Computing Query Probability with Incidence Algebras
Technical Report UW-CSE-10-03-02
University of Washington

Nilesh Dalvi, Karl Schnaitter and Dan Suciu

Revised: August 24, 2010

Abstract

We describe an algorithm that evaluates queries over probabilistic databases using Mobius' inversion formula in incidence algebras. The queries we consider are unions of conjunctive queries (equivalently: existential, positive First Order sentences), and the probabilistic databases are tuple-independent structures. Our algorithm runs in PTIME on a subset of queries called "safe" queries, and is complete, in the sense that every unsafe query is hard for the class $FP^{\#P}$. The algorithm is very simple and easy to implement in practice, yet it is non-obvious. Mobius' inversion formula, which is in essence inclusion-exclusion, plays a key role for completeness, by allowing the algorithm to compute the probability of some safe queries even when they have some subqueries that are unsafe. We also apply the same lattice-theoretic techniques to analyze an algorithm based on lifted conditioning, and prove that it is incomplete.

1 Introduction

In this paper we show how to use *incidence algebras* to evaluate *unions of conjunctive queries* over probabilistic databases. These queries correspond to the select-project-join-union fragment of the relational algebra, and they also correspond to *existential positive formulas* of First Order Logic. A probabilistic database, also referred to as a *probabilistic structure*, is a pair $(\mathbf{A}, P)$ where $\mathbf{A} = (A, R_1^A, \ldots, R_k^A)$ is first order structure over vocabulary $R_1, \ldots, R_k$, and P is a function that associates to each tuple t in $\mathbf{A}$ a number $P(t) \in [0, 1]$. A probabilistic structure defines a probability distribution on the set of substructures $\mathbf{B}$ of $\mathbf{A}$ by:

$$P_{\mathbf{A}}(\mathbf{B}) \quad = \quad \prod_{i=1}^{k}\Big(\prod_{t \in R_i^B} P(t) \times \prod_{t \in R_i^A - R_i^B} (1 - P(t))\Big) \tag{1}$$

We describe a simple, yet quite non-obvious algorithm for computing the probability of an existential, positive FO sentence Φ, $P_{\mathbf{A}}(\Phi)^1$, based on Mobius' inversion formula in incidence algebras. The algorithm runs in polynomial time in the size of $\mathbf{A}$. The algorithm only applies to certain sentences, called *safe* sentences, and is sound and complete in the following way. It is sound, in that it computes correctly the probability for each safe sentence, and it is complete in that, for every fixed unsafe sentence Φ, the data complexity of computing Φ is $FP^{\#P}$-hard. This establishes a dichotomy for the complexity of unions of conjunctive queries over probabilistic structures. The algorithm is more general than, and significantly simpler than a previous algorithm for conjunctive sentences [5].

The existence of $FP^{\#P}$-hard queries on probabilistic structures was observed by Grädel et al. [8] in the context of query reliability. In the following years, several studies [4, 6, 12, 11], sought to identify classes of tractable queries. These works provided conditions for tractability only for conjunctive queries without self-joins. The only exception is [5], which considers conjunctive queries with self-joins. 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 [12].

Our results have applications to probabilistic inference on positive Boolean expressions [7]. For every tuple t in a structure $\mathbf{A}$, let X_t be a distinct Boolean variable. Every existential positive FO sentence Φ defines a positive DNF Boolean expression over the variables X_t, sometimes called *lineage expression*, whose probability is the same as $P_{\mathbf{A}}(\Phi)$. Our result can be used to classify the complexity of computing the probability of Positive DNF formulas defined by a fixed sentence Φ. For example, the two sentences[2]

$$\begin{aligned}\Phi_1 &= R(x), S(x,y) \vee S(x,y), T(y) \vee R(x), T(y)\\ \Phi_2 &= R(x), S(x,y) \vee S(x,y), T(y)\end{aligned}$$

define two classes of positive Boolean DNF expressions (lineages):

$$\begin{aligned}F_1 &= \bigvee_{a\in R,(a,b)\in S} X_a Y_{a,b} \vee \bigvee_{(a,b)\in S, b\in T} Y_{a,b}, Z_b \vee \bigvee_{a\in R, b\in S} X_a Y_b\\ F_2 &= \bigvee_{a\in R,(a,b)\in S} X_a Y_{a,b} \vee \bigvee_{(a,b)\in S, b\in T} Y_{a,b}, Z_b\end{aligned}$$

Our result implies that, for each such class of Boolean formulas, either all formulas in that class can be evaluated in PTIME in the size of the formula, or the complexity for that class is hard for $FP^{\#P}$; e.g. F_1 can be evaluated in PTIME using our algorithm, while F_2 is hard.

The PTIME algorithm we present here relies in a critical way on an interesting connection between existential positive FO sentences and incidence algebras [17]. By using the Mobius inversion formula in incidence algebras we resolve a major difficulty

[1]This is the *marginal probability* $P_{\mathbf{A}}(\Phi) = \sum_{\mathbf{B}:\mathbf{B}\models\Phi} P_{\mathbf{A}}(\mathbf{B})$.

[2]We omit quantifiers and drop the conjunct they are clear from the context, e.g. $\Phi_2 = \exists x \exists y (R(x) \wedge S(x,y) \vee S(x,y) \wedge T(y))$.

of the evaluation problem: a sentence that is in PTIME may have a subexpression that is hard. This is illustrated by Φ_1 above, which is in PTIME, but has Φ_2 as a subexpression, which is hard; to evaluate Φ_1 one must avoid trying to evaluate Φ_2. Our solution is to express $P(\Phi)$ using Mobius' inversion formula: subexpressions of Φ that have a Mobius value of zero do not contribute to $P(\Phi)$, and this allows us to compute $P(\Phi)$ without computing its hard subexpressions. The Mobius inversion formula corresponds to the inclusion/exclusion principle, which is ubiquitous in probabilistic inference: the connection between the two in the context of probabilistic inference has already been recognized in [9]. However, to the best of our knowledge, ours is the first application that exploits the full power of Mobius inversion to remove hard subexpressions from a computation of probability.

Another distinguishing, and quite non-obvious aspect of our approach is that we apply our algorithm on the CNF, rather than the more commonly used DNF representation of existential, positive FO sentences. This departure from the common representation of existential, positive FO is necessary in order to handle correctly existential quantifiers.

We call sentences on which our algorithm works *safe*; those on which the algorithm fails we call *unsafe*. We prove a theorem stating that the evaluation problem of a safe query is in PTIME, and of an unsafe query is hard for $FP^{\#P}$: this establishes both the completeness of our algorithm and a dichotomy of all existential, positive FO sentences. The proof of the theorem is in two steps. First, we define a simple class of sentences called *forbidden sentences*, where each atom has at most two variables, and a set of simple *rewrite rules* on existential, positive FO sentences; we prove that the safe sentences can be characterized as those that cannot be rewritten into a forbidden sentence. Second, we prove that every forbidden sentence is hard for $FP^{\#P}$, using a direct, and rather difficult proof which we include in [3]. Together, these two results prove that every unsafe sentence is hard for $FP^{\#P}$, establishing the dichotomy. Notice that our characterization of safe queries is reminiscent of minors in graph theory. There, a graph H is called a minor of a graph G if H can be obtained from G through a sequence of edge contractions. "Being a minor of" defines a partial order on graphs: Robertson and Seymour's celebrated result states that any minor-closed family is characterized by a finite set of forbidden minors. Our characterization of safe queries is also done in terms of forbidden minors, however the order relation is more complex and the set of forbidden minors is infinite.

In the last part of the paper, we make a strong claim: that using Mobius' inversion formula is a *necessary* technique for completeness. Today's approaches to general probabilistic inference for Boolean expressions rely on combining (using some advanced heuristics) a few basic techniques: independence, disjointness, and conditioning. In conditioning, one chooses a Boolean variable X, then computes $P(F) = P(F \mid X)P(X) + P(F \mid \neg X)(1 - P(X))$. We extended these techniques to unions of conjunctive queries, an approach that is generally known as *lifted inference* [13, 16, 15] and given a PTIME algorithm based on these three techniques. The algorithm performs conditioning on subformulas of Φ instead of Boolean variables. We prove that this algorithm is not complete, by showing a formula Φ (Fig. 2) that is computable in PTIME, but for which it is not possible to compute using lifted inference that combines conditioning, independence, and disjointness on subformulas. On the other hand, we note that conditioning has certain practical advantages that are lost by Mobius' inversion formula: by repeated conditioning on Boolean variables, one can construct a Free Binary Decision Diagram [18], which has further applications beyond probabilistic inference. There seems to be no procedure to convert Mobius' inversion formula into FBDDs; in fact, we conjecture that the formula in Fig. 2 does not have an FBDD whose size is polynomial in that of the input structure.

Finally, we mention that a different way to define classes of Boolean formulas has been studied in the context of the *constraint satisfaction problem* (CSP). Creignou et al. [2, 1] showed that the counting version of the CSP problem has a dichotomy into PTIME and $FP^{\#P}$-hard. These results are orthogonal to ours: they define the class of formulas by specifying the set of Boolean operators, such as and/or/not/majority/parity etc, and do not restrict the shape of the Boolean formula otherwise. As a consequence, the only class where counting is in PTIME is defined by affine operators: all classes of monotone formulas are hard. In contrast, in our classification there exist classes of formulas that are in PTIME, for example the class defined by Φ_1 above.

2 Background and Overview

Prior Results A very simple PTIME algorithm for conjunctive queries without self-joins is discussed in [4, 6]. When the conjunctive query is connected, the algorithm chooses a variable that occurs in all atoms (called a *root variable*) and projects it out, computing recursively the probabilities of the sub-queries; if no root variable exists, then the query is $FP^{\#P}$-hard. When the conjunctive query is disconnected, then the algorithm computes the probabilities of the connected components, then multiples them. Thus, the algorithm alternates between two steps, called independent projection, and independent join. For example, consider the conjunctive query[3]:

$$\varphi \quad = \quad R(x,y), S(x,z)$$

The algorithm computes its probability by performing the following steps:

$$\begin{aligned}
P(\varphi) &= 1-\prod_{a\in A}(1-P(R(a,y),S(a,z))) \\
P(R(a,y),S(a,z)) &= P(R(a,y))\cdot P(S(a,z)) \\
P(R(a,y)) &= 1-\prod_{b\in A}(1-P(R(a,b))) \\
P(S(a,z)) &= 1-\prod_{c\in A}(1-P(S(a,c)))
\end{aligned}$$

The first line projects out the root variable x, where A is the active domain of the probabilistic structure: it is based on fact that, in $\varphi \equiv \bigvee_{a\in A}(R(a,y),S(a,z))$, the sub-queries $R(a,y), S(a,z)$ are independent for distinct values of the constant a. The

[3]All queries are Boolean and quantifiers are dropped; in complete notation, φ is $\exists x.\exists y.\exists z.R(x,y),S(x,z)$.

second line applies independent join; and the third and fourth lines apply independent project again.

This simple algorithm, however, cannot be applied to a query with self-joins because both the projection and the join step are incorrect. For a simple example, consider $R(x, y), R(y, z)$. Here y is a root variable, but the queries $R(x, a), R(a, z)$ and $R(x, b), R(b, z)$ are dependent (both depend on $R(a, b)$ and $R(b, a)$). Hence, it is not possible to do an independent projection on y. In fact, this query is $FP^{\#P}$-hard.

Queries with self-joins were analyzed in [5] based on the notion of an *inversion*. In a restricted form, an inversion consists of two atoms, over the same relational symbol, and two positions in those atoms, such that the first position contains a root variable in the first atom and a non-root variable in the second atom, and the second position contains a non-root / root pair of variables. In our example above, the atoms $R(x, y)$ and $R(y, z)$ and the positions 1 and 2 form an inversion: position 1 has variables x and y (non-root / root) and position 2 has variables y and z (root / non-root). The paper describes a first PTIME algorithm for queries without inversions, by expressing its probability in terms of several sums, each of which can be reduced to a polynomial size expression. Then, the paper notices that some queries with inversion can also be computed in polynomial time, and describes a second PTIME algorithm that uses one sum (called *eraser*) to cancel the effect of a another, exponentially sized sum. The algorithm succeeds if it can erase all exponentially sized sums (corresponding to sub-queries with inversions).

Our approach The algorithm that we describe in this paper is both more general (it applies to unions of conjunctive queries), and significantly simpler than either of the two algorithms in [5]. We illustrate it here on a conjunctive query with a self-join (S occurs twice):

$$\varphi \quad = \quad R(x_1), S(x_1, y_1), S(x_2, y_2), T(x_2)$$

Our algorithm starts by applying the inclusion-exclusion formula:

$$\begin{aligned} & P(R(x_1), S(x_1, y_1), S(x_2, y_2), T(x_2)) = \\ & \qquad P(R(x_1), S(x_1, y_1)) + P(S(x_2, y_2), T(y_2)) \\ & \qquad -P(R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(x_2)) \end{aligned}$$

This is the dual of the more popular inclusion-exclusion formula for disjunctions; we describe it formally in the framework of incidence algebras in Sec. 3. The first two queries are without self-joins and can be evaluated as before. To evaluate the query on the last line, we simultaneously project out both variables x_1, x_2, writing the query as:

$$\psi \quad = \quad \bigvee_{a \in A} (R(a), S(a, y_1) \vee S(a, y_2), T(a))$$

The variables x_1, x_2 are chosen because they satisfy the following conditions: they occur in all atoms, and for the atoms with the same relation name (S in our case) they occur in the same position. We call such a set of variables *separator variables* (Sec. 4). As a consequence, sub-queries $R(a), S(a, y_1) \vee S(a, y_2), T(a)$ corresponding to distinct constants a are independent. We use this independence, then rewrite the sub-query into CNF and apply the inclusion/exclusion formula again:

$$
\begin{aligned}
&P(\psi) = 1 - \prod_{a \in A}(1 - P(R(a), S(a, y_1) \vee S(a, y_2), T(a))) \\
&R(a), S(a, y_1) \vee S(a, y_2), T(a) \equiv (R(a) \vee T(a)) \wedge S(a, y) \\
&P((R(a) \vee T(a)) \wedge S(a, y)) \\
&\quad = \quad P(R(a) \vee T(a)) + P(S(a, y)) - P(R(a) \vee T(a) \vee S(a, y)) \\
&\quad = \quad P(R(a)) + P(T(a)) - P(R(a)) \cdot P(T(a)) \\
&\qquad\quad -1 + (1 - P(R(a)))(1 - P(T(a))) \prod_{b \in A}(1 - P(S(a, b)))
\end{aligned}
$$

In summary, the algorithm alternates between applying the inclusion/exclusion formula, and performing a simultaneous projection on separator variables: when no separator variables exists, then the query is $FP^{\#P}$-hard. The two steps can be seen as generalizations of the independent join, and the independent projection for conjunctive queries without self-joins.

Ranking Before running the algorithm, a rewriting of the query is necessary. Consider $R(x, y), R(y, x)$: it has no separator variable because neither x nor y occurs in both atoms on the same position. After a simple rewriting, however, the query can be evaluated by our algorithm: partition the relation $R(x, y)$ into three sets, according to $x < y$, $x = y$, $x > y$, call them $R^<, R^=, R^>$, and rewrite the query as $R^<(x, y), R^>(y, x) \vee R^=(z)$. Now x, z is a separator, because the three relational symbols are distinct. We call this rewriting *ranking* (Sec. 5). It needs to be done only once, before running the algorithm, since all sub-queries of a ranked queries are ranked. A similar but more general rewriting called *coverage* was introduced in [5]: ranking corresponds to the canonical coverage.

