The Dichotomy of Probabilistic Inference for Unions of Conjunctive Queries[1]

Nilesh Dalvi
Facebook
and
Dan Suciu
University of Washington

We study the complexity of computing a query on a probabilistic database. We consider unions of conjunctive queries, UCQ, which are equivalent to positive, existential First Order Logic sentences, and also to non-recursive datalog programs. The tuples in the database are independent random events. We prove the following dichotomy theorem. For every UCQ query, either its probability can be computed in polynomial time in the size of the database, or is #P-hard. Our result also has applications to the problem of computing the probability of positive, Boolean expressions, and establishes a dichotomy for such classes based on their structure. For the tractable case, we give a very simple algorithm that alternates between two steps: applying the inclusion/exclusion formula, and removing one existential variable. A key and novel feature of this algorithm is that it avoids computing terms that cancel out in the inclusion/exclusion formula, in other words it only computes those terms whose Mobius function in an appropriate lattice is non-zero. We show that this simple feature is a key ingredient needed to ensure completeness. For the hardness proof, we give a reduction from the counting problem for positive, partitioned 2CNF, which is known to be #P-complete. The hardness proof is non-trivial, and combines techniques from logic, classical algebra, and analysis.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Systems—*Query Processing*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Mobius function, Mobius inversion formula, probabilistic database

1. INTRODUCTION

We study the complexity of computing a query on a probabilistic database. Our work is motivated by probabilistic databases [Cavallo and Pittarelli 1987; Dalvi and Suciu 2004; Sen and Deshpande 2007; Dalvi and Suciu 2007b; Sarma et al. 2008; Olteanu et al. 2009; Olteanu and Huang 2009; Suciu et al. 2011], the model counting problem, and the problem of computing the probability of propositional formulas [Creignou and Hermann 1996; Darwiche 2000; Darwiche and Marquis 2002; Wegener 2004; Golumbic et al. 2006; Domingos and Lowd 2009].

A *probabilistic database* is a pair $\mathbf{D} = (D, P)$ where D is a database instance (i.e. a set of tuples) and $P : D \rightarrow [0, 1]$ associates a probability with each tuple $t \in D$. This defines a probability space, where the outcomes are the subsets W of D, and each tuple t is included in W independently, with probability $P(t)$. Thus, the probability of a world W is:

[1]This work was partially supported by NSF IIS-0713576 and IIS-1115188.

$$P_{\mathbf{D}}(W) = \prod_{t \in W} P(t) \times \prod_{t \in D-W} (1 - P(t)) \quad (1)$$

A query, Q, is sentence in First Order Logic. Given Q, $P_{\mathbf{D}}(Q)$ denotes its *marginal* probability, the probability that Q is true on a randomly chosen world W:

$$P_{\mathbf{D}}(Q) = \sum_{W \subseteq D : W \models Q} P_{\mathbf{D}}(W) \quad (2)$$

The problem is this. For a fixed query Q, what is the complexity of computing $P_{\mathbf{D}}(Q)$ as a function of the size of $\mathbf{D}$? We consider the data complexity, where Q is fixed and the input consists only of $\mathbf{D}$. The queries that we study are Unions of Conjunctive Queries, for which we use an alternative, but equivalent syntax, as positive, existential First Order formulas (sentences with connectives $\wedge, \vee, \exists$). This set of queries has the same expressive power as *non-recursive datalog* programs [Abiteboul et al. 1995].

We prove in this paper the following Dichotomy Theorem (Theorem 4.21). For any query Q, the problem "given $\mathbf{D}$, compute $P_{\mathbf{D}}(Q)$" is either in PTIME or #P hard. Moreover, one can decide between the two cases through syntactic analysis on Q.

Using simple transformations, our results also apply to the positive, universal subset of First Order Logic (sentences with connectives $\wedge, \vee, \forall$). Such sentences occur in knowledge representation, like, for example, Markov Logic Networks [Richardson and Domingos 2006; Domingos and Lowd 2009]. MLNs have been demonstrated to be effective at a variety of tasks, such as Information Extraction [Poon and Domingos 2007], Record Linkage [Singla and Domingos 2006], Natural Language Processing [Poon and Domingos 2010].

Our dichotomy result also implies a dichotomy for probabilistic inference on positive Boolean formulas. For each query Q, its *lineage* on a database instance $\mathbf{D}$ is a Boolean formula Φ_Q^D that has one Boolean variable X_t for each tuple t in the database: the lineage is defined such that it is true on an truth assignment θ iff the query is true on the world W consisting of all tuples t for which $\theta(X_t)$ is true. The lineage formula is derived naturally from the query expression Q and the database $\mathbf{D}$ (reviewed in Sect. 2), and has a DNF representation of polynomial size in the database. It turns out that the probability of the query Q is equal to the probability of the Boolean formula $\Phi_Q^{\mathbf{D}}$, where each Boolean variable X_t is set independently to `true` with a probability $P(t)$. Thus, our dichotomy theorem implies a dichotomy for the classes of DNF expressions defined by query lineages. In fact, it applies equally well to their dual CNF expressions, obtained by switching $\wedge$ and $\vee$. For a taste of how our results apply to CNF expressions, Table I shows several queries Q, the corresponding CNF expressions Φ (the duals of their lineages), and indicates whether computing $P(\Phi)$ is in PTIME or #P-hard.

A special case of the probability computation problem is the *model counting* problem: given a Boolean expression Φ, count the number of satisfying assignments $\#\Phi$. [Valiant 1979] showed that #SAT, the counting version of the SAT problem, is #P-complete. [Provan and Ball 1983] have shown that for a very simple class of

Query	Φ = The Dual of the Lineage	$P(\Phi)$
$q_J = R(x_1), S(x_1, y_1), S(x_2, y_2), T(x_2)$	$\Phi = \bigwedge_{ijkl}(Y_i \vee X_{ij} \vee X_{kl} \vee Z_k)$	PTIME
$q_U = R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(x_2)$	$\Phi = \bigwedge_{ij}(Y_i \vee X_{ij}) \wedge \bigwedge_{ij}(X_{ij} \vee Z_i)$	PTIME
$h_1 = R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(y_2)$	$\Phi = \bigwedge_{ij}(Y_i \vee X_{ij}) \wedge \bigwedge_{ij}(X_{ij} \vee Z_j)$	#P-hard
$q_V = R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(y_2)$ $\vee R(x_3), T(y_3)$	$\Phi = \bigwedge_{ij}(Y_i \vee X_{ij}) \wedge \bigwedge_{ij}(X_{ij} \vee Z_j) \wedge$ $\bigwedge_{ij}(Y_i \vee Z_j)$	PTIME
$q_H =$ $(R(x_1), S_1(x_1, y_1) \vee S_2(x_2, y_2), S_3(x_2, y_2))$ $\wedge(S_1(x_3, y_3), S_2(x_3, y_3) \vee S_3(x_4, y_4), T(y_4))$	$\Phi = \bigwedge_{i_1 j_1 \ldots i_4 j_4}$ [$((Y_{i_1} \vee X^1_{i_1 j_1}) \wedge (X^2_{i_2 j_2} \vee X^3_{i_2 j_2})) \vee$ $((X^1_{i_3 j_3} \vee X^2_{i_3 j_3}) \wedge (X^3_{i_4 j_4} \vee Z_{j_4}))$]	#P-hard
$q_W =$ $(R(x_1), S_1(x_1, y_1) \vee S_2(x_2, y_2), S_3(x_2, y_2))$ $\wedge(R(x_3), S_1(x_3, y_3) \vee S_3(x_4, y_4), T(y_4))$ $\wedge(S_1(x_5, y_5), S_2(x_5, y_5) \vee S_3(x_6, y_6), T(y_6))$	$\Phi = \bigwedge_{i_1 j_1 \ldots i_6 j_6}$ [$((Y_{i_1} \vee X^1_{i_1 j_1}) \wedge (X^2_{i_2 j_2} \vee X^3_{i_2 j_2})) \vee$ $((Y_{i_3} \vee X^1_{i_3 j_3}) \wedge (X^3_{i_4 j_4} \vee Z_{j_4})) \vee$ $((X^1_{i_5 j_5} \vee X^2_{i_5 j_5}) \wedge (X^3_{i_6 j_6} \vee Z_{j_6}))$]	PTIME

Table I. Some simple applications of the dichotomy result. The first column shows a query: all variables are existentially quantified. The middle column shows the Boolean expression obtained by taking the dual of their lineage expression (switching $\wedge, \vee$). The Boolean variables Y_i, X^k_{ij}, Z_j correspond to the tuples $R(i), S_k(i, j), T(j)$ respectively. The first four expressions are already in CNF. The last two expressions are not in CNF, but can be rewritten in CNF with only a constant factor increase in their size. The last column indicates whether $P(Q)$, or, equivalently, $P(\Phi)$, is tractable or not. Consider the row for h_1, where $P(\Phi)$ is hard for #P. Compare it to q_U, where Z_j becomes Z_i, and the complexity of $P(\Phi)$ is PTIME. Also, compare it to q_V, where a new set of clauses is added to Φ and the complexity is also PTIME. A similar comparison can be done for q_H and q_V.

Boolean expressions, called *partitioned, positive, 2CNF* (reviewed in Sect. 5), the counting problem is already #P-complete. Our PTIME algorithm for $P(\Phi)$ extends immediately to the counting problem, since $\#\Phi = 2^n P(\Phi)$, where n is the number of Boolean variables, and the probability $P(\Phi)$ is computed by setting each Boolean variable independently to `true` with probability 1/2. However, the hardness results do not extend immediately: for most queries for which computing the probability is #P-hard, it is open whether the model counting problem is also #P-hard.

[Creignou and Hermann 1996] proved a dichotomy theorem for the *generalized satisfiability counting problem*, #GenSAT. They consider Boolean expressions that are conjunctions of *logical relations*, also known as generalized clauses, and establish a dichotomy for the counting problem based on the type of logical relations: they

show that the counting problem is in PTIME iff every logical relation is affine. Thus, they restrict the formulas based on the generalized clauses, and allow otherwise arbitrary structure. We do the opposite: we restrict the structure, and obtain many interesting tractable cases that are not identified by Creignou and Hermann's result. This is illustrated in Table I: all clauses in this table are of the form $x \vee y \vee z \vee \ldots$ and therefore fall in the #P-class of Creignou and Hermann. By restricting the structure, we obtain many interesting classes in PTIME.

We discuss now in some more details the two parts of our dichotomy result: the PTIME algorithm for the tractable queries, and the hardness proof for the intractable queries.

The Algorithm We have first described the PTIME algorithm in [Dalvi et al. 2010]. The version presented here in Sect. 3 makes several improvements, both in presentation and in performance. The algorithm has two simple steps. First, it uses inclusion/exclusion: this removes the outer-most $\wedge$ operations in a query. Second, it removes the outermost $\exists x$ quantifier by substituting it with a constant. The variable x must satisfy certain properties and is called a *separator variable*; if no separator variable exists, the algorithm fails. The algorithm alternates between these two steps, until it reaches ground atoms, for which it looks up their probabilities in the database.

The inclusion/exclusion is the dual of the popular version: the algorithm computes $P(q_1 \wedge q_2 \wedge \ldots)$ in terms of $P(q_{i_1} \vee q_{i_2} \vee \ldots)$, rather than the other way around. For that reason, we express Q in a non-traditional syntax, as a Boolean CNF expression over connected, conjunctive queries. In particular, even if one starts with a conjunctive query, the algorithm may eventually introduce $\vee$ (see Example 3.1). In other words, the algorithm does not become any simpler if restricted to conjunctive queries, since it eventually reaches unions of conjunctive queries. The use of CNF rather than the DNF representation, and of the dual inclusion/exclusion formula are somewhat surprising features of the algorithm.

Terms of the inclusion/exclusion formula may cancel out, if they correspond to equivalent queries. In some extreme cases, some of these terms are #P-hard, but they cancel out and all remaining terms are PTIME (see Example 4.7). Thus, it is critical for the algorithm to recognize these cancellations, and we do this by replacing the inclusion/exclusion formula with Mobius' inversion formula in a lattice [Stanley 1997]. More precisely, we construct a lattice for the query Q, called the *CNF-lattice* because it is based on the CNF representation of the query. The elements of the lattice represent precisely the terms of the inclusion/exclusion formula. The algorithm computes recursively only those terms where the Mobius function is $\neq 0$. While the inclusion/exclusion formula, or, equivalently, Mobius' inversion formula, have been used before in probabilistic inference [Knuth 2005], the strong connection between the zeroes of the Mobius function and the complexity of probabilistic inference is novel.

The power of our simple algorithm has been highlighted by recent results in [Jha and Suciu 2011]. Several notions of tractability for computing $P(\Phi)$ were discussed there: read-once Boolean expressions [Golumbic et al. 2006], polynomial-size OBDDs and FBDDs [Wegener 2000; Wegener 2004], and polynomial-size d-DNNFs [Darwiche 2000; Darwiche and Marquis 2002]. It was shown that they form a strict hierarchy for

unions of conjunctive queries: in particular, the queries q_J, q_V, q_W, q_9, shown in Table I, Fig. 1 and Fig. 2, separate these classes (e.g. q_J is not read-once, but has a polynomial-size OBDD, etc). It is further conjectured that q_9 (Fig. 2), whose probability can be computed in PTIME using our algorithm, does not have a polynomial size d-DNNF.

The Hardness Proof The hardness proof of our dichotomy theorem is entirely new, and forms the main technical contribution of this paper. We prove the following. Call a query Q *unsafe* if our algorithm fails on Q (formally defined in Def. 4.14). We prove that for any unsafe query Q there exists a PTIME Turing machine with an oracle for computing $P_{\mathbf{D}}(Q)$ that solves the counting problem for the partitioned-positive-2CNF problem. The latter was shown to be #P-hard by [Provan and Ball 1983]. This implies that $P_{\mathbf{D}}(Q)$ is hard for #P. In our proof, the oracle for $P_{\mathbf{D}}(Q)$ is invoked polynomially many times, on the same database instance D, but with varying probabilities of the tuples in the database. The reduction is quite complex, but we have broken it down in several smaller steps, which we explain next.

The first step, described in Sect. 6, is called *leveling*. A leveled database generalizes the notion of a k-partite graph. Its active domain is partitioned into subsets, called levels, and for every relation its attributes belong to different levels. We prove that every unsafe query Q can be *leveled*, by constructing a leveled, unsafe query Q' such that the evaluation problem for Q' can be reduced to that for Q. For example, consider the query $Q = \exists x.\exists y.\exists z.R(x,y) \wedge R(y,z)$, in our notation written as $Q = R(x,y), R(y,z)$, which checks whether a graph has a path of length 2. To show that the query is #P-hard we specialize it to 4-partite graphs, $Q' = R^{12}(x,y), R^{23}(y,z) \vee R^{23}(x,y), R^{34}(y,z)$, and prove that Q' is #P-hard; we call Q' a leveled query. This first step allows us to restrict the hardness proof to leveled queries.

The second step, described in Sect. 7, applies some simple rewriting rules to simplify an unsafe, leveled query Q to another unsafe, leveled query Q' such that the evaluation problem for Q' can be reduced to that for Q. We prove that these rewrite rules can be applied as long as the query has ≥ 3 levels. Thus, every unsafe query Q can be rewritten to an unsafe query Q' with two levels: we call these *forbidden queries*. For example, the query Q' above can be rewritten by collapsing levels 1 and 4 to a single constant, a and b respectively: $R^{12}(a,y), R^{23}(y,z) \vee R^{23}(x,y), R^{34}(y,b)$, which, up to renaming of the relational symbols and of variables, is equivalent to $h_1 = R(x), S(x,y) \vee S(x,y), T(y)$, which is a forbidden query (also shown in Table I): to show that Q' is hard, it suffices to prove that h_1 is hard. The difficulty of this second step is in showing that progress can always be made, as long as there are at least 3 levels.

The third step proves that every forbidden query Q is hard for #P. This is the most interesting part of the proof, and we start by describing the main intuition and introducing the main techniques in Sect. 5, then give the full proof details in Sect. 8. (Readers interested in the main proof techniques may go directly to Sect. 5, which is mostly self-contained.) This step is a direct reduction from Provan and Ball's partitioned-positive-2CNF counting problem, #Φ. The reason why it is difficult is that the lineage of the forbidden query is, in general, quite different from a PP2CNF

formula. There is n direct relationship between $P_D(Q)$ and $\#\Phi$, instead $P_D(Q)$ is a linear function of polynomially many parameters of Φ (such as, the number of truth assignments that satisfy exactly k clauses, etc). The reduction consists of running the oracle for $P_D(Q)$ repeatedly, constructing a system of linear equations, then solving it. The reduction works iff the matrix of the linear system is non-singular, and this happens iff the Jacobian of a certain set of multi-linear polynomials is non-zero. The size of the Jacobian depends only the query Q, and is 4×4 for the simplest forbidden query. It is far from obvious why the Jacobian would be non-zero, and, in fact, it *is* zero for all queries Q that are not forbidden queries. To prove that the Jacobian is non-zero, we construct the database D in a special way, in which the Jacobian turns out to be Cauchy's double alternant, $\det\left(\frac{1}{x_i+y_j}\right)_{ij}$, which has a simple closed form and is non-zero iff all values $x_1, \ldots, x_n$ are distinct and all values $y_1, \ldots, y_n$ are distinct. The values $x_1, \ldots, x_n$ represent probabilities of tuples in the database and we can easily choose them to be distinct. However, the values $y_1, \ldots, y_n$ depend on the query, and it is not obvious at all why they should be distinct: in fact, for non-forbidden queries they are *not* distinct. Here, we prove that these values are distinct if a certain multilinear polynomial (representing the query's probability) is irreducible: the latter happens if and only if the query is forbidden. We describe these three techniques from classical algebra and from analysis in Sect. 5; a fourth, quite non-obvious technique that uses Mobius' inversion formula to compute $P_D(Q)$, is described in Sect. 8, which contains all details of the hardness proof.

Previous work on Conjunctive Queries In previous work [Dalvi and Suciu 2007a] we described a PTIME algorithm for conjunctive queries, and claimed that it is complete, in the sense that every query on which the algorithm fails is #P-hard. The completeness proof consisted of showing that, if the algorithms gets stuck on a query Q, then there exists a reduction from h_k to Q, where h_k is a query for which we prove hardness directly (defined in Example 4.6). But the proof was very complex, since it combined all of Sect. 6, Sect. 7, and parts of Sect. 8 in one single giant step, and was very sensitive to small changes in the algorithm. While initial proof sketches seemed to work, we were not able to finalize the proof, and re-started, leading to the current work. In hindsight, the difficulties we encountered in [Dalvi and Suciu 2007a] came from working on the standard, DNF representation of a conjunctive query; that form prevents induction on the query's structure, making the algorithm more complex, and making the hardness proof impossibly difficult. In Appendix A we clarify in detail the status of the algorithm in [Dalvi and Suciu 2007a]. In a nutshell, the algorithm is sound (after fixing some minor mistakes, as we explain in Appendix A), in that it computes correctly the probability whenever it succeeds, and is also complete for queries over vocabularies of arities ≤ 2; it remains open whether it is complete in general. The plan of relying only on the h_k queries to prove hardness was probably also doomed to fail. While every non-hierarchical query rewrites to $h_0 = R(x), S(x,y), T(y)$ (Prop. 8.8), in general hierarchical queries do not rewrite to h_k, but to the more complex *forbidden queries*, discussed in Sect. 8.

Related work [Grädel et al. 1998] were the first to give an example of a query Q for which $P(Q)$ is #P-hard; their work was done in the context of query reliability.

In the following years, several studies [Dalvi and Suciu 2004; Dalvi and Suciu 2007b; Olteanu et al. 2009; Olteanu and Huang 2009], sought to identify classes of tractable queries. These works provided conditions for tractability only for conjunctive queries without self-joins, with the exception of [Dalvi and Suciu 2007a]. We extend those results to a larger class of queries, and at the same time provide a very simple algorithm. Some other prior work is complimentary to ours, e.g., the results that consider the effects of functional dependencies [Olteanu et al. 2009].

2. DEFINITIONS AND BACKGROUND

2.1 Problem Definition

We fix a relational vocabulary $\mathbf{R} = (R_1, \ldots, R_k)$, and denote $arity(R_i)$ the arity of a relation R_i. A *database instance* over $\mathbf{R}$ is $D = (R_1^D, \ldots, R_k^D)$, where each R_i^D is a finite relation. We call the elements of R_i^D *tuples*. With some abuse of notation we also denote D as the set of all tuples, i.e. $D = R_1^D \cup \ldots \cup R_k^D$, where the union is understood to be a disjoint union. The *active domain*, $ADom(D)$ or simply $ADom$ when D is understood from the context, is the set of all constants occurring in D. The *size* of the database instance D is $n = |ADom|$. A *tuple-independent probabilistic database* is a pair $\mathbf{D} = (D, P)$, where $P : D \to [0, 1]$; its semantics is the probability space $P_{\mathbf{D}}$ defined by Eq. 1, and its size is the size of D. For complexity results, we assume that the probabilities are given as rational numbers, and the size of their representation is bounded by a polynomial in n.

In this paper we discuss *unions of conjunctive queries* (UCQ); we consider only Boolean queries. We first review their traditional syntax, then introduce alternative syntactic representations that we will use throughout the paper. A *conjunctive query*, CQ, is an expression of the form $q = \exists x_1 \ldots \exists x_k . g_1 \wedge \ldots \wedge g_m$, where each g_i is some relational atom $R(\bar{x})$ with variables and/or constants. In the traditional syntax we drop all existential quantifiers and replace $\wedge$ with comma. A *union of conjunctive queries* is an expression of the form $Q = q_1 \vee \ldots \vee q_p$, where each q_i is a conjunctive query. Thus, the traditional syntaxes for CQ and UCQ are:

$$q = g_1, g_2, \ldots, g_m \tag{3}$$

$$Q = q_1 \vee \ldots \vee q_p \tag{4}$$

The UCQ examples in Table I follows this traditional syntax.

In this paper we will represent queries as existential, positive FO formulas:

DEFINITION 2.1. *A* query expression *is given by the following grammar:*

$$Q = R(\bar{x}) \mid \exists x . Q_1 \mid Q_1 \wedge Q_2 \mid Q_1 \vee Q_2 \tag{5}$$

Here $R(\bar{x})$ is a relational atom with variables and/or constants. Thus, in this paper, a query expression Q is an existential, positive FO formula (no negation and no universal quantifiers). Since we only consider Boolean queries, Q must be a closed formula, i.e. without free variables. Given a database instance D and a query expression Q, we write $D \models Q$ if the query Q is true on D; we refer the reader to [Libkin 2004] for an introduction to FO formulas and their interpretation on first order structures. If $\mathbf{D}$ is a probabilistic database, then the semantics of Q on $\mathbf{D}$ is given by the probability $P_{\mathbf{D}}(Q)$, see Eq. 2. When $\mathbf{D}$ is clear from the context, then we drop the index and write $P(Q)$.

Two query expressions Q, Q' are *equivalent* if $D \models Q$ iff $D \models Q'$, for all database instances D; if Q and Q' are equivalent, then $P_{\mathbf{D}}(Q) = P_{\mathbf{D}}(Q')$.

Every UCQ query given in the traditional syntax Eq. 4 is also an existential, positive FO formula. The converse is also known to hold: every existential, positive FO formula (given by Eq. 5) is equivalent to some UCQ query. Therefore, in this paper we will denote UCQ to be the set of query expressions given by Eq. 5.

$Var(Q)$ denotes the set of variables in Q. Throughout the paper we make the standard assumption that each existential quantifier uses a distinct variable. It can be enforced by renaming existential variables used twice; for example, we do not write a query like $\exists x.R(x) \vee \exists x.S(x)$, because it uses $\exists x$ twice, but instead rewrite it as $\exists x.R(x) \vee \exists y.S(y)$, or also as $\exists x.(R(x) \vee S(x))$. It follows that, if Q_1, Q_2 are two Boolean queries occurring in the context $Q_1 \vee Q_2$ or $Q_1 \wedge Q_2$, then $Var(Q_1) \cap Var(Q_2) = \emptyset$. When we apply a Boolean identity that repeats a subexpression, then assume that variables are renamed: e.g. in the identity $Q_1 \vee (Q_2 \wedge Q_3) = (Q_1 \vee Q_2) \wedge (Q_1 \vee Q_3)$ we assume that the two occurrences of Q_1 on the right use disjoint sets of variables.

DEFINITION 2.2. *A variable z is called a* root variable *in Q if it occurs in all atoms of Q.*

For example, given the query $\exists x.\exists y.(R(x) \wedge S(x,y))$, x is a root variable while y is not a root variable. The query $\exists x.\exists y.R(x) \wedge S(x,y) \wedge T(y)$ has no root variable.

An *attribute* is a pair (R, i), where $R \in \mathbf{R}$ is a relation name, and $i \in [arity(R)]$. Let Q be any query expression. The *attribute graph of Q* is the undirected graph whose nodes are attributes (R, i), and whose edges are pairs $((R, i), (S, j))$ s.t. Q contains two atoms $R(\ldots)$ and $S(\ldots)$ that have the same variable x on attributes i and j respectively.

DEFINITION 2.3. *A* level *of a query expression Q is a connected component of its attribute graph.*

For a simple example the query $\exists x.\exists y.\exists z.R(x,y) \wedge S(y,z)$ has three levels, $\{(R,1)\}$, $\{(R,2),(S,1)\}$, and $\{(S,2)\}$, while the query $\exists x.\exists y.\exists z.R(x,y) \wedge R(y,z)$ has a single level, $\{(R,1),(R,2)\}$.

If Z is a level and $x \in Var(Q)$ a variable, we say that x occurs on level Z if the query has some atom R that contains x on attribute i, and $(R,i) \in Z$. Denote $Var_Z(Q)$ the set of variables that occur on level Z.

In general the attribute graph, and the levels, depend on the query expression: for example the attribute graph for $\exists x.R(x) \vee \exists y.S(y)$ has two levels, while it has only one level for the equivalent expression $\exists z.(R(z) \vee S(z))$.

The *query evaluation problem on probabilistic databases* is the following. Given a UCQ query Q and a probabilistic database $\mathbf{D}$, compute $P_{\mathbf{D}}(Q)$. We are interested in the *data complexity*: for a fixed query Q, what is the complexity of computing $P_{\mathbf{D}}(Q)$ as a function of the size of $\mathbf{D}$?

DEFINITION 2.4. *$UCQ(P)$ is the class of UCQ queries, Q, s.t. there exists an algorithm that, given a probabilistic database $\mathbf{D}$, computes the probability $P_{\mathbf{D}}(Q)$ in PTIME in the size of $\mathbf{D}$.*

The main result of this paper is a complete syntactic characterization of the class

$UCQ(P)$ (assuming $FP \neq \#P$), and establishes the following dichotomy: for every query not in $UCQ(P)$, computing $P_{\mathbf{D}}(Q)$ is hard for $\#P$.

2.2 Review: Query Lineage

We annotate each tuple t in a database instance D with a distinct Boolean variable X_t from a set $\mathbf{X}$. Let Q be a query expression. The *lineage* of Q on D is the Boolean expression Φ_Q^D, or simply Φ_Q if D is understood from the context, defined inductively as follows.

$$\Phi_{R(\bar{a})} = X_{R(\bar{a})} \qquad \Phi_{\exists x.Q} = \bigvee_{a \in ADom} \Phi_{Q[a/x]} \tag{6}$$

$$\Phi_{Q_1 \wedge Q_2} = \Phi_{Q_1} \wedge \Phi_{Q_2} \qquad \Phi_{Q_1 \vee Q_2} = \Phi_{Q_1} \vee \Phi_{Q_2} \tag{7}$$

For a simple example, if $Q = \exists x.(R(x) \wedge \exists y.S(x,y))$ and D is a database instance consisting of three tuples, $D = \{R(a), S(a,b_1), S(a,b_2)\}$, annotated with the Boolean variables X, Y_1, Y_2, then $\Phi_Q^D = X \wedge (Y_1 \vee Y_2)$

Notice that, for a fixed query expressions Q, the size of the lineage Φ_Q^D is polynomial in n (the size of $ADom$). If Q is a standard UCQ query expression (Eq. 4) then Φ_Q^D is in DNF. In general, if Q is any expression given by Eq. 5, then Φ_Q^D can be converted to a DNF expression in time polynomial in n.

If $D_i = (R_1^{D_i}, \ldots, R_k^{D_i})$, $i = 1, 2$ are two database instances then we write $D_1 \subseteq D_2$ whenever $R_1^{D_1} \subseteq R_1^{D_2}$, ..., $R_k^{D_1} \subseteq R_k^{D_2}$. A subset $W \subseteq D$ is called a *possible world* or a *world*. Each world W defines the truth assignment $\theta_W : \mathbf{X} \to \{\texttt{false}, \texttt{true}\}$, given by $\theta_W(X_t) = \texttt{true}$ iff $t \in W$. The following fact is folklore:

PROPOSITION 2.5. *Let Q be a query expression, D a database instance, and Φ_Q^D its lineage. Then, for all $W \subseteq D$, we have $W \models Q$ iff $\Phi_Q^D[\theta_W] = \texttt{true}$.*

It follows that, for any two query expressions Q_1, Q_2, the logical implication $\Phi_{Q_1}^D \Rightarrow \Phi_{Q_2}^D$ holds iff for every possible world $W \subseteq D$, $W \models Q_1$ implies $W \models Q_2$. In particular, two equivalent query expressions have equivalent lineage expressions.

The lineage expression gives us an alternative way to compute the query probability $P_{\mathbf{D}}(Q)$ on a probabilistic database $\mathbf{D} = (D, P)$. Assign each Boolean variable X_t independently to $\texttt{true}$ with probability $P(t)$ and let $P(\Phi_Q^D)$ denote the probability that the expression Φ_Q^D becomes $\texttt{true}$; then $P_{\mathbf{D}}(Q) = P(\Phi_Q^D)$. Since each query expression Q defines a family of Boolean expressions, namely the family Φ_Q^D where D ranges over all finite databases, the dichotomy result in this paper is also a characterization of families of Boolean expressions that are computable in PTIME, and establish a dichotomy on these families. Such families of formulas occur in other contexts beyond probabilistic databases, for example in Markov Logic Networks[Domingos and Lowd 2009] and in Model Counting [Gomes et al. 2009].

2.3 Review: Query Containment and Minimization

We briefly review the classical results on query containment and query minimization for CQ and for UCQ, and refer the reader to [Abiteboul et al. 1995] for further detail. These results assume the traditional syntax given by Eq. 3 and Eq. 4. Given a conjunctive query q, we denote D_q the *canonical database*: its active domain

consists of all constants and variables in q, and it has a tuple for each atom in q. The following is a classic result by Chandra and Merlin [Chandra and Merlin 1977]. Given two conjunctive queries q, q', the following statements are equivalent: (1) the logical implication $q \Rightarrow q'$ holds; (2) there exists a homomorphism $q' \rightarrow q$; (3) $D_q \models q'$. Containment and equivalence are NP complete problems. A conjunctive query q given by Eq. 3 is *minimized* if there is no other equivalent expression q' with strictly fewer relational atoms. Each conjunctive query q is equivalent to a unique minimized query (up to isomorphism). The following is a classic result by Sagiv and Yannakakis [Sagiv and Yannakakis 1980]. Given two unions of conjunctive queries, $Q = \bigvee_i q_i$ and $Q' = \bigvee_j q'_j$, the following are equivalent: (1) the logical implication $Q \Rightarrow Q'$ holds; (2) $\forall i.\exists j$ such that $q_i \Rightarrow q_j$. A UCQ given by the expression Eq. 4 is *minimized* if each q_i is a minimized conjunctive query, and $q_i \Rightarrow q_j$ implies $i = j$ (in other words no two distinct queries are contained). Each UCQ query Q is equivalent to a unique minimal UCQ query (up to isomorphism).

2.4 FO Reductions

In several places of the hardness proof we use polynomial time reductions from some query Q to another query Q'. The reductions we need are very simple: all are FO mappings, and in most cases they have an even simpler form, which we call linear FO mappings.

Fix two relational vocabularies $\mathbf{R} = (R_1, \ldots, R_k)$ and $\mathbf{R}' = (R'_1, \ldots, R'_m)$. An *FO mapping* from $\mathbf{R}$ to $\mathbf{R}'$ is a tuple $\mathbf{F} = (F_1, \ldots, F_m)$, where each F_i is relational algebra expression of arity $arity(R'_i)$ over the vocabulary $\mathbf{R}$. All FO-mappings in this paper are combinations of selections, projections, and unions (we don't need joins), denoted using the standard symbols $\sigma, \Pi, \cup$. We write $\mathbf{F}(D)$ for the result of applying $\mathbf{F}$ to an instance D.

We extend an FO mapping to a mapping between probabilistic databases. If $\mathbf{D} = (D, P)$, then we assume that each tuple in t has an extra attribute whose value is the probability of the tuple, $P(t)$. The FO mapping is defined over a schema that includes this attribute, and must return a value for the probability of each output tuple. In all examples in this paper, the output probability is either copied from the input, or is defined to be 1.

DEFINITION 2.6. *A reduction $Q \leq^{FO}_{prob} Q'$ consists of an FO mapping $\mathbf{F}$ from the vocabulary of Q to that of Q', and an algorithm with inputs $(\mathbf{D}, P_{\mathbf{F}(\mathbf{D})}(Q'))$ that computes $P_{\mathbf{D}}(Q)$ in time polynomial in the size of $\mathbf{D}$. Equivalence, $Q \equiv^{FO}_{prob} Q'$, is defined as $Q \leq^{FO}_{prob} Q'$ and $Q' \leq^{FO}_{prob} Q$.*

The intuition is very simple. To compute $P_{\mathbf{D}}(Q)$, we first apply the mapping $\mathbf{F}$ and obtain $\mathbf{D}' = \mathbf{F}(\mathbf{D})$, use an oracle to compute $p = P_{\mathbf{D}'}(Q')$, then we run the algorithm on the database $\mathbf{D}$ and the value p to compute $P_{\mathbf{D}}(Q)$. Obviously, if $Q' \in UCQ(P)$ then $Q \in UCQ(P)$; also, if Q is #P-hard, then Q' is #P-hard.

For example, consider the queries $h_1 = R(x_0), S(x_0, y_0) \vee S(x_1, y_1), T(y_1)$ and $h'_1 = R(x_0), S(x_0, y_0), S(x_1, y_1), T(y_1)$. Then we claim that $h_1 \leq^{\text{FO}}_{\text{prob}} h'_1$. Indeed, denote $h_{10} = R(x_0), S(x_0, y_0)$ and $h_{11} = S(x_1, y_1), T(y_1)$: we will show below (Example 2.14) that both $P(h_{10})$ and $P(h_{11})$ can be computed in PTIME in the size of the probabilistic databases. Then $P(h_1) = P(h_{10} \vee h_{11}) = P(h_{10}) + P(h_{11}) - P(h'_1)$; thus, given an oracle for computing $p = P(h'_1)$, we can compute $P(h_1)$ in

polynomial time, since the expression $P(h_{10})+P(h_{11})$ can be computed in PTIME. This shows $h_1 \leq^{\text{FO}}_{\text{prob}} h'_1$. We will prove in Prop. 5.2 that h_1 is #P-hard, and this implies that h'_1 is also #P-hard.

A *linear FO* mapping is a mapping $\mathbf{F}$ satisfying the following three properties: (1) $\mathbf{F}(D_1) \cup \mathbf{F}(D_2) = \mathbf{F}(D_1 \cup D_2)$, (2) $\mathbf{F}(D_1) \cap \mathbf{F}(D_2) = \mathbf{F}(D_1 \cap D_2)$, and (3) $|\mathbf{F}(D)| \leq |D|$. If $\mathbf{F}$ is linear, then for any single tuple t we have either $\mathbf{F}(\{t\}) = \{t'\}$ or $\mathbf{F}(\{t\}) = \emptyset$; furthermore, $\mathbf{F}(D) = \bigcup_{t \in D} \mathbf{F}(\{t\})$. A linear mapping $\mathbf{F}$ establishes an injective function $\mathbf{f} : D' \to D$ from the tuples in $D' = \mathbf{F}(D)$ to those in D, by $\mathbf{f}(t') = t$ where t is the unique tuple s.t. $\mathbf{F}(\{t\}) = \{t'\}$. [2]

For an illustration, if $R(A, B, C)$ is a relation of arity 3 and $R'(B, C)$ has arity 2, then $R' = \Pi_{BC}(\sigma_{A=a}(R))$ defines a linear mapping $\mathbf{F}$ from R to R', where a is constant. The function $t = \mathbf{f}(t')$ pads the tuple t' with the constant a, i.e. if $t' = (b, c)$, then $t = (a, b, c)$. Thus, given a tuple $t = (a_1, b_1, c_1)$, $\mathbf{F}(\{t\})$ is either $\{(b_1, c_1)\}$ or $\emptyset$ (depending on whether $a_1 = a$ or $a_1 \neq a$). In the other direction, given $t' = (b_1, c_1)$, $\mathbf{f}(t')$ is always defined as (a, b_1, c_1).

Definition 2.7. *Let Q, and Q' be two queries. A lineage reduction, $Q \leq^{FO}_{lin,+} Q'$, consists of a pair $(\mathbf{F}, \mathbf{E})$ where $\mathbf{F}$ is a linear mapping, and $\mathbf{E}$ an FO mapping, such that the following holds. Let D be any database, annotated with Boolean variables $\mathbf{X} = \{X_t \mid t \in D\}$, let $D' = \mathbf{F}(D) \cup \mathbf{E}(D)$, and let $\mathbf{Y}$ be a set of Boolean variables disjoint from $\mathbf{X}$. Annotate each tuple $t' \in \mathbf{F}(D)$ with $X_{\mathbf{f}(t')}$, and annotate tuples in $\mathbf{E}(D) - \mathbf{F}(D)$ with variables from $\mathbf{Y}$, then*[3]*: $\Phi^D_Q \equiv \Phi^{D'}_{Q'}[\mathbf{Y} = \textbf{\textit{true}}]$. When $\mathbf{E} = \emptyset$, then the condition becomes $\Phi^D_Q \equiv \Phi^{D'}_{Q'}$, and we say that the reduction is* strict *and write $Q \leq^{FO}_{lin} Q'$.*

Lineage equivalence, $Q \equiv^{FO}_{lin,+} Q'$, is defined as $Q \leq^{FO}_{lin,+} Q'$ and $Q' \leq^{FO}_{lin,+} Q$; similarly, strict lineage equivalence, $Q \equiv^{FO}_{lin} Q'$, means $Q \leq^{FO}_{lin} Q'$, $Q' \leq^{FO}_{lin} Q$.

The intuition is very simple. Suppose $Q \leq^{\text{FO}}_{\text{lin},+} Q'$. To compute the lineage Φ^D_Q, first compute $D' = \mathbf{F}(D) \cup \mathbf{E}(D)$, then compute the lineage $\Phi^{D'}_{Q'}$. This is a Boolean expression that uses the Boolean variables X_t occurring in D, and also extra variables $Y_{t'}$ for the extra tuples $t' \in \mathbf{E}(D)$: by setting all the latter variables $Y_{t'} = \texttt{true}$, the lineage $\Phi^{D'}_{Q'}$ becomes Φ^D_Q.

The lineage reduction $Q \leq^{\text{FO}}_{\text{lin},+} Q'$ is simpler than $Q \leq^{\text{FO}}_{\text{prob}} Q'$, because it gives us a very simple algorithm for computing $P(Q)$ using an oracle for $P(Q')$: simply set the probabilities of all extra tuples to 1, and obtain $P_D(Q) = P_{D'}(Q')$. We call the extra tuples $\mathbf{E}(D)$ *deterministic* tuples. For a simple example, if $h_0 = R(x), S(x, y), T(y)$ and $Q = R(x), S(x, y), T(y), U(x, y, z)$, then $h_0 \leq^{\text{FO}}_{\text{lin},+} Q$: the linear mapping $\mathbf{F}$ maps R, S, T to themselves, while $\mathbf{E}$ computes $U = (ADom)^3$. The lineage of Q has extra Boolean variables associated to the tuples in U, but once we set these to `true` its lineage becomes that of h_0. Thus, to compute $P(h_0)$

[2]The function $\mathbf{f}$ is obtained as follows. If $\mathbf{F}$ is a linear mapping, then by property (1), $\mathbf{F}(D) = \bigcup_{t \in D} \mathbf{F}(\{t\})$. Let $t' \in D' = \mathbf{F}(D)$. Then there exists t s.t. $t' \in \mathbf{F}(\{t\})$. We prove that t is unique, and set $\mathbf{f}(t') = t$. Indeed, if $t' \in \mathbf{F}(\{t_1\})$ for $t_1 \neq t$ then by property (2) $t' \in \mathbf{F}(\{t\}) \cap \mathbf{F}(\{t_1\}) = \mathbf{F}(\{t\} \cap \{t_1\}) = \mathbf{F}(\emptyset)$, and the latter is $\emptyset$ by property (3), which is a contradiction. It also follows that $\mathbf{f}$ is injective.

[3]By $\Phi[\mathbf{Y} = \textbf{true}]$ we mean the formula Φ where all variables $Y \in \mathbf{Y}$ are substituted with `true`.

using an oracle for $P(Q)$, simply create fake tuples for U and set their probabilities to 1. In general, we have:

PROPOSITION 2.8. *If $Q \leq^{FO}_{lin,+} Q'$ then $Q \leq^{FO}_{prob} Q'$.*

If no deterministic tuples are needed, then we call the reduction strict; in that case the lineages are equivalent, $\Phi^D_Q \equiv \Phi^{D'}_{Q'}$. Of course, every strict reduction is also a reduction, i.e. $Q \leq^{\text{FO}}_{\text{lin}} Q'$ implies $Q \leq^{\text{FO}}_{\text{lin},+} Q'$. Most lineage reductions in this paper are strict. For an example of a strict reduction, consider $Q = S(x, y)$ and $Q' = R(a, x, y)$ where a is a constant, then $Q \leq^{\text{FO}}_{\text{lin}} Q'$ because, given an instance S we can construct an instance R by padding all tuples in S with a: $R = \{(a, b, c) \mid (b, c) \in S\}$. The lineage of Q on S is the same as the lineage of Q' on R, and $P(Q) = P(Q')$.

2.5 Eliminating Constants through Query Shattering

We describe here a process to eliminate all constants from a query Q, without affecting its evaluation problem. Intuitively, if a is a constant in the query, then we replace a relational atom $R(x)$ with two atoms, $R(x), x = a$ and $R(x), x \neq a$; this process is called *shattering* [de Salvo Braz et al. 2005; Domingos and Lowd 2009]. We then replace the two predicates with new relation names, $R_a()$ and $R_*(x)$ respectively, thus eliminating the constant a. In general, for any query Q we construct another query Q', over a different vocabulary $\mathbf{R}'$, such that Q' has no constants and $Q \equiv^{\text{FO}}_{\text{lin}} Q'$. We call the transformation from Q to Q' *shattering*. We actually describe a more general shattering, which works only one way, $Q' \leq^{\text{FO}}_{\text{lin}} Q$, and which we need in our hardness proof; then we give a sufficient condition under which $Q' \equiv^{\text{FO}}_{\text{lin}} Q$.

Let A be a set of constants. Fix a regular vocabulary $\mathbf{R}$. A *shattered vocabulary*, $\mathbf{R}_A$, consists of relational symbols of the form R_τ, where $R \in \mathbf{R}$ is a relation name, and $\tau : [k] \to A \cup \{*\}$ is called its *shatter*, where $k = arity(R)$. The set of attributes is $Attr(R_\tau) = \{i \mid \tau(i) = *\}$; in other words, the arity of R_τ is equal to the number of $*$'s in the shatter τ. We make the assumption that relational instances D' over $\mathbf{R}_A$ do not contain any constants in A: $ADom(D') \cap A = \emptyset$.

Let Q be a query over the vocabulary $\mathbf{R}$, let A be any set of constants, and $\mathbf{R}_A$ any shattered vocabulary. We define the *shattered query*, Q_A as follows. For any function $\rho : Var(Q) \to A \cup \{*\}$ denote Q_ρ the query obtained by replacing each atom $R(\bar{x}) = R(x_1, \ldots, x_k)$ with $R_\tau((x_i)_{\rho(x_i)=*})$; it includes only those variables x_i of $R(\bar{x})$ whose shattering is $*$. The shattering τ of R_τ is defined as follows: for each $i = 1, k$, if x_i is a variable then $\tau(i) = \rho(x_i)$, and if x_i is a constant a then $\tau(i) = a$. If the symbol R_τ is not in the vocabulary $\mathbf{R}_A$ (for example, if x_i is a constant $\notin A$), then we define $R_\tau \equiv \texttt{false}$. Define the shattered query to be $Q_A = \bigvee_\rho Q_\rho$. Clearly, Q_A has no constants.

PROPOSITION 2.9. *If Q_A is the shattered query for Q over some shattered vocabulary $\mathbf{R}_A$, then $Q_A \leq^{FO}_{lin} Q$.*

PROOF. Define the following linear mapping $\mathbf{F}_1$ from $\mathbf{R}_A$ to $\mathbf{R}$. Given an instance D', for each shattered relational symbol R_τ and each tuple $t' \in R^{D'}_\tau$ define $\mathbf{F}_1(\{t'\}) = \{t\}$ where the tuple t obtained from t' by padding with the constants defined by τ. (For example, if $t' = (u_1, u_2)$ and $\tau = a * ba * a$, then

$t = (a, u_1, b, a, u_2, a)$.) The function $t' \mapsto t$ is injective, because we assumed that D' has no constants from A, and therefore the mapping $\mathbf{F}_1(D') = \bigcup_{t' \in D'} \mathbf{F}_1(\{t'\})$ is a linear mapping. Clearly, if $ADom'$ is the active domain of D', then the active domain of D is $ADom \subseteq ADom' \cup A$.

Next, we prove $\Phi_Q^D \equiv \Phi_{Q_A}^{D'}$, and for that we show that for every world $W' \subseteq D'$, $W' \models Q_A$ if and only if $W \models Q$ where $W = \mathbf{F}_1(W')$. Assuming $W \models Q$, there exists a valuation $\theta : Var(Q) \to ADom$ s.t. $W \models Q[\theta]$, where $Q[\theta]$ is a Boolean expression over ground tuples. Define the following variable shatter ρ. For each $x \in Var(Q)$, if $\theta(x) = a \in A$ then define $\rho(x) = a$, otherwise define $\rho(x) = *$. Since $Var(Q_\rho) \subseteq Var(Q)$, θ is also a valuation for Q_ρ, and $Q_\rho[\theta]$ is the same Boolean expression as $Q[\theta]$, where each ground tuple t has been replaced with $t' = \mathbf{f}_1(t)$. Therefore, $t \in W$ iff $t' \in \mathbf{F}_1(W)$, and $W \models Q[\theta]$ iff $W' \models \theta[Q_\rho]$. Since we assumed $W \models Q$, we have $W' \models Q_A$. Conversely, if $W' \models Q_A$, since $Q = \bigvee_\rho Q_\rho$ there exists a variable shattering ρ such that $W' \models Q_\rho$. Let $\theta' : Var(Q_\rho) \to ADom'$ such that $W' \models Q_\rho[\theta']$. Define $\theta : Var(Q) \to ADom$ as follows. For each $x \in Var(Q)$, if $\rho(x) = *$, then $x \in Var(Q_\rho)$ and therefore θ' is defined on x, and we set $\theta(x) = \theta'(x)$; if $\rho(x) = a \in A$, then define $\theta(x) = a$. It follows that $Q[\theta]$ is the same Boolean expression as $Q_\rho[\theta']$ up to the renaming of tuples t' to $\mathbf{f}(t')$, therefore $W' \models Q_\rho[\theta']$ implies $W \models Q[\theta]$. This completes the proof of $Q \leq_{\text{lin}}^{\text{FO}} Q_A$. □

Notice that the proposition holds for *any* shattered vocabulary $\mathbf{R}_A$. That is, we can choose $\mathbf{R}_A$ freely, all that is required is that Q_A be shattered according to the procedure we outlined here. We will use this throughout the paper for hardness proofs: to show that Q is hard, it suffices to find some shattering Q_A that is hard. However, for the algorithm, we need the converse of the proposition, and this holds only if $\mathbf{R}_A$ is "complete". For example, consider $Q = R(a), S(a) \vee R(x), T(x)$ and the shattered vocabulary $R_a(), R_*(x), S_a(), T_*(x)$: note that we did not include $T_a()$. The shattered query is $Q_a = R_a(), S_a() \vee R_*(x), T_*(x)$, and by the proposition we have $Q_a \leq_{\text{lin}}^{\text{FO}} Q$, but the converse fails, $Q \not\leq_{\text{lin}}^{\text{FO}} Q_a$. However, if we include $T_a()$ in the shattered vocabulary, then the shattered query is $Q'_a = R_a(), S_a() \vee R_a(), T_a() \vee R_*(x), T_*(x)$ and now $Q \equiv_{\text{lin}}^{\text{FO}} Q'_a$. We generalize this observation.

Let $Const(Q)$ denote the set of constants in Q, and $Const_Z(Q)$ the set of constants that occur on level Z: this definition is similar to $Var_Z(Q)$ given after Def. 2.3. For each level Z let A_Z^* be the following set. If $Var_Z(Q) = \emptyset$ then $A_Z^* = Const_Z(Q)$; otherwise, $A_Z^* = Const_Z(Q) \cup \{*\}$. The *completely shattered vocabulary* $\mathbf{R}_A$ is the following. For every relational symbol R, $\mathbf{R}_A$ contains all shatterings R_τ, for all $\tau \in \prod_i A_{Z_i}^*$, where Z_i is the level of the attribute (R, i). Thus, in the complete shattering each attribute (R, i) is shattered with all constants in $Const_{Z_i}(Q)$ and, if Z_i contains at least one variable, then the attribute is also shattered with $*$. The completely shattered query, Q_A, is the shattering of Q over the vocabulary $\mathbf{R}_A$.

