarrow A_s \wedge B_t \leftrightarrow B_s} S$ more concisely as $R \bowtie S$ and $(R \bowtie T) - S$.

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

$$\mathcal{E}(Q_1 \bowtie_\rho Q_2) = \mathcal{E}(Q_1) \cup \mathcal{E}(Q_2) \qquad \mathcal{E}(Q_1 -_\rho Q_2) = \mathcal{E}(Q_1)$$
$$\mathcal{E}(\pi_{A_1,\ldots,A_n}(Q)) = \{A_1, \ldots, A_n\} \qquad \mathcal{E}(\pi_{-[B]}(Q)) = \mathcal{E}(Q) - [B] \qquad \mathcal{E}(R) = \text{sch}(R).$$

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

2.2. The Relational Calculus Subset RC$^\exists$

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$^\exists$. 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:

$$F ::= R(X_1, \ldots, X_n) \mid \exists_X(F_1) \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid \neg(F_1),$$

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 $\mathcal{B}(\mathbf{X})$ the set of propositional formulas over variables $\mathbf{X}$. 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, $\mathbf{X}$ and $\mathbf{Y}$, a DNF formula is *bipartite* over $\mathbf{X}$ and $\mathbf{Y}$ if each clause has the form $x \wedge y$ with variables $x \in \mathbf{Y}$ and $y \in \mathbf{Y}$. The set of positive bipartite DNF formulas is denoted by 2DNF. 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 $\mathbf{X}$ of variables, we denote by $\mathcal{I}$ the set of possible assignments of all variables from $\mathbf{X}$ to constants $\top$ and $\bot$. For a formula Ψ, its set of assignments is denoted by $\mathcal{I}(\Psi) = \{I : \text{vars}(\Psi) \to \{\top, \bot\}\}$. A *model* of Ψ is a satisfying assignment, that is, an assignment I that maps Ψ to $\top$, also denoted by $I \models \Psi$. The set of models of Ψ is denoted by $\mathcal{M}(\Psi)$. 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 $\mathbf{X}$ be a set of independent random and variables. For each variable $x \in \mathbf{X}$, let P_x be the probability of x being true; we assume $P_x > 0$ without loss of generality. The probability mass function $\Pr(I) = [\prod_{x\in\mathbf{X}}^{I(x)=\top} P_x] \cdot [\prod_{x\in\mathbf{X}}^{I(x)=\bot}(1 - P_x)]$ for each assignment $I \in \mathcal{I}$ and the probability measure $\Pr(E) = \sum_{I\in E} \Pr(I)$ for all $E \subseteq \mathcal{I}$ define the probability space $(\mathcal{I}, 2^{\mathcal{I}}, \Pr)$ that we call the *probability space induced by* $\mathbf{X}$.

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

$$\Pr(\Psi = \top) = \sum_{I\in\mathcal{I}, I\models\Psi} \Pr(I). \tag{1}$$

We write P_Ψ 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, then the model counting problem reduces to the probability computation problem: $P_\Psi = 2^{-|\mathrm{vars}(\Psi)|}\#\Psi$, 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 $\mathbf{X}$ of variables is a directed acyclic graph where inner nodes are labeled with variables from $\mathbf{X}$ and terminal nodes are $\top$ (true) and $\bot$ (false). Each inner node has two outgoing edges, and for the case its variable is set to true (solid edge) and false (dotted edge), respectively. Each root-to-leaf path in a BDD is a (possibly partial) assignment of variables. A BDD is *ordered* (OBDD) if there is a total order Π on its variables such that the variables visited by each path are in Π-order. A *level* in an OBDD corresponds to all nodes labeled with the same variable. The *width*[1] of a BDD is the maximum number of edges crossing the section of the OBDD between the nodes of any two consecutive levels, where edges incident to the same node are counted as one.

In this article, we use the following results on OBDDs:

LEMMA 2.1 ([WEGENER 2004]). *Given an OBDD for a formula Ψ, the probability P_Ψ can be computed in time linear in the size of the OBDD.*

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

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 $r_1 \xrightarrow{\top} \neg u_1 \xrightarrow{\bot} r_2 \xrightarrow{\bot} \bot$ encodes that under any truth assignment ν with $\nu(r_1) = \top$ and $\nu(\neg u_1) = \nu(r_2) = \bot$, the expression $\Psi_1 = (r_1\neg u_1 \vee r_2\neg u_2) \wedge (t_1 \vee t_2)$ becomes false. The width of the left two OBDDs is three: There are three edges with different sinks crossing from level of r_2 to $\neg u_2$ and respectively from t_1 to $\neg v_1$.

The rightmost OBDD in Figure 4 represents the disjunction of the two leftmost OBDDs and has width five. Intuitively, a disjunction of two OBBDs is computed in a top-down lockstep traversal of the input OBDDs using, for example, the APPLY algorithm [Wegener 2004]. For each node of a variable x in both or one of the OBDDs, there is a node x in the output 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.

[1]A different notion of BDD width refers to the maximum number of nodes in any level.

ALGORITHM 1: Computing Annotated Result Tuples for RA Queries on pc-Tables

$$
\begin{aligned}
[\![R]\!] &= \text{R}\\
[\![\delta_{A\to B}(Q)]\!] &= \text{select } R.*,\ R.A \text{ as } B,\ R.\Phi \text{ from } ([\![Q]\!])\ R\\
[\![\sigma_\rho(Q)]\!] &= \text{select } R.* \text{ from } ([\![Q]\!])\ R \text{ where } \rho\\
[\![\pi_{A_1,\ldots,A_n}(Q)]\!] &= \text{select } R.A_1,\ldots,R.A_n,\ \bigvee(R.\Phi) \text{ as } \Phi \text{ from } ([\![Q]\!])\ \text{R group by } R.A_1,\ldots,R.A_n\\
[\![Q_1 \bowtie_\rho Q_2]\!] &= \text{select } R.*,\ S.*,\ R.\Phi \wedge S.\Phi \text{ as } \Phi \text{ from } ([\![Q_1]\!])\ R, ([\![Q_2]\!])\ S \text{ where } \rho\\
[\![Q_1 - Q_2]\!] &= \text{select } R.*,\ R.\Phi \wedge \neg S.\Phi \text{ as } \Phi \text{ from } ([\![Q_1]\!])\ R \text{ left out join } ([\![Q_2]\!])\ S \text{ on } R.* = S.*
\end{aligned}
$$

We can compute the probability of a BDD in one bottom-up pass. We exemplify for the OBDD of Ψ_1, where by $P(@x)$ we denote the probability at node x:

$$
\begin{aligned}
P(@t_2) &= P_{t_2}\cdot P_\top + P_{\neg t_2}\cdot P_\bot = P_{t_2}\\
P(@t_1) &= P_{t_1}\cdot P_\top + P_{\neg t_1}\cdot P(@t_2) = P_{t_1} + (1-P_{t_1})\cdot P(@t_2)\\
P(@\neg u_2) &= P_{\neg u_2}\cdot P(@t_1) + (1-P_{\neg u_2})\cdot P_\bot = (1-P_{u_2})\cdot P(@t_1)\\
P(@r_2) &= P_{r_2}\cdot P(@\neg u_2) + (1-P_{r_2})\cdot P_\bot = P_{r_2}\cdot P(@\neg u_2)\\
P(@\neg u_1) &= P_{\neg u_1}\cdot P(@t_1) + (1-P_{\neg u_1})\cdot P(@r_2)\\
P(@r_1) &= P_{r_1}\cdot P(@\neg u_1) + (1-P_{r_1})\cdot P(@r_2).
\end{aligned}
$$

The probability of Ψ_1 is the probability of the OBDD, which is $P(@r_1)$.

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 $\mathbf{X}$ 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 $\mathcal{D}$ represents a finite set of possible worlds: Each total assignment I of the variables in $\mathbf{X}$ defines a possible world representing a relational database consisting of exactly those tuples in $\mathcal{D}$ 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 $\mathcal{D}$, the query evaluation problem is to compute the distinct tuples in the results of Q in the worlds of $\mathcal{D}$ together with their probabilities. The probability $P(t \in Q(\mathcal{D}))$ of a tuple t is the probability that t is in the result of Q in a world randomly drawn from $\mathcal{D}$. We adopt the *intensional approach* to query evaluation [Suciu et al. 2011]: For each result tuple t, first construct a propositional formula $\Phi_{t\in Q(\mathcal{D})}$ that annotates t such that $P(t \in Q(\mathcal{D})) = P(\Phi_{t\in Q(\mathcal{D})})$, then compute $P(\Phi_{t\in Q(\mathcal{D})})$ as per Equation (1). The annotation of a Boolean query Q is denoted by $\Phi_{Q(\mathcal{D})}$; when the context is clear, we often omit the explicit reference to the database $\mathcal{D}$ and simply write Φ_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 $\mathcal{D}$. This is achieved by rewriting Q into a query Q^a such that standard relational evaluation of Q^a on $\mathcal{D}$ yields a pc-table representing the results of Q in the worlds of $\mathcal{D}$. 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 Φ; for a relation R, we consider that this column is not selected by the selector `R.*`. 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.*, S.*` 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 Φ_1 if t is in Q_1 with annotation Φ_1 and t is not in Q_2 and has annotation $\Phi_1 \wedge \neg\Phi_2$ if t is in Q_1 with annotation Φ_1 and in Q_2 with annotation Φ_2. These two cases are implemented in $[\![\cdot]\!]$ 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 Φ and how annotations are propagated through the subqueries of the depicted relational algebra query. The query result is the empty tuple annotated with the formula $\Phi = x_1y_1 \vee x_1y_2$.

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.

$\overset{\text{Lemma 3.5}}{\Rightarrow}$ Q is equivalent to a relational calculus query Q_{RC} that is RC-hierarchical (Definition 3.2) and $\exists$-consistent (Definition 3.3).

$\overset{\text{Lemma 3.8}}{\Rightarrow}$ For any tuple-independent database $\mathcal{D}$, we compute an OBDD o for the formula annotating the query result $Q_{RC}(\mathcal{D})$ in time and of size $\mathcal{O}(|\mathcal{D}| \cdot 2^{|Q_{RC}|})$.

$\overset{\text{Corollary 3.9}}{\Rightarrow}$ The probability of Φ can be computed in one bottom-up pass over the OBDD o, so in time $\mathcal{O}(|\mathcal{D}| \cdot 2^{|Q_{RC}|})$. □

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 $\neg(Q_1 \wedge Q_2) \equiv \neg Q_1 \vee \neg Q_2$. 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 RC$^\exists$ 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 $Q_1 \wedge \neg Q_2$, 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 1RA^- queries, the $\text{RC}^\exists$ rewritings may have repeating relation symbols.

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

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

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

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

Intuitively, $\exists$-consistency for an $\text{RC}^\exists$ query that is a conjunction or disjunction of subqueries means that these subqueries have compatible join orders, that is, noncontradicting $>_\exists$ 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 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 $\text{RC}^\exists$ queries:

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

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

M		N		R		T		U		V	
A	Φ	B	Φ	A	Φ	B	Φ	A	Φ	B	Φ
1	m_1	1	n_1	1	r_1	1	t_1	1	u_1	1	v_1
2	m_2	2	n_2	2	r_2	2	t_2	2	u_2	2	v_2

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

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

and, similarly to the first two OBDDs in Figure 4, can be represented by OBDDs with one node per variable and width 2 under the following variable orders:

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

ALGORITHM 2: Translation Function $[\![\cdot]\!]$ from 1RA$^-$ to RC$^\exists$.

$$[\![R]\!] = \{\{sch(R)\} \mid R(sch(R))\}$$
$$[\![\pi_{-X}(Q)]\!] = \{H_{[\![Q]\!]} \setminus \{x\} \mid \textsc{Push}^{\exists X}(F_{[\![Q]\!]})\}$$
$$[\![Q_1 \bowtie_{\wedge_i(X_i=Y_i)} Q_2]\!] = \{H_{[\![Q_1]\!]} \cup H_{[\![Q_2]\!]} \mid \textsc{Expand}(F_{[\![Q_1]\!]} \wedge F_{[\![Q_2]\!]}[\forall i : X_i/Y_i])\}$$
$$[\![Q_1 -_{\wedge_i(X_i\leftrightarrow Y_i)} Q_2]\!] = \{H_{[\![Q_1]\!]} \mid \textsc{Expand}(F_{[\![Q_1]\!]} \wedge \textsc{Push}^{\neg}(F_{[\![Q_2]\!]}[\forall i : X_i/Y_i]))\}$$

PUSH$^{\exists X}$ (Query Q)
- **if** *X does not occur in Q* **then**
  - **return** Q
- **if** $Q = \exists_Y(Q')$ **then**
  - **return** $\exists_Y(\textsc{Push}^{\exists X}(Q'))$
- **else if** $Q = \bigvee_i Q_i$ **then**
  - **return** $\bigvee_i \textsc{Push}^{\exists X}(Q_i)$
- **else if** $Q = \neg Q'$ *such that X is a root variable in Q'* **then**
  - **return** $\exists_X(\neg Q')$
- **else if** $Q = (\bigwedge_i Q_i) \wedge (\bigwedge_j Q'_j)$ *such that Xis a root variable in all Q_i and X does not appear in any Q'_j* **then**
  - ▷˜Note that the second conjunct may be empty.
  - **return** $\exists_X(\bigwedge_i Q_i) \wedge (\bigwedge_j Q'_j)$
- **else**
  - **fail**

PUSH$^{\neg}$(Query Q)
- **if** $Q = \exists_X(Q')$ **then**
  - **return** $\neg\exists_X(Q')$
- **else if** $Q = \neg Q'$ **then**
  - **return** Q'
- **else if** $Q = \bigvee_i Q_i$ **then**
  - **return** $\bigwedge_i \textsc{Push}^{\neg}(Q_i)$
- **else if** $Q = \bigwedge_i Q_i$ **then**
  - **return** $\bigvee_i \textsc{Push}^{\neg}(Q_i)$
- **else if** $Q = R(sch(R))$ **then**
  - **return** $\neg R(sch(R))$

EXPAND(Query Q)
- **if** $Q = \exists_X Q'$ *or* $Q = \neg Q'$ *or* $Q = R(sch(R))$ **then return** Q
- **else if** $Q = (\bigvee_i Q_i) \wedge (\bigvee_j Q'_j)$ **then return** $\bigvee_{i,j} \textsc{Expand}(Q_i) \wedge \textsc{Expand}(Q'_j)$
- **else if** $Q = (\bigwedge_i Q_i \vee \bigwedge_j Q'_j)$ **then return** $(\bigwedge_i \textsc{Expand}(Q_i)) \vee (\bigwedge_j \textsc{Expand}(Q'_j))$

Now consider the query $Q_{123} = Q_1 \vee Q_2 \vee Q_3$. As we show in Example 3.6 in the next section, this is obtained via translation of a hierarchical 1RA$^-$ query to relational calculus. It is RC-hierarchical and $\exists$-consistent. The variable orders Π_1, Π_2, and Π_3 are compatible in the sense that they can be extended into an order Π_{123} over all variables:

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

Following Lemma 2.1, the disjunction of the OBDDs of Φ_1, Φ_2, and Φ_3 can be represented by an OBDD of width at most 2^3 for the annotation $\Phi_1 \vee \Phi_2 \vee \Phi_3$ of query Q_{123}.