Incidence Algebras An immediate consequence of using the inclusion-exclusion formula is that sub-queries that happen to cancel out do not have to be evaluated. This turns out to be a fundamental property of the algorithm that allows it to be complete since, as we have explained, some queries are in PTIME but may have sub-queries that are hard. This cancellation is described by the Mobius inversion formula, which groups equal terms in the inclusion-exclusion expansion under coefficients called the *Mobius function*. Using this notion, it is easy to state when a query is PTIME: this happens if and only if all its sub-queries that have a non-zero Mobius function are in PTIME. Thus, while the algorithm itself could be described without any reference to the Mobius inversion formula, by simply using inclusion-exclusion, the Mobius function gives a key insight into what the algorithm does: it recurses only on sub-queries whose Mobius function is non-zero. In fact, we prove the following result (Theorem 6.6): for every finite lattice, there exists a query whose sub-queries generate precisely that lattice, such that all sub-queries are in PTIME except that corresponding to the bottom of the lattice. Thus, the query is in PTIME iff the Mobius function of the lattice bottom is zero. In other words, any formulation of the algorithm must identify, in some way, the elements with a zero Mobius function in an arbitrary lattice: queries are as general as any lattice. For that reason we prefer to expose the Mobius function in the algorithm rather than hide it under the inclusion/exclusion formula.

Lifted Inference At a deeper level, lattices and their associated Mobius function help us understand the limitations of alternative query evaluation algorithms. In Sec. 7 we study an evaluation algorithm based on lifted conditioning and disjointness. We show that conditioning is equivalent to replacing the lattice of sub-queries with a certain sub-lattice. By repeated conditioning one it is sometimes possible to simplify the lattice sufficiently to remove all hard sub-queries whose Mobius function is zero. However, we given an example of a lattice with 9 elements (Fig 2) whose bottom element has the Mobius function equal to zero, but where no conditioning can further restrict the lattice. Thus, the algorithm based on lifted conditioning makes no progress on this lattice, and cannot evaluate the corresponding query. By contrast, our algorithm based on Mobius' inversion formula will easily evaluate the query by skipping the bottom element (since its Mobius function is zero). Thus, our new algorithm based on Mobius' inversion formula is more general than existing techniques based on lifted inference. Finally, we comment on the implications for the completeness of the algorithm in [5].

In the rest of the paper we will refer to conjunctive queries and unions of conjunctive queries as conjunctive sentences, and existential positive FO sentences (or just positive FO sentences) respectively.

3 Existential Positive FO and Incidence Algebras

We describe here the connection between positive FO and incidence algebras. We start with basic notations.

3.1 Existential Positive FO

Fix a vocabulary $\bar{R} = \{R_1, R_2, \ldots\}$. A *conjunctive* sentence φ is a first-order logical formula obtained from positive relational atoms using $\wedge$ and $\exists$:

$$\varphi = \exists \bar{x}.(r_1 \wedge \ldots \wedge r_k) \tag{2}$$

We allow the use of constants. $Var(\varphi) = \bar{x}$ denotes the set of variables in φ, and $Atoms(\varphi) = \{r_1, \ldots, r_k\}$ the set of atoms. Consider the undirected graph where the nodes are $Atoms(\varphi)$ and edges are pairs (r_i, r_j) s.t. r_i, r_j have a common variable. A *component* of φ is a connected component in this graph. Each conjunctive sentence φ can be written as:

$$\varphi = \gamma_1 \wedge \ldots \wedge \gamma_p$$

where each γ_i is a component; in particular, γ_i and γ_j do not share any common variables, when $i \neq j$.

A *disjunctive* sentence is an expression of the form:

$$\varphi' = \gamma'_1 \vee \ldots \vee \gamma'_q$$

where each γ'_i is a single component.

An *existential, positive* sentence Φ is obtained from positive atoms using $\wedge$, $\exists$ and $\vee$; we will refer to it briefly as *positive sentence*. We write a positive sentence either in DNF or in CNF:

$$\Phi = \varphi_1 \vee \ldots \vee \varphi_m \tag{3}$$

$$\Phi = \varphi'_1 \wedge \ldots \wedge \varphi'_M \tag{4}$$

where φ_i are conjunctive sentences in DNF (3), and φ'_i are disjunctive sentences in CNF (4). The DNF can be rewritten into the CNF by:

$$\Phi = \bigvee_{i=1,m} \bigwedge_{j=1,p_i} \gamma_{ij} = \bigwedge_f \bigvee_i \gamma_{if(i)}$$

where f ranges over functions with domain $[m]$ s.t. $\forall i \in [m]$, $f(i) \in [p_i]$. This rewriting can increase the size of the sentence exponentially[4]. Finally, we will often drop $\exists$ and $\wedge$ when clear from the context.

A classic result by Sagiv and Yannakakis [14] gives a necessary and sufficient condition for a logical implication of positive sentences written in DNF: if $\Phi = \bigvee_i \varphi_i$ and $\Phi' = \bigvee_j \varphi'_j$, then:

$$\Phi \Rightarrow \Phi' \quad \text{iff} \quad \forall i.\exists j.\varphi_i \Rightarrow \varphi'_j \tag{5}$$

No analogous property holds for CNF: $R(x,a), S(a,z)$ logically implies $R(x,y), S(y,z)$ (where a is a constant), but $R(x,a) \not\Rightarrow R(x,y), S(y,z)$ and $S(a,z) \not\Rightarrow R(x,y), S(y,z)$. We show in Sec. 5 a rewriting technique that enforces such a property.

3.2 Incidence Algebras

Next, we review the basic notions in incidence algebras following Stanley [17]. A finite *lattice* is a finite ordered set $(\hat{L}, \leq)$ where every two elements $u, v \in \hat{L}$ have a least upper bound $u \vee v$ and a greatest lower bound $u \wedge v$, usually called *join* and *meet*. Since it is finite, it has a minimum and a maximum element, denoted $\hat{0}, \hat{1}$. We denote $L = \hat{L} - \{\hat{1}\}$ (departing from [17], where L denotes $\hat{L} - \{\hat{0}, \hat{1}\}$). L is a meet-semilattice. The *incidence algebra* $I(\hat{L})$ is the algebra[5] of real (or complex) matrices t of dimension $|\hat{L}| \times |\hat{L}|$, where the only non-zero elements t_{uv} (denoted $t(u,v)$) are for $u \leq v$; alternatively, a matrix can be seen as a linear function $t : \mathbf{R}^{\hat{L}} \to \mathbf{R}^{\hat{L}}$. Two matrices are of key importance in incidence algebras: $\zeta_{\hat{L}} \in I(\hat{L})$, defined as $\zeta_{\hat{L}}(u,v) = 1$ forall $u \leq v$; and its inverse, the Mobius function $\mu_{\hat{L}} : \{(u,v) \mid u, v \in \hat{L}, u \leq v\} \to Z$, defined by:

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

[4]Our algorithm runs in PTIME *data complexity*; we do not address the expression complexity in this paper.

[5]An *algebra* is a vector space plus a multiplication operation [17].

We drop the subscript and write μ when $\hat{L}$ is clear from the context.

The fact that μ is the inverse of ζ means the following thing. Let $f : \hat{L} \to \mathbf{R}$ be a real function defined on the lattice. Define a new function g as $g(v) = \sum_{u \leq v} f(u)$. Then $f(v) = \sum_{u \leq v} \mu(u,v) g(u)$. This is called Mobius' inversion formula, and is a key piece of our algorithm. Note that it simply expresses the fact that $g = \zeta(f)$ implies $f = \mu(g)$.

3.3 Their Connection

A *labeled lattice* is a triple $\hat{\mathbf{L}} = (\hat{L}, \leq, \lambda)$ where $(\hat{L}, \leq)$ is a lattice and λ assigns to each element in $u \in \hat{L}$ a positive FO sentence $\lambda(u)$ s.t. $\lambda(u) \equiv \lambda(v)$ iff $u = v$.

Definition 3.1 *A D-lattice is a labeled lattice $\hat{\mathbf{L}}$ where, forall $u \neq \hat{1}$, $\lambda(u)$ is conjunctive, forall u, v, $\lambda(u \wedge v)$ is logically equivalent to $\lambda(u) \wedge \lambda(v)$, and $\lambda(\hat{1}) \equiv \bigvee_{u < \hat{1}} \lambda(u)$.*

A C-lattice is a labeled lattice $\hat{\mathbf{L}}$ where, forall $u \neq \hat{1}$, $\lambda(u)$ is disjunctive, forall u, v, $\lambda(u \wedge v)$ is logically equivalent to $\lambda(u) \vee \lambda(v)$, and $\lambda(\hat{1}) = \bigwedge_{u < \hat{1}} \lambda(u)$.

In a D-lattice, $u \leq v$ iff $\lambda(u) \Rightarrow \lambda(v)$. This is because $\lambda(u) = \lambda(u \wedge v)$ is logically equivalent to $\lambda(u) \wedge \lambda(v)$. Similarly, in a C-lattice, $u \leq v$ iff $\lambda(v) \Rightarrow \lambda(u)$. If $\hat{\mathbf{L}}$ is a D- or C-lattice, we say $\hat{\mathbf{L}}$ *represents* $\Phi = \lambda(\hat{1})$.

Proposition 3.2 (Inversion formula for positive FO) *Fix a probabilistic structure $(\mathbf{A}, P)$ and a positive sentence Φ; denote $P_{\mathbf{A}}$ as P. Let $\hat{\mathbf{L}}$ be either a D-lattice or a C-lattice representing Φ. Then:*

$$P(\Phi) = P(\lambda(\hat{1})) \quad = \quad -\sum_{v < \hat{1}} \mu_L(v, \hat{1}) P(\lambda(v)) \tag{6}$$

Proof: The proof for the D-lattice is from [17]. Denote $f(u) = P(\lambda(u) \wedge \neg(\bigvee_{v<u} \lambda(v)))$. Then:

$$P(\lambda(u)) = \sum_{v \leq u} f(v) \quad \Rightarrow \quad f(u) = \sum_{v \leq u} \mu(v,u) P(\lambda(v))$$

The claim follows by setting $u = \hat{1}$ and noting $f(\hat{1}) = 0$. For a C-lattice, write $\lambda'(u) = \neg\lambda(u)$. Then $P(\lambda(\hat{1})) = 1 - P(\lambda'(\hat{1})) = 1 + \sum_{v < \hat{1}} \mu(v, \hat{1}) P(\lambda'(v))$ and the claim follows from the fact that $\sum_{v \in \hat{L}} \mu(v, \hat{1}) = 0$. □

The proposition generalizes the well known inclusion/exclusion formula (for D-lattices), and its less well known dual (for C-lattices):

$$\begin{aligned}
P(a \vee b \vee c) &= P(a) + P(b) + P(c) \\
&\quad -P(a \wedge b) - P(a \wedge c) - P(b \wedge c) + P(a \wedge b \wedge c) \\
P(a \wedge b \wedge c) &= P(a) + P(b) + P(c) \\
&\quad -P(a \vee b) - P(a \vee c) - P(b \vee c) + P(a \vee b \vee c)
\end{aligned}$$

We show how to construct a canonical D-lattice, $\hat{\mathbf{L}}_D(\Phi)$ that represents a positive sentence Φ. Start from the DNF in Eq.(3), and for each subset $s \subseteq [m]$ denote $\varphi_s =$

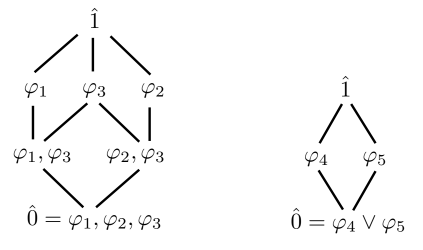


Figure 1: The D-lattice (a) and the C-lattice (b) for Φ (Ex. 3.3).

$\bigwedge_{i \in s} \varphi_i$. Let $\hat{L}$ be the set of these conjunctive sentences, up to logical equivalence, and ordered by logical implication (hence, $|L| \leq 2^m$). Label each element $u \in \hat{L}$, $u \neq \hat{1}$, with its corresponding φ_s (choose any, if there are multiple equivalent ones), and label $\hat{1}$ with $\bigvee_{s \neq \emptyset} \varphi_s$ ($\equiv \Phi$). We denote the resulting D-lattice $\hat{\mathbf{L}}_D(\Phi)$. Similarly, $\hat{\mathbf{L}}_C(\Phi)$ is the C-lattice that represents Φ, obtained from the CNF of Φ in Eq.(4), setting $\varphi'_s = \bigvee_{i \in s} \varphi'_i$.

The first main technique of our algorithm is this. Given Φ, compute its C-lattice, then use Eq.(6) to compute $P(\Phi)$; we explain later why we use the C-lattice instead of the D-lattice. It remains to compute the probability of disjunctive sentences $P(\lambda(u))$: we show this in the next section. The power of this technique comes from the fact that, whenever $\mu(u, \hat{1}) = 0$, then we do not need to compute the corresponding $P(\lambda(u))$. As we explain in Sec. 7 this is strictly more powerful than the current techniques used in probabilistic inference, such as lifted conditioning.

Example 3.3 Consider the following positive sentence:

$$\begin{aligned}
\Phi &= R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(y_2) \vee R(x_3), T(y_3) \\
&= \varphi_1 \vee \varphi_2 \vee \varphi_3
\end{aligned}$$

The Hasse diagram of the D-lattice $\mathbf{L}_D(\Phi)$ is shown in Fig. 1 (a). There are eight subsets $s \subseteq [3]$, but only seven up to logical equivalence, because[6] $\varphi_1, \varphi_2 \equiv \varphi_1, \varphi_2, \varphi_3$. The values of the Mobius function are, from top to bottom: $1, -1, -1, -1, 1, 1, 0$, hence the inversion formula is[7]:

$$P(\Phi) = P(\varphi_1) + P(\varphi_2) + P(\varphi_3) - P(\varphi_1 \varphi_3) - P(\varphi_2 \varphi_3)$$

The Hasse diagram of the C-lattice $\mathbf{L}_C(\Phi)$ is shown in Fig. 1 (b). To see this, first

[6]There exists a homomorphism $\varphi_1, \varphi_2, \varphi_3 \rightarrow \varphi_1, \varphi_2$ that maps $R(x_3)$ to $R(x_1)$ and $T(y_3)$ to $T(y_2)$.

[7]One can arrive at the same expression by using inclusion-exclusion instead of Mobius' inversion formula, and noting that $\varphi_1, \varphi_2 \equiv \varphi_1, \varphi_2 \varphi_3$, hence these two terms cancel out in the inclusion-exclusion expression.

express Φ in CNF:

$$\begin{aligned}
\Phi &= (R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(y_2) \vee R(x_3)) \wedge \\
&\quad (R(x_4), S(x_4, y_4) \vee S(x_5, y_5), T(y_5) \vee T(y_6)) \\
&= (R(x_3) \vee S(x_2, y_2), T(y_2)) \wedge (R(x_4), S(x_4, y_4) \vee T(y_6)) \\
&= \varphi_4 \wedge \varphi_5
\end{aligned}$$

Note that $\hat{0}$ is labeled with $\varphi_4 \vee \varphi_5 \equiv R(x_3) \vee T(y_6)$. The inversion formula here is:

$$P(\Phi) = P(\varphi_4) + P(\varphi_5) - P(\varphi_4 \vee \varphi_5)$$

where $\varphi_4 \vee \varphi_5 \equiv R(x_3) \vee T(y_6)$.

3.4 Minimization

By *minimizing* a conjunctive sentence φ we mean replacing it with an equivalent sentence φ_0 that has the smallest number of atoms. A disjunctive sentence $\Phi = \bigvee \gamma_i$ is *minimized* if every conjunctive sentence is minimized and there is no homomorphism $\varphi_i \Rightarrow \varphi_j$ for $i \neq j$. If such a homomorphism exists, then we call φ_j redundant: clearly we can remove it from the expression Φ without affecting its semantics.