PROPOSITION 2.10. *If Q_A is the complete shattering of Q then $Q \equiv_{lin}^{FO} Q_A$.*

PROOF. By Prop. 2.9 we have $Q_A \leq_{\text{lin}}^{\text{FO}} Q$, so we only need to prove $Q \leq_{\text{lin}}^{\text{FO}} Q_A$. Let D be a database instance over the vocabulary $\mathbf{R}$. We denote $A_Z = A_Z^* - \{*\}$ for any level Z. Define the mapping $\mathbf{F}_2$ as follows: $\mathbf{F}_2(D) = D'$, where each relation $R_\tau^{D'}$ is obtained from R^D in two steps. (1) Select from R^D only those tuples that

have the constants prescribed by τ. That is, for $i = 1, arity(R)$, if $\tau(i) = a \in A_{Z_i}$ then keep the tuple t only if $t_i = a$, and if $\tau(i) = *$ then keep t only if $t_i \notin A_{Z_i}$, where Z_i is the level of the attribute (R, i). (2) Project the resulting tuples on the attributes in $Attr(R_\tau)$. Denote the resulting database D'. Going in the other direction, denote $D_0 = \mathbf{F}_1(D')$, where $\mathbf{F}_1$ is the mapping defined in the proof of Prop. 2.9. We have already shown that $\Phi_{Q_A}^{D'} = \Phi_Q^{D_0}$. We claim that $\Phi_Q^{D_0} = \Phi_Q^D$. Note that $D_0 \subseteq D$. Let t be an omitted tuple, i.e. $t \in R^D$, but $t \notin R^{D_0}$. Then there exists an attribute i such that $t_i \notin A_{Z_i}$ and no shattering $R_\tau \in \mathbf{R}_A$ has $\tau(i) = *$. But that implies $* \notin A^*_{Z_i}$, which means that $Var_{Z_i}(Q) = \emptyset$. All atoms in Q that refer to R have some constant in $Const_{Z_i}(Q)$ on attribute position i, meaning that the tuples $t \in D - D_0$ do not occur in the lineage Φ_Q^D, proving that $\Phi_Q^{D_0} = \Phi_Q^D$. □

EXAMPLE 2.11. *We show the complete shattering of* $q = R(x, a), R(a, x) = \exists x.(R(x, a) \wedge R(a, x))$. *The shattered vocabulary is:* $R_{aa}(), R_{*a}(A), R_{a*}(B), R_{**}(A, B)$. *Given a probabilistic database* $\mathbf{D} = (D, P)$, *define* $\mathbf{D'} = (D', P')$ *by splitting* $R(A, B)$ *into four relations:* $R_{aa}^{D'} = \Pi_\emptyset(\sigma_{A=a,B=a}(R^D))$, $R_{*a}^{D'} = \Pi_A(\sigma_{A\neq a,B=a}(R^D))$, $R_{a*}^{D'} = \Pi_B(\sigma_{A=a,B\neq a}(R^D))$, $R_{**}^{D'} = \Pi_{AB}(\sigma_{A\neq a,B\neq a}(R^D))$. *Notice that* R_{aa} *is a relation of arity zero. For the shattered query* q_a *we note that there are two shatters for* x, *denoted* a *and* $*$. *Thus* $q_a = q_a \vee q_* = R_{aa}() \vee R_{*a}(x), R_{a*}(x)$.

2.6 Special Query Expressions

We introduce here five special query expressions.

Consider a conjunctive query in traditional notation $q = g_1, g_2, \ldots, g_m$, and define the following undirected graph: the nodes are the atoms g_i, and the edges are pairs (g_i, g_j) s.t. g_i, g_j share a common variable. The *components of the query* q are the connected components of this graph; q is called a *component* if it is connected, i.e. if it has a single component. We denote a component with c. For example, if $q = R(x), S(x, y), T(z)$ then it has two components: $c_1 = R(x), S(x, y)$ and $c_2 = T(z)$, and we write $q \equiv c_1 \wedge c_2$, or $q \equiv c_1, c_2$. Clearly, each conjunctive query q can be written uniquely as a conjunction of components $q = c_1, c_2, \ldots, c_k$.

A conjunctive-, disjunctive, DNF-, and CNF-query expression is defined as a Boolean expression of that particular kind, over components. Denote components with c, conjunctive queries with q, disjunctive queries with d, and both DNF and CNF expressions with Q. They are defined by the following grammar:

$c =$	connected conjunctive query — component
$q = c_1 \wedge \ldots \wedge c_k$	conjunctive query expression
$d = c_1 \vee \ldots \vee c_k$	disjunctive query expression
$Q = q_1 \vee \ldots \vee q_k$	DNF query expression
$Q = d_1 \wedge \ldots \wedge d_k$	CNF query expression

In addition, we will use comma instead of $\wedge$ inside a conjunctive query, and will drop existential quantifiers when clear from the context. Note that the existential quantifiers are always pushed down to the components.

The terms DNF or CNF are justified because these are indeed propositional DNF or CNF expressions, where the atoms are components. Clearly, every UCQ query is equivalent to some DNF expression: start from its traditional expression Eq. 4, and

write each conjunctive query as a conjunction of components. Every UCQ query is also equivalent to an CNF expression: simply convert the DNF to CNF.

EXAMPLE 2.12. *We illustrate these notations with the following query Q_V, which we first show in its traditional UCQ syntax:*

$$Q_V = R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(y_2) \vee R(x_3), T(y_3)$$

Note that $R(x_3), T(y_3)$ is not a component, since it is disconnected: $(\exists x_3.R(x_3)) \wedge (\exists y_3.T(y_3))$. Q_V is equivalent to the following DNF expression:

$$\begin{aligned} Q_V \equiv & \exists x_1.\exists y_1.[R(x_1) \wedge S(x_1, y_1)] \vee \exists x_2.\exists y_2.[S(x_2, y_2) \wedge T(y_2)] \vee [(\exists x_3.R(x_3)) \wedge (\exists y_3.T(y_3))] \\ = & c_1 \vee c_2 \vee (c_3 \wedge c_4) \end{aligned}$$

Next, let's rewrite it as a CNF query expressions:

$$\begin{aligned} Q_V \equiv & (c_1 \vee c_2 \vee c_3) \wedge (c_1 \vee c_2 \vee c_4) \\ = & [\exists x_1.\exists y_1.(R(x_1) \wedge S(x_1, y_1)) \vee \exists x_2.\exists y_2.(S(x_2, y_2) \wedge T(y_2)) \vee \exists x_3.R(x_3)] \\ & \wedge [\exists x'_1.\exists y'_1.(R(x'_1) \wedge S(x'_1, y'_1)) \vee \exists x'_2.\exists y'_2.(S(x'_2, y'_2) \wedge T(y'_2)) \vee \exists y'_3.T(y'_3)] \\ \equiv & [\exists x_3.R(x_3) \vee \exists x_2.\exists y_2.(S(x_2, y_2) \wedge T(y_2))] \wedge [\exists x'_1.\exists x'_2.(R(x'_1) \wedge S(x'_1, y'_1)) \vee \exists y'_3.T(y_3)] \\ \equiv & (R(x_3) \vee S(x_2, y_2), T(y_2)) \wedge (R(x'_1), S(x'_1, y'_1) \vee T(y'_3)) \end{aligned}$$

We minimized the two disjunctive queries on lines 2 and 3, then we dropped the quantifiers and replaced $\wedge$ with comma.

Finally, we consider the containment problem for these five special query expressions. Let Q, Q' be two query expressions, given as propositional Boolean expressions E, E' over components $c_1, c_2, \ldots$ If propositional logical implication $E \Rightarrow E'$ holds, then query containment $Q \Rightarrow Q'$ also holds, but the converse is not true in general. We give below necessary and sufficient conditions for containment.

PROPOSITION 2.13. *Assume all queries are without constants. Then:*

Component containment:	$c \Rightarrow c'$	*iff*	$\exists h : c' \to c \quad$ *iff* $\quad D_c \models c'$
Conjunctive query containment:	$(\bigwedge_i c_i) \Rightarrow (\bigwedge_j c'_j)$	*iff*	$\forall j.\exists i.c_i \Rightarrow c'_j$
Disjunctive query containment:	$(\bigvee_i c_i) \Rightarrow (\bigvee_j c'_j)$	*iff*	$\forall i.\exists j.c_i \Rightarrow c'_j$
DNF query containment:	$(\bigvee_i q_i) \Rightarrow (\bigvee_j q'_j)$	*iff*	$\forall i.\exists j.q_i \Rightarrow q'_j$
CNF query containment:	$(\bigwedge_i d_i) \Rightarrow (\bigwedge_j d'_j)$	*iff*	$\forall j.\exists i.d_i \Rightarrow d'_j$

PROOF. A component is a special case of a conjunctive query, hence the first containment follows from Chandra and Merlin' result [Chandra and Merlin 1977]. For the remainder four statements, the "if" direction is immediate, so we prove the "only if" direction. For conjunctive queries, $(\bigwedge_i c_i) \Rightarrow (\bigwedge_j c'_j)$ implies the existence of a homomorphism $f : (\bigwedge_j c'_j) \to (\bigwedge_i c_i)$. Since f must map each variable to a variable (there are no constants in c_i), for each connected component c'_j, its image $f(c'_j)$ is connected, hence it must be a subset of some connected component c_i, proving

Algorithm 1 Algorithm from [Dalvi and Suciu 2004; Dalvi and Suciu 2007b] computing $P(q)$ for a conjunctive query without self-joins

Write q as a conjunction of components $q = c_1, \ldots, c_m$.

/* q has 2 or more components */
if $m \geq 2$ **then return** $P(c_1) \cdot P(c_2) \cdots P(c_m)$ /* independent join */

/* q is a single component: denote it c */
if c has no variables **then return** $P(t)$ /* $c =$ a ground tuple t */
if c has a root variable z **then return** $1 - \prod_{a \in ADom}(1 - P(c[a/z]))$
/* independent project */
Otherwise **FAIL**

$c_i \Rightarrow c'_j$. The claims for disjunctive- and DNF-query expressions follow immediately from Sagiv and Yannakakis's criterion for UCQ [Sagiv and Yannakakis 1980]. It remains to prove the claim for CNF query expressions. Suppose the contrary: there exists an index j such that $\forall i$, $d_i \not\Rightarrow d'_j$. Thus, forall i, by Sagiv and Yannakakis' criterion we obtain that $\exists k_i$ s.t $c_{ik_i} \not\Rightarrow d'_j$, where c_{ik_i} is one of the components of the disjunctive query d_i. On the other hand, we have $\forall i$, $c_{ik_i} \Rightarrow d_i$, which implies that $\bigwedge_i c_{ik_i} \Rightarrow (\bigwedge_i d_i) \Rightarrow (\bigwedge_j d'_j) \Rightarrow d'_j$. The left side is a conjunctive query, the right side a disjunctive query. Applying Sagiv and Yannakakis' criteria a second time, we obtain a component c'_{jl} of d'_j s.t. $\bigwedge_i c_{ik_i} \Rightarrow c'_{jl}$. By the first item of the proposition (containment of conjunctive queries), there exists i s.t $c_{ik_i} \Rightarrow c'_{jl}$, contradicting the fact that $c_{ik_i} \not\Rightarrow d'_j$. □

The statements for conjunctive queries and CNF queries fails if the queries have constants: for example $(R(x, a)) \wedge (S(a, z)) \Rightarrow R(x, y), S(y, z)$ (where a is a constant), but neither $R(x, a) \not\Rightarrow R(x, y), S(y, z)$ nor $S(a, z) \not\Rightarrow R(x, y), S(y, z)$. This is one reason why we shatter queries.

Using the proposition, we will *minimize* a query expression in each of the five forms. A component is minimized using the standard procedure given by Chandra and Merlin [Chandra and Merlin 1977]. A conjunctive query $\bigwedge_i c_i$ is minimized by first minimizing each c_i, then removing every c_i for which there exists $j \neq i$ s.t. $c_j \Rightarrow c_i$. A disjunctive query $\bigvee_i c_i$ is minimized by first minimizing each c_i, then removing every c_i for which there exists $j \neq i$ s.t. $c_i \Rightarrow c_j$. A DNF query expression $\bigvee_i q_i$ is minimized by first minimizing each conjunctive query q_i then removing every q_i for which there exists $j \neq i$ s.t. $q_i \Rightarrow q_j$. Finally, a CNF query expression $\bigwedge_i d_i$ is minimized by first minimizing each disjunctive query d_i then removing every query d_i for which there exists $j \neq i$ s.t. $d_j \Rightarrow d_i$. In all five cases, the minimized query expression is unique up to isomorphism. We need minimized CNF expressions in Sect. 7 (in Lemma 7.9).

2.7 Review: Computing the Probability of a Conjunctive Query With No Self-Joins

A conjunctive query is said to be without *self-joins* if no two atoms have the same relational symbol. A simple algorithm for computing the probability of a conjunctive query without self-joins was first described in [Dalvi and Suciu 2004; Dalvi and Suciu 2007b]; we review it here briefly as algorithm 1. The algorithm

proceeds inductively on the structure on the query. If the query is disconnected, then it multiplies the probabilities of its components: this is correct because the query has no self-joins, and therefore distinct components use disjoint sets of relational symbols and are independent probabilistic events. If the query is a single ground tuple t then it looks up and returns the tuple's probability in the probabilistic database. If the query is connected and has variables, then it chooses a root variable z and substitutes it successively with all constants in the active domain: the resulting queries $c[a/z], a \in ADom$ are independent, and their probabilities are combined to compute $P(c)$. Finally, if the query has no root variable, then the algorithm fails. The two steps are called *independent join* and *independent project* respectively, because they can be implemented as a join, or as a project operator in Relational Algebra, over independent probabilistic events [Dalvi and Suciu 2004].

The algorithm was proven in [Dalvi and Suciu 2004; Dalvi and Suciu 2007b] to be complete for conjunctive queries without self-joins, in the following sense. For any query q, the algorithm either succeeds in computing the probability of q, or, if it fails, then the query is hard for $\#P$. For example, the query $h_0 = R(x), S(x,y), T(y)$ is connected and has no root variable: therefore it is hard for $\#P$ (we review the proof in Sect. 5). We briefly illustrate the algorithm on an example.

EXAMPLE 2.14. *Consider* $q = R(x), S(x,y) = \exists x.\,(R(x) \wedge \exists y.S(x,y))$. *algorithm 1 computes its probability as follows:*

$$\begin{aligned} P(q) &= 1 - \prod_{a \in ADom} (1 - P(R(a), \exists y.S(a,y))) \\ P(R(a), \exists y.S(a,y)) &= P(R(a)) \cdot P(\exists y.S(a,y)) \\ P(\exists y.S(a,y)) &= 1 - \prod_{b \in ADom} (1 - P(S(a,b))) \end{aligned}$$

However, the algorithm cannot be applied beyond conjunctive queries without self-joins, as the following example shows.

EXAMPLE 2.15. *Consider the query* $q_J = R(x_1), S(x_1,y_1), T(x_2), S(x_2,y_2)$, *which has two components, and can be written as* $q_J = c_1, c_2$, *where* $c_1 = R(x_1), S(x_1,y_1)$ *and* $c_2 = T(x_2), S(x_2,y_2)$. *We cannot multiply the probabilities of the two components, because they share the common symbol* S*: in general,* $P(q_J) \neq P(c_1)P(c_2)$. *For example, if the probabilistic database has three tuples* $\mathbf{D} = \{R(a), S(a,b), T(b)\}$ *with probabilities* p, s, t *respectively, then* $P(q_J) = p \cdot s \cdot t$, $P(c_1) = p \cdot s$, $P(c_2) = s \cdot t$, *and* $P(q) \neq P(c_1)P(c_2)$, *unless* $s = 0$ *or* $s = 1$.

3. A SIMPLE ALGORITHM

We start by describing a very simple algorithm for computing the probability of a UCQ query. In Sect. 4, we will make two small revisions that make the algorithm complete. We motivate the algorithm with an example:

EXAMPLE 3.1. *Continuing Example 2.15, we compute* $P(q_J)$ *using the inclusion-exclusion formula:*

$$P(c_1, c_2) = P(c_1) + P(c_2) - P(c_1 \vee c_2)$$

We have seen in Example 2.14 how to compute $P(c_1)$, and $P(c_2)$ is computed similarly. We show here how to compute $P(d)$, where:

$$\begin{aligned} d =& c_1 \vee c_2 = R(x_1), S(x_1, y_1) \vee T(x_2), S(x_2, y_2) \\ \equiv& \exists z.[(R(z) \wedge \exists y_1.S(z, y_1)) \vee (T(z) \wedge \exists y_2.S(z, y_2))] \\ \equiv& \bigvee_{a \in ADom} [(R(a) \wedge \exists y_1.S(a, y_1)) \vee (T(a) \wedge \exists y_2.S(a, y_2))] \end{aligned}$$

Rewrite the inner query as $R(a), \exists y_1.S(a, y_1) \vee T(a), \exists y_2.S(a, y_2) \equiv (R(a) \vee T(a)) \wedge (\exists y.S(a, y))$ and obtain:

$$P(d) = 1 - \prod_{a \in ADom} (1 - P((R(a) \vee T(a)) \wedge \exists y.S(a, y)))$$

We used the fact that the queries $(R(a) \vee T(a)) \wedge \exists y.S(a, y)$ for $a \in ADom$ are independent. Next, $P((R(a) \vee T(a)) \wedge \exists y.S(a, y)) = P((R(a) \vee T(a))) \times P(\exists y.S(a, y))$. Thus, $P(d)$ can be computed in PTIME in the size of the database.

The example illustrates an important point: in order to compute the probability of a conjunctive query with self-joins, we had to compute the probability of a disjunctive query $c_1 \vee c_2$ as an intermediate step. In other words, the class of conjunctive queries is not a "natural" class to study: the natural class is that of unions of conjunctive queries, UCQ.

To describe the algorithm we need two definitions. First:

Definition 3.2. *Let Q be a query expression. A variable z is called a* separator variable *if Q starts with $\exists z$, i.e. $Q = \exists z.Q_1$, for some query expression Q_1, and (a) z is a root variable (i.e. it appears in every atom), (b) for every relation symbol R, there exists an attribute (R, i_R) s.t. every atom with symbol R has z in position i_R.*

If Q has a separator variable, then it is equivalent to a disjunctive query d. Indeed, $Q = \exists z.Q_1$, and after expanding Q_1 in DNF we obtain:

$$\begin{aligned} Q \equiv& \exists z.(q_1 \vee \ldots \vee q_m) \equiv \exists z.q_1 \vee \exists z.q_2 \vee \ldots \vee \exists z.q_m \\ \equiv& \exists x_1.q_1[x_1/z] \vee \ldots \vee \exists x_m.q_m[x_m/z] = d \end{aligned}$$

Each conjunctive query q_i is connected, because z is a root variable and therefore $\bigvee q_i$ is a disjunctive query. (The substitution of z with fresh variables x_i was needed to ensure that all existential quantifiers use distinct variables.)

Conversely, with some abuse, we say that a disjunctive query d has a separator variable if we can revert the process above. If $d = \exists x_1.c_1 \vee \ldots \vee \exists x_m.c_m$, then we say that "$z = x_1 = \ldots = x_m$ is a separator variable", or that "$x_1, \ldots, x_m$ are separator variables" if z is a separator variable in the expression $\exists z.(c_1[z/x_1] \vee \ldots \vee c_m[z/x_m])$. For example, consider $d = R(x_1), S(x_1, y_1) \vee T(x_2), S(x_2, y_2)$ (Example 3.1): we say that it has the separator variable $z = x_1 = x_2$, by which we mean that we rewrite it as $\exists z.[R(z), \exists y_1.S(z, y_1) \vee T(z), \exists y_2.S(z, y_2)]$ and now z is formally a separator variable.

Clearly not every disjunctive query has a separator variable. A trivial example is $R(x), S(x, y), T(y)$, doesn't even have a root variable. More subtly, $R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(y_2)$ does not have a separator variable: rewriting it as $\exists z.(R(z), \exists y_1.S(z, y_1) \vee$

$T(z), \exists x_2.S(x_2, z))$ does not help because z occurs on the first position in $S(z, y_1)$ and on the second position in $S(x_2, z)$.

We have:

PROPOSITION 3.3. *Let d be a query expression with a separator variable z. Then the events $d[a/z]$, for $a \in ADom$ are independent probabilistic events. In particular:*

$$P(d) = 1 - \prod_{a \in ADom} (1 - P(d[a/z])) \tag{8}$$

PROOF. Let $a \neq b$ be two distinct constants in the active domain. Consider the lineage expressions of the queries $d[a/z]$ and $d[b/z]$. We claim that these two Boolean expressions do not share any common Boolean variables. Indeed, suppose they share X_t, where t is a ground tuple: $t = R(c_1, c_2, \ldots)$. Since z is a separator variable, every ground tuple of R occurring in the lineage of $d[a/z]$ has the constant a in position i_R: in other words, $c_{i_R} = a$. Reasoning similarly for $d[b/z]$, we also conclude that the ground tuple contains b on position i_R, i.e. $c_{i_R} = b$. This is a contradiction because $a \neq b$. □

The second definition is:

DEFINITION 3.4. *Let $\mathbb{Q} = \{Q_1, Q_2, \ldots, Q_k\}$ be a set of query expressions. The* co-occurrence graph *of $\mathbb{Q}$ is the following undirected graph: the nodes are $1, 2, \ldots, k$, and the edges are pairs (i, j) s.t. there exists a relational symbol that occurs both in Q_i and in Q_j. Let $K_1, \ldots, K_m$ be the connected components (they form a partition on $[k]$).*

Let $Q = d_1 \wedge \ldots \wedge d_k$ be a UCQ query written in CNF, and $K_1, \ldots, K_m$ the connected components for the set of queries $\{d_1, \ldots, d_k\}$. The *symbol-components* of Q are

$$Q_1 = \bigwedge_{i \in K_1} d_i, \qquad Q_2 = \bigwedge_{i \in K_2} d_i \qquad \ldots \qquad Q_m = \bigwedge_{i \in K_m} d_i$$

Then we have:

$$P(Q) = P(Q_1) \cdot P(Q_2) \cdots P(Q_m) \tag{9}$$

If $m = 1$, then we say that Q is *symbol-connected*.

Similarly, if $d = c_1 \vee \ldots \vee c_k$ is a disjunctive query, and $K_1, \ldots, K_m$ are connected components of the set of queries $\{c_1, \ldots, c_k\}$, then the *symbol-components* of d are:

$$d_1 = \bigvee_{i \in K_1} c_i, \qquad d_2 = \bigvee_{i \in K_2} c_i \qquad \ldots \qquad d_m = \bigvee_{i \in K_m} c_i$$

Then we have:

$$P(d) = 1 - (1 - P(d_1)) \cdot (1 - P(d_2)) \cdots (1 - P(d_m)) \tag{10}$$

If $m = 1$, then we say that d is *symbol-connected*.

The algorithm for computing the probability of a UCQ query is shown in algorithm 2. The input query Q is assumed to be without constants (it must be shattered first). The query is processed inductively. The algorithm goes through three syntactic expressions: CNF (line 1), disjunctive (line 10), and DNF (in line 17 $d[a/z]$ is a

Algorithm 2 Algorithm for Computing $P(Q)$
`Input:` A ranked UCQ Q; a Probabilistic database with active domain $ADom$
`Output:` $P(Q)$

1: Write Q in CNF and minimize it.
2: Compute the symbol-components: $Q = Q_1 \wedge \ldots \wedge Q_m$
3: **if** $m \geq 2$ **then return** $P(Q_1) \cdot P(Q_2) \cdots P(Q_m)$ /* independent join */
4:
5: Q is a symbol-connected CNF: $Q = d_1 \wedge \ldots \wedge d_k$
6: **if** $k \geq 2$ **then return** $-\sum_{s \subseteq [k], s \neq \emptyset} (-1)^{|s|} P(\bigvee_{i \in s} d_k)$ /*inclusion/exclusion*/
7: /* replaced with Mobius' inversion formula in Sect. 4.2: */
8: /* **return** $-\sum_{v < \hat{1}, \mu(v,\hat{1}) \neq 0} \mu_L(v, \hat{1}) P(d_v)$ */
9:
10: Q is a disjunctive query: minimize it and denote it d
11: Compute the symbol-components: $d = d_1 \vee \ldots \vee d_m$
12: **if** $m \geq 2$ **then return** $1 - (1 - P(d_1)) \cdots (1 - P(d_m))$ /* indep. union */
13:
14: d is symbol-connected, disjunctive query $d = c_1 \vee \ldots \vee c_k$
15: **if** d has no variables **then return** $P(t)$ /* $d = c_1 =$ ground tuple t */
16:
17: **if** d has a separator variable z **then return** $1 - \prod_{a \in ADom} (1 - P(d[a/z]))$
/* independent project */
18: Otherwise **FAIL**

DNF expression). Each recursive call is on a simpler subexpression, thus eventually the algorithm reaches ground tuples. There are four main steps: an independent join, the inclusion/exclusion formula, an independent union, and an independent project. Independent join and independent project generalize those of algorithm 1. In fact, if we apply algorithm 2 to a conjunctive queries without self-joins, it proceeds identically to algorithm 1.

The inclusion/exclusion formula in step 6 is the dual of the more familiar one, because it is applied to a conjunction, like:

$$P(d_1 \wedge d_1 \wedge d_3) =$$
$$P(d_1) + P(d_2) + P(d_3) - P(d_1 \vee d_2) - P(d_1 \vee d_3) - P(d_2 \vee d_3) + P(d_1 \vee d_2 \vee d_3)$$

This is the dual of the more familiar inclusion/exclusion formula:

$$P(d_1 \vee d_1 \vee d_3) =$$
$$P(d_1) + P(d_2) + P(d_3) - P(d_1 \wedge d_2) - P(d_1 \wedge d_3) - P(d_2 \wedge d_3) + P(d_1 \wedge d_2 \wedge d_3)$$

In step 17 we introduce a constant a: before proceeding, we do a complete shattering of $d[a/z]$ (Prop. 2.10). The shattered query may be a larger than $d[a/z]$, for an example, if $d = \exists x.[R(x,x) \vee \exists y.R(x,y), S(x,y)]$, then $d[a/x] = R(a,a) \vee R(a,y), S(a,y)$, and the shattered query is $R_{aa}() \vee R_{aa}(), S_{aa}() \vee R_{a*}(y), S_{a*}(y)$. Progress is still ensured by the fact that the maximum arity of any relational symbol decreases by 1. However, for practical purposes, shattering during query evaluation is undesirable. It is much better to first rank the query (as we explain in the next section), and then run the algorithm.

Note how the algorithm reaches the end of the recursion. If one of the c_i's in a disjunctive query $d = \bigvee_i c_i$ is a ground tuple, then it is a symbol-component by itself because, due to shattering, a "ground tuple" is a relational symbol of arity zero, $R()$, and cannot occur in any other component c_j, because that would make c_j disconnected. Thus, if c_i is a ground tuple $R()$, then it is a symbol-component. Since we process each symbol component separately in step 11, it means that when we reach c_i the query consists of exactly one ground tuple. At this point, the algorithm simply looks up its probability in the database.

Finally, if the algorithm reaches a disjunctive query without separator, then it fails.

EXAMPLE 3.5. *We illustrate the algorithm on the query described in Example 2.12,* $Q_V = R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(y_2) \vee R(x_3), T(y_3)$. *The algorithm first writes it in CNF,* $Q_V = d_1 \wedge d_2 = (R(x_3) \vee S(x_2, y_2), T(y_2)) \wedge (R(x'_1), S(x'_1, y'_1) \vee T(y'_3))$. *Here* d_1, d_2 *are disjunctive queries. The set* $\{d_1, d_2\}$ *is symbol-connected, so the algorithm applies the inclusion-exclusion formula:*

$$\begin{aligned} P(Q_V) =& P(d_1) + P(d_2) - P(d_1 \vee d_2) \\ =& P(R(x_3) \vee S(x_2, y_2), T(y_2)) + P(R(x'_1), S(x'_1, y'_1) \vee T(y'_3)) - P(R(x_3) \vee T(y'_3)) \end{aligned}$$

We discuss only the first disjunctive query, d_1, *the other two are similar. It has two symbol-components, hence* $P(d_1) = 1-(1-P(R(x_3)))(1-P(S(x_2, y_2), T(y_2)))$; *the last expression,* $P(S(x_2, y_2), T(y_2))$ *has separator variable* y_2, *and is actually similar to the query in Example 2.14.*

PROPOSITION 3.6. *Fix a query expression* Q *and let* k *be the largest arity of any relational symbol used by the vocabulary of* Q. *Then, on any input database* $\mathbf{D}$, *algorithm 2 runs in time* $O(n^k)$, *where* n *is the size of the active domain of* $\mathbf{D}$. *Furthermore, if the algorithm succeeds, then it computes correctly* $P_{\mathbf{D}}(Q)$.

PROOF. For the running time note that the only step that depends on the database is the independent-project (step 17) where $P(d[a/z])$ is computed n times, once for each a in the active domain. The running time $O(n^k)$ follows by induction on k, since each recursive call to $d[a/z]$ reduces k by one, because z occurs in all relational atoms. The independent join (step 3) is correct by Eq. 9; the inclusion/exclusion formula (step 6) is obviously correct; independent union (step 12) is correct by Eq. 10, and independent project (step 17) is correct by Eq. 8. □

4. THE MAIN RESULT

In this section we state the main result of the paper. Let's call a query Q *safe* if algorithm 2 succeeds in computing $P(Q)$; otherwise call it unsafe; for now this is an informal definition, since we have not yet finished describing the algorithm. Clearly, if Q is safe then Q is in PTIME: our result is that, if Q is unsafe, then it is #P-hard. This proves a dichotomy for UCQ queries, they are either safe and in PTIME, or unsafe and #P-hard. It also says that the algorithm is complete, in that it succeeds on all queries in $UCQ(P)$ (assuming $FP \neq \#P$).

Let's consider an unsafe query. Call it *immediately unsafe* if algorithm 2 fails immediately. In general, if a query is unsafe, then the algorithm performs a few steps, then gets stuck by reaching an immediately unsafe query.

algorithm 2 is not yet complete, for two reasons, each requiring a different fix. First, it may fail immediately on some query that is in PTIME: we fix this by *ranking* the query. Second it may start with a PTIME query, apply some steps, then reach a #P-hard query and get stuck: we fix this by replacing inclusion/exclusion with Mobius' inversion formula.

4.1 Ranking

Ranking is the process that splits an atomic predicate containing two variables x, y into three predicates, $x < y$, $x = y$, $x > y$. It is analogous to shattering, Sect. 2.5.

Consider this query $q = R(x, y), R(y, x)$; x is not a separator variable because it occurs on the first position in $R(x, y)$ and on the second position in $R(y, x)$. If we do an independent project on x we obtain an incorrect result, because the events $q[a/x]$, $a \in ADom$ are not independent (the lineages of $R(a_i, y), R(y, a_i)$ and $R(a_j, y), R(y, a_j)$ share in common the tuples $R(a_i, a_j)$ and $R(a_j, a_i)$). Similarly, y is not a separator variable. Thus, algorithm 2 fails immediately on q. But $q \in UCQ(P)$ by the following simple argument. Let X_{ij} denote the Boolean variable representing the $R(a_i, a_j)$. Then the lineage of Q is:

$$\Phi_q = \bigvee_{i,j}(X_{ij} \wedge X_{ji}) \equiv \bigvee_{i<j}(X_{ij} \wedge X_{ji}) \vee \bigvee X_{ii}$$

The last expression is a read-once Boolean expression[Gurvich 1977], meaning that each Boolean variable occurs only once, and its probability can be computed in linear time. Notice that here we have used an order on the underlying active domain, $a_1, a_2, \ldots, a_n$. We will assume from now on that the domain is ordered.

DEFINITION 4.1. *Let Q be a query expression. We say that Q is* ranked *if Q has no constants and there exists a partial order on its variables, $(Var(Q), \preceq)$, such that, for every atom $R(\bar{x})$ of arity $k \geq 2$ in Q, and for any two positions $1 \leq i < j \leq k$ if x, y are the variables occurring on positions i and j, then $x \prec y$.*

For example, if Q is ranked and has an atom of the form $R(\ldots x, \ldots y, \ldots)$ where x occurs before y, then it cannot have an atom $S(\ldots y, \ldots, x, \ldots)$. Our earlier query $R(x, y), R(y, x)$ is not ranked. Notice that in a ranked query no variable is repeated in the same atom (as in $R(\ldots x, \ldots x, \ldots)$).

We state some simple properties of ranked queries. If Q_1 and Q_2 are ranked, then so are $Q_1 \vee Q_2$ and $Q_1 \wedge Q_2$, because of our assumption that existential variables are distinct. Conversely if $Q_1 \vee Q_2$ or $Q_1 \wedge Q_2$ is ranked, then both Q_1 and Q_2 are ranked. Ranked queries are preserved under Boolean equivalences. For example, consider $Q_1 \vee (Q_2 \wedge Q_3) \equiv (Q_1 \vee Q_2) \wedge (Q_1 \vee Q_3)$: if the first query is ranked, then so is the second, and vice versa. Finally, ranked queries are also preserved by the following logical equivalence: $\exists x_1.Q_1 \vee \exists x_2.Q_2 \equiv \exists z.(Q_1[z/x_1] \vee Q_2[z/x_2])$.

By definition a ranked query is also shattered (has no constants). If d is ranked and z is a separator variable, then the complete shatter of $d[a/z]$ (see line 17 of the algorithm) is isomorphic to $d[a/z]$ and is obtained as follows. Let i_R be the unique position on which the separator variable occurs in atoms with relation symbol R. Then the shattered vocabulary contains a single shattered symbol for R, where attribute i_R shatters to a and all other attributes shatter to $*$. For an

example, referring to Example 3.1, $d[a/z] = R(a), S(a, y_1) \vee T(a), S(a, y_2)$ shatters to $R_a(), S_{a*}(y_1) \vee T_a(), S_{a*}(y_2)$. In general, the complete shattering of $d[a/z]$ is a query that is isomorphic to $d[a/z]$, where each relation symbol has an arity decreased by one. In this paper we will blur the distinction between the query $d[a/z]$ and its shattering. In practice, one need not do any runtime shattering in step 17 of the algorithm, instead this step can be implemented using a selection operator [Suciu et al. 2011].

We will show that each query Q can be ranked. When running algorithm 2, if we rank the query first, then all subqueries processed recursively by the algorithms are already ranked. In other words, ranking must be done only once. Ranking is somewhat similar to shattering Prop. 2.10.

PROPOSITION 4.2. *Let Q be a query expression over a vocabulary $\mathbf{R}$. Then there exists a ranked query expression Q' over some vocabulary $\mathbf{R}'$ such that $Q \equiv^{FO}_{lin} Q'$.*

Thus, by Prop. 2.8, $Q \equiv^{\text{FO}}_{\text{prob}} Q'$ and the queries Q, Q' have the same complexity.

PROOF. First, shatter the query (Prop. 2.10) to eliminate constants, so we assume Q has no constants. Let R_i be a relational symbol, and let its arity be k. For each $m \leq k$, a *ranking of arity* m for R_i is a surjective function $\tau : [k] \to [m]$. For each $R_i \in \mathbf{R}$, the new vocabulary $\mathbf{R}'$ has a symbol $R_{i,\tau}$ of arity m for every ranking τ of arity m for R_i, and for every $m = 1, k$.

We assume an ordered domain. A k-tuples $\bar{a} = (a_1, \ldots, a_k)$ is a function $\bar{a} : [k] \to A$. We say that $\bar{a}$ is strictly ordered if $a_1 < a_2 < \ldots < a_k$. Each tuple $\bar{a}$ can be uniquely written as $\bar{a} = \bar{a}^< \circ \tau[\bar{a}]$, where $\tau[\bar{a}] : [k] \to [m]$ and $\bar{a}^< : [m] \to A$ is strictly ordered: namely, m is the number of distinct values in $\bar{a}$, $\bar{a}^<$ lists all distinct values in increasing order,, and $\tau[\bar{a}]$ returns the rank of each value. If $f : A \to B$ is a strictly monotone function, then $(f \circ \bar{a})^< = f \circ \bar{a}^<$ and $\tau[f \circ \bar{a}] = \tau[\bar{a}]$

We define now the query Q'. Assume w.l.o.g. that Q is in prenex normal form, i.e. Q has the form $\exists x_1 \ldots \exists x_n.E$, where E is a quantifier-free expression and $X = \{x_1, \ldots, x_n\}$ is the set of variables. Fix a total order on X, $x_1 < \ldots < x_n$, and denote $X_p = \{x_1, \ldots, x_p\}$ for $p \leq n$. A *variable ranking* is a surjective function $\rho : X \to X_p$, for some $p \leq n$. Given ρ, we define the following ranked query: $Q_\rho = \exists x_1 \ldots \exists x_p.E_\rho$, where the expression E_ρ is obtained from E by replacing each atom $R_i(\bar{x})$ with $R_{i,\tau[\bar{y}]}(\bar{y}^<)$, where $\bar{y} = \rho \circ x$; notice that the variables in the new atom are strictly ordered, hence the query Q_ρ is ranked. Define $Q' = \bigvee_\rho Q_\rho$.

We define now the linear mapping $D' = \mathbf{F}(D)$, which maps a database D over the vocabulary $\mathbf{R}$ into a database over the ranked vocabulary $\mathbf{R}'$. For each relation symbol $R_i \in \mathbf{R}$, and each ranking τ of arity m, define $R^{D'}_{i,\tau} = \{\bar{a}^< \mid \bar{a} \in R^D : \tau[\bar{a}] = \tau\}$; it consists of all tuples whose ranking is τ. Since each tuple $\bar{a} \in R^D_i$ has a unique ranking $\tau = \tau[\bar{a}]$, it will be mapped to a unique output relation $R^{D'}_{i,\tau}$, hence $\mathbf{F}$ is both linear and invertible (i.e. we can compute D from D').

We prove that $\Phi^D_Q \equiv \Phi^{D'}_{Q'}$, which completes the proof of the proposition. Let $W \subseteq D$ and let $W' = \mathbf{F}(W)$. Any valuation $\theta : X \to ADom$ can be uniquely written as $\theta = \theta' \circ \rho$, where $\rho : X \to X_p$ and $\theta' : X_p \to ADom$ is strictly monotone. For any atom $g = R(\bar{x})$ in Q, denote t its image under θ; further, let its corresponding ranked atom be $g' = R_{i,\tau}((\rho \circ x)^<)$ where $\tau = \tau[\rho \circ \bar{x}]$ and let t' be its image under θ'. We claim that $t \in W$ iff $t' \in W'$. First, we show that the claim proves: $W \models Q$

iff $W' \models Q'$. Indeed, if $W \models Q$ then there exists a valuation $\theta : X \to ADom$ s.t. $W \models Q[\theta]$, and therefore $W' \models Q_\rho[\theta']$ because $Q_\rho[\theta']$ is a Boolean expression over ground tuples that is essentially the same as $Q[\theta]$, and, conversely, if $W' \models Q_\rho$ for some ρ, then there exists a valuation $\theta' : X_p \to ADom$ s.t. $W' \models Q_\rho[\theta']$, which implies $W \models Q[\theta' \circ \rho]$ by the same argument. We now prove the claim. Notice that the image of $R(\bar{x})$ under θ is $\theta \circ \bar{x}$. If it belongs to R_i^W, then by definition of $\mathbf{F}$, $(\theta \circ \bar{x})^< \in R_{i,\tau}^{W'}$, (same τ because $\tau[\rho \circ \bar{x}] = \tau[\theta' \circ \rho \circ \bar{x}] = \tau[\theta \circ \bar{x}]$), and the claim follows from the fact that this is the same as the image of $R_{i,\tau}((\rho \circ x)^<)$ under θ' because $(\theta \circ \bar{x})^< = \theta' \circ (\rho \circ x)^<$. □

EXAMPLE 4.3. *We illustrate with the query $q = R(x,y), R(y,x)$. There are three possible rankings for the relation $R(A,B)$, denoted τ_1, τ_2, τ_3, which represent the following predicates: $A < B$; $A > B$; and $A = B$. We show now the transformed database. Given an instance R^D, we create three relations, $R_1^{D'} = \sigma_{A<B}(R^D)$; $R_2^{D'} = \Pi_{BA}(\sigma_{A>B}(R^D))$; and $R_3^{D'} = \Pi_A(\sigma_{A=B}(R^D))$. To construct the new query q', we note that there are three variable rankings: $\rho_1(x,y) = (x,y)$, $\rho_2(x,y) = (x,x)$, $\rho_3(y,x) = (x,y)$, leading to three queries, $q_1 = R_1(x,y), R_2(x,y)$, $q_2 = R_3(x), R_3(x)$, and $q_3 = R_2(x,y), R_1(x,y)$. After taking their union and minimizing, we obtain the following ranked query: $Q' \equiv [R_1(x,y), R_2(x,y)] \vee R_3(z)$. When evaluated by algorithm 2, the query will be processed as an independent union (since there are two symbol-components), and each subquery is safe.*

Thus, we update algorithm 2 to first rank the query. Ranking needs to be done only on the input query: all sub-queries processed recursively are already ranked.

Assuming the query is ranked, we can now prove that if it is immediately unsafe, then it is #P-hard. To define formally "immediately unsafe", let's see how the algorithm can get stuck immediately. Of course, the query is ranked. It must also be a disjunctive query: otherwise, it has the form $Q_1 \wedge Q_2$ and we can apply the inclusion-exclusion formula. It must be symbol-connected, otherwise we apply an independent union. It must have variables, otherwise it is a ground tuple. It must have no separator variable. And, finally, it must be minimized[4].

DEFINITION 4.4. *An* immediately unsafe query *is a ranked, disjunctive, symbol-connected query that has variables, but no separator variable.*

THEOREM 4.5 MAIN HARDNESS THEOREM. *If d is immediately unsafe then computing $P(d)$ is #P-hard.*

This is the most difficult step of the dichotomy result, and we will present the proof starting with Sect. 5. Here we illustrate it with several queries that will be important further in the paper:

[4]The only step in the algorithm where minimizing the query is necessary is right before checking for a the existence of a separator, line 17. If we fail to minimize d then may not have a separator, although its minimized expression does. For example $R(x), S(x,y), T(y) \vee R(z)$, which minimizes to $R(z)$. Interestingly, minimization is not required before the Mobius step (line 7): one can prove that, for any two equivalent CNF expressions $Q \equiv Q'$, the set of lattice elements that have $\mu(u, \hat{1}) \neq 0$ is the same in $L(Q)$ and in $L(Q')$.

EXAMPLE 4.6. *The following queries are immediately unsafe, and, thus, #P-hard.*

$$h_0 = R(x_0), S_1(x_0, y_0), T(y_0)$$
$$h_1 = R(x_0), S_1(x_0, y_0) \vee S_1(x_1, y_1), T(y_1)$$
$$h_2 = R(x_0), S_1(x_0, y_0) \vee S_1(x_1, y_1), S_2(x_1, y_1) \vee S_2(x_2, y_2), T(y_2)$$
$$h_3 = R(x_0), S_1(x_0, y_0) \vee S_1(x_1, y_1), S_2(x_1, y_1) \vee S_2(x_2, y_2), S_3(x_2, y_2) \vee S_3(x_3, y_3), T(y_3)$$
$$\dots$$
$$h_k = R(x_0), S_1(x_0, y_0) \vee S_1(x_1, y_1), S_2(x_1, y_1) \ \vee \dots \vee S_k(x_k, y_k), T(y_k)$$

Note that these examples are tight. If we remove any one component from h_k, then the resulting query becomes symbol-disconnected, and it is safe.

4.2 Mobius Inversion Formula

The second problem in algorithm 2 is that we may start with a query in $UCQ(P)$, but after some steps the algorithm reaches a #P-hard query and gets stuck. We illustrate this with an example.

EXAMPLE 4.7. *Consider the CNF query expression $Q_W = d_1 \wedge d_2 \wedge d_3$ where:*

$$d_1 = R(x_1), S_1(x_1, y_1) \vee S_2(x_2, y_2), S_3(x_2, y_2)$$
$$d_2 = R(x_3), S_1(x_3, y_3) \vee S_3(x_4, y_4), T(y_4)$$
$$d_3 = S_1(x_5, y_5), S_2(x_5, y_5) \vee S_3(x_6, y_6), T(y_6)$$

(We have taken the components of h_3 in Example 4.6 and arranged them in some way.) The inclusion/exclusion formula applied to Q_W has seven terms. Denote:

$$d_{12} = d_1 \vee d_2 = R(x_1), S_1(x_1, y_1) \vee S_2(x_2, y_2), S_3(x_2, y_2) \vee S_3(x_4, y_4), T(y_4)$$
$$d_{23} = d_2 \vee d_3 = R(x_3), S_1(x_3, y_3) \vee S_1(x_5, y_5), S_2(x_5, y_5) \vee S_3(x_4, y_4), T(y_4)$$
$$d_{123} = d_1 \vee d_3 = d_1 \vee d_2 \vee d_3 \equiv h_3$$

Then:

$$P(Q_W) =$$
$$= P(d_1) + P(d_2) + P(d_3) - P(d_1 \vee d_2) - P(d_1 \vee d_3) - P(d_2 \vee d_3) + P(d_1 \vee d_2 \vee d_3)$$
$$= P(d_1) + P(d_2) + P(d_3) - P(d_{12}) - P(d_{23})$$

On the first line we have $P(h_3)$ occurring twice, because both $d_1 \vee d_3$ and $d_1 \vee d_2 \vee d_3$ are equivalent to h_3. Therefore, if we allow the algorithm to proceed recursively then it will eventually fail on h_3. But the two occurrences of h_3 cancel out, and the expression simplifies to the second line. All five remaining queries are safe. For example, d_{12} has two symbol-components, RS_1 and $S_2S_3 \vee S_3T$, and each is safe. This proves that Q_W is in $UCQ(P)$.

We fix algorithm 2 by replacing the inclusion-exclusion formula with Mobius' inversion formula in a lattice. We review here the basic definitions, following Stanley [Stanley 1997]. A finite *lattice* is a finite ordered set $(L, \leq)$ where every two elements $u, v \in L$ have a greatest lower bound $u \triangle v$ and a least upper bound $u \triangledown v$, called *meet* and *join*. (The standard notations are $\wedge$ and $\vee$, but we reserve these

for logical OR and AND operations on queries.) Since the lattice is finite, it has a minimum and a maximum element, denoted $\hat{0}, \hat{1}$. We say that v *covers* u if $u < v$ and there is no w such that $u < w < v$. The *atoms* are all elements that cover $\hat{0}$; the *co-atoms* are all elements covered by $\hat{1}$; u is called *atomic* if it is the join of atoms; u is called *co-atomic* if it is the meet of co-atoms; the entire lattice is atomic (co-atomic) if all elements are atomic (co-atomic).

The Mobius function[5] is the function $\mu_L : L \times L \rightarrow \mathbf{Z}$ defined by:

$$\begin{aligned}\mu_L(u,u) =& 1\\ \mu_L(u,v) =& -\sum_{w:u<w\leq v} \mu_L(w,v)\end{aligned}$$

We drop the subscript and write μ when L is clear from the context. Note that, whenever $u \not\leq v$, $\mu(u,v) = 0$.

The Mobius function has the following property. Let $f : L \rightarrow \mathbf{R}$ be any real function defined on the lattice. Define a new function g as $g(v) = \sum_{u\leq v} f(u)$. Then, one can recover f from g by:

$$f(v) = \sum_{u\leq v} \mu(u,v)g(u) \tag{11}$$

Let $Q = d_1 \wedge \ldots \wedge d_k$ be a query in CNF, where $k \geq 1$. For any set $s \subseteq [k]$, define $\bar{s} = \{i \mid d_i \Rightarrow \bigvee_{j\in s} d_j\}$. The following three properties follow immediately: $s \subseteq s' \Rightarrow \bar{s} \subseteq \bar{s}'$; $s \subseteq \bar{s}$; $\bar{\bar{s}} = \bar{s}$. Hence, $s \mapsto \bar{s}$ is a closure operator. A set s is *closed* if $\bar{s} = s$. Denote $L(Q)$ the set of closed sets.

Definition 4.8. *Given a UCQ query Q, its* CNF lattice *is $(L(Q), \leq)$, where $L(Q)$ consists of all closed subsets of $[k]$, and $u \leq v$ if $u \supseteq v$.*

The CNF lattice has the following properties:

—Each element $u \in L(Q)$, $u \neq \hat{1}$, represents a disjunctive query $d_u = \bigvee_{i\in u} d_i$.

—For all $u, v \in L$, $u \leq v$ iff $d_u \Leftarrow d_v$; in other words, the lattice order corresponds to reverse implication (reverse query containment). Hence, $d_u \equiv d_v$ iff $u = v$.

—For all $u, v \in L(Q)$, $d_{u\triangle v} = d_u \vee d_v$. In other words, lattice-meet (denoted $\triangle$ and traditionally written $\wedge$) corresponds to query-or ($\vee$).

—$Q \equiv \bigwedge_{u\in L, u\neq\hat{1}} d_u$. Notice that $\wedge$ is query-and (not the lattice meet!).