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

Our evaluation algorithm for hierarchical 1RA$^-$ queries relies on a translation of 1RA$^-$ queries into equivalent RC$^\exists$ queries. The translation function $[\![\cdot]\!]$, 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 RC$^\exists$ query as follows:

—Every $\exists$ quantifier is pushed as deep as possible in the RC$^\exists$ 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 $RC^{\exists}$ query (as per $\neg(A \wedge B) \rightarrow \neg A \vee \neg B$ 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 $RC^{\exists}$ expressions Q_1 and Q_2,

$$Q_1 \vee Q_1 \wedge Q_2 \rightarrow Q_1 \text{ and } \neg\exists_X(Q_1) \wedge \neg\exists_X(Q_1 \wedge Q_2) \rightarrow \neg\exists_X(Q_1).$$

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 1RA$^-$ query Q_{RA}, the $RC^{\exists}$ rewriting $Q_{RC} = [\![Q_{RA}]\!]$ satisfies the following properties:*

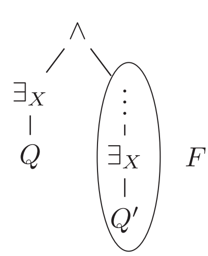
(*a*) *Q_{RC} is equivalent to Q_{RA}.*
(*b*) *Q_{RC} is canonicalised.*
(*c*) *Q_{RC} is a disjunction of disjunction-free $RC^{\exists}$ queries.*
(*d*) *For every variable X occurring in Q_{RC}, Q_{RC} has no subquery of the form $\exists_X(Q) \wedge F(\exists_X)$, where F is an expression that contains the quantifier $\exists_X$.*
(*e*) *Q_{RC} is 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 PUSH$^{\exists_X}$ 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 (PUSH$^{\neg}$) and expanding conjunctions of disjunctions (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 $E = \exists_X(Q) \wedge F(\exists_X)$. Then the parse tree of Q_{RC} contains a subtree of the following form:



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

Let us consider the possible transformation sequences that could have led to E. Since Q_{RA} is nonrepeating and since Q_{RC} is canonicalised, the two $\exists_X$ quantifiers originate from one $\exists_X$ quantifier that has been applied to the entire subquery 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 $\textsc{Push}^{\exists_X}$. Since $\textsc{Push}^{\exists_X}$ does not distribute $\exists_X$ over a conjunction, the conjunction operator on top of this subquery 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 (e). We show this property by induction on the different cases in the translation function $[\![\cdot]\!]$ in Algorithm 2.

Base case: $Q_{RA} = R$. Since $[\![R]\!] = \{\{sch(R)\} \mid R(sch(R))\}$ does not contain existential quantifiers, $[\![R]\!]$ is vacuously RC-hierarchical.

Hypothesis: The $\mathrm{RC}^{\exists}$ expressions obtained by translating subqueries 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 nonroot $\exists_X$ quantifier is the case for $[\![\pi_{-X}(Q_{RA})]\!]$. By the induction hypothesis, let $Q_{RC} = [\![Q_{RA}]\!]$ be the RC-hierarchical $\mathrm{RC}^{\exists}$ 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 $\mathrm{RC}^{\exists}$ queries, and it does not contain an expression of the form $\exists_X(Q) \wedge F(\exists_X)$. We show that if $\textsc{Push}^{\exists_X}(Q_{RC})$ fails, then Q_{RA} is nonhierarchical. Conversely, for any hierarchical query Q_{RA}, $\textsc{Push}^{\exists_X}$ always succeeds in pushing $\exists_X$ so 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 $\bigwedge_i q_i$ 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 $\exists_Y$: $\exists_Y(\textsc{Push}^{\exists_X}(q))$. We next analyse $\textsc{Push}^{\exists_X}(q)$.
  - (ii) $q_i = R$ or $q_i = \neg R$. Then $\textsc{Push}^{\exists_X}$ returns $\exists_X R$ 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 $[\![\cdot]\!]$ does not commute $\neg$ and $\exists$, Q_{RA} must contain a subquery 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 nonhierarchical.

(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 $Q_{RC} = \left(\bigwedge_i Q_i\right) \wedge \left(\bigwedge_j Q'_j\right)$ applies in $\textsc{Push}^{\exists_X}$ 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_l, 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 (1iii). In the former case, if q_l 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 (b). The relations R, S, T thus render Q_{RA} nonhierarchical. On the other hand, q_l 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 (b). Then the relation in q_l in which X occurs, together with S and T, render Q_{RA} nonhierarchical.

Property (f). Assume Q_{RC} contains two subqueries $Q_a = \exists_X Q(\exists_Y)$ 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 subqueries 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) $\exists_X[Q(X) \wedge Q'(\exists_Y)]$. We consider the four combinations of these cases for the subqueries 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 canonicalised and since the order of $\exists$ and $\neg$ is never swapped by the function $[\![\cdot]\!]$, the structure of Q_a implies that Q_{RC} has a subquery of the form $\pi_{-X}(Q'_1 - \pi_{-Y}(Q''_1))$ and the structure of Q_b implies that Q_{RC} has a subquery of the form $\pi_{-Y}(Q'_2 - \pi_{-X}(Q''_2))$. This is a contradiction.
(2) They are of type (ii): $Q_a = \exists_X[Q_1(X) \wedge Q'_1(\exists_Y)]$ and $Q_b = \exists_Y[Q_2(Y) \wedge Q'_2(\exists_X)]$. Then Q_1 contains a relation $R^{[X][\neg Y]}$, Q_2 contains a relation $T^{[\neg X][Y]}$, and, since Q_{RC} and all its subqueries are hierarchical, Q'_1 contains a relation $S^{[X][Y]}$. Q_{RA} is thus nonhierarchical since it contains the relations $R^{[X][\neg Y]}, S^{[X][Y]}, T^{[\neg X][Y]}$. 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 $Q_b = \exists_Y[Q_2(Y) \wedge Q'_2(\exists_X)]$. Since there is no negation between $\exists_Y$ and $\exists_X$ in Q_b, there is also no difference operator between π_{-X} and π_{-Y} in Q_{RA}. Conversely, the negation between $\exists_X$ and $\exists_Y$ in Q_a requires a difference operator between π_{-X} and π_{-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 (d) disallows subqueries of the form $\exists_X(Q_1) \wedge \exists_X(Q_2)$, $\exists_X(Q_1) \wedge \neg\exists_X(Q_2)$, $\exists_X(Q_1) \wedge \neg\exists_Y(Q_2 \wedge \neg\exists_X(Q_3))$ but does not forbid $\neg\exists_X(Q_1) \wedge \neg\exists_X(Q_2)$ or $\exists_X(Q_1) \vee \exists_X(Q_2)$.

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

$$Q_a = \pi_\emptyset\big[M \times N -_{A\leftrightarrow A_1, B\leftrightarrow B_1} \big[R \times T -_{A_1\leftrightarrow A_2, B_1\leftrightarrow B_2} U \times V\big]\big]$$
$$Q_b = \pi_\emptyset\big[\pi_A(M \times N) -_{A\leftrightarrow A_1} \pi_{A_1}\big[R \times T -_{A_1\leftrightarrow A_2, B_1\leftrightarrow B_2} U \times V\big]\big].$$

Query Q_a translates to Q_{123} from Example 3.4, where subsumed subqueries are removed:

$$\begin{aligned}
[\![Q_a]\!] &= \exists_A\exists_B\big(M(A) \wedge N(B) \wedge \neg[R(A) \wedge T(B) \wedge \neg(U(A) \wedge V(B))]\big)\\
&= \exists_A\exists_B\big(M(A) \wedge N(B) \wedge \neg[R(A) \wedge T(B) \wedge (\neg U(A) \vee \neg V(B))]\big)\\
&= \exists_A\exists_B\big(M(A) \wedge N(B) \wedge \neg[R(A) \wedge T(B) \wedge \neg U(A) \vee R(A) \wedge T(B) \wedge \neg V(B)]\big)\\
&= \exists_A\exists_B\big(M(A) \wedge N(B) \wedge (\neg R(A) \vee \neg T(B) \vee U(A)) \wedge (\neg R(A) \vee \neg T(B) \vee V(B))\big)\\
&= \exists_A\exists_B\big(M(A) \wedge N(B) \wedge (\neg R(A) \vee \neg T(B) \vee U(A) \wedge V(B))\big)\\
&= \exists_A\big(M(A) \wedge \neg R(A)\big) \wedge \exists_B N(B) \vee \exists_A M(A) \wedge \exists_B\big(N(B) \wedge \neg T(B)\big)\\
&\quad \vee \exists_A\big(M(A) \wedge U(A)\big) \wedge \exists_B\big(N(B) \wedge V(B)\big) = Q_{123}.
\end{aligned}$$

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):

$$\begin{aligned}
[\![Q_b]\!] &= \exists_A\big[M(A) \wedge \exists_B(N(B)) \wedge \neg\exists_{B_1}\big(R(A) \wedge T(B_1) \wedge \neg(U(A) \wedge V(B_1))\big)\big]\\
&= \exists_A\big[M(A) \wedge \exists_B(N(B)) \wedge \neg\exists_{B_1}\big(R(A) \wedge T(B_1) \wedge \neg U(A) \vee R(A) \wedge T(B_1) \wedge \neg V(B_1)\big)\big]\\
&= \exists_A\big[M(A) \wedge \exists_B(N(B)) \wedge \big(\neg R(A) \vee \neg\exists_{B_1}(T(B_1)) \vee U(A)\big)\\
&\qquad\qquad\qquad\qquad\qquad \wedge \big(\neg R(A) \vee \neg\exists_{B_1}(T(B_1) \wedge \neg V(B_1))\big)\big]\\
&= \exists_A\big[M(A) \wedge \exists_B(N(B)) \wedge \big(\neg R(A) \vee \neg\exists_{B_1}(T(B_1)) \vee U(A) \wedge \neg\exists_{B_1}(T(B_1) \wedge \neg V(B_1))\big)\big]\\
&= \exists_A\big(M(A) \wedge \neg R(A)\big) \wedge \exists_B N(B) \quad \vee \quad \exists_A M(A) \wedge \exists_B N(B) \wedge \neg\exists_{B_1} T(B_1)\\
&\quad \vee \exists_A\big(M(A) \wedge U(A)\big) \wedge \exists_B N(B) \wedge \neg\exists_{B_1}\big(T(B_1) \wedge \neg V(B_1)\big).
\end{aligned}$$

Both $\text{RC}^{\exists}$ queries $[\![Q_a]\!]$ and $[\![Q_b]\!]$ 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 subqueries (Property (f)).

The query translation can lead to large $\text{RC}^{\exists}$ queries: A conservative upper bound on their sizes would be a nonelementary function of the size of the input 1RA^{-} 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 1RA^{-} queries where for all projections $\pi_{-X}(Q)$ that are right descendants of a difference operator, attributes in the equivalence class $[X]$ occur in all relation symbols of Q (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 Φ of the $\text{RC}^{\exists}$ query Q_{RC}, which is the rewriting of an input hierarchical 1RA^{-} query Q_{RA} as per Lemma 3.5. This OBDD has a total order Π over the Boolean variables annotating the input tuples that can be derived from the structure of Q_{RC}. Let us first exemplify the construction of this order.

Example 3.7. The Boolean hierarchical 1RA^{-} query $\pi_{\emptyset}[R \bowtie \pi_X(S - T)]$, over a probabilistic database with schema $(R(X), S(X, Y), T(X, Y))$, translates to the $\text{RC}^{\exists}$ query

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

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 $R(x) \wedge \exists_Y(S(x, Y) \wedge \neg T(x, Y))$; 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 $X = x_2$):

$$\begin{aligned}
&R(x_1), S(x_1, y_1), T(x_1, y_1), S(x_1, y_2), T(x_1, y_2), S(x_1, y_3), T(x_1, y_3) \ldots,\\
&R(x_2), S(x_2, y_1), T(x_2, y_1), S(x_2, y_2), T(x_2, y_2), S(x_2, y_3), T(x_2, y_3) \ldots.
\end{aligned}$$

The annotations can be concatenated in lexicographically ascending order of the values x_i (any order of values x_i for root variables leads to the same worst-case OBDD size): We first consider all annotations for $X = x_1$ and then all annotations with $X = x_2$, and so on. For all annotations for $X = x_1$, we first consider those for $Y = y_1$ and 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 $\mathrm{RC}^{\exists}$ query $Q_1 = \exists_A\big(R(A) \wedge \neg U(A)\big) \wedge \exists_B T(B)$ from Example 1.5 is the conjunction of two RC-hierarchical and $\exists$-consistent subqueries: $Q_A = \exists_A\big(R(A) \wedge \neg U(A)\big)$ and $Q_B = \exists_B T(B)$. Similarly, Q_1's annotation $\Psi_1 = (r_1\neg u_1 \vee r_2\neg u_2) \wedge (t_1 \vee t_2)$ is the conjunction of two formulas: $\varphi_A = r_1\neg u_1 \vee r_2\neg u_2$ annotating Q_A and $\varphi_B = t_1 \vee t_2$ 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 φ_A and φ_B and then concatenate them to obtain the overall variable order for the OBDD of $\Psi_1 = \varphi_A \wedge \varphi_B$. Since A is root, the OBDD variable order for φ_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 φ_B.