For the purpose of D-lattices, it doesn't matter if we minimize the sentence or not: if the sentence is not minimized, then one can show that all lattice elements corresponding to redundant terms have the Mobius function equal to zero. More precisely, any two D-lattices that represent the same sentence have the same set of elements with a non-zero Mobius function: we state this fact precisely in the remainder of this section. A similar fact does not hold in general for C-lattices, but it holds over ranked structures (Sec. 5.2).

An element u in a lattice *covers* v if $u > v$ and there is no w s.t. $u > w > v$. An *atom*[8] is an element that covers $\hat{0}$; a *co-atom* is an element covered by $\hat{1}$. An element u is called *co-atomic* if it is a meet of coatoms. Let L_0 denote the set of co-atomic elements: L_0 is a meet semilattice, and $\hat{L}_0 = L_0 \cup \{\hat{1}\}$ is a lattice. The proof is Appendix I.

Proposition 3.4 *(1) If $u \in L$ and $\mu_{\hat{L}}(u, \hat{1}) \neq 0$ then u is co-atomic. (2) Forall $u \in L_0$, $\mu_{\hat{L}}(u, \hat{1}) = \mu_{\hat{L}_0}(u, \hat{1})$.*

Let $\hat{\mathbf{L}}$ and $\hat{\mathbf{L}}'$ be D-lattices representing the sentences Φ and Φ'. If $\Phi \equiv \Phi'$, then $\hat{\mathbf{L}}$ and $\hat{\mathbf{L}}'$ have the same co-atoms, up to logical equivalence. Indeed, we can write Φ as the disjunction of co-atom labels in $\hat{\mathbf{L}}$, and one co-atom cannot imply another. Thus, by applying Eq.(5) in both directions, we get a one-to-one correspondence between the co-atoms of $\hat{\mathbf{L}}$ and $\hat{\mathbf{L}}'$, indicating logical equivalence. It follows from Prop. 3.4 that, when D-lattices represent equivalent formulas, the set of labels $\lambda(u)$ where $\mu(u, \hat{1}) \neq 0$ are equivalent. Thus, an algorithm that inspects only these labels is independent of the particular representation of a sentence.

[8]Not to be confused with a relational atom r_i in (2).

A similar property fails on C-lattices, because Eq.(5) does not extend to CNF. For example, $\Phi = R(x,a), S(a,z)$ and $\Phi' = R(x,a), S(a,z), R(x',y'), S(y',z')$ are logically equivalent, but have different co-atoms. The co-atoms of Φ are $R(x,a)$ and $S(a,z)$ (the C-lattice is V-shaped, as in Fig. 1 (b)), and the co-atoms of Φ' are $R(x,a)$, $(R(x',y'), S(y',z'))$, and $S(a,z)$ (the C-lattice is W-shaped, as in Fig. 1 (a)).

4 Independence and Separators

Next, we show how to compute the probability of a disjunctive sentence $\bigvee_i \gamma_i$; this is the second technique used in our algorithm, and consists of eliminating, simultaneously, one existential variable from each γ_i, by exploiting independence.

Let φ be a conjunctive sentence. A *valuation* h is a substitution of its variables with constants; $h(\varphi)$, is a set of ground tuples. We call two conjunctive sentences φ_1, φ_2 *tuple-independent* if for all valuations h_1, h_2, we have $h_1(\varphi_1) \cap h_2(\varphi_2) = \emptyset$. Two positive sentences Φ, Φ' are *tuple-independent* if, after expressing them in DNF, $\Phi = \bigvee_i \varphi_i$, $\Phi' = \bigvee_j \varphi'_j$, all pairs φ_i, φ'_j are independent.

Let $\Phi_1, \ldots, \Phi_m$ be positive sentences s.t. any two are tuple-independent. Then:

$$P(\bigvee_i \Phi_i) \quad = \quad 1 - \prod_i (1 - P(\Phi_i))$$

This is because the m lineage expressions for Φ_i depend on disjoint sets of Boolean variables, and therefore they are independent probabilistic events. In other words, tuple-independence is a sufficient condition for independence in the probabilistic sense. Although it is only a sufficient condition, we will abbreviate tuple-independence with independence in this section.

Let φ be a positive sentence, $V = \{x_1, \ldots, x_m\} \subseteq Var(\varphi)$, and a a constant. Denote $\varphi[a/V] = \varphi[a/x_1, \ldots, a/x_m]$ (all variables in V are substituted with a).

Definition 4.1 *Let $\varphi = \bigvee_{i=1,m} \gamma_i$ be a disjunctive sentence. A* separator *is a set of variables $V = \{x_1, \ldots, x_m\}$, $x_i \in Var(\gamma_i)$, such that for all $a \neq b$, $\varphi[a/V]$, $\varphi[b/V]$ are independent.*

Proposition 4.2 *Let φ be a disjunctive sentence with a separator V, and $(\mathbf{A}, P)$ a probabilistic structure with active domain D. Then:*

$$P(\varphi) \quad = \quad 1 - \prod_{a \in D} (1 - P(\varphi[a/V])) \tag{7}$$

The claim follows from the fact that $\varphi \equiv \bigvee_{a \in D} \varphi[a/V]$ on all structures whose active domain is included in D.

In summary, to compute the probability of a disjunctive sentence, we find a separator, then apply Eq.(7): each expression $\varphi[a/V]$ is a positive sentence, simpler than the original one (it has strictly fewer variables in each atom) and we apply again the inversion formula. This technique, by itself, is not complete: we need to "rank" the relations in order to make it complete, as we show in the next section. Before that, we illustrate with an example.

Example 4.3 Consider $\varphi = R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(x_2)$. Here $\{x_1, x_2\}$ is a separator. To see this, note that for any constants $a \neq b$, the sentences $\varphi[a] = R(a), S(a, y_1) \vee S(a, y_2), T(a)$ and $\varphi[b] = R(b), S(b, y_1) \vee S(b, y_2), T(b)$ are independent, because the former only looks at tuples that start with a, while the latter only looks at tuples that start with b.

Consider $\varphi = R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(y_2)$. This sentence has no separator. For example, $\{x_1, x_2\}$ is not a separator because both sentences $\varphi[a]$ and $\varphi[b]$ have the atom $T(y_2)$ in common: if two homomorphisms h_1, h_2 map y_2 to some constant c, then $T(c) \in h_1(\varphi[a]) \cap h_2(\varphi[b])$, hence they are dependent. The set $\{x_1, y_2\}$ is also not a separator, because $\varphi[a]$ contains the atom $S(a, y_1)$, $\varphi[b]$ contains the atom $S(x_2, b)$, and these two can be mapped to the common ground tuple $S(a, b)$.

We end with a necessary condition for V to be a separator.

Definition 4.4 *If γ is a component, a variable of γ is called a* root variable *if it occurs in all atoms of γ.*

Note that components do not necessarily have root variables, e.g., $R(x), S(x, y), T(y)$. We have:

Proposition 4.5 *If V is a separator of $\bigvee_i \gamma_i$, then each separator variable $x_i \in Var(\gamma_i)$ is a root variable for γ_i.*

The claim follows from the fact that, if r is any atom in φ_i that does not contain x_i: then r is unchanged in $\gamma_i[a]$ and in $\gamma_i[b]$, hence they are not independent.

5 Ranking

In this section, we define a simple restriction on all formulas and structures that simplifies our later analysis: we require that, in each relation, the attributes may be strictly ordered $A_1 < A_2 < \ldots$ We show how to alter any positive sentence and probabilistic structure to satisfy this constraint, without changing the sentence probability. This is a necessary preprocessing step for our algorithm to work, and a very convenient technique in the proofs.

5.1 Ranked Structures

Throughout this section, we use $<$ to denote a total order on the active domain of a probabilistic structure (such an order always exists). In our examples, we assume that $<$ is the natural ordering on integers, but the order may be chosen arbitrarily in general.

Definition 5.1 *A relation instance R is* ranked *if every tuple $R(a_1, \ldots, a_k)$ is such that $a_1 < \cdots < a_k$. A probabilistic structure is* ranked *if all its relations are ranked.*

To motivate ranked structures, we observe that the techniques given in previous sections do not directly lead to a complete algorithm. We illustrate by reviewing the example in Sec. 2: $\gamma = R(x, y), R(y, x)$. This component is connected, so we cannot use Mobius inversion to simplify it, and we also cannot apply Eq.(7) because there is no separator: indeed, $\{x\}$ is not a separator because $R(a, y), R(y, a)$ and $R(b, y), R(y, b)$ are not independent (they share the tuple $R(a, b)$), and by symmetry neither is $\{y\}$. However, consider a structure with a unary relation R_{12} and binary relations $R_{1<2}$, $R_{2<1}$ defined as:

$$\begin{array}{ll} R_{12} = \pi_{X_1}(\sigma_{X_1=X_2}(R)) & R_{2<1} = \pi_{X_2X_1}(\sigma_{X_2<X_1}(R)) \\ R_{1<2} = \sigma_{X_1<X_2}(R) & \end{array}$$

Here, we use X_i to refer to the i-th attribute of R. This is a ranked structure: in both relations $R_{1<2}$ and $R_{2<1}$ the first attribute is less than the second. Moreover: $\gamma \equiv R_{12}(z) \vee R_{1<2}(x, y), R_{2<1}(x, y)$ and now $\{z, x\}$ is a separator, because $R_{1<2}(a, y), R_{2<1}(a, y)$ and $R_{1<2}(b, y), R_{2<1}(b, y)$ are independent. Thus, Eq.(7) applies to the formula over the ranked structure, and we can compute the probability of γ in polynomial time.

Definition 5.2 *A positive sentence is in* reduced form *if each atom $R(x_1, \ldots, x_k)$ is such that (a) each x_i is a variable (i.e. not a constant), and (b) the variables $x_1, \ldots, x_k$ are distinct.*

We now prove that the evaluation of any sentence can be reduced to an equivalent sentence over a ranked structure, and we further guarantee that the resulting sentence is in reduced form.

Proposition 5.3 *Let Φ_0 be positive sentence. Then, there exists a sentence Φ in reduced form such that for any structure $\mathbf{A}_0$, one can construct in polynomial time a ranked structure $\mathbf{A}$ such that $P_{\mathbf{A}_0}(\Phi_0) = P_{\mathbf{A}}(\Phi)$.*

Proof: Let $R(X_1, \ldots, X_k)$ be a relation symbol and let ρ be a maximal, consistent conjunction of order predicates involving attributes of R and the constants occurring in Φ_0. Thus, for any pair of attributes names or constants y, z, ρ implies exactly one of $y < z$, $y = z$, $y > z$. Before describing Φ, we show to construct the ranked structure $\mathbf{A}$ from an unranked one $\mathbf{A}_0$. We say X_j is *unbound* if $\rho \not\Rightarrow X_j = c$ for any constant c. The ranked structure will have one symbol R^ρ for every symbol R in the unranked structure and every maximal consistent predicate ρ. The instance $\mathbf{A}$ is computed as $R^\rho = \pi_{\bar{X}}(\sigma_\rho(R))$ where $\bar{X}$ contains one X_j in each class of unbound attributes that are equivalent under ρ, listed in increasing order according to ρ. Clearly $\mathbf{A}$ can be computed in PTIME from $\mathbf{A}_0$.

We show now how to rewrite any positive sentence Φ_0 into an equivalent, reduced sentence Φ over ranked structures s.t. $P_{\mathbf{A}_0}(\Phi_0) = P_{\mathbf{A}}(\Phi)$. We start with a conjunctive sentence $\varphi = r_1, \ldots, r_n$ and let R_i denote the relation symbol of r_i. Consider a maximally consistent predicate ρ_i on the attributes of R_i, for each $i = 1, n$, and let $\rho \equiv \rho_1, \ldots, \rho_n$ be the conjunction. We say that ρ is *consistent* if there is a valuation h such that $h(\varphi) \models \rho$. Given a consistent ρ, divide the variables into equivalence classes of variables that ρ requires to be equal, and choose one representative variable from each class. Let $r_i^{\rho_i}$ be the result of changing $R_i(x_1, \ldots, x_k)$ to $R_i^{\rho_i}(y_1, \ldots, y_m)$, where $y_1, \ldots, y_m$ are chosen as follows. Consider the unbound attribute classes in R_i, in increasing order according to ρ_i. Choose y_p to be the representative of a variable that occurs in the position of an attribute in the p-th class of unbound attributes. This works because the position of any unbound attribute X must have a variable: if there is a constant a, then $h(r_i) \models X = a$ for all valuations h. But $\rho_i \Rightarrow X \neq a$ so this contradicts the assumption that ρ is consistent. Using a similar argument, we can show that each y_i is distinct, so $r_i^{\rho_i}$ is in reduced form. Furthermore, $\varphi \equiv \bigwedge_\rho r_1^{\rho_1}, \ldots, r_n^{\rho_n}$ where the disjunction ranges over all maximal ρ_i such that ρ is consistent. For a positive sentence Φ_0, we apply the above procedure to each conjunctive sentence in the DNF of Φ_0 to yield a sentence in reduced form on the ranked relations R^ρ. $\square$

Example 5.4 Let $\varphi = R(x,a), R(a,x)$. If we define the ranked relations $R_1 = \pi_{X_2}(\sigma_{X_1=a}(R))$, $R_2 = \pi_{X_1}(\sigma_{X_2=a}(R))$, and $R_{12} = \pi_\emptyset(\sigma_{X_1=X_2=a}(R))$, we have $\varphi \equiv R_1(x), R_2(x) \vee R_{12}()$.

Next, consider $\varphi = R(x), S(x,x,y), S(u,v,v)$. Define

$$\begin{aligned} S_{123} &= \pi_{X_1}(\sigma_{X_1=X_2=X_3}(S)) \\ S_{23<1} &= \pi_{X_2X_1}(\sigma_{X_2=X_3<X_1}((S)) \end{aligned}$$

and so on. We can rewrite φ as:

$$\begin{aligned} \varphi \equiv\ & R(x), S_{123}(x) \\ & \vee R(x), S_{12<3}(x,y), S_{1<23}(u,v) \vee R(x), S_{12<3}(x,y), S_{23<1}(v,u) \\ & \vee R(x), S_{3<12}(y,x), S_{1<23}(u,v) \vee R(x), S_{3<12}(y,x), S_{23<1}(v,u) \end{aligned}$$

and note that these relations are ranked.

Thus, when computing $P_{\mathbf{A}}(\Phi)$, we may conveniently assume w.l.o.g. that $\mathbf{A}$ is ranked and Φ is in reduced form. When we replace separator variables with a constant as in Eq.(7), we can easily restore the formula to reduced form. Given a disjunctive sentence φ in reduced form and a separator V, we remove a from $\varphi[a/V]$ as follows. For each relation R, suppose the separator variables occur at position X_i of R. Then we remove all rows from R where $X_i \neq a$, reduce the arity of R by removing column i, and remove $x_i = a$ from all atoms $R(x_1, \ldots, x_k)$ in $\varphi[a/V]$.

We end this section with two applications of ranking. The first shows a homomorphism theorem for CNF sentences.

Proposition 5.5 *Assume all structures to be ranked, and all sentences to be in reduced form.*

- *If φ, φ' are conjunctive sentences, and φ is satisfiable over ranked structures*[9] *then $\varphi \Rightarrow \varphi'$ iff there exists a homomorphism $h : \varphi' \to \varphi$.*
- *Formula (5) holds for positive sentences in DNF.*
- *The dual of (5) holds for positive sentences in CNF:*

$$\bigwedge_i \varphi_i \Rightarrow \bigwedge_j \varphi'_j \quad \textit{iff} \quad \forall j.\exists i.\varphi_i \Rightarrow \varphi'_j$$

[9]Meaning: it is satisfied by at least one (ranked) structure.