—If the CNF query expression $Q = d_1 \wedge \ldots \wedge d_k$ is minimized (see the discussion at the end of Sect. 2.6), then each singleton set $\{1\}, \ldots, \{k\}$ is closed, and is a co-atom in $L(Q)$. Indeed, if $j \in \overline{\{i\}}$ then $d_j \Rightarrow d_i$: since Q is minimized, it follows that $j = i$, hence $\{i\}$ is closed. To see that it is a co-atom, suppose $\{i\} \leq u$: then $d_u \Rightarrow d_i$, and by Prop. 2.13 $\forall j \in u$, $d_j \Rightarrow d_i$, which implies $u = \{i\}$.

—If Q is minimized, then the lattice is co-atomic. Indeed, for any $u \in L$ we have $u = \triangle_{i\in u}\{i\}$.

[5]The standard definition of the Mobius function is on a poset, i.e. one does not need it to be a lattice. But in this paper we are only interested in the Mobius function on lattices.

PROPOSITION 4.9 MOBIUS INVERSION FORMULA FOR QUERIES IN CNF. *Let Q be a UCQ, with CNF lattice $(L(Q), \leq)$. Then:*

$$P(Q) = -\sum_{v<\hat{1}} \mu_L(v, \hat{1}) P(d_v) \tag{12}$$

PROOF. Denote $R = \neg Q$, and, for every $v \in L$, denote $e_v = \neg d_v$. The following three properties hold: (a) $R \equiv e_{\hat{1}} \equiv \bigvee_{v \in L(Q), v \neq \hat{1}} e_v$; (b) $u \leq v$ iff $e_u \Rightarrow e_v$; (c) for all $u, v \in L(Q)$, $e_{u \triangle v} = e_u \wedge e_v$. We first prove:

$$P(R) = -\sum_{v<\hat{1}} \mu_L(v, \hat{1}) P(e_v) \tag{13}$$

Here we follow [Stanley 1997]. For all $u \in L(Q)$, denote $f(u) = P(e_u \wedge \neg(\bigvee_{v<u} e_v))$. Then:

$$P(e_u) = \sum_{v \leq u} f(v) \qquad \Rightarrow \qquad f(u) = \sum_{v \leq u} \mu_L(v, u) P(e_v)$$

The claim Eq. 13 follows by setting $u = \hat{1}$ and noting $f(\hat{1}) = 0$. From here, Eq. 12 follows from the following three observations: $P(Q) = 1 - P(R)$; for all $v \in L(Q)$, $P(d_v) = 1 - P(e_v)$; and $\sum_{v \in L} \mu(v, \hat{1}) = 0$. □

The second update to algorithm 2 is to replace the inclusion-exclusion formula (line 6) with Mobius' inversion formula (line 7). Notice that in the Mobius' inversion formula we have explicitly avoided the lattice where $\mu(v, \hat{1}) = 0$: this minor detail is crucial for the algorithm to be complete.

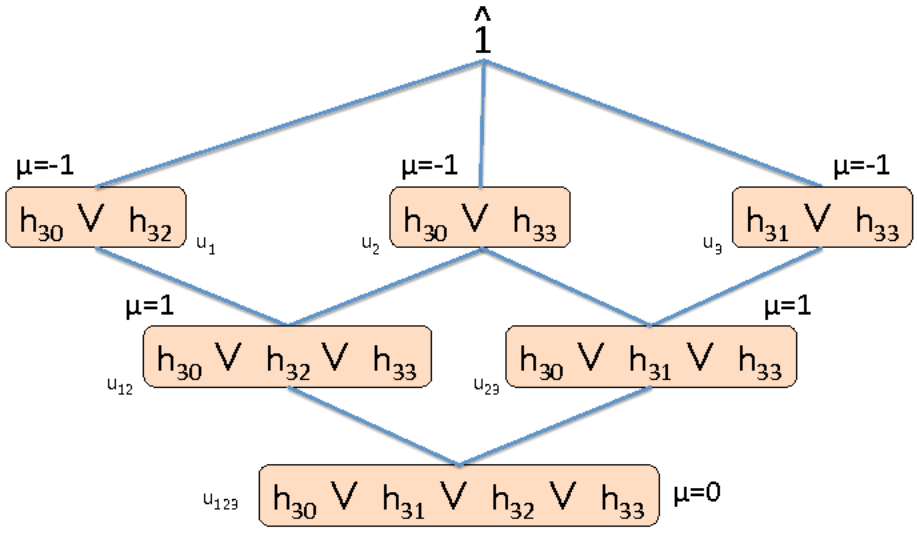
EXAMPLE 4.10. *Consider Q_W from Example 4.7. Write it as $Q_W = (h_{30} \vee h_{32}) \wedge (h_{30} \vee h_{33}) \wedge (h_{31} \vee h_{33})$, where:*

$$h_{30} = R(x_0), S_1(x_0, y_0) \qquad h_{31} = S_1(x_1, y_1), S_2(x_1, y_1)$$
$$h_{32} = S_2(x_2, y_2), S_3(x_2, y_2) \qquad h_{33} = S_3(x_3, y_3), T(y_3)$$

Figure 1 shows the CNF lattice for Q_W. The least element is the #P-hard query $h_3 \equiv h_{30} \vee h_{31} \vee h_{32} \vee h_{33} \equiv h_3$ from Example 4.6; the other five queries are safe. The figure shows the Mobius function $\mu(v, \hat{1})$ at each lattice element v: it is 0 for the least element, and $+1$ or -1 for all other elements. That means that the modified algorithm will make recursive calls only for the five save queries, and not for the unsafe query h_3.

This completes our two updates to the algorithms. One question is whether Mobius inversion formula, or even the inclusion-exclusion formula is an overkill for computing query probabilities. Established algorithms in probabilistic inference do not use the inclusion-exclusion formula, but instead use independence, conditional independence, Shannon's expansion formula, and disjoint-OR [Darwiche and Marquis 2002; Darwiche 2009]. Could one perhaps design a complete algorithm for UCQ's by using only those primitives? The following theorem highlights the difficulties.

THEOREM 4.11 REPRESENTATION THEOREM. *Let L be any finite lattice. Then there exists a UCQ query Q such that (a) its CNF lattice $L(Q)$ is isomorphic to L, (b) the query $d_{\hat{0}}$ is #P-hard, and (c) for all $u \in L(Q)$, $u \neq \hat{0}, \neq \hat{1}$, d_u is in $UCQ(P)$.*

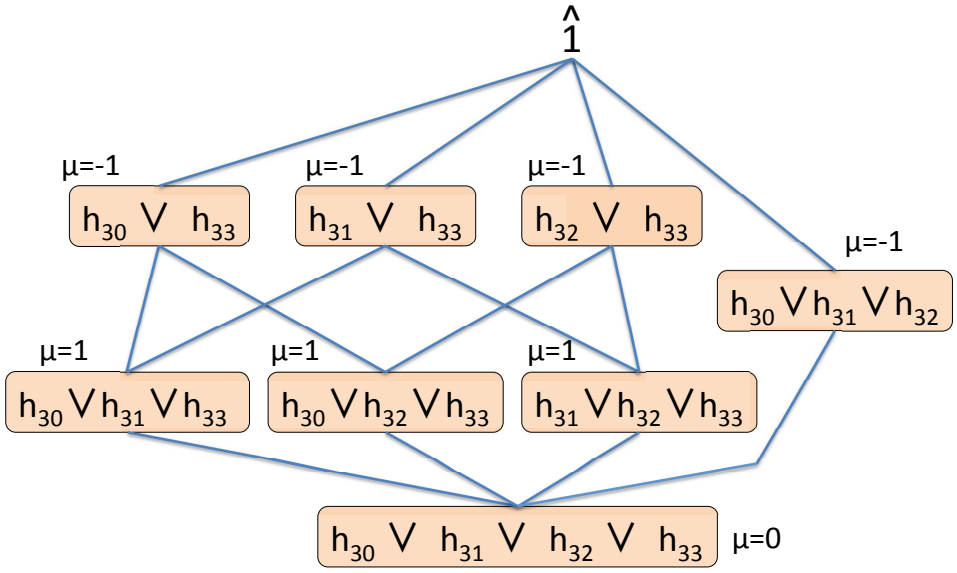


$$Q_W = (h_{30} \vee h_{32}) \wedge (h_{30} \vee h_{33}) \wedge (h_{31} \vee h_{33})$$

Fig. 1. The CNF Lattice for Q_W; see Example 4.10.

It follows that Q is in $UCQ(P)$ iff $\mu_L(\hat{0}, \hat{1}) = 0$; if an algorithm determines whether Q is in PTIME, it also determines whether $\mu_L(\hat{0}, \hat{1}) = 0$. In other words, any complete algorithm must distinguish between the cases $\mu = 0$ and $\mu \neq 0$ for an arbitrary lattice, and this seems difficult to achieve with probabilistic inference primitives other than inclusion/exclusion. The theorem also offers a recipe for constructing difficult benchmarks for PTIME algorithms: choose a complex lattice with $\mu(\hat{0}, \hat{1}) = 0$, associate the query Q and see if the algorithm can compute $P(Q)$ in PTIME.

Proof. Call an element $r \in L$ *join irreducible* if $r \neq \hat{0}$ and whenever $v_1 \bigtriangledown v_2 = r$, then either $v_1 = r$ or $v_2 = r$. (Recall that $\bigtriangledown$ is join: traditionally written $\vee$ in lattice theory.) All atoms are join irreducible, but a join irreducible is not necessarily an atom. Let $R = \{r_0, r_1, \ldots, r_k\}$ be all join irreducible elements in L; then $\forall u \in L$, $u = \bigtriangledown\{r \mid r \in R, r \leq u\}$. For every $u \in L$ denote $R_u = \{r \mid r \in R, r \not\leq u\}$. We start by proving that the mapping $u \mapsto R_u$ from L to subsets of R is strictly anti-monotone. Obviously, if $u \leq v$ then $R_u \supseteq R_v$. For the converse, assume $R_u \supseteq R_v$. By taking the complements of these two sets w.r.t. R we obtain $\{r \mid r \in R, r \leq u\} \subseteq \{r \mid r \in R, r \leq v\}$, which implies $u = \bigtriangledown\{r \mid r \in R, r \leq u\} \leq \bigtriangledown\{r \mid r \in R, r \leq v\} = v$. It follows that the mapping $u \mapsto R_u$ is strictly antimonotone,



$$Q_9 = (h_{30} \vee h_{33}) \wedge (h_{31} \vee h_{33}) \wedge (h_{32} \vee h_{33}) \wedge (h_{30} \vee h_{31} \vee h_{32})$$

Fig. 2. The CNF Lattice for Q_9; see Example 4.10.

and, hence, injective. In particular $R_u = R$ iff $u = \hat{0}$.

Next, we prove that $R_{u \triangle v} = R_u \cup R_v$ (recall that $\triangle$ means meet). By anti-monotonicity we have $R_{u \triangle v} \supseteq R_u$, and $R_{u \triangle v} \supseteq R_v$. Suppose that $r \in R_{u \triangle v}$, we show that $r \in R_u$ or $r \in R_v$ by contradiction: if $r \notin R_u$ then $r \leq u$, and similarly $r \leq v$, which implies $r \leq u \triangle v$, contradicting the fact that $r \in R_{u \triangle v}$. Therefore $R_{u \triangle v} = R_u \cup R_v$.

Define the following $k+1$ components:

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

Fix an order $r_0, r_1, \ldots, r_k$ of the elements in R. For every $u \in L$ denote $d_u = \bigvee_{r_i \in R_u} h_{ki}$ and define $Q = \bigwedge_{u < \hat{1}} d_u$. We claim that the CNF lattice of Q, $L(Q)$, is isomorphic to L. Indeed, by definition $L(Q)$ consists of all queries $d_s = \bigvee_{u \in s} d_u$, for all sets $s \subseteq L - \{\hat{1}\}$, up to logical equivalence. We show that the mapping

$u \mapsto d_{\{u\}} (= d_u)$ is an isomorphism $L \to L(Q)$. First, it is injective, because different elements $u \neq v$ correspond to inequivalent queries $d_u \not\equiv d_v$ (since $R_u \neq R_v$). Second it is surjective. Indeed, let $s = \{u_1, \ldots, u_m\}$ be a set, s.t. $u_i \in L - \{\hat{1}\}$, for $i = 1, m$, and denote $v = u_1 \triangle \ldots \triangle u_m$. Then:

$$d_s = d_{u_1} \vee \ldots \vee d_{u_k} = \bigvee_{r_i \in R_{u_1}} h_{ki} \vee \ldots \vee \bigvee_{r_i \in R_{u_m}} h_{ki} = \bigvee_{r_i \in R_{u_1} \cup \ldots \cup R_{u_m}} h_{ki} = \bigvee_{r_i \in R_v} h_{ki} = d_v$$

Hence d_s is the image of v under the mapping $u \mapsto d_u$. Thus, L is isomorphic to $L(Q)$.

Finally, the query $d_{\hat{0}} = \bigvee_{i=0,k} h_{ki} = h_k$ is #P-hard, as explained in Example 4.6. This completes the proof. □

EXAMPLE 4.12. *We illustrate first with the lattice* $L = \{\hat{1}, u_1, u_2, u_3, u_{12}, u_{23}, u_{123}\}$ *in Figure 1. There are four join-irreducible elements,* $R = \{u_1, u_{12}, u_{23}, u_3\}$*, thus:*

$$R_{u_1} = \{u_{23}, u_3\} \qquad R_{u_2} = \{u_1, u_3\} \qquad R_{u_3} = \{u_1, u_{12}\}$$
$$R_{u_{12}} = \{u_1, u_{23}, u_3\} \qquad R_{u_{23}} = \{u_1, u_{12}, u_3\} \qquad R_{u_{123}} = \{u_1, u_{12}, u_{23}, u_3\}$$

Let us associate to the four elements in R *the components* $h_{33}, h_{31}, h_{32}, h_{30}$ *in this order, respectively. Then we obtain the following query:*

$$Q = (h_{32} \vee h_{30}) \wedge (h_{33} \vee h_{30}) \wedge (h_{33} \vee h_{31}) \wedge$$
$$(h_{33} \vee h_{32} \vee h_{30}) \wedge (h_{33} \vee h_{31} \vee h_{30}) \wedge (h_{33} \vee h_{31} \vee h_{32} \vee h_{30})$$

which, after minimization, becomes Q_W*. Notice that if we chose a different mapping from* R *to the four components of* h_3*, then we obtain a different query. For example, if we map them to* $h_{30}, h_{31}, h_{32}, h_{33}$ *then, after minimizing we obtain* $Q = (h_{32} \vee h_{33}) \wedge (h_{30} \vee h_{33}) \wedge (h_{30} \vee h_{31})$*.*

As a final example, we show a more complex lattice in Figure 2, which, through the same process, generates a query called Q_9*. This query too is in UCQ*(P)*, because the only #P-hard query is at the bottom of the lattice, where* $\mu(\hat{0}, \hat{1}) = 0$*. These are illustrations of how the theorem can be used to generate difficult benchmark queries, for computing* $P(Q)$ *in PTIME. To the best of our knowledge, algorithm 2 is the only algorithm known to date that can compute* $P(Q_9)$ *in polynomial time in the size of the input database, see [Jha and Suciu 2011].*

4.3 The Dichotomy Theorem

We now state and prove our main result, that every unsafe query is #P-hard, which implies the dichotomy. The proof is based on Theorem 4.5, which we prove in the rest of the paper. We first define formally what it means for a query to be unsafe. We define a set of rewrite rules between UCQ queries, $Q \to Q'$, and say that Q is unsafe if there exists a rewriting $Q \stackrel{*}{\to} Q'$ where Q' is immediately unsafe.

If Q is a query and R a relational symbol, denote $Q[R = \texttt{false}]$ (and $Q[R = \texttt{true}]$ respectively) the result of replacing in Q every atom that refers to R with `false` (or with `true` respectively).

DEFINITION 4.13. *Let* Q *and* Q' *be two ranked queries, not necessarily over the same vocabulary. We define the following three rewrite rules, in notation* $Q \to Q'$*, and say that* Q *rewrites to* Q'*, if one of the following conditions hold:*

—$Q' \equiv Q[R = \texttt{false}]$, *or* $Q' \equiv Q[R = \texttt{true}]$. *We call this a* deterministic rewrite rule. *Thus,* $Q \to Q[R = \texttt{false}]$ *and* $Q \to Q[R = \texttt{true}]$.

—Q *is a CNF expression, and* $Q' \equiv d_v$, *where* v *is some element in lattice* $L(Q)$ *s.t.* $\mu(v, \hat{1}) \neq 0$. *We call this a* subquery rewrite rule. *Thus,* $Q \to d_v$.

—*There exists a set of constants* A *s.t.* $|A| \leq |Var(Q)|$ *and a shattered vocabulary* $\mathbf{R}_A$ *such that* $Q' = Q_A$, *the shattered query on* $\mathbf{R}_A$. *We call this a* shatter rewrite rule. *Thus,* $Q \to Q_A$.

Denote $\stackrel{*}{\to}$ *the reflexive and transitive closure of* $\to$.

DEFINITION 4.14. *Let* Q *be a ranked query.* Q *is called* unsafe *if there exists a rewriting* $Q \stackrel{*}{\to} Q'$, *s.t.* Q' *is immediately unsafe (Def. 4.4). Otherwise,* Q *is called* safe.

EXAMPLE 4.15. *For a simple illustration, the following query is unsafe:*

$$d = R(z_0, x_0), S(z_0, x_0, y_0) \vee S(z_1, x_1, y_1), T(z_1, y_1)$$

To show this, we give a rewriting to h_1, *shown in Example 4.6, which is immediately unsafe. In the third line we assimilate* $d[a/z]$ *with its complete shatter.*

$$\begin{aligned} d = \;& \exists z.(R(z, x_0), S(z, x_0, y_0) \vee S(z, x_1, y_1), T(z, y_1)) \\ \to \;& R(a, x_0), S(a, x_0, y_0) \vee S(a, x_1, y_1), T(a, y_1) && = d[a/z] \\ = \;& R_{a*}(x_0), S_{a**}(x_0, y_0) \vee S_{a**}(x_1, y_1), T_{a*}(y_1) \\ = \;& h_1 \end{aligned}$$

If algorithm 2 starts with a query Q and reaches recursively another query Q', then we claim that $Q \stackrel{*}{\to} Q'$. We check this by examining each line where the algorithm makes a recursive call. In line 3, the input query has several symbol components $Q = Q_1 \wedge \ldots \wedge Q_m$, and the recursive call is on each Q_i. Fix i, and set $R = \texttt{true}$ for all relation symbols that do not occur in Q_i: then $Q_j \stackrel{*}{\to} \texttt{true}$ for $j \neq i$, and we have $Q = Q_1 \wedge \ldots \wedge Q_k \stackrel{*}{\to} Q_i$. In line 7 the input is a CNF query $Q = d_1 \wedge \ldots \wedge d_k$ and the recursive call is on d_v, for $v \in L(Q), \mu(v, \hat{1}) \neq 0$: clearly $Q \to d_v$. In line 12 the recursive call is for a symbol-component d_i: here we set $R = \texttt{false}$ for all relation symbols that do not occur in d_i and obtain $d = d_1 \vee \ldots \vee d_k \stackrel{*}{\to} d_i$. Finally, in line 17 we have immediately $d \to d[a/z]$.

Therefore, if Q is *safe*, by Def. 4.14, then the algorithm will succeed, and Q is in $UCQ(P)$. The converse is not obvious: if Q is unsafe, then it is not immediately clear that the algorithm will fail, because if $Q \stackrel{*}{\to} Q'$ then, in general, the algorithm does not have to reach Q'. But if $Q \stackrel{*}{\to} Q'$ and Q' immediately unsafe then Q' is #P hard, and we will prove (using the lemmas below) that Q is also #P-hard; thus, unless $FP = \#P$, the algorithm must fail, perhaps by reaching another immediately unsafe query Q''.

LEMMA 4.16. $Q[R = \texttt{false}] \leq^{FO}_{lin} Q$ *and* $Q[R = \texttt{true}] \leq^{FO}_{lin,+} Q$.

PROOF. We need to define a linear mapping $\mathbf{F}$ from D (on which we want to evaluate $Q[R = \texttt{false}]$) to D' (on which we evaluate Q). The mapping simply copies the entire database, except for the relation R, which it leaves empty: $R^{D'} = \emptyset$, $S^{D'} = S^D$ for any other symbol S. Obviously, $\Phi^D_{Q[R=\texttt{false}]} = \Phi^{D'}_Q$. For the other

reduction, we add a second mapping, $\mathbf{E}$, which defines $R^{D'} = (ADom)^k$, where k is the arity of R: since the Boolean variables $\mathbf{Y}$ of these tuples are set to `true`, we have $\Phi^D_{Q[R=\texttt{true}]} = \Phi^{D'}_Q[\mathbf{Y} = \mathbf{true}]$. □

Call a subquery rewrite rule $Q \to d_v$ (where $\mu(v, \hat{1}) \neq 0$) a *maximal rewrite rule* if: for all $u > v$, if $\mu(u, \hat{1}) \neq 0$, then d_u is safe.

LEMMA 4.17. *If $Q \to d_v$ is a maximal rewrite rule, then $d_v \leq^{FO}_{prob} Q$.*

PROOF. Recall that the query Q is logically equivalent to $\bigwedge_{u \in L(Q)} d_u$. Define $Z = \{z \mid v \not\leq z < \hat{1}\}$ and $V = \{u \mid v \leq u < \hat{1}\}$. Thus, Z and V form a partition of $L(Q) - \{\hat{1}\}$, $Z \cup V = L(Q)$. Let $Q_v = \bigwedge_{u \in V} d_u$. We prove two claims: (1) $Q_v \leq^{\text{FO}}_{\text{lin},+} Q$ and (2) $d_v \equiv^{\text{FO}}_{\text{prob}} Q_v$. These two claims prove the lemma.

We prove claim (1). Let D' be a database instance (on which we want to compute the lineage for Q_v). We map it to a database consisting of two parts, $D = \mathbf{F}(D) \cup \mathbf{E}(D)$. The first part is a linear mapping, $\mathbf{F}(D)$, and its tuples are in 1-to-1 correspondence with tuples in D'; for the second part we will set their Boolean variables $\mathbf{Y} = \mathbf{true}$; then we prove that $\Phi^D_Q[\mathbf{Y} = \mathbf{true}] = \Phi^{D'}_{Q_v}$. Take $\mathbf{F}$ to be the identity, $\mathbf{F}(D') = D'$. Define $\mathbf{E}$ as follows: $\mathbf{E}(D) = \bigcup_{z \in Z} D_z = D_Z$, where the databases D_z have disjoint active domains, and satisfy $D_z \models d_z$, and $D_z \not\models d_v$ (which also implies that $D_z \not\models d_u$ for any $u \in V$). We describe now how to construct D_z. By the definition of the CNF lattice, $v \not\leq z$ implies $d_z \not\Rightarrow d_v$. Consider the components of the disjunctive query $d_z = c_1 \vee c_2 \vee \ldots$ By Prop. 2.13, there exists i such that $c_i \not\Rightarrow d_v$. Define D_z to be the canonical database of c_i (if there are several such i, choose one arbitrarily). Since the component c_i is a conjunctive query without constants, its canonical database is invariant under isomorphisms of the domain, and we can choose D_z such that its active domain is disjoint from the active domain of D', and from that of any other database $D_{z'}$, for $z' \neq z$. Define $\mathbf{E}(D) = D_Z = \bigcup_{z \in Z} D_z$. Notice that D_Z depends only on the query expression, and not on the database D'. Finally, the output database (on which we compute Q) is $D = D' \cup D_Z$. Let $\mathbf{X}$ denote a set of Boolean variables for the tuples in D' and let $\mathbf{Y}$ denote Boolean variables for the tuples in D_Z. We need to prove $\Phi^{D'}_{Q_v} = \Phi^D_Q[\mathbf{Y} = \mathbf{true}]$. We have: $\Phi^D_Q = \Phi^D_{\bigwedge_u d_u} = \bigwedge_{u < \hat{1}} \Phi^D_{d_u}$. If $z \in Z$ then $\Phi^D_{d_z}[\mathbf{Y} = \mathbf{true}] = \texttt{true}$, because d_z is true on the database D_z, and all Boolean variables corresponding to tuples in D_z are set to `true`. Thus, $\bigwedge_{u < \hat{1}} \Phi^D_{d_u}[\mathbf{Y} = \mathbf{true}] = \bigwedge_{u \in V} \Phi^D_{d_u}[\mathbf{Y} = \mathbf{true}]$. Next, notice that $\Phi^D_{d_u} = \Phi^{D'}_{d_u}$, because, by construction of D_z, d_u cannot use any of the tuples in D_z. This proves $\Phi^D_Q[\mathbf{Y} = \mathbf{true}] = \bigvee_{u \in V} \Phi^{D'}_{d_u} = \Phi^{D'}_{Q_v}$, completing the proof of claim (1).

Claim (2) follows immediately from the fact that, on any probabilistic database $\mathbf{D}$, $P_{\mathbf{D}}(Q_v) = -\sum_{u \in V} \mu(u, \hat{1}) P_{\mathbf{D}}(d_u)$. Denote $S = -\sum_{u \in V - \{v\}} \mu(u, \hat{1}) P_{\mathbf{D}}(d_u)$: this quantity is computable in PTIME, since all queries d_u are either safe, or $\mu(u, \hat{1}) = 0$. Thus, $P_{\mathbf{D}}(Q_v) = S - \mu(v, \hat{1}) P_{\mathbf{D}}(d_v)$, and the claim follows from the fact that $\mu(v, \hat{1}) \neq 0$. □

LEMMA 4.18. *If $Q \to Q_A$ is a shattering rewrite rule, then $Q_A \leq^{FO}_{lin} Q$.*

PROOF. Follows immediately from Prop. 2.9. □

Call a sequence of rewritings $Q \stackrel{*}{\to} Q'$ *maximal* if every subquery rewriting rule

$Q \to d_v$ is maximal (no restrictions on the other two rewrite rules). The three lemmas imply:

COROLLARY 4.19. *If $Q \stackrel{*}{\to} Q'$ is a maximal rewriting, then $Q' \leq^{FO}_{prob} Q$.*

Finally, the last step in proving the dichotomy result is:

LEMMA 4.20. *If Q is unsafe, then there exists a maximal rewriting $Q \stackrel{*}{\to} Q'$ s.t. Q' is immediately unsafe.*

PROOF. By induction on the size of Q. If Q is unsafe then there exists a rewriting $Q \to Q''$ s.t. Q'' is unsafe. If this reduction is a non-maximal rule, meaning $Q'' = d_v$ and v is not maximal with properties (a) and (b), then choose a maximal u: then $Q \to d_u$ is a maximal reduction rule, and by induction we have a maximal rewriting $d_u \stackrel{*}{\to} Q'$ for some immediately unsafe query Q'. □

We can now state and prove the main result of the paper. Its proof is based on Theorem 4.5, which we haven't proven yet, but will prove in the remainder of this paper.

THEOREM 4.21 DICHOTOMY. *Consider an arbitrary UCQ query Q and let Q' be any ranked query that is lineage-equivalent to Q (Prop. 4.2). Then one of the following holds:*

—If Q' is safe, then $Q \in UCQ(P)$.

—If Q' is unsafe, then computing $P(Q)$ is #P-hard.

PROOF. Since $Q \equiv^{\text{FO}}_{\text{lin}} Q'$ we have both $Q \leq^{\text{FO}}_{\text{lin}} Q'$ and $Q' \leq^{\text{FO}}_{\text{lin}} Q$, which implies that we have both $Q \leq^{\text{FO}}_{\text{prob}} Q'$ and $Q' \leq^{\text{FO}}_{\text{prob}} Q$.

For the first item, we note that if Q' is safe then algorithm 2 never fails, hence it computes $P(Q')$ in PTIME; since $Q \leq^{\text{FO}}_{\text{prob}} Q'$, we can compute $P(Q)$ in PTIME given $P(Q')$.

We prove the second item. By Lemma 4.20 there exists a maximal rewriting $Q' \stackrel{*}{\to} Q''$ to an immediately unsafe Q''. By Corollary 4.19, we have $Q'' \leq^{\text{FO}}_{\text{prob}} Q'$, and we already established $Q' \leq^{\text{FO}}_{\text{prob}} Q$. By Theorem 4.5 Q'' is #P-hard, hence so is Q. □

5. THE MAIN PROOF TECHNIQUES

In the rest of the paper we will prove the main technical result of this paper, Theorem 4.5: every immediately unsafe query is #P-hard. The complexity class #P was introduced by [Valiant 1979] and consists of all function problems of the following type: given a polynomial-time, non-deterministic Turing machine, compute the number of accepting computations. For a Boolean expression Φ, let $\#\Phi$ denote the number of satisfying assignments (valuations). The model counting problem, "given Φ, compute $\#\Phi$", is denoted #SAT, and is obviously in #P. Our proof of Theorem 4.5 relies on the following theorem by [Provan and Ball 1983]:

THEOREM 5.1. *Let $X_1, \ldots, X_{n_1}$ and $Y_1, \ldots, Y_{n_2}$ be two disjoint sets of Boolean variables, and let $E \subseteq [n_1] \times [n_2]$ be a bipartite graph. The* Positive, Partitioned

2-CNF *propositional formula defined by E is:*

$$\Phi = \bigwedge_{(i,j)\in E} (X_i \vee Y_j)$$

Denote #PP2CNF the problem "given a PP2CNF formula Φ, compute $\#\Phi$". Then, #PP2CNF is hard for #P.

It follows immediately that the dual problem, #PP2DNF, where the formula is $\bigvee_{(i,j)\in E} X_iY_j$ is also #P-hard.

The proof of Theorem 4.5 consists of three major steps: Theorem 6.3, Theorem 7.3, and Theorem 8.1. The first step takes an immediately unsafe query d and constructs a *leveled* query d^L s.t. $d^L \leq^{\text{FO}}_{\text{prob}} d$. The second, applies repeatedly maximal rewrite rules $d^L \stackrel{*}{\rightarrow} d'$ as long as d' is still unsafe; we call the last query in the rewriting, d', a *forbidden query*. Thus, $d' \leq^{\text{FO}}_{\text{prob}} d^L$, and d' is forbidden. Finally, the third step proves that every forbidden query is #P-hard, by direct reduction from #PP2CNF. These three steps imply our main result, because $d' \leq^{\text{FO}}_{\text{prob}} d$ and d' is #P-hard implies that d is #P-hard. The last step, showing that every forbidden query is #P-hard, is the most interesting one. It makes novel use of some techniques from algebra and analysis, which we consider to be the most interesting aspect of the entire proof. In this section we present these techniques and illustrate them by showing #P-hardness for four forbidden queries: h_0, h_1, h_4 and h_2 (defined in Example 4.6). The complete proof of the third step is in Sect. 8.

5.1 Warmup: h_0 and h_1 are #P-Hard

As a gentle warmup, we review here the #P-hardness proofs for h_0 and h_1 from [Dalvi and Suciu 2004; Dalvi and Suciu 2006]:

PROPOSITION 5.2. *The queries $h_0 = R(x), S(x,y), T(y)$ and $h_1 = R(x_0), S(x_0,y_0) \vee S(x_1,y_1), T(y_1)$ are #P-hard.*

PROOF. Let $E \subseteq [n_1] \times [n_2]$ and denote $\Phi = \bigvee_{(i,j)\in E} X_iY_j$. Define the instance: $R^D = [n_1]$, $S^D = E$, $T^D = [n_2]$, and the probabilities are $P(R^D(i)) = 1/2$, $P(S^D(i,j)) = 1$, $P(T^D(j)) = 1/2$. Then $\#\Phi = 2^n P(h_0)$ where $n = n_1 + n_2$ is the total number of variables in Φ. Thus, given an oracle for computing the query probability $P(h_0)$, we can compute $\#\Phi$; since the latter is #P-hard, so is the former.

The proof for h_1 is more involved. Start with a PP2CNF, $\Phi = \bigwedge_{(i,j)\in E}(X_i \vee Y_j)$, and let $m = |E|$ be the number of clauses. We show how to compute $\#\Phi$ using an oracle for computing $P(h_1)$. As before we set $R^D = [n_1], S^D = E, T^D = [n_2]$, and $P(R^D(i)) = P(T^D(j)) = 1/2$. Now we set $P(S^D(i,j)) = 1 - z$, where $z \in (0,1)$ will be chosen below. We will compute $P(\neg h_1)$. If $W \subseteq D$, $W = (R^W, S^W, T^W)$ is a possible world, its probability, given by Eq. 1, is $P(W) = 1/2^n P(S^W) = 1/2^n z^{m-c}(1-z)^c$ where $c = |S^W|$ (because the contribution of a tuple in R^D or in T^D is 1/2 regardless of whether it is in the world W or not). By definition $P(\neg h_1) = \sum_{W:\neg(W\models h_1)} P(W)$.

For any valuation θ of the Boolean variables in Φ, denote E_θ the following event (i.e predicate on the world $W = (R^W, S^W, T^W)$):

$$E_\theta \equiv \ \forall i.(i \in R^W \text{ iff } \theta(X_i) = \mathit{true}) \ \ \wedge \ \ \forall j.(j \in T^W \text{ iff } \theta(Y_j) = \mathit{true}) \qquad (14)$$

In other words, the event E_θ fixes the relations R^W and T^W according to θ, and leaves S^W totally unspecified. Its probability is $P(E_\theta) = P(\theta) = 1/2^n$.

The events E_θ are disjoint, so we can write $P(\neg h_1)$ as:

$$P(\neg h_1) = \sum_\theta P(\neg h_1 | E_\theta) \cdot P(E_\theta) = \frac{1}{2^n} \sum_\theta P(\neg h_1 | E_\theta) \tag{15}$$

We will compute now the conditional probability, $P(\neg h_1 | E_\theta)$. Let $K(\theta) = \{(i,j) \in E \mid \theta(X_i \vee Y_j) = \mathit{true}\}$ be the set of clauses in Φ that are satisfied by θ. Its cardinality $|K(\theta)|$ is a number between 0 and m. Then, we claim that:

$$P(\neg h_1 | E_\theta) = z^{|K(\theta)|} \tag{16}$$

E_θ fixes the relations R^W and T^W, so we need to compute the probability of the event $(R^W, S^W, T^W) \not\models h_1$ over the random choices of S^W. Fix an edge $(i,j) \in E = S^D$. If a world W contains either both tuples $\{R(i), S(i,j)\}$ or both tuples $\{S(i,j), T(j)\}$, then the query h_1 is true. We consider two cases. (1) θ satisfies $X_i \vee Y_j$, in other words $(i,j) \in K(\theta)$. Then, W contains either $R(i)$ or $T(j)$, so in order for h_1 to be false it must not contain $S(i,j)$, and the probability of this event is z. (2) θ does not satisfy $X_i \vee Y_j$, in other words $(i,j) \notin K(\theta)$. In that case h_1 is false on $\{R(i), S(i,j), T(j)\} \cap W$, and we have no restriction on $S(i,j)$. This proves Eq. 16.

Now we can compute $P(\neg h_1)$ by combining Eq. 15 and Eq. 16. For any k, $0 \leq k \leq m$, let $\#k$ be the number of valuations θ that satisfy exactly k clauses, i.e. $\#k = |\{\theta \mid k = |K(\theta)|\}|$. Then, Eq. 15 and Eq. 16 become:

$$P(\neg h_1) = \frac{1}{2^n} \sum_{k=0,m} \#k \cdot z^k \tag{17}$$

This is a polynomial in z of degree m, with coefficients $\#0, \#1, \ldots, \#m$. In other words, an oracle for $P(\neg h_1)$ allows us to compute the polynomial at an arbitrary value z. Its coefficients $\#0, \#1, \ldots, \#m$ are unknown, and, in particular, $\#\Phi = \#m$, because $\#\Phi$ represents the number of valuations that satisfy all m clauses. We will use the oracle to compute *all* coefficients, then return $\#m$. Choose any $m+1$ distinct values $z_0, z_1, \ldots, z_m \in (0,1)$, and construct $m+1$ different database instances R, S, T as described above: they are identical except for the probability of tuples in S, which is $1 - z_i$ for the i'th instance. The oracle gives us the value, say p_i, of the polynomial on z_i. Form a linear system of $m+1$ equations with $m+1$ unknowns $\#0, \ldots, \#m$. Its matrix is a Vandermonde matrix $V(z_0, z_1, \ldots, z_m)$, and is non-singular, since $z_i \neq z_j$, for $i \neq j$. Thus, we can solve the system in PTIME, and return the last unknown, $\#m$. □

5.2 Proof Technique 1: The Jacobian

Our next step is to prove that h_4 is #P-hard, and we will use it to motivate the first technique from analysis, the Jacobian. The proof starts with the same simple reduction from #PP2CNF as for h_1, but, as we shall see, gets more involved. Recall:

$$\begin{aligned} h_4 = & R(x_0), S_1(x_0, y_0) \vee S_1(x_1, y_1), S_2(x_1, y_1) \vee \\ & S_2(x_2, y_2), S_3(x_2, y_2) \vee S_3(x_3, y_3), S_4(x_3, y_3) \vee S_4(x_4, y_4), T(y_4) \end{aligned}$$

Define the following database instance D:

$$R^D = [n_1], \qquad S_1^D = S_2^D = S_3^D = S_4^D = E, \qquad T^D = [n_2]$$

and define the probabilities as follows, for all $i \in [n_1], j \in [n_2]$ and $(i,j) \in E$:

$$P(R^D(i)) = P(T^D(j)) = 1/2$$
$$P(S_1^D(i,j)) = 1 - z_1, P(S_2^D(i,j)) = 1 - z_2, P(S_3^D(i,j)) = 1 - z_3, P(S_4^D(i,j)) = 1 - z_4$$

The values $z_1, z_2, z_3, z_4 \in (0,1)$ will be chosen later. As before, given a valuation θ of the Boolean variables in Φ, denote E_θ the event Eq. 14: thus, E_θ consists of the worlds W where R^W and T^W are set according to θ and $S_1^W, S_2^W, S_3^W, S_4^W$ are arbitrary. As before, $P(\neg h_4) = 1/2^n \cdot \sum_\theta P(\neg h_4 | E_\theta)$. We compute the conditional probability. Call the set of tuples $D(i,j) = \{R(i), S_1(i,j), \ldots, S_4(i,j), T(j)\}$ a *block*. We claim that h_4 is false in a world iff it is false in each block: indeed, h_4 is false iff each of its components $R(x_0), S_1(x_0,y_0)$, and $S_1(x_1,y_1), S_2(x_1,y_2)$, etc, is false, and this happens iff each component is false in each block. For example, if the component $R(x_0), S_1(x_0,y_0)$ is true in a world W, then W must contain two tuples of the form $R(i), S_1(i,j)$, hence the component is true in the block $D(i,j) \cap W$. We use the Boolean variables $U, V, Z_1, \ldots, Z_4$ to denote the following events in the block $D(i,j)$:

$$U \equiv i \notin R^W \qquad V \equiv j \notin T^W$$
$$Z_1 \equiv (i,j) \notin S_1^W \quad Z_2 \equiv (i,j) \notin S_2^W \quad Z_3 \equiv (i,j) \notin S_3^W \quad Z_4 \equiv (i,j) \notin S_4^W$$

The event that h_4 is false on the block $D(i,j)$ is given by the following Boolean expression, which is exactly the dual Boolean function of the lineage of h_4 on $D(i,j)$:

$$Y(U,V) = (U \vee Z_1) \wedge (Z_1 \vee Z_2) \wedge (Z_2 \vee Z_3) \wedge (Z_3 \vee Z_4) \wedge (Z_4 \vee V) \tag{18}$$

The variables U and V are set by θ, in one of four possible ways. Let $Y_{11} \equiv Y[U = \texttt{false}, V = \texttt{false}]$, $Y_{12} \equiv Y[U = \texttt{false}, V = \texttt{true}]$, etc, then:

$$\begin{aligned} Y_{11} &= Z_1 \wedge (Z_2 \vee Z_3) \wedge Z_4 \\ Y_{12} &= Z_1 \wedge (Z_2 \vee Z_3) \wedge (Z_3 \vee Z_4) \\ Y_{11} &= (Z_1 \vee Z_2) \wedge (Z_2 \vee Z_3) \wedge Z_4 \\ Y_{12} &= (Z_1 \vee Z_2) \wedge (Z_2 \vee Z_3) \wedge (Z_3 \vee Z_4) \end{aligned} \tag{19}$$

Notice that $P(Z_k) = P(\neg S_k^D(i,j)) = z_k$, $k = 1, 4$. Thus, the probabilities $y_{uv} = P(Y_{uv})$ are given as follows (the details of these complicated expressions are not important, since we will replace them later with better behaved expressions):

$$\begin{aligned} y_{11} &= P(Y_{11}) = z_1 \cdot (z_2 + z_3 - z_2 \cdot z_3) \cdot z_4 \\ y_{12} &= P(Y_{12}) = z_1 \cdot (z_2 \cdot (1 - z_3) \cdot z_4 + z_3) \\ y_{21} &= P(Y_{11}) = (z_1 \cdot (1 - z_2) \cdot z_3 + z_2) \cdot z_4 \\ y_{22} &= P(Y_{12}) = z_1 \cdot z_3 + z_2 \cdot z_3 + z_2 \cdot z_4 - z_1 \cdot z_2 \cdot z_3 - z_2 \cdot z_3 \cdot z_4 \end{aligned} \tag{20}$$

For an example, suppose $\theta(X_i) = \texttt{true}$ and $\theta(Y_j) = \texttt{false}$; then, conditioned on E_θ, we have $i \in R^W$ and $j \notin T^W$, thus $U = \texttt{false}, V = \texttt{true}$, and the event *"$h_4$ is false on the block $D(i,j)$"* is given by Y_{12}; the probability of $\neg h_4$ on the

block $D(i,j)$ is y_{12}. In general, the probability $P(\neg h_4|E_\theta)$ is the product of the probabilities that h_4 is false on each block, because E_θ ensures that different blocks are independent. Denoting $1 \equiv \texttt{true}$, $2 \equiv \texttt{false}$, we define for each $u, v \in \{1,2\}$ the set:

$$K_{uv}(\theta) = \{(i,j) \mid (i,j) \in E, \theta(X_i) = u, \theta(Y_j) = v\} \tag{21}$$

For example, $K_{12} = \{(i,j) \mid (i,j) \in E, \theta(X_i) = \texttt{true}, \theta(Y_j) = \texttt{false}\}$. Then:

$$P(\neg h_4|E_\theta) = y_{11}^{|K_{11}(\theta)|} \cdot y_{12}^{|K_{12}(\theta)|} \cdot y_{21}^{|K_{21}(\theta)|} \cdot y_{22}^{|K_{12}(\theta)|} \tag{22}$$

Notices how this formula generalizes Eq. 16. For any tuple $\mathbf{k} = (k_{11}, k_{12}, k_{21}, k_{22}) \in (\{0, \ldots, m\})^4$ let $\#\mathbf{k}$ be the number of valuations θ with the property that $|K_{uv}(\theta)| = k_{uv}$ for all $u, v \in \{1,2\}$. Note that $\#\Phi = \sum_{k_{22}=0} \#\mathbf{k}$. Then:

$$P(\neg h_4) = 1/2^n \sum_{\mathbf{k}} \#\mathbf{k} \cdot y_{11}^{k_{11}} \cdot y_{12}^{k_{12}} \cdot y_{21}^{k_{21}} \cdot y_{22}^{k_{22}} \tag{23}$$

The corresponding formula for h_1 was Eq. 17; it was simpler, because in the case of h_1 it happens that $Y_{11} \equiv Y_{12} \equiv Y_{21}$, and this lead to the simpler expression.

We will attempt now to extend the argument in the proof of h_1: use an oracle for $P(\neg h_4)$ to compute the polynomial above at several points, then solve for the $(m+1)^4$ unknowns $\#\mathbf{k}$; then return $\#\Phi = \sum_{k_{22}=0} \#\mathbf{k}$. Suppose we could set each of the four variables y_{uv} independently to $m+1$ distinct values in $y_{uv,0}, \ldots, y_{uv,m} \in (0,1)$. Therefore, there are $(m+1)^4$ possible combinations for $\bar{y} = (y_{11}, \ldots, y_{22})$. For each of them, construct the database D, use the oracle to compute $P(\neg h_4)$, and this gives the value of the polynomial Eq. 23. We obtain a system of $(m+1)^4$ linear equations, with as many unknowns. The matrix of this system is a Kronecker product of four Vandermonde matrices, $V(y_{11,0}, \ldots, y_{11,m}) \otimes \cdots \otimes V(y_{22,0}, \ldots, y_{22,m})$, and therefore is non-singular[6]. But we cannot set the y_{uv} variables independently. Instead, we can only construct the database D, by setting the probabilities $z_1, \ldots, z_4$ independently. Here, we use the fact that the Jacobian of the function $\bar{z} \mapsto \bar{y}$ is non-zero.

DEFINITION 5.3. *Let $\bar{F} = (F_1, \ldots, F_k)$ be k functions in variables $\bar{z} = (z_1, \ldots, z_k)$. The Jacobian the following matrix:*

$$\frac{\partial \bar{F}}{\partial \bar{z}} = \begin{pmatrix} \frac{\partial F_1}{\partial z_1} & \cdots & \frac{\partial F_1}{\partial z_k} \\ & \cdots & \\ \frac{\partial F_k}{\partial z_1} & \cdots & \frac{\partial F_k}{\partial z_k} \end{pmatrix}$$

The Jacobian determinant, or simply Jacobian, is $\texttt{det}(\frac{\partial \bar{F}}{\partial \bar{z}})$.

The inverse function theorem states that if $\texttt{det}(\frac{\partial \bar{F}}{\partial \bar{z}}) \neq 0$ at some point in $\bar{z} \in \mathbb{R}^k$, then it is invertible in some neighborhood of $\bar{z}$. We apply this to complete the hardness proof for h_4. One can check through direct calculations[7] that $\texttt{det}(\frac{\partial \bar{y}}{\partial \bar{z}}) \neq 0$, where $\bar{y} = (y_{11}, \ldots, y_{22})$ are the four functions in Eq. 20. This leads to the following naive algorithm for computing $\#\Phi$. Choose $m+1$ distinct values for y_{uv} in $(0,1)$

[6]Recall that $\texttt{det}(A \otimes B) = \texttt{det}(A) \cdot \texttt{det}(B)$.

[7]The Jacobian is a multivariate polynomial in $z_1, \ldots, z_4$ where each variable has degree ≤ 3. Its structure is quite interesting, for example it appears to be divisible by $(z_2 - z_3)$, but a full analysis

such that all $(m+1)^4$ four dimensional points $\bar{y} \in (0,1)^4$ are in a neighborhood where the Jacobian determinant is non-zero. For each point, compute the inverse function, obtaining $\bar{z} \in (0,1)^4$. Each corresponds to a probabilistic database D, on which we compute $P(h_4)$. Repeating this $(m+1)^4$ times guarantees that the resulting system of $(m+1)^4$ equations has a non-singular matrix, so we can solve for $\#\mathbf{k}$, and compute $\#\Phi$. This algorithm is naive because inverting the mapping $\bar{z} \to \bar{y}$ creates some rounding issues. In Sect. 8.5 we show an improved algorithm, that avoids the need to invert the mapping, and can be run entirely with rational numbers.

However, this construction works only for h_4. It is unclear how to extend it to more complex queries. One difficulty is proving in general that the Jacobian is non-zero. Another difficulty is choosing the four independent variables $z_1, \ldots, z_4$. For example, in h_2 there are only two relations S_1, S_2, giving us only two variables z_1, z_2: were do we get the other two? It also turns out that for some queries we need more than 4 variables. For these reasons, our proof relies on a different reduction that allows us to introduce arbitrarily many variables z_i, and also simplifies significantly the multilinear polynomials y_{uv}. We illustrate it next.

5.3 Proof Technique 2: The Cauchy Double Alternant

We illustrate our second technique by proving that h_2 is #P-hard. We construct a different database D, whose Jacobian is Cauchy's Double Alternant, which has a closed form and is easier to check when it is non-zero. While we illustrate this on h_2, the construction generalizes, as we show in Sect. 8. Recall:

$$h_2 = R(x_0), S_1(x_0, y_0) \vee S_1(x_1, y_1), S_2(x_1, y_1) \vee S_2(x_2, y_2), T(y_2)$$

Given $E \subseteq [n_1] \times [n_2]$, define the following sets of constants:

$$\begin{array}{lll} A = \{a_i \mid i \in [n_1]\} & B = \{b_j \mid i \in [n_2]\} & \\ \breve{A}_k = \{\breve{a}_{ij,k} \mid (i,j) \in E\} & \breve{B}_k = \{\breve{b}_{ij,k} \mid (i,j) \in E\} & k = 1,4 \\ \breve{A} = \bigcup_{k=1,4} \breve{A}_k & \breve{B} = \bigcup_{k=1,4} \breve{B}_k & \end{array} \tag{24}$$

We call the index k the slice number; thus, there are four slices. In each slice, the constant $\breve{a}_{ij,k}$ is unique for i, j: the accent is meant to indicate that it depends on both indices, i, j unlike the other constants a_i that depend only on one index.