This variable order derived from the structure of the $\mathrm{RC}^{\exists}$ query rewritings leads to polysize 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 $\mathrm{RC}^{\exists}$ query Q_{RC} that satisfies the properties of Lemma 3.5, the annotation Φ of Q_{RC} on a tuple-independent database $\mathcal{D}$ can be represented by an OBDD of size $\mathcal{O}(|\mathcal{D}| \cdot 2^{|Q_{RC}|})$.*

PROOF. We prove the lemma for Boolean queries Q_{RC}; the non-Boolean case follows as per discussion in the proof of Lemma 3.1. Let the relation symbols in Q_{RC} be $R_1, \ldots, R_n$, let the query variables be $X_1, \ldots, X_m$, and let $\mathrm{ADom}(X_i)$ be the active domain of variable X_i. The annotation of tuple t of relation R_i is denoted by $R_i(t)$, for example, the annotation of tuple (a, b) in relation R_1 is $R_1(a, b)$. We assume without loss of generality that the order of the query variables $X_1, \ldots, X_m$ is such that $X_i >_{\exists} X_j \Leftrightarrow i < j$ with respect to the nesting order $>_{\exists}$ defined by the $\exists$-consistency of Q_{RC}; that is, $i < j$ allows for the quantifier nesting $\exists_{X_i} Q(\exists_{X_j})$ but not $\exists_{X_j} Q(\exists_{X_i})$. Since Q_{RC} is canonicalised and $\exists$-consistent (Lemma 3.5), we can assume without loss of generality that the query variables in each relation symbol R occur in $>_{\exists}$ order (we can always relabel the query and database schema such that the query variables occur in $>_{\exists}$-order). For example, Q_{RC} may contain $R(X_1, X_5, X_7)$ but not $R(X_7, X_1, X_5)$. Furthermore, we assume a total order over the active domain of the database such that for any $x_i \in \mathrm{ADom}(X_i)$ and $x_j \in \mathrm{ADom}(X_j)$ it holds that $x_i < x_j \Leftrightarrow i < j$; similarly, for relation names $R_1 < R_2 < \cdots < R_n$, where in addition the relation names are not part of the active domains of query variables and occur before the domain constants in this order.

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

$$R(t) <_{\Pi} R'(t') \Leftrightarrow \texttt{string}(R(t)) <_{\mathrm{lex}} \texttt{string}(R'(t')),$$

where $<_{\mathrm{lex}}$ is the lexicographic order on strings as defined by the total order of the active domain of the database and the relation names. The order Π is uniquely defined by the order of the relation symbols and the order on the active domain of $\mathcal{D}$.

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

—The base case is a Boolean variable $R(t)$ which corresponds to a trivial Π-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 $Q_{RC} = Q_1 \wedge Q_2$ or $Q_{RC} = Q_1 \vee Q_2$, then, by induction hypothesis, the annotations of Q_1 and Q_2 have Π-OBDDs of width $2^{|Q_1|}$ and $2^{|Q_2|}$, respectively. Then, by Lemma 2.1, the annotation of Q_{RC} has a Π-OBDD of width $2^{|Q_1|} \cdot 2^{|Q_2|} = 2^{|Q_1|+|Q_2|} = 2^{|Q_{RC}|}$.
—If $Q_{RC} = \neg Q$, then, by induction hypothesis, Q has a Π-OBDD of width $2^{|Q|}$. Swapping the leaf nodes $\top$ and $\bot$ in this OBDD yields the required Π-OBDD for Q_{RC}.
—If $Q_{RC} = \exists_{X_i} Q$, then for every $x_l \in \text{ADom}(X_i)$ the annotations Φ_l of queries $Q[x_l/X_i]$ 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 Φ_l has a Π-OBDD of width $2^{|Q|}$ by induction hypothesis. Let $\text{ADom}(X_i) = \{x_1, \ldots, x_h\}$ such that $x_k <_{\text{lex}} x_l$ if and only if $k < l$. The annotation Φ of Q_{RC} is the disjunction $\bigvee_{x_l \in \text{ADom}(X_i)} \Phi_l$. Since the formulas Φ_l are over disjoint sets of variables for distinct values of l, an OBDD for their disjunction is obtained by their concatenation in which the leaf node $\bot$ of the OBDD for Φ_l is replaced by the root node of the OBDD for Φ_{l+1}.
  It remains to show that this construction yields an OBDD over order Π. First, the OBDD for each annotation Φ_l is over order Π by induction hypothesis; we show that for any two annotations $R(t_k)$ in Φ_k and $R'(t_l)$ in Φ_l with $k < l$, it holds that $R'(t_k) <_\Pi R'(t_l)$; by the definition of $<_\Pi$, this is equivalent to showing $t_k R <_{\text{lex}} t_l R'$. The strings t_k and t_l are identical in the first $i-1$ places since, by construction, the occurrences of each variable X_j with $j < i$ are set to the same constant. The lexicographic order of t_k and t_l—and hence the Π order of $R(t_k)$ in Φ_k and of $R'(t_l)$ in Φ_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 $2^{|Q_{RC}|} = 2^{|Q|}$, because the OBDD concatenation leaves the width unchanged. □

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

We can now use both Lemmata 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, 3.8). *Let Q_{RC} be a $\text{RC}^\exists$ query satisfying the properties of Lemma 3.5, $\mathcal{D}$ a tuple-independent database, and φ the formula annotating the query result $Q_{RC}(\mathcal{D})$. Then the probability of φ can be computed in time $\mathcal{O}(|\mathcal{D}| \cdot 2^{|Q_{RC}|})$.*

4. DATABASE CONSTRUCTION SCHEME USED IN HARDNESS REDUCTIONS

We next present a database construction scheme that prescribes how to populate relations used in a nonhierarchical 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 nonhierarchical 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 nonhierarchical queries.

We use two finite sets of constants, $\mathbf{K}_A$ and $\mathbf{K}_B$, and a constant $\blacksquare$ distinct from those in $\mathbf{K}_A$ and $\mathbf{K}_B$. In this section, the projection operator π_A^{Φ} retains both the attribute A and the annotation column Φ; in contrast, π_A only selects the attribute A without the annotation column Φ. 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 Φ be a total function on $\mathbf{K}_A$. A query Q (and its particular case of a relation) is *A-reducible to* $(\mathbf{K}_A, \Phi)$ if the $[A]$-attributes of Q are filled with all values from $\mathbf{K}_A$, all attributes not in $[A]$ are filled with $\blacksquare$, and the annotation of a tuple identified by $a \in \mathbf{K}_A$ is $\Phi(a)$:

$$\pi_A^{\Phi}(Q) = \{(a|\Phi(a)) \mid a \in \mathbf{K}_A\} \qquad \text{for any attribute } A \in [A]$$
$$\pi_C(Q) = \{(\blacksquare)\} \qquad \text{for any attribute } C \notin [A].$$

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

$$\pi_\emptyset^{\Phi}(Q) = \{(\Phi)\}$$
$$\pi_C(Q) = \{(\blacksquare)\} \qquad \text{for any attribute } C.$$

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

$$Q^{[A]} \in \mathcal{Q}^A \quad \text{if} \quad \mathtt{red}_A(Q) = \mathbf{K}_A|\Phi \text{ or } \mathtt{red}_A(Q) = \mathbf{K}_A|\neg\Phi, \tag{2}$$
$$Q^{[A]} \in \mathcal{Q}_{\text{fill}} \quad \text{if} \quad \mathtt{red}_A(Q) = \mathbf{K}_A|\Phi_\top, \tag{3}$$
$$Q^{[\neg A]} \in \mathcal{Q}_{\text{fill}} \quad \text{if} \quad \mathtt{red}_\emptyset(Q) = \blacksquare|\Phi_\top, \tag{4}$$
$$Q \in \mathcal{Q}_\emptyset \quad \text{if} \quad Q = \emptyset. \tag{5}$$

Queries in $\mathcal{Q}^A$ are relations in which the values of $[A]$-attributes are populated with values from $\mathbf{K}_A$, and values for attributes not in $[A]$ are set to $\blacksquare$. There is a functional dependency $[A] \to \Phi$ such that every tuple is represented by its A value a and has a corresponding annotation $\Phi(a)$ or $\neg\Phi(a)$. Queries in $\mathcal{Q}_{\text{fill}}$ are like $\mathcal{Q}^A$ queries with the difference that every tuple is annotated with $\top$. Queries in $\mathcal{Q}_\emptyset$ are empty relations.

Example 4.1. Given the domain $\mathbf{K}_A = \{a_1, a_2, a_3\}$, the following relation X over the distinguished attribute A and two attributes B, C with $B, C \notin [A]$ satisfies the properties of a $\mathcal{Q}^A$ query, and relation Y is a $\mathcal{Q}_{\text{fill}}$ query.

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

A_x	B_x	C_x	Φ
a_1	$\blacksquare$	$\blacksquare$	x_1
a_2	$\blacksquare$	$\blacksquare$	x_2
a_3	$\blacksquare$	$\blacksquare$	x_3

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

A_y	B_y	C_y	Φ
a_1	$\blacksquare$	$\blacksquare$	$\top$
a_2	$\blacksquare$	$\blacksquare$	$\top$
a_3	$\blacksquare$	$\blacksquare$	$\top$

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

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

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

Figure 6 shows how $\mathcal{Q}^A$, $\mathcal{Q}_{\text{fill}}$, and $\mathcal{Q}_{\emptyset}$ queries are propagated through query operators: For query classes $\mathcal{Q}_1$ and $\mathcal{Q}_2$, the rightmost column in the table shows the query class $\mathcal{Q}_1\ Op\ \mathcal{Q}_2$ for an operator Op that is join or difference.

Example 4.2. Continuing Example 4.1, the join $X \bowtie Y$ on the corresponding A, B, C attributes of $\mathcal{Q}^A$ query X and $\mathcal{Q}_{\text{fill}}$ query Y yields the following relation:

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

A_x	A_y	B_x	B_y	C_x	C_y	Φ
a_1	a_1	■	■	■	■	x_1
a_2	a_2	■	■	■	■	x_2
a_3	a_3	■	■	■	■	x_3

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

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

A_y	B_y	C_y	Φ
x_1	■	■	$\neg \mathbf{x_1}$
x_2	■	■	$\neg \mathbf{x_2}$
x_3	■	■	$\neg \mathbf{x_3}$

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

LEMMA 4.3. *Given an attribute A, a relation X exporting A, and a query Q containing X and exporting A. If $X \in \mathcal{Q}^A$, then the relations in Q can be filled such that $Q \in \mathcal{Q}^A$.*

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

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

We populate every $\text{rels}_{\text{fill}}$ relation as a $\mathcal{Q}_{\text{fill}}$ query and every $\text{rels}_{\emptyset}$ relation as a $\mathcal{Q}_{\emptyset}$-query. The following inductive argument shows that every operator on the path in Q

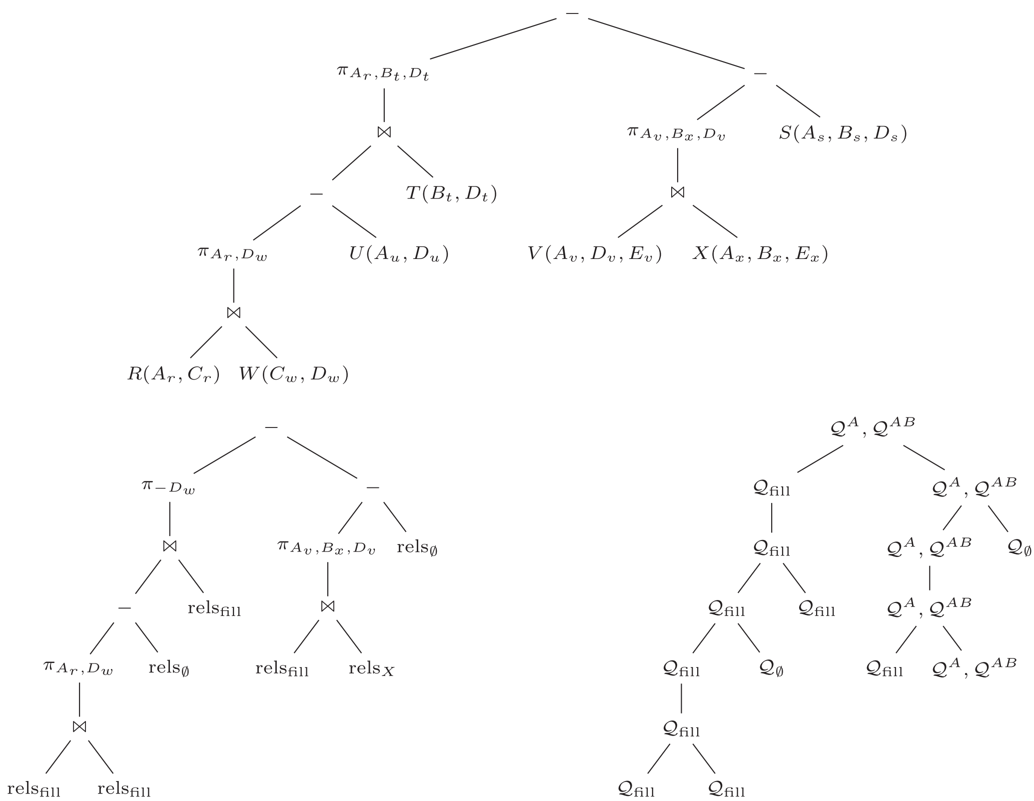


Fig. 7. A query Q (top) and the corresponding partitioning of its relations into rels_X, $\text{rels}_\emptyset$, and $\text{rels}_{\text{fill}}$ relations (bottom-left), assuming that relation X and its annotations are to be preserved by the filling. The bottom-right figure shows how the query classes of the subqueries of Q are propagated through query operators for two cases: (left) we preserve values and annotations for attributes in $[A]$, and then all subqueries rooted in operators on the path from X to the root of Q are $\mathcal{Q}^A$ queries; (right) we preserve for attributes in $[A]$ and $[B]$ and then these subqueries become $\mathcal{Q}^{AB}$-queries. Since X has odd polarity in Q, the annotations of tuples in Q are the negated annotations of the corresponding tuples in X.

between X and the root of Q is a $\mathcal{Q}^A$ query: First, this trivially holds at X itself. Now let Op be an operator on the path between X and the root of Q. We have the cases:

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

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

Example 4.4. Consider the query in Figure 7 (top). We would like to preserve the attribute A_x in relation X, where we use $\mathbf{K}_A = \{a_1, a_2, a_3\}$ as domain for $[A]$ and annotations $\Phi(a_i) = x_i$. The bottom-left graph shows the partition of Q's relations into $\text{rels}_X = \{X\}$, $\text{rels}_{\text{fill}} = \{R, W, T, V\}$, and $\text{rels}_\emptyset = \{U, S\}$. We set $U = S = \emptyset$, and for relations in $\text{rels}_{\text{fill}}$ we fill all attributes in $[A] = \{A_r, A_u, A_v\}$ with $\mathbf{K}_A$ and all other attributes with $\square$. The bottom-right graph shows how the $\mathcal{Q}^A$, $\mathcal{Q}_{\text{fill}}$, and $\mathcal{Q}_\emptyset$ subqueries are propagated through Q's operators. The children of the top difference operator are the following relations:

Left subquery of root(Q)

A_r	B_t	D_t	Φ
a_1	$\square$	$\square$	$\top$
a_2	$\square$	$\square$	$\top$
a_3	$\square$	$\square$	$\top$

Right subquery of root(Q)

A_v	B_x	D_v	Φ
a_1	$\square$	$\square$	x_1
a_2	$\square$	$\square$	x_2
a_3	$\square$	$\square$	x_3

Then the relation represented by Q is

Q

A_r	B_t	D_t	Φ
a_1	$\square$	$\square$	$\neg x_1$
a_2	$\square$	$\square$	$\neg x_2$
a_3	$\square$	$\square$	$\neg x_3$

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.