Proof:

- Let $V = Var(\varphi)$ and construct a graph on V such that (u, v) is an edge whenever some atom of φ imposes the constraint $u < v$ based on ranking. This graph is acyclic because there exists a valuation h from φ to a ranked structure, which means $h(u) < h(v)$ for all edges (u, v). Let $x_1, \ldots, x_{|V|}$ be a topological ordering of V, i.e., such that $i < j$ for all edges (x_i, x_j). Replace each x_i with i in φ, and let $\mathbf{A}$ be the ranked structure consisting of all atoms in φ after this mapping. Clearly $\mathbf{A} \models \varphi$, hence $\mathbf{A} \models \varphi'$: the latter gives a valuation $\varphi' \to \mathbf{A}$, which, composed with the mapping $i \mapsto x_i$, gives a homomorphism $\varphi' \to \varphi$.
- Standard argument, omitted.
- Assume for contradiction that there exists p, s.t. $\forall i$, it is not the case that $\varphi_i \Rightarrow \varphi'_p$. Let $\mathbf{A}_i$ be a structure s.t. $\mathbf{A}_i \models \varphi_i$ but $\mathbf{A}_i \not\models \varphi_p$. The active domain of $\mathbf{A}_i$ is unconstrained by φ_i because the sentence is in reduced form, and hence contains no constants. Hence we may assume w.l.o.g. that for $i_1 \neq i_2$, the structures $\mathbf{A}_{i_1}$ and $\mathbf{A}_{i_2}$ have disjoint active domains. Define $\mathbf{A} = \bigcup_i \mathbf{A}_i$. Then $\mathbf{A} \models \bigvee \varphi_i$, implying that $\mathbf{A} \models \bigvee \varphi'_j$. In particular, $\mathbf{A} \models \varphi'_p$, hence there exists a valuation $\varphi'_p \to \mathbf{A}$. Since the active domains of each $\mathbf{A}_i$ are disjoint, its image must be contained in a single $\mathbf{A}_i$, contradicting the fact that $\varphi_i \not\Rightarrow \varphi'_p$.

□

The first two items are known to fail for conjunctive sentences with order predicates: for example $R(x, y), R(y, x)$ logically implies $R(x, y), x \leq y$, but there is no homomorphism from the latter to the former. They hold for ranked structures because there is a strict total order on the attributes of each relation. The last item implies the following. If $\hat{\mathbf{L}}$ and $\hat{\mathbf{L}}'$ are two C-lattices representing equivalent sentences, then they have the same co-atoms. In conjunction with Prop. 3.4, this implies that an algorithm that ignores lattice elements where $\mu(u, \hat{1}) = 0$ does not depend on the representation of the positive sentence. This completes our discussion at the end of Sec. 3.

The second result shows how to handle atoms without variables.

Proposition 5.6 *Let γ_0, γ_1 be components in reduced form s.t. $Var(\gamma_0) = \emptyset$, $Var(\gamma_1) \neq \emptyset$. Then γ_0, γ_1 are independent.*

Proof: Note that γ_0 contains a single atom $R()$; if it had two atoms then it is not a component. Since γ_1 is connected, each atom must have at least one variable, hence it cannot have the same relation symbol $R()$. □

Let $\varphi = \bigvee \gamma_i$ be a disjunctive sentence, $\varphi_0 = \bigvee_{i:Var(\gamma_i)=\emptyset} \gamma_i$ and $\varphi_1 = \bigvee_{i:Var(\gamma_i)\neq\emptyset} \gamma_i$. It follows that:

$$P(\varphi) \quad = \quad 1 - (1 - P(\varphi_0))(1 - P(\varphi_1)) \qquad (8)$$

5.2 Finding a Separator

Assuming structures to be ranked, we give here a necessary and sufficient condition for a disjunctive sentence in reduced form to have a separator, which we use both in the algorithm and to prove hardness for $FP^{\#P}$. We need some definitions first.

Let $\varphi = \gamma_1 \vee \ldots \vee \gamma_m$ be a disjunctive sentence, in reduced form. Throughout this section we assume that φ is minimized and that $Var(\gamma_i) \cap Var(\gamma_j) = \emptyset$ for all $i \neq j$ (if not, then rename the variables). Two atoms $r \in Atoms(\gamma_i)$ and $r' \in Atoms(\gamma_j)$ are called *unifiable* if they have the same relational symbol. We may also say r, r' *unify*. It is easy to see that γ_i and γ_j contain two unifiable atoms iff they are not tuple-independent. Two variables x, x' are *unifiable* if there exist two unifiable atoms r, r' such that x occurs in r at the same position that x' occurs in r'. This relationship is reflexive and symmetric. We also say that x, x' are *recursively unifiable* if either x, x' are unifiable, or there exists a variable x'' such that x, x'' and x', x'' are recursively unifiable.

A variable x is *maximal* if it is only recursively unifiable with root variables. Hence all maximal variables are root variables. The following are canonical examples of sentences where each component has a root variable, but there are no maximal variables:

$$\begin{aligned}
&h_0 = R(x_0), S_1(x_0, y_0), T(y_0)\\
&h_1 = R(x_0), S_1(x_0, y_0) \vee S_1(x_1, y_1), T(y_1)\\
&h_2 = R(x_0), S_1(x_0, y_0) \vee S_1(x_1, y_1), S_2(x_1, y_1) \vee S_2(x_2, y_2), T(y_2)\\
&\ldots\\
&h_k = R(x_0), S_1(x_0, y_0) \vee S_1(x_1, y_1), S_2(x_1, y_1) \vee\\
&\qquad \ldots \vee S_{k-1}(x_{k-1}, y_{k-1}), S_k(x_{k-1}, y_{k-1}) \vee (S_k(x_k, y_k), T(y_k)
\end{aligned}$$

In each h_k, $k \geq 1$, the root variables are x_{i-1}, y_i for $i = 1, k-1$, and there are no maximal variables.

Maximality propagates during unification: if x is maximal and x, x' unify, then x' must be maximal because otherwise x would recursively unify with a non-root variable.

Let W_i be the set of maximal variables occurring in γ_i. If an atom in γ_i unifies with an atom in γ_j, then $|W_i| = |W_j|$ because the two atoms contain all maximal variables in each component, and maximality propagates through unification. Since the structures are ranked, for every i there exists a total order on the maximal variables in W_i: $x_{i1} < x_{i2} < \ldots$ The *rank* of a variable $x \in W_i$ is the position where it occurs in this order. The following result gives us a means to find a separator if it exists:

Proposition 5.7 *A disjunctive sentence has a separator iff every component has a maximal variable. In that case, the set comprising maximal variables with rank* 1 *forms a separator.*

Proof: Consider the disjunctive sentence $\varphi = \bigvee_{i=1}^{m} \gamma_i$ and set of variables $V = \{x_1, \ldots, x_m\}$ s.t. $x_i \in Var(\gamma_i)$, $i = 1, m$. It is straightforward to show that V is a separator iff any pair of unifiable atoms have a member of V in the same position. Hence, if V is a separator, then each $x_i \in V$ can only (recursively) unify with another $x_j \in V$. Since x_j is a root variable (Prop. 4.5), each $x_i \in Var(\gamma_i)$ is maximal, as desired.

Now suppose every component has a maximal variable. Choose V such that x_i is the maximal variable in γ_i with rank 1. If two atoms r, r' unify, then they have maximal variables occurring in the same positions. In particular, the first maximal variable has rank 1, and thus is in V. We conclude that V is a separator. $\square$

Algorithm 6.1 Algorithm for Computing $P(\Phi)$
`Input`: Positive sentence Φ in reduced form;
Ranked structure $(\mathbf{A}, p)$ with active domain D
`Output`: $P(\Phi)$

```
1: Function MobiusStep($\Phi$) /* $\Phi$ = positive sentence */
2:   Let $\hat{\mathbf{L}} = \hat{\mathbf{L}}_C(\Phi)$ be a C-lattice representing $\Phi$
3:   Return $\sum_{u<\hat{1}} \mu_{\hat{L}}(u, \hat{1}) *$IndepStep$(\lambda(u))$
4: □
5: Function IndepStep($\varphi$) /* $\varphi = \bigvee_i \gamma_i$ */
6:   Minimize $\varphi$ (Sec. 3.4)
7:   Let $\varphi = \varphi_0 \vee \varphi_1$
8:     where: $\varphi_0 = \bigvee_{i:Var(\gamma_i)=\emptyset} \gamma_i, \varphi_1 = \bigvee_{i:Var(\gamma_i)\neq\emptyset} \gamma_i$
9:   Let $V$ = a separator for $\varphi_1$ (Sec. 5.2)
10:  If (no separator exists) then FAIL (UNSAFE)
11:  Let $p_0 = P(\varphi_0)$
12:  Let $p_1 = 1 - \prod_{a\in D}(1 -$ MobiusStep$(\varphi_1[a/V]))$
13:    /* Note: assume $\varphi_1[a/V]$ is reduced (Sec.5) */
14:  Return $1 - (1 - p_0)(1 - p_1)$.
15: □
```

For a trivial illustration of this result, consider the disjunctive sentence $R(x, y), S(x, y) \vee S(x', y'), T(x', y')$. All variables are root variables, and the sets of maximal variables are $W_1 = \{x, y\}$, $W_2 = \{x', y'\}$. We break the tie by using the ranking: choosing arbitrarily rank 1, we obtain the separator $\{x, x'\}$. (Rank 2 would gives us the separator $\{y, y'\}$). A more interesting example is:

Example 5.8 In φ, not all root variables are maximal:

$$\begin{aligned}\varphi \;=\; & R(z_1, x_1), S(z_1, x_1, y_1) \vee S(z_2, x_2, y_2), T(z_2, y_2) \vee \\ & R(z_3, x_3), T(z_3, y_3)\end{aligned}$$

The root variables are z_1, x_1, z_2, y_2, z_3. The sets of maximal variables in each component are $W_1 = \{z_1\}$, $W_2 = \{z_2\}$, $W_3 = \{z_3\}$, and the set $\{z_1, z_2, z_3\}$ is a separator.

6 The Algorithm

Algorithm 6.1 takes as input a ranked probabilistic structure $\mathbf{A}$ and a positive sentence Φ in reduced form (Def 5.2), and computes the probability $P(\Phi)$, or fails. The algorithm proceeds recursively on the structure of the sentence Φ. The first step applies the Mobius inversion formula Eq.(6) to the C-lattice for Φ, expressing $P(\Phi)$ as a sum of several $P(\varphi)$, where each φ is a disjunctive sentence. Skipping those φ's where the Mobius function is zero, for all others it proceeds with the second step. Here, the algorithm first minimizes $\varphi = \bigvee \gamma_i$, then computes $P(\bigvee \gamma_i)$, by using Eq.(8), and Eq.(7). For the latter, the algorithm needs to find a separator first, as described in Sec. 5.2: if none exists, then the algorithm fails.

The expression $P(\varphi_0)$ represents the base case of the algorithm: this is when the recursion stops, when all variables have been substituted with constants from the structure $\mathbf{A}$. Notice that φ_0 is of the form $\bigvee r_i$, where each r_i is a ground atom. Its probability is $1 - \prod_i(1 - P(r_i))$, where P is the probability function of the probabilistic structure $(\mathbf{A}, P)$. We illustrate the algorithm with two examples.

Example 6.1 Let $\Phi = R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(y_2) \vee R(x_3), T(y_3)$. This example is interesting because, as we will show, the subexpression $R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(y_2)$ is hard (it has no separator), but the entire sentence is in PTIME. The algorithm computes the C-lattice, shown in Fig. 1 (b), then expresses $P(\Phi) = P(\varphi_4) + P(\varphi_5) - P(\varphi_6)$ where $\varphi_6 = R(x) \vee T(y)$ (see Example 3.3 for notations). Next, the algorithm applies the independence step to each of φ_4, φ_5, φ_6; we illustrate here for $\varphi_4 = R(x_3) \vee S(x_2, y_2), T(y_2)$ only; the other expressions are similar. Here, $\{x_3, y_2\}$ is a set of separator variables, hence:

$$P(\varphi_4) \quad = \quad 1 - \prod_{a \in A}(1 - P(R(a) \vee S(x_2, a), T(a)))$$

Next, we apply the algorithm recursively on $R(a) \vee S(x_2, a), T(a)$. In CNF it becomes[10] $(R(a) \vee S(x_2, a))(R(a) \vee T(a))$, and the algorithm returns $P(R(a) \vee S(x_2, a)) + P(R(a) \vee T(a)) - P(R(a) \vee S(x_2, a) \vee T(a))$. Consider the last of the three expressions (the other two are similar): its probability is

$$1 - (1 - P(R(a) \vee T(a))) \prod_{b \in A}(1 - P(S(b, a)))$$

Now we have finally reached the base case, where we compute the probabilities of sentences without variables: $P(R(a) \vee T(a)) = 1 - (1 - P(R(a)))(1 - P(T(a)))$, and similarly for the others.

Example 6.2 Consider the sentence φ in Example 5.8. Since this is already CNF (it is a disjunctive sentence), the algorithm proceeds directly to the second step. The separator is $V = \{z_1, z_2, z_3\}$ (see Ex. 5.8), and therefore:

$$P(\varphi) \quad = \quad 1 - \prod_{a \in A}(1 - P(\varphi[a/V])$$

where $\varphi[a/V]$ is:

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

After reducing the sentence (i.e. removing the constant a), it becomes identical to Example 6.1.

In the rest of this section we show that the algorithm is complete, meaning that, if it fails on a positive sentence Φ, then Φ is $FP^{\#P}$-hard.

[10]Strictly speaking, we would have had to rewrite the sentence into a reduce form first, by rewriting $S(x_2, a)$ into $S_{2<a}(x_2)$, etc.

6.1 Safe Sentences

The sentences on which the algorithm terminates (and thus are in PTIME) admit characterization as a minor-closed family, for a partial order that we define below.

Let φ be a disjunctive sentence. A *level* is a non-empty set of variables[11] W such that every atom in φ contains at most one variable in W and for any unifiable variables x, x', if $x \in W$ then $x' \in W$. In particular, a separator is a level W that has one variable in common with each atom; in general, a level does not need to be a separator. For a variable $x \in W$, let n_x be the number of atoms that contain x; let $n = \max_x n_x$. Let $A = \{a_1, \ldots, a_k\}$ be a set of constants not occurring in φ s.t. $k \leq n$. Denote $\varphi[A/W]$ the sentence obtained as follows: substitute each variable $x \in W$ with some constant $a_i \in A$ and take the union of all such substitutions:

$$\varphi[A/W] \quad = \quad \bigvee_{\theta : W \to A} \varphi[\theta]$$

Note that $\varphi[A/W]$ is not necessarily a disjunctive sentence, since some components γ_i may become disconnected in $\varphi[A/W]$. Moreover, since our vocabulary is ranked, we assume that in $\varphi[A/W]$ have a new symbol S_a for every symbol S that contains an attribute on the level W, and every constant $a \in A$: thus, $S_a(x_1, x_2, \ldots, y_1, y_2, \ldots)$ denotes $S(x_1, x_2, \ldots, a, y_1, y_2, \ldots)$.