---

is beyond our scope here. Instead, we compute its value at $z_1 = z_3 = 2/3, z_2 = z_4 = 1/3$.

$$J = \begin{pmatrix} (z_2 + z_3 - z_2z_3)z_4 & z_1(1-z_3)z_4 & z_1(1-z_2)z_4 & z_1(z_2+z_3-z_2z_3) \\ z_2(1-z_3)z_4 + z_3 & z_1(1-z_3)z_4 & z_1(1-z_2z_4) & z_1z_2(1-z_3) \\ (1-z_2)z_3z_4 & (1-z_1z_3)z_4 & z_1(1-z_2)z_4 & z_1(1-z_2)z_3 + z_2 \\ z_3 - z_2z_3 & z_3 + z_4 - z_1z_3 - z_3z_4 & z_1 + z_2 - z_1z_2 - z_2z_4 & z_2 - z_2z_3 \end{pmatrix}$$

$$= \frac{1}{27} \begin{pmatrix} 7 & 2 & 4 & 14 \\ 19 & 2 & 16 & 2 \\ 4 & 5 & 4 & 17 \\ 12 & 9 & 18 & 3 \end{pmatrix}$$

Using an online determinant calculator we obtain the determinant of the integer matrix above as -432, thus $\mathtt{det}(J) = -432/27^4 = -\frac{2^4}{3^9}$. It follows that $\mathtt{det}(J)$ is not identically zero.

Next, we define:

$$R^D = A \cup \breve{A}, \qquad T^D = B \cup \breve{B},$$
$$S_1^D = S_2^D = \bigcup_{k=1,4} \{(a_i, \breve{b}_{ij,k}), (\breve{a}_{ij,k}, \breve{b}_{ij,k}), (\breve{a}_{ij,k}, b_j) \mid (i,j) \in E\} \tag{25}$$

We set the probabilities $P(R^D(a_i)) = P(T^D(b_j)) = 1/2$. The other probabilities, for $P(R^D(\breve{a}_{ij,k}))$ and $P(T^D(\breve{b}_{ij,k}))$ and for all tuples in S_1^D, S_2^D are set as follows. In each slice k fix a distinguished tuple, and set its probability to $1 - z_k$, where $z_k \in (0,1)$; for all non-distinguished tuples, set their probability to some constant. We will describe below how to choose the distinguished tuple and the constants. For now, to make the discussion concrete, let's assume the distinguished tuple is the second S_1 tuple, hence $P(S_1^D(\breve{a}_{ij,k}, \breve{b}_{ij,k})) = 1 - z_k$ for $k = 1, 4$; and assume that all the other probabilities[8] are $= 1/3$: $P(S_1^D(a_i, \breve{b}_{ij,k})) = P(S_2^D(a_i, \breve{b}_{ij,k})) = P(T^D(\breve{b}_{ij,k})) = \ldots = 1/3$. But we will revisit this choice later.

To compute $P(\neg h_2)$ we notice that the database D is "partitioned", we say that it is *leveled*. More precisely, referring to the schema $R(x), S_1(x,y), S_2(x,y), T(y)$, the domain of the attribute x is partitioned into A and $\breve{A}$, call them levels 0 and 1 respectively. Similarly, the attribute y is partitioned into B and $\breve{B}$, we also call them levels 0,1. Define the *leveled* vocabulary to be $R^0, R^1, S_1^{01}, S_2^{01}, S_1^{11}, S_2^{11}, S_1^{10}, S_2^{10}, T^0, T^1$ and denote D^L the same database instance, viewed in this new vocabulary. For example, in D^L, $R^0 = A$, $R^1 = \breve{A}$, $S_1^{01} \subseteq A \times \breve{B}$, $S_1^{11} \subseteq \breve{A} \cup \breve{B}$, etc, and $T^0 = B, T^1 = \breve{B}$. Note that by construction there are no tuples from $A \times B$, hence we do not have symbols S_1^{00} or S_2^{00}. The database instance D^L has a zig-zag-zig shape. We specialize h_2 to the leveled database, and obtain:

$$\begin{aligned} h_2^L = & R^0(x_0^0), S_1^{01}(x_0^0, \breve{y}_0^1) \vee S_1^{01}(x_1^0, \breve{y}_1^1), S_2^{01}(x_1^0, \breve{y}_1^1) \vee S_2^{01}(x_2^0, \breve{y}_2^1), T^1(\breve{y}_2^1) \\ & \vee R^1(\breve{x}_0^1), S_1^{11}(\breve{x}_0^1, \breve{y}_0^1) \vee S_1^{11}(\breve{x}_1^1, \breve{y}_1^1), S_2^{11}(\breve{x}_1^1, \breve{y}_1^1) \vee S_2^{11}(\breve{x}_2^1, \breve{y}_2^1), T^1(\breve{y}_2^1) \\ & \vee R^1(\breve{x}_0^1), S_1^{10}(\breve{x}_0^1, y_0^0) \vee S_1^{10}(\breve{x}_1^1, y_1^0), S_2^{10}(\breve{x}_1^1, y_1^0) \vee S_2^{10}(\breve{x}_2^1, y_2^0), T^0(y_2^0) \end{aligned}$$

Intuitively, the query h_2 on D is "the same" as h_2^L on D^L. For example, assume h_2 is true in a world $W \subseteq D$; for the sake of the discussion, assume that its first component $R(x_0), S_1(x_0, y_0)$ is true in W. Then W contains either two tuples of the form $R(a_i), S_1(a_i, \breve{b}_{ij,k})$, or two tuples of the form $R(\breve{a}_{ij,k}), S_1(\breve{a}_{ij,k}, \breve{b}_{ij,k})$, or two tuples of the form $R(\breve{a}_{ij,k}), S_1(\breve{a}_{ij,k}, b_j)$; this implies that one of the three components $R^0(x_0^0), S_1^{01}(x_0^0, \breve{y}_0^1)$ or $R^1(\breve{x}_0^1), S_1^{11}(\breve{x}_0^1, \breve{y}_0^1)$ or $R^1(\breve{x}_0^1), S_1^{10}(\breve{x}_0^1, y_0^0)$ of h_2^L is true. In fact, we will prove formally in Prop. 6.9 that the two queries have the same lineage expressions, $\Phi_{h_2^L}^{D^L} \equiv \Phi_{h_2}^D$, and therefore their probabilities are equal, $P_D(h_2) = P_{D^L}(h_2^L)$. To recap, the probabilities in D^L are as follows: all tuples in R^0 and T^0 have probability $1/2$, those in S_1^{11} have probability $1 - z_k$, for $k = 1, 4$ and all others have probability $1/3$ (including the tuples in R^1 and T^1).

For technical purposes, it is important to note that h_2^L is symbol-connected: reading its expression from left to right to left to right, the components are connected by the symbols $S_1^{01}, S_2^{01}, T^1, S_2^{11}, S_1^{11}, R^1, S_1^{10}, S_2^{10}$.

[8]Our choice of 1/3 is arbitrary, just to make it look different from $R^D(a_i)$ and $T^D(b_j)$, which must be set to 1/2.

Let us now compute $P(\neg h_2^L)$ on the database D^L. As before, to each truth assignment θ we associate the event E_θ that fixes the tuples in R^0 and T^0; as before, we have $P(\neg h_2^L) = 1/2^n \sum_\theta P(\neg h_2^L|E_\theta)$, so we need to compute $P(\neg h_2^L|E_\theta)$. Fix an edge $(i,j) \in E$. There are four slices connecting a_i and b_j, and $\neg h_2^L$ must be false in each of them. Fix a slice k; it consists of the following ten tuples:

$$\begin{array}{lll} R^0(a_i), & S_1^{01}(a_i, \breve{b}_{ij,k}), S_2^{01}(a_i, \breve{b}_{ij,k}), & T^1(\breve{b}_{ij,k}) \\ & S_1^{11}(\breve{a}_{ij,k}, \breve{b}_{ij,k}), S_2^{11}(\breve{a}_{ij,k}, \breve{b}_{ij,k}), & \\ R^1(\breve{a}_{ij,k}), & S_1^{10}(\breve{a}_{ij,k}, b_j), S_2^{10}(\breve{a}_{ij,k}, b_j), & T^0(b_j) \end{array}$$

As before, denote U, V the events $a_i \notin R^0$, $b_i \notin T^0$, and $Z_1, \ldots, Z_8$ the events $(a_i, \breve{a}_{ij}) \notin S_1^{01}$, $(a_i, \breve{b}_{ij}) \notin S_2^{01}$, $\breve{b}_{ij} \notin T^1$, $(\breve{a}_{ij,k}, \breve{b}_{ij,k}) \notin S_2^{11}$, $\ldots$, $(\breve{a}_{ij,k}, b_j) \notin S_2^{10}$ (reading the slice in left to right to left to right order). Then, the event that h_2^L is false in slice i, j, k is given by the following Boolean expressions:

$$Y_{UV} = (U \vee Z_1) \wedge (Z_1 \vee Z_2) \wedge \ldots \wedge (Z_7 \vee Z_8) \wedge (Z_8 \vee V) \tag{26}$$

By instantiating U, V in all possible ways to `true`, `false`, we obtain four Boolean expressions, $Y_{11}, Y_{12}, Y_{21}, Y_{22}$ (similar to Eq. 19). The probability of each Z_i is some constants (say, $P(Z_i) = 1/3$), except for the distinguished tuple, which is Z_5; here $P(Z_5) = P(\neg S_1(\breve{a}_{ij,k}, \breve{b}_{ij,k})) = z_k$, where k is the slice number. Therefore, for each $u, v \in [2]$, the probability of Y_{uv} is a simple linear function in z_k:

$$P(Y_{uv}) = a_{uv} \cdot z_k + b_{uv} \qquad\qquad u, v \in \{1, 2\}$$

where $a_{uv}, b_{uv} \in \mathbb{R}$ are eight constants. The probability that h_2^L is false in *all* four slices is y_{uv}:

$$y_{uv} = (a_{uv} \cdot z_1 + b_{uv}) \cdot (a_{uv} \cdot z_2 + b_{uv}) \cdot (a_{uv} \cdot z_3 + b_{uv}) \cdot (a_{uv} \cdot z_4 + b_{uv}) \tag{27}$$

because the four events Y_{UV} are independent, since any two slices share only the endpoint tuples $R^0(a_i)$ and $T^0(b_j)$ that are fixed for the given values of U, V. Therefore, $P(\neg h_2|E_\theta)$ is given by Eq. 22, and $P(\neg h_2)$ is given by Eq. 23, with the only difference that the four functions $y_{11}, \ldots, y_{22}$ in Eq. 20 are replaced with the four functions in Eq. 27. As before, we use an Oracle to compute $P(\neg h_2)$ repeatedly, form a system of $(m+1)^4$ linear equations, and compute all coefficients $\#\mathbf{k}$, then compute $\#\Phi$.

Next, we examine the Jacobian of the function $\bar{z} \mapsto \bar{y}$, which turns out to be Cauchy's double alternant. We need a few definitions. Throughout this paper, if F, G are two multivariate polynomials, we write $F = G$ to mean that they are identical, i.e. have the same coefficients.

Definition 5.4. *(1) Let F be a multivariate polynomial and x a variable. We say that F* depends *on x if x occurs in F. (2) Two multivariate polynomials F, G are* equivalent *if there exists a constant $c \neq 0$ s.t. $F = c \cdot G$.*

A linear polynomial $f(z) = a \cdot x + b$ depends on x iff $a \neq 0$. Two linear polynomials $a_1 \cdot x + b_1$ and $a_2 \cdot x + b_2$ are equivalent if either none depends on x or if they have the same root: $-b_1/a_1 = -b_2/a_2$.

Definition 5.5. *We call a set of linear polynomials $f_1(x), \ldots, f_m(x)$* non-degenerate *if each depends on x and no two are equivalent.*

PROPOSITION 5.6. *Let $f_1(x), \ldots, f_m(x)$ be a set of non-degenerate linear polynomials. Define the following m multivariate polynomials, where $x_1, \ldots, x_m$ are m distinct variables:*

$$F_i(x_1, \ldots, x_m) \;=\; f_i(x_1) \cdot f_i(x_2) \cdots f_i(x_n)$$

Then, for all $\bar{v} = (v_1, \ldots, v_m) \in R^m$ such that $v_i \neq v_j$ for $i \neq j$, the Jacobian is non-zero at $\bar{v}$:

$$\mathtt{det}(\frac{\partial \bar{F}}{\partial \bar{x}}(\bar{v})) \neq 0$$

PROOF. We can assume w.l.o.g. that $a_1 = \ldots = a_m = 1$. Then the polynomials can be written as $f_i(x) = x + y_i$, where $y_1, \ldots, y_n$ are distinct values (because the polynomials are inequivalent), and the Jacobian matrix is:

$$\frac{\partial \bar{F}}{\partial \bar{x}} \;=\; \left(\prod_{k \neq j} (x_k + y_i) \right)_{ij}$$

Recall that the *Cauchy's double alternant* [Krattenthaler 1999] is the determinant of the matrix $\left(\frac{1}{x_i + y_j}\right)_{ij}$, and its value is $\prod_{i<j}(x_i - x_j)(y_i - y_j) / \prod_{i,j}(x_i + y_j)$. Thus, we obtain:

$$det\left(\frac{\partial \bar{F}}{\partial \bar{x}}\right) \;=\; \prod_{i<j}(x_i - x_j)(y_i - y_j)$$

□

Thus, in order to complete the hardness proof for h_2, it suffices to prove that the four linear polynomials occurring in Eq. 27 are non-degenerate linear polynomials. To prove that, we need a third technique from algebra, irreducible polynomials.

5.4 Proof Technique 3: Irreducible Multivariate Polynomials

Let's revert our decision of setting $P(Z_1), \ldots, P(Z_8)$ to be constants, and setting $P(Z_5) = z_k$. Instead, we define each to be a distinct variable, $P(Z_i) = x_i$, $i = 1, 8$. Then, for each $u, v \in [2]$, $P(Y_{uv}) = F_{uv}(x_1, \ldots, x_8)$ is a multilinear polynomial in $x_1, \ldots, x_8$. If we substitute 7 of the variables with constants and keep only the variable x_i, then F_{uv} becomes a linear polynomial $P(Y_{uv}) = f_{uv}(x_i) = a_{uv} \cdot x_i + b_{uv}$. We will prove that it is possible to choose a variable x_i and constants for the other variables such that the four linear polynomials f_{uv} are non-degenerate.

Let F, G denote multilinear polynomials with k variables $x_1, \ldots, x_k$. Recall that equality $F = G$ means that two polynomials have exactly the same coefficients and should not be confused with equality at a particular point $\bar{u} \in \mathbb{R}^k$, which we write $F[\bar{u}] = G[\bar{u}]$. F is called *irreducible* if whenever $F = G \cdot H$, then either G or H is a constant. A classical theorem in algebra is that every multi-variate polynomial over the reals (in fact, over any field) admits a unique (up to equivalence) factorization as a product of irreducible polynomials. If x_i is a variable and $\bar{u} \in \mathbb{R}^{k-1}$, then we denote $F[\bar{u}](x_i)$ the linear polynomial in x_i obtained by substituting the other $k-1$ variables with $\bar{u}$. Clearly, $F[\bar{u}](x_i) = a \cdot x_i + b$, for some $a, b \in \mathbb{R}$.

DEFINITION 5.7. *Let $F_1, \ldots, F_m$ be multilinear polynomials with real coefficients over variables $x_1, \ldots, x_k$. Call a variable x_i* distinguished *if there exists $\bar{u} \in \mathbb{R}^{k-1}$ such that the set of linear polynomials $F_1[\bar{u}](x_i), \ldots, F_m[\bar{u}](x_i)$ is non-degenerate (Def. 5.5). We call $\bar{u}$ the* distinguishing values.

For example, in $F_1 = x_1 \cdot x_2$ and $F_2 = x_1 \cdot x_2 + x_1$, the variable x_2 is distinguished: $F_1[1](x_2) = x_2$ and $F_2[1](x_2) = x_2 + 1$, which are non-equivalent (they have different roots). The variable x_1 is not distinguished: set $x_2 = c$ for any value $c \in \mathbb{R}$, and $F_1[c](x_1) = c \cdot x_1$ and $F_2[c](x_1) = (c+1) \cdot x_1$, which are equivalent.

PROPOSITION 5.8. *Let $F_1, \ldots, F_m$ be irreducible, multi-linear polynomials, over the same k variables. Let x be any variable, s.t. every F_i depends on x, for $i = 1, m$. Then x is a distinguished variable iff $F_1, \ldots, F_m$ are inequivalent; moreover, there exists distinguishing values in $(0,1)^{k-1}$.*

PROOF. The only if direction is obvious: if F_i, F_j are equivalent, then for any variable x and for any values $\bar{u} \in \mathbb{R}^{k-1}$ of the other variables, $F_i[\bar{u}](x), F_j[\bar{u}](x)$, are also equivalent. We prove the if direction. Fix x any variable, and let $\bar{y}$ denote the other $k-1$ variables, different from x. Write $F_i(x) = A_i \cdot x - B_i$ where A_i, B_i are multilinear polynomials in $\bar{y}$. A_i is not identically 0, because F_i depends on x. We prove that for any distinct i, j, the polynomials $A_i \cdot B_j$ and $A_j \cdot B_i$ are not identical. Suppose otherwise, i.e. $A_i \cdot B_j = A_j \cdot B_i$. Consider their decomposition into prime factors. We can write each polynomial as:

$$A_i = U \cdot V \qquad B_j = W \cdot Z \qquad A_j = U \cdot Z \qquad B_i = V \cdot W$$

where U consists of the irreducible factors occurring in both A_i and A_j, V consists of the irreducible factors occurring in both A_i and B_i, etc. Then $F_i(x) = U \cdot V \cdot x - V \cdot W = V \cdot (U \cdot x - W)$ and similarly $F_j(x) = Z \cdot (U \cdot x - W)$. We know that $U \neq 0$, because $A_i \neq 0$ (and $A_j \neq 0$). Since both F_i and F_j are irreducible, both V and Z must be constants. But then F_i and F_j are equivalent, which is a contradiction. This proves the claim that $A_i \cdot B_j \neq A_j \cdot B_i$, i.e. the two polynomials are not identical.

Define the following polynomial:

$$H(\bar{y}) = (\prod_i A_i) \times (\prod_{i<j} (A_i \cdot B_j - A_j \cdot B_i))$$

Here $H(\bar{y})$ is a multivariate polynomial in $\bar{y}$, which is not identically zero, since we have shown that none of its factors is zero. Hence there are values $\bar{y} = \bar{v} \in (0,1)^{k-1}$ s.t. $H[v] \neq 0$. (Otherwise, if H is zero on an open set, then it is identically zero). We check now that the set of linear polynomials $F_1[\bar{v}](x), \ldots, F_m[\bar{v}](x)$ is non-degenerate. First, $A_i[\bar{v}] \neq 0$ implies that $F_i[\bar{v}](x) = A_i[\bar{v}] \cdot x - B_i[\bar{v}]$ depends on x. Next, forall $i \neq j$, $A_i[\bar{v}] \cdot B_j[\bar{v}] \neq A_i[\bar{v}] \cdot B_i[\bar{v}]$, which means that the two linear polynomials $F_i[\bar{v}](x)$ and $F_j[\bar{v}](x)$ are inequivalent. □

If F is a multilinear polynomial that depends on a variable x, then there exists a unique (up to equivalence) irreducible factor G of F that depends on x. Indeed, writing $F = G_1 \cdot G_2 \cdots G_n$, only one factor G_i may depend on x since the degree of x in F is 1.

COROLLARY 5.9. *Let $F_1, \ldots, F_m$ be multi-linear polynomials, and let x be a variable, s.t. every F_i depends on x, for $i = 1, m$. Denote G_i the irreducible factor of F_i that depends on x. Then x is distinguished iff $G_1, \ldots, G_m$ are inequivalent.*

The proof is immediate. To see an example, consider the two polynomials F_1, F_2 above; their factorizations are $x_1 \cdot x_2$ and $x_1 \cdot (x_2 + 1)$. The factors depending on x_1 are identical and therefore x_1 is not distinguished. The factors that depend on x_2 differ, and therefore x_2 is distinguished.

Using Corollary 5.9 we can complete the proof of the hardness of h_2. The last step is to establish the connection between the irreducible factors of $F = P(Y)$ and the structure of the Boolean expression Y. We will prove in Corollary 8.14 that $F = F_1 \cdot F_2 \cdots F_m$ has m factors iff Y can be written as $Y \equiv Y_1 \wedge Y_2 \wedge \ldots \wedge Y_m$ such that for all $i \neq j$, Y_i, Y_j do not share common Boolean variables. Let's examine the four Boolean expressions defined in Eq. 26:

$$\begin{aligned}
Y_{11} =& Z_1 \wedge [(Z_2 \vee Z_3) \wedge \ldots \wedge (Z_6 \vee Z_7)] \wedge Z_8 \\
Y_{12} =& Z_1 \wedge [(Z_2 \vee Z_3) \wedge \ldots \wedge (Z_6 \vee Z_7) \wedge (Z_7 \vee Z_8)] \\
Y_{21} =& [(Z_1 \vee Z_2) \wedge (Z_2 \vee Z_3) \wedge \ldots \wedge (Z_6 \vee Z_7)] \wedge Z_8 \\
Y_{22} =& [(Z_1 \vee Z_2) \wedge (Z_2 \vee Z_3) \wedge \ldots \wedge (Z_6 \vee Z_7) \wedge (Z_7 \vee Z_8)]
\end{aligned}$$

The corresponding multilinear polynomials $F_{uv} = P(Y_{uv})$, for $u, v \in \{1, 2\}$ have variables $x_1, \ldots, x_8$. Their irreducible factors are easily read from the four equations above. $F_{11} = x_1 \cdot G_{11} \cdot x_8$ has three irreducible factors. $F_{12} = x_1 \cdot G_{12}$ has two factors and so does $F_{21} = G_{21} \cdot x_8$. Finally, $F_{22} = G_{22}$ is irreducible. The irreducible factors $G_{11}, G_{12}, G_{21}, G_{22}$ are also inequivalent, because they correspond to inequivalent Boolean expressions. It follows that all their common variables, $x_2, \ldots, x_7$, are distinguished. On the other hand, x_1 is not distinguished, because it belongs to the same factor in F_{11} and in F_{12}, and similarly x_8 is not distinguished. Choose (arbitrarily) x_5 as the distinguished variable, rename it to z, and let $\bar{v} \in (0, 1)^7$ be distinguishing values of the other variables, given by Prop. 5.8. The four resulting linear polynomials $f_{uv}(z) = a_{uv} \cdot z + b_{uv}$ are inequivalent, proving that the Jacobian of the mapping $(z_1, \ldots, z_4) \mapsto (y_{11}, \ldots, y_{22})$ given in Eq. 27 is non-zero. This completes the proof that h_2 is #P-hard.

It is interesting to note that our proof is tight, in the following sense. Let h_4^- be obtained from h_k by removing a middle component:

$$h_4^- = R(x_0), S_1(x_0, y_0) \vee S_1(x_1, y_1), S_2(x_1, y_1) \vee S_3(x_3, y_3), S_4(x_3, y_3) \vee S_4(x_4, y_4), T(y_4)$$

Repeating the same construction leads to a Boolean formula $Y(U, V)$ where U and V are disconnected. More precisely, $Y(U, V) = Y^L(U) \wedge Y^R(V)$. Then each Y_{uv} is the conjunction of two independent Boolean formulas, and leads to the following factorization of the polynomials:

$$\begin{array}{llll}
Y_{11} = Y_1^L \wedge Y_1^R & Y_{12} = Y_1^L \wedge Y_2^R & Y_{21} = Y_2^L \wedge Y_1^R & Y_{22} = Y_2^L \wedge Y_2^R \\
F_{11} = F_1^L \cdot F_1^R & F_{12} = F_1^L \cdot F_2^R & F_{21} = F_2^L \cdot F_1^R & F_{22} = F_2^L \cdot F_2^R
\end{array}$$

No variable z_i is a distinguishing variable, because each occurs in two identical factors. Thus, the hardness proof fails for h_4^-, and this is to be expected because

h_4^- is in PTIME, since algorithm 2 succeeds on h_4^-.

6. QUERY LEVELING

Recall from Def. 2.3 that, for a query expression Q, a *level* Z is a set of attributes that are joined, directly or indirectly, by the query.

DEFINITION 6.1. *A disjunctive query is called* leveled *if every level Z contains at most one attribute from every relational symbol R.*

EXAMPLE 6.2. *All queries h_k in Example 4.6 are leveled. For example, the query $h_1 = R(x_0), S(x_0, y_0) \vee S(x_1, y_1), T(y_1)$ has two levels, $X = \{(R, 1), (S, 1)\}$ and $Y = \{(S, 2), (T, 1)\}$, and the two attributes of S are on distinct levels. Similarly, queries q_W, q_9 in Fig. 1, Fig. 2 are leveled. The following two queries are not leveled:*

$$d_1 = R(x, y), R(y, z)$$
$$d_2 = R(x, y), S(y, z) \vee R(x', y'), S(x', y')$$

Indeed, both d_1 and d_2 have a single level.

The main result in this section is:

THEOREM 6.3 LEVELING. *Let d be an immediately unsafe query. Then there exists a leveled query d^L, which is also immediately unsafe, s.t. $d^L \leq^{FO}_{prob} d$. The query d^L is over a new vocabulary, whose maximum arity is the same as the maximum arity of the vocabulary for d.*

In general, if an unsafe query is not leveled, then it may not be possible to further simplify it using the rules in Def. 4.13. For example, consider d_1: if we attempt a shattering rewrite rule, $d \to d_{1,a}$, by shattering the relation R into R_{*a}, then the shattered query is $d_{1,a} = R(x, a), R(a, a)$, which is safe. The purpose of leveling is to allow us to simplify the query, as we will show in the next section. The idea is to specialize d to a *leveled structure*, which generalizes the notion of a k-partite graph. We illustrate this on an example.

EXAMPLE 6.4. *We will prove that the query $d_1 = R(x, y), R(y, z)$ is #P-hard, by showing that $h_1 \leq^{FO}_{lin} d_1$. The input to d_1 is a graph $R(A, B)$. Our idea is to restrict the input graph to be a k-partite graph; the question is how large should k be. If $k = 2$ then the graph is bipartite, and $d_1 \equiv$ `false` (there are no paths of length 2). If $k = 3$ then the 3-partite graph has three sets of nodes A, B, C, and two kinds of edges $R^{12}(A, B)$, $R^{23}(B, C)$ (we assume there are no edges from A to C): the query specializes to $d_1^L = R^{12}(x, y), R^{23}(y, z)$, which is a safe query, so this still does not help us prove hardness. Consider $k = 4$; there are three kinds of edges, $R^{12}(A, B), R^{23}(B, C), R^{34}(C, D)$, and the query specializes to $d_1^L = R^{12}(x_0, y_0), R^{23}(y_0, z_0) \vee R^{23}(y_1, z_1), R^{34}(z_1, u_1)$. If we shatter levels x_0 and u_1 to constants a, b respectively, then $d_1^L \to d_1^L[a/x_0, b/u_1] = R^{12}(a, y_0), R^{23}(y_0, z_0) \vee R^{23}(y_1, z_1), R^{34}(z_1, b)$, which is essentially h_1 (see Example 4.6), up to the renaming of the relation names. Thus, $h_1 \leq^{FO}_{lin} d$, proving that d is #P-hard.*

6.1 Connection Between a Level and a Separator

We first need to introduce a technical result.

PROPOSITION 6.5. *Let d be a ranked, disjunctive query that is symbol-connected. Then d has a separator iff there exists a level Z that contains only root variables. Moreover, in this case the set of variables in level Z forms a separator.*

PROOF. We start with the "only if" direction. Write d as $d = \exists z.(c_1 \vee \ldots \vee c_m)$ where z is a separator variable. Let Z be the set of attributes where z occurs. Write d equivalently as $d' = \exists x_1.c_1[x_1/z] \vee \ldots \vee \exists x_m.c_m[x_m/z]$; we show that Z is a level in d'. Suppose $(R,i),(S,j)$ is an edge in the attribute graph; then there is a join predicate in d, $R(\ldots), S(\ldots)$, which joins positions i and j. If (R,i) is in Z, then the atom $R(\ldots)$ contains z on that position, and, therefore, so must the atom $S(\ldots)$; correspondingly, in d' both contain a common variable x_k, proving that $(S,j) \in Z$. It also follows that Z has only root variables, namely $x_1, \ldots, x_k$, which form the separator.

To prove the "if" direction, assume Z is a level that contains only root variables. Then Z contains at least one variable from each component c_i, because the query is symbol-connected; also, Z contains at least one attribute from each relational symbol R. We prove that it contains exactly one attribute from each symbol R. For each R, let i_R be the smallest attribute in Z: that is, $(R,i_R) \in Z$, and, if $(R,j) \in Z$ then $i_R \leq j$. Call (R,i_R) the minimal Z-attribute in R. We claim that if a minimal Z-attribute (R,i_R) is connected to some other attribute (S,j) in the attribute graph, then (S,j) is also a minimal Z-attribute. Indeed, consider a component containing $R(\ldots), S(\ldots)$ and a common variable x on position i_R in R and on position j in S. By assumption, x is a root variable. Suppose $i_S < j$: then the variable y on position i_S in S is also a root variable, and must also occur in R, on some position i, and obviously $(R,i) \in Z$. Because the query is ranked (Def. 4.1), we must have $i < i_R$, contradicting the fact that i_R was minimal. Thus, Z contains exactly one attribute in each relation, and exactly one root variable in each component. We prove now that it is a separator. Let x_i be the root variable in c_i on level Z. Write d as $\exists z.(c_1[z/x_1] \vee c_2[z/x_2] \vee \ldots)$. Now z is a separator variable: it is clearly a root variable, and in every atom with relation symbol R, the variable z occurs precisely on the attribute that is at the level Z. □

In the rest of the paper we will use the criteria in Prop. 6.5 as the definition of a separator in a leveled query, instead of the official Def. 3.2.

The condition that the query is ranked is essential. To see a counterexample, consider $R(x,y), R(y,x)$. This query is not ranked, and there is a single single level containing both attributes $(R,1)$ and $(R,2)$. Therefore this level contains the variables x,y, and thus contains only root variables; yet, they do not form a separator.

6.2 Leveled Database Instances

Let $\mathbf{R}$ be a relational vocabulary. Fix a set L, and call its elements *types*, or *level types*. We assume $|L| \geq arity(R)$ for every $R \in \mathbf{R}$. A *leveling* of a relation R of arity k is an injective function $\bar{\tau} : [k] \to L$. We denote with $R^{\bar{\tau}}$ the leveled relational symbol, of the same arity as R. A *leveled vocabulary*, $\mathbf{R}^L$, is a set of

leveled relational symbols R^τ. We say that $\mathbf{R}^L$ is *complete* if it contains all leveled relational symbols $R^{\bar{\tau}}$, where $R \in \mathbf{R}$ and $\bar{\tau}$ is a leveling for R.

Let D^L be a database instance over $\mathbf{R}^L$. Given $\tau \in L$, denote $ADom_\tau(D^L)$, or simply $ADom_\tau$ when D^L is clear from the context, the active domain of all attributes having level type τ. We say that D^L is a *leveled database instance* if for all $\tau_1 \neq \tau_2$, $ADom_{\tau_1} \cap ADom_{\tau_2} = \emptyset$. We will assume throughout the paper that any database instance over a leveled vocabulary is leveled.

For example, a bipartite graph $R(A, B)$ is a leveled structure, because we may assume that the active domains of A and B are disjoint. A 3-partite graph may have several vocabularies: one example is $R^{12}(A, B)$, $R^{23}(B, C)$, where edges go only from A to B and from B to C; another example is $R^{12}(A, B)$, $R^{23}(B, C)$, $R^{31}(C, A)$, with three kinds of edges, from A to B, from B to C, and from C to A. In both cases, the 3-partite graph is leveled: the active domains of A, B, and C are disjoint.

DEFINITION 6.6. *The* level-forgetting mapping *maps an instance D^L over a leveled vocabulary $\mathbf{R}^L$ to the instance $D = \mathbf{F}(D^L)$ over $\mathbf{R}$ defined as: $R^D = \bigcup_{\bar{\tau}} (R^{\bar{\tau}})^{D^L}$, for all $R \in \mathbf{R}$.*

That is, the mapping simply takes the union of all different levelings $R^{\bar{\tau}}$ of R. Notice that the union is disjoint, any two distinct levelings $R^{\bar{\tau}_1}, R^{\bar{\tau}_2}$ have at least one attribute on different levels, and these have disjoint active domains.

DEFINITION 6.7. *Let d^L be a query on the leveled vocabulary $\mathbf{R}^L$. The* level-forgetting query*, d, is the query over $\mathbf{R}$ obtained from d^L by forgetting all levelings of all relational symbols.*

LEMMA 6.8. *Let d be the level-forgetting query for d^L and D be the level-forgetting instance for D^L. Then $\Phi_{d^L}^{D^L} \Rightarrow \Phi_d^D$.*

PROOF. Suppose $W^L \models d^L$ for a world $W^L \subseteq D^L$. Then there exists a component c_i^L of d^L and a valuation $\theta : c_i^L \to W^L$. This is also a valuation $c_i \to W$, proving that $W \models c_i$. □

Note that the converse, $\Phi_d^D \Rightarrow \Phi_{d^L}^{D^L}$, does not hold in general. For example, consider the 3-partite graph $R^{12}(A, B)$, $R^{23}(B, C)$ and the leveled query $d^L = R^{12}(x, y)$, which simply checks that $R^{12} \neq \emptyset$. The level-forgetting query is $d = R(x, y)$ which checks that the entire graph is non-empty: the former implies the latter but not conversely.

6.3 General Query Leveling

Leveling takes a query d, and a leveled vocabulary, and transforms d into a leveled query d^L over that vocabulary, such that d is equivalent to the level-forgetting query for d^L, and the converse of Lemma 6.8 holds, i.e. $\Phi_{d^L}^{D^L} \equiv \Phi_d^D$. In particular, $d^L \leq_{\text{lin}}^{\text{FO}} d$.

We start by showing how to level a component, c. Let $\mathbf{R}$ be the vocabulary used by the query c, and let $\mathbf{R}^L$ be any leveled vocabulary. Let $\bar{\rho} : Var(c) \to L$ be a function from variables to level types. For each atom $R(x_1, \ldots, x_k)$, define $\bar{\tau}$ to be the following function $[k] \to L$: $\bar{\tau}(i) = \bar{\rho}(x_i)$ for $i = 1, \ldots, k$. We say

that $\bar{\rho}$ is a *leveling* of c if for all atoms $R(\ldots)$, the function $\bar{\tau}$ is injective[9], and the vocabulary $\mathbf{R}^L$ contains the symbol $R^{\bar{\tau}}$. Denote $c^{\bar{\rho}}$ the component over the vocabulary $\mathbf{R}^L$ obtained by replacing each atom $R(\ldots)$ with $R^{\bar{\tau}}(\ldots)$. The *leveling* of c is $c^L = \bigwedge_\rho c^{\bar{\rho}}$, where $\bar{\rho}$ ranges over all levelings of c. Notice that, if we forget the leveling of $c^{\bar{\rho}}$ we obtain back the query c: this implies that if we forget the leveling in c^L we obtain $c \vee c \vee \ldots \equiv c$. If $d = c_1 \vee \ldots \vee c_m$ is a disjunctive query, then its leveling is $d^L = c_1^L \vee \ldots \vee c_m^L$.

We prove that if d^L is the leveling of d, then the converse of Lemma 6.8 holds. We illustrate with a brief example, on the 3-partite graph $R^{12}(A,B)$, $R^{23}(B,C)$; the leveling of the query $d = R(x,y)$ is $d^L = R^{12}(x_1,y_1) \vee R^{23}(x_2,y_2)$; both queries check whether the entire graph is non-empty, and, thus, are equivalent (up to the level-forgetting mapping $D^L \mapsto D$).

PROPOSITION 6.9. *Let d^L be the leveling of d over the leveled vocabulary $\mathbf{R}^L$. Let D^L be a leveled instance, and D be obtained from D^L through the level-forgetting mapping. Then $\Phi_{d^L}^{D^L} \equiv \Phi_d^D$. It follows that $d^L \leq_{lin}^{FO} d$.*

PROOF. By Lemma 6.8, $\Phi_{d^L}^{D^L} \Rightarrow \Phi_d^D$. For the converse, let $W \subseteq D$ be s.t. $W \models d$. Then there exists i s.t. $W \models c_i$. Let $\theta : Var(c_i) \to ADom$ be the valuation that maps the variables in c_i to the active domain, such that for every atom g in c_i, $\theta(g)$ is a tuple in W. For each variable x_i, define $\bar{\rho}(x_i)$ to be the unique level l s.t. $\theta(x_i) \in ADom_l$. Denoting $W' = \mathbf{F}^{-1}(W)$ (this separates the tuples in W' by their level annotation) we have $W' \models c_i^{\bar{\rho}}$. □

We will use Prop. 6.9 in several places in this paper. It holds for *any* leveled vocabulary $\mathbf{R}^L$, the only requirement is that the query d^L be leveled according to our procedure. We have already used leveling in Sect. 5.3, where we leveled the query h_2 and obtained h_2^L. The leveled vocabulary was incomplete, in that it included relations $S_1^{14}, S_1^{34}, S_1^{32}$ but did not include, e.g, S_1^{12}; but the query h_2^L was leveled according to the procedure described in this section.

We now prove that d^L has several nice properties: it is minimized, and it is leveled (as per Def. 6.1).

LEMMA 6.10. *Let $d = \bigvee_i c_i$. Then for any c_i and any $\bar{\rho} : Var(c_i) \to L$, $c_i^{\bar{\rho}}$ is a minimized, non-redundant component of d^L.*

PROOF. If $c_i^{\bar{\rho}}$ is not minimized then there exists a homomorphism from $c_i^{\bar{\rho}}$ into a strict subset of its atoms: this extends to a homomorphism from c_i into the same subset, by simply forgetting the leveling, contradicting the fact that c_i is minimized. Similarly, any homomorphism $c_1^{\bar{\rho}_1} \to c_2^{\bar{\rho}_2}$ extends to a homomorphism $c_1 \to c_2$ by simply forgetting the leveling, showing that each component is non-redundant. □

LEMMA 6.11. *Let d^L be the leveling of a disjunctive query d. Then for every level Z, all attributes in Z have the same level type.*

PROOF. Let $(R^{\bar{\tau}}, i)$ and $(S^{\bar{\sigma}}, j)$ be two attributes that are connected in the attribute graph of d^L. By definition of the attribute graph, there exists a component $c^{\bar{\rho}}$ that contains two atoms $R^{\bar{\tau}}(\ldots)$ and $S^{\bar{\sigma}}(\ldots)$ that have the same variable x on

[9]This is always the case if c is ranked (Def. 4.1).

positions i and j respectively. By construction, $\bar{\tau}(i) = \bar{\rho}(x) = \bar{\sigma}(j)$, proving that $(R^{\bar{\tau}}, i)$ and $(S^{\bar{\sigma}}, j)$ have the same level type. □

COROLLARY 6.12. *Let d^L be the leveling of d. Then d^L is leveled.*

PROOF. Let $(R^{\bar{\tau}}, i)$ and $(R^{\bar{\tau}}, j)$ be two attributes of the same relation symbol $R^{\bar{\tau}}$ that belong to a common level Z. By the previous lemma, they have the same level type, $\bar{\tau}(i) = \bar{\tau}(j)$. It follows that $i = j$. □

EXAMPLE 6.13. *We briefly illustrate leveling on d_1, d_2 in Example 6.2. Choose $L = 4$ and the leveled vocabulary $R^{12}, R^{23}, S^{23}, S^{34}$. Their levelings are:*

$$d_1^L = R^{12}(x^1, y^2), R^{23}(y^2, z^3) \vee R^{23}(x^2, y^3), R^{34}(y^3, z^4)$$
$$d_2^L = R^{23}(x^2, y^3), S^{34}(y^3, u^4) \vee R^{12}(x^1, y^2), S^{23}(y^2, z^3) \vee R^{23}(x^2, y^3), S^{23}(x^2, y^3)$$

Each expression is minimized, leveled, and we have $d_1^L \leq_{lin}^{FO} d_1$ and $d_2^L \leq_{lin}^{FO} d_2$. Therefore, if d_1^L, d_2^L are #P-hard, then so are d_1, d_2.

So far we did not require d to be ranked: these results hold even for unranked queries. For example, suppose $d = R(x, y), R(y, x)$, and $L = 2$. Then $d^L = R^{12}(x, y), R^{21}(y, x) \vee R^{21}(x, y), R^{12}(y, x) \equiv R^{12}(x, y), R^{21}(y, x)$. The new query is leveled (there are two levels: $\{(R^{12}, 1), (R^{21}, 2)\}$ and $\{(R^{12}, 2), (R^{21}, 1)\}$), and $d^L \leq_{\text{lin}}^{\text{FO}} d$. In fact, it is even ranked, if one switches the order of the attributes in R^{21}. More generally, one can prove that every leveled query is ranked. However, leveling is not a substitute for Prop. 4.2, because the reduction in Prop. 6.9 is only in one direction: in general, it is not the case that $d \leq_{\text{prob}}^{\text{FO}} d^L$. For an extreme example, consider the unranked query $d = R(x, x)$. It has no leveling at all, hence $\mathbf{R}^L = \emptyset$, and the leveling is $d^L = \bigvee_{\emptyset}() = \texttt{false}$. It still holds that $d^L \leq_{\text{lin}}^{\text{FO}} d$, because the only database instance D^L over the empty vocabulary is the empty tuple, leading to $\mathbf{F}(D^L) = D = \emptyset$, and $\Phi^{()}_{\texttt{false}} = \Phi^{\emptyset}_d$. But $d \not\leq_{\text{lin}}^{\text{FO}} d^L$.

6.4 Leveling Immediately Unsafe Queries

The next proposition proves Theorem 6.3. Recall that, by definition, an immediately unsafe query is by definition ranked.

PROPOSITION 6.14. *Let $d = c_1 \vee \ldots \vee c_m$ be an immediately unsafe query over vocabulary $\mathbf{R}$. Let L be such that $|L| \geq |Var(c_i)|$ for all $i = 1, m$, and let $\mathbf{R}^L$ be the complete leveling vocabulary for $\mathbf{R}$. Then any symbol-component of d^L is immediately unsafe.*

PROOF. While d is symbol-connected by definition, d^L might not be[10]. Let d' be any symbol component of d^L. Assume it has a separator; by Prop. 6.5 there exists a level Z' that contains only root variables. The level Z' is a set of attributes in $\mathbf{R}^L$, $(R^{\bar{\tau}}, i)$; let's denote Z to be the set of attributes from the $\mathbf{R}$ vocabulary obtained by dropping the leveling: $Z = \{(R, i) \mid (R^{\bar{\tau}}, i) \in Z'\}$. We prove that (a) Z is a level, and (b) it contains only root variables. By Prop. 6.5 it is a separator for d (here we need d to be ranked), which is a contradiction, proving that d' has no separator.

[10]For a trivial example, the 2-leveling of $d = R(x)$ is $d' = R^1(x^1) \vee R^2(x^2)$ and is not symbol-connected.

To prove (a), let $(R, i) \in Z$ and suppose there exists a component c_k that joins attribute (R, i) with (S, j). Denote $g_1 = R(\ldots)$ and $g_2 = S(\ldots)$ the two atoms that have a common variable x on positions i and j respectively. By definition of Z, there exists an attribute $(R^{\bar\tau}, i) \in Z'$. Consider the leveling $\rho : Var(c_k) \to L$ of the component c_k defined as follows: ρ maps the variables in $g_1 = R(\ldots)$ according to $\bar\tau$, and maps all other variables in c_k to distinct level-types. That is, we choose ρ to be injective and this is possible by $|L| \geq |Var(c_k)|$. Consider the component c_k^ρ: it must appear in d^L (because the vocabulary $\mathbf{R}^L$ is complete); it belongs to d' (because it contains the symbol $R^{\bar\tau}$ hence it is symbol-connected to d'); and it contains the two atoms $R^{\bar\tau}(\ldots)$ and $S^{\bar\sigma}(\ldots)$ that join the attributes $(R^{\bar\tau}, i)$ and $(S^{\bar\sigma}, j)$. (Here $\bar\sigma$ is the leveling imposed by $\bar\rho$ on the atom g_2.) This proves that $(S^{\bar\sigma}, j) \in Z'$, and therefore $(S, j) \in Z$.

To prove (b), suppose $(R, i) \in Z$ and there exists a component c_k that has an atom $g = R(\ldots)$ where the variable x on position i is not a root variable. As before, there exists $(R^{\bar\tau}, i) \in Z'$, and we can extend $\bar\tau$ to a leveling of the entire component c_k, $\rho : Var(c_k) \to L$. Then x is not a root variable in c_k^ρ either, and it occurs on position i of an atom $R^{\bar\tau}(\ldots)$, contradicting the assumption that Z' contains only root variables. □

It is interesting to notice that the proof in the proposition leads to a different leveling those shown in Example 6.13. For example, it levels $R(x, y), R(y, z)$ by using only three levels, leading to the following:

$$\begin{aligned}
&R^{12}(x^1, y^2), R^{23}(y^2, z^3) \vee R^{23}(x^2, y^3), R^{31}(y^3, z^1) \vee R^{31}(x^3, y^1), R^{12}(y^1, z^2)\\
\vee &R^{13}(x^1, y^3), R^{32}(y^3, z^2) \vee R^{32}(x^3, y^2), R^{21}(y^2, z^1) \vee R^{21}(x^2, y^1), R^{13}(y^1, z^3)\\
\vee &R^{12}(x^1, y^2), R^{21}(y^2, z^1) \vee R^{13}(x^1, y^3), R^{31}(y^3, z^1) \vee R^{21}(x^2, y^1), R^{12}(y^1, z^2)\\
\vee &R^{13}(x^1, y^3), R^{31}(y^3, z^1) \vee R^{31}(x^3, y^1), R^{13}(y^1, z^3) \vee R^{32}(x^3, y^2), R^{23}(y^2, z^3)
\end{aligned}$$

This query has no separator, as can be seen immediately from the first two components of the first line: their root variables are y^2 and y^3 respectively, but they occur on different positions in R^{23}.

7. REWRITING TO A FORBIDDEN QUERY

The rewrite rules $\to$ in Def. 4.13 never get stuck: one can always apply another step, e.g. choose any relational symbol R, and shatter it into $R_{a*\cdots*}$ and $R_{*\cdots*}$. While we could have defined the rewrite rules in a more restricted fashion, to always guarantee termination, our more general rules ensure that queries can be simplified to some quite elegant forms. For example in Sect. 8.2 we show that every non-hierarchical query can be rewritten to $h_0 = R(x), S(x, y), T(y)$ (in other words h_0 is the only forbidden query!), while in Section 8.4.7 we make critical use of the full power of the rewrite rules, in order to prove hardness of the most difficult forbidden queries. In this section, however, our goal is prove that a query can be simplified, and for that we need to define a measure of progress. Intuitively, Q_1 is simpler than Q_2, in notation $Q_1 < Q_2$, if it either is over a vocabulary with smaller arities, or is over the same vocabulary but has fewer atoms.

Given a query Q, define its sequence to be $seq(Q) = \mathbf{a} = (a_0, a_1, a_2, \ldots) \in \mathbb{N}^{\mathbb{N}}$ where a_k denotes the number of relation symbols of arity k occurring in Q. The

sequence has finite support, i.e. $a_i = 0$ for all but finitely many values of i. Consider the lexicographic order $\mathbf{a} < \mathbf{b}$ if $\exists i$ s.t. $a_i < b_i$ and $\forall j > i$, $a_j = b_j$. It is known that this is a well ordering, meaning that there is no infinitely decreasing sequence $\mathbf{a}_1 > \mathbf{a}_2 > \ldots$ Let $at(Q)$ denote the set of atoms in the minimized CNF expressions of Q (Sect. 2.6).

DEFINITION 7.1. *Let Q_1, Q_2 be two queries, possibly over different vocabularies. We define the partial order*[11] *$Q_1 < Q_2$ if $seq(Q_1) < seq(Q_2)$ or $seq(Q_1) = seq(Q_2)$ and $|at(Q_1)| < |at(Q_2)|$.*

Then $Q_1 < Q_2$ is a well ordering. Indeed, the lexicographic order on sequences of natural numbers is a well ordering and, furthermore, the lexicographic ordering of two well orderings is also a well ordering.

DEFINITION 7.2. *A* forbidden query *is a leveled, unsafe query Q such that for any maximal rewriting $Q \stackrel{*}{\to} Q'$ where $Q > Q'$, the query Q' is safe.*

Thus, by definition, if Q is unsafe and not forbidden, then $Q \stackrel{*}{\to} Q'$ to some other unsafe query $Q' < Q$. By repeating this argument for Q', we eventually reach a forbidden query, because $<$ is a well ordering. Thus, *every* unsafe query Q rewrites to a forbidden query.

A forbidden query Q is not only unsafe, but it is even immediately unsafe: otherwise, we apply one step of algorithm 2 and obtain a maximal rewriting $Q \to Q'$ where Q' is also unsafe and $Q > Q'$ (since at each step the algorithm either reduces the number of atoms in the query, or shatters d to $d[a/z]$ thus reducing the query's sequence). Thus, a forbidden query is a disjunctive query, and we will denote it d. The converse fails in general because, if d is immediately unsafe, it only means that the algorithm is stuck, but we may still make progress by applying the rewriting rules. For example, $d = R(x_0, z_0), S(x_0, y_0) \vee S(x_1, y_1), T(y_1)$ is immediately unsafe, but it is not a forbidden, because we can shatter it $d \to R_{*a}(x_0), S_{**}(x_0, y_0) \vee S_{**}(x_1, y_1), T_*(y_1)$, which still unsafe.