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 Φ^{AB} be a total function on $\mathbf{K}_A \times \mathbf{K}_B$, and let Φ^A be a total function on $\mathbf{K}_A \cup \mathbf{K}_A \times \mathbf{K}_B$ such that $\Phi^A(a) = \bigvee_{b \in \mathbf{K}_B} \Phi^A(a, b)$ for all $a \in \mathbf{K}_A$; $\Phi_\top$ is the constant function $\Phi_\top(.) = \top$. As before, a query (or relation) Q is *A-reducible to* $(\mathbf{K}_A, \Phi^A)$, if

$$\pi^\Phi_A(Q) = \{(a|\Phi^A(a)) \mid a \in \mathbf{K}_A\} \qquad \text{for any attribute } A \in [A]$$
$$\pi_C(Q) = \{(\square)\} \qquad \text{for any attribute } C \notin [A]$$

Similarly, Q is AB-reducible to $(\mathbf{K}_A \times \mathbf{K}_B, \Phi^{AB})$ if

$$\pi^\Phi_{AB}(Q) = \{(a, b|\Phi^{AB}(a, b)) \mid a \in \mathbf{K}_A, b \in \mathbf{K}_B\} \qquad \text{for attributes } A \in [A], B \in [B]$$
$$\pi_C(Q) = \{(\square)\} \qquad \text{for any attribute } C \notin [A] \cup [B]$$

By $\texttt{red}_{AB}(Q) = \mathbf{K}_A \times \mathbf{K}_B|\Phi^{AB}$ we denote that Q is AB-reducible to $(\mathbf{K}_A \times \mathbf{K}_B, \Phi^{AB})$.

We next define the following classes of queries:

$$Q^{[A][\neg B]} \in \mathcal{Q}^A \text{ if } \texttt{red}_A(Q) = \mathbf{K}_A|\Phi^A \text{ or } \texttt{red}_A(Q) = \mathbf{K}_A|\neg\Phi^A, \tag{6}$$
$$Q^{[A][B]} \in \mathcal{Q}^A \text{ if } \texttt{red}_{AB}(Q) = \mathbf{K}_A \times \mathbf{K}_B|\Phi^A \text{ or } \texttt{red}_{AB}(Q) = \mathbf{K}_A \times \mathbf{K}_B|\neg\Phi^A, \tag{7}$$
$$Q^{[A][B]} \in \mathcal{Q}^{AB} \text{ if } \texttt{red}_{AB}(Q) = \mathbf{K}_A \times \mathbf{K}_B|\Phi^{AB} \text{ or } \texttt{red}_{AB}(Q) = \mathbf{K}_A \times \mathbf{K}_B|\neg\Phi^{AB}, \tag{8}$$
$$Q^{[A][\neg B]} \in \mathcal{Q}_{\text{fill}} \text{ if } \texttt{red}_A(Q) = \mathbf{K}_A|\Phi_\top, \tag{9}$$
$$Q^{[A][B]} \in \mathcal{Q}_{\text{fill}} \text{ if } \texttt{red}_{AB}(Q) = \mathbf{K}_A \times \mathbf{K}_B|\Phi_\top, \tag{10}$$
$$Q^{[\neg A][\neg B]} \in \mathcal{Q}_{\text{fill}} \text{ if } \texttt{red}_\emptyset(Q) = \square|\Phi_\top, \tag{11}$$
$$Q \in \mathcal{Q}_\emptyset \text{ if } Q = \emptyset. \tag{12}$$

Queries from these classes are propagated by query operators as depicted in Figure 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 $X \in \mathcal{Q}^A$ and exports A and not B, then the relations in Q can be filled such that $Q \in \mathcal{Q}^A$ regardless whether Q exports A and not B, cf. Equation* (6)*, or exports*

X

A_x	B_x	E_x	$\mathbf{\Phi}$
a_1	b_1	■	x_{11}
a_1	b_2	■	x_{12}
a_2	b_1	■	x_{21}
a_2	b_2	■	x_{22}
a_3	b_1	■	x_{31}
a_3	b_2	■	x_{32}

Q_{RWU}

A_r	D_w	$\mathbf{\Phi}$
a_1	■	$\top$
a_2	■	$\top$
a_3	■	$\top$

Q_{RWUT}

A_r	B_t	D_t	$\mathbf{\Phi}$
a_1	b_1	■	$\top$
a_1	b_2	■	$\top$
a_2	b_1	■	$\top$
a_2	b_2	■	$\top$
a_3	b_1	■	$\top$
a_3	b_2	■	$\top$

Q_{VXS}

A_v	B_x	D_v	$\mathbf{\Phi}$
a_1	b_1	■	x_{11}
a_1	b_2	■	x_{12}
a_2	b_1	■	x_{21}
a_2	b_2	■	x_{22}
a_3	b_1	■	x_{31}
a_3	b_2	■	x_{32}

$Q = Q_{RWUT} - Q_{VXS}$

A_r	B_t	D_t	$\mathbf{\Phi}$
a_1	b_1	■	$\neg x_{11}$
a_1	b_2	■	$\neg x_{12}$
a_2	b_1	■	$\neg x_{21}$
a_2	b_2	■	$\neg x_{22}$
a_3	b_1	■	$\neg x_{31}$
a_3	b_2	■	$\neg x_{32}$

Fig. 8. Relations used in Example 4.6.

both A and B, cf. Equation (7). *If* $X \in \mathcal{Q}^{AB}$ *and Q exports A and B, then the relations in Q can be filled such that* $Q \in \mathcal{Q}^{AB}$*, 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:

$$\text{rels}_X = \{X\}$$
$$\text{rels}_\emptyset = \text{set of relations that are right descendants of an operator in } \mathcal{OP}_-$$
$$\text{rels}_{\text{fill}} = \text{all other relations}$$

We first show $Q^{[A][B]} \in \mathcal{Q}^A$ given an A-reducible relation $X \in \mathcal{Q}^A$, cf. Equation (7). The relations in Q are populated depending on their types. Every $\text{rels}_{\text{fill}}$ relation is populated as a $\mathcal{Q}_{\text{fill}}$ query: Each attribute in $[A]$ ($[B]$) with constants in $\mathbf{K}_A$ (respectively, $\mathbf{K}_B$) and the other attributes with ■. Each relation in $\text{rels}_{\text{fill}}$ that exports attributes in both $[A]$ and $[B]$ is populated such that its projection on any attribute in $[A]$ ($[B]$) is $\mathbf{K}_A$ (respectively, $\mathbf{K}_B$); its projection on any pair (A, B) of attributes with $A \in [A]$ and $B \in [B]$ is $\mathbf{K}_A \times \mathbf{K}_B$; and all attributes not in $[A]$ and $[B]$ take the value ■. Every $\text{rels}_\emptyset$ relation is a $\mathcal{Q}_\emptyset$-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 in Q between X and the root of Q is a $\mathcal{Q}^A$ query. Note, however, that the lowest $\bowtie$-operator on the path from X to the root of Q that introduces a $[B]$-attribute marks the transition from a $\mathcal{Q}^A$ subquery 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 $\mathcal{Q}^A$-relation that does not export $[B]$ and the $\mathcal{Q}_{\text{fill}}$-relation that does export $[B]$.

Finally, the case of a query $Q^{[A][B]}$ that contains a $\mathcal{Q}^{AB}$-relation $X^{[A][B]}$ follows as above and $Q \in \mathcal{Q}^{AB}$, 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 Figure 7, we would now like to preserve relation X with respect to $[A]$ and $[B]$. We take the domains $\mathbf{K}_A = \{a_1, a_2, a_3\}$ and $\mathbf{K}_B = \{b_1, b_2\}$ and the relation $X \in \mathcal{Q}^{AB}$ as depicted in Figure 8.

The sets of relations rels_X, $\text{rels}_{\text{fill}}$, and $\text{rels}_\emptyset$ are identical to those in Example 4.4 and are depicted in the bottom-left graph in Figure 7. The $\mathcal{Q}^{AB}$ and $\mathcal{Q}_{\text{fill}}$ classes through the query is depicted in the bottom-right graph. However, the relations represented by the different subqueries may now be of different types.

Let us first consider the subquery Q_{RWU} consisting of relations R, W, U. Attribute A_r is filled with $\mathbf{K}_A$, and attributes C_r, C_w, D_w are set to ■; relation U is set to $\emptyset$; all annotations are $\top$. Then $Q_{RWU} \in \mathcal{Q}_{\text{fill}}$ query by Equation (9). The relation Q_{RWU} is shown in Figure 8.

In relation T, attribute B_t is filled with $\mathbf{K}_B$ and D_t is filled with ■; 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 Figure 8. Q_{RWUT} is a $\mathcal{Q}_{\text{fill}}$ query by virtue of Equation (10).

On the right side of the topmost operator in Q, the subquery Q_{VXS} consisting of relations V, X, S is in $\mathcal{Q}^{AB}$; it is depicted in Figure 8. This leads to the $\mathcal{Q}^{AB}$-query $Q = Q_{RWUT} - Q_{VXS}$ as depicted in the figure.

5. NONHIERARCHICAL 1RA$^-$ QUERIES ARE #P-HARD

In this section we show the following result:

Lemma 5.1. *The data complexity of any nonhierarchical 1RA$^-$ query is #P-hard.*

Proof. Given a 1RA$^-$ query Q and any 2DNF formula Ψ, we use a reduction from the model-counting problem $\#\Psi$ by means of a construction of a database $\mathcal{D}$ such that Ψ and the query result $Q(\mathcal{D})$ have the same probability. The reduction depends on structural properties of Q. We show that the nonhierarchical 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 Ψ.

The proof steps are summarised as follows:

$$
\begin{array}{c}
Q \text{ is nonhierarchical} \\
\underset{\text{Proposition 5.4}}{\Leftrightarrow} \\
Q \text{ has a match with a pattern in Figure 9} \\
\underset{\text{Lemma 5.8}}{\Leftrightarrow} \\
Q \text{ has an annotation-preserving match with a pattern in Figure 9} \\
\underset{\text{Lemma 5.10}}{\Rightarrow} \\
Q \text{ is \#P-hard.} \quad \square
\end{array}
$$

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 $\text{Op}_1, \text{Op}_2 \in \{\bowtie, -\}$ is a binary tree with leaves A, B, and AB, root node Op_1, and inner node 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 ($\bowtie$) 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

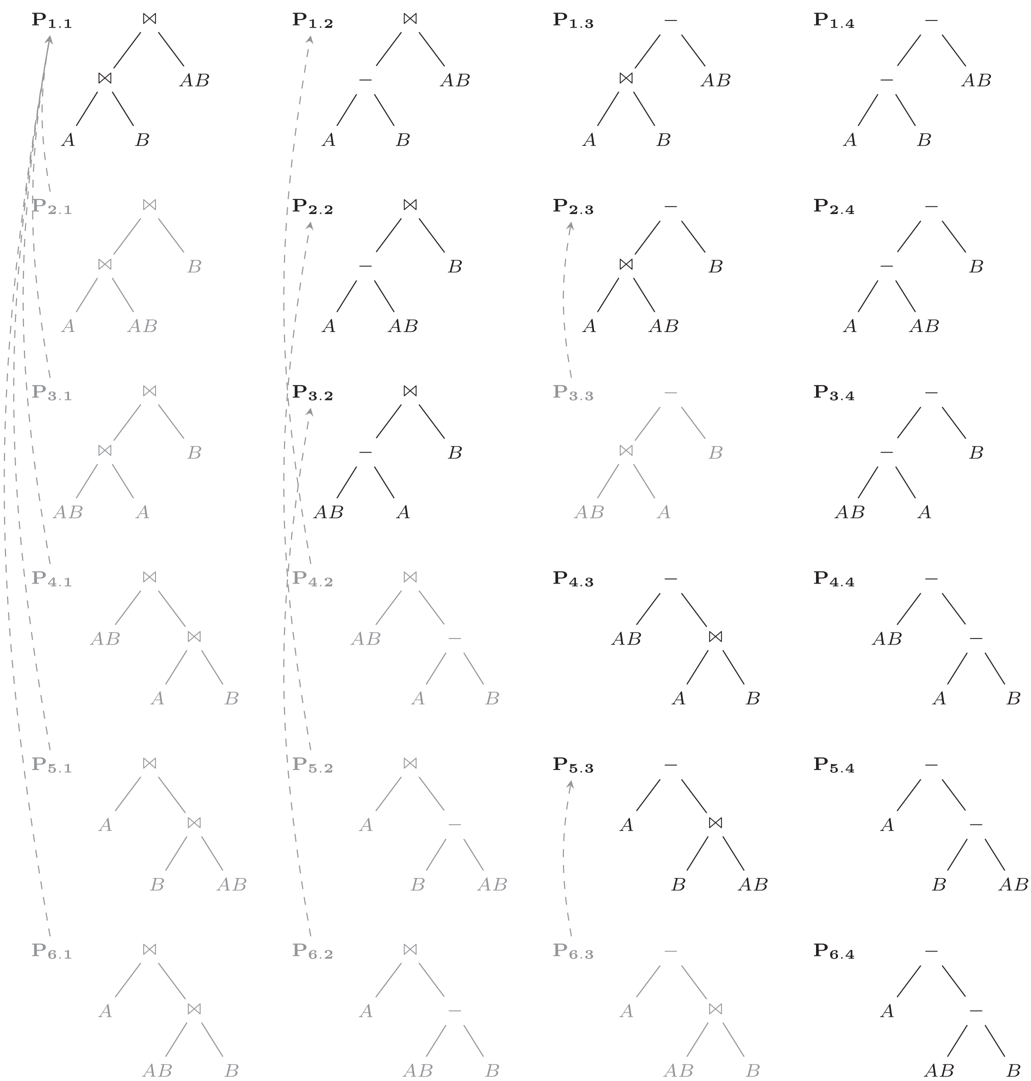


Fig. 9. The 24 query patterns $P_{1.1}, \ldots, P_{6.4}$. The 10 grey patterns can by reduced to other patterns as indicated by the arrows, since the labels A and B are symmetric and can be swapped, and the join ($\bowtie$) operator is commutative and its subqueries can also be swapped. Further 24 patterns can be obtained by swapping A and B in the above patterns.

only consider 14 patterns (those shown in dark colour and not the source of directed arrows).