Definition 6.3 *Define the following rewrite rule $\Phi \to \Phi_0$ on positive sentences. Below, $\varphi, \varphi_0, \varphi_1$, denote disjunctive sentences:*

$$\begin{array}{rcll} \varphi & \to & \varphi[A/W] & \textit{W is a level, A is a set of constants} \\ \varphi_0 \vee \varphi_1 & \to & \varphi_1 & \textit{if } Var(\varphi_0) = \emptyset \\ \Phi & \to & \varphi & \exists u \in \mathbf{L}_C(\Phi).\mu(u, \hat{1}) \neq 0, \varphi = \lambda(u) \end{array}$$

The second and third rules are called simple *rules. The first rule is also simple if W is a separator and $|A| = 1$.*

The first rewrite rule allows us to substitute variables with constants; the second to get rid of disjuncts without any variables; the last rule allows us to replace a CNF sentence Φ with one element of its C-lattice, provided its Mobius value is non-zero. The transitive closure $\xrightarrow{*}$ defines a partial order on positive sentences.

Definition 6.4 *A positive sentence Φ is called* unsafe *if there exists a sequence of simple rewritings $\Phi \xrightarrow{*} \varphi$ s.t. φ is a disjunctive sentence without separators. Otherwise it is called* safe.

Thus, the set of safe sentences can be defined as the downwards closed family (under the partial order defined by simple rewritings) that does not contain any disjunctive sentence without separators. The main result in this paper is:

Theorem 6.5 (Soundness and Completeness) *Fix a positive sentence Φ.*

[11] No connection to the maximal sets W_i in Sec. 5.2.

Soundness *If Φ is safe then, for any probabilistic structure, Algorithm 6.1 terminates successfully (i.e. doesn't fail), computes correctly $P(\Phi)$, and runs in time $O(n^k)$, where n is the size of the active domain of the structure, and k the largest arity of any symbol in the vocabulary.*

Completeness *If Φ is unsafe then it is hard for $FP^{\#P}$.*

Soundness follows immediately, by induction: if the algorithms starts with Φ, then for any sentence Φ_0 processed recursively, it is the case that $\Phi \xrightarrow{*} \Phi_0$, where all rewrites are simple. Thus, if the algorithm ever gets stuck, Φ is unsafe; conversely, if Φ is safe, then the algorithm will succeed in evaluating it on any probabilistic structure. The complexity follows from the fact that each recursive step of the algorithm removes one variable from every atom, and traverses the domain D once, at a cost $O(n)$. Completeness is harder to prove, and we discuss it in Sec. 6.3.

For a simple illustration, consider the sentence:

$$\varphi \quad = \quad R(z_1, x_1), S(z_1, x_1, y_1) \vee S(z_2, x_2, y_2), T(z_2, y_2)$$

To show that it is hard, we substitute the separator variables z_1, z_2 with a constant a, and obtain

$$\varphi \quad \to \quad R(a, x_1), S(a, x_1, y_1) \vee S(a, x_2, y_2), T(a, y_2)$$

Since the latter is a disjunctive sentence without a separator, it follows that φ is hard.

6.2 Discussion

An Optimization The first step of the algorithm can be optimized, as follows. If the DNF sentence $\Phi = \bigwedge \gamma_i$ is such that the relational symbols appearing in γ_i are distinct for different i, then the first step of the algorithm can be optimized to compute $P(\Phi) = \prod_i P(\gamma_i)$ instead of using Mobius' inversion formula. To see an example, consider the sentence $\Phi = R(x), S(y)$, which can be computed as

$$P(\Phi) \quad = \quad (1 - \prod_a (1 - P(R(a))))(1 - \prod_a (1 - P(S(a))))$$

Without this optimization, the algorithm would apply Mobius' inversion formula first:

$$\begin{aligned} P(\Phi) &= P(R(x)) + P(S(y)) - P(R(x) \vee S(y)) \\ &= 1 - \prod_a (1 - P(R(a))) + 1 - \prod_a (1 - S(a)) \\ &\quad - 1 + \prod_a (1 - P(R(a)) - P(S(a)) + P(R(a)) \cdot P(S(a))) \end{aligned}$$

The two expressions are equal, but the former is easier to compute.

A Justification We justify here two major choices we made in the algorithm: using the C-lattice instead of the D-lattice, and relying on the inversion formula with the Mobius function instead of some simpler method to eliminate unsafe subexpressions.

To see the need for the C-lattice, let's examine a possible dual algorithm, which applies the Mobius step to the D-lattice. Such an algorithm fails on Ex. 5.8, because here the D-lattice is $2^{[3]}$, and the Mobius function is $+1$ or -1 for every element of the lattice. The lattice contains $R(z_1, x_1), S(z_1, x_1, y_1), S(z_2, x_2, y_2), T(z_2, y_2)$, which is unsafe[12]. Thus, the dual algorithm fails.

To see the need of the Mobius inversion, we prove that an existential, positive FO sentence can be "as hard as any lattice".

Theorem 6.6 (Representation theorem) *Let $(\hat{L}, \leq)$ be any lattice. There exists a positive sentence Φ such that: $\mathbf{L}_D(\Phi) = (\hat{L}, \leq, \lambda)$, $\lambda(\hat{0})$ is unsafe, and for all $u \neq \hat{0}$, $\lambda(u)$ is safe. The dual statement holds for the C-lattice.*

Proof: Call an element $r \in L$ *join irreducible* if whenever $v_1 \vee v_2 = r$, then either $v_1 = r$ or $v_2 = r$. (Every atom is join irreducible, but the converse is not true in general.) Let $R = \{r_0, r_1, \ldots, r_k\}$ be all join irreducible elements in L. For every $u \in L$ denote $R_u = \{r \mid r \in R, r \leq u\}$, and note that $R_{u \wedge v} = R_u \cup R_v$. Define the following components[13]:

$$\begin{aligned}
\gamma_0 &= R(x_1), S_1(x_1, y_1) \\
\gamma_i &= S_i(x_{i+1}, y_{i+1}), S_{i+1}(x_{i+1}, y_{i+1}) \;\; i = 1, k-1 \\
\gamma_k &= S_k(x_k, y_k), T(y_k)
\end{aligned}$$

Consider the sentences Φ and Ψ below:

$$\Phi = \bigvee_{u < \hat{1}} \bigwedge_{r_i \in R_u} \gamma_i \qquad \Psi = \bigwedge_{u < \hat{1}} \bigvee_{r_i \in R_u} \gamma_i$$

Then both $\hat{\mathbf{L}}_D(\Phi)$ and $\hat{\mathbf{L}}_C(\Psi)$ satisfy the theorem. □

The theorem says that the lattice associated to a sentence can be as complex as any lattice. There is no substitute for checking if the Mobius function of a sub-query is zero: for any complex lattice $\hat{L}$ one can construct a sentence Φ that generates that lattice and where the only unsafe sentence is at $\hat{0}$: then Φ is safe iff $\mu_{\hat{L}}(\hat{0}, \hat{1}) = 0$.

6.3 Outline of the Completeness Proof

In this section we give an outline of the completeness proof and defer details to the Appendix. We have seen that Φ is unsafe iff there exists a rewriting $\Phi \xrightarrow{*} \varphi$ where φ has no separators. Call a rewriting *maximal* if every instance of the third rule in Def. 6.3, $\Phi \rightarrow \lambda(u)$, is such that for all lattice elements $v > u$, $\lambda(v)$ is safe: that is u is a maximal unsafe element in the CNF lattice. Clearly, if Φ is unsafe then there exists a maximal rewriting $\Phi \xrightarrow{*} \varphi$ where φ has no separators. We prove the following:

[12]It rewrites to $R(a, x_1), S(a, x_1, y_1), S(a, x_2, y_2), T(a, y_2) \longrightarrow R(a, x_1), S(a, x_1, y_1) \vee S(a, x_2, y_2), T(a, y_2)$.

[13]That is, $\bigvee \gamma_i = h_k$.

Lemma 6.7 *If $\Phi \xrightarrow{*} \varphi$ is a maximal rewriting, then there exists a PTIME algorithm for evaluating $P_{\mathbf{A}}(\varphi)$ on probabilistic structure $\mathbf{A}$, with a single access to an oracle for computing $P_{\mathbf{B}}(\Phi)$ on probabilistic structures $\mathbf{B}$.*

Thus, to prove that every unsafe sentence is hard, it suffices to prove that every sentence without separators is hard. To prove the latter, we will continue to apply the same rewrite rules to further simplify the sentence, until we reach an unsafe sentence where each atom has at most two variables: we call it a forbidden sentence. Then, we prove that all forbidden sentences are hard.

However, there is a problem with this plan. We may get stuck during rewriting before reaching a sentence with two variables per atom. This happens when a disjunctive sentence has no level, which prevents us from applying any rewrite rule. We illustrate here a simple sentence without a level:

$$\varphi \quad = \quad R(x, y), S(y, z) \vee R(x', y'), S(x', y')$$

Each consecutive pair of variables in the sequence x, x', y, y', z is unifiable. This indicates that no level exists, because it would have to include all variables, while by definition a level may have at most one variable from each atom; hence, this sentence does not have any level. While this sentence already has only two variables per atom, it illustrates where we may get stuck in trying to apply a rewriting.

To circumvent this, we transform the sentence (with two variables or more) as follows. Let $V = Var(\varphi)$. A *leveling* is a function $l : V \rightarrow [L]$, where $L > 0$, s.t. for all $i \in [L]$, $l^{-1}(i)$ is a level. Conceptually, l partitions the variables into levels, and assigns an integer to each level. This, in turn, associates exactly one level to each relation attribute, since unifiable variables must be in the same level. We also call φ an *L-leveled sentence*, or simply leveled sentence. Clearly, a leveled sentence has a level: in fact it has L disjoint levels. We show that each sentence is equivalent to a leveled sentence, on some restricted structures.

Call a structure $\mathbf{A}$ *L-leveled* if there exists a function $l : A \rightarrow [L]$ s.t. if two constants $a \neq b$ appear in the same tuple then $l(a) \neq l(b)$, and if they appear in the same column of a relation then $l(a) = l(b)$. For example, consider a single binary relation $R(A, B)$. An instance of R represents a graph. There are no 1-leveled instances, because for every tuple (a, b) we must have $l(a) \neq l(b)$. A 2-leveled instance is a bipartite graph. There are no 3-leveled structures, except if one level is empty. For a second example, consider two relations $R(A, B)$, $S(A, B)$. (Recall that our structures are ranked, hence for every tuple $R(a, b)$ or $S(a, b)$ we have $a < b$). An example of a 3-leveled structure is a 3-partite graph where the R-edges go from partition 1 to partition 2 and the S-edges go from partition 2 to partition 3.

Proposition 6.8 *Let φ be a disjunctive sentence that has no separators. Then there exists $L > 0$ and an L-leveled sentence φ^L s.t. that φ^L has no separator and the evaluation problem of φ^L over L-leveled structures can be reduced in PTIME to the evaluation problem of φ.*

The proof is in Appendix II. We illustrate the main idea on the example above. We choose $L = 4$ and the leveled sentence φ becomes:

$$\begin{aligned}\varphi^L &= R_{23}(x_2, y_3), S_{34}(y_3, u_4) \vee R_{12}(x_1, y_2), S_{23}(y_2, z_3) \vee \\ &\quad R_{23}(x_2', y_3'), S_{23}(x_2', y_3')\end{aligned}$$

Here φ^L is leveled, and still does not have a separator. It is also easy to see that if a probabilistic structure $\mathbf{A}$ is 4-leveled, then φ and φ^L are equivalent over that structure. Thus, it suffices to prove hardness of φ^L on 4-leveled structures: this implies hardness of φ.

To summarize, our hardness proof is as follows. Start with a disjunctive sentence without separators, and apply the leveling construct once. Then continue to apply the rewritings 6.3: it is easy to see that, whenever $\varphi \rightarrow \varphi'$ and φ is leveled, then φ' is also leveled; in other words we only need to level once.

Definition 6.9 *A* forbidden sentence *is a disjunctive sentence φ that has no separator, and is 2-leveled; in particular, every atom has at most two variables.*

All sentences h_k in Sec. 5.2 are forbidden sentences. Another example of a forbidden sentence is:

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

On the other hand, $R(x, y), S(y, z), T(x, z)$ is not a forbidden sentence because it has 3 levels.

We prove the following in Appendix II.

Theorem 6.10 *Suppose φ is leveled, and has no separator. Then there exists a rewriting $\varphi \overset{*}{\rightarrow} \varphi'$ s.t. φ' is a forbidden sentence.*

The level L may decrease after rewriting. In this theorem we must be allowed to use non-simple rewritings $\varphi \rightarrow \varphi[A/W]$, where W is not a separator (of course) and A has more than one constant. We show in Appendix II examples were the theorem fails if one restricts A to have size 1.

Finally, the completeness of the algorithm follows from the following theorem, which is technically the hardest result of this work. The proof is in Appendix III.

Theorem 6.11 *If φ is a forbidden sentence then it is hard for $FP^{\#P}$ over 2-leveled structures.*

7 Lifted Inference

Conditioning and disjointness are two important techniques in probabilistic inference. The first expresses the probability of some Boolean expression Φ as $P(\Phi) = P(\Phi \mid X) * P(X) + P(\Phi \mid \neg X) * (1 - P(X))$ where X is a Boolean variable. Disjointness allows us to write $P(\Phi \vee \Psi) = P(\Phi) + P(\Psi)$ when Φ and Ψ are exclusive probabilistic events. Recently, a complementary set of techniques called *lifted inference* has been shown to be very effective in probabilistic inference [13, 16, 15], by doing inference at the logic formula level instead of at the Boolean expression level. In the case of conditioning, *lifted conditioning* uses a sentence rather than a variable to condition.

We give an algorithm that uses lifted conditioning and disjointness in place of the Mobius step of Algorithm 6.1. When the algorithm succeeds, it runs in PTIME in the size of the probabilistic structure. However, we also show that the algorithm is incomplete; in fact, we claim that no algorithm based on these two techniques only can be complete.

Given $\Phi = \bigvee \varphi_i$, our goal is to compute $P(\Phi)$ in a sequence of conditioning/disjointness steps, without Mobius' inversion formula. The second step of Algorithm 6.1 (existential quantification based on independence) remains the same and is not repeated here. For reasons discussed earlier, that step requires that we have a CNF representation of the sentence, $\Psi = \bigwedge \varphi_i$, but both conditioning and disjointness operate on disjunctions, so we apply De Morgan's laws $P(\bigwedge \varphi_i) = 1 - P(\bigvee \neg\varphi_i)$. Thus, with some abuse of terminology we assume that our input is a D-lattice, although its elements are labeled with negations of disjunctive sentences.

We illustrate first with an example.

Example 7.1 Consider the sentence in Example 6.1:

$$\begin{aligned}\Phi &= R(x_1), S(x_1, y_1) \vee S(x_2, y_2), T(y_2) \vee R(x_3), T(y_3) \\ &= \varphi_1 \vee \varphi_2 \vee \varphi_3\end{aligned}$$

We illustrate here directly on the DNF lattice, without using the negation. (This works in our simple example, but in general one must start from the CNF, then negate.) The Hasse diagram of the DNF lattice is shown in Fig. 1. First, let's revisit Mobius' inversion formula:

$$P(\Phi) = P(\varphi_1) + P(\varphi_2) + P(\varphi_3) - P(\varphi_1\varphi_3) - P(\varphi_2\varphi_3)$$

The only unsafe sentence in the lattice is the bottom element of the lattice, where φ_1 and φ_2 occur together, but that disappears from the sum because $\mu(\hat{0}, \hat{1}) = 0$. We show how to compute Φ by conditioning on φ_3. We denote $\bar{\varphi} = \neg\varphi$ for a formula φ:

$$\begin{aligned}P(\Phi) &= P(\varphi_3) + P((\varphi_1 \vee \varphi_2) \wedge \bar{\varphi}_3) \\ &= P(\varphi_3) + P((\varphi_1, \bar{\varphi}_3) \vee (\varphi_2, \bar{\varphi}_3)) \\ &= P(\varphi_3) + P(\varphi_1, \bar{\varphi}_3) + P(\varphi_2, \bar{\varphi}_3) \\ &= P(\varphi_3) + (P(\varphi_1) - P(\varphi_1, \varphi_3)) + (P(\varphi_2) - P(\varphi_2, \varphi_3))\end{aligned}$$

The first line is conditioning (φ_3 is either true, or false), and the third line is based on mutual exclusion: φ_1 and φ_2 become mutually exclusive when φ_3 is false. We expand one more step, because our algorithm operates only on positive sentences: this is the fourth line. This last expansion may be replaced with a different usage, e.g. the construction of a BDD, not addressed in this paper. All sentences in the last line are safe, and the algorithm can proceed recursively.