How simple is a forbidden query? We prove in this section:

THEOREM 7.3. *A forbidden query has exactly two levels.*

In the next section we prove that any forbidden query d is #P-hard. This implies that every unsafe query is hard, because any unsafe query Q has some maximal rewriting to some forbidden query $Q \stackrel{*}{\to} d$, and, by Corollary 4.19, $d \leq^{\text{FO}}_{\text{prob}} Q$, thus Q is #P-hard. Notice that we do not need to worry about the length of the reduction $Q \stackrel{*}{\to} d$, and not even whether it is decidable to find a forbidden query d from Q. To prove that Q is #P-hard it suffices to show that such a d exists, because the complexity of the reduction is measured in terms of the size of the database. Thus, for the purpose of the hardness proof for Q, computing the forbidden query d from Q can be done in time $O(1)$ in the size of the database.

In the rest of this section we prove Theorem 7.3.

[11] We do not define $\leq$, and only use $<$; if $seq(Q_1) = seq(Q_2)$ and $|at(Q_1)| = |at(Q_2)|$ then, in general, there is no relationship between Q_1 and Q_2.

7.1 Shattering of a Level Z

To prove Theorem 7.3, we will show that if Q has ≥ 3 levels, then there exists a shattering rewriting $Q \to Q_A$ such that $Q' < Q$. It turns out that for that we only need a restricted kind o shattering, which shatters an entire level Z with the constants A.

DEFINITION 7.4. *Let Q be a query. The* shattering of a level Z with the constants A, *denoted $Q_{A/Z}$, is the query obtained by shattering Q w.r.t. the following vocabulary. Every attribute in Z shatters to $a_1, a_2, \ldots, a_k$ (but not to $*$), and every attribute not in Z shatters only to $*$.*

We give an alternative definition, and describe it only for a disjunctive query, $d = \bigvee_i c_i$, since we only need to apply shattering to disjunctive queries in this section. Fix a level Z, and let $Var_Z(c_i)$ is the set of variables in c_i that are in level Z. Let $A = \{a_1, a_2, \ldots\}$ be a set of constants; we will always take $|A| \leq 2 \cdot Var(d)$.

$$\begin{aligned} \Theta_Z(c_i, A) =& \{\theta \mid \theta : Var_Z(c_i) \to A\} \\ d[A/Z] =& \bigvee_i \bigvee_{\theta \in \Theta_Z(c_i, A)} c_i[\theta] \end{aligned} \tag{28}$$

Thus, $\Theta_Z(c_i, A)$ is the set of substitutions of Z-variables, and $d[A/Z]$ is obtained by substituting the Z-variables in all possible ways with constants from A. By construction, the complete shattering of $d[A/Z]$ is precisely $d_{A/Z}$ (see the definition of the complete shattering right before Prop. 2.10). Thus, throughout this section, we will blur the distinction between shattering of a level Z, $d_{A/Z}$, and substituting level Z with constants, $d[A/Z]$. To prove Theorem 7.3 we show:

PROPOSITION 7.5. *If d is leveled and immediately unsafe, with ≥ 3 levels, then there exists a level Z s.t. for any set of constants A, if $|A| \leq 2 \cdot |Var_Z(d)|$ then $d[A/Z]$ is unsafe.*

This implies Theorem 7.3. Indeed, consider a forbidden query d: it is also immediately unsafe, hence, if it has ≥ 3 levels, then by the proposition above $d \to d[A/Z]$. We claim that $seq(d[A/Z]) < seq(d)$, contradicting the fact that d is forbidden: indeed, if k is the largest arity of a relation that contains an attribute on level Z, then the vocabulary of $d[A/Z]$ has at least one fewer relation or arity k, and the same number of relations of arity $> k$. Before proving the proposition, we illustrate it with two examples.

EXAMPLE 7.6. *Consider $d = R(x_0), S(x_0, y_0) \vee S(x_1, y_1), T(y_1, z)$. It has three levels, containing the variables $\{x_0, x_1\}$, $\{y_0, y_1\}$, $\{z\}$. Apply the rewrite rule $d \to d[a/z] = R(x_0), S(x_0, y_0) \vee S(x_1, y_1), T(y_1, a)$. After shattering, $d[a/z]$ is isomorphic to h_1, and by Lemma 4.18 $h_1 \leq^{FO}_{prob} d$. Therefore d is #P-hard.*

For a more difficult example, consider proving that $d = R(x_1, y_1), S(x_1, z_1) \vee R(x_2, y_2), T(y_2, z_2) \vee S(x_3, z_3), T(y_3, z_3)$ is #P-hard. Let $Z = \{z_1, z_2, z_3\}$ be the level that contains the variables z_1, z_2, z_3. Suppose we rewrite $d \to d[a/Z] = R(x_1, y_1), S(x_1, a) \vee R(x_2, y_2), T(y_2, a) \vee S(x_3, a), T(y_3, a)$. After shattering, the query becomes $d = R(x_1, y_1), S_{*a}(x_1) \vee R(x_2, y_2), T_{*a}(y_2) \vee S_{*a}(x_3), T_{*a}(y_3)$. But this query is essentially Q_V (Example 3.5), up to renaming of relation symbols,

and is safe. So substitution with one *constant was insufficient for proving hardness. However, the idea works if we substitute with* two *constants. The shattered query is:*

$$\begin{aligned} d[\{a,b\}/Z] =& R(x_1,y_1), S_{*a}(x_1) \vee R(x_1',y_1'), S_{*b}(x_1') \\ & \vee R(x_2,y_2), T_{*a}(y_2) \vee R(x_2',y_2'), T_{*b}(y_2') \\ & \vee S_{*a}(x_3), T_{*a}(y_3) \vee S_{*b}(x_3'), T_{*b}(y_3') \end{aligned}$$

Let's see why the query on the right is unsafe. Consider its CNF expression, which is a conjunction of four disjunctive queries. It suffices to show that one of them is unsafe, and we do this for:

$$\begin{aligned} d' =& R(x_1,y_1), S_{*a}(x_1) \vee R(x_1',y_1'), S_{*b}(x_1') \\ & \vee R(x_2,y_2), T_{*a}(y_2) \vee R(x_2',y_2'), T_{*b}(y_2') \\ & \vee S_{*a}(x_3) \vee T_{*b}(y_3') \\ \equiv& R(x_1',y_1'), S_{*b}(x_1') \vee R(x_2,y_2), T_{*a}(y_2) \vee S_{*a}(x_3) \vee T_{*b}(y_3') \end{aligned} \tag{29}$$

We can get rid of the last two components by rewriting $d' \rightarrow d'[S_{*b} = \texttt{false}, T_{*b} = \texttt{false}]$*, and the resulting query is isomorphic to* h_1*. Thus, we have the following rewriting rules:* $d \rightarrow d[\{a,b\}/Z] \rightarrow d' \rightarrow h_1$*, which is a maximal rewriting, hence* $h_1 \leq^{FO}_{prob} d$ *by Corollary 4.19, and* d *is #P-hard. We will return to this query in Example 7.13.*

7.2 The CNF Expression for $d[A/Z]$

Fix $d = c_1 \vee \ldots \vee c_m$. In general, $d[A/Z]$ (Eq. 28) is not a disjunctive query, because a query $c_i[\theta]$ may be disconnected. To prove that $d[A/Z]$ is unsafe, we must convert it to CNF first.

A *Z-subcomponent* of c_i is a connected component of the following graph: the nodes are the atoms in c_i, and edges are pairs of atoms that share at least one variable that is not in Z. Denote $sc_Z(c_i) = \{s_1, \ldots, s_m\}$ the Z-subcomponents of c_i. In other words, if we apply one of the substitutions θ in Eq. 28, then c_i decomposes into the components $s_1, \ldots, s_m$:

$$c_i[\theta] = \bigwedge_{s \in sc_Z(c_i)} s[\theta] \tag{30}$$

The DNF expression for $d[A/Z]$ is:

$$d[A/Z] = \bigvee_{i, \theta \in \Theta_Z(c_i, A)} \bigwedge_{s \in sc_Z(c_i)} s[\theta] \tag{31}$$

Next, we apply the standard conversion from DNF to CNF. Let k be a function $k : \prod_i \Theta_Z(c_i, A) \rightarrow \bigcup_i sc_Z(c_i)$ such that $\forall i, \theta \in \Theta_Z(c_i, A)$, $k(i,\theta) \in sc_Z(c_i)$. Let K be the set of all such functions. Then the CNF expression is given by:

$$d[A/Z] = \bigwedge_{k \in K} d'_k \tag{32}$$

$$\text{where: } d'_k = \bigvee_{i=1,m} \left(\bigvee_{\theta \in \Theta_Z(c_i,A); s := k(i,\theta)} s[\theta] \right) \qquad \text{for all } k \in K \tag{33}$$

The CNF expression given above is, in general, not minimized, as we saw in Example 7.6, Eq. 29.

Let $\theta \in \Theta_Z(c_i, A)$ be an injective function. Then $s[\theta]$ is already minimized; in other words, $s[\theta]$ is isomorphic to s. To see this, suppose otherwise, that there exists a homomorphism from $s[\theta] \to s[\theta]$ that is not surjective; since θ is injective, we can extend it to a homomorphism $s \to s$ by defining it to be the identity on the variables in $Var_Z(c_i)$. Extend this to a homomorphism $c_i \to c_i$ by defining it to be the identity on all other subcomponents. Since it is not injective, it means that c_i is not minimized, contradiction. Thus, if θ is injective, then $s[\theta]$ is minimized; we call it an injective subcomponent.

We say that the subcomponent $s[\theta]$ *occurs syntactically* in d'_k, if $\exists i = 1, m$ and $\exists \theta \in \Theta_Z(c_i, A)$ s.t. $s = k(i, \theta)$; this does not prevent $s[\theta]$ from disappearing during minimization. Let S be a set of subcomponents $s[\theta]$. We say that S *occurs in the minimized CNF of* $d[A/Z]$ if after minimizing Eq. 32, one of the conjuncts d'_k contains every $s[\theta] \in S$. We will prove Prop. 7.5 by showing that there exists a set S of injective subcomponents that occurs in the minimized CNF, and $\bigvee S$ has no separator. The following lemma is an immediate consequence of Prop. 2.13 and the subsequent discussion on query minimization.

LEMMA 7.7. *A set S occurs in the minimized CNF of $d[A/Z]$ if the following conditions are satisfied:*

(1) *Every $s[\theta] \in S$ is minimized.*
(2) *There exists $k \in K$, such that for all $s[\theta] \in S$ (a) $s[\theta]$ occurs syntactically in d'_k, and (b) $s[\theta]$ is not redundant: for every other $s'[\theta']$ in d'_k, $s[\theta] \not\Rightarrow s'[\theta']$.*
(3) *The disjunctive query d'_k from the previous point is not redundant, meaning that there is no other disjunctive query d'_l s.t. $d'_l \Rightarrow d'_k$.*

In the rest of this section we will prove the following proposition, which implies Prop. 7.5.

PROPOSITION 7.8. *Let $d = c_1 \vee \ldots \vee c_m$ be a leveled, immediately unsafe query, with ≥ 3 levels. Then there exists a level Z with the following property. For any set of constants A s.t. $|A| \geq 2 \cdot \max_i |Var_Z(c_i)|$, then there exists a set S of subcomponents such that: (a) S occurs in the minimized $d[A/Z]$, and (b) $\bigvee S$ has no separator.*

The proposition implies Prop. 7.5. Indeed, if d has ≥ 3 levels, let d'_k be the disjunctive query in the minimized CNF expression for $d[A/Z]$ that contains the set S. Then $d[A/Z] \to d'_k$ is a maximal subquery rewriting rule (since d'_k is a coatom in the CNF lattice, and $\mu = -1$). Furthermore, let d'' be the symbol-component of d'_k that contains $\bigvee S$: obviously, we have $d'_k \stackrel{*}{\to} d''$, (simply remove the disconnected components by setting their relational symbols $R = \texttt{false}$). Since $\bigvee S$ has no separator, neither does d'', hence d'' is immediately unsafe, proving that $d[A/Z]$ is unsafe.

7.3 Base Case: No Root Variable

In this section we prove Prop. 7.8 for the case when there exists a component c_i that has no root variable; such query is called *non-hierarchical* in [Dalvi and Suciu 2004;

Dalvi and Suciu 2007b]. No disjunctive query that contains c_i can have a separator, since by definition the separator must include a root variable from each component. Thus, our proof consists of showing that a non-hierarchical subcomponent of $c_i[\theta]$ occurs in the minimized $d[A/Z]$.

For $x \in Var(c_i)$, denote $at(x)$ the set of atoms that contain x. Let x be such that $at(x)$ is a maximal set; $at(x)$ does not contain all atoms in c_i, because x is not a root variable. Since c_i is connected, there exists some other variable y s.t. all three sets $at(x) \cap at(y)$, $at(x) - at(y)$ and $at(y) - at(x)$ are nonempty. Choose Z to be any level distinct from the two levels containing x and y: such a level exists since d has ≥ 3 levels[12]. Since x, y occur together in some atom, there exists $s \in sc_Z(c_i)$ s.t. $x, y \in Var(s)$. Let $\theta \in \Theta_Z(c_i, A)$ be any injective substitution. Then $s[\theta]$ is non-hierarchical, because $at(x) \cap at(y)$, $at(x) - at(y)$ and $at(y) - at(x)$ still holds in $s[\theta]$, and there is no set $at(u)$ strictly larger than $at(x)$. Thus, $s[\theta]$ is immediately unsafe. We show that $s[\theta]$ occurs in the minimized CNF expression for $d[A/Z]$, which proves Prop. 7.8 for this case. We prove a more general statement.

LEMMA 7.9. *Suppose $d = \bigvee_i c_i$ is a minimized disjunctive query, and Z a level. Then, for any component c_i, for any injective function $\theta \in \Theta_Z(c_i, A)$, and any subcomponent $s \in sc_Z(c_i)$, the expression $s[\theta]$ occurs in the minimized CNF expression $d[A/Z]$.*

PROOF. We verify the three conditions of Lemma 7.7. (1) is immediate, since θ is injective.

To prove (2), define the following function $k \in K$. On the input i, θ, $k(i, \theta) = s$. On any other input j, θ', we define $k(j, \theta')$ to be a subcomponent $s' \in sc_Z(c_j)$ chosen s.t. $c_i[\theta] \not\Rightarrow s'[\theta']$: this further implies that $s[\theta] \not\Rightarrow s'[\theta']$ (because $c_i[\theta] \Rightarrow s'[\theta']$), proving (2). It remains to show that such an s' exists. Assume the contrary, i.e. for all $s' \in sc_Z(c_j)$, the logical implication $c_i[\theta] \Rightarrow s'[\theta']$ holds. Then $c_i[\theta] \Rightarrow c_j[\theta'](\equiv \bigvee_{s' \in sc_Z(c_j)} s'[\theta'])$. This means that there exists a homomorphism $c_j[\theta'] \rightarrow c_i[\theta]$, which implies that there exists a homomorphism $c_j \rightarrow c_i$ (because θ is injective). But the original query d was minimized, hence $i = j$, and furthermore c_i is also minimized, hence every endomorphism is an automorphism, implying that $\theta = \theta'$, which contradicts the assumption that (j, θ') is different from (i, θ). This completes the proof of (2).

To prove (3), let d'_k, $k \in K$ be defined as above. Let $L \subseteq K$ denote the indices of the non-redundant disjunctive queries d'_l, and assume each d'_l is minimized. Thus $d[A/Z] = \bigwedge_{l \in L} d'_l$. Assuming d'_k is redundant, there exists $l \in L$, s.t. $d'_l \Rightarrow d'_k$. We claim that d'_l also contains $s[\theta]$, which proves (3). To see this, notice that $c_i[\theta] \Rightarrow d[A/Z] \Rightarrow d'_l$ (because $c_i \Rightarrow d$), therefore:

$$c_i[\theta] \Rightarrow d'_l \Rightarrow d'_k$$

Now we apply Sagiv and Yannakakis' disjunctive query containment criterion (see Prop. 2.13) twice. First, since $c_i[\theta]$ is a component, there exists a component $s''[\theta'']$ in d'_l s.t. $c_i[\theta] \Rightarrow s''[\theta'']$; second, we obtain component $s'[\theta']$ in d'_k s.t. $s''[\theta''] \Rightarrow s'[\theta']$.

[12]It is entirely possible that Z has no attributes in c_i. For example c_i may be $R(x), S(x, y), T(y)$, while Z contains attributes from other relations. We still make progress simplifying d to $d[A/Z]$.

In summary:

$$c_i[\theta] \Rightarrow s''[\theta''] \Rightarrow s'[\theta']$$

Because of the way we constructed d'_k, the only component $s'[\theta']$ that can be logically implied by $c_i[\theta]$ is $s[\theta]$, hence the implications become:

$$c_i[\theta] \Rightarrow s''[\theta''] \Rightarrow s[\theta]$$

Since $c_i[\theta] = \bigwedge_{s_1 \in sc_Z(c_i)} s_1[\theta]$, by the criterion for conjunctive query containment in Prop. 2.13 there exists s_1 such that:

$$s_1[\theta] \Rightarrow s''[\theta''] \Rightarrow s[\theta]$$

But the only subcomponent s_1 s.t. $s_1[\theta] \Rightarrow s[\theta]$ is $s_1 = s$: otherwise, if such an s_1 exists, then the query c_i could be further minimized (by removing the subcomponent s), contradicting the fact that c_i was minimized. Thus, $s''[\theta'']$ is equivalent to $s[\theta]$, which proves the claim that d'_l contains $s[\theta]$. □

7.4 Case 1: Non-splitting Level

From now on we assume that every component c_i has a root variable. If there is only one root variable in c_i, and the level Z contains that variable, then $c_i[\theta]$ is disconnected, for all $\theta \in \Theta_Z(c_i, A)$. In this case we say that Z is *splitting* c_i. In all other cases, $c_i[\theta]$ is connected, and we say that Z is *non-splitting* for c_i.

Case 1 is when there exists a level, Z that is non-splitting for all c_i. We prove that Prop. 7.8 holds for this level. In this case, every $c_i[\theta]$ is connected, and $d[A/Z] = \bigvee c_i[\theta]$ is a disjunctive query. The minimized CNF for $d[A/Z]$ has a single conjunct, which is the disjunction of all non-redundant $c_i[\theta]$. By Lemma 7.9, if $\theta \in \Theta_Z(c_i, A)$ is injective, then $c_i[\theta]$ is non-redundant in $d[A/Z]$. Intuitively, $d[A/Z]$ contains an entire copy of $d = \bigvee_i c_i$, and since d has no separator, neither can $d[A/Z]$ have one. We prove this formally.

LEMMA 7.10 CASE 1. *Let $d = c_1 \vee \ldots \vee c_m$ be an L-leveled, immediately unsafe query, Z a non-splitting level, and A be a set of constants s.t. $|A| \geq \max_i |Var_Z(c_i)|$. Let S be a maximal set $S \subseteq \{c_i[\theta] \mid \theta \in \Theta_Z(c_i, A), \theta$ injective$\}$ s.t. S is symbol-connected. Then S occurs in the minimized $d[A/Z]$ and is has no separator.*

PROOF. By Lemma 7.9, every $c_i[\theta]$ occurs in $d[A/Z]$. We show that $d' = \bigvee S$ has no separator. Note that d' is a disjunctive query over the shattered vocabulary for $d[A/Z]$. We denote $\mathbf{R}$ and $\mathbf{R}'$ the vocabularies for d and d' respectively. If the relation symbol $R \in \mathbf{R}$ contains an attribute in Z, then it has $|A|$ shatterings, denoted R_a, one for every $a \in A$; we denote the attributes in R and R_a with the same letter, thus, if (R, i) is an attribute then so is (R_a, i), *unless* (R, i) belongs to the level Z (in which case i is no longer an attribute in R_a). If R does not contain an attribute in Z, then the only shattering is itself.

Suppose d' has a separator: then there exists a level V' of d' that contains only root variables. Define the following set of attributes V from the vocabulary $\mathbf{R}$. V contains all attributes (R, i) s.t. either R is not shattered and $(R, i) \in V$, or R is shattered and there exists $a \in A$ s.t. $(R_a, i) \in V$. Notice that $V \cap Z = \emptyset$. We prove that (a) V is a level and (b) it contains only root variables. This means that d has a separator, contradicting the assumption that it is immediately unsafe.

To prove (a), we need to show that V is closed under edges of the attribute graph. Let $(R,i) \in V$ and suppose there exists an edge (R,i), (S,j) in the attribute graph. Assume that both R and S are shattered: the other cases are similar (and simpler) and omitted. By the definition of V, there exists $(R_a,i) \in V'$. We claim that (S,j) is also in V. Let c_k be some component in d where two atoms $R(\ldots), S(\ldots)$ join attributes (R,i) and (S,j); let x be the join variable. Let z be the variable on level Z in the atom $R(\ldots)$: $z \neq x$ since the attribute $(R,i) \notin Z$. Choose any injective function $\theta : Var_Z(c_k) \to A$ that maps z to a. Then $c_k[\theta]$ contains two atoms $R_a(\ldots), S_b(\ldots)$ with a common variable on attributes (R_a,i) and (S_b,j); here b is either a, if the atom $S(\ldots)$ has the same variable z on level Z, or a different constant $b \neq a$, if the atom $S(\ldots)$ has a different variable z' on level Z. It follows that $(S_b,j) \in V'$, which implies $(S,j) \in V$.

To prove (b), let $R(\ldots)$ be an atom in some c_k that contains an attribute $(R,i) \in V$. Let x be the variable in this position. We assume R is shattered (the other case is similar), and let z be the variable on level Z: it must be different from x since (R,i) cannot be in Z. Choose any injective function $\theta \in \Theta_Z(c_k)$ that maps z to a. Then $c_k[\theta]$ belongs to S (because it is symbol-connected via R_a), hence x is a root variable in $c_k[\theta]$, and it must also be a root variable in c_k (because $c_k[\theta]$ and c_k are isomorphic, since we choose θ to be injective). □

EXAMPLE 7.11. *We illustrate Case 1. Consider:*

$$d = c_1 \vee c_2 = R(x), S(x,y,z) \vee S(x',y',z'), T(y')$$

Level $Z_1 = \{x, x'\}$ splits c_1 into two subcomponents $R(x)$ and $S(x,y,z)$, while level $Z_2 = \{y, y'\}$ splits c_2 into two subcomponents $S(x',y',z')$ and $T(y')$. Therefore $Z_3 = \{z, z'\}$ is the only non-splitting level. We choose level Z_3 and rewrite[13]

$$d[a/Z_3] = R(x), S(x,y,a) \vee S(x',y',a), T(y')$$

and this query is still without separator.

Note that it would have been a mistake to choose Z_1:

$$\begin{aligned} d[a/Z_1] &= R(a), S(a,y,z) \vee S(a,y',z'), T(y') \\ &= (R(a) \vee S(a,y',z'), T(y')) \wedge (S(a,y,z) \vee S(a,y',z'), T(y')) \\ &= (R(a) \vee S(a,y',z'), T(y')) \wedge S(a,y,z) \end{aligned}$$

and $d[a/Z_1]$ is a safe query (because all three elements of its V-shaped CNF lattice have separators). Thus, Case 1 is necessary.

From now on we will assume that Case 1 does not apply, i.e. every level Z splits some component.

7.5 Case 2: Two Related Levels

We assume that every component c_i has a root variable, and every level Z splits some component. Fix three levels, call them Z_1, Z_2, Z_3. For each $i = 1,2,3$, let C_i be the set of components c_k that are split by the level Z_i; by assumption, $C_i \neq \emptyset$,

[13] In this example it suffices to choose $A = \{a\}$; choosing a larger set $A = \{a,b\}$ leads to a longer but similar query.

for $i = 1, 2, 3$. Notice that a level Z_i does *not* split any component $c_k \in C_j$ for $j \neq i$. This is because c_k is split by level Z_j, hence it has a single root variable, which is on level Z_j; therefore, it is not split by level Z_i. In particular, the three sets C_1, C_2, C_3 are disjoint.

To prove Prop. 7.8 in this case, we use the following idea. Let $c_1 \in C_1$ and $c_2 \in C_2$, and assume they have a common symbol. Then we choose the third level, $Z = Z_3$, and consider the query $d[A/Z_3]$. Both $c_1[\theta_1]$ and $c_2[\theta_2]$ are connected, because Z_3 does not split c_1 and c_2; also assume $c_1[\theta_1]$ and $c_2[\theta_2]$ have a common relational symbol (for that we will choose θ_1, θ_2 to be compatible). We claim that $c_1[\theta_1] \vee c_2[\theta_2]$ has no separator. This is because the only root variable in $c_1[\theta_1]$ is on level Z_1 and the only root variable in $c_2[\theta_2]$ is on level Z_2, hence no level contains both root variables. Thus, we prove Prop. 7.8 by showing that the set $S = \{c_1[\theta_1], c_2[\theta_2]\}$ occurs in the minimized $d[A/Z]$.

More precisely, in case Case 2 we make the following assumption. We assume that there exists $c_1 \in C_1$ and $c_2 \in C_2$ and there exists $s_1 \in sc_{Z_1}(c_1)$ such that $c_2 \Rightarrow s_1$. We say that c_1 and c_2 are *related*. In particular, c_1 and c_2 have a common symbol, because there exists a homomorphism $s_1 \to c_2$.

LEMMA 7.12 CASE 2. *Let $c_1 \in C_1$ and $c_2 \in C_2$. Assume that there exists $s_1 \in sc_{Z_1}(c_1)$ such that $c_2 \Rightarrow s_1$, and that $|A| \geq |Var_{Z_3}(c_1)| + |Var_{Z_3}(c_2)|$. Then, there exists two injective functions $\theta_i \in \Theta_{Z_3}(c_i, A)$, for $i = 1, 2$ the set $S = \{c_1[\theta_1], c_2[\theta_2]\}$ occurs in the minimized CNF expression for $d[A/Z_3]$, and the query $c_1[\theta_1] \vee c_2[\theta_2]$ is symbol-connected and has no separator.*

PROOF. We have already seen that $c_1[\theta_1] \vee c_2[\theta_2]$ has no separator, so we will prove the other conditions, by extending the proof idea from Lemma 7.9. To simplify the notation, we assume w.l.o.g. that the two components c_1, c_2 referred by the lemma are the first two in the disjunctive query $d = \bigvee_{i=1,m} c_i$.

We need to establish the three conditions of Lemma 7.7. To satisfy (1), we need to choose θ_1, θ_2 injective. Let $\theta_2 \in \Theta_{Z_3}(c_2, A)$ be any injective function, and denote $A_2 = Im(\theta_2)$. Let h denote the homomorphism $h : s_1 \to c_2$. We define $\theta_1 \in \Theta_{Z_3}(c_1, A)$ as follows. For every $x \in Var_{Z_3}(c_1)$, if $x \in Var_{Z_3}(s_1)$ then $\theta_1(x) = \theta_2(h(x))$, otherwise define $h(x)$ to be a fresh constant in $A - A_2$. Let $A_1 = Im(\theta_1) - A_2$. Because of the way we defined θ_1, the homomorphism h extends to a homomorphism $h' : s_1[\theta_1] \to c_2[\theta_2]$, which also implies that $c_1[\theta_1]$ and $c_2[\theta_2]$ share some common relational symbol.

We now prove (2). Define the following function k such that $k(c_1, \theta_1) = c_1$, $k(c_2, \theta_2) = c_2$, and for (c_j, θ') different from both (c_1, θ_1) and (c_2, θ_2); $k(c_j, \theta') = s' \in sc_{Z_3}(c_j, A)$ where s' satisfies two properties:

$$c_1[\theta_1] \not\Rightarrow s'[\theta'] \qquad \text{and} \qquad c_2[\theta_2] \not\Rightarrow s'[\theta'] \tag{34}$$

We need to prove that such an s' exists. From the proof of Lemma 7.9 we know that there exists $s'_1 \in sc_{Z_3}(c_j)$ such that $c_1[\theta_1] \not\Rightarrow s'_1[\theta']$, and there exists $s'_2 \in sc_{Z_3}(c_j)$ such that $c_2[\theta_2] \not\Rightarrow s'_1[\theta']$. If Z_3 does not split c_j, then both components s'_1 and s'_2 are equal to c_j, and we define $k(c_j, \theta') = c_j$. So assume that Z_3 splits c_j: then c_j has a root variable z, on level Z. Denote $a = \theta'(z)$; the constant a occurs in every atom of $c_j[\theta']$. We consider three cases. Case 1: $a \notin A_1 \cup A_2$. Here we define $k(c_j, \theta') = s'_1$ (arbitrarily): Eq. 34 holds because neither $c_1[\theta_1]$ nor $c_2[\theta_2]$

contain the constant a. Case 2: $a \in A_2$. Then we define $k(c_j, \theta) = s_2'$. The the second condition in Eq. 34 holds, we need to check the first condition. Suppose otherwise; then there exists a homomorphism $g : s'[\theta'] \to c_1[\theta_1]$. On one hand all atoms in $s'[\theta']$ have the constant $a \in A_2$, on the other hand the only atoms in $c_1[\theta_1]$ that could contain a are those in $s_1[\theta_1]$ (by the construction of θ_1), hence g is a homomorphism $g : s'[\theta'] \to s_1[\theta_1]$. Compose it with $h' : s_1[\theta_1] \to c_2[\theta_2]$, to obtain a homomorphism $s'[\theta'] \to c_2[\theta_2]$, contradicting the second condition in Eq. 34. Case 3: $a \in A_1$. Then we define $k(c_j, \theta') = s_1'$. The first condition in Eq. 34 is now automatic, while the second condition follows from the fact that no atom in $c_2[\theta_2]$ has a constant from A_1.

Finally, the proof of (3) is similar to the proof in Lemma 7.9, and is omitted. □

EXAMPLE 7.13. *We illustrate Case 2 with:*

$$d = c_1 \vee c_2 \vee c_3 = R(x_1, y_1), S(x_1, z_1) \vee R(x_2, y_2), T(y_2, z_2) \vee S(x_3, z_3), T(y_3, z_3)$$

Level $Z_1 = \{x_1, x_2, x_3\}$ splits c_1 into $R(x_1, y_1)$ and $S(x_1, z_1)$; similarly level $Z_2 = \{y_1, y_2, y_3\}$ splits c_2, and level $Z_3 = \{z_1, z_2, z_3\}$ splits c_3. Thus, every level is splitting, and Case 1 does not apply. Case 2 applies here. For example there is a homomorphism from the subcomponent $R(x_1, y_1)$ to c_2, showing that c_1 and c_2 are related. We therefore choose level Z_3, and choose two constants $A = \{a, b\}$ and rewrite to:

$$\begin{aligned} d[A/Z_3] =& c_1[A/Z_3] \vee c_2[A/Z_3] \vee S(x_3, a), T(y_3, a) \vee S(x_3, b), T(y_3, b) \\ =& (c_1[A/Z_3] \vee c_2[A/Z_3] \vee S(x_3, a) \vee S(x_3, b)) \wedge \\ & (c_1[A/Z_3] \vee c_2[A/Z_3] \vee S(x_3, a) \vee T(y_3, b)) \wedge \\ & (c_1[A/Z_3] \vee c_2[A/Z_3] \vee T(y_3, a) \vee S(x_3, b)) \wedge \\ & (c_1[A/Z_3] \vee c_2[A/Z_3] \vee T(y_3, a) \vee T(y_3, b)) \end{aligned}$$

The CNF expression is given in the last four rows, and is the conjunction of four disjunctive queries. Consider the second:

$$d_2' = c_1[A/Z_3] \vee c_2[A/Z_3] \vee S(x_3, a) \vee T(y_3, b)$$

Expanding the first two expressions results in 4 components. Two of these are redundant, because one contains $S(x_1, a)$ and the other contains $T(y_2, b)$. However, the following two components are not redundant:

$$d_2' = \ldots \vee R(x_1, y_1), S(x_1, b) \vee R(x_2, y_2), T(y_2, a) \vee \ldots$$

Obviously, d_2' has no separator.

In the next example we show why it is important to pick the level according to our rule (as the third level, if two levels are related). Consider:

$$d = c_1 \vee c_2 \vee c_3 =$$
$$U(x, y', z'), V(x, y'', z'') \vee R(y), S(x'', y, z''), U(x'', y, z'') \vee V(x', y', z), S(x', y', z), T(z)$$

Here $Z_1 = \{x, x', x''\}$ splits c_1 into two subcomponents: $U(x, y', z')$ and $V(x, y'', z'')$, and there is a homomorphism from the first to c_2: here we must pick level $Z_3 = \{z, z', z''\}$. There is also a homomorphism from the second component to c_3; here we should pick level $Z_2 = \{y, y', y''\}$. Either choice is fine, but it would be a mistake

to pick $Z_1 = \{x, x', x''\}$, since Z_2, Z_3 are unrelated. To see that, consider expanding it with a set of constants $A = \{a, b, c, \ldots\}$:

$$d[A/Z_1] = \bigvee_{v \in A} U(v, y', z')V(v, y'', z'') \vee c_2[A/Z_1] \vee c_3[A/Z_1] = \bigwedge_k d'_k$$

Each d'_k contains $c_2[A/Z_1] \vee c_3[A/Z_1]$ and, for each constant $v \in A$, it contains either $U(v, y', z')$ or $V(v, y'', z'')$ (for a total of $2^{|A|}$ disjunctive queries d_k). However, each d'_k has a separator. Indeed:

$$c_2[A/Z_1] \vee c_3[A/Z_1] = \bigvee_{a \in A} (R(y), S(a, y, z''), U(a, y, z'') \vee V(a, y', z), S(a, y', z), T(z))$$

For any fixed constant a the disjunction of the two components above has no separator: in particular the entire expression above has no separator. However, for every constant a we must include in d'_k either $U(a, y', z')$ or $V(a, y'', z'')$, and either the first or the second component above becomes redundant. If A had a single constant a, then it is obvious that d'_k has a separator; one can check that d'_k continues to have a separator for an arbitrary A (just choose separately, a separator for each constant $a \in A$). This shows that we cannot expand on Z_1. On the other hand, expanding on level Z_3 with a single constant a we obtain:

$$\begin{aligned} d[a/Z_3] &= U(x, y', a), V(x, y'', a) \vee R(y), S(x'', y, a), U(x'', y, a) \vee V(x', y', a), S(x', y', a), T(a) \\ &= (U(x, y', a), V(x, y'', a) \vee R(y), S(x'', y, a), U(x'', y, a) \vee V(x', y', a), S(x', y', a)) \wedge \\ &\quad (U(x, y', a), V(x, y'', a) \vee R(y), S(x'', y, a), U(x'', y, a) \vee T(a)) \end{aligned}$$

Neither conjunct has a separator.

7.6 Case 3: Three Unrelated Levels

We continue to assume that every component has a root variable, and every level is splitting. We use the same notations Z_1, Z_2, Z_3, for three arbitrary, but fixed levels, and C_1, C_2, C_3 for the non-empty sets of components split by each of these three levels. The last case we need to consider is when there are no related pairs of components. The query d is symbol-connected (by definition of an immediately unsafe query), therefore, for each $i \neq j$ there exists a path in the co-occurrence graph from a query in C_i to a query in C_j. Let n_{ij} be the length of the shortest such path. Assume n_{12} is the minimum of n_{12}, n_{13}, n_{23} (break ties arbitrarily). Then we pick level Z_3.

Let a shortest path from some $c_1 \in C_1$ to some $c_2 \in C_2$ be:

$$c^0(\in C_1), c^1, c^2, \ldots, c^n(\in C_2)$$

Notice that, except for the first and last component, no other component on this path belongs to $C_1 \cup C_2 \cup C_3$, otherwise we would have a shorter path. By definition, any two consecutive components share a common relational symbol; therefore, assuming $|A| \geq \max_i |Var_{Z_3}(c_i)|$, we can choose substitutions $\theta_i \in \Theta_{Z_3}(c^i, A)$ such that any two consecutive components in the path:

$$c^0[\theta_0], c^1[\theta_1], c^2[\theta_2], \ldots, c^n[\theta_n]$$

share a common relational symbol. Each $c^i[\theta_i]$ is connected, because none of the c^i's belongs to C_3. Define $S = \{c^i[\theta_i] \mid i = 0, n\}$. This set is symbol-connected, and

$\bigvee S$ has no separator, because the only root variable in c^0 is on level Z_1, and the only root variable in c^n is on level Z_2. Thus, Prop. 7.8 follows in Case 3 from the following lemma:

LEMMA 7.14 CASE 3. *Let A be a set of constants such that $|A| \geq \max_i(|Var_{Z_3}(c_i)|)$. Then the set S defined above occurs in the minimized CNF expression of $d[A/Z_3]$, and $\bigvee S$ has no separator.*

PROOF. We need to prove three items of Lemma 7.7; item (1) is satisfied since all θ_i' are injective.

We prove item (2). Define the following function k. For each $i = 0, n$, define $k(c^i, \theta_i) = c^i$; for every c_j, θ' distinct from all (c^i, θ_i) define $k(c_j, \theta') = s' \in sc_{Z_3}(c_j)$ such that the following condition holds:

$$c^i[\theta_i] \not\Rightarrow s'[\theta'] \qquad \text{for all } i = 0, n \tag{35}$$

We need to prove that such an s' exists. If level Z_3 does not split c_j, in other words if $c_j \notin C_3$, then we take $s' = c_j$: there cannot be a homomorphism $c_j[\theta'] \rightarrow c^i[\theta_i]$ because that would imply a homomorphism $c_j \rightarrow c^i$ (since θ_i is injective), contradicting the fact that $d = \bigvee c_j$ is minimized. So assume that the level Z_3 splits c_j. Notice that, for every $s' \in sc_{Z_3}(c_j)$, we have $c^0[\theta_0] \not\Rightarrow s'$ because $c^0 \in C_1$ and $c_j \in C_3$ are not related (we have treated related components in Case 2). Similarly, $c^n[\theta_n] \not\Rightarrow s'$. So we only need to check Eq. 35 for $i = 1, n-1$. If $n = 0$ then it holds vacuously. If $n > 1$ then it is satisfied by any choice of s', by the following argument: if there exists a homomorphism $s[\theta'] \rightarrow c^i[\theta_i]$, then c_j and c^i have a common symbol, and there is a shorter path from c_j to either c^0 or c^n (depending on whether i is in the first half or the second half) contradicting the assumption that n is the length of the shortest path. Finally, if $n = 1$, then we use the argument in the proof of Lemma 7.9 to argue that there exists $s' \in sc_{Z_3}(c_j)$ s.t. $c^1[\theta_1] \not\Rightarrow s'[\theta']$. This completes the proof of claim (2).

We now prove item (3), using the same idea as in Lemma 7.9. Let d'_k be the disjunctive query that contains syntactically all $c^i[\theta_i]$, and let $L \subseteq K$ be the set of indices of the non-redundant disjunctive queries d'_l; assume each d'_l is minimized. Assume that there exists $l \in L$ such that $d'_l \Rightarrow d'_k$. As before, we have:

$$\bigvee_{i=0,n} c^i[\theta_i] \Rightarrow d'_l \Rightarrow d'_k$$

For each $i = 0, n$, by Sagiv and Yannakakis' criterion there exists $s''_i[\theta'']$ in d'_l, and by the same criterion again there exists $s'_i[\theta'_i]$ in d'_k such that:

$$c^i[\theta_i] \Rightarrow s''_i[\theta''] \Rightarrow s'_i[\theta'_i]$$

By our construction, then only $s'_i[\theta'_i]$ that can be implied by $c^i[\theta_i]$ is itself, hence all three expression above are logically equivalent, proving that d'_l contains $c^i[\theta_i]$. Since this holds for every $i = 0, n$, it follows that d'_l contains the entire set S. (We have skipped a step from Lemma 7.9, because now $c^i[\theta_i]$ is connected.) □

EXAMPLE 7.15. *We illustrate with an example:*

$$\begin{aligned} d = & A(x_1), R(x_1, y_1, z_1) \vee B(y_2), S(x_2, y_2, z_2) \vee C(z_3), T(x_3, y_3, z_3) \\ & \vee R(x, y, z), S(x, y, z), T(x, y, z) \end{aligned}$$

Denote the four components above c_1, c_2, c_3, c_4, There are three levels, Z_1, Z_2, Z_3, (Z_1 contains x_1, x_2, x_3, x etc) and each level Z_i splits c_i, hence $C_i = \{c_i\}$, $i = 1, 2, 3$. The path connecting c_1 to c_2 is c_1, c_4, c_2, and is of minimal length. Choose the level Z_3, and a single constant a. Then that path continues to have no separator in $d[a/Z_3]$. Indeed, $d[a/Z_3]$ is:

$$A(x_1), R(x_1, y_1, a) \vee B(y_2), S(x_2, y_2, a) \vee$$
$$C(a), T(x_3, y_3, a) \vee R(x, y, a), S(x, y, a), T(x, y, a)$$

And its CNF expression is:

$$d[a/Z_3] =$$
$$(A(x_1), R(x_1, y_1, a) \vee B(y_2), S(x_2, y_2, a) \vee C(a) \vee R(x, y, a), S(x, y, a), T(x, y, a)) \wedge$$
$$(A(x_1), R(x_1, y_1, a) \vee B(y_2), S(x_2, y_2, a) \vee T(x_3, y_3, a) \vee R(x, y, a), S(x, y, a), T(x, y, a))$$

Both disjunctive queries are non-redundant. The first is already minimized, and the path AR, RST, BS has no separator. The second minimizes and becomes safe.

8. FORBIDDEN QUERIES ARE HARD

In this section we prove that forbidden queries (Def. 7.2) are hard:

THEOREM 8.1. *If d is a forbidden query, then computing $P(d)$ is #P-hard.*

This completes the proof of the DichotomyTheorem 4.21.

By Theorem 7.3 we know that d has exactly two levels, which we denote X and Y respectively. Variables that occur in these two levels are denoted $x_0, x_1, x_2, \ldots$ and $y_0, y_1, y_2, \ldots$ respectively. Every symbol must have arity ≤ 2, and therefore there are three kinds of symbols: binary symbols, $S(x, y)$; unary symbols $R(x)$ on level X, called *left unary symbols*, and unary symbols $T(y)$ on level Y, called *right unary symbols*.

For illustration, all queries h_k are forbidden queries. We show more complex forbidden queries in Sect. 8.1 and elsewhere in this section.

Throughout this section we fix an instance of the #PP2CNF problem:

$$\Phi = \bigwedge_{(i,j)\in E} (X_i \vee Y_j) \qquad \text{where } E \subseteq [n_1] \times [n_2] \tag{36}$$

Denote $|E| = m$, and denote $n = n_1 + n_2$.

8.1 Classification of Forbidden Queries

Let $d = \bigvee_i c_i$ be a forbidden query. By definition, it also means that each c_i is minimized, and none is redundant. Let X and Y denote its two levels. We say X is a root in c_i if c_i has a root variable that is on level X.

DEFINITION 8.2. *A component c_i is a* left component *if Y is not a root; it is a* right component *if X is not a root.*

If c_i has no root variable then it is both a left and a right component; in that case it is called *non-hierarchical* (Sect. 7.3).

DEFINITION 8.3. *The query d is called* non-hierarchical*, if it has a component c_i that is not hierarchical.*

DEFINITION 8.4. *Let d be hierarchical, and c_i be a left component. If c_i has exactly two variables x and y then we say that it is of Type 1. Otherwise we say that it is of Type 2. Similarly for right components.*

For example, in h_k the left component $R(x_0), S_1(x_0, y_0)$ is of type 1 and the right component $S_k(x_k, y_k), T(y_k)$ is also of type 1. In $d_\diamond = S_1(x, y_1), S_2(x, y_2) \vee S_1(x_1, y), S_2(x_2, y)$ both the left component and the right components are of type 2. We now show that, in a hierarchical forbidden query, all left components must have the same type, either 1 or 2; thus, there are four types, 1-1, 1-2, 2-1, 2-2, where type 1-1 means that all left and all right components are of type 1, type 1-2 means that all left components are of type 1 and all right components are of type 2, etc.

Recall the co-occurrence graph of d (Def. 3.4): nodes are the components c_i, and edges are pairs of components (c_i, c_j) that share a common relational symbol. A *left-right path* is a path $c_0, c_1, \ldots, c_k$ s.t. c_0 is a left component and c_k is a right component. A *strict* path is a left-right path of minimal length.

PROPOSITION 8.5. *Every strict path $c_0, c_1, \ldots, c_k$ contains all relational symbols in d.*

PROOF. Let S be a symbol that does not occur on the strict path, and consider the rewrite rule $d \to d[S = \texttt{false}]$; denote the latter d'. In other words, d' is obtained from d by simply removing all components that contain S: all remaining components are non-redundant. In particular, all components $c_0, c_1, \ldots, c_k$ are kept in d'. Then d' is unsafe, because this set of components does not have a separator: in c_0 the only root variable is on level X, and in c_k the only root variable is on level Y, hence no level contains only root variables. It follows that d' is unsafe, contradicting the fact that d is a forbidden query. □

Consider a left component of Type 1, $R_1(x), R_2(x), \ldots, S_1(x, y), S_2(x, y), \ldots$ It must have at least one unary symbol, else y would be a root variable. We show that there is only one unary symbol. In fact, we prove something more general:

PROPOSITION 8.6. *Every hierarchical, forbidden query d has at most one left unary symbol $R(x)$. In particular, if it has any left component of Type 1, then it has exactly one left unary symbol. Similarly on the right.*

PROOF. Suppose d has two left unary symbols R and R_1, and let $c_0, c_1, \ldots, c_k$ be strict left-right path. Here c_0 is a left component, c_k is a right component, while the others are neither left nor right (otherwise we could find a shorter left-right path), hence they do not contain either R or R_1. Define $d' = d[R_1 = \texttt{true}]$. After minimization, d' contains those components $c_i[R_1 = \texttt{true}]$ that are non-redundant. We prove that the entire path is non-redundant. This proves the proposition, since it shows that the symbol-component of d' containing the path has no separator, hence d' is unsafe, contradicting the fact that d is forbidden. To prove that the path is non-redundant, denote $c'_i = c_i[R_1 = \texttt{true}]$; we prove that c'_i is not redundant in d'. If $i > 0$ then $c'_i = c_i$, and if there exists a homomorphism $c[R_1 = \texttt{true}] \to c_i$, then we have another strict path $c, c_i, c_{i+1}, \ldots, c_k$ (because c contains R_1 hence is a left component, and c, c_i have common symbols, because of the homomorphism). This path contains all symbols, including R, hence c must contain R, making the

homomorphism $c[R_1 = \texttt{true}] \to c_i$ impossible. If $i = 0$, then any homomorphism $c[R_1 = \texttt{true}] \to c_0[R_1 = \texttt{true}]$ extends to a homomorphism $c \to c_0$, by mapping every atom $R_1(x_1)$ in c to the unique atom $R_1(x)$ in c_0. (Here we use the fact that c_0 is hierarchical, which guarantees that the atom $R_1(x)$ is unique.) □

Consider now a left component of Type 2. It has one of these two forms: $S_1(x, y_{i_1}), S_2(x, y_{i_2}), \ldots$ or $R(x), S_1(x, y_{i_1}), S_2(x, y_{i_2}), \ldots$ We prove that the latter is impossible.

PROPOSITION 8.7. *Let d be a hierarchical, forbidden query. If it has any left component of Type 2, then it has no left unary symbols.*

PROOF. Suppose the contrary: d contains a the left unary symbol R. Then we prove that any left component c is of Type 1. Assume the contrary, that c has two or more variables $y_1, y_2, \ldots$ We consider two cases.

—There exists a strict path starting at c: $c_0 = c, c_1, \ldots, c_k$. We consider two sub-cases. First, when c_1 contains all binary symbols occurring in all left components. We choose such a binary symbol S, as follows. If c_0 does not contain the unary symbol R, then we choose S such that it does not occur in c_0: such an S exists, otherwise, by our assumption, there exists a homomorphism $c_0 \to c_1$; if c_0 contains the unary symbol R, then we choose S arbitrarily. In either case, $c_0[S = \texttt{true}]$ is a left component. We claim that $d[S = \texttt{true}]$ rewrites to an unsafe query, contradicting the fact that d is forbidden. Indeed, all components $c_0[S = \texttt{true}], c_1[S = \texttt{true}], c_2, \ldots, c_k$ are non-redundant in $d[S = \texttt{true}]$ (the proof is identical to that of Prop. 8.6 and omitted), and this is indeed a path, because there exists at least one binary symbol other than S in c_0 (since c_0 has two or more y-variables), and that symbol is common to $c_0[S = \texttt{true}]$ and $c_1[S = \texttt{true}]$. The second sub-case is when c_1 does not contain all binary symbols from all left components, then we claim that $d[R = \texttt{true}]$ rewrites to a forbidden query. First we note that $c_0[R = \texttt{true}]$ is a left component, because it has two or more y-variables. (Note that c_0 does not need to contain R.) Next we show that the path $c_0[R = \texttt{true}], c_1, \ldots, c_k$ is non-redundant, by adapting the same argument from the proof of Prop. 8.6. The proof for the non-redundancy of $c_0[R = \texttt{true}]$ is the same, so consider some $i \geq 1$, and assume there is a homomorphism $c[R = \texttt{true}] \to c_i$. This implies that c contains R, hence it is a left component, hence $c, c_i, \ldots, c_k$ is a strict path, hence $i = 1$. Let S be the left binary symbol missing from c_1. Since the path starting at c is strict, S must occur in either c or c_1, implying that it occurs in c. But this contradicts the existence of the homomorphism $c[R = \texttt{true}] \to c_1$. Thus, if there exists a strict path starting at c, then c must be of Type 1.

—There is no strict path starting at c. We prove that this is impossible. Consider any strict path, and let c_0 be its first component. The path contains all relational symbols, hence c_0 contains R, and we have proved in the first case that c_0 is of type 1. All symbols in c must also occur on the path, and they can only occur in c_0, otherwise we could construct a strict path starting at c. But this means that there exist a homomorphism $c \to c_0$ (here we use the fact that c_0 has a single y-variable), contradicting the fact that d was minimized.