Definition 5.3. A $1\mathrm{RA}^{-}$ 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: $A \mapsto R^{[A][\neg B]}$, $B \mapsto T^{[\neg A][B]}$, $AB \mapsto S^{[A][B]}$, $\mathrm{Op}_1 \mapsto \mathrm{Op}_1$, and $\mathrm{Op}_2 \mapsto \mathrm{Op}_2$. 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 nonhierarchical property:

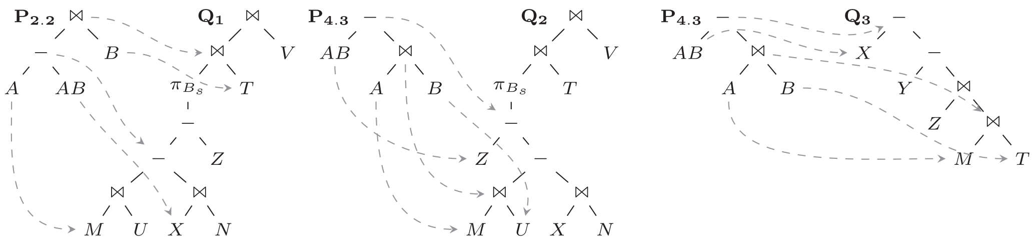


Fig. 10. Patterns $P_{2.2}$ and $P_{4.3}$ and parse trees of queries Q_1, Q_2, Q_3 over the schema $M(A_m)$, $N(A_n)$, $T(B_t, C_t)$, $U(B_u)$, $V(B_v, C_v)$, $X(A_x, B_x)$, $Y(A_y, B_y)$, $Z(A_z, B_z)$. Q_1 is an (M, X, T)-match of pattern $P_{2.2}$; it also matches other patterns and is an annotation-preserving (M, X, T)-match of $P_{2.2}$, since Op_2 (the least common ancestor of M and X) is left-deep. Although Q_2 is an (M, X, T)-match of $P_{2.2}$, it is not an annotation-preserving match of $P_{2.2}$, since Op_2 is a right descendant of the topmost difference operator. However, Q_2 is an annotation-preserving (M, Z, U)-match of pattern $P_{4.3}$. Query Q_3 is an annotation-preserving (M, X, T)-match of pattern $P_{4.3}$.

Proposition 5.4. *A 1RA$^-$ query is nonhierarchical if and only if it matches one of the patterns in Figure 9.*

Proof. $\Rightarrow$: Let Q be a nonhierarchical 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 Op_1 and Op_2 in Q. The patterns are exhaustive in the sense that there is exactly one pattern per possible combination of the operators $\bowtie$, $-$.

$\Leftarrow$: Every query Q that matches a pattern contains three distinct relation symbols R, S, T that render Q nonhierarchical 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.

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

Example 5.6. In Figure 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 Op_1 and Op_2 if: (1) Q is an (R, S, T)-match of P; (2) For every difference operator Op_- in Q, if Op_1 is a right descendant of Op_-, then Op_- does not export $[A]$ or $[B]$; (3) If Op_2 is a left descendant of Op_1 in Q, then Op_2 is left-deep in the subquery rooted at Op_1. We say that Q is an *annotation-preserving* (R, S, T)*-match* of P to emphasise the relations establishing the match.

Figure 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 subqueries rooted at $o_1, \ldots, o_n$. If Q_i is the first subquery in the above order that matches a pattern in Figure 9, then Q_i 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 Op_1 and 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_i already matches a pattern P. Without loss of generality, we assume that Q_i is an $(R^{[A][\neg B]}, S^{[A][B]}, T^{[\neg A][B]})$-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 $o_i = Op_1$ by construction.

Property 2. Proof by contradiction. Assume that there is an operator $o_j = -$ that is an ancestor of $o_i = Op_1$ such that 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 subquery 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 subquery (in the in-order sequence of subqueries) 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 subquery Q_L of Op_1. Then there is a topmost 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 subquery 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 X, S, Z 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 subquery in the in-order sequence of subqueries of Q that matches a pattern, because the subquery 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 subquery 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 subquery in the in-order sequence of subqueries of Q that matches a pattern, because the subquery 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 Figure 10. The subquery rooted at the topmost 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 Figure 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 Figure 9 is #P-hard.*

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

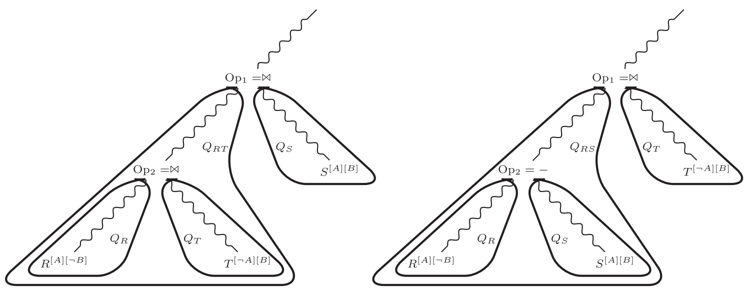


Fig. 11. Schematic illustrations of queries that are annotation-preserving matches for pattern $P_{1.1}$ (left) and $P_{2.2}$ (right). A curly edge indicates that further operators may occur on this path. By Definition 5.7, the operator Op_2 is left-deep in the left subquery Q_{RT}, or Q_{RS} respectively, of the operator Op_1, that is, it is the left descendant of any difference operator on the path between Op_1 and Op_2.

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 Figure 9, 10 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 $\Psi = \bigvee_{(i,j)\in E} x_i y_j$ be a 2DNF formula with $|E|$ clauses over disjoint variable sets $\mathbf{X}$ and $\mathbf{Y}$. We construct in polynomial time a tuple-independent database $\mathcal{D}$ using the database construction scheme in Section 4 such that the annotation of the query result $Q(\mathcal{D})$ is either Ψ and hence $P_{Q(\mathcal{D})} = P_\Psi = \#\Psi \cdot 2^{-|\text{vars}(\Psi)|}$ or $\neg\Psi$ and then $P_{Q(\mathcal{D})} = 1 - P_\Psi$.

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

By Definition 5.7, 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, subqueries Q_R, Q_S, Q_T, of Q are defined to be the left or right subqueries of Op_1 or Op_2 that contain exactly one of R, S, or T, respectively, cf. Figure 11. Additionally, if Op_2 has subqueries Q_R, Q_S, then Q_{RS} is the subquery of Op_1 that contains Q_R and Q_S. Subqueries Q_{RT} and Q_{ST} are defined similarly for matches in which R and T or S and T are descendants of Op_2, respectively.

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

Among the patterns, $P_{1.1}$ is the only one needed to show hardness of nonhierarchical 1RA^- queries without negation, that is, of nonrepeating conjunctive queries studied in prior work [Dalvi and Suciu 2007a]. Interestingly, the reduction for some patterns such as $P_{5.3}$ 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 $P_{1.1}$, $P_{1.2}$, $P_{1.3}$, and $P_{1.4}$.

Pattern $P_{1.1}$. Let Q be a query that is an annotation-preserving match of $P_{1.1}$. Figure 11 (left) depicts such a query Q, where $Q_R, Q_S, Q_T, Q_{RT}, Q_{RST}$ denote subqueries of Q. By Definition 5.7, Q contains operators $Op_1 = \bowtie$ and $Op_2 = \bowtie$ and relations $R^{[A][\neg B]}, S^{[A][B]}, T^{[\neg A][B]}$. Every operator on the path $R - Op_2 - Op_1 - S$ exports $[A]$, and every operator on the path $T - Op_2 - Op_1 - S$ exports $[B]$. Moreover, the operator Op_1 expresses a join on both $[A]$ and $[B]$.

Since Q is an annotation-preserving match, it satisfies the following additional structural properties: (1) If Op_1 is a right descendant of a difference operator, then this operator does not export $[A]$ or $[B]$. (2) Operator Op_2 is left-deep in Q_{RT}, that is, it is the left descendant of any difference operator on the path between Op_1 and Op_2.

$\mathcal{Q}^A$-relation R		
A_r	$[\neg A]$	$\mathbf{\Phi}$
$\mathtt{x}_1$	■	$\neg^{\mathtt{pol}} x_1$
$\mathtt{x}_2$	■	$\neg^{\mathtt{pol}} x_2$
...		
$\mathtt{x}_{\mid\mathbf{X}\mid}$	■	$\neg^{\mathtt{pol}} x_{\mid\mathbf{X}\mid}$

$\mathcal{Q}^B$-relation T		
B_t	$[\neg B]$	$\mathbf{\Phi}$
$\mathtt{y}_1$	■	$\neg^{\mathtt{pol}} y_1$
$\mathtt{y}_2$	■	$\neg^{\mathtt{pol}} y_2$
...		
$\mathtt{y}_{\mid\mathbf{Y}\mid}$	■	$\neg^{\mathtt{pol}} y_{\mid\mathbf{Y}\mid}$

$\mathcal{Q}^{AB}$-relation S			
A_s	B_s	$[\neg A], [\neg B]$	$\mathbf{\Phi}$
$\mathtt{x}_1$	$\mathtt{y}_1$	■	$\neg^{\mathtt{pol}} \top$
$\mathtt{x}_1$	$\mathtt{y}_2$	■	$\neg^{\mathtt{pol}} \top$
$\mathtt{x}_1$	$\mathtt{y}_3$	■	$\neg^{\mathtt{pol}} \bot$
...			
$\mathtt{x}_{\mid\mathbf{X}\mid}$	$\mathtt{y}_{\mid\mathbf{Y}\mid}$	■	$\neg^{\mathtt{pol}} \bot$

Fig. 12. Relations R, S, T for the hardness reduction of a query with an annotation-preserving match for pattern $P_{1.1}$. The filling of S assumes that the formula Ψ is $x_1y_1 \vee x_1y_2$ 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, $\neg^{\mathtt{pol}}$ stands for $\neg^{\mathtt{pol}(Q_R,R)}$, for T the polarity is $\neg^{\mathtt{pol}(Q_T,T)}$, and for S it is $\neg^{\mathtt{pol}(Q_S,S)}$.

We populate the relations R, S, T as $\mathcal{Q}^A$, $\mathcal{Q}^B$, and $\mathcal{Q}^{AB}$-relations following Equations (6) and (8). We thus fill the relation R with constants from $\mathbf{K}(\mathbf{X})$ for $[A]$-attributes and annotations Φ_R, the relation T with constants from $\mathbf{K}(\mathbf{Y})$ for $[B]$-attributes and annotations Φ_T, and the relation S with the Cartesian product of the two sets of constants for attributes $[A]$ and $[B]$, and annotations Φ_S. All other attributes are set to ■. The annotation functions are defined as follows:

$$\Phi_R : \mathbf{K}(\mathbf{X}) \cup \mathbf{K}(\mathbf{X}) \times \mathbf{K}(\mathbf{Y}) \to \mathbf{X} \cup \mathbf{Y} \qquad \Phi_R(\mathtt{x_i}) = \Phi_R(\mathtt{x_i}, \mathtt{y_j}) = x_i$$

$$\Phi_S : \mathbf{K}(\mathbf{X}) \times \mathbf{K}(\mathbf{Y}) \to \mathbf{X} \cup \mathbf{Y} \qquad \Phi_S(\mathtt{x_i}, \mathtt{y_j}) = \begin{cases} \top & \text{if } (i, j) \in E \\ \bot & \text{if } (i, j) \notin E \end{cases}$$

$$\Phi_T : \mathbf{K}(\mathbf{Y}) \cup \mathbf{K}(\mathbf{Y}) \times \mathbf{K}(\mathbf{X}) \to \mathbf{X} \cup \mathbf{Y} \qquad \Phi_T(\mathtt{y_j}) = \Phi_T(\mathtt{y_j}, \mathtt{x_i}) = y_j.$$

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

$$\mathtt{red}_A(R) = \mathbf{K}(\mathbf{X})|\neg^{\mathtt{pol}(Q_R,R)}\Phi_R$$
$$\mathtt{red}_{AB}(S) = \mathbf{K}(\mathbf{X}) \times \mathbf{K}(\mathbf{Y})|\neg^{\mathtt{pol}(Q_S,S)}\Phi_S$$
$$\mathtt{red}_B(T) = \mathbf{K}(\mathbf{Y})|\neg^{\mathtt{pol}(Q_T,T)}\Phi_T.$$

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

Figure 12 depicts the relations R, S, and T. By applying the results of Section 4 and Lemma 4.5, the remaining relations in Q_R, Q_S, and Q_T can be filled such that $Q_R \in \mathcal{Q}^A$, $Q_T \in \mathcal{Q}^B$, and $Q_S \in \mathcal{Q}^{AB}$, 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 subqueries Q_R, Q_S, Q_T, the latter relations take the following simple form[2]:

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

Let us now define the following annotations:

$$\Phi_{RT} : \mathbf{K}(\mathbf{X}) \times \mathbf{K}(\mathbf{Y}) \to \mathbf{X} \cup \mathbf{Y} \qquad \Phi_{RT}(\mathtt{x_i}, \mathtt{y_j}) = x_i y_j$$

$$\Phi_{RST} : \mathbf{K}(\mathbf{X}) \times \mathbf{K}(\mathbf{Y}) \to \mathbf{X} \cup \mathbf{Y} \qquad \Phi_{RST}(\mathtt{x_i}, \mathtt{y_j}) = \begin{cases} x_i y_j & \text{if } (i, j) \in E \\ \bot & \text{if } (i, j) \notin E \end{cases}.$$

[2]The match of Q with pattern $P_{1.1}$ does not prohibit Op_2 from expressing a join on $[B]$. In that case, the subqueries of Op_2 are $Q_R^{[A][B]}$ and $Q_T^{[A][B]}$ and satisfy $\mathtt{red}_{AB}(Q_R) = \mathbf{K}(\mathbf{X}) \times \mathbf{K}(\mathbf{Y})|\Phi_R$ and $\mathtt{red}_{AB}(Q_T) = \mathbf{K}(\mathbf{Y}) \times \mathbf{K}(\mathbf{X})|\Phi_T$ in addition to the given reductions. Both the case of Op_2 carrying a join $[B]$ and the case of Op_2 not carrying this join are covered by the definition of $\mathcal{Q}^A$ in Equations (6) and (7).