For lifted conditioning to work, it is of key importance that we choose the correct subformula to condition. Consider what would happen if we conditioned on φ_1 instead:

$$\begin{aligned} P(\Phi) &= P(\varphi_1) + P((\varphi_2 \vee \varphi_3) \wedge \bar{\varphi}_1) \\ &= P(\varphi_1) + P(\varphi_2 \wedge \bar{\varphi}_1) + P(\varphi_3 \wedge \bar{\varphi}_1) \end{aligned}$$

Now we are stuck, because the expression $\varphi_2 \wedge \bar{\varphi}_1$ is hard.

Algorithm 7.1 computes the probability of a DNF formula using conditionals and disjointness. The algorithm operates on a DNF lattice (the negation of a CNF sentence). The algorithm starts by minimizing the expression $\bigvee_i \varphi_i$, which corresponds to removing all elements that are not co-atomic from the DNF lattice L (Prop 3.4). Recall that $\Phi = \bigvee_{u<\hat{1}} \lambda(u)$.

Next, the algorithm chooses a particular sub-lattice E, called the *cond-lattice*, and conditions on the disjunction of all sentences in the lattice. We define E below: first we show how to use it. Denote $u_1, \ldots, u_k$ the minimal elements of $\{z \mid \neg(\exists x \in E. z \leq x)\}$. For any subset $S \subseteq L$, denote $\Phi_S = \bigvee_{u \in S, u<\hat{1}} \lambda(u)$; in particular, $\Phi_L = \Phi$.

The *conditioning* and the *disjointness* rules give us:

$$\begin{aligned} P(\Phi_L) &= P(\Phi_E) + P(\Phi_{L-E} \wedge (\neg\Phi_E)) \\ &= P(\Phi_E) + \sum_{i=1,k} (\Phi_{[u_i,\hat{1}]} \wedge (\neg\Phi_E)) \end{aligned}$$

We have used here the fact that, for $i \neq j$, the sentences $\Phi_{[u_i,\hat{1}]}$ and $\Phi_{[u_j,\hat{1}]}$ are disjoint given $\neg\Phi_E$. Finally, we do this:

$$P(\Phi_{[u_i,\hat{1}]} \wedge (\neg\Phi_E)) = P(\Phi_{[u_i,\hat{1}]}) - P(\Phi_{[u_i,\hat{1}]\wedge E})$$

where $[u_i, \hat{1}] \wedge E = \{u \wedge v \mid u \geq u_i, v \in E - \{\hat{1}\}\}$

This completes the high level description of the algorithm. We show now how to choose the cond-lattice, then show that the algorithm is incomplete.

7.1 Computing the Cond-Lattice

Fix a lattice $(\hat{L}, \leq)$. The set of zero elements, Z, and the set of z-atoms ZA are defined as follows[14]:

$$\begin{aligned} Z &= \{z \mid \mu_L(z, \hat{1}) = 0\} \\ ZA &= \{a \mid a \text{ covers some element } z \in Z\} \end{aligned}$$

The algorithm reduces the problem of computing $P(\Phi_L)$ for the entire lattice L to computing $P(\Phi_K)$ for three kinds of sub-lattices K: E, $[u_i, \hat{1}] \wedge E$, and $[u_i, \hat{1}]$. The

[14]"Covers" is defined in Sec. 3.

goal is to choose E to avoid computing unsafe sentences. We assume the worse: that every zero element $z \in Z$ is unsafe (if a non-zero element is unsafe then the sentence is hard). So our goal is: choose E s.t. for any sub-lattice K above, if z is a zero element and $z \in K$, then $\mu_K(z, \hat{1}) = 0$. That is, we can't necessarily remove the zeros in one conditioning step, but if we ensure that they continue to be zeroes, they will eventually be eliminated.

The *join closure* of $S \subseteq L$ is $cl(S) = \{\bigvee_{u \in s} u \mid s \subseteq S\}$. Note that $\hat{0} \in cl(S)$. The join closure is a join-semilattice and is made into a lattice by adding $\hat{1}$.

Definition 7.2 *Let L be a lattice. The cond-lattice $E \subseteq L$ is $E = \{\hat{1}\} \cup cl(Z \cup ZA)$.*

Next, we show that, if some element $z \in Z$ remains in one of the smaller lattices, then its Mobius function is the same as at was in L: since we assumed $\mu_L(z, \hat{1}) = 0$, this implies that z continues to be a Zero in the smaller lattice.

We start with the lattices $[u_i, \hat{1}]$: here the claim holds vacuously, because $Z \cap [u_i, \hat{1}] = \emptyset$.

Next, consider the lattice E. We prove in Appendix I:

Proposition 7.3 *For all $z \in Z$, $\mu_L(z, \hat{1}) = \mu_E(z, \hat{1})$.*

Finally, consider the lattices $[u_i, \hat{1}] \wedge E$. We prove in Appendix I:

Proposition 7.4 *For all $z \in Z$, and all $u \in L$, $\mu_E(z, \hat{1}) = \mu_{[u,\hat{1}] \wedge E}(z, \hat{1})$.*

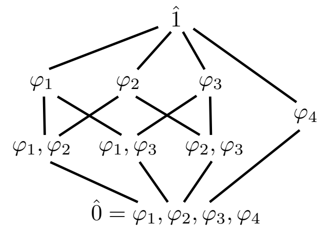
Combined with Prop. 7.3, we obtain $\mu_L(z, \hat{1}) = \mu_{[u,\hat{1}] \wedge E}(z, \hat{1})$.

Example 7.5 Consider Example 7.1. The cond-lattice for Fig. 1 (a) is

$$\begin{aligned} E &= cl(\{\hat{0}, (\varphi_1, \varphi_3), (\varphi_3, \varphi_2)\}) \\ &= \{\hat{0}, (\varphi_1, \varphi_3), (\varphi_3, \varphi_2), \varphi_3, \hat{1}\} \end{aligned}$$

Notice that this set is not co-atomic: in other words, when viewed as a sentence, it minimizes to φ_3, and thus we have gotten rid of $\hat{0}$.

To get a better intuition on how conditioning works from a lattice-theoretic perspective, consider the case when $Z = \{\hat{0}\}$. In this case ZA is the set of atoms, and E is simply the set of all atomic elements; usually this is a strict subset of L, and conditioning partitions the lattice into E, $[u_i, \hat{1}] \wedge E$, and $[u_1, \hat{1}]$. When processing E recursively, the algorithm retains only co-atomic elements. Thus, conditioning works by repeatedly removing elements that are not atomic, then elements that are not co-atomic, until $\hat{0}$ is removed, in which case we have removed the unsafe sentence and we can proceed arbitrarily.



$$\begin{array}{ll}
\gamma_1 = R(x_1), S_1(x_1, y_1) & \varphi_1 = \gamma_3, \gamma_4 \\
\gamma_2 = S_1(x_2, y_2), S_2(x_2, y_2) & \varphi_2 = \gamma_2, \gamma_4 \\
\gamma_3 = S_2(x_3, y_3), S_3(x_3, y_3) & \varphi_3 = \gamma_1, \gamma_4 \\
\gamma_4 = S_3(x_4, y_4), T(y_4) & \varphi_4 = \gamma_1, \gamma_2, \gamma_3
\end{array}$$
$$\Phi = \varphi_1 \vee \varphi_2 \vee \varphi_3 \vee \varphi_4$$

Figure 2: A lattice that is atomic, coatomic, and $\mu(\hat{0}, \hat{1}) = 0$. Its sentence Φ is given by Th. 6.6 (compare to h_3 in Sec. 5.2).

7.2 Incompleteness

Assume $\hat{0}$ is an unsafe sentence, and all other sentences in the lattice are safe. Lifted conditioning proceeds by repeatedly replacing the lattice L with a smaller lattice E, obtained as follows: first retain only the atomic elements ($= cl(Z \cup ZA)$), then retain only the co-atomic elements (this is minimization of the resulting formula). Conditioning on any formula other than E is a bad idea, because then we get stuck having to evaluate the unsafe formula at $\hat{0}$. Thus, lifted conditioning repeatedly trims the lattice to the atomic-, then to the co-atomic-elements, until, hopefully, $\hat{0}$ is removed. Proposition 3.4 implies that, if $\hat{0}$ is eventually removed this way, then $\mu(\hat{0}, \hat{1}) = 0$. But does the converse hold ?

Fig.2 shows a lattice where this process fails. Here $\mu(\hat{0}, \hat{1}) = 0$; by Th. 6.6 there exists a sentence Φ that generates this lattice, where $\hat{0}$ is unsafe and all other elements are safe (Φ is shown in the Figure). Yet the lattice is both atomic and co-atomic. Hence cond-lattice is the entire lattice $E = L$. We cannot condition on any formula and still have $\mu(\hat{0}, \hat{1}) = 0$ in the new lattice. In other words, no matter what formula we condition on, we will eventually get stuck having to evaluate the sentence at $\hat{0}$. On the other hand, Mobius' inversion formula easily computes the probability of this sentence, by exploiting directly the fact that $\mu(\hat{0}, \hat{1}) = 0$.

8 Conclusions

We have proposed a simple, yet non-obvious algorithm for computing the probability of an existential, positive sentence over a probabilistic structure. For every *safe* sentence, the algorithm runs in PTIME in the size of the input structure; every *unsafe* sentence is hard. Our algorithm relies in a critical way on Mobius' inversion formula, which allows it to avoid attempting to compute the probability of sub-sentences that are hard. We have also discussed the limitations of an alternative approach to computing probabilities, based on conditioning and independence.

Algorithm 7.1 Compute $P\Phi)$ using lifted conditional
Input: $\Phi = \bigvee_{i=1,m} \varphi_i$, $L = L_{DNF}(\Phi)$
Output $P(\Phi)$

```
1: Function Cond$(L)$
2: If $L$ has a single co-atom Then proceed with IndepStep
3: Remove from $L$ all elements that are not co-atomic (Prop 3.4)
4: Let $Z = \{u \mid u \in L, \mu_L(u, \hat{1}) = 0\}$
5: Let $ZA = \{u \mid u \in L, u$ covers some $z \in Z\}$
6: If $Z = \emptyset$ Then $E := [u, \hat{1}]$ for arbitrary $u$
7:    Else $E := cl(Z \cup ZA)$
8: If $E = L$ then FAIL (unable to proceed)
9: Let $u_1, \ldots, u_k$ be the minimal elements of $L - E$
10: Return $\mathbf{Cond}(E) + \sum_{i=1,k} \mathbf{Cond}(u_i) - \mathbf{Cond}([u_i, \hat{1}] \wedge E)$
```

Acknowledgments We thank Christoph Koch and Paul Beame for pointing us (independently) to incidence algebras, and the anonymous reviewers for their comments. This work was partially supported by NSF IIS-0713576.

References

[1] N. Creignou and M. Hermann. Complexity of generalized satisfiability counting problems. *Inf. Comput*, 125(1):1–12, 1996.

[2] Nadia Creignou. A dichotomy theorem for maximum generalized satisfiability problems. *J. Comput. Syst. Sci.*, 51(3):511–522, 1995.

[3] N. Dalvi, K. Schnaitter, and D. Suciu. Computing query probability with incidence algebras. Tehnical Report UW-CSE-10-03-02, University of Washington, March 2010.

[4] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, Toronto, Canada, 2004.

[5] N. Dalvi and D. Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *PODS*, pages 293–302, 2007.

[6] N. Dalvi and D. Suciu. Management of probabilistic data: Foundations and challenges. In *PODS*, pages 1–12, Beijing, China, 2007. (invited talk).

[7] Adnan Darwiche. A differential approach to inference in bayesian networks. *Journal of the ACM*, 50(3):280–305, 2003.

[8] E. Grädel, Y. Gurevich, and C. Hirsch. The complexity of query reliability. In *PODS*, pages 227–234, 1998.

[9] Kevin H. Knuth. Lattice duality: The origin of probability and entropy. *Neurocomputing*, 67:245–274, 2005.

[10] C. Krattenthaler. Advanced determinant calculus. *Seminaire Lotharingien Combin*, 42 (The Andrews Festschrift):1–66, 1999. Article B42q.

[11] Dan Olteanu and Jiewen Huang. Secondary-storage confidence computation for conjunctive queries with inequalities. In *SIGMOD*, pages 389–402, 2009.

[12] Dan Olteanu, Jiewen Huang, and Christoph Koch. Sprout: Lazy vs. eager query plans for tuple-independent probabilistic databases. In *ICDE*, pages 640–651, 2009.

[13] D. Poole. First-order probabilistic inference. In *IJCAI*, 2003.

[14] Yehoushua Sagiv and Mihalis Yannakakis. Equivalences among relational expressions with the union and difference operators. *Journal of the ACM*, 27:633–655, 1980.

[15] P. Sen, A.Deshpande, and L. Getoor. Bisimulation-based approximate lifted inference. In *UAI*, 2009.

[16] Parag Singla and Pedro Domingos. Lifted first-order belief propagation. In *AAAI*, pages 1094–1099, 2008.

[17] Richard P. Stanley. *Enumerative Combinatorics*. Cambridge University Press, 1997.

[18] Ingo Wegener. BDDs–design, analysis, complexity, and applications. *Discrete Applied Mathematics*, 138(1-2):229–251, 2004.

Part I
Lattice Theory

A Background

We review here a few results from lattice theory that we need throughout the paper.

Proposition A.1 *[17, pp.159, exercise 30] Let $(\hat{L}, \leq)$ be a finite lattice. A mapping $x \to \bar{x}$ on $\hat{L}$ is called a* closure *if forall $x, y \in \hat{L}$: (a) $x \leq \bar{x}$, (b) if $x \leq y$ then $\bar{x} \leq \bar{y}$, and (c) $\bar{\bar{x}} = \bar{x}$. A* closed element *is an element x s.t. $x = \bar{x}$. Denote $\bar{L}$ the subset of closed elements. Then:*

$$\mu_{\bar{L}}(\bar{x}, \bar{y}) \quad = \quad \sum_{z \in \hat{L} : \bar{z} = \bar{y}} \mu_{\hat{L}}(\bar{x}, z)$$

Corollary A.2 *Let $E \subseteq \hat{L}$ be a subset that is closed under meet. If all coatoms are in E then forall $u \in E$, $\mu_E(u, \hat{1}) = \mu_{\hat{L}}(u, \hat{1})$.*

Proof: The mapping $x \to \bar{x} = \bigwedge_{y \in E : y \leq x} y$ is a closure. The proposition follows from the fact that the only element z s.t. $\bar{z} = \hat{1}$ is $z = \hat{1}$ (because all coatoms are closed). $\Box$

Finally, we review the form of the Mobius function on a product space, which we need in Part III. Here we use the notation $\bar{L} = L \cup \{\hat{1}\}$ for the completion of a semilattice L.