□

In the remainder of this section we will give three hardness proofs, one for each of the following three types of forbidden queries.

Non-hierarchical. We will prove the only non-hierarchical, forbidden query is $h_0 = R(x), S(x,y), T(y)$, introduced in Example 4.6.

Type 1-1. All left and right components are of type 1. These include all queries h_k, and others, e.g.:

$$d = R(x_0), S_1(x_0,y_0), S_2(x_0,y_0) \vee S_1(x_1,y_1), S_3(x_1,y_1) \\ \vee S_2(x_2,y_2), S_4(x_2,y_2) \vee S_3(x_3,y_3), S_4(x_3,y_3), T(y_3)$$

Type 2-2. All left and right components are of type 2, for example:

$$\begin{aligned} d_{\texttt{diamond}} =& S_1(x,y_1), S_2(x,y_1), S_3(x,y_1), S_1(x,y_2), S_2(x,y_2), S_4(x,y_2) \\ & \vee S_3(x_1,y), S_4(x_1,y), S_1(x_1,y), S_3(x_2,y), S_4(x_2,y), S_2(x_2,y) \\ d_f =& S(x,y_1), S_1(x,y_1), S(x,y_2), S_2(x,y_2) \vee S_1(x,y), S_2(x,y), S_3(x,y), S_4(x,y) \\ & S'(x_1,y), S_3(x_1,y), S'(x_2,y), S_4(x_2,y) \end{aligned} \tag{37}$$

In general, a left component of type 2 may have arbitrarily many variables y_i, and similarly a right component may have arbitrarily many x_i's, but we will not show examples with more than 2 variables (except for one in Example 8.40) because they are very verbose: the simplest type 2-2 query is $d_{\texttt{diamond}}$. We will illustrate during most of our discussion with $d_\diamond = S_1(x,y_1), S_2(x,y_2) \vee S_1(x_1,y), S_2(x_2,y)$, which is actually *not* a forbidden query (see Example 8.37), but we will use it as a surrogate for $d_{\texttt{diamond}}$. It is actually quite difficult to check if a query of type 2-2 is forbidden: for example, in order to prove that $d_{\texttt{diamond}}$ is forbidden one has to prove that for all maximal rewritings $d_{\texttt{diamond}} \stackrel{*}{\to} d$, where $d_{\texttt{diamond}} > d$ (Def. 7.1), the query d is safe. A decision procedure for checking whether a query is forbidden is beyond the scope of this paper.

We omit the hardness proofs for queries of types 1-2 and 2-1. These are queries where the left components are of type 1, and the right components of type 2, or vice versa. Their hardness proofs are straightforward adaptations of the proof for 1-1 and 2-2 respectively.

8.2 Hardness Proof for Non-hierarchical Forbidden Query

PROPOSITION 8.8. *The only non-hierarchical forbidden query is $h_0 = R(x), S(x,y), T(y)$. Therefore, it is #P-hard, by Prop. 5.2.*

PROOF. Let $d = \bigvee_i c_i$ be a leveled, immediately unsafe query, with 2 levels, and let c be a non-hierarchical component of d. We will prove that (A) $d \stackrel{*}{\to} h_0$, and (B) either $d > h_0$ (Def. 7.1) or d is h_0 up to renaming or relational symbols. This proves that h_0 is the only non-hierarchical forbidden query.

We start with (A), and prove it in two steps. First we shatter $d \to d_A$ s.t. c_A has only two variables, then we do some deterministic rewritings s.t. $d_A \stackrel{*}{\to} h_0$.

Let $S(x,y)$ be any binary atom in c. Since neither x nor y are root variables, there exists atoms $g_1 = P(x,y_1)$ and $g_2 = Q(x_1,y)$ that contain one but not the

other. P_1 may be unary, $P(x)$, or binary, $P(x, y_1)$, and similarly Q may be unary, $Q(y)$, or binary, $Q(x_2, y)$. Let A be a set consisting of a distinct constant c_z for any variable $z \in Var(c_i)$, other than x, y. We will define a shattered vocabulary $\mathbf{R}_A$ such that the shattered query d_A is still forbidden. We define directly the shattering c_ρ of the component c, Sect. 2.5, as the following function $\rho : Var(c) \to A \cup \{*\}$: $\rho(x) = \rho(y) = *$ and for every variable z other than x, y, $\rho(z) = c_z$. We define $\mathbf{R}_A$ to consists of all shattered relational symbols that occur in c_ρ. We claim: (1) c_ρ is not redundant in the shattered query d_A; (2) c_ρ has only two variables x, y, and (3) c_ρ is non-hierarchical, and, thus, d_A is still unsafe (Sect. 7.3).

We prove (1) Suppose there exists a homomorphism $c'_{\rho'} \to c_\rho$. Since in c_ρ every variable shattered to a different constant, this implies that there exists a homomorphism $c' \to c$, thus c is redundant in d, contradicting the fact that d has no redundant components. Claim (2) follows directly from the definition of ρ.

We prove claim (3), by noting that c_ρ contains the following three atoms: $S_{**}(x, y)$, $P_*(x)$ (or $P_{*c_{y_1}}(x)$), and $Q_*(y)$ (or $Q_{c_{x_2}*}(y)$).

Therefore, $c_A \equiv R_1(x), R_2(x), \ldots, S_1(x, y), S_2(x, y), \ldots, T_1(y), T_2(y), \ldots$ We apply a sequence of deterministic rewrite rules until there is a single R, a single S, and a single T. Suppose otherwise, e.g. there are two S-symbols, S_1, S_2 (and possibly others). Consider the rewrite rule $d \to d[S_1 = \texttt{true}]$. We claim that (1) $c[S_1 = \texttt{true}]$ is not redundant in $d[S_1 = \texttt{true}]$, and (2) $c[S_1 = \texttt{true}]$ is non-hierarchical. The two claims imply that $d[S_1 = \texttt{true}]$ is unsafe, contradicting the fact that d is final.

To prove (1), assume that there exists a homomorphism $f : c'[S_1 = \texttt{true}] \to c[S_1 = \texttt{true}]$. Since c has only two variables x, y and c' is leveled, the homomorphisms maps the variables $x_1, x_2, \ldots$ in c' to x and the variables $y_1, y_2, \ldots$ in c' to y. We claim that it is also a homomorphism $c' \to c$: indeed, any atom $S_1(x_i, y_j)$ is mapped to $S_1(x, y)$ which is indeed present in c. The arguments for the case when c contains two R's or two T's are similar and omitted.

This completes the proof of (A): $d \stackrel{*}{\to} h_0 \equiv R(x), S(x, y), T(y)$. Next, we prove (B), that $d > h_0$ or d *is* h_0. Here we do a case analysis. If $d \not> h_0$ then $seq(d) \leq seq(h_0) = (2, 1, 0, 0, \ldots)$ (because h_0 has 2 unary, and 1 binary relation). We first rule out the case $seq(d) < seq(h_0)$: indeed, if $seq(d) = (1, 1, 0, 0, \ldots)$ then d has two symbols R and S, or S and T. But any 2-leveled component consisting of atoms $R(x_i)$ and $S(x_i, y_j)$ minimizes to $R(x), S(x, y)$, which is safe. Thus, $seq(d) = seq(h_0)$, and the symbols in d are either R, S, T, or R_1, R_2, S, or S, T_1, T_2. If $|at(d)| > 3$ then by definition we have $d > h_0$ proving our claim, so we may assume that d has 3 atoms (it cannot have fewer than 3 because it has 3 distinct relational symbols). Also, d must have a single component c, otherwise it is a disjunction $c_1 \vee c_2$ which is symbol-disconnected (since c_1, c_2 together have only three atoms, and they must have use different symbols). Now we examine each possible vocabulary for d. If it is R, S, T, then the only non-hierarchical component with 3 atoms is $R(x), S(x, y), T(y)$; if it is R_1, R_2, S then there is no non-hierarchical component with 3 atoms: the smallest is $R_1(x_1), S(x_1, y), R_2(x_2), S(x_2, y)$ which has 4 atoms. Similarly for S, T_1, T_2. □

For a simple illustration, consider $d = R(x_1), S(x_1, y_1), S(x_2, y_1), S(x_2, y_2), T(y_2)$. We prove that it is not a forbidden query. The proof of the proposition allows us to

keep any of the three binary symbols, and shatter all other variables to constants. If we decide to keep $S(x_1, y_1)$, then the query shatters to $R_*(x_1), S_{**}(x_1, y_1), S_{a*}(y_2), S_{ab}(), T_b()$: it is disconnected, by unsafe, because of the subquery rewrite rule $\rightarrow R_*(x_1), S_{**}(x_1, y_1), S_{a*}(y_2)$ If we decide to keep $S(x_2, y_1)$, then the query shatters to $R_a(), S_{a*}(y_1), S(x_2, y_1), S_{*b}(x_2), T_b()$, which also rewrites to $R(x), S(x, y), T(y)$, up to symbol renaming.

8.3 Hardness proof for Forbidden Queries of Type 1-1

Let d be a forbidden query where both left and right components are of Type 1. We prove here that it is #P-hard, by a reduction from the #PP2CNF problem. Recall that Φ denotes the PP2CNF, Eq. 36.

8.3.1 *The Leveled Query and Database.* We proceed similarly to the hardness proof for h_2 in Sect. 5.3. We first construct a leveled query d^L, then prove that d^L.

We assume d is minimized, and denote $\mathbf{R}$ its vocabulary: it has exactly two unary relations, $R(x), T(y)$ (the left and right unary relation), and several binary relations $S(x, y)$. There are two levels, X and Y. We consider a new leveling, where both X and Y split into two new levels. With some abuse, we denote the two X-levels 0 and 1, and denote the two Y-levels also 0 and 1. Thus, there are four unary relations, R^0, R^1, T^0, T^1, and each binary relation S is leveled into S^{01}, S^{11}, S^{10} (notice there is no S^{00}). This is an incompletely leveled vocabulary (there are no symbols S^{00}), which we denote $\mathbf{R}^L$. Let m be the number of symbols in $\mathbf{R}^L$, other than R^0 and T^0; we will also refer to these symbols as $P_1, \ldots, P_m$, in other words $\mathbf{R}^L = \{R^0, P_1, \ldots, P_q, T^0\}$. Note that P_q may denote either some S^{01}, or T^1, or some S^{11}, or R^1, or some S^{10}, where $S \in \mathbf{R}$ is a binary symbol. Denote d^L the leveled query d for this vocabulary (see Sect. 6.3); by Lemma 6.10, d^L is minimized. We will prove that d^L is #P-hard; by Prop. 6.9, it follows that d is #P-hard.

Given the PP2CNF expression Φ (Eq. 36), define the sets of constants $A, B, \breve{A}_k, \breve{B}_k$ as in Eq. 24, and let D^L be following leveled database instance D^L:

$$
\begin{aligned}
&R^0 = A; \quad R^1 = \breve{A}; \quad T^0 = B; \quad T^1 = \breve{B} \\
&S^{01} = \{(a_i, \breve{b}_{ij,k}) \mid (i,j) \in E, k \in [4]\}; && \forall S \in \mathbf{R} \\
&S^{11} = \{(\breve{a}_{ij,k}, \breve{b}_{ij,k}) \mid (i,j) \in E, k \in [4]\}; && \forall S \in \mathbf{R} \\
&S^{10} = \{(\breve{a}_{ij,k}, b_i) \mid (i,j) \in E, k \in [4]\} && \forall S \in \mathbf{R}
\end{aligned}
$$

Denote $D_k(a_i, b_j)$ the set of tuples consisting only of the constants $a_i, \breve{a}_{ij,k}, \breve{b}_{ij,k}, b_j$. Let its probabilities be as follows. First, $P(R^0(a_i)) = P(T^0(b_j)) = 1/2$. To each other symbol $P_q \in \mathbf{R}^L$, associate a distinct real variable, x_q. For each pair of constants $a_i \in A, b_j \in B$, there are four tuples in the relation instance P_q, corresponding to the four slices $k = 1, 4$; for example, an instance of a binary symbol S^{01} has the four tuples $(a_i, \breve{b}_{ij,k})$, $k = 1, 4$, while the unary symbol T^1 has the four tuples $(\breve{b}_{ij,k})$, $k = 1, 4$. Set the probabilities of all four tuples to be $1 - u_q$ (same for all i, j, k), where u_q is a *distinguishing value*, to be determined below. There is one exception. Choose any binary symbol $S \in \mathbf{R}$: it is arbitrary, but fixed. Let P_{q_0} denote be its symbol with leveling 11, that is P_{q_0} is the same as S^{11}. In all four slices $k = 1, 4$, set the probability of the tuples in P_{q_0} to $1 - z_k$, for $k = 1, 4$ (same for all i, j, but distinct for k); we will show below that, for any choice of S, this is a *distinguished variable*. This completes the description of D^L. We will relate now

$P_{D^L}(\neg d^L)$ and $\#\Phi$.

We establish now a technical property of the leveled query that we need later. Fix any strict left-right path in the forbidden query d: $c_0, c_1, \ldots, c_k$. Recall that c_0 contains R, c_k contains T, and no other components contain R or T. Consider the following path in d^L:

$$c_0^{01}, c_1^{01}, \ldots, c_k^{01}, c_k^{11}, c_{k-1}^{11}, \ldots, c_0^{11}, c_0^{10}, \ldots, c_k^{10}$$

Notice that c_k^{01} and c_k^{11} are connected by the symbol T^1, and similarly c_0^{11}, c_0^{10} are connected by the symbol R^1 (see for example h_2^L in Sect. 5.3). All relational symbols in $\mathbf{R}^L$ occur on this path. R^0 and T^0 occur only in the first, and last component respectively. This implies:

LEMMA 8.9. *d^L is symbol-connected. Moreover, the symbols R^0 and P_{q_0} are connected in $d^L[T^0 = \texttt{false}]$ and in $d^L[T^0 = \texttt{true}]$, and similarly P_{q_0} and T^0 are connected in $d^L[R^0 = \texttt{false}]$ and in $d^L[R^0 = \texttt{true}]$.*

PROOF. Connectedness is immediate. The query $d^L[T^0 = \texttt{false}]$ knocks off the last component only, c_k^{10}, so R^0 and P_{q_0} are still connected (since it is in the 11-leveling portion of the path. In the query $d^L[T^0 = \texttt{true}]$ the last component is replaced by $c_k^{10}[T^0 = \texttt{true}]$. As a consequence, some other components c_i^{10} may become redundant. However, no component with leveling 11 or 01 becomes redundant, so R^0 and P_{q_0} continue to be connected. □

8.3.2 *The Dual Lineage Expressions for Queries of Type 1-1.* We define formally dual lineages and blocks.

DEFINITION 8.10. *Let Q be a query and D be a database, where each tuple t_i is annotated with a Boolean variable X_i. Let $\Phi_Q^D(X_1, \ldots, X_n)$ be the lineage of Q on D. The* dual lineage *is defined as:*

$$Y(Z_1, \ldots, Z_n) = \neg\Phi_Q^D(\neg Z_1, \ldots, \neg Z_n)$$

That is, while the lineage Φ is a positive DNF formula saying when the query is true, the dual lineage Y is a positive CNF formula saying when the query is false. The Boolean variables Z_i represent the event $t_i \notin W$, and the dual lineage Y represents the event $W \models \neg Q$. We have illustrated dual lineage expressions in Table I.

DEFINITION 8.11. *Let $d = \bigvee_i c_i$ be a disjunctive query. A* database block *for d is an instance D such that (a) every relation symbol has exactly one tuple in D, and (b) every component c_i is true in D.*

If D is a block for d, then the dual lineage is a CNF expression where (a) the Boolean variables Z are in one-to-one correspondence with relational symbols $S \in \mathbf{R}$, (b) the clauses in Y are in one-to-one correspondence with the components c_i in d.

Returning to our query d^L, we will proceed as in Sect. 5.3. By construction, for each $a_i \in A, b_j \in B$, and slice number $k \in [4]$, $D_k(a_i, b_j)$ is a block for d^L. Let Z_q denote a Boolean variable denoting the event "the tuple P_q is false" (for the given i, j, k), and define its probability as $P(Z_q) = x_q$. Similarly, let U, V be boolean variables denoting the event "$a_i \notin R^0$ and " $b_j \notin T^0$". Let $Y(U, V)$ the dual lineage

expression of d^L on the block $D_k(a_i, b_j)$; thus, $Y(U, V)$ says "d^L is false on the block $D_k(a_i, b_j)$". Note that no clause is $Y(U, V)$ is redundant: if the set of variables in one clause where included in the set of variables of another clause, then we have a homomorphism between the corresponding components, contradicting the fact that d^L is minimized.

8.3.3 *The Expansion Formula.* Two blocks $D_{k_1}(a_i, b_{j_1})$ and $D_{k_2}(a_i, b_{j_2})$ share the common tuple $R^0(a_i)$, thus the two events "d^L is false on the first block" and "d^L is false on the second block" are not independent. To make them independent, we follow exactly the method in Sect. 5.3. Fix an assignment θ of the Boolean variables in PP2CNF Φ, and let E_θ denote the event defined by Eq. 14: thus, E_θ fixes the tuples in R^0 and T^0, and imposes no restrictions on the other tuples. We will compute $P_{D^L}(\neg d^L | E_\theta)$. For any block $D_k(a_i, b_j)$, d^L must be false on that block. For each $u, v \in [2]$, let $Y_{uv} = Y[U = u, V = v]$ (where 1 denotes `false` and 2 denotes `true`), and let:

$$f_{uv} = P(Y_{uv}) \qquad\qquad u, v \in [2]$$

Thus, f_{uv} is the probability that d^L is false on the block $D_k(a_i, b_j)$, given that $a_i \in R^0$ (when $u = 1$) or $a_i \notin R^0$ (when $u = 2$) and similarly for $b_j \in T^0$ or $b_j \notin T^0$, depending on v. Recall that f_{uv} is a multilinear polynomial in $x_1 = P(Z_1), \ldots, x_m = P(Z_m)$, and we will write it as $f_{uv}(x_1, \ldots, x_m)$. However, we have chosen one distinguished variable x_{q_0}, which will be substituted with z_k, and all other variables are substituted with distinguished constants $\bar{u} \in \mathbb{R}^{m-1}$, and therefore $f_{uv}[\bar{u}](z_k) = a_{uv} \cdot z_k + b_{uv}$ is a linear function in z_k.

For any a_i, b_j, the query d^L must be false in all four blocks, $D_k(a_i, b_j)$, $k = 1, 4$, and this probability is:

$$y_{uv} = f_{uv}[\bar{u}](z_1) \cdot f_{uv}[\bar{u}](z_2) \cdot f_{uv}[\bar{u}](z_3) \cdot f_{uv}[\bar{u}](z_4) \tag{38}$$

This is a multilinear polynomial in $z_1, \ldots, z_4$. Therefore, $P(\neg d^L | E_\theta)$ is given by Eq. 22, which we don't repeat here. The probability $P(\neg d^L)$ is therefore given by the following *expansion formula*:

$$P_{D^L}(\neg d^L) = 1/2^n \sum_{\mathbf{k}} \#\mathbf{k} \cdot y_{11}^{k_{11}} \cdot y_{12}^{k_{12}} \cdot y_{21}^{k_{21}} \cdot y_{22}^{k_{22}} \tag{39}$$

Notice that this is the same as Eq. 23 (see there the definitions of $\#\mathbf{k}$.

Thus, given an oracle for $P(d^L)$ we can compute expression Eq. 39. We repeat $(m+1)^4$ times the following process: choose values for $\bar{z} = (z_1, \ldots, z_4)$, construct the probabilistic database D^L, and compute $P(d^L)$. This results in a system of $(m+1)^4$ linear equations, with unknowns $\#\mathbf{k}$. We must prove that its matrix is non-singular. Its matrix is the Kronecker product of four Vandermonde matrices: we argued informally in Sect. 5.2 (and will prove formally in Sect. 8.5) that, if the 4-dimensional function $\bar{z} \to \bar{y}$ has a non-zero Jacobian, then we can choose in PTIME the values $\bar{z}$ such that the matrix is indeed non-singular, and thus solve the system, find all coefficients $\#\mathbf{k}$ and compute $\#\Phi = \sum_{k_{22}=0} \#\mathbf{k}$. To complete the hardness proof for queries of type 1-1, we will prove that x_{q_0} is a distinguished variable for the four multilinear polynomials $f_{11}, \ldots, f_{22}$, Def. 5.7: by Prop. 5.6. We do this in the remainder of this section.

8.3.4 *Irreducible Boolean Functions*

DEFINITION 8.12. *Let Y be a Boolean function. If $Y \equiv Y_1 \wedge Y_2$ and Y_1, Y_2 do not share any Boolean variables, then we call Y_1, Y_2* independent factors *of Y. Call Y* irreducible*, if it has no independent factors other than* `true` *and Y.*

Let X_i, $i = 1, n$ be the Boolean variables in Y. Denote $x_i = P(X_i)$ the probability that X_i is true, and let $F_Y(\bar{x}) = P(Y)$ be the probability of Y. F_Y is also known as the *arithmetization* of Y. Clearly F_Y is a multilinear polynomial. We prove that any Boolean function Y admits a unique decomposition into independent irreducible factors $Y = Y_1 \wedge \cdots \wedge Y_m$, which, in turn, correspond to the irreducible factors of $F_Y = F_1 \cdot F_2 \cdots F_m$.

LEMMA 8.13. *(1) If $Y \equiv Y_1 \wedge Y_2$ is a decomposition into two independent factors, then $F_Y = F_{Y_1} \cdot F_{Y_2}$. (2) Conversely, if $F_Y = F_1 \cdot F_2$ then there exists a decomposition into two independent factors $Y \equiv Y_1 \wedge Y_2$ such that F_1, F_{Y_1} are equivalent, and F_2, F_{Y_2} are equivalent.*

PROOF. (1) Follows immediately from independence $P(Y) = P(Y_1) \cdot P(Y_2)$. (2) Conversely, suppose $F_Y = F_1 \cdot F_2$; then we will define Y_1, Y_2 such that $Y \equiv Y_1 \wedge Y_2$ where Y_1, Y_2 do not share common variables. We define Y_1 to be a function on only those variables X_i for which x_i occurs in F_1, and similarly for Y_2: since the product $F_1 \cdot F_2$ is multilinear, F_1, F_2 do not share any common variables, and therefore Y_1, Y_2 will not share any common variables either. Y_1 is defined as follows. Consider a truth assignment θ_1 on its variables; consider the corresponding assignment to the real variables, i.e. $x_i = 0$ if $\theta_1(X_i) =$ `false` and $x_i = 1$ if $\theta_1(X_i) =$ `true`. Set $Y_1[\theta_1] =$ `false` iff $F_1 = 0$. Define Y_2 similarly from F_2. We prove that $Y \equiv Y_1 \wedge Y_2$. Consider any truth assignment θ, and let θ_1, θ_2 be its restriction to the variables in Y_1 and Y_2 respectively. We will prove that $Y[\theta] =$ `false` iff $Y_1[\theta_1] =$ `false` or $Y_2[\theta_2] =$ `false`. Indeed, if $Y[\theta] =$ `false` then we have $F = 0$ for the corresponding values of the real variables x_i; hence $F_1 \cdot F_2 = 0$, so either $F_1 = 0$ or $F_2 = 0$, implying either $Y_1[\theta_1] =$ `false` or $Y_2[\theta_2] =$ `false`. The converse is similar. □

Let Y be a positive, CNF expression. Each clause is a disjunction of variables $X_{i_1} \vee X_{i_2} \vee \ldots$, and we assume no clause is redundant. The *primal graph* of Y, or the *co-occurrence graph*, $G(Y)$, has a node for each variable X_i, and an edge (X_i, X_j) whenever the two variables co-occur in a clause. It follows that Y is irreducible iff G is connected. Indeed, if G is disconnected we can write $Y \equiv Y_1 \wedge Y_2$ where Y_1 and Y_2 are sets of clauses that do not share variables; conversely if $Y \equiv Y_1 \wedge Y_2$ then, by uniqueness of the CNF representation for positive formulas, the graph $G(Y)$ is the disjoint union of $G(Y_1)$ and $G(Y_2)$. This implies:

COROLLARY 8.14. *Let F_Y be the arithmetization of a positive, CNF formula Y. (1) F_Y is irreducible iff $G(Y)$ is connected. (2) If Y has m connected components then F_Y has m irreducible factors, $F_Y = F_1 \cdot F_2 \cdots F_m$, which are the arithmetizations of the formulas corresponding to these components.*

We carry over the notion of distinguished variable for multilinear polynomials (Def. 5.7) to Boolean functions.

DEFINITION 8.15. *Let $Y_1, \ldots, Y_m$ be Boolean functions. Call a Boolean variable Z* distinguished *if each function Y_i depends on Z and, denoting Y_i^0 the independent, irreducible factor of Y_i that depends on Z, all m functions $Y_1^0, \ldots, Y_m^0$ are inequivalent.*

By Corollary 5.9 Z is distinguished iff its corresponding real variable z is distinguished for the arithmetizations of the Boolean functions.

LEMMA 8.16. *Let $Y(U)$ be an irreducible positive, Boolean formula, where U is one of its Boolean variables. Denote $Y_1 = Y[U = \texttt{false}]$ and $Y_2 = Y[U = \texttt{true}]$. Then Y_1, Y_2 have no common independent factors.*

PROOF. Suppose otherwise, and denote Y^0 a common factor. Then we have $Y_1 = Y^0 \wedge Y_1'$ and $Y_2 = Y^0 \wedge Y_2'$ where Y_1' and Y_2' have no Boolean variables in common with Y^0. Since Y is positive, $Y \equiv Y_1 \vee (U \wedge Y_2) \equiv (Y^0 \wedge Y_1') \vee (Y^0 \wedge U \wedge Y_2') \equiv Y^0 \wedge (Y_1' \vee U \wedge Y_2')$, contradicting the fact that Y is irreducible □

We say that two variables X_1, X_2 are disconnected in Y if there exists a factorization $Y = Y_1 \wedge Y_2$ into independent factors such that X_1 occurs in Y_1 and X_2 occurs in Y_2. Equivalently, X_1, X_2 are disconnected in the primal graph $G(Y)$.

COROLLARY 8.17. *Let $Y(U, V)$ be an irreducible Boolean formula, where U, V are two of its variables. Let Z be another variable, and assume that U, Z are connected in both $Y[V = \texttt{false}]$ and $Y[V = \texttt{true}]$, and that Z, V are connected in both $Y[U = \texttt{false}]$ and $Y[U = \texttt{true}]$. Denote $Y_{uv} = Y[U = u, V = v]$, $u, v \in [2]$ (where 1 stands for* `false` *and 2 for* `true`*). Then Z is distinguished in the four functions Y_{uv}, $u, v \in [2]$.*

PROOF. Denote Y_{uv}^0 the irreducible independent factor of Y_{uv} that contains Z. We apply the previous lemma four times. First, we apply it to the irreducible independent factor of $Y[V = v]$ that contains U and Z, for each $v \in [2]$, then we apply it to the irreducible independent factor of $Y[U = u]$ for each $u \in [2]$. We obtain:

$$Y_{12}^0 \not\equiv Y_{22}^0 \qquad Y_{21}^0 \not\equiv Y_{22}^0$$
$$Y_{12}^0 \not\equiv Y_{22}^0 \qquad Y_{21}^0 \not\equiv Y_{22}^0$$

To prove $Y_{12}^0 \not\equiv Y_{21}^0$ it suffices to notice that $Y_{11} \equiv Y_{12} \wedge Y_{21}$: if $Y_{12}^0 \equiv Y_{21}^0$ then both would be equivalent to Y_{11}^0, which we already showed is not the case. □

8.3.5 *The Distinguished Variables.* Now we complete the hardness proof of d^L. Consider the dual lineage of d^L on a block $D_k(a_i, b_j)$, which we denoted $Y(U, V)$. We verify that the variables U, V and $Z = Z_{q_0}$ meet the conditions of Corollary 8.17: this follows from Lemma 8.9, since U, Z_{q_0}, V correspond to the relational symbols R^0, P_{q_0}, T^0. Thus, Z_{q_0} is a distinguished Boolean variable, and therefore x_{q_0} is a distinguished variable for the polynomials $f_{11}, \ldots, f_{22}$. By definition, there are distinguishing values $\bar{u}$ such that $f_{11}[\bar{u}](z), \ldots, f_{22}[\bar{u}](z)$ are non-degenerate. Finally, by Prop. 5.6 the Jacobian $(z_1, \ldots, z_4) \mapsto (y_{11}, \ldots, y_{22})$ is nonzero, where y_{uv} is given by Eq. 38. It follows that we can choose values of the variables $\bar{z}$ s.t. linear equations given by Eq. 39 has a non-singular matrix, proving that we can use the oracle for $P(\neg d^L)$ to compute all coefficients $\#\mathbf{k}$, and finally obtain $\#\Phi = \sum_{\mathbf{k}: k_{22}=0} \#\mathbf{k}$. This proves that d^L is #P-hard.

8.4 Hardness proof for Forbidden Queries of Type 2-2

By definition, a forbidden query of type 2-2 has no unary symbols, preventing us for using the conditional probability $P(\neg d|E_\theta)$ as for the type 1-1 queries. Because of that, we need a new idea, which replaces the conditional probability with Mobius inversion formula. We give the proof in more details than for queries of Type 1-1, because it is more subtle. As a running example, we will use the following query:

$$d_\diamond = S_1(x, y_1), S_2(x, y_2) \vee S_1(x_1, y), S_2(x_2, y)$$

While $d_\diamond$ is unsafe, symbol-connected, and has two levels, it is actually not a forbidden query, as we show in Example 8.37: we use it as a surrogate for the more verbose $d_{\texttt{diamond}}$ shown at the end of Sect. 8.1.

8.4.1 *The Leveled Query and Database.* Let $d = \bigvee_{i=1,k} c_i$ be a forbidden query of type 2-2; the vocabulary, $\mathbf{R}$, consists only of binary relations. Let $n_X = \max_i |Var_X(c_i)|$ and $n_Y = \max_i |Var_Y(c_i)|$. Consider the following leveled vocabulary $\mathbf{R}^L$. Level X is split into levels $0, 1, \ldots, n_X$, and level Y is split into levels $0, 1, \ldots, n_Y$. For each relation $S \in \mathbf{R}$ there are the following levelings:

$$S^{0\tau_2}, \qquad S^{\tau_1\tau_2}, \qquad S^{\tau_1 0} \qquad \forall \tau_1 \in [n_X], \forall \tau_2 \in [n_Y]$$

There is no leveling S^{00}. Let d^L denote the leveling of d according to $\mathbf{R}^L$. As we discussed in Sect. 6, a leveling of a component c_i is given by a function ρ from its variables to levels; in our case the function has two parts, $\rho = (\rho_X, \rho_Y)$, where $\rho_X : Var_X(c_i) \to \{0\} \cup [n_X]$ and $\rho_Y : Var_Y(c_i) \to \{0\} \cup [n_Y]$; the corresponding leveling is denoted c_i^ρ, or $c_i^{\rho_X\rho_Y}$. If the image of ρ_X is 0, i.e. $\rho_X(x_j) = 0$ for all variables $x_j \in Var_X(c_i)$, then we write the leveling as $c_i^{0\rho_Y}$; similarly for $c_i^{\rho_X 0}$. There are no levelings c_i^{00} because there are no symbols S^{00}. Accordingly, we classify the components of d^L into three sets:

$$\begin{aligned}
\mathcal{C}^{0*} &= \{c_i^{0\rho_Y} \mid i = 1, k; Im(\rho_Y) \neq \{0\}\} \\
\mathcal{C}^{**} &= \{c_i^{\rho_X\rho_Y} \mid i = 1, k; Im(\rho_X) \neq \{0\}, Im(\rho_Y) \neq \{0\}\} \\
\mathcal{C}^{*0} &= \{c_i^{\rho_X 0} \mid i = 1, k; Im(\rho_X) \neq \{0\}\}
\end{aligned}$$

Therefore, we can write the leveled query as:

$$d^L = d^{0*} \vee d^{**} \vee d^{*0}$$

where d^{0*} is the disjunction of all components in $\mathcal{C}^{0*}$, etc.

Next, we describe the database instance D^L, associated to a PP2CNF expression Φ given by Eq. 36 at the beginning of this section. Let s, s_1, s_2 be three numbers, which depend only on the query d; they represent slice numbers, and will be defined later. Then:

$$D^L = \bigcup_{(i,j)\in E, k=1,s} D_k(a_i, b_j) \cup \bigcup_{i\in[n_1], k=1,s_1} D_k(a_i, \cdot) \cup \bigcup_{j\in[n_2], k=1,s_2} D_k(\cdot, b_j)$$

where for each edge $(i, j) \in E$ and for each slice numbers $k \in [s], k \in [s_1]$, and

$k \in [s_2]$, the three sets mentioned above are:

$$D_k(a_i, b_j) = \bigcup_{S,\tau_1,\tau_2} \{S^{0\tau_2}(a_i, \breve{b}_{ij,k}), S^{\tau_1\tau_2}(\breve{a}_{ij,k}, \breve{b}_{ij,k}), S^{\tau_1 0}(\breve{a}_{ij,k}, b_j)\}, k \in [s]$$

$$D_k(a_i, \cdot) = \bigcup_{S,\tau_1,\tau_2} \{S^{0\tau_2}(a_i, \breve{b}_{i\cdot,k}), S^{\tau_1\tau_2}(\breve{a}_{i\cdot,k}, \breve{b}_{i\cdot,k}), S^{\tau_1 0}(\breve{a}_{i\cdot,k}, \dot{b}_{i,k})\}, k \in [s_1]$$

$$D_k(\cdot, b_j) = \bigcup_{S,\tau_1,\tau_2} \{S^{0\tau_2}(\dot{a}_{j,k}, \breve{b}_{\cdot j,k}), S^{\tau_1\tau_2}(\breve{a}_{\cdot j,k}, \breve{b}_{\cdot j,k}), S^{\tau_1 0}(\breve{a}_{\cdot j,k}, b_j)\}, k \in [s_2]$$

In each case, the union is taken over $S \in \mathbf{R}$, $\tau_1 \in [n_X]$ and $\tau_2 \in [n_Y]$. All sets of the form $D_k(a_i, b_j)$, $D_k(a_i, \cdot)$, or $D_k(\cdot, b_j)$ are isomorphic, and each is a block for d^L (Def. 8.11). The first block $D_k(a_i, b_j)$ is similar to that used for type 1-1 queries, but now we use more than 4 slices, because we need more than 4 distinguished variables. There is a block $D_k(a_i, b_j)$ for each edge $(i, j) \in E$, and each slice number $k = 1, s$. It contains three kinds of tuples: from A to $\breve{B}$, from $\breve{A}$ to $\breve{B}$ and from $\breve{A}$ to B, (see Eq. 24 for the definition of $A, B, \breve{A}, \breve{B}$). The three kinds of tuples correspond to levelings $0\tau_2$, $\tau_1\tau_2$, and $\tau_1 0$, where $\tau_1 \in [n_X], \tau_2 \in [n_Y]$, thus the block $D_k(a_i, b_j)$ has a zig-zag-zig shape. The other two blocks $D_k(a_i, \cdot)$ and $D_k(\cdot, b_j)$ are isomorphic to $D_k(a_i, b_j)$, and we call them "dangling" blocks: one can think of $D_k(a_i, \cdot)$ as corresponding to an edge $(i, \cdot)$ that starts at i and ends no-where, and similarly fro $D_k(\cdot, b_j)$. The blocks have two important properties: (1) no two blocks share any tuples and, (2) the only constants that occur in more than one block are a_i and b_j (the dangling endpoints $\dot{b}_{i,k}$ and $\dot{a}_{j,k}$ are not shared).

The tuple probabilities as for queries of type 1-1. We rename all symbols in $\mathbf{R}^L$ to $P_1, \ldots, P_m$, in order be able to refer to them using a single index. Consider a block $D_k(a_i, b_j)$. We associate to each symbol P_q a variable x_q; the probability of the unique tuple P_q in this block is defined as $1 - u_q$, where u_q is a distinguishing value for the variable x_q, to be determined below. This probability depends only on the symbol P_q, and not on i, j, k. There is one exception: we choose an arbitrary, but fixed, binary relation S, and an arbitrary pair of non-zero levelings, say 11, and designate S^{11} the distinguished tuple. Let P_{q_0} be the distinguished tuple (i.e. P_{q_0} and S^{11} are the same symbol). We define the probability of the distinguished tuples to be $1 - z_k$, where k is the slice number of the block. This probability is independent of i, j. Since there are s slices, we have s variables $z_1, \ldots, z_s$. We repeat the same process for the left dangling blocks $D_k(a_i, \cdot)$, introducing s_1 fresh variables $z'_1, \ldots, z'_{s_1}$, then for the right dangling blocks $D_k(\cdot, b_j)$, introducing s_2 variables $z''_1, \ldots, z''_{s_2}$. This completes the definition of the tuple probabilities.

In the rest of this section we show how to use an oracle for $P_{D^L}(\neg d^L)$ to compute $\#\Phi$, thus reducing the #PP2CNF problem to d^L. By Prop. 6.9 $d^L \leq^{\text{FO}}_{\text{lin}} d$, which implies that d is #P-hard.

8.4.2 *The Events $E^{0*}_{\mathbf{u}}$ and $E^{*0}_{\mathbf{v}}$.* Given a possible world $W \subseteq D^L$, we denote $W_k(a_i, b_j) = D_k(a_i, b_j) \cap W$, and similarly for $W_k(a_i, \cdot)$ and $W_k(\cdot, b_j)$. To compute $P(\neg d^L)$, we start with a simple observation:

LEMMA 8.18. *Let $c^\rho \in \mathcal{C}^{**}$. Then $W \models c^\rho$ iff $\exists i, j, k$ s.t. $W_k(a_i, b_j) \models c^\rho$ or $W_k(a_i, \cdot) \models c^\rho$ or $W_k(\cdot, b_j) \models c^\rho$.*

PROOF. Clearly, if c^ρ is true in some block, say $W_k(a_i, b_j)$, then it is true in the world W. We prove the converse. Assume c^ρ has a root variable x (the case when it has root variable y is similar). We cannot have $\rho_X(x) = 0$, because x is the single variable in level X, and that would imply $Im(\rho_X) = \{0\}$ contradicting $c^\rho \in \mathcal{C}^{**}$, thus $\rho_X(x) > 0$. Thus, any valuation that maps c^ρ to W must map x either to a constant of the form $\breve{a}_{ij,k}$, or $\breve{a}_{i\cdot,k}$, or $\breve{a}_{\cdot j,k}$ (but not a_i): by construction of the blocks, it maps all atoms in c^ρ to $W_k(a_i, b_j)$, or to $W_k(a_i, \cdot)$, or to $W_k(\cdot, b_j)$ respectively. □

However, a component in $\mathcal{C}^{0*}$ may span multiple blocks, and similarly for components in $\mathcal{C}^{*0}$. We cannot compute $P(\neg d^L)$ by conditioning on the events E_θ, as we did for queries of type 1-1, because we do not have the unary symbols. Instead, we will use a different kind of events, $E^{0*}_{\mathbf{u}}$ and $E^{*0}_{\mathbf{v}}$. We illustrate with our running example:

EXAMPLE 8.19. *The leveling of $d_\diamond$ is $d^L_\diamond = d^{0*}_\diamond \vee d^{**}_\diamond \vee d^{*0}_\diamond$, where:*

$$
\begin{aligned}
d^{0*}_\diamond &= \bigvee_{\tau_1,\tau_2\in[2]} S_1^{0\tau_1}(x,\breve{y}_1), S_2^{0\tau_2}(x,\breve{y}_2) \quad \vee \quad \bigvee_{\tau\in[2]} S_1^{0\tau}(x_1,\breve{y}), S_2^{0\tau}(x_2,\breve{y}) \\
d^{**}_\diamond &= \bigvee_{\tau_2,\tau\in[2]} S_1^{0\tau}(x_1,\breve{y}), S_2^{\tau_2\tau}(\breve{x}_2,\breve{y}) \quad \vee \quad \bigvee_{\tau_1,\tau\in[2]} S_1^{\tau_1\tau}(\breve{x}_1,\breve{y}), S_2^{0\tau}(x_2,\breve{y}) \\
&\vee \bigvee_{\tau,\tau_1,\tau_2\in[2]} S_1^{\tau\tau_1}(\breve{x},\breve{y}_1), S_2^{\tau\tau_2}(\breve{x},\breve{y}_2) \quad \vee \quad \bigvee_{\tau_1,\tau_2,\tau\in[2]} S_1^{\tau_1\tau}(\breve{x}_1,\breve{y}), S_2^{\tau_2\tau}(\breve{x}_2,\breve{y}) \\
&\vee \bigvee_{\tau,\tau_1\in[2]} S_1^{\tau\tau_1}(\breve{x},\breve{y}_1), S_2^{\tau 0}(\breve{x},y_2) \quad \vee \quad \bigvee_{\tau,\tau_2\in[2]} S_1^{\tau 0}(\breve{x},y_1), S_2^{\tau\tau_2}(\breve{x},\breve{y}_2) \\
d^{*0}_\diamond &= \bigvee_{\tau\in[2]} S_1^{\tau 0}(\breve{x},y_1), S_2^{\tau 0}(\breve{x},y_2) \quad \vee \quad \bigvee_{\tau_1,\tau_\in[2]} S_1^{\tau_1 0}(\breve{x}_1,y), S_2^{\tau_2 0}(\breve{x}_2,y)
\end{aligned}
$$

Consider first a component in $d^{**}_\diamond$, for example $c = S_1^{01}(x_1,\breve{y}), S_2^{21}(\breve{x}_2,\breve{y})$. By Lemma 8.18, to check that c is false in a world W, it suffices to check that it false in every block of W. Indeed, suppose it is true in W, and assume the valuation maps $\breve{y}$ to $\breve{b}_{ij,k}$: then it must map the two atoms to $S_1^{01}(a_i,\breve{b}_{ij,k}), S_2^{21}(\breve{a}_{ij,k},\breve{b}_{ij,k})$, and both are in the same block $W_k(a_i,b_j)$; similarly if $\breve{y}$ is mapped to $\breve{b}_{i\cdot,k}$ or $\breve{b}_{\cdot j,k}$.

Consider now a component in $d^{0*}_\diamond$, say $c = S_1^{01}(x,\breve{y}_1), S_2^{01}(x,\breve{y}_2)$. Now it is not sufficient to check that c is false in every block: indeed, a valuation may map the first atom to $S_1^{01}(a_i,\breve{b}_{ij_1,k_1})$, and the second atom to $S_2^{01}(a_i,\breve{b}_{ij_2,k_2})$, which are in different blocks $W_{k_1}(a_i,b_{j_1})$ and $W_{k_2}(a_i,b_{j_2})$. To justify our solution, suppose for a moment that $|A| = 1$, that is, we have a single constant $a_i \in A$. Denote E_1 the following event (a property on the random world $W \subseteq D^L$):

$$E_1 : \qquad W \not\models \exists\breve{y}_1.S_1^{01}(a_i,\breve{y}_1)$$

By requiring the event E_1 to hold, we make a commitment that $S_1^{01}(x,\breve{y}_1)$ will be false at $x = a_i$. We still need to check that the entire query $d^L_\diamond$ is false, but now we can simplify its component $S_1^{01}(x,\breve{y}_1), S_2^{01}(x,\breve{y}_2)$ to just $S_1^{01}(x,\breve{y}_1)$. This new component is false in W iff it is false in all blocks $D_k(a_i,b_j)$,, $D_k(a_i,\cdot)$, or

$D_k(\cdot, b_j)$. *Similarly, denote* E_2 *the event:*

$$E_2 : \qquad W \not\models \exists \breve{y}_2.S_2^{01}(a_i, \breve{y}_2)$$

In this case, the component simplifies to $S_2^{01}(x, \breve{y}_2)$. *Thus, to compute* $P(\neg d_\diamond^L)$, *we will reason by cases, considering the events* E_1 *and* E_2. *This is similar to what we did for Type 1-1 queries in Sect. 5, where we conditioned on* $R(a_i)$ *and* $\neg R(a_i)$. *However, there are two important differences from Type 1-1 queries: the events* E_1, E_2 *are not exclusive, and they are not exhaustive. Here we make an important observation: if* $d_\diamond^L$ *is false, then either* E_1 *is true, or* E_2 *is true, or both are true. To see this, suppose that both* E_1 *and* E_1 *are false: then* $\exists \breve{y}_1.S_1^{01}(a_i, \breve{y}_1)$ *is true in the world* W, *and also* $\exists \breve{y}_2.S_2^{01}(a_i, \breve{y}_2)$ *is true, implying that* $S_1^{01}(x, \breve{y}_1), S_2^{01}(x, \breve{y}_2)$ *is true, contradicting the fact that* $d_\diamond^L$ *is false. Therefore, in our toy example where there is a single constant* a_i, $P(\neg d_\diamond^L) = P(\neg d_\diamond^L \wedge (E_1 \vee E_2))$. *We compute the latter using the inclusion-exclusion formula. This achieves our goal: in* $P(E_1 \wedge (\neg d_\diamond^L))$, *our component simplifies to just* $S_1^{01}(x, \breve{y}_1)$, *so the blocks are again independent, and similarly in* $P(E_2 \wedge (\neg d_\diamond^L))$. *In essence, we will replace the Shannon expansion formula* $\sum_\theta P(\cdot|E_\theta)P(E_\theta)$ *that we used for Type 1-1 queries, with Mobius' inversion formula* $-\sum_u \mu(u, \hat{1})P(\cdot \wedge E_u)$.

We describe now the events $E_{\mathbf{u}}^{0*}$ and $E_{\mathbf{v}}^{*0}$ that generalize the example.

DEFINITION 8.20. *Let* c_i *be a component with the following* X*-subcomponents* $sc_X = \{c_{i1}, c_{i2}, \ldots\}$ *(Sect. 7.2). The* X-split *of* c_i *is* $\sigma_X(c_i) = \bigwedge_j c_{ij}[x_j/X]$, *where* x_j *is a fresh variable for each subcomponent* c_{ij} *of* c_i. *If* $d = \bigvee_i c_i$, *then we define its* X*-split to be* $\sigma_X(d) = \bigvee_i \sigma_X(c_i)$. *The* Y*-split is defined similarly.*

For a simple example, consider $c = S(x, y_1), S_1(x, y_1), S(x, y_2), S_2(x, y_2)$; then $\sigma_X(c) = S(x_1, y_1), S_1(x_1, y_1), S(x_2, y_2), S_2(x_2, y_2)$.

For any constant a, $c_i[a/X] \equiv \sigma_X(c_i)[a/X]$; that is, the two queries become equivalent if we substitute all variables x_i in level X with the same constant a. Note that $\sigma_X(c_i)$ is, in general, a conjunctive query, which may be disconnected. Define:

$$Q^{0*} = \sigma_X(d^{0*}) \qquad\qquad Q^{*0} = \sigma_Y(d^{*0})$$

Thus, Q^{0*} is a union of conjunctive queries of two kinds: if $c_i = S_1(x, y_{i_1}), S_2(x, y_{i_2}), \ldots$ has root variable x, then $\sigma_X(c_i^{0\rho_Y}) = S_1(x_{i_1}, y_{i_1}), S_2(x_{i_2}, y_{i_2}), \ldots$ and is possible disconnected (unless c_i has also a root variable y, i.e. $i_1 = i_2 = \ldots$); if $c_i = S_1(x_{i_1}, y), S_2(x_{i_2}, y), \ldots$ has root variable y, then $\sigma_X(c_i^{0\rho_Y}) = S_1(x, y), S_2(x, y), \ldots$ (because c_i has a unique X-subcomponent). Both Q^{0*} and Q^{*0} are Unions of Conjunctive Queries and their connection to d^{0*}, d^{*0} is captured by:

$$Q^{0*}[a/X] = d^{0*}[a/X] \qquad\qquad Q^{*0}[b/Y] = d^{*0}[b/Y]$$

for any constants a, b. Write these queries as CNF query expressions (Sect. 2.6):

$$Q^{0*} = \bigwedge_{j=1,l} d_j^{0*} \qquad\qquad Q^{*0} = \bigwedge_{j=1,r} d_j^{*0} \tag{40}$$

We define the *left lattice* $\hat{L}$ and the *right lattice* $\hat{R}$ to be their CNF lattices, $\hat{L} = L(Q^{0*})$, $\hat{R} = L(Q^{*0})$ (Def. 4.8). In this section we follow standard notation [Stanley 1997], and denote a lattice with $\hat{L}$; when we drop the hat, we mean

$L = \hat{L} - \{\hat{1}\}$, which is a meet semilattice. For each $u \in L$, we let d_u^{0*} denote the disjunctive query associated to the lattice element u (see the definition of d_u, right after Def. 4.8); recall that $d_j^{0*} = d_{\{j\}}^{0*}$ represents a coatom in the lattice. For $v \in R$, we denote similarly d_v^{*0}.

Every component in d_u^{0*} and d_r^{*0} has both x and y as root variables, i.e. it looks like this $S_1(x, y), S_2(x, y), \ldots$. Therefore, if the component is true in a world, then it is true in a block of that world, because no two blocks share two constants. Recall from Eq. 24 that $A = \{a_i \mid i \in [n_X]\}$ and $B = \{b_j \mid j \in [n_Y]\}$:

DEFINITION 8.21. *For any lattice element $u \in L$, and constant $a_i \in A$, denote $E_u^{0*}(a_i)$ the following event on a random world W:*

$$E_u^{0*}(a_i): \qquad W \not\models d_u^{0*}[a_i/X]$$

Let $\mathbf{u} \in L^A$ be a function from A to L. Denote the following event: $E_{\mathbf{u}}^{0*} = \bigwedge_{a_i \in A} E_{\mathbf{u}(a_i)}^{0*}(a_i)$. Define similar events on the "right", $E_v^{*0}(b_j)$ and $E_{\mathbf{v}}^{*0}$.