The subquery $Q_R \bowtie Q_T$ is populated as follows:

$$\texttt{red}_{AB}(Q_R \bowtie Q_T) = \mathbf{K}(\mathbf{X}) \times \mathbf{K}(\mathbf{Y})|\Phi_{RT}.$$

By applying Lemma 4.5 to Q_{RT}, the annotations in the subquery $Q_R \bowtie Q_T$ can be preserved by Q_{RT}. Since Op_2 is left-deep in Q_{RT}, Op_2 has even polarity in Q_{RT} and hence the annotations of tuples in Q_{RT} carry the same sign as in $Q_R \bowtie Q_T$. This yields

$$\texttt{red}_{AB}(Q_{RT}) = \mathbf{K}(\mathbf{X}) \times \mathbf{K}(\mathbf{Y})|\Phi_{RT} \qquad \texttt{red}_{AB}(Q_{RT} \bowtie Q_S) = \mathbf{K}(\mathbf{X}) \times \mathbf{K}(\mathbf{Y})|\Phi_{RST}.$$

The subquery $Q_{RST} = Q_{RT} \bowtie Q_S$ rooted at Op_1 thus has exactly one tuple (x_i, y_j) annotated with $x_i y_j$ for each such clause in Ψ, and one tuple (x_i, y_j) annotated with $\perp$ for each clause $x_i y_j$ not in Ψ.

Since by Definition 5.7 there does not exist a difference operator above Op_1 that exports [A] or [B], we can use the techniques of Lemma 4.5 to fill the relations representing subqueries of Q that are not descendants of Op_1 such that the annotations of Q_{RST} are preserved by any operator above Op_1. Finally, since by Definition 5.3 Q does not export [A] or [B], those attributes are eventually projected out above Op_1, yielding as a 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 $\neg^{\texttt{pol}(Q, Op_1)}\Psi$. The probability $P_{Q(\mathcal{D})}$ of query Q on the database constructed above is then P_Ψ or $1 - P_\Psi$.

Pattern $P_{1.2}$. A query that is an annotation-preserving match for $P_{1.2}$ has the same form as depicted in Figure 11 (left), except that $Op_2 = -$. The annotation functions Φ_R, Φ_S, Φ_{RT}, and Φ_{RST} are as in the case of pattern $P_{1.1}$, and the annotation function Φ_T carries an additional negation to account for the flipped polarity of T (when compared to pattern $P_{1.1}$) due to the difference operator Op_2:

$$\Phi_T : \mathbf{K}(\mathbf{Y}) \cup \mathbf{K}(\mathbf{Y}) \times \mathbf{K}(\mathbf{X}) \rightarrow \mathbf{X} \cup \mathbf{Y} \qquad \Phi_T(\mathtt{y_j}) = \Phi_T(\mathtt{y_j}, \mathtt{x_i}) = \neg y_j.$$

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

The operator 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:

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

The subquery $Q_R - Q_T$ is thus populated following $\texttt{red}_{AB}(Q_R - Q_T) = \mathbf{K}(\mathbf{X}) \times \mathbf{K}(\mathbf{Y})|\Phi_{RT}$. 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 $P_{1.2}$, except for the definition of Φ_S which carries an extra negation to account for the swapped polarity of S:

$$\Phi_S : \mathbf{K}(\mathbf{X}) \times \mathbf{K}(\mathbf{Y}) \rightarrow \mathbf{X} \cup \mathbf{Y} \qquad \Phi_S(\mathtt{x_i}, \mathtt{y_j}) = \begin{cases} \perp & \text{if } (i, j) \in E \\ \top & \text{if } (i, j) \notin E \end{cases}$$

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 Figure 11 (right) for the case of $P_{2.2}$. The query Q satisfies these structural constraints:

—Any operator on the path $R - Op_2 - S$ exports [A], and every operator on the path $S - Op_2 - Op_1 - T$ exports [B]. The operator Op_2 expresses an equality condition on [A], and the operator 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 Op_1 is a right descendant of a difference operator, then this operator does not export $[A]$ or $[B]$.
—The operator Op_2 is left-deep in Q_{RS}, that is, it is the left descendant of any difference operator on the path between Op_1 and Op_2.

Pattern $P_{2.2}$. Relations R, S, T are filled using Φ_R, Φ_S, and Φ_T exactly as in the case of pattern $P_{1.3}$. Additionally, define the following annotation functions:

$$\Phi_{RS} : \mathbf{K}(\mathbf{Y}) \cup \mathbf{K}(\mathbf{Y}) \times \mathbf{K}(\mathbf{X}) \to \mathbf{X} \cup \mathbf{Y} \qquad \Phi_{RS}(\mathtt{y_j}, \mathtt{x_i}) = \begin{cases} x_i & \text{if } (i, j) \in E \\ \bot & \text{if } (i, j) \notin E \end{cases}$$

$$\Phi_{RS}(\mathtt{y_j}) = \bigvee_{(i,j)\in E} x_i$$

$$\Phi_{RST} : \mathbf{K}(\mathbf{Y}) \cup \mathbf{K}(\mathbf{Y}) \times \mathbf{K}(\mathbf{X}) \to \mathbf{X} \cup \mathbf{Y} \qquad \Phi_{RST}(\mathtt{y_j}, \mathtt{x_i}) = \begin{cases} x_i y_j & \text{if } (i, j) \in E \\ \bot & \text{if } (i, j) \notin E \end{cases}$$

$$\Phi_{RST}(\mathtt{y_j}) = y_j \wedge \bigvee_{(i,j)\in E} x_i.$$

Then $Q_R - Q_S$ and Q_{RS} are $\mathcal{Q}^B$-queries that satisfy

$$\mathtt{red}_B(Q_R - Q_S) = \mathbf{K}(\mathbf{Y})|\Phi_{RS} \qquad \mathtt{red}_B(Q_{RS}) = \mathbf{K}(\mathbf{Y})|\Phi_{RS}$$

and Q_T is a Q^B-query with

$$\mathtt{red}_B(Q_T) = \mathbf{K}(\mathbf{Y})|\Phi_T.$$

Finally, the join $Q_{RS} \bowtie Q_T$ satisfies

$$\mathtt{red}_B(Q_{RS} \bowtie Q_T) = \mathbf{K}(\mathbf{Y})|\Phi_{RST}$$

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 $\neg^{\mathtt{pol}(Q, Op_1)}\Psi$.

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:

$$\Phi_T : \mathbf{K}(\mathbf{Y}) \cup \mathbf{K}(\mathbf{Y}) \times \mathbf{K}(\mathbf{X}) \to \mathbf{X} \cup \mathbf{Y} \qquad \Phi_T(\mathtt{y_j}) = \Phi_T(\mathtt{y_j}, \mathtt{x_i}) = \neg y_j.$$

5.2.3. Reduction for Patterns $P_{3.2}$ and $P_{3.4}$. 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 Φ_R and Φ_S are negated:

$$\Phi_R : \mathbf{K}(\mathbf{X}) \cup \mathbf{K}(\mathbf{X}) \times \mathbf{K}(\mathbf{Y}) \to \mathbf{X} \cup \mathbf{Y} \qquad \Phi_R(\mathtt{x_i}) = \Phi_R(\mathtt{x_i}, \mathtt{y_j}) = \neg x_i$$

$$\Phi_S : \mathbf{K}(\mathbf{X}) \times \mathbf{K}(\mathbf{Y}) \to \mathbf{X} \cup \mathbf{Y} \qquad \Phi_S(\mathtt{x_i}, \mathtt{y_j}) = \begin{cases} \top & \text{if } (i, j) \in E \\ \bot & \text{if } (i, j) \notin E. \end{cases}$$

Then, with Φ_{RS} as for $P_{2.2}$, subqueries $Q_S - Q_R$ and Q_{RS} are $\mathcal{Q}^B$-queries and satisfy

$$\mathtt{red}_B(Q_S - Q_R) = \mathbf{K}(\mathbf{Y})|\Phi_{RS} \qquad \mathtt{red}_B(Q_{RS}) = \mathbf{K}(\mathbf{Y})|\Phi_{RS}.$$

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 Φ_R and Φ_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 Ψ as a database $\mathcal{D}$ such that the annotation of $Q(\mathcal{D})$ is exactly Ψ, 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 $\mathcal{D}$ and another formula Υ from Ψ such that $P_{Q(\mathcal{D})} = P_\Upsilon$ and linearly many calls to an oracle for P_Υ suffice to determine $\#\Psi$.

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

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

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

$$\Upsilon = \bigvee_{i=1}^{|E|} \neg z_i \wedge \neg\psi_i \qquad \text{or, equivalently,} \qquad \neg\Upsilon = \bigwedge_{i=1}^{|E|} (z_i \vee \psi_i). \tag{13}$$

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

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

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

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

It remains to show how Υ can be encoded as the annotation of a query that is an annotation-preserving match of one of the patterns $P_{4.3}$ and $P_{4.4}$; given this encoding, any algorithm that evaluates $P_{Q(\mathcal{D})}$ 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 Figure 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.

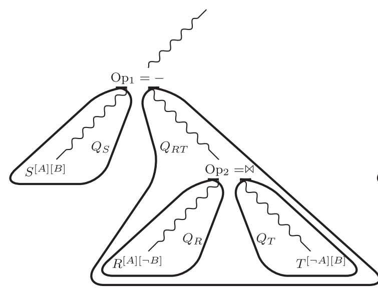


Fig. 13. Schematic illustration of a query that is an annotation-preserving match of pattern $P_{4.3}$ (left) or $P_{5.3}$ (right). A curly path indicates that other operators may occur on it.

We distinguish two cases depending on the polarity of Op_2 in the subquery Q_{RT}: If the polarity is odd, then we can use a filling similar to that of pattern $P_{1.1}$; if it is even, then we fill to obtain formula Υ as outlined above.

Case 1: Odd polarity ($\texttt{pol}(Q_{RT}, Op_2) = 1$). We fill R, S, T like in the case of pattern $P_{1.1}$ such that Q_R is a $\mathcal{Q}^A$ query, Q_T is a $\mathcal{Q}^B$-query, and Q_S is a $\mathcal{Q}^{AB}$-query, and the annotation functions are as follows:

$$\Phi_R : \mathbf{K}(\mathbf{X}) \cup \mathbf{K}(\mathbf{X}) \times \mathbf{K}(\mathbf{Y}) \to \mathbf{X} \cup \mathbf{Y} \qquad \Phi_R(\texttt{x}_\texttt{i}) = \Phi_R(\texttt{x}_\texttt{i}, \texttt{y}_\texttt{j}) = x_i$$

$$\Phi_S : \mathbf{K}(\mathbf{X}) \times \mathbf{K}(\mathbf{Y}) \to \mathbf{X} \cup \mathbf{Y} \qquad \Phi_S(\texttt{x}_\texttt{i}, \texttt{y}_\texttt{j}) = \begin{cases} \top & \text{if } (i, j) \in E \\ \bot & \text{if } (i, j) \notin E \end{cases}$$

$$\Phi_T : \mathbf{K}(\mathbf{Y}) \cup \mathbf{K}(\mathbf{Y}) \times \mathbf{K}(\mathbf{X}) \to \mathbf{X} \cup \mathbf{Y} \qquad \Phi_T(\texttt{y}_\texttt{j}) = \Phi_T(\texttt{y}_\texttt{j}, \texttt{x}_\texttt{i}) = y_j$$

$$\Phi_{RT} : \mathbf{K}(\mathbf{X}) \times \mathbf{K}(\mathbf{Y}) \to \mathbf{X} \cup \mathbf{Y} \qquad \Phi_{RT}(\texttt{x}_\texttt{i}, \texttt{y}_\texttt{j}) = x_i y_j$$

$$\Phi_{RST} : \mathbf{K}(\mathbf{X}) \times \mathbf{K}(\mathbf{Y}) \to \mathbf{X} \cup \mathbf{Y} \qquad \Phi_{RST}(\texttt{x}_\texttt{i}, \texttt{y}_\texttt{j}) = \begin{cases} x_i y_j & \text{if } (i, j) \in E \\ \bot & \text{if } (i, j) \notin E. \end{cases}$$

In other words, R consists of a tuple with A-value $\texttt{x}_\texttt{i}$ and annotation x_i for each variable $x_i \in \mathbf{X}$ that occurs in Ψ; T consists of a tuple with B-value $\texttt{y}_\texttt{j}$ and annotation y_j for each variable $y_j \in \mathbf{Y}$ that occurs in Ψ; S consists of a tuple with (A, B)-values $(\texttt{x}_\texttt{i}, \texttt{y}_\texttt{j})$ and annotation $\top$ for each clause $x_i y_j$ in Ψ. 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 differing 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 $\mathbf{K}(\mathbf{X})$ and $\mathbf{Y}$ and annotation $\top$. Any attribute that is neither in $[A]$ nor in $[B]$ is filled with the constant ■.

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 $\mathcal{Q}^{AB}$-query with annotations

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

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

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

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

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

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

$$\Phi_S : \mathbf{K}(\mathbf{X}) \times \mathbf{K}(\mathbf{Y}) \to \mathbf{X} \cup \mathbf{Y} \qquad \Phi_S(\mathtt{x_i}, \mathtt{y_j}) = \begin{cases} \neg z_k & \text{if } x_i y_j \text{ is a clause } \psi_k \text{ in } \Psi \\ \bot & \text{else} \end{cases}$$

$$\Phi_{RST} : \mathbf{K}(\mathbf{X}) \times \mathbf{K}(\mathbf{Y}) \to \mathbf{X} \cup \mathbf{Y} \quad \Phi_{RST}(\mathtt{x_i}, \mathtt{y_j}) = \begin{cases} \neg z_k \wedge \neg \psi_k & \text{if } x_i y_j \text{ is a clause } \psi_k \text{ in } \Psi \\ \bot & \text{else.} \end{cases}$$

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

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

since Op_1 has even polarity in Q_{RT}. As before, the final projection $\pi_{-[A]-[B]}$ yields one answer tuple whose annotation is the disjunction of the annotation of $Q_S - Q_{RT}$ which is exactly Υ, cf. Equation (13).

Pattern $P_{4.4}$. The reduction is equivalent to the case of pattern $P_{4.3}$ when the sign of the annotation functions Φ_T is flipped regardless of the polarity $\texttt{pol}(Q_{RT}, Op_2)$ of Op_2 in Q_{RT}.

5.2.5. Reduction for Patterns $P_{5.3}$ and $P_{5.4}$. By Definition 5.7, a query Q that has an annotation-preserving match with one of $P_{5.3}$ and $P_{5.4}$ if Op_1 is a right descendant of a difference operator; then this operator does not export $[A]$ or $[B]$.

Pattern $P_{5.3}$. Figure 13 (right) gives a schematic illustration of a query matching $P_{5.3}$. 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 $P_{2.*}$ or $P_{3.*}$; if this relation is $X^{[\neg A][B]}$, then Q is an annotation-preserving (R, S, X)-match of one of the patterns $P_{1.*}$.