Lemma A.3 *Let $(L, \leq)$ be a meet-semilattice, and let $\bar{L} = L \cup \{\hat{1}\}$ be its completion to a lattice. Let $(L^X, \leq)$ be the product space of the meet-semilattice, and $\overline{L^X} = L^X \cup \{\hat{1}\}$ be its completion. Then, forall $f \in L^X$, $\mu_{\overline{L^X}}(f, \hat{1}) = -\prod_{a \in X} \mu_{\bar{L}}(f(a), \hat{1})$*

Proof: Recall that if K is any lattice, then $\mu_{K^X}(f, g) = \prod_{a \in X} \mu_K(f(a), g(a))$. That is, the Mobius function of the cartesian product is the product of the Mobius functions. The lattice $\overline{L^X}$, is not a cartesian product. In fact, it is a strict subset of the cartesian product $(\bar{L})^X$. However, for every $f, g \in L^X$, the subset $[f, g] \subseteq L^X$ is a lattice, and is the cartesian product of lattices, $[f, g] = \prod_{a \in X} [f(a), g(a)]$. Let $f \in L^X$. Using the standard identity for the Mobius function we have:

$$\mu_{\overline{L^X}}(f, \hat{1}) = -\sum_{g \in \overline{L^X} : f \leq g < \hat{1}} \mu_{\overline{L^X}}(f, g)$$

If $g \in \overline{L^X}$ and $g < \hat{1}$ then $g \in L^X$, hence:

$$\begin{aligned}
\mu_{\overline{L^X}}(f, \hat{1}) &= -\sum_{g \in L^X : f \leq g} \mu_{\overline{L^X}}(f, g) \\
&= -\sum_{g \in L^X : f \leq g} \mu_{[f,g]}(f, g) = -\sum_{g \in L^X : f \leq g} \left(\prod_{a \in X} \mu_{[f(a), g(a)]}(f(a), g(a)) \right) \\
&= -\prod_{a \in X} \sum_{u \in L : f(a) \leq u} \mu_L(f(a), u) = -\prod_{a \in X} (-\mu_L(f(a), \hat{1})) \\
&= (-1)^{|X|+1} \prod_{a \in X} \mu_L(f(a), \hat{1})
\end{aligned}$$

$\Box$

B Proofs

We can prove now Prop. 3.4, Prop. 7.3, and Prop 7.4.

Proof: (of Proposition 3.4) (1) See [17]. (2) Apply Corollary A.2 to the set of coatomic elements. $\Box$

Proof: (Of Prop. 7.3) It is a standard fact in incidence algebras that $\mu_L(x, y)$ depends only on the sublattice $[x, y]$. Thus, $\mu_L(z, \hat{1}) = \mu_{[z,\hat{1}]}(z, \hat{1})$ and $\mu_E(z, \hat{1}) = \mu_{E \cap [z,\hat{1}]}(z, \hat{1})$. We need to prove $\mu_{[z,\hat{1}]}(z, \hat{1}) = \mu_{E \cap [z,\hat{1}]}(z, \hat{1})$, and for that we use the dual of Corollary A.2, in the lattice $[z, \hat{1}]$. By construction of E, the set $E \cap [z, \hat{1}]$ is closed under joins, and contains all atoms of $[z, \hat{1}]$, which proves the claim. $\Box$

Proof: (Of Prop 7.4) The proof is identical to that of Prop. 7.3. First, notice that $E \subseteq [u, \hat{1}] \wedge E$, because every element $x \in E$ can be written as $\hat{1} \wedge x$. Denote $L' = [u, \hat{1}] \wedge E$. Then E is a subset of L' that is closed under joins, and contains all atoms of $[z, \hat{1}]$. Then, as in Prop. 7.3, we have $\mu_{E \cap [z,\hat{1}]}(z, \hat{1}) = \mu_{([u,\hat{1}] \wedge E) \cap [z,\hat{1}]}(z, \hat{1})$ □

Part II
Rewriting to Forbidden Queries

C Rewriting Preserves Hardness

We prove here Lemma 6.7, which states that, if $\Phi \xrightarrow{*} \varphi$ is a maximal rewriting, then the evaluation problem for φ can be reduced to that for Φ.

Proof: We consider each type of rewriting (1) $\varphi \rightarrow \varphi[C/W]$ and $\varphi[C/W]$, where W is a level and C is a set of constants. Suppose we are given an input structure $\mathbf{A}$ for $\varphi[C/W]$. Recall that this structure is over a different vocabulary, where each symbol S containing an element of the level W is replaced with $|C|$ symbols S_a, forall $a \in C$. Given the structure $\mathbf{A}$, we construct a new structure $\mathbf{B}$ for the vocabulary for φ as follows: for every tuple $S_a(b_1, b_2, \ldots, c_1, c_2, \ldots)$ in $\mathbf{A}$ we create a tuple $S_a(b_1, b_2, \ldots, a, c_1, c_2, \ldots)$ in $\mathbf{B}$, with the same probability. All tuples belonging to relations that do not have any attributes in the level W we simply copy from the structure $\mathbf{A}$ to $\mathbf{B}$. Obviously, $P_A(\varphi[C/W]) = P_B(\varphi)$.

(2) Assume $\varphi = \varphi_0 \vee \varphi_1$ and φ_1 is unsafe. Let $\mathbf{A}$ be a structure. Remove from $\mathbf{A}$ all ground tuples that occurs in φ_0; this does not affect $P(\varphi_1)$, because the alphabet is ranked, hence φ_0 and φ_1 are independent; call $\mathbf{B}$ the new structure. Then $P_A(\varphi_1) = P_B(\varphi)$.

(3) Assume $\Phi \rightarrow \varphi$, where $\varphi = \lambda(u)$, for $u \in \mathbf{L}_C$ s.t. $\mu(u, \hat{1}) \neq 0$. Here we will choose u s.t. it is a maximal element in the lattice with this property. In other words, forall $v > u$, $\lambda(v)$ is safe. That is, we choose a particular rewriting from Φ into a sentence without separator. Let $\mathbf{A}$ be an input structure. We construct a new structure $\mathbf{B}$ by adding some more tuples to $\mathbf{A}$, as follows. Write $\varphi = \bigvee \gamma_i$. Let $v \in L$ be such that $u \not< v$, and denote $\varphi' = \lambda(v) = \bigvee_j \gamma'_j$. Since $\varphi' \not\Rightarrow \varphi$, there exists a component γ'_j s.t. for any i, $\gamma'_j \not\Rightarrow \gamma_i$. Let $\mathbf{A}_v$ be a structure obtained from γ'_j by substituting a fresh constant for each variable; that is, $\mathbf{A}_v \models \gamma'_j$. Moreover, set the probabilities of all these tuples to 1. We claim that $P_{\mathbf{A}}(\varphi) = P_{\mathbf{A} \cup \mathbf{A}_v}(\varphi)$. Suppose a component γ_i of φ is true on some substructure of $\mathbf{A} \cup \mathbf{A}_v$. Since γ_i is a component, it must be true in either a substructure of $\mathbf{A}$ or in $\mathbf{A}_v$: to prove this we use that fact that the only constants shared between $\mathbf{A}$ and $\mathbf{A}_v$ are those in C (the constants in Φ), and none of the variables in γ_i can be mapped to C (because the vocabulary is ranked w.r.t. C), hence, since γ_i is a component it is mapped either entirely to one or the other. But γ_i cannot be mapped to $\mathbf{A}_v$, because that would mean that there exists a homomorphism $\gamma_i \rightarrow \gamma'_j$, implying $\gamma'_j \Rightarrow \gamma_i$. Thus, we can add to $\mathbf{A}$ all structures $\mathbf{A}_v$ for all v s.t. $u \not< v$, without affecting the probability of φ. Denote $\mathbf{B}$ the resulting structure. We have: , $P(\varphi)$ can be reduced

in PTIME to $P(\Phi)$, because $P_B(\Phi) = P_B(\bigwedge_{v \in L} \lambda(v)) = P_B(\bigwedge_{v \geq u} \lambda(v))$ because for all $v \not\geq u$, $\lambda(v)$ is true on the substructure $\mathbf{A}_v$, and all those tuples have probability 1. Mobius' inversion formula gives us gives us $P_B(\Phi) = -\sum_{v \geq u} \mu(v, \hat{1}) P_B(\lambda(v))$. For $v = u$, we have $\mu(u, \hat{1}) \neq 0$ and $P_B(\lambda(u)) = P_A(\lambda(u)) = P_A(\varphi)$. For $v > u$ we have that $\lambda(v)$ is safe, and therefore we can compute $P_B(\lambda(v))$ in PTIME using Algorithm 6.1. This gives us $P_A(\varphi)$. $\square$

D Leveling

We give now the Proof of Prop. 6.8.

We start with a more formal definition of a level. Let V denote all variables in the disjunctive sentence $\varphi = \gamma_1 \vee \ldots \vee \gamma_m$ (we assume that γ_i and γ_j have disjoint sets of variables, when $i \neq j$). Let A denote the set of all attributes in the vocabulary: that is A is a set of pairs $a = (i, j)$, where i is the index of a relation symbol R_i and $j = 1, \texttt{arity}(R_i)$. Define two functions:

$$f : V \to A \qquad f(v) = \{a \mid v \text{ appears on the position of the attribute } a\}$$
$$g : A \to V \qquad g(a) = \{v \mid a \text{ contains the variable } v\}$$

When extended to sets, f, g form a Galois connection. Recall that f and g define a bijection between the closed subsets of V and the closed subsets of A. We call a *level* any closed subset of V; equivalently, a level is any closed subset of A. Thus, every sentence φ defines, through f, g above, a family of closed subsets of V and A which we call levels. One can check that φ is *leveled* if for any two distinct attributes a, b of the same relation symbol R, $\bar{a} \neq \bar{b}$. We also assume w.l.o.g. that $l^{-1}(k) \neq \emptyset$, for all $k \in [L]$.

Consider a disjunctive sentence $\varphi = \bigvee_{i=1,m} \gamma_i$ without separators. Suppose $Var(\gamma_i) \cap Var(\gamma_j) = \emptyset$ for $i \neq j$, and let $L = |Var(\varphi)|^{m^2}$. We will construct a new L-leveled sentence $varphi^L$ over a new vocabulary, s.t. for any L-leveled structure $\mathbf{A}$ for φ^L can be translated in PTIME to a structure $\mathbf{B}$ for φ s.t. $p_{\mathbf{A}}(\varphi^L) = p_{\mathbf{B}}(\varphi)$, and, furthermore, φ^L has no separator.

We start by describing the new vocabulary, which we call the *leveled vocabulary*. For every relational symbol R of arity k and every k-tuple $\bar{n} = (n_1, \ldots, n_k)$ of k distinct numbers $1 \leq n_1 < n_2 < \ldots < n_k \leq L$, define a new relational symbol $R^{\bar{n}}$. The mapping from an L-leveled structure $\mathbf{A}$ over the new vocabulary to a structure $\mathbf{B}$ over the vocabulary for φ is straightforward: take $R^{\mathbf{B}}$ to be the union of all $(R^{\bar{n}})^{\mathbf{A}}$. This union is disjoint: that is, if $\bar{n} \neq \bar{m}$, then no two tuples in $R^{\bar{n}}$ and $R^{\bar{m}}$ can be equal, because the structure $\mathbf{A}$ is leveled, which implies that attributes at different levels do not share any common constants. Thus, we define the probability of a tuple in $\mathbf{B}$ to be the same as that of the unique tuple in $\mathbf{A}$ that generated it.

Next, we define the leveled sentence φ^L. First, we level the variables. For every variable $x \in Var(\varphi)$ and every $n = 1, \ldots, L$ define a new variable x^n: here n is the level of the new variable. A *valid* atom over the leveled vocabulary is an atom of the form $a = R^{n_1 n_2 \ldots n_k}(x_1^{n_1}, \ldots, x_k^{n_k})$, i.e. where each variable has the same level as the corresponding attribute in the relation symbol. The *eraser* of the atom a is the atom $e(a) = R(x_1, \ldots, x_k)$ over the old vocabulary. Given any component (i.e. connected conjunctive sentence) γ^L, its eraser $e(\gamma^L) = \gamma$ is a conjunctive component over the old vocabulary. Finally, consider our disjunctive sentence φ. Define φ^L to be the disjunction of all connected components γ^L s.t. $e(\gamma^L)$ is a component in φ.

Proposition D.1 *Let* $\mathbf{A}$ *be a leveled structure over the leveled vocabulary, and* $\mathbf{B}$ *be the corresponding unleveled structure, over the old vocabulary. Then* $P_{\mathbf{A}}(\varphi^L) = P_{\mathbf{B}}(\varphi)$.

Proof: There is a one to one correspondence between possible worlds $\mathbf{A}_0 \subseteq \mathbf{A}$ and possible worlds $\mathbf{B}_0 \subseteq \mathbf{B}$, which preserves their probabilities. We will show that $\mathbf{A}_0 \models \varphi^L$ iff $\mathbf{B}_0 \models \varphi$, which proves the proposition. Clearly, if $\mathbf{A}_0 \models \varphi^L$, then $\mathbf{B}_0 \models \varphi$: simply take any component γ^L in φ^L and valuation $\theta^L : \gamma^L \to \mathbf{A}_0$ and notice that it extends to a valuation $\theta : e(\gamma^L) \to \mathbf{B}_0$. Conversely, suppose there exists a valuation $\theta : \gamma \to \mathbf{B}_0$. Each tuple $t \in \mathbf{B}_0$ comes from a tuple $t^L \in \mathbf{A}_0$, hence θ induces a leveling over the variables, and, therefore, a leveled component γ^L s.t. θ extends to $\theta^L : \gamma^L \to \mathbf{A}_0$. This proves the claim. $\Box$

Next, we prove that φ^L has no separator. By Prop 5.7 we need to show that some component γ^L has no maximal variable, i.e. each variable $x \in Var(\gamma^L)$ unifies (indirectly) with a non-root variable.

The *level range* of a leveled relational symbol $R^{n_1 \cdots n_k}$ is the interval $[n_1, n_k]$. The level range of a leveled component γ^L is the interval $[n', n'']$, where n' is the minimum level of any variable in φ^L, and n'' is the maximum level of any variable in φ^L.

Lemma D.2 *Let* γ_1 *unify with* γ_2 *(i.e. they share a relational symbol* R*), and let their common atom* R *have arity* k*. Let* $v_1 = Var(\gamma_1)$ *and* $v_2 = Var(\gamma_2)$*, and* $v = \max(v_1, v_2)$*. Then there exists leveled instances* φ_1^L, φ_2^L *s.t. the atom* R *has the same leveling* $R^{\bar{n}}$ *in both leveled components* φ_1^L *and* φ_2^L*, and the width of the leveled range of both* φ_1^L *and* φ_2^L *is at most* $2v - 2k + (k-1)(v-k) \leq v^2$.

Proof: Start by assigning levels to the atom R, and the variables occurring in R: leave a gap of size $v - 1$ between each two consecutive levels. Next assign a consistent leveling to φ_1: this is possible since we have left gaps wide enough, i.e. the remaining $v_1 - k$ variables can either occupy entirely a single gap, or be below, or above. The total gap is at most $2(v-k) + (k-1)(v-k)$. Similarly for φ_2^L. $\Box$

Corollary D.3 *Let* $S = (\gamma_1, \ldots, \gamma_p)$ *be a sequence of components in* φ *s.t. any two consecutive components unify. Then there exists levelings s.t. any two consecutive components unify, and the width of the leveled range in each* γ_i^L *is at most* $|Var(\varphi)|^p$*. We denote* $S^L = (\gamma_1^L, \ldots, \gamma_p^L)$.