LEMMA 8.22. *Let W be a world such that $W \models \neg d^L$, and let $a_i \in A$. Then there exists $u \in L$ such that W satisfies the event $E_u^{0*}(a_i)$.*

PROOF. (The proof generalizes our discussion in Example 8.19.) We prove that the lattice element u can be taken of the form $u = \{j\}$ (i.e. a co-atom in the lattice $\hat{L}$); in this case, $d_u^{0*} = d_j^{0*}$ is one of the disjunctive queries in Eq. 40. Suppose the contrary, that $W \models d_1^{0*}[a_i/X]$, ..., $W \models d_l^{0*}[a_i/X]$. Then $W \models d_1^{0*}[a_i/X] \wedge \ldots \wedge d_l^{0*}[a_i/X]$, which means that $W \models Q^{0*}[a_i/X]$. By $Q^{0*}[a_i/X] \equiv d^{0*}[a_i/X]$, we obtain $W \models d^{0*}$, hence $W \models d^L$, which is a contradiction. □

Therefore, if we fix a_i and assume that d^L is false, then one of the events $E_u^{0*}(a_i)$, $u \in L$ must hold.

LEMMA 8.23. *Fix $a_i \in A$; the set of events $\{E_u^{0*}(a_i) \mid u \in L\}$ is a meet semilattice isomorphic to L. $\{E_{\mathbf{u}} \mid \mathbf{u} \in L^A\}$ is a meet semilattice isomorphic to L^A.*

PROOF. We prove that $u \mapsto E_u^{0*}(a_i)$ is an order isomorphism. If $u, v \in L$ and $u \leq v$, then $d_u^{0*} \Leftarrow d_v^{0*}$, implying $E_u^{0*}(a_i) \subseteq E_v^{0*}(a_i)$, thus the mapping is monotone. If $u \not\leq v$ then $d_u^{0*} \not\Leftarrow d_v^{0*}$ and by Prop. 2.13 there exists a component c in d_v^{0*} s.t. $d_u^{0*} \not\Leftarrow c$. $D_k(a_i, b_j)$ is a block, and has one tuple for each relational symbol. Denote W be the world consisting of exactly the tuples in c. We have $W \models c$, thus $W \not\models d_v^{0*}$, but $W \not\models d_u^{0*}$ (otherwise d_u^{0*} has a component c' s.t. $W \models c'$, which implies $c' \Leftarrow c$), implying $E_u^{0*}(a_i) \not\subseteq E_v^{0*}(a_i)$, which proves that the mapping is injective. □

LEMMA 8.24. *Let $\mathbf{u} \in L^A$ and $\mathbf{v} \in R^B$. Conditioned on the event $E_{\mathbf{uv}} = E_{\mathbf{u}}^{0*} \wedge E_{\mathbf{v}}^{*0}$, the event $W \models \neg d^L$ is equivalent to the following conjunction of events:*

$$\bigwedge_{a_i \in A, b_j \in B, k \in [s]} W_k(a_i, b_j) \not\models d^{**} \wedge \bigwedge_{a_i \in A, k \in [s_1]} W_k(a_i, \cdot) \not\models (d^{**} \vee d^{*0}) \wedge \bigwedge_{b_j \in B, k \in [s_1]} W_k(\cdot, b_j) \not\models (d^{0*} \vee d^{**})$$

PROOF. If d^L is false in W, then it is obviously false in every block. Conversely, suppose $d^L = d^{0*} \vee d^{**} \vee d^{*0}$ is true in W. If d^{**} is true, then it is true in some block, by Lemma 8.18. Assume w.l.o.g. that d^{0*} is true. For all $a_i \in A$, the event $E_{\mathbf{u}}^{0*}$ implies that $d_{\mathbf{u}(a_i)}^{0*}[a_i/X]$ is false, hence $d^{0*}[a_i/X]$ is false. Therefore one of

$d^{0*}[\breve{a}_{ij,k}/X]$, $d^{0*}[\breve{a}_{i\cdot,k}/X]$, $d^{0*}[\breve{a}_{\cdot j,k}/X]$, or $d^{0*}[\dot{a}_{j,k}/X]$ is true; by construction, the query is true in $D_k(a_i, b_j)$, or $D_k(a_i, \cdot)$, or $D_k(\cdot, b_j)$, or $D_k(\cdot, b_j)$. □

For each $u \in L, v \in R$, denote the following queries:

$$d_{uv} = d_u^{0*} \vee d^{**} \vee d_v^{*0} \qquad d_{u\cdot} = d_u^{0*} \vee d^{**} \vee d^{*0} \qquad d_{\cdot v} = d^{0*} \vee d^{**} \vee d_v^{*0}$$

COROLLARY 8.25. *The event $W \models \neg d^L$ is equivalent to the following:*

$$\bigvee_{\mathbf{u}\in L^A, \mathbf{v}\in R^B} \left(\bigwedge_{a_i,b_j,k} W_k(a_i, b_j) \not\models d_{uv} \wedge \bigwedge_{a_i,k} W_k(a_i, \cdot) \not\models d_{u\cdot} \wedge \bigwedge_{b_j,k} W_k(\cdot, b_j) \not\models d_{\cdot v} \right)$$

PROOF. By Lemma 8.22 we have $(W \models \neg d_L) \equiv \bigvee_{\mathbf{u}\in L^A, \mathbf{v}\in R^B} [E_{\mathbf{uv}} \wedge (W \models \neg d_L)]$, and the rest follows from the previous lemma. □

EXAMPLE 8.26. *We illustrate this on our running example (Example 8.19):*

$$d_\diamond^{0*} = \bigvee_{\tau_1,\tau_2\in[2]} S_1^{0\tau_1}(x, \breve{y}_1), S_2^{0\tau_2}(x, \breve{y}_2) \vee \bigvee_{\tau\in[2]} S_1^{0\tau}(x_1, \breve{y}), S_2^{0\tau}(x_2, \breve{y})$$

$$\begin{aligned}
Q^{0*} &= \bigvee_{\tau_1,\tau_2\in[2]} S_1^{0\tau_1}(x_1, \breve{y}_1), S_2^{0\tau_2}(x_2, \breve{y}_2) \vee \bigvee_{\tau\in[2]} S_1^{0\tau}(x, \breve{y}), S_2^{0\tau}(x, \breve{y}) \\
&= (S_1^{01}(x_1, \breve{y}_1) \vee S_1^{02}(x_1, \breve{y}_1) \vee \bigvee_{\tau\in[2]} S_1^{0\tau}(x, \breve{y}), S_2^{0\tau}(x, \breve{y})) \\
&\wedge (S_2^{01}(x_2, \breve{y}_2) \vee S_2^{02}(x_2, \breve{y}_2) \vee \bigvee_{\tau\in[2]} S_1^{0\tau}(x, \breve{y}), S_2^{0\tau}(x, \breve{y})) \\
&= (S_1^{01}(x_1, \breve{y}_1) \vee S_1^{02}(x_1, \breve{y}_1)) \wedge (S_2^{01}(x_2, \breve{y}_2) \vee S_2^{02}(x_2, \breve{y}_2)) = d_1^{0*} \wedge d_2^{0*}
\end{aligned}$$

The left semilattice L has three elements and their associated queries are the following:

$$\begin{aligned}
d_1^{0*} &= S_1^{01}(x, \breve{y}) \vee S_1^{02}(x, \breve{y}) \qquad d_2^{0*} = S_2^{01}(x, \breve{y}) \vee S_2^{02}(x, \breve{y}) \\
d_{12}^{0*} &= d_1^{0*} \vee d_2^{0*} = S_1^{01}(x, \breve{y}) \vee S_1^{02}(x, \breve{y}) \vee S_2^{01}(x, \breve{y}) \vee S_2^{02}(x, \breve{y})
\end{aligned}$$

Lemma 8.22 says that, if $d_\diamond^L$ is false in a world W, then, for every $a_i \in A$, either $d_1^{0*}[a_i/x]$, or $d_2^{0*}[a_i/x]$, or both (meaning $d_{12}^{0*}[a_i/x]$) are false in W.

The right lattice R is isomorphic to L, and has three elements, $d_1^{*0}, d_2^{*0}, d_{12}^{*0}$. If $d_\diamond^L$ is false, then for each block $D_k(a_i, b_j)$, there are 9 cases for its endpoints: one of d_u^{0*}, $u \in L$, is false on a_i, and one of d_v^{*0}, $v \in R$ is false on b_j. Thus, one of the 9 queries $d_{uv} = d_u^{0*} \vee d^{**} \vee d_v^{*0}$ is false on the block $D_k(a_i, b_j)$. These are "simple" queries, since each component has exactly two variables, x, y.

The next step is to compute $P(\neg d^L)$ by applying Mobius' inversion formula to Corollary 8.25. The number of terms in the disjunction $\bigvee_{\mathbf{u},\mathbf{v}}$ is exponential in the size of the PP2CNF Φ, yet we must express $P(\neg d^L)$ with a formula of polynomial size. First, we compute the probabilities of the events in each block, $P(W_k(a_i, b_j) \not\models d_{uv})$ etc, and the Mobius function for the lattice of events $(\mathbf{u}, \mathbf{v}) \in L^A \times R^B$.

8.4.3 *The Multilinear Polynomials.* For each $u \in L, v \in R$ define:

$$\begin{aligned} f_{uv}(\bar{x}) =& P_{D_k(a,b)}(\neg d_{uv}) \\ f_{u\cdot}(\bar{x}') =& P_{D_k(a,\cdot)}(\neg d_{u\cdot}) \\ f_{\cdot v}(\bar{x}'') =& P_{D_k(\cdot,b)}(\neg d_{\cdot v}) \end{aligned} \tag{41}$$

Recall that each tuple in the block $D_k(a,b)$ has an associated real variable x_q; therefore, all three functions above are multilinear polynomials in $\bar{x} = (x_1, x_2, \ldots)$. Once we choose the distinguished tuple, each variable is substituted with a constant, $x_q = u_q$, except for the distinguished variable which becomes $x_{q_0} = z_k$. Therefore, denoting $D(a,b) = \bigcup_{k=1,s} D_k(a,b)$, $D(a,\cdot) = \bigcup_{k=1,s_1} D_k(a,\cdot)$, $D(\cdot,b) = \bigcup_{k=1,s_2} D_k(\cdot,b)$:

$$\begin{aligned} y_{uv} =& P_{D(a,b)}(\neg d_{uv}) = \prod_{k=1,s} f_{uv}[\bar{u}](z_k) \\ y_{u\cdot} =& P_{D(a,\cdot)}(\neg d_{u\cdot}) = \prod_{k=1,s_1} f_{u\cdot}[\bar{u}](z'_k) \\ y_{\cdot v} =& P_{D(\cdot,b)}(\neg d_{\cdot v}) = \prod_{k=1,s_2} f_{\cdot v}[\bar{u}](z''_k) \end{aligned}$$

In the first line z_k substitutes the distinguished variable x_{q_0} in f_{uv}, and the other variables are substituted with distinguished constants; similarly in lines 2 and 3. We will prove later that such a distinguished variables exists, then use Prop. 5.6 to prove that their Jacobian is non-zero.

8.4.4 *The Strict Lattice Product.* We now compute the Mobius function of the lattice of events $(\mathbf{u}, \mathbf{v}) \in L^A \times R^B$. By Lemma 8.23 the semilattice of this lattice is the product $L^A \times R^B$; the lattice turns out to be a *strict lattice product*.

Given two lattices $\hat{L}_1, \hat{L}_2$, their product $\hat{L}_1 \times \hat{L}_2$ is also a lattice, and , $\mu_{\hat{L}_1 \times \hat{L}_2}((u,v),(x,y)) = \mu_{\hat{L}_1}(u,x) \cdot \mu_{\hat{L}_2}(v,y)$. Define the *strict product* of $\hat{L}_1$ and $\hat{L}_2$ as:

$$\widehat{L_1 \times L_2} = (L_1 \times L_2) \cup \{\hat{1}\}$$

Thus, the strict product does not include elements of the form $(u, \hat{1})$ or $(\hat{1}, v)$; instead, it contains $\hat{1}$, which can be thought of as $(\hat{1}, \hat{1})$. This is indeed a lattice, because L_i is a meet semilattice, for $i = 1, 2$, hence $L_1 \times L_2$ is a meet semilattice, and after completing with $\hat{1}$ it becomes a lattice. We prove that its Mobius function is given by:

LEMMA 8.27. $\mu_{\widehat{L_1 \times L_2}}((u,v),\hat{1}) = -\mu_{\hat{L}_1}(u,\hat{1}) \cdot \mu_{\hat{L}_2}(v,\hat{1})$.

PROOF.

$$\begin{aligned} \mu_{\widehat{L_1 \times L_2}}((u,v),\hat{1}) &= - \sum_{(u,v) \le (x,y) < \hat{1}} \mu_{\widehat{L_1 \times L_2}}((u,v),(x,y)) \\ &= - \sum_{u \le x < \hat{1}, v \le y < \hat{1}} \mu_{\hat{L}_1}(u,x) \times \mu_{\hat{L}_2}(v,y) = -\mu_{\hat{L}_1}(u,\hat{1}) \cdot \mu_{\hat{L}_2}(v,\hat{1}) \end{aligned}$$

Here, we used the fact that the interval $[(u,v),(x,y)]$ of the strict product $\widehat{L_1 \times L_2}$

is isomorphic to the same interval in the regular product $\hat{L}_1 \times \hat{L}_2$, because $x < \hat{1}$ and $y < \hat{1}$. □

Given a lattice $\hat{L}$, we denote $\widehat{L^n}$ the strict cartesian product:

$$\widehat{L \times \ldots \times L}$$

It follows from the lemma that $\mu_{\widehat{L^n}}(\bar{u}, \hat{1}) = -(-1)^n \prod_i \mu_{\hat{L}}(u_i, \hat{1})$.

8.4.5 *The Expansion Formula.* Now we apply Mobius' inversion formula to Corollary 8.25. Denote:

$$LL = \{u \mid u \in L, \mu_{\hat{L}}(u, \hat{1}) \neq 0\}$$
$$RR = \{v \mid v \in R, \mu_{\hat{R}}(v, \hat{1}) \neq 0\}$$
$$s_1 = |LL| \quad s_2 = |RR| \quad s = s_1 \cdot s_2$$

Now we can finally define the number of slices s, s_1, and s_2: they are the number of lattice elements with non-zero Mobius values in $\widehat{L \times R}$, L, and R respectively.

Recall that the PP2CNF is $\Phi = \bigwedge_{(i,j)\in E}(X_i \vee Y_j)$, $E \subseteq [n_1] \times [n_2]$ and that $m = |E|$ and $n = n_1 + n_2$.

Definition 8.28. *Let $\mathbf{u} \in LL^A$ and $\mathbf{v} \in RR^B$. For every $u \in LL$ and $v \in RR$ denote the following sets:*

$$K_{uv}(\mathbf{u}, \mathbf{v}) = \{(i,j) \mid (i,j) \in E, \mathbf{u}(a_i) = u, \mathbf{v}(b_j) = v\}$$
$$K_{u\cdot}(\mathbf{u}) = \{i \mid i \in [n_1], \mathbf{u}(a_i) = u\}$$
$$K_{\cdot v}(\mathbf{v}) = \{j \mid j \in [n_2], \mathbf{v}(b_j) = v\}$$

Given an $(s + s_1 + s_2)$-tuple $\mathbf{k} = (k_{uv}, k_{u\cdot}, k_{\cdot v})_{u\in LL, v\in RR} \in (\{0, \ldots, m\})^{s+s_1+s_2}$, denote:

$$\#\mathbf{k} = |\{(\mathbf{u}, \mathbf{v}) | \mathbf{u} \in (LL)^A, \mathbf{v} \in (RR)^B, \forall u \in LL, v \in RR : |K_{uv}(\mathbf{u}, \mathbf{v})| = k_{uv}, |K_{u\cdot}(\mathbf{u})| = k_{u\cdot}, |K_{\cdot v}(\mathbf{v})| = k_{\cdot v}\}|$$

This definition generalizes Eq. 21. There, we started from a valuation $\theta : \{X_i \mid i \in [n_1]\} \cup \{Y_j \mid j \in [n_2]\} \to \{\texttt{true}, \texttt{false}\}$ and defined $K_{uv}(\theta)$ for $u, v \in [2]$. Now, instead of the valuation θ we have the two functions $\mathbf{u}, \mathbf{v}$.

The expansion formula for $P(\neg d^L)$ is given by:

Proposition 8.29. *The probability of $\neg d^L$ on the database D^L is given by:*

$$P(\neg d^L) = (-1)^n \sum_{\mathbf{k}} \prod_{u\in LL, v\in RR} y_{uv}^{k_{uv}} \cdot \prod_{u\in LL} (\mu_{\hat{L}}(u, \hat{1}) \cdot y_{u\cdot})^{k_{u\cdot}} \cdot \prod_{v\in RR} (\mu_{\hat{R}}(v, \hat{1}) \cdot y_{\cdot v})^{k_{\cdot v}}$$

Proof. By direct application of the Mobius inversion formula to Corollary 8.25:

$$\begin{aligned} P(\neg d^L) =& P(\bigvee_{\mathbf{u},\mathbf{v}} (E_{\mathbf{uv}} \wedge \neg d^L)) = -\sum_{\mathbf{u},\mathbf{v}} (-1) \cdot \mu(\mathbf{u}, \hat{1}) \cdot \mu(\mathbf{v}, \hat{1}) \cdot P(E_{\mathbf{uv}} \wedge \neg d^L) \\ =& (-1)^n \sum_{\mathbf{u},\mathbf{v}} (\prod_{a_i} \mu(\mathbf{u}(a_i), \hat{1}) \cdot y_{\mathbf{u}(a_i)\cdot}) \cdot (\prod_{b_j} \mu(\mathbf{v}(b_j), \hat{1}) \cdot y_{\cdot\mathbf{v}(b_j)}) \cdot (\prod_{a_i, b_j} y_{\mathbf{u}(a_i)\mathbf{v}(b_j)}) \\ =& (-1)^n \sum_{\mathbf{k}} \prod_{u\in LL, v\in RR} y_{uv}^{k_{uv}} \cdot \prod_{u\in LL} (\mu_{\hat{L}}(u, \hat{1}) \cdot y_{u\cdot})^{k_{u\cdot}} \cdot \prod_{v\in RR} (\mu_{\hat{R}}(v, \hat{1}) \cdot y_{\cdot v})^{k_{\cdot v}} \end{aligned}$$

In the second line, all terms where $\mu(\mathbf{u}(a_i), \hat{1}) = 0$ for some element a_i disappear, hence it suffices to consider only those functions $\mathbf{u} \in L^A$ whose range is LL, hence $\mathbf{u} \in LL^A$; similarly for $\mathbf{v} \in RR^B$. The last line simply groups these functions by the number of elements a_i and b_j that return a fixed value $u \in LL$ or $v \in RR$. □

We can now explain the role of the dangling blocks $D_k(a, \cdot)$, $D_k(\cdot, b)$: their role is to absorb the powers of the Mobius function above. Without the dangling blocks we would have had terms of the form $(\mu_{\hat{L}}(u, \hat{1}))^{k_u}$. The dangling blocks introduce the multilinear polynomials $y_{u\cdot}$, which are multiplied by these Mobius functions. Define the new multilinear polynomials:

$$Y_{u\cdot} = \mu_{\hat{L}}(u, \hat{1}) \cdot y_{u\cdot}, \forall u \in L \qquad Y_{\cdot v} = \mu_{\hat{R}}(v, \hat{1}) \cdot y_{\cdot v}, \forall v \in R$$

Then Prop. 8.29 becomes:

$$P(\neg d^L) = (-1)^n \sum_{\mathbf{k}} \prod_{u \in LL, v \in RR} y_{uv}^{k_{uv}} \cdot \prod_{u \in LL} Y_{u\cdot}^{k_{u\cdot}} \cdot \prod_{v \in RR} Y_{\cdot v}^{k_{\cdot v}} \tag{42}$$

This is now similar to Eq. 39 which we used to prove hardness for queries of type 1-1. We must prove that, collectively, the $s + s_1 + s_2$ functions y_{uv}, $Y_{u\cdot}$, $Y_{\cdot v}$ have a non-zero Jacobian. The three groups depend on different variables: there are distinguished variables z_k for the regular blocks $D_k(a_i, b_j)$, for $k = 1, s$, which occur only in y_{uv}; there are different distinguished variables z'_k for the dangling blocks $D_k(a_i, \cdot)$, for $k = 1, s_1$, which occur only in $Y_{u\cdot}$; and there are different distinguished variables z''_k for $Y_{\cdot v}$. Furthermore, the mapping $\bar{z}' \mapsto (Y_{u\cdot})_{u \in LL}$ has a non-zero Jacobian iff the mapping $\bar{z}' \mapsto (y_{u\cdot})_{u \in LL}$ has a non-zero Jacobian, because all numbers $\mu_{\hat{L}}(u, \hat{1})$ are $\neq 0$, and similarly for $Y_{\cdot v}$. Thus, it suffices to prove that each of the following three functions has a non-zero Jacobian:

$$\bar{z} \mapsto (y_{uv})_{u \in LL, v \in RR} \qquad \bar{z}' \mapsto (y_{u\cdot})_{u \in LL} \qquad \bar{z}'' \mapsto (y_{\cdot v})_{v \in RR}$$

For that, we will prove that each of the three sets of multivariate polynomials in Eq. 41 has a distinguished variable. We claim that this completes the proof of hardness of d^L. By repeatedly calling the Oracle for $P(\neg d^L)$, we can form a system of equations from Eq. 42, whose matrix is non-singular, and, thus, solve for all coefficients $\#\mathbf{k}$. We show how these can be used to compute $\#\Phi$. Fix any element $u_1 \in LL$ and associate it with `false` and associate all other elements $u_2 \in LL$ with `true`; similarly, $v_1 \in RR$ denotes `false`, all other elements denote `true`. Then:

$$\#\Phi = \sum_{\mathbf{k}: k_{u_1 v_1} = 0} \#\mathbf{k}$$

8.4.6 *The Dual Lineage Expressions for Queries of Type 2-2.* Our goal is to prove that each of the three sets of multivariate polynomials in Eq. 41 has a distinguished variable. As for type 1-1 queries, we examine the Boolean functions representing the dual lineage expressions. The blocks $D_k(a_i, b_j)$ $D_k(a_i, \cdot)$, $D_k(\cdot, b_j)$ are isomorphic; consider any of them, associate a Boolean variable $Z_1, Z_2, \ldots, Z_m$ to each leveled symbol $P_1, P_2, \ldots, P_m$, and denote $Y_u^{0*}, Y^{0*}, Y^{**}, Y^{*0}, Y_v^{*0}$, for $u \in LL, v \in RR$ the dual lineage expressions of the queries $d_u^{0*}, d^{0*}, d^{**}, d^{*0}, d_v^{*0}$ on this block. Then the multivariate polynomials in Eq. 41 are the arithmetization of the

following Boolean expressions:

$$\begin{aligned} Y_{uv} &= Y_u^{0*} \wedge Y^{**} \wedge Y_v^{*0} && u \in LL, v \in RR \\ Y_{u\cdot} &= Y_u^{0*} \wedge Y^{**} \wedge Y^{*0} && u \in LL \\ Y_{\cdot v} &= Y^{0*} \wedge Y^{**} \wedge Y^{*0} && v \in RR \end{aligned} \tag{43}$$

For example, $f_{uv} = P(Y_{uv})$, etc. We prove that, in each of the three lines, the set of Boolean functions in that line has a distinguished variable Z_{q_0}. In fact, we will prove something stronger: that all Boolean functions $Y_{uv}, Y_{u\cdot}, Y_{\cdot v}$ are irreducible. We show here why this implies that they have distinguished variables, then will prove the claim in the remainder of this section.

Given $d = \bigvee_i c_i$, denote $Sym(c_i)$ the set of relational symbols occurring in c_i. Recall from Sect. 8.3 that the dual lineage Y of d on block has one clause for each component c_i, consisting of one Boolean variable for each symbol in $Sym(c_i)$. We will blur the distinction between $Sym(c_i)$ and the clause corresponding to c_i.

Let c_i, c_j be two distinct components. If $Sym(c_j) \subseteq Sym(c_i)$ then the clause corresponding to c_i is redundant. For example if $d = S_1(x, y_1), S_2(x, y_2), S_3(x, y_3) \vee S_1(x_1, y), S_2(x_2, y)$, then its dual lineage has two clauses $Y = (Z_1 \vee Z_2 \vee Z_3) \wedge (Z_1 \vee Z_2)$, but the first clause is redundant, and $Y \equiv Z_1 \vee Z_2$. Thus, some non-redundant components may have a redundant clauses. However, we show that, if a component c_i is leveled injectively, then its corresponding clause is non-redundant.

LEMMA 8.30. *Let Let c_i be a component, and $c_i^{\rho_X \rho_Y}$ a leveling such that both $\rho_X : Var_X(c_i) \to \{0\} \cup [n_X]$ and $\rho_Y : Var_Y(c_i) \to \{0\} \cup [n_Y]$ are injective functions. Let c_j^{ρ} be any other leveled component. If $Sym(c_j^{\rho}) \subseteq Sym(c_i^{\rho_X \rho_Y})$ then the implication $c_i \Rightarrow c_j$ holds.*

In particular, if d has no redundant components, then all clauses corresponding to injective levelings $c_i^{\rho_X \rho_Y}$ are non-redundant.

PROOF. Let $f : c_j^{\rho'} \to c_i^{\rho_X \rho_Y}$ be the function that maps each atom in $c_j^{\rho'}$ to the atom with the same relation symbol in $c_i^{\rho_X \rho_Y}$. We prove that this is a homomorphism $c_j \to c_i$, contradicting the fact that d was minimized. We claim that if two atoms have the same variable in $c_j^{\rho'}$, then the corresponding atoms in $c_i^{\rho_X \rho_Y}$ also have the same variable: this immediately proves that f is a homomorphism $c_j \to c_i$. Assume w.l.o.g. that $c_i^{\rho_X \rho_Y}$ has root variable $\breve{x}$: then our claim holds vacuously on the x_i-variables in c_j. Suppose two atoms in $c_j^{\rho'}$ have the same y-variable: then their levelings must have the same level for the second attribute, namely $\tau = \rho'(y)$, i.e. the atoms are $S_1^{\tau_1 \tau}(x_1, y)$ and $S_2^{\tau_2 \tau}(x_2, y)$. By assumption, $c_i^{\rho_X \rho_Y}$ also contains two atoms $S_1^{\tau_1 \tau}(\ldots)$ and $S_2^{\tau_2 \tau}(\ldots)$ and, since we have chosen ρ_Y to be injective, it follows that they must have the same x-variable. □

The condition that the leveling is injective is necessary. To see this, consider $S_1(x, y_1), S_2(x, y_2), S_3(x, y_3) \vee S_1(x_1, y), S_2(x_2, y)$: if the first component is leveled as $S_1^{11}(\breve{x}, \breve{y}_1), S_2^{11}(\breve{x}, \breve{y}_2), S_3^{11}(\breve{x}, \breve{y}_3)$ then it's clause is made redundant by similar leveling of the second component. This explains why we use n_X and n_Y extra levels for queries of type 2-2.

PROPOSITION 8.31. *Suppose that all Boolean functions Y_{uv}, $u \in LL, v \in RR$ defined in Eq. 43 are irreducible (as per Def. 8.12). Then they have a distinguished Boolean variable. Similarly, if all Boolean functions $Y_{u\cdot}$, $u \in LL$ are irreducible then they have a distinguished variable; and similarly for the set of Boolean function $Y_{\cdot v}$, $v \in RR$.*

PROOF. We prove the statement for the set of functions Y_{uv} only; the other two are similar. It suffices to prove two facts: (1) there exists a Boolean variable Z_{q_0} s.t. all functions Y_{uv} depend on it, and (2) all functions are inequivalent. Choose Z_{q_0} to be the a Boolean representing a symbol S^{11}. (The choice of 11 is arbitrary, any leveling $\tau_1\tau_2$ with $\tau_1 \neq 0$ and $\tau_2 \neq 0$ works.) We will prove that in each CNF expression Y_{uv} there exists a clause containing Z_{q_0}. Indeed, let c_i be any component that contains S, and let $c_i^{\rho_X\rho_Y}$ be any injective leveling of c_i s.t. (a) $\forall x \in Var_X(c_i)$, $\rho_X(x) > 0$ and $\forall y \in Var_Y(c_i)$, $\rho_Y(y) > 0$. (b) at least one S-atom is leveled into S^{11}. Such a leveling always exists because we have $n_X \geq |Var_X(c_i)|$ and $n_Y \geq |Var_Y(c_i)|$. By the previous lemma, the clause $Sym(c_i^{\rho_X\rho_Y})$ is not redundant in Y^{**}. It remains to prove the clause is not redundant in $Y^{0*} \wedge Y^{**} \wedge Y^{*0}$. This follows from the fact that all clauses in Y^{0*} have symbols with levelings 0τ, hence none of these symbols belongs to $Sym(c_i^{\rho_X\rho_Y})$, and similarly for Y^{*0}. Now we prove (2): the Boolean expressions Y_{uv} are inequivalent. Suppose otherwise, $Y_{u_1v_1} \equiv Y_{u_2v_2}$. Set to `true` all variables Z_i corresponding to symbols $S^{\tau_1\tau_2}$ where $\tau_1 \geq 1$, and denote Y', Y'' the resulting Boolean expressions. Equivalently, Y', Y'' are the dual lineages of the query obtained from $d_{u_1v_1}$ and $d_{u_2v_2}$ by setting all symbols $S^{\tau_1\tau_2}$ where $\tau_1 \geq 1$ to `false`: these queries are precisely $d_{u_1\cdot}$ and $d_{u_2\cdot}$, hence $Y' \equiv Y_{u_1\cdot}$ and $Y'' \equiv Y_{u_2\cdot}$. If $Y' \equiv Y''$ then we have $d_{u_1\cdot} \equiv d_{u_2\cdot}$, which implies $u_1 = u_2$ (see the discussion immediately after Def. 4.8). Similarly, we prove that $v_1 = v_2$. Therefore, all irreducible factors containing Z_{q_0} are inequivalent, implying that the real variable z_{q_0} is a distinguished variable for the multilinear polynomials f_{uv}. The proofs for $Y_{u\cdot}$ and for $Y_{\cdot v}$ are similar and omitted. □

It remains to prove that the Boolean functions Y_{uv}, $Y_{u\cdot}$, $Y_{\cdot v}$ are irreducible. For that we first need to establish some properties of queries of type 2-2.

8.4.7 *The Properties of Forbidden Queries of Type 2-2.* We prove here two properties of queries of type 2-2. This is the only place were we need the full power of the shatter rewrite rule in Def. 4.13: so far we only needed shattering of an entire level, $d[A/Z]$, but in this section we need general shattering.

Throughout this section we fix a strict left-to-right path in d: $c_0, c_1, \ldots, c_k$ (defined in Sect. 8.1). We have $k \geq 1$, because c_0 is a left component and c_k is a right component, and they must be distinct. If $k = 1$ then we say that d is a *short* query. If $k > 1$ we say that d is a *long* query; in this case, each of $c_1, \ldots, c_{k-1}$ has root variables both x and y. The two properties are:

PROPOSITION 8.32. *Let c be any component. Then there exists $i > 0$ such that $Sym(c) \cap Sym(c_i) \neq \emptyset$, and there exists $i < k$ such that $Sym(c) \cap Sym(c_i) \neq \emptyset$,*

In particular, if c is any left component, then $c, c_1, \ldots, c_k$ is also a strict left-right path: indeed c has no common symbols with $c_2, \ldots, c_k$ (otherwise we obtain a shorter path), hence the proposition implies that c, c_1 must have some common

symbol. Similarly, if c is a right component, then $c_0, \ldots, c_{k-1}, c$ is also a left-right path.

PROPOSITION 8.33. *The following hold:*

$$\forall s \in sc_X(c_0), \forall s' \in sc_Y(c_1): \quad Sym(s) \not\subseteq Sym(s') \tag{44}$$

$$\forall s \in sc_Y(c_{k-1}), \forall s' \in sc_X(c_k): \quad Sym(s) \not\subseteq Sym(s') \tag{45}$$

We illustrate with an example.

EXAMPLE 8.34. *Consider the query $q_{diamond}$ shown at the end of Sect. 8.1. There are two connected components, c_0, c_1, which also form a strict left-to-right path. The first component c_0 has two X-subcomponents, $S_1(x,y_1), S_2(x,y_1), S_3(x,y_1)$ and $S_1(x,y_2), S_2(x,y_2), S_4(x,y_2)$ and their sets of symbols are $\{S_1,S_2,S_3\}$ and $\{S_1,S_2,S_4\}$; the second component c_1 has two Y-subcomponents $S_3(x_1,y), S_4(x_1,y), S_1(x_1,y)$ and $S_3(x_2,y), S_4(x_2,y), S_2(x_2,y)$ and their sets of symbols are $\{S_1,S_3,S_4\}$ and $\{S_2,S_3,S_4\}$. None of the former two sets is included in the latter, and vice versa.*

On the other had, consider our running example $q_\diamond$. This is not *a forbidden query, as we show shortly, we only used it as a simpler surrogate for $q_{diamond}$. It also has two components: the first has the X-subcomponents $S_1(x,y_1)$ and $S_2(x,y_2)$ with symbol-sets $\{S_1\},\{S_2\}$, and the second has two subcomponents with two symbol sets $\{S_1\},\{S_2\}$. They are pairwise equal, thus violating the proposition.*

We prove the propositions through a sequence of lemmas.

LEMMA 8.35. *Let c be any component of a forbidden query. (1) For any two distinct X-subcomponents, $s, s' \in sc_X(c)$, we have $Sym(s) \not\subseteq Sym(s')$. (2) Let c' be any other component distinct from c, and let $s' \in sc_X(c')$; then $Sym(c) \not\subseteq Sym(s')$.*

PROOF. Suppose otherwise, $Sym(s) \subseteq Sym(s')$. Then there exists a homomorphism $s' \to s$ (since both s and s' have only two variables, x, y and x, y' respectively); this contradicts the fact that c is minimized. Similarly, if $Sym(c) \subseteq Sym(s')$, then we can define a homomorphism $c \to s'$ and, thus, $c \to c'$, which implies $c' \Rightarrow c$, contradicting the fact that d has no redundant components. □

For any symbol S, denote $sc_X(c,S) = \{s \mid S \in Sym(s), s \in sc_X(c)\}$ the set of subcomponents of c that contain the symbol S; similarly for $sc_Y(c,S)$.

LEMMA 8.36. *If $S_1 \in Sym(c_0) \cap Sym(c_1)$ and $S \in Sym(c_0) \cup Sym(c_1)$ then one of the following relations holds:*

$$sc_X(c_0,S_1) \subseteq sc_X(c_0,S) \qquad or \qquad sc_Y(c_1,S_1) \subseteq sc_Y(c_1,S)$$

Similarly, for $S_1 \in Sym(c_{k-1}) \cap Sym(c_k)$ and $S \in Sym(c_{k-1}) \cup Sym(c_k)$ we have $sc_X(c_{k-1},S_1) \subseteq sc_X(c_{k-1},S)$ or $sc_Y(c_k,S_1) \subseteq sc_Y(c_k,S)$.

PROOF. Suppose otherwise and assume w.l.o.g. $S \neq S_1$. Then there exists two subcomponents $s_0 \in sc_X(c_0,S_1) - sc_X(c_0,S)$ and $s_1 \in sc_Y(c_1,S_1) - sc_Y(c_1,S)$. Let A, B be two sets of constants such that $|A| = |sc_X(c_0,S)|$ and $|B| = |sc_Y(c_1,S)|$, and define the following shattered vocabulary $\mathbf{R}_{AB}$:

(1) S shatters into S_{a*} and S_{*b} for all $a \in A, b \in B$; it does not shatter into S_{**}.
(2) Every symbol $S' \neq S$ shatters into S'_{**}, and:

(a) if $S' \in sc_X(c_0, S)$ then it also shatters into S'_{a*} for all $a \in A$, and
(b) if $S' \in sc_Y(c_1, S)$ then it also shatters into S'_{*b} for all $b \in B$.

Denote d_{AB} the shattered query w.r.t. this vocabulary, thus $d \to d_{AB}$ by Def. 4.13. Note that $seq(d_{AB}) < seq(d)$, because we reduce by 1 the number of relations of arity 2. We prove that d_{AB} is unsafe, contradicting the fact that d is forbidden (recall Def. 7.2).

Denote $c_{0,A}$ a shattering of c_0, with the following properties: all variables y_i that occur in some S-atom are shattered into distinct constants $a_i \in A$, i.e. if the S-atoms are $S(x, y_1), S(x, y_2), \ldots$ then they are shattered to $S_{*a_1}(x), S_{*a_2}(x), \ldots$; all variables y_i that do not occur in S are shattered to $*$; this means that, if a variable y_i does not occur in S, then any atom $S'(x, y_i)$ shatters to $S'_{**}(x, y_i)$. It follows that (a) $c_{0,A}$ is not redundant in d_{AB} (because we used distinct constants), (b) $c_{0,A}$ has only x as root variable (since either c_0 contains S, and in that case $c_{0,A}$ has at least one unary symbol $S_{*a}(x)$, or c_0 does not contain S and in that case all its variables y_i are shattered to $**$, hence $c_{0,A}$ has at least two variables y_i), and (c) it contains the symbol $S_{1,**}$ (because the subcomponent s_0 contains S_1 but does not contain S, and therefore it is shattered into $**$).

Next we consider two cases, depending on whether d is short or long. If it is short, then c_1 is a right component, and we consider the shattering $c_{1,B}$ obtained similarly to $c_{0,A}$: thus, $c_{1,B}$ is non-redundant, does not have root variable x, and contains the symbol $S_{1,**}$. Therefore, d_{AB} is unsafe, because $c_{0,A}$ and $c_{1,B}$ are symbol-connected, the only root level in the first component is X, and the only root level in the second component is Y.

If d is long, then it has both x and y as root variables, and it has a single Y-component, $sc_Y(c_1) = \{c_1\}$. If $S \in Sym(c_1)$, then $sc_Y(c_1, S) = sc_Y(c_1)$, contradicting our assumption that $sc_Y(c_1, S_1) \not\subseteq sc_Y(c_1, S)$. Therefore $S \notin Sym(c_1)$; we have $S \in Sym(c_0)$. In fact, S does not occur in any component $c_2, c_3, \ldots, c_k$, otherwise we could find a shorter left-right path. Consider the shattering of $c_1, \ldots, c_k$ where all variables (x_i or y) are shattered to $*$, denote them $c_{1,*}, \ldots, c_{k,*}$. They are non-redundant, symbol-connected, and the only root level in $c_{k,*}$ is Y. Moreover, $c_{0,A}$ shares the symbol $S_{1,**}$ with $c_{1,*}$ and therefore the set $c_{0,A}, c_{1,*}, \ldots, c_{k,*}$ does not have a separator. □

In the proof of Lemma 8.36 we used arbitrary shattering; if we restrict the shattering rule in Sect. 7 to shatter only an entire level $d \to d[A/Z]$, then the lemma fails. We illustrate with an example.

EXAMPLE 8.37. *The query $d_\diamond$ does not satisfy Lemma 8.36, because $sc_X(c_0, S_1) \not\subseteq sc_X(c_0, S_2)$ (the former is $\{S_1(x, y_1)\}$, while the latter is $\{S_2(x, y_2)\}$), and similarly $sc_Y(c_1, S_1) \not\subseteq sc_X(c_1, S_2)$. We show that $d_\diamond$ is not forbidden, by applying the shattering described in the lemma: we shatter S_2 to $S_{2,a*}$ and $S_{2,*b}$, and shatter S_1 only to $S_{1,**}$. The shattered query is: $S_{1,**}(x, y_1), S_{2,*b}(x) \vee S_{1,**}(x_1, y), S_{2,a*}(y)$, which is isomorphic to $h_1 = R(x), S(x, y) \vee S(x, y), T(y)$ up to symbol renaming. Thus, $h_1 \leq^{FO}_{lin} d_\diamond$ by Prop. 2.9.*

LEMMA 8.38. *$\forall s \in sc_X(c_0)$, $Sym(s) \cap Sym(c_1) \neq \emptyset$ and $\forall s \in sc_Y(c_k)$, $Sym(c_{k-1}) \cap Sym(s) \neq \emptyset$.*

PROOF. Suppose $Sym(s) \cap Sym(c_1) = \emptyset$ and let S_1 be any symbol in $Sym(c_0) \cap Sym(c_1)$ (it exists by the definition of a left-right path $c_0, c_1, \ldots, c_k$) and let $s_1 \in sc_X(c_0, S_1)$ be any subcomponent that contains S_1. Then $Sym(s) \subseteq Sym(s_1)$ because, for any $S' \in Sym(s)$, we have $sc_X(c_0, S_1) \subseteq sc_X(c_0, S')$ by Lemma 8.36 (since S' does not occur in c_1), which implies $s_1 \in sc_X(c_0, S')$, in other words, s_1 contains S'. Thus $Sym(s) \subseteq Sym(s_1)$, contradicting Lemma 8.35. □

Denote $U_X(c) = \bigcap_{s \in sc_X(c)} Sym(s)$: these are the symbols that occur in *all* subcomponents of s. We call a symbol $S \in U_X(c)$ *X-ubiquitous* in c. Note that if c has root variable y, then every symbol is X-ubiquitous, because $sc_X(c) = \{c\}$. We define similarly $U_Y(c)$.

LEMMA 8.39. *Every symbol occurring in c_0 or c_1 is either X-ubiquitous in c_0 or is Y-ubiquitous in c_1. In other words, $Sym(c_0) \cup Sym(c_1) \subseteq U_X(c_0) \cup U_Y(c_1)$. Similarly, $Sym(c_{k-1}) \cup Sym(c_k) \subseteq U_X(c_{k-1}) \cup U_Y(c_k)$.*

PROOF. We consider three cases.

Case 1. $S \in Sym(c_0) - Sym(c_1)$, and S is not X-ubiquitous in c_0. Let $s \in sc_X(c_0) - sc_X(c_0, S)$ be any subcomponent that does not contain S. We claim that s has no common symbols with c_1. Indeed, if $S_1 \in Sym(s) \cap Sym(c_1)$ then by Lemma 8.36 we must have either $sc_X(c_0, S_1) \subseteq sc_X(c_0, S)$ (which is not possible because $s \in sc_X(c_0, S_1)$ and $s \notin sc_X(c_0, S)$) or $sc_Y(c_1, S_1) \subseteq sc_Y(c_1, S)$ (which is not possible because $sc_Y(c_1, S) = \emptyset$). Thus, the subcomponent s has no symbols in common with c_1, which is a contradiction by Lemma 8.38.

Case 2. $S \in Sym(c_1) - Sym(c_0)$ and S is not Y-ubiquitous in c_1. Then $k = 1$: otherwise, if $k > 1$ then c_1 has two root variables, x and y, and every symbol $S \in Sym(c_1)$ is ubiquitous in c_1. This case is symmetric to case 1.

Case 3. $S_1 \in Sym(c_0) \cap Sym(c_1)$, and suppose that it is not X-ubiquitous in c_0 and not Y-ubiquitous in c_1; in particular, $k = 1$. Let $s \in sc_X(c_0)$ and $s' \in sc_Y(c_1)$ be two subcomponents that do not contain S_1. By Lemma 8.38 the subcomponent s contains some common symbol S with c_1, i.e. $S \in Sym(s) \cap Sym(c_1)$. By Lemma 8.36:

$$sc_Y(c_1, S) \subseteq sc_Y(c_1, S_1) \tag{46}$$

because $sc_X(c_0, S) \not\subseteq sc_X(c_0, S_1)$. Choose similarly $S' \in Sym(c_0) \cap Sym(s')$, hence:

$$sc_X(c_0, S') \subseteq sc_X(c_0, S_1) \tag{47}$$

Now we apply Lemma 8.36 to S' and S and derive a contradiction. More precisely, we show that neither $sc_X(c_0, S') \subseteq sc_X(c_0, S)$ nor $sc_Y(c_1, S') \subseteq sc_Y(c_1, S)$ holds. For the second non-containment we use the fact that $s' \in sc_Y(c_0, S')$; by construction we have $s' \notin sc_Y(c_0, S_1)$, which implies $s' \notin sc_Y(c_0, S)$ by Eq. 46, proving the second non-containment. For the first non-containment, we prove that we can choose $S \in Sym(s)$ such that there exists $s_0 \in sc_X(c_0, S')$ s.t. $s_0 \notin sc_X(c_0, S)$. Indeed, choose any $s_0 \in sc_X(c_0, S') \subseteq sc_X(c_0, S_1)$. By Lemma 8.35 $Sym(s) \not\subseteq Sym(s_0)$. By Case 1, if a symbol $S \in Sym(s)$ does not occur in $Sym(c_1)$, then S is ubiquitous, hence $S \in Sym(s_0)$. This implies that there exists $S \in Sym(s)$ such that (a) S occurs in c_1, and (b) does not occur in s_0. This is the symbol we choose, and prove the claim. □

When $k > 1$ then the lemma says that all symbols that occur in c_0 but not in c_1 are X-ubiquitous in c_0; similarly, all symbols that occur in c_k but not in c_{k-1} are Y-ubiquitous in c_k.

EXAMPLE 8.40. *We illustrate with the query $d_{diamond}$ (Eq. 37). In the first component, $c_0 = S_1(x, y_1), S_2(x, y_1), S_3(x, y_1), S_1(x, y_2), S_2(x, y_2), S_4(x, y_2)$, both S_1, S_2 are ubiquitous; in $c_1 = S_3(x_1, y), S_4(x_1, y), S_1(x_1, y), S_3(x_2, y), S_4(x_2, y), S_2(x_2, y)$, the symbols S_3, S_4 are ubiquitous. Every symbol occurring in c_0 or c_1 is ubiquitous in either the first of the second component. On the other hand, the query $q_\diamond$ does not satisfy the lemma (because it is not a forbidden query).*

To see a more complex query of type 2-2 consider $d = c_0 \vee c_1 \vee c_2$ where:

$$c_0 = S(x, y_1), S_2(x, y_1), S_3(x, y_1), S(x, y_2), S_1(x, y_2), S_3(x, y_2), S(x, y_3), S_1(x, y_3), S_2(x, y_3)$$
$$c_1 = S_1(x, y), S_2(x, y), S_3(x, y), S_4(x, y), S_5(x, y), S_6(x, y)$$
$$c_2 = S'(x_4, y), S_5(x_4, y), S_6(x_4, y), S'(x_5, y), S_4(x_5, y), S_6(x_5, y), S'(x_6, y), S_4(x_6, y), S_5(x_6, y)$$

In the left component c_0, S is ubiquitous, and all other symbols occur in c_1; in c_2, S' is ubiquitous. The reader may check that the query is indeed forbidden: for example, $d[S_1 = \texttt{true}]$ is safe, because $c_0[S_1 = \texttt{true}]$ minimizes to $S(x, y), S_2(x, y), S_3(x, y)$ and is no longer a left component. The left semi-lattice of leveled query d^L has 7 points, and so does the right semi-lattice. In general, one can show that any left lattice L and right lattice R can be realized, i.e. there is a forbidden query of type 2-2 having these as left and right lattice respectively.

We now prove Prop. 8.32. Given a component c, we show that it has some common symbols with $c_1, \ldots, c_k$. Suppose otherwise: then all its symbols are in c_0 (since the strict path contains all symbols), and it shares no symbols with c_1. By Lemma 8.39 every symbol $S \in Sym(c)$ is in $U_X(c_0)$ (because it does not occur in c_1). Let $s \in sc_X(c_0)$ be any subcomponent: by definition of the ubiquitous, $U_X(c_0) \subseteq Sym(s)$, hence $Sym(c) \subseteq Sym(s)$, contradicting Lemma 8.35. The proof that c has a common symbol with $c_0, \ldots, c_{k-1}$ is similar and omitted.

LEMMA 8.41. *Let $s_0 \in sc_X(c_0)$ and $s_1 \in sc_Y(c_1)$. Then $Sym(s_0) \not\subseteq Sym(s_1)$.*

PROOF. Suppose $Sym(s_0) \subseteq Sym(s_1)$. Then $Sym(c_0) \subseteq Sym(s_1)$ because every symbol in c_0 is either ubiquitous in c_0, hence it occurs in s_0, or is ubiquitous in c_1, which by definition means that it occurs in s_1. But this is a contradiction by Lemma 8.35. □

This proves Eq. 44 of the proposition for the case when $i = 1$; for $i > 1$ the proposition holds vacuously because c_0 and c_i have no common symbols.