$[B]$ *is not exported by* Op_1. The subquery Q_{ST} contains a projection operator $Op_\pi = \pi_{-[B]}$ such that every operator between Op_π and Op_1 exports $[A]$ but not $[B]$, and every operator between Op_π and Op_2 exports $[A]$ and $[B]$. Let Q_π be the subquery rooted at Op_π. We first show that one may assume without loss of generality that Op_2 is left-deep in Q_π. Assume to the contrary that there is a difference operator Op_- between Op_π and Op_2 that has Op_2 as a right descendant; clearly, Op_- exports $[A]$ and $[B]$ and hence its left subquery contains relations $X^{[A][\neg B]}$ and $Y^{[\neg A][B]}$ or it contains a relation $Z^{[A][B]}$. In the former case, Q is an annotation-preserving (R, S, Y)-match of pattern $P_{5.4}$; in the latter case, Q is an annotation-preserving (R, Z, T)-match of pattern $P_{6.4}$. In both cases, the new operator Op_2 is left-deep in Q_π. Within this second case, we analyse two subcases depending on the polarity of Op_π in Q_{ST}:

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

R

A_r	Φ
1	$\top$
2	$\top$

T

B_t	Φ
$\mathtt{x}_1$	$\neg x_1$
$\mathtt{y}_1$	$\neg y_1$
$\mathtt{y}_2$	$\neg y_2$

S

A_s	B_s	Φ
1	$\mathtt{x}_1$	$\top$
1	$\mathtt{y}_1$	$\top$
1	$\mathtt{y}_2$	$\bot$
2	$\mathtt{x}_1$	$\top$
2	$\mathtt{y}_1$	$\bot$
2	$\mathtt{y}_2$	$\top$

$Q_T \bowtie Q_S$

A_s	B_s	Φ
1	$\mathtt{x}_1$	$\neg x_1$
1	$\mathtt{y}_1$	$\neg y_1$
1	$\mathtt{y}_2$	$\bot$
2	$\mathtt{x}_1$	$\neg x_1$
2	$\mathtt{y}_1$	$\bot$
2	$\mathtt{y}_2$	$\neg y_2$

$Q_\pi = Q_{ST}$

A_s	Φ
1	$\neg x_1 \vee \neg y_1$
2	$\neg x_1 \vee \neg y_2$

Q_{RST}

A_r	Φ
1	$x_1 y_1$
2	$x_1 y_2$

Fig. 14. Relations R, S, T for the hardness reduction of a query that is an annotation-preserving match of pattern $P_{5.3}$ where (1) Op_1 does not export $[B]$ and (2) the projection operator $\pi_{-[B]}$ on the path between Op_1 and Op_2 has even polarity in Q_{ST} (the subquery 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 subqueries Q_R, Q_S, and Q_T. The database is with respect to the formula $\Psi = \psi_1 \vee \psi_2 = x_1 y_1 \vee x_1 y_2$.

functions:

$$\begin{aligned}
\Phi_R &: \mathbf{N} \to \mathbf{V} & \Phi_R(i) &= \top \\
\Phi_S &: \mathbf{N} \times \mathbf{K}(\mathbf{V}) \to \mathbf{V} & \Phi_S(i, \mathtt{v}) &= \begin{cases} \top & \text{if clause } \psi_i \text{ contains variable } v \\ \bot & \text{else} \end{cases} \\
\Phi_T &: \mathbf{K}(\mathbf{V}) \cup \mathbf{K}(\mathbf{V}) \times \mathbf{N} \to \mathbf{V} & \Phi_T(\mathtt{v}) &= \Phi_T(\mathtt{v}, i) = \neg v \\
\Phi_{ST} &: \mathbf{N} \times \mathbf{K}(\mathbf{V}) \to \mathbf{V} & \Phi_{ST}(i, \mathtt{v}) &= \begin{cases} \neg v & \text{if clause } \psi_i \text{ contains variable } v \\ \bot & \text{else} \end{cases} \\
\Phi_{\pi ST} &: \mathbf{N} \to \mathbf{V} & \Phi_{\pi ST}(i) &= \neg \psi_i \\
\Phi_{RST} &: \mathbf{N} \to \mathbf{V} & \Phi_{RST}(i) &= \psi_i.
\end{aligned}$$

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

$$\mathtt{red}_{AB}(Q_T \bowtie Q_S) = \mathbf{N} \times \mathbf{K}(\mathbf{V}) | \Phi_{ST}$$

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

$$\mathtt{red}_A(Q_\pi) = \mathbf{N} | \Phi_{\pi ST} \qquad \mathtt{red}_A(Q_{ST}) = \mathbf{N} | \Phi_{\pi ST}.$$

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

$$\mathtt{red}_A(Q_R) = \mathbf{N} | \Phi_R \qquad \mathtt{red}_A(Q_{RST}) = \mathbf{N} | \Phi_{RST}.$$

As in the previous cases, the final projection $\pi_{-[A]}$ yields a nullary relation whose only tuple is annotated with $\neg^{\mathtt{pol}(Q, Op_1)} \Psi$.

Example 5.11. Figure 14 shows how R, S, and T are filled for the formula $\Psi = x_1 y_1 \vee x_1 y_2$ and a query matching the pattern $P_{5.3}$ and how these annotations are propagated through the query operators.

Case 2: Odd polarity ($\mathtt{pol}(Q_{ST}, Op_\pi) = 1$). 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 $\mathcal{Q}^B$-query, Q_S is a $\mathcal{Q}^{AB}$-query, Q_R is a $\mathcal{Q}^A$ query, and then Q_{ST}

is a $\mathcal{Q}^A$ query, where the annotation functions for R, S, and T are as for the pattern $P_{1.1}$, that is:

$$\Phi_R : \mathbf{K(X)} \cup \mathbf{K(X)} \times \mathbf{K(Y)} \to \mathbf{X} \cup \mathbf{Y} \qquad \Phi_R(\mathtt{x_i}) = \Phi_R(\mathtt{x_i}, \mathtt{y_j}) = x_i$$

$$\Phi_S : \mathbf{K(X)} \times \mathbf{K(Y)} \to \mathbf{X} \cup \mathbf{Y} \qquad \Phi_S(\mathtt{x_i}, \mathtt{y_j}) = \begin{cases} \top & \text{if } (i, j) \in E \\ \bot & \text{if } (i, j) \notin E \end{cases}$$

$$\Phi_T : \mathbf{K(Y)} \cup \mathbf{K(Y)} \times \mathbf{K(X)} \to \mathbf{X} \cup \mathbf{Y} \qquad \Phi_T(\mathtt{y_i}) = \Phi_T(\mathtt{y_j}, \mathtt{x_i}) = y_i$$

$$\Phi_{ST} : \mathbf{K(X)} \times \mathbf{K(Y)} \to \mathbf{X} \cup \mathbf{Y} \qquad \Phi_{ST}(\mathtt{x_i}, \mathtt{y_j}) = \begin{cases} y_j & \text{if } (i, j) \in E \\ \bot & \text{if } (i, j) \notin E \end{cases}$$

$$\Phi_{\pi ST} : \mathbf{K(X)} \to \mathbf{X} \cup \mathbf{Y} \qquad \Phi_{\pi ST}(\mathtt{x_i}) = \bigvee_{(i,j)\in E} y_j$$

$$\Phi_{RST} : \mathbf{K(X)} \to \mathbf{X} \cup \mathbf{Y} \qquad \Phi_{RST}(\mathtt{x_i}) = x_i \wedge \bigvee_{(i,j)\in E} y_j.$$

and obtain the following reductions:

$$\mathtt{red}_A(Q_R) = \mathbf{K(X)}|\Phi_R \qquad \mathtt{red}_{AB}(Q_T \bowtie Q_S) = \mathbf{K(X)} \times \mathbf{K(Y)}|\Phi_{ST}$$
$$\mathtt{red}_{AB}(Q_S) = \mathbf{K(X)} \times \mathbf{K(Y)}|\Phi_S \qquad \mathtt{red}_A(Q_\pi) = \mathbf{K(X)}|\Phi_{\pi ST}$$
$$\mathtt{red}_B(Q_T) = \mathbf{K(Y)}|\Phi_T \qquad \mathtt{red}_A(Q_{ST}) = \mathbf{K(X)}|\neg\Phi_{\pi ST},$$

where the sign of the annotations of $\mathtt{red}_A(Q_\pi)$ and $\mathtt{red}_A(Q_{ST})$ is flipped because Op_π has odd polarity in Q_{ST}. This yields

$$\mathtt{red}_A(Q_{RST}) = \mathbf{K(X)}|\Phi_{RST}$$

for the subquery rooted at $Op_1 = -$ and the annotation $\neg^{\mathtt{pol}(Q, Op_1)}\Psi$ for the query Q.

Pattern $P_{5.4}$. The analysis of the pattern $P_{5.4}$ is analogous to the case of the pattern $P_{5.3}$, where the sign of the annotation function Φ_S is flipped.

5.2.6. Reduction for the Pattern $P_{6.4}$. The analysis of pattern $P_{6.4}$ is analogous to the case of pattern $P_{5.3}$, where the sign of the annotation function Φ_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 among sets of entities: set division, set inclusion, set equality, set difference, and set incomparability. For example, a set-inclusion query could find noncritical 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 Figures 15 and 5. We analyse the data complexity of their exact computation on tuple-independent databases: For tractable queries, we give an explicit $\mathcal{O}(|\mathcal{D}|)$-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 #2DNF.

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 1RA^- 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.

S

sid	item	Φ
1	a	x_1
1	b	x_2
1	c	x_3
2	a	y_1
2	b	y_2
3	a	z_1
3	c	z_2

$S_{\subseteq}$

s_1	s_2	Φ
1	1	$(x_1 \vee x_2 \vee x_3)$
1	2	$(x_1 \vee x_2 \vee x_3)(y_1 \vee y_2)\neg(x_1\neg y_1 \vee x_2\neg y_2 \vee x_3)$
1	3	$(x_1 \vee x_2 \vee x_3)(z_1 \vee z_2)\neg(x_1\neg z_1 \vee x_2 \vee x_3\neg z_2)$
2	1	$(y_1 \vee y_2)(x_1 \vee x_2 \vee x_3)\neg(y_1\neg x_1 \vee y_2\neg x_2)$
2	2	$(y_1 \vee y_2)$
2	3	$(y_1 \vee y_2)(z_1 \vee z_2)\neg(y_1\neg z_1 \vee y_2)$
3	1	$(z_1 \vee z_2)(x_1 \vee x_2 \vee x_3)\neg(z_1\neg x_1 \vee z_2\neg x_3)$
3	2	$(z_1 \vee z_2)(y_1 \vee y_2)\neg(z_1\neg y_1 \vee z_2)$
3	3	$(z_1 \vee z_2)$

$S_{=}$

s_1	s_2	Φ
1	1	$\Phi(s_1 \subseteq s_1)$
1	2	$\Phi(s_1 \subseteq s_2) \wedge \Phi(s_2 \subseteq s_1)$
1	3	$\Phi(s_1 \subseteq s_3) \wedge \Phi(s_3 \subseteq s_1)$
2	2	$\Phi(s_2 \subseteq s_2)$
2	3	$\Phi(s_2 \subseteq s_3) \wedge \Phi(s_3 \subseteq s_2)$
3	3	$\Phi(s_3 \subseteq s_3)$

Symmetric cases omitted

$$
\begin{aligned}
S_{\subseteq} &= \pi_{s_1}\delta_{\text{sid}\to s_1}(S) \bowtie \pi_{s_2}\delta_{\text{sid}\to s_2}(S) - \\
&\quad \pi_{s_1,s_2}\left(\delta_{\text{sid}\to s_1}(S) \bowtie \pi_{s_2}\delta_{\text{sid}\to s_2}(S) - \delta_{\text{sid}\to s_2}(S) \bowtie \pi_{s_1}\delta_{\text{sid}\to s_1}(S)\right)\\
S_{=} &= \pi_{s_1,s_2}[S_{\subseteq} \bowtie_{s_1=s_4 \wedge s_2=s_3} \delta_{s_1\to s_3, s_2\to s_4}(S_{\subseteq})]\\
S_{\neg} &= [\pi_{s_1}(\delta_{\text{sid}\to s_1}(S)) \bowtie \pi_{s_2}(\delta_{\text{sid}\to s_2}(S))] - S_{\subseteq}\\
S_{<>} &= \pi_{s_1,s_2}[S_{\neg} \bowtie_{s_1=s_4 \wedge s_2=s_3} \delta_{s_1\to s_3, s_2\to s_4}(S_{\neg})]\\
S_{d=} &= \sigma_{s_1\neq s_2}S_{=}\\
S_{d\subseteq} &= \sigma_{s_1\neq s_2}S_{\subseteq}\\
S_{\subset} &= S_{\subseteq} - S_{=}
\end{aligned}
$$

Fig. 15. Definition of queries for computing set inclusion, equality, and incomparability. The tables show an example database (S) and the result of the set inclusion ($S_{\subseteq}$) and equality ($S_{=}$) queries. In the table for $S_{=}$, $\Phi(s_i \subseteq s_j)$ is the annotation of the tuple (i, j) in relation $S_{\subseteq}$.

6.1. Tractable Quantified Queries

Assume we are given a tuple-independent relation S with schema $S(\text{sid}, \text{item})$ 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 in/strictly included in/equal to/incomparable with s_2. The corresponding queries are denoted by $S_{\subseteq}$, $S_{\subset}$, $S_{=}$, and $S_{<>}$, respectively, and defined in Figure 15; the queries $S_{d=}$ and $S_{d\subseteq}$ are the restrictions of $S_{=}$ and $S_{\subseteq}$ to pairs of different set identifiers. The set-division quantified query is given in Figure 5.

Example 6.1. Relation S from Figure 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 $s_2 \subset s_1$ and $s_3 \subset s_1$, 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 $S_{\subseteq}$. 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 $\neg(x_1\neg y_1 \vee x_2\neg y_2)$. If c is in s_1, then $(1, 2)$ may not be in $S_{\subseteq}$; this is expressed by $\neg x_3$. The disjunctions $x_1 \vee x_2 \vee x_3$ and $y_1 \vee y_2$ ensure that the two sets have at least one item and are thus recorded in S. The direct application of the translation $[\![S_{\subseteq}]\!]$ according to Algorithm 1 yields an equivalent yet syntactically slightly different annotation than that depicted in Figure 15.