Proof: By induction, using the previous lemma, and observing that every arity k is $\leq |Var(\gamma_i)|$, for every i. The induction step is the following. Start with a leveling for $\gamma_1, \ldots, \gamma_{p-1}$: $\gamma_1^L, \ldots, \gamma_{p-1}^L$. Modify the leveling by spreading each level to create new gaps of size $|Var(\varphi)|$ between any two consecutive levels: thus, the spread has increased by a factor of $|Var(\varphi)|$. Now we can create a leveling for φ_p by placing its new variables appropriately in the right gaps. $\Box$

Finally, we have:

Corollary D.4 *Fix γ, and let $S_1, S_2, \ldots, S_t$ be t sequences of unifications. Then there exists a leveling of all components occurring in all sequences s.t. all sequences $S_1^L, \ldots, S_t^L$ are valid sequences of unifications in φ^L and the widths of their spreads is at most $|Var(\varphi)^{pt}$, where p is the maximum length of any sequence S_i.*

The proof extends the one above.

Example D.5 Let:

$$\varphi \quad = \quad R(x,y), S(y,z) \vee R(x',y'), S(x',y') = \gamma_0 \vee \gamma_1$$

To level this sentence, we choose $L = 4$, and consider three leveling labels for both R and S: 12, 23, and 34. The sentence becomes:

$$\begin{aligned} \varphi^L \quad = \quad & R_{23}(x_2, y_3), S_{34}(y_3, u_4) \vee \\ & R_{12}(x_1, y_2), S_{23}(y_2, z_3) \vee \\ & R_{23}(x_2', y_3'), S_{23}(x_2', y_3') \end{aligned}$$

And inversion is $(x_2, y_3), (x_2', y_3'), (y_2, z_3)$.

E Rewriting to a Forbidden Query

We give here the Proof of Theorem 6.10. Let φ be a leveled, disjunctive sentence without separators, and with at least three distinct levels. More precisely, if V is its set of variables, then there exists a leveling function $l : V \to [L]$ where $L \geq 3$, and there is no leveling function $V \to [2]$. We will prove that φ rewrites to a simpler disjunctive sentence, still without separator.

E.1 Definitions

Definition E.1 *The co-occurrence graph of $\varphi = \bigvee_i \gamma_i$ is the following undirected graph. The nodes are all components γ_i and there exists an edge from a component γ_i to γ_j if γ_i and γ_j share a common relational symbol.*

The connected components of the co-occurrence graph partition φ into $\varphi_1 \vee \varphi_2 \vee \ldots$ s.t. φ_i and φ_j do not share any common predicate, for $i \neq j$. Obviously, φ has a separator iff every φ_i has a separator. Since φ has no separator, assume w.l.o.g. that φ_1 also has no separator. Then $\varphi \xrightarrow{*} \varphi_1$. To see this, simply rewrite all the other φ_j by substituting their variables with constants (i.e. repeated applications of the first rule in Def. 6.3), thus $\varphi \xrightarrow{*} \varphi_0 \vee \varphi_1$, where φ_0 consists only of constant atoms. Then apply the second rewrite rule in Def. 6.3, $\varphi_0 \vee \varphi_1 \to \varphi_1$. Therefore, to prove Theorem 6.10, it suffices to assume w.l.o.g. that the co-occurrence graph of φ is connected.

Given a level $l \in L$ and a component γ_i, denote $Var(\gamma_i, l)$ the variables in γ_i that are on level l. Given a set of constants A, denote:

$$\begin{aligned}
\Theta_l(\gamma_i, A) &= A^{Vars(\gamma_i,l)} \\
\gamma_i[A/l] &= \bigvee_{\theta \in \Theta_l(\gamma_i,A)} \gamma_i[\theta] \\
\varphi[A/l] &= \bigvee_i \gamma_i[A/l]
\end{aligned}$$

Thus, $\Theta_l(\gamma_i, A)$ denotes all substitutions of variables at level l in γ_i with constants from A, and $\varphi[A/l]$ is the sentence obtained by substituting variables at level l with constants from A.

The main difficulty in the proof of Theorem 6.10 is the fact that $\gamma_i[\theta]$ may no longer be a connected conjunctive sentence. If that happens, then we must write $\varphi[A/l]$ in CNF, then apply the third rewrite rule in Def 6.3. We examine now when $\gamma_i[\theta]$ is disconnected.

Let γ_i be a component and l a level. Define the level-l subcomponents of γ_i as follows. Construct a graph where the nodes are the atoms of γ_i and there exists an edge between two atoms if they share at least one variable that is not on level l. Denote $sc_l(\gamma_i) = \{\sigma_1, \ldots, \sigma_m\}$ the level l-subcomponents of γ_i. Obviously, if $\theta \in \Theta_l(\gamma_i, A)$, then the connected components in $\gamma_i[\theta]$ are precisely the subcomponents in $sc_l(\gamma_i)$. That is:

$$\gamma_i[\theta] = \bigwedge_{\sigma \in sc_l(\gamma_i)} \sigma[\theta]$$

and each $\sigma[\theta]$ is a separate connected component. Consider the CNF expression:

$$\begin{aligned}
\varphi[A/l] &= \bigvee_i \bigvee_{\theta \in \Theta_l(\gamma_i,A)} \gamma_i[\theta] \\
&= \bigwedge_k \varphi_k
\end{aligned}$$

Each φ_k has the form:

$$\varphi_k = \bigvee_i \bigvee_{\theta \in \Theta_l(\gamma_i,A)} \sigma[\theta] \tag{9}$$

where $\sigma \in sc_l(\gamma_i)$. In other words, the CNF is obtained as follows. For each component γ_i and each substitution θ of its l-level variables, choose a subcomponent $\sigma \in sc_l(\gamma_i)$, and take the disjunction $\bigvee_{i,\theta} \sigma[\theta]$; this gives once disjunctive sentence φ_k. Repeat this for all possible choices of σ and take the conjunction of the resulting expressions.

We assume that φ is minimized. In particular there are no homomorphisms $\gamma_i \to \gamma_j$ when $i \neq j$. However, such homomorphisms may exists between the components of φ_k as expressed in Eq.(9). We establish below a sequence of results that give us sufficient conditions for such homomorphisms not to exists.

Lemma E.2 *Let γ_i and γ_j be two components, and l a level. Suppose that forall $\sigma_k \in sc_l(\gamma_i)$ there exists a homomorphism $h_k : \sigma_k \to \gamma_j$ such that forall k_1, k_2 and all $x \in Var(\gamma_i, l)$, $h_{k_1}(x) = h_{k_2}(x)$; that is, for all subcomponents, their homomorphisms agree on the variables at level l. Then there exists a homomorphism $\gamma_i \to \gamma_j$. In particular, if φ is minimized, then $i = j$.*

Proof: The proof follows immediately from the definition of subcomponents. Any two subcomponents σ_{k_1} and σ_{k_2} share only variables at level l, and their homomorphisms agree on these variables. Hence, we can extend the homomorphisms to a single homomorphism on the entire γ_i. $\square$

This immediately implies:

Corollary E.3 *Let φ be minimized, and let $i \neq j$. Then for any function $h : Var(\gamma_i, l) \to Var(\gamma_j)$ there exists a subcomponent $\sigma \in sc_l(\sigma_i)$ s.t. there is no homomorphism $\sigma \to \gamma_j$ that agrees with h on the variables at level l.*

Fix a component γ_i, a subcomponent $\sigma_i \in sc_l(\gamma_i)$, and an injective substitution $\theta_i : Vars(\gamma_i, l) \to A$; such a substitution exists whenever $|A| \geq |Vars(\gamma_i, l)|$. Let φ_k be a disjunctive sentence given by Eq.(9); the conjunction of all the φ_k's forms $\varphi[A/l]$.

Definition E.4 *We say that φ_k* contains *$\sigma_i[\theta_i]$, if after minimizing φ_k one of its components is (isomorphic to) $\sigma_i[\theta_i]$.*

Recall that, if $\varphi_{k_1} \Rightarrow \varphi_{k_2}$ then we need to remove φ_{k_1} from the CNF expression $\varphi[A/l] = \bigwedge_k \varphi_k$. However, if φ_{k_1} contains $\sigma_i[\theta_i]$, then so does φ_{k_2}, by the containment criteria of Sagiv and Yannakakis [14] (see Eq.(5)). Thus, to prove that $\sigma_i[\theta_i]$ appears in $\varphi[A/l]$ it suffices to show that there exists some k s.t. φ_k contains it. We do not need to worry whether φ_k is redundant.

Recall that φ_k is uniquely determined by choosing, for each component γ_j and substitution $\theta \in \Theta_j(\gamma_j, A)$, a subcomponent $\sigma \in sc_l(\gamma_j)$: then φ_k is the disjunction $\bigvee_{j,\theta} \sigma[\theta]$. We say that $\sigma \in sc_l(\gamma_j)$ is *compatible* with $\sigma_i[\theta_i]$ and θ if there exists a homomorphism $\sigma[\theta] \to \sigma_i[\theta_i]$. Otherwise we call it *incompatible*.

Corollary E.5 *If φ is minimized then forall $\sigma_i \in sc_l(\gamma_i)$, $\theta_i \in \Theta_l(\gamma_i, A)$ and $\theta \in \Theta_l(\gamma_j, A)$, if θ_i is injective then there exists a subcomponent $\sigma \in sc_l(\gamma_j)$ s.t. $\sigma[\theta]$ is incompatible with $\sigma_i[\theta_i]$.*

Proof: Since θ_i is injective it admits a left inverse $g : A \to Var(\gamma_i, l)$: that is, g is a partial function s.t. $g(\theta_i(x)) = x$ forall $x \in Var(\gamma_i, l)$. Define $h = g \circ \theta$; this is a function $h : Var(\gamma_j, l) \to Var(\gamma_i, l)$, which is possibly partial. If it is a total function, then choose $\sigma \in sc_l(\gamma_j)$ as given in Corollary E.3, i.e. such that there exists no homomorphism $\sigma \to \gamma_i$ that extends h: it follows that there is no homomorphism $k : \sigma[\theta] \to \sigma_i[\theta_i]$ because if there were one then $g \circ k \circ \theta$ would give a homomorphism $\sigma \to \sigma_i$ that extends h, which is a contradiction. If h is not a total function, then choose $\sigma \in sc_l(\gamma_j)$ arbitrary: there is no homomorphism $\sigma[\theta] \to \sigma_i[\theta_i]$ because $\sigma[\theta]$ uses constants in A that do not occur in $\sigma_i[\theta_i]$ (this is why h is not total). $\square$

Corollary E.6 *Fix γ_i, $\sigma \in sc_l(\gamma_i)$ and $\theta_i \in \Theta_l(\gamma_i, A)$ s.t. θ_i is injective. Let φ_k be a disjunctive sentence in the CNF expression of $\varphi[A/l]$ obtained as follows: for each $(\gamma_j, \theta) \neq (\gamma_i, \theta_i)$, choose a component $\sigma \in sc_l(\gamma_j)$ that is incompatible with $\sigma_i[\theta_i]$; for γ_i, θ_i, choose the subcomponent σ_i. Then φ_k contains $\sigma_i[\theta_i]$. In particular, there exists φ_k s.t. contains $\sigma_i[\theta_i]$.*

The proof follows immediately from the previous corollary.

E.2 Root Variables

Recall that a root variable for γ_i is a variable that occurs in all atoms. Suppose γ_i has no root variables: then γ_i is not a hierarchical sentence [4, 6]. It is possible to show that directly that every non-hierarchical sentence is #P-hard, and thus remove them completely from the discussion: we have proven this in [5] for conjunctive queries with self-joins, and also include a proof here in Theorem J.1. However, Theorem 6.10 also holds for non-hierarchical queries, which is a result of independent interest from their hardness property, and therefore we treat non-hierarchical queries together with the rest. We give here the proof of Theorem 6.10 for the case when one component γ_i has no root variables; in particular, it is non-hierarchical.

Clearly, if γ_i has no root variables, then $\varphi = \bigvee_i \gamma_i$ does not have a separator: this follows from the fact that a separator must include a root variable from each component. Notice that γ_i has at least two variables x, y s.t. (a) there exists an atom that contains both x and y; in particular $l(x) \neq l(y)$; (b) there exists an atom that contains x but not y, and (c) there exists an atom that contains y but not x. Let l be a level that is different from $l(x)$ and $l(y)$; choose A a set of constants s.t. $|A| \geq |Var(\gamma_j, l)|$ forall γ_j, and consider the CNF expression (9) for $\varphi[A/l]$. Let $\sigma_i \in sc_l(\gamma_i)$ be the subcomponent that contains the variables x and y, and let $\theta_i \in \Theta_l(\gamma_i, A)$ be any injective substitution. By Corollary E.6 there exists a disjunctive sentence φ_k in the CNF expression of $\varphi[A/l]$ that contains $\sigma_i[\theta_i]$, even after minimization. Then, φ_k does not have a separator: indeed, $\sigma_i[\theta_i]$ continues to have the two variables x, y, and they still satisfy conditions (a), (b), (c), hence none of them is a root variable. Thus, φ_k does not have a separator. Then, the two step rewriting:

$$\varphi \rightarrow \varphi[A/l] \rightarrow \varphi_k$$

proves Theorem 6.10.

Thus, from now on we will assume that every component γ_i in φ has at least one root variable.

E.3 The Proof

If l is a level, then $|sc_l(\gamma_i)| > 1$ iff γ_i has exactly one root variable, and its level is l. This is because γ_i must have a root variable, and if it has at least one root variable on a level other than l, then that variable will keep all atoms of γ_i connected, hence $|sc_l(\gamma_i)| = 1$.

Denote:

$$U = \{l \mid \forall i.|sc_l(\gamma_i)| = 1\}$$

The following is obvious:

Lemma E.7 *If $l \in U$, then $\varphi[A/l]$ is a disjunctive sentence.*

When $U \neq \emptyset$ then we will choose any level in U. Suppose $U = \emptyset$. By assumption there are at least three distinct levels, call them 1,2,3. For $i = 1, 2, 3$, let:

$$\Gamma_i = \{\gamma \mid |sc_i(\gamma)| > 1\}$$

By assumption $\Gamma_i \neq \emptyset$, for all $i = 1, 2, 3$ (otherwise we have $i \in U$).

To prove Theorem 6.10 we need to show how to choose the level l such that $\varphi[A/l]$ has no separator. We consider three cases:

Case 1 If $U \neq \emptyset$ then choose l to be any level in U.

For the remaining cases we assume $U = \emptyset$ and consider $\Gamma_1, \Gamma_2, \Gamma_3$ defined above.

Case 2 Fix three components $\gamma_i \in \Gamma_i$, for $i = 1, 2, 3$. If there exists $\sigma \in sc_i(\gamma_i)$ and a homomorphism $\sigma \to \gamma_j$ for $j \neq i$, then we choose l to be the third level, i.e. $l \neq i, l \neq j$. (If there are multiple choices for i, j, pick one arbitrarily and choose l as described).

Case 3 For any $i \neq j$, let n_{ij} be the length of the shortest path in the co-occurrence graph from some $\gamma_i \in \Gamma_i$ to some $\gamma_j \in \Gamma_j$. Let i, j be such that $n_{ij} = \min(n_{12}, n_{13}, n_{23})$. Then choose l to be the third level, i.e. different from i and j.

Once we picked the level l, then we choose a set of constants A s.t. $|A| > |2Var(\gamma_i, l)|$ forall i, and rewrite:

$$\varphi \longrightarrow \varphi[A/l]$$

Next, we write $\varphi[A/l]$ in CNF using Equation (9). We will prove below that there exists k such that φ_k has no separator.

Before we give the proof, we illustrate with a few examples. We assume that the variables x, y, z are assigned to levels $1, 2, 3$ respectively.

Example E.8 We illustrate Case 1. Consider:

$