8.4.8 *The Dual Lineage Expressions Are Irreducible*

PROPOSITION 8.42. *For all $u \in LL, v \in RR$, each of the Boolean expressions $Y_{uv}, Y_{u\cdot}, Y_{\cdot v}$ (Eq. 43) is irreducible.*

PROOF. (or Prop. 8.42) We prove the statement for Y_{uv} only; the proof for $Y_{u\cdot}$ and $Y_{\cdot v}$ is similar. Recall that $Y_{uv} = Y_u^{0*} \wedge Y^{**} \wedge Y_v^{*0}$, where the three dual lineages are for the queries d_u^{0*}, d^{**} and d_v^{*0} respectively. Fix a left-to-right path in d: $c_0, \ldots, c_k$. Define the *core* to be the following set of leveled components of d^L: it

consists of all leveled component $c^{\rho_X \rho_Y}$ where c is one of c_i, $i = 0, k$ and ρ_X, ρ_Y are injective, except for levelings $c_0^{0\rho_Y}$ and $c_k^{\rho_X 0}$. In other words, the core contains the following components: (a) $c_0^{\tau\rho_Y}$ where $\tau \neq 0$, (b) $c_i^{\tau_1,\tau_2}$ for $0 < i < k$ and for any τ_1, τ_2 (except 00, since that leveling does not exists), and (c) $c_k^{\rho_X \tau}$, with $\tau > 0$. We prove the following, for all $u \in LL, v \in RR$: (1) every core component occurs as a non-redundant component in d_{uv}; (2) every component in d_{uv} shares a common symbol with some core component; and (3) the core is symbol-connected. The three claims prove the proposition. Indeed, (1) and Lemma 8.30 implies that every core component corresponds to a non-redundant clause in Y_{uv}, (3) implies that all these clauses connected in the primal graph, and together with (2) it implies that the primal graph is connected. Therefore, by Corollary 8.14, Y_{uv} is irreducible.

We prove claim (1). Recall that $d_{uv} = d_u^{0*} \vee d^{**} \vee d_v^{*0}$. We first show that each core component appears in one of these three queries. Indeed, both $c_0^{\tau\rho_Y}$ and $c_k^{\rho_X \tau}$ are in $\mathcal{C}^{**}$, hence they appear in d^{**} because $\tau \neq 0$ and neither ρ_X nor ρ_Y can be identically 0, because they are injective. If $k = 1$ (d is short) then there are no other core components, so assume $k > 1$ (d is long) and consider a core component $c_i^{\tau_1\tau_2}$ where $0 < i < k$. If $\tau_1, \tau_2 \neq 0$ then it belongs to $\mathcal{C}^{**}$. If $\tau_1 = 0$, then it belongs to $\mathcal{C}^{0*}$. Since both X and Y are root levels in c_i, the X-split is itself, $\sigma_X(c_i^{0\tau_2}) = c_i^{0\tau_2}$, and therefore it appears in all disjunctive queries d_j^{0*} that define Q^{0*} (Eq. 40). Therefore, $c_i^{0\tau_2}$ appears in d_u^{0*} for every $u \in LL$. Similarly for $c_i^{\tau_1 0}$. Next, we prove that no core component is redundant in d_{uv}. Suppose otherwise, that $c_i^{\rho_X\rho_Y} \Rightarrow c^\rho$ for some component c^ρ of d_{uv}. By Lemma 8.30, c^ρ cannot be a component in d^L, thus it must be a subcomponent of d^L (which occur in d_u^{0*} and d_v^{*0}). Assume w.l.o.g. that c is in d_u^{0*}, which means that $c \in sc_X(c')$ where c' is a left component; it also means that its leveling is $\rho = 0\tau$, i.e. $c^\rho = c^{0\tau}$ and all its symbols are of the form $S^{0\tau}$. We have $Sym(c^\rho) \subseteq Sym(c_i^{\rho_X\rho_Y})$. Since a left component c' shares symbols only with c_0 and c_1, we must have $i = 0$ or $i = 1$. We prove that $i = 1$; otherwise, if $i = 0$ then by construction the core contains only levelings $c_0^{\tau_0\rho_Y}$ where $\tau_0 > 0$, which has no symbols of the form $S^{0\tau}$. Thus, $i = 1$. Since ρ_X is injective, only one Y-subcomponent $s \in sc_Y(c_1)$ can be leveled 0τ, implying that $Sym(c^{0\tau}) \subseteq Sym(s^{0\tau})$, and hence $Sym(c) \subseteq Sym(s)$. This is impossible, by Prop. 8.33.

We prove claim (2). Let c^ρ be any component in d_{uv}. Suppose it contains some symbol $S^{\tau_1\tau_2}$ where $\tau_1 > 0$ and $\tau_2 > 0$. The strict path $c_0, \ldots, c_k$ contains all symbols, so there exists c_i that contains S. Construct any injective leveling of c_i, $c_i^{\rho_X\rho_Y}$ that contains the symbol $S^{\tau_1\tau_2}$: since neither $\rho_X \equiv 0$ nor $\rho_Y \equiv 0$, this component is in the core, even if $i = 0$ or $i = k$. Suppose now that all symbols in c^ρ have leveling $S^{0\tau}$. Then by Prop. 8.32 c has a common symbol S with some c_i where $i > 0$. As before, we construct an injective leveling $c_i^{\rho_X\rho_Y}$ that contains $S^{0\tau}$, and, since $i > 0$, this leveling is in the core.

It remains to prove claim (3). For any $(\tau_1, \tau_2) \neq (0, 0)$ define the following sequence of core components:

$$P_{\tau_1\tau_2} = [c_0^{\tau_1\rho_Y},]c_1^{\tau_1\tau_2}, \ldots, c_i^{\tau_1\tau_2}, \ldots [, c_k^{\rho_X\tau_2}] \tag{48}$$

The sequence contains all components $c_i^{\tau_1\tau_2}$ for $0 < i < k$. If $\tau_1 \neq 0$ then we include the first component, by choosing some injective ρ_Y s.t. $c_0^{\tau_1\rho_Y}$ and $c_1^{\tau_1\tau_2}$

share a common symbol; similarly, if $\tau_2 \neq 0$ then we include the last component in the sequence. Thus, $P_{\tau_1\tau_2}$ is a subset of the core, and is symbol-connected. Moreover, every component in the core belongs to some $P_{\tau_1\tau_2}$. We will prove that all these sequences are connected. We start by showing that, for all $\tau \neq 0$, any two components $c_0^{\tau\rho_Y}$ and $c_0^{\tau\rho'_Y}$ are symbol connected. Notice that the range of ρ_Y is $\{0\} \cup [n_Y]$, thus contains at least one more level than the number of variables in $Var_Y(c_0)$. Therefore, there exists $t, t' \in \{0\} \cup [n_Y]$ s.t. $t \notin Im(\rho_Y)$ and $t' \notin Im(\rho'_Y)$. Case 1: $t = t'$, thus ρ_Y, ρ'_Y avoid a common level. Define ρ''_Y and ρ'''_Y as follows:

$$\rho''_Y(y_1) = t \qquad \rho''_Y(y_j) = \rho_Y(y_j), \forall j > 1$$
$$\rho'''_Y(y_1) = t \qquad \rho'''_Y(y_j) = \rho'_Y(y_j), \forall j > 1$$

Then both ρ''_Y and ρ'''_Y are injective, and any two consecutive components in the sequence below share a common symbol:

$$c_0^{\tau\rho_Y}, c_0^{\tau\rho''_Y}, c_0^{\tau\rho'''_Y}, c_0^{\tau\rho'_Y}$$

Case 2: $t \neq t'$. Define ρ''_Y as follows:

$$\rho''_Y(y_i) = \begin{cases} \rho_Y(y_i) & \text{if } \rho_Y(y_i) \neq t' \\ t & \text{if } \rho_Y(y_i) = t' \end{cases}$$

Then ρ''_Y is injective, and the two components $c_0^{\tau\rho_Y}$, $c_0^{\tau\rho''_Y}$ share at least one common symbol. Now we apply case 1 to $c_0^{\tau\rho''_Y}$ and $c_0^{\tau\rho'_Y}$. This proves that $c_0^{\tau\rho_Y}$ and $c_0^{\tau\rho'_Y}$ are symbol connected. Similarly, $c_k^{\rho_X\tau}$ and $c_k^{\rho'_X\tau}$ are symbol connected, for all $\tau \neq 0$. Finally, we show that any two paths $P_{\tau_1\tau_2}$ and $P_{\tau_3\tau_4}$ defined by Eq. 48 are symbol-connected. Suppose first that $\tau_1 \neq 0$ and $\tau_4 \neq 0$. Then $P_{\tau_1\tau_2}$ is connected to $P_{\tau_1\tau_4}$ at the left end, and $P_{\tau_1\tau_4}$ is connected to $P_{\tau_3\tau_4}$ at the right end. Similarly for the case when $\tau_3 \neq 0$ and $\tau_2 \neq 0$. It remains to consider the case $\tau_1 = \tau_3 = 0$ (and similarly $\tau_2 = \tau_4 = 0$). In this case we have $\tau_2 \neq 0$ and $\tau_4 \neq 0$, therefore $P_{0\tau_2}$ is connected to $P_{1\tau_2}$, which is connected to $P_{1\tau_4}$, which is connected to $P_{0\tau_4}$. □

8.4.9 *Discussion.* We explain here that the use of the general shattering rule $d \to d_A$ in Sect. 7 is essential in order to prove hardness of queries of type 2-2; in other words, the more restricted shattering $d \to d[A/Z]$ was sufficient for Sect. 7, but is insufficient here. Recall that the general shattering allowed us to rewrite $d_\diamond$ to h_1. It turns out that $d_\diamond$ still has a distinguished variable, although some of its 9 Boolean functions Y_{uv} are disconnected; in this regards, it behaves much like a query of type 1-1. However, the following query does not have any distinguished variables.

Example 8.43. *Consider the following query:*

$$d = S_1(x, y_1), S_2(x, y_2), S_3(x, y_3) \vee$$
$$S_2(x_1, y), S_3(x_1, y), S_1(x_2, y), S_3(x_2, y), S_1(x_3, y), S_2(x_3, y)$$

The query is not forbidden, because it does not satisfy Prop. 8.33. (Another way to see this is that it has no ubiquitous symbols, hence it fails Lemma 8.39.) Had we adopted a set of rewrite rules that only allows shattering of the form $d[A/Z]$ then the query were forbidden: there are only two levels, and the queries $d[S_1 = \texttt{false}] =$

false, and $d[S_1 = \textit{true}] \equiv S_1(x,y_1), S_2(x,y_2), S_3(x,y_3) \vee S_2(x_1,y), S_3(x_1,y)$ *are safe. We show here that our hardness proof for type 2-2 queries fails on* d*. To see this, notice that* $n_X = n_Y = 3$*, and the left semilattice has 7 elements. The three coatoms are:*

$$\begin{aligned} d_1^{0*} &= \bigvee_{\tau\in[3]} \big(S_1^{0\tau}(x,y_1) \vee \\ &\qquad S_2^{0\tau}(x_1,y), S_3^{0\tau}(x_1,y), S_1^{0\tau}(x_2,y), S_3^{0\tau}(x_2,y), S_1^{0\tau}(x_3,y), S_2^{0\tau}(x_3,y)\big) \\ &= \bigvee_{\tau\in[3]} S_1^{0\tau}(x,y_1) \\ d_2^{0*} &= \bigvee_{\tau\in[3]} S_2^{0\tau}(x,y_2) \\ d_3^{0*} &= \bigvee_{\tau\in[3]} S_3^{0\tau}(x,y_3) \end{aligned}$$

In addition the semilattice contains all unions, $d_{12}^{0*} = d_1^{0*} \vee d_2^{0*}$*, etc. We claim that the connected component containing the symbols* $S_i^{\tau_1\tau_2}$ *with levelings* $\tau_1, \tau_2 > 0$ *is the same in* $d_{12}^{0*} \vee d^{**}$ *and in* $d_{123}^{0*} \vee d^{**}$*. This is because all components in* d^{**} *that have common symbols with* d^{0*} *(i.e. symbols of the form* $S_i^{0\tau}$*) are redundant in both* $d_{12}^{0*} \vee d^{**}$ *and* $d_{123}^{0*} \vee d^{**}$*. For simple illustration, the subcomponent* $S_2^{01}(x_1,y), S_3^{01}(x_1,y), S_1^{11}(x_2,y), S_3^{11}(x_2,y), S_1^{21}(x_3,y), S_2^{21}(x_3,y)$*, is redundant in both queries. In other words, the two queries can be written as* $d_{12}^{0*} \vee d'$ *and* $d_{123}^{0*} \vee d'$*, where* d' *has no common symbols with* d^{0*}*. This implies that none of the symbols in* d' *can serve as distinguished variable. A simpler argument implies that none of the symbols* S_i^{0*} *can serve as distinguished variable: for any* τ*, the symbol* $S_3^{0\tau}$ *does not occur at all in* $d_{12}^{0*} \vee d^{**}$*, hence it cannot serve as distinguished variable, and similarly for* $S_1^{0\tau}$ *and* $S_2^{0\tau}$*.*

8.5 Finding a Non-singular Matrix in PTIME

Finally, we show how to solve the linear systems of equations derived from Eq. 39 (for queries of type 1-1) or Prop. 8.29 (for queries of type 2-2) in PTIME, given that the Jacobian of its $\bar{y}$ functions is non-zero.

The principle that we apply is very simple. Given a polynomial in a single variable x, $f(x) = \sum_k a_k x^k$, one can find in PTIME in n a value v s.t. $f(v) \neq 0$: indeed, simply choose $n+1$ distinct real values $v_0, v_1, \ldots, v_n$ and compute $f(v_j)$ for $j = 0, \ldots, n$. At least one $\neq 0$, otherwise the polynomial is identical zero. We apply the same principle to our matrix.

More precisely, we prove the following. Recall that whenever we write $P = Q$ for two multivariate polynomials, we mean that they are identical.

PROPOSITION 8.44. *Let* $\bar{x} = (x_1, \ldots, x_m)$ *be* m *variables, and* $\bar{G}(\bar{x}) = (G_1(\bar{x}), \ldots, G_n(\bar{x}))$ *be* n *multivariate polynomials in* $\bar{x}$*. Consider* n *distinct copies of* $\bar{x}$*, denoted* $\bar{x}_i$*,* $i = 1, n$*, and let* $\bar{X} = (\bar{x}_i)_{i=1,n}$ *be the set of* $m \cdot n$ *distinct variables. Define the following* $n \times n$ *matrix of polynomials:*

$$M(\bar{G}) \;=\; (G_j(\bar{x}_i))_{i,j=1,n} \tag{49}$$

Then there exists an algorithm that runs in time $(n+1)^{O(m)}$ *and either determines that* $det(M) = 0$ *(as a multivariate polynomial in* $\bar{X}$*) or finds values* $\bar{V}$ *s.t.* $det(M(\bar{G})[\bar{V}/\bar{X}]) \neq 0$.

We explain how to use the proposition in the case of queries of type 1-1; the discussion extends also to queries of type 2-2. Consider Eq. 39. For each term $\mathbf{k} = (k_{11}, k_{12}, k_{21}, k_{22})$, define the polynomial $G_{\mathbf{k}}(z_1, z_2, z_3, z_4) = y_{11}^{k_{11}} \cdots y_{22}^{k_{22}}$. There are $(m+1)^4$ polynomials $G_{\mathbf{k}}$. One can think of the matrix of the system of $(m+1)^4$ copies of Eq. 39 as the matrix $M(\bar{G})$, where each row $i = 1, (m+1)^4$ has a different set of values $\bar{z}_i$. The proposition gives us a PTIME procedure to find these $(m+1)^4$ values such that the resulting matrix is non-singular. Thus, we run the oracle for $P(d^L)$ on the database obtained by setting the distinguished variables $(z_1, \ldots, z_4)$ to the values $\bar{z}_i$, for each $i = 1, (m+1)^4$: the proposition ensures that the resulting matrix is non-singular, and we can therefore invert it in PTIME. In the rest of this section we prove the propositions, using two lemmas.

LEMMA 8.45. *Let* $P(x_1, \ldots, x_m)$ *be a multivariate polynomial of degree* n*, with* $m = O(1)$ *variables. Suppose we have an Oracle that, given values* $v_1, \ldots, v_m$*, computes* $P(v_1, \ldots, v_m)$ *in time* T*. Then there exists an algorithm that runs in time* $O((n+1)^m T)$ *and either determines that* $P = 0$ *or returns a set of values* $\bar{v} = (v_1, \ldots, v_m)$ *s.t.* $P(\bar{v}) \neq 0$.

PROOF. By induction on m. Choose $n+1$ distinct values for the last variable: $x_m = v_m^0$, $x_m = v_m^1$, ..., $x_m = v_m^n$. For each value v_m^i we substitute $x_m = v_m^i$ in P. We obtain a new polynomial, $Q = P[x_m/v_m^i]$, with $m-1$ variables. By induction, we can find in time $O((n+1)^{m-1}T)$ a set of values $v_1, \ldots, v_{m-1}$ s.t. $Q[x_1/v_1, \ldots, x_{m-1}/v_{m-1}] \neq 0$, or determine that none exists. If for some i we find such values, then augment them with v_m^i and return $(v_1, \ldots, v_{m-1}, v_m^i)$. If $Q = P[v_m^i/x_m] = 0$ for all $i = 0, n$, then P is divisible by $\prod_i (x_m - v_m^i)$, by Hilbert's Nullstellensatz. Since the degree of x_m in P is at most n, it follows $P = 0$. □

Fix n multivariate polynomials $G_1, \ldots, G_n$. Called them *linearly independent* if, for any constants $c_1, \ldots, c_n$, if $c_1 G_1 + \ldots + c_n G_n = 0$ then $c_1 = \ldots = c_n = 0$.

LEMMA 8.46. *The polynomials* $G_1, \ldots, G_n$ *are linearly dependent iff* $det(M(\bar{G})) = 0$*, where* M *is given by Eq. 49.*

PROOF. The "only if" direction follows immediately from the fact that the columns in M are linearly dependent. For the "if" direction, we prove by induction on n that, if n polynomials $G_1, \ldots, G_n$ are linearly independent, then $det(M) \neq 0$. Let M' be the $(n-1) \times (n-1)$ upper-left minor of M. Since $G_1, \ldots, G_{n-1}$ are also linearly independent, $det(M') \neq 0$. Hence, there exists values $\bar{V}' = (\bar{v}_1, \ldots, \bar{v}_{n-1})$ s.t. $det(M')[\bar{V}'] \neq 0$. Define the matrix $M'' = M[\bar{V}']$: that is, we substitute $\bar{x}_1, \ldots, \bar{x}_{n-1}$ with the values $\bar{V}'$, and keep only the variables $\bar{x}_N$. We prove that $det(M'') \neq 0$. The matrix M'' has in the last row the polynomials $G_1(\bar{x}_n), \ldots, G_n(\bar{x}_n)$, and has constants in all other rows. Its value is a linear combination of the polynomials in the last row: $det(M'') = c_1 \cdot G_1 + \ldots + c_n \cdot G_n$. The last coefficient, $c_n = det(M')[\bar{V}']$, is non-zero. Since at least one coefficient is non-zero and we assumed the polynomials to be independent, it follows that $det(M'') \neq 0$. This implies $det(M) \neq 0$. □

We can now prove Prop. 8.44.We proceed by induction on n. Consider the upper left minor M' of dimensions $(n-1) \times (n-1)$. Apply induction to the matrix M'. If we determine that $det(M') = 0$, then this implies that $G_1, \ldots, G_{n-1}$ are linearly dependent, and therefore so are $G_1, \ldots, G_n$, which implies $det(M) = 0$. Otherwise, we have computed a set of values $\bar{V}' = (\bar{v}_1, \ldots, \bar{v}_{n-1})$ that make the upper left minor M' non-singular. Denote $M'' = M[\bar{x}_1 = \bar{v}_1, \ldots, \bar{x}_{n-1} = \bar{v}_{n-1}]$. The last row of this matrix consists of $G_1(\bar{x}_n), \ldots, G_n(\bar{x}_n)$; all other entries are constants. Denote $P(\bar{x}_n) = det(M)$. This is a multivariate polynomial. We cannot compute the polynomial explicitly, since it has exponentially many coefficients. However, for any values $\bar{v}_n$, we can compute $P[\bar{x}_n = \bar{v}_n]$ in time $O(n^3)$, by first substituting $\bar{x}_n = \bar{v}_n$ in all polynomials in the last row of M'', then computing the determinant. We use this oracle and Lemma 8.45 to compute in time $O((n+1)^{m+3})$ a set of values $\bar{v}_n$ for which $P[\bar{x}_n = \bar{v}_n] \neq 0$; if no such value exists, then $G_1, \ldots, G_n$ are linearly dependent, since P is a linear combination of $G_1, \ldots, G_n$, and therefore $det(M) = 0$. Otherwise, return the vector consisting of $\bar{V}'$ concatenated with $\bar{v}_n$.

9. CONCLUSIONS

In this paper we have studied a fundamental computational problem connecting logic an probability theory: given a query and a probabilistic database, compute the probability of the query on that database. We have established a dichotomy for unions of conjunctive queries (also known as the positive, existential fragment of First Order logic): for every query Q, either $P(Q)$ can be computed in PTIME in the size of the database, or it is #P-hard. We call the query safe in the first case, and unsafe in the second case. Safety/unsafety can be decided solely on the syntax of the query, where "syntax" includes, unexpectedly, the Mobius function of a certain lattice derived from the query. For safe queries we have given a simple, yet quite non-obvious algorithm, which alternates between an inclusion/exclusion step over the CNF representation of the query, and an elimination step of one existential variable. For the hardness proof, we have made novel use of techniques from classical algebra and calculus, such as the use of the Jacobian, Cauchy's double alternant, and irreducible multivariate polynomials, in order to prove hardness by reduction from Provan and Ball's *Positive Partitioned 2CNF* counting problem.

Acknowledgments We thank Christoph Koch and Paul Beame for pointing us (independently) to incidence algebras, and Phokion Kolaitis for discussions on the complexity classes #P and $\text{FP}^{\#P}$.

REFERENCES

Abiteboul, S., Hull, R., and Vianu, V. 1995. *Foundations of Databases*. Addison Wesley Publishing Co.

Cavallo, R. and Pittarelli, M. 1987. The theory of probabilistic databases. In *Proceedings of VLDB*. 71–81.

Chandra, A. and Merlin, P. 1977. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of 9th ACM Symposium on Thoery of Computing*. Boulder, Colorado, 77–90.

Creignou, N. and Hermann, M. 1996. Complexity of generalized satisfiability counting problems. *Inf. Comput 125,* 1, 1–12.

Dalvi, N., Schnaitter, K., and Suciu, D. 2010. Computing query probability with incidence algebras. In *PODS*. 203–214.

Dalvi, N. and Suciu, D. 2004. Efficient query evaluation on probabilistic databases. In *VLDB*.

Dalvi, N. and Suciu, D. 2007a. The dichotomy of conjunctive queries on probabilistic structures. In *PODS*. 293–302.

Dalvi, N. and Suciu, D. 2007b. Management of probabilistic data: Foundations and challenges. In *PODS*. Beijing, China, 1–12. (invited talk).

Dalvi, N. N. and Suciu, D. 2006. The dichotomy of conjunctive queries on probabilistic structures. *CoRR abs/cs/0612102*.

Darwiche, A. 2000. On the tractable counting of theory models and its application to belief revision and truth maintenance. *CoRR cs.AI/0003044*.

Darwiche, A. 2009. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press.

Darwiche, A. and Marquis, P. 2002. A knowledge compilation map. *J. Artif. Int. Res. 17,* 1, 229–264.

de Salvo Braz, R., Amir, E., and Roth, D. 2005. Lifted first-order probabilistic inference. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*. 1319–1125.

Domingos, P. and Lowd, D. 2009. *Markov Logic: An Interface Layer for Artificial Intelligence*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Golumbic, M. C., Mintz, A., and Rotics, U. 2006. Factoring and recognition of read-once functions using cographs and normality and the readability of functions associated with partial k-trees. *Discrete Applied Mathematics 154,* 10, 1465–1477.

Gomes, C. P., Sabharwal, A., and Selman, B. 2009. Model counting. In *Handbook of Satisfiability*. 633–654.

Grädel, E., Gurevich, Y., and Hirsch, C. 1998. The complexity of query reliability. In *PODS*. 227–234.

Gurvich, V. 1977. Repetition-free boolean functions. *Uspekhi Mat. Nauk 32*, 183–184.

Jha, A. and Suciu, D. 2011. Knowledge compilation meets database theory : Compiling queries to decision diagrams. (To appear in ICDT).

Knuth, K. H. 2005. Lattice duality: The origin of probability and entropy. *Neurocomputing 67*, 245–274.

Krattenthaler, C. 1999. Advanced determinant calculus. *Seminaire Lotharingien Combin 42 (The Andrews Festschrift)*, 1–66. Article B42q.

Libkin, L. 2004. *Elements of Finite Model Theory*. Springer.

Olteanu, D. and Huang, J. 2009. Secondary-storage confidence computation for conjunctive queries with inequalities. In *SIGMOD*. 389–402.

Olteanu, D., Huang, J., and Koch, C. 2009. Sprout: Lazy vs. eager query plans for tuple-independent probabilistic databases. In *ICDE*. 640–651.

Poon, H. and Domingos, P. 2007. Joint inference in information extraction. In *AAAI*. 913–918.

Poon, H. and Domingos, P. 2010. Unsupervised ontology induction from text. In *ACL*. 296–305.

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

Richardson, M. and Domingos, P. 2006. Markov logic networks. *Machine Learning 62,* 1-2, 107–136.

Sagiv, Y. and Yannakakis, M. 1980. Equivalences among relational expressions with

the union and difference operators. *Journal of the ACM 27*, 633–655.

Sarma, A. D., Theobald, M., and Widom, J. 2008. Exploiting lineage for confidence computation in uncertain and probabilistic databases. In *ICDE*. 1023–1032.

Sen, P. and Deshpande, A. 2007. Representing and querying correlated tuples in probabilistic databases. In *ICDE*.

Singla, P. and Domingos, P. 2006. Entity resolution with markov logic. In *ICDM*. 572–582.

Stanley, R. P. 1997. *Enumerative Combinatorics*. Cambridge University Press.

Suciu, D., Olteanu, D., Ré, C., and Koch, C. 2011. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers.

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

Wegener, I. 2000. *Branching programs and binary decision diagrams: theory and applications*. SIAM.

Wegener, I. 2004. BDDs–design, analysis, complexity, and applications. *Discrete Applied Mathematics 138,* 1-2, 229–251.

A. PRIOR WORK ON CONJUNCTIVE QUERIES

In previous work [Dalvi and Suciu 2007a; Dalvi and Suciu 2006] we described a PTIME algorithm for conjunctive queries, and claimed that it is complete, in the sense that every query on which the algorithm fails is #P-hard. We clarify here the status of that algorithm. First, the algorithm is sound (after fixing some minor mistakes, as we explain below): whenever it succeeds on a query, it correctly computes its probability. Second, its completeness remains open; as we explain below, the algorithm is complete on queries over vocabularies of arities ≤ 2, but it remains open whether it is complete in general. Our initial proof consisted of showing that, if the algorithms gets stuck on a query Q, then $h_k \leq^{\text{FO}}_{\text{prob}} Q$, where k is is the length of the inversion without eraser. But the proof was very complex, since it combined all of Sect. 6, Sect. 7, and parts of Sect. 8 in one single giant step. It was also very sensitive to small changes in the algorithm, and we could not resolve some gaps in the proof.

With the techniques developed in this paper we have a much better understanding of the algorithm in [Dalvi and Suciu 2007a] and its limitations. We review it below. We will use the terminology in the current paper, and will use apostrophes when referring to the terminology in [Dalvi and Suciu 2007a].

Notations

Although the goal in [Dalvi and Suciu 2007a] was to study only conjunctive queries, due to ranking, the algorithm considered unions of conjunctive queries:

$$Q = q_1 \vee \ldots \vee q_n \tag{50}$$

The "strict coverage" in [Dalvi and Suciu 2007a] means, in our current terminology, that the query is ranked. Let $\mathcal{F} = \{f_1, \ldots, f_k\}$ be the set of all components in all conjunctive: they were called "factors" in [Dalvi and Suciu 2007a], but we will continue to refer to them as components, to have a consistent terminology in this paper. A "coverage" is a pair $\mathcal{C} = (\mathcal{F}, C)$ where $\mathcal{F}$ is a set of components, $|\mathcal{F}| = k$, and $C = \{c_1, \ldots, c_n\}$ where each $c_i \subseteq [k]$ represents the components in the conjunctive query q_i. The coverage C uniquely determines the UCQ query Q.

Let f be a component, and x a variable. Denote $at(x)$ the set of atoms that contain a variable x. Call f *hierarchical* if for any two distinct variables x, y we have either $at(x) \subseteq at(y)$ or $at(x) \cap at(y) = \emptyset$ or $at(x) \supseteq at(y)$. We will assume that all components are hierarchical: otherwise, the query is #P-hard.

Recall (Def. 4.1) that a ranking of f_i is a partial order on the variables, $(Var(f_i), \preceq_i)$, such that, for any atom $S(x_1, \ldots, x_m)$ in f_i, $x_1 \prec_i x_2 \prec_i \ldots \prec_i x_m$. We say that the ranking is *compatible* with the hierarchy if $at(x) \supset at(y)$ implies $x \prec_i y$, for any two variables x, y. If S is a relational symbol of arity m and π is a permutation on $[m]$ then denote S^π the relational symbol obtained from S by permuting the attributes according to π. If Π denotes a set of permutations $(\pi_S)_{S \in \mathbf{R}}$, one for each symbol in the vocabulary, then f_i^Π denotes the query where each atom $S(x_1, \ldots x_m)$ is replaced with $S(x_{\pi_S(1)}, \ldots, x_{\pi_S(m)})$. A set of components $\{f_1, \ldots, f_k\}$ is called *inversion free* if each component is hierarchical, and if there exists a set of permutations Π for the vocabulary such that, for each $i = 1, m$ there exists a ranking $(Var(f_i^\Pi), \preceq_i)$ that is compatible with the hierarchy on f_i^Π. We show three simple examples. First, the two components $f_1 = R(x_1), S(x_1, y_1)$ and $f_2 = T(x_2), S(x_2, y_2)$ are inversion free, because the ranking $x_1 \prec_1 y_1$ is compatible with $at(x_1) \supset at(y_1)$ and the ranking $x_2 \prec_2 y_2$ is compatible with $at(x_2) \supset at(y_2)$. Second, consider the components $f_1 = R(y_1), S(x_1, y_1)$ and $f_2 = T(y_2), S(x_2, y_2)$. This is also inversion free: first permute the attributes of S, s.t. $S^\pi(y, x) = S(x, y)$ then the two components become $f_1^\pi = R(y_1), S^\pi(y_1, x_1)$ and $f_2^\pi = T(y_2), S^\pi(y_2, x_2)$, now they are obviously inversion-free. Finally, consider $f_1 = R(x_1), S(x_1, y_1)$ and $f_2 = T(y_2), S(x_2, y_2)$: this has an inversion because no matter how we permute the attributes of S we cannot make it compatible with *both* hierarchy orders $at(x_1) \supset at(y_1)$ and $at(x_2) \subset at(y_2)$.

We will omit mentioning the permutation Π for an inversion-free set of components $F = \{f_1, \ldots, f_k\}$ when clear from the context, and will only indicate the rankings $\preceq_1, \ldots, \preceq_m$. Inversion-free sets have the following nice properties. (1) F has a separator. Indeed, let x_i be the minimal variable in $Var(f_i)$ w.r.t. the order $\preceq_i$. Then $at(x_i)$ is a maximal set, hence it includes all atoms in f_i, because f_i is hierarchical, and, moreover, it occurs on the first position in each atom; therefore the set of variables $x_1, \ldots, x_m$ forms a separator. (2) Let a be a constant, and let $F' = \{g_1, \ldots, g_p\}$ be the set of all components of all queries $f_i[a/x_i]$, excluding the components that consist of a single ground tuple. If g_j is a component of $f_i[a/x_i]$, denote $\preceq'_j$ the restriction of $\preceq_i$ to $Var(g_j)$. Then $\preceq'_j$ is a ranking for g_j compatible with its hierarchy. In particular, F' is inversion-free.

Review of the Algorithm from [Dalvi and Suciu 2007a]

Call a non-empty subset $\sigma \subseteq [k]$ a "signature", and denote f_σ the conjunctive query $\bigwedge_{i \in \sigma} f_i$. Define the lattice $K(\mathcal{C})$ to be $(K \cup \{\hat{1}\}, \leq)$ where $K = \{\sigma \mid f_\sigma \Rightarrow Q\}$, and $\sigma_1 \leq \sigma_2$ if $\sigma_1 \supseteq \sigma_2$. Note $\sigma_1 \leq \sigma_2$ implies $f_{\sigma_1} \Rightarrow f_{\sigma_2}$, but the converse is not true in general: we may have two distinct components such that $f_1 \Rightarrow f_2$, yet $\{1\} \not\leq \{2\}$. The lattice meet corresponds to set union, i.e. $\sigma_1 \wedge \sigma_2 = \sigma_1 \cup \sigma_2$. Assuming the expression Q in Eq. 50 has no redundant queries q_i, the co-atoms of the lattice K are precisely the sets $c_1, c_2, \ldots, c_n \in C$.

We did not introduce the notion of a lattice in [Dalvi and Suciu 2007a], but defined a quantity that is related to the Mobius function. For $s \subseteq [n]$, let its signature

be $sig(s) = \bigcup_{i \in s} c_i$. In Definition 2.10 we set: $N(\mathcal{C}, \sigma) = (-1)^{|\sigma|} \sum_{s \subseteq C, sig(s)=\sigma} (-1)^{|s|}$. Then, one can check that:

$$N(\mathcal{C}, \sigma) = \begin{cases} (-1)^{|\sigma|} \mu_K(\sigma, \hat{1}) & \text{if } \sigma \in K \\ 0 & \text{if } \sigma \notin K \end{cases}$$

To compute $P(Q)$ we will use Mobius' inversion formula on Eq. 50, and for that we need to define a different lattice. Call a set $s \subseteq [n]$ a "cover", define its "signature" to be $sig(s) = \bigcup_{i \in s} c_i \subseteq [k]$, and denote $q_s = f_{sig(s)}$. Equivalently, $q_s = \bigwedge_{i \in s} q_i$ and, in particular $q_{\{i\}} = q_i$. Define the *closure* of a cover $s \subseteq [n]$ to be $\bar{s} = \{i \mid q_s \Rightarrow q_i\} \subseteq [n]$. Define the DNF lattice of the coverage C to be $L(\mathcal{C}) = (L, \leq)$ where $L \subseteq [n]$ is the set of closed sets, and $s_1 \leq s_2$ if $q_{s_1} \Rightarrow q_{s_2}$. By Prop. 2.13 the latter holds iff $\forall j \in s_2$, $\exists i \in s_1$ such that $f_i \Rightarrow f_j$ and, since s_1 is closed, it implies that $j \in s_1$. Thus, $s_1 \leq s_2$ iff $s_1 \supseteq s_2$. The definition of the DNF lattice is dual to that of the CNF lattice in Def. 4.8.

Thus, the lattice K consists of all subsets of $[k]$ that contain at least some c_i, while L consists of all subsets of $[n]$ that are closed; we need K because that's what we used in [Dalvi and Suciu 2007a], and we need L because that is the lattice were we can compute $P(Q)$. Their relationship is the following. L is a meet-sublattice of K, through the injection $s \mapsto sig(s)$: indeed, if $sig(s_1) = sig(s_2) = \sigma$ then $q_{s_1} \equiv f_\sigma \equiv q_{s_2}$ proving that $\bar{s}_1 = \bar{s}_2$ and, hence, $s_1 = s_2$ because s_1, s_2 are closed. Moreover, L and K have the same sets of co-atoms, modulo the injection, namely $c_1, c_2, \ldots, c_n$ in K and $\{1\}, \ldots, \{n\}$ in L. The following property holds for a meet-sublattice L that contains all co-atoms [Suciu et al. 2011, Corollary 4.17]:

$$\begin{aligned} \mu_K(sig(s), \hat{1}) &= \mu_L(s, \hat{1}) \\ \mu_K(\sigma, \hat{1}) &= 0 \qquad \text{if } \sigma \neq sig(s), \forall s \in L \end{aligned} \tag{51}$$

The Expansion Theorem 2.11 in [Dalvi and Suciu 2007a] is, at its essence, Mobius' inversion formula:

$$P(Q) = -\sum_{s \in L} \mu_L(s, \hat{1}) P(q_s) = -\sum_{\sigma \in K} \mu_K(\sigma, \hat{1}) P(f_\sigma) = -\sum_{\sigma \subseteq [k]} N(\mathcal{C}, \sigma)(-1)^{|\sigma|} P(f_\sigma) \tag{52}$$

In Theorem 2.11, the term $(-1)^{|\sigma|} P(f_\sigma)$ is replaced with a more complex formula, (an inclusion-exclusion expression over the active domain), which we do not need in our discussion here.

There are two algorithms in [Dalvi and Suciu 2007a], both starting from Eq. 52.

Algorithm 1: Inversion-Free Queries. The first algorithm shows how to compute $P(f_\sigma)$ in PTIME when the set of components of f_σ is inversion-free: this is Theorem 3.8 [Dalvi and Suciu 2007a]. While this algorithm is definitely more complicated than our (more general) algorithm 2, it is quite clearly described in [Dalvi and Suciu 2007a], and does not require further review. In [Dalvi and Suciu 2007a] we only use this result to claim that Q is in PTIME if the set of all components is inversion free, the following stronger result holds immediately: if for every σ s.t. $N(\mathcal{C}, \sigma) \neq 0$, f_σ is inversion-free, then Q is in PTIME.

Thus, by ignoring terms $P(f_\sigma)$ where $N(\mathcal{C}, \sigma) = 0$, we obtain the following completeness result for Algorithm 1:

PROPOSITION A.1. *If all relational symbols in the vocabulary have arity ≤ 2, then Algorithm 1 in [Dalvi and Suciu 2007a] is complete.*

PROOF. Let Q be any UCQ query, and consider its CNF lattice (Def. 4.8). The lattice elements are labeled with queries of the form $f_{i_1} \vee \ldots \vee f_{i_m}$. If the set $\{f_{i_1}, \ldots, f_{i_m}\}$ is inversion free then their union has a separator and, because all symbols are binary, the converse holds too. It follows that the only non-zero terms $P(f_{i_1} \vee \ldots \vee f_{i_m})$ in Mobius' inversion formula on the CNF lattice (Eq. 12) are over sets of components that are inversion-free. Use inclusion-exclusion to convert these terms to sums of probability of conjuncts, i.e. $P(f_{i_1} \vee \ldots \vee f_{i_m}) = -\sum_{\sigma \subseteq \{i_1,\ldots,i_m\}, \sigma \neq \emptyset} P(f_\sigma)$. It follows that the only terms that don't vanish in Eq. 52 are inversion-free. □

We illustrate Algorithm 1 on two examples

$$\begin{aligned} Q_V =& R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(y_2) \vee R(x_3), T(y_3) \\ Q_W =& (h_{30} \vee h_{32}) \wedge (h_{30} \vee h_{33}) \wedge (h_{31} \vee h_{33}) \\ =& h_{30}, h_{31} \vee h_{30}, h_{33} \vee h_{32}, h_{33} \end{aligned}$$

The first, Q_V, is from Example 3.5, the second is shown in Fig. 1. In both cases the DNF lattice L is shaped like a W and is isomorphic to the lattice shown in Fig. 1. The only lattice element where q_s has an inversion is the bottom element, but it also has $\mu_L(s, \hat{1}) = 0$. By Eq. 51 and Eq. 52, the only elements where f_σ has an inversion has $N(\mathcal{C}, \sigma) \neq 0$.

Algorithm 2: Erasers The major limitation of the approach in [Dalvi and Suciu 2007a] is that, by working on the DNF rather than the CNF representation, we could not use the projection rule (step 17 of algorithm 2). This required a much more complex approach, using a technique called "eraser". It is open whether this approach is complete. We describe here the main idea, pointing out and fixing some errors in [Dalvi and Suciu 2007a], and proving its soundness. We will illustrate on the following example:

$$Q'_V = R(z_1, x_1), S(z_1, x_1, y_1) \vee S(z_2, x_2, y_2), T(z_2, y_2) \vee R(z_3, x_3), T(z_3, y_3) = f_1 \vee f_2 \vee f_3$$

The variables z_1, z_2, z_3 form a separator, so algorithm 2 would simply substitute them with a constant, then the query becomes isomorphic to Q_V. Instead, the algorithm in [Dalvi and Suciu 2007a] applies the inclusion/exclusion formula obtaining 7 terms, of which two have inversions: $P(f_1, f_2)$ and $P(f_1, f_2, f_3)$:

$$P(Q'_V) = P(f_1) + P(f_2) + P(f_3) - P(f_1, f_3) - P(f_2, f_3) - P(f_1, f_2) + P(f_1, f_2, f_3)$$

Let $f_1, f_2 \in \mathcal{F}$ be two components, and let $S(x_1, x_2, \ldots, x_m)$ be an atom in f_1 (where $x_1, \ldots, x_m$ must be distinct variables, because the query is ranked), and let $S(y_1, y_2, \ldots, y_m)$ be an atom in f_2. Denote $f_{12} = \theta(f_1, f_2)$ the MGU (Most General Unifier) that unifies the two S-atoms. An "eraser" is a component $f_3 \in \mathcal{F}$ with the following properties: (a) the logical implication $\theta(f_1, f_2) \Rightarrow f_3$ holds (this is query containment); (b) for any $\sigma \subseteq [k]$, $N(\sigma \cup \{1, 2\}) = N(\sigma \cup \{1, 2, 3\})$, and (c) for each $i = 1, 2, 3$ there exists a partial order $(Var(f_i), \preceq_i)$ that is compatible with the hierarchy in f_i such that there exists two permutations Π_1, Π_2 on the attributes of

the vocabulary for which $\preceq_1, \preceq_3$ are rankings for $f_1^{\Pi_1}, f_3^{\Pi_1}$ and $\preceq_2, \preceq_3$ are rankings for $f_2^{\Pi_2}, f_3^{\Pi_2}$. In particular, the set $\{f_1, f_3\}$ is inversion-free, and the set $\{f_2, f_3\}$ is inversion-free, but not necessarily the set $\{f_1, f_2, f_3\}$.

In our running example, f_3 is an eraser for f_1, f_2. Indeed, $\theta(f_1, f_2) \equiv R(z, x), S(z, x, y), T(z, y)$ and condition (a) holds trivially. To check condition (b) it suffices to notice that $N(\mathcal{C}, \{1, 2\}) = -1 = N(\mathcal{C}, \{1, 2, 3\})$. To check the condition (c), we order the variables as follows:

$$f_1 : z_1 \prec_1 x_1 \prec_1 y_1$$
$$f_2 : z_2 \prec_2 y_2 \prec_2 x_2$$
$$f_3 : z_3 \prec_3 x_3, \quad z_3 \prec_3 y_3$$

Then $\prec_1, \prec_3$ are compatible with the hierarchies on f_1, f_3 respectively (no permutation is needed); for f_1, f_3, consider the permutation that switches the x, y attributes in S, i.e. $S^\pi(z, y, x) = S(z, x, y)$: then f_2^π and f_3^π become $f_2^\pi = S^\pi(z_2, y_2, x_2), T(z_2, y_2)$ and $f_3^\pi = f_3 = R(z_3, x_3), T(z_3, y_3)$. Notice that f_3 remains unchanged under the two permutations, but that in f_1 and f_2 we permuted differently the order of the attributes in S.

We now state the soundness of the eraser technique, and sketch a proof. In [Dalvi and Suciu 2007a] soundness was proven using Lemma 4.13, which in essence says that, the terms $P(f_\sigma, f_1, f_2)$ and $P(f_\sigma, f_1, f_2, f_3)$ can be reduced to inversion-free queries. Its proof is correct (assuming our current definition of erasers, see the discussion below), but quite complicated; to illustrate it we give here a different, much simpler proof, assuming for simplicity that $\sigma = \emptyset$. Let be x_1, x_2, x_3 the minimial variables under the order $\preceq_1, \preceq_2, \preceq_3$, and write $f_i \equiv \exists x_i.g_i$ for $i = 1, 2, 3$, where g_i is an expression where x_i is a free variable: note that x_i occurs in all atoms, on the first position. The two terms have opposite signs in Eq. 52, and they can be grouped as follows,

$$P(f_1, f_2) - P(f_1, f_2, f_3) = P(f_1, f_2, \neg f_3) = P(f_1, \neg f_3) + P(f_2, \neg f_3) - P((f_1 \vee f_2), \neg f_3)$$

The first two terms are inversion-free: more precisely $P(f_1, \neg f_3) = P(f_1) - P(f_1, f_3)$ where each of the two queries f_1 and f_1, f_3 are inversion free. So we focus on the last query above and write it as:

$$(f_1 \vee f_2), \neg f_3 \equiv [\exists y.(g_1[y/x_1] \vee g_2[y/x_2])] \wedge \neg f_3 \equiv [\exists y.(g_1[y/x_1] \vee g_2[y/x_2]) \wedge \neg g_3[y/x_3]] \wedge \neg f_3$$

Denote $g_i' = g_i[y/x_i]$ and, for a fixed constant a, denote $f_i' = f_i[a/x_i] = g_i[a/x_i]$. Then:

$$P((f_1 \vee f_2), \neg f_3) = P(\exists y.[(g_1' \vee g_2') \wedge \neg g_3'] \wedge \neg f_3) = P(\exists y.[(g_1' \vee g_2') \wedge \neg g_3'] \vee f_3) - P(f_3)$$
$$P(\exists y.[(g_1' \vee g_2') \wedge \neg g_3'] \vee f_3) = P(\exists y.[[(g_1' \vee g_2') \wedge \neg g_3'] \vee g_3']) = 1 - \prod_{a \in ADom} (1 - P((f_1' \vee f_2') \wedge \neg f_3'))$$

Looking at the query $(f_1' \vee f_2') \wedge \neg f_3'$, we consider two cases. The first is when the atom S in f_1, f_2 had arity 1: in that case, after substituting with a constant a it becomes a ground atom (arity 0 after shattering), the two S-atoms in $f_1[a/x_1]$ and $f_2[a/x_2]$ unify, which means that $f_1', f_2' \Rightarrow f_3'$, which further implies that the two events $f_1', \neg f_3'$ and $f_2', \neg f_3'$ are exclusive: hence we have $P((f_1' \vee f_2') \wedge \neg f_3') = P(f_1' \wedge \neg f_3') + P(f_2' \wedge \neg f_3')$, and every subquery is inversion-free. The second case is when the atom S had arity 2 or larger. Then we apply inductively the same

argument to $P((f_1' \vee f_2') \wedge \neg f_3')$. This proves that $P(f_1, f_2) - P(f_1, f_2, f_3)$ can be reduced to computing the probability of several inversion-free queries.

Finally, we correct three mistakes in [Dalvi and Suciu 2007a]. First, there we applied an eraser to every pair of components f_1, f_2 that share a common symbol. When no eraser f_3 existed, then we considered adding the unifier $\theta(f_1, f_2)$ as a new component, when this was hierarchical. In fact, we add *all* unifiers that satisfied a condition called "hierarchical", and used that as eraser. As we saw in Prop. A.1, adding unnecessary unifiers is not needed when there are no inversions: in fact, once we start adding such unifiers, we no longer know whether completeness (Prop. A.1) still holds. Thus, it is open which unifiers should be added in general. A second, minor point is that we defined erasers slightly more generally to consists of a set of components, rather than one single component. It is open whether these more general erasers increase the expressive power, so we did not mention them here. The third is more important: the definition of an eraser was too weak, making the algorithm unsound. The weaker condition is insufficient for proving Lemma 4.13, and therefore the algorithm is not sound as stated. To obtain soundness, the definition of an eraser must be restricted to condition (c) (and further refined to handle chains of unifiers): we have sketched a proof of soundness earlier. We illustrate this issue with two examples:

$$\begin{aligned}
Q_1 =& [S_1(x, y_1), S_2(x, y_1), S_3(x, y_1)], [S_1(x, y_2), S_2(x, y_2), S_4(x, y_2)] \vee \\
& [S_1(x_1, y), S_3(x_1, y), S_4(x_1, y)], [S_2(x_2, y), S_3(x_2, y), S_4(x_2, y)] \\
Q_2 =& [S(x, y_1), S_1(x, y_1)], [S(x, y_2), S_2(x, y_2)] \vee \\
& S_1(x, y), S_2(x, y), S_3(x, y), S_4(x, y) \vee \\
& [S'(x_1, y), S_3(x_1, y)], [S'(x_2, y), S_4(x_2, y)]
\end{aligned}$$

The first query has two components, denote them f_1, f_2, and each component has two subcomponents, shown in square brackets. Consider unifying the atom $S_1(x, y_1)$ with $S_1(x_1, y)$: the resulting query minimizes to $\theta(f_1, f_2) = f_3 = S_1(x, y), S_2(x, y), S_3(x, y), S_4(x, y)$: this is a hierarchical query and, therefore, according to our definition [Dalvi and Suciu 2007a; Dalvi and Suciu 2006], would be included in $\mathcal{F}^*$. This implies that all unifiers belong to $\mathcal{F}^* = \{f_1, f_2, f_3\}$; by Lemma 4.13 the query should be in PTIME, which is incorrect, because Q_1 is a forbidden query, hence it is #P-hard. In our new definition, f_3 is not an eraser for f_1, f_2 because it cannot be ranked to be simultaneously inversion-free with both f_1 and f_2. The problem is further illustrated in Q_2, where there are three components f_1, f_3, f_2 and any unification between f_1, f_3 results in $S(x, y), S_1(x, y), S_2(x, y), S_3(x, y), S_4(x, y)$, which further unifies with f_2 resulting in $S(x, y), S'(x, y), S_1(x, y), S_2(x, y), S_3(x, y), S_4(x, y)$: both are considered hierarchical unifiers in [Dalvi and Suciu 2007a; Dalvi and Suciu 2006] and the query is classified incorrectly as being in PTIME (the query is a forbidden query). The definition of erasers must insist that the variable order in a component be the same in all usages of that component.