Remark 6.2. Since a set is equal to itself, one would expect that (i, i) occurs in $S_{=}$ with probability 1 for all sets i from S. However, as shown in Figure 15, the annotation of $(1, 1) \in S_{=}$ 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 $S(sid, item)$ that defines sets and their items, and $S_{\subseteq}$, $S_{d\subseteq}$, $S_{\subset}$, $S_{=}$, $S_{d=}$, and $S_{<>}$ be the quantified queries defined in Figure 15. Any tuple in the answer to these queries has an annotation of size $\mathcal{O}(|S|)$ and its probability can be computed in time $\mathcal{O}(|S|)$.*

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

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

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

$$\begin{aligned}\Sigma_i^{xy} &= \neg(x_i\neg y_i \vee \ldots \vee x_k\neg y_k),\\ \Sigma_i^{x,xy} &= (x_i \vee \ldots \vee x_k)\Sigma_i^{xy},\\ \Sigma_i^{x,y,xy} &= (y_i \vee \ldots \vee y_n)\Sigma_i^{x,xy}.\end{aligned}$$

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

$$\begin{aligned}\Sigma_i^{xy} &= (x_i y_i \vee \neg x_i)\Sigma_{i+1}^{xy}\\ \Sigma_i^{x,xy} &= x_i y_i \Sigma_{i+1}^{xy} \vee \neg x_i \Sigma_{i+1}^{x,xy}\\ \Sigma_i^{x,y,xy} &= x_i y_i \Sigma_{i+1}^{xy} \vee \neg x_i [y_i \Sigma_{i+1}^{x,xy} \vee \neg y_i \Sigma_{i+1}^{x,y,xy}]\end{aligned}$$

Each of the above three formulas has a constant number of variables and refers recursively to at most three subformulas 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 $\mathcal{O}(|S|)$ time complexity for probability computation of $\Sigma_1^{x,y,xy}$ and of annotations in $S_{\subseteq}$. Similar recurrences can be obtained for $S_{=}$ and $S_{<>}$ 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. Figure 5 for a query evaluation example. Similarly to set-relation 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 $\mathcal{O}(|R| + |S|)$ and its probability can be computed in time $\mathcal{O}(|R| + |S|)$.*

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 $(x_1 \vee \ldots \vee x_m)\neg(y_1\neg x_1 \vee \ldots \vee y_k\neg x_k \vee y_{k+1} \vee \ldots \vee y_n)$, 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 $\neg(y_{k+1} \vee \ldots \vee y_n)$ 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 $(x_1 \vee \ldots \vee x_m)\neg(y_1\neg x_1 \vee \ldots \vee y_k\neg x_k)$. Let

$$\begin{aligned}\Sigma_i^x &= (x_i \vee \ldots \vee x_m)\\ \Sigma_i^{xy} &= \neg(y_i\neg x_i \vee \ldots \vee y_k\neg x_k)\\ \Sigma_i^{x,xy} &= \Sigma_i^x\,\Sigma_i^{xy}.\end{aligned}$$

Using Shannon expansion, we can decompose them as follows:

$$\begin{aligned}\Sigma_i^x &= x_i \vee \neg x_i\,\Sigma_{i+1}^x\\ \Sigma_i^{xy} &= [x_i \vee \neg x_i\neg y_i]\,\Sigma_{i+1}^{xy}\\ \Sigma_i^{x,xy} &= x_i\,\Sigma_{i+1}^{xy} \vee \neg x_i\neg y_i\,\Sigma_{i+1}^{x,xy}.\end{aligned}$$

These recurrence formulas share the properties of those for set-relation 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. □

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 $\pi_\emptyset(S \div I)$, $\pi_x(S_{d=})$, $\pi_x(S_{d\subseteq})$, $\pi_x(S_\subset)$, and $\pi_x(S_{<>})$ 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 $\pi_\emptyset(S \div I)$, cf. Figure 5 for its definition and an example; the reductions for the remaining queries are analogous.

Let $\Phi = c_1 \vee \cdots \vee c_n$ 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 Φ 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 Φ 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 Φ and not in c_i; one such tuple is a pair of the set i and a constant representing a variable in Φ 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 Φ. □

Although Boolean relational division $\pi_\emptyset(S \div I)$ 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 that 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 $\neg x_1^i, \ldots, x_{n+k_i}^i$. Following the annotation pattern in Figure 5, the annotation of the query becomes

$$\Phi = \bigvee_{i=1}^{m} \left(\neg x_1^i \vee \ldots \vee \neg x_{n+k_i}^i\right)\neg\left(y_1 x_1^i \vee \ldots \vee y_n x_n^i\right).$$

By negating Φ and removing redundant terms, we obtain:

$$\neg\Phi = \left[\bigwedge_{i=1}^{m}\bigwedge_{j=1}^{n+k_i} x_j^i\right] \vee \left[\bigwedge_{i=1}^{m} y_1x_1^i \vee \ldots \vee y_nx_n^i\right].$$

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

$$P(\neg\Phi) = P\left[\bigwedge_{i=1}^{m}\bigwedge_{j=1}^{n+k_i} x_j^i\right] + P\left[\bigwedge_{i=1}^{m} y_1x_1^i \vee \ldots \vee y_nx_n^i\right] - P\left[\bigwedge_{i=1}^{m}\bigwedge_{j=1}^{n+k_i} x_j^i\right] \cdot P\left[y_1 \vee \cdots \vee y_n\right].$$

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*

$$\bigwedge_{i=1}^{m} y_1x_1^i \vee \ldots \vee y_nx_n^i \qquad \textit{is} \qquad \sum_{j=1}^{n}\binom{n}{j}(2^n - 2^{n-j})^m.$$

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 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^i 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, for example, guarded negation [Bárány 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. Nonrepeating 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) subqueries obtained by choosing one term at each union and checking whether all of them are hierarchical. This approach fails since such subqueries are not necessarily $\exists$-consistent. For instance, the nonrepeating relational algebra query $Q = \pi_\emptyset[S - (R \bowtie S_1 \cup T \bowtie S_2)]$ over database schema $(S(A, B), R(A), S_1(A, B), T(B), S_2(A, B))$ has two hierarchical union-free subqueries under $\pi_\emptyset$: $\pi_\emptyset(S - (R \bowtie S_1))$ and $\pi_\emptyset(S - (T \bowtie S_2))$. However, these subqueries cannot be rewritten to $\exists$-consistent RC$^\exists$ queries, since they have roots A and B, respectively; it can be further shown that Q is #P-hard.

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

7.2. Nonrepeating $\mathrm{RC}^{\exists}$

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

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

8. RELATED WORK

Negation is a source of complexity already for databases with incomplete information and without probabilities [Abiteboul 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, 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 nonrepeating 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 $1\mathrm{RA}^{-}$ strictly generalises the dichotomy for nonrepeating conjunctive queries. Whereas tractable $1\mathrm{RA}^{-}$ 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, 2009; Jha and Suciu 2012], and queries with aggregates and group-by clauses [Ré 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. Similarly 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 ∃-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 on 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 nonrepeating conjunctive queries [Olteanu and Huang 2008]. For UCQ queries, the inversion-freenes 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, 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 [Valiant 1979; Provan and Ball 1983]. The complexity class #P was originally defined by Valiant [1979]. An overview of various topics in probabilistic databases has been compiled recently [Suciu et al. 2011].

9. CONCLUSION

This article discusses a fundamental computational aspect of query processing in probabilistic databases, namely the classification of nonrepeating 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, for example, with restricted union or repeating relation symbols, would still admit an efficient characterisation of tractable queries.

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 transitively joined attributes, while they are explicitly given in the case of conjunctive queries. Since the hierarchical property can be decided in AC_0 [Dalvi and Suciu 2007b] and since deciding whether two attributes are transitively 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 Q and indicates by three Boolean flags whenever one of the relations $R^{[A][\neg B]}$, $S^{[A][B]}$, or $T^{[\neg A][B]}$ has been found. This amounts to checking whether two attributes are transitively joined in Q, that is, 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 $\bowtie_\rho$ with $(X = Y) \in \rho$ or an operator $-_\rho$ with $(X \leftrightarrow Y) \in \rho$.

ALGORITHM 3: Algorithm to Decide the Hierarchical Property for $1RA^-$ Queries

```
IsHierarchical(Query Q)
  foreach pair of attributes A,B occurring in Q do
    HasA, HasB, HasAB ← false
    foreach relation symbol X in Q do
      if X exports [A] and not [B] then
        HasA ← true
      if X exports [A] and [B] then
        HasAB ← true
      if X exports [B] and not [A] then
        HasB ← true
    if HasA and HasB and HasAB then return "No"
  return "Yes"
```

COROLLARY A.1 ([DALVI AND SUCIU 2007B; REINGOLD 2008]). *The decision problem* IsHierarchical *is in LOGSPACE.*

ACKNOWLEDGMENTS

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

REFERENCES

Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Addison-Wesley, Reading, MA.

Serge Abiteboul, Paris Kanellakis, and Gösta Grahne. 1991. On the representation and querying of sets of possible worlds. *Theor. Comput. Sci.* 78, 1 (1991), 158–187.

Vince Bárány, Balder ten Cate, and Martin Otto. 2012. Queries with guarded negation. *Proc. VLDB* 5, 11 (2012), 1328–1339.

Paul Beame, Guy Van den Broeck, Eric Gribkoff, and Dan Suciu. 2015. Symmetric weighted first-order model counting. In *Proc. PODS*. 313–328.

Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam Junior Hruschka, and Tom Mitchell. 2010. Toward an architecture for never-ending language learning. In *AAAI*.

Nilesh Dalvi and Dan Suciu. 2004. Efficient query evaluation on probabilistic databases. In *VLDB* 864–875.

Nilesh Dalvi and Dan Suciu. 2007a. Efficient query evaluation on probabilistic databases. *VLDB J.* 16, 4 (2007), 523–544.

Nilesh Dalvi and Dan Suciu. 2007b. Management of probabilistic data: Foundations and challenges. In *Proc. PODS*. 1–12.

Nilesh N. Dalvi and Dan Suciu. 2012. The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM* 59, 6 (2012), 30.

Guy Van den Broeck. 2011. On the completeness of first-order knowledge compilation for lifted probabilistic inference. In *NIPS*. 1386–1394.

Xin Luna Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. 2014. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *SIGKDD*. 601–610.

Robert Fink, Larisa Han, and Dan Olteanu. 2012. Aggregation in probabilistic databases via knowledge compilation. *Proc. VLDB* 5, 5 (2012), 490–501.

Robert Fink, Jiewen Huang, and Dan Olteanu. 2013. Anytime approximation in probabilistic databases. *VLDB J.* 22, 6 (2013), 823–848.

Robert Fink and Dan Olteanu. 2014. A dichotomy for non-repeating queries with negation in probabilistic databases. In *PODS*. 144–155.

Robert Fink, Dan Olteanu, and Swaroop Rath. 2011. Providing support for full relational algebra queries in probabilistic databases. In *ICDE*. 315–326.

Erich Grädel, Yuri Gurevich, and Colin Hirsch. 1998. The complexity of query reliability. In *Proc. PODS*. 227–234.

Todd J. Green, Grigoris Karvounarakis, and Val Tannen. 2007. Provenance semirings. In *PODS*. 31–40.

Eric Gribkoff, Guy Van den Broeck, and Dan Suciu. 2014a. Understanding the complexity of lifted inference and asymmetric weighted model counting. In *UAI*. 280–289.

Eric Gribkoff, Dan Suciu, and Guy Van den Broeck. 2014b. Lifted probabilistic inference: A guide for the database researcher. *IEEE Data Eng. Bull.* 37, 3 (2014), 6–17.

M. Grohe, Y. Gurevich, D. Leinders, N. Schweikardt, J. Tyszkiewicz, and J. V. den Bussche. 2009. Database query processing using finite cursor machines. *Theor. Comput. Syst.* 44, 4 (2009), 533–560.

Jiewen Huang, Lyublena Antova, Christoph Koch, and Dan Olteanu. 2009. MayBMS: A probabilistic database management system. In *Proc. SIGMOD*. 1071–1074.

T. Imielinski and W. Lipski. 1984. Incomplete information in relational databases. *J. ACM* 31, 4 (1984), 761–791.

Abhay Kumar Jha and Dan Suciu. 2012. On the tractability of query compilation and bounded treewidth. In *Proc. ICDT*. 249–261.

Abhay Kumar Jha and Dan Suciu. 2013. Knowledge compilation meets database theory: Compiling queries to decision diagrams. *Theor. Comput. Syst.* 52, 3 (2013), 403–440.

Sanjeev Khanna, Sudeepa Roy, and Val Tannen. 2011. Queries with difference on probabilistic databases. *Proc. VLDB* 4, 11 (2011), 1051–1062.

Paraschos Koutris and Dan Suciu. 2011. Parallel evaluation of conjunctive queries. In *Proc. PODS*. 223–234.

Dan Olteanu and Jiewen Huang. 2008. Using OBDDs for efficient query evaluation on probabilistic databases. In *Proc. SUM*. 326–340.

Dan Olteanu and Jiewen Huang. 2009. Secondary-storage confidence computation for conjunctive queries with inequalities. In *Proc. SIGMOD*. 389–402.

Dan Olteanu and Hongkai Wen. 2012. Ranking query answers in probabilistic databases: Complexity and efficient algorithms. In *ICDE*. 282–293.

Dan Olteanu and Jakub Závodný. 2012. Factorised representations of query results: Size bounds and readability. In *Proc. ICDT*. 285–298.

Dan Olteanu and Jakub Závodný. 2015. Size bounds for factorised representations of query results. *ACM Trans. Database Syst.* 40, 1 (2015), 2.

J. Scott Provan and Michael O. Ball. 1983. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput.* 12, 4 (1983), 777–788.

Ralf Rantzau. 2004. Frequent itemset discovery with SQL using universal quantification. In *Database Support Data Mining Appl*. 194–213.

Sudhir Rao, Antonio Badia, and Dirk Van Gucht. 1996. Providing better support for a class of decision support queries. In *Proc. SIGMOD*. 217–227.

Christopher Ré and Dan Suciu. 2009. The trichotomy of HAVING queries on a probabilistic database. *VLDB J.* 18, 5 (2009), 1091–1116.

Omer Reingold. 2008. Undirected connectivity in log-space. *J. ACM* 55, 4 (2008), 17.

Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. 2011. *Probabilistic Databases*. Morgan & Claypool Publishers.

Leslie Valiant. 1979. The complexity of enumeration and reliability problems. *SIAM J. Comput.* 8 (1979), 410–421.

Ting-You Wang, Christopher Ré, and Dan Suciu. 2008. Implementing NOT EXISTS predicates over a probabilistic database. In *QDB/MUD*. 73–86.

Ingo Wegener. 2004. BDDs–design, analysis, complexity, and applications. *Discrete Appl. Math.* 138, 1–2 (2004), 229–251.

Received November 2014; revised June 2015; accepted October 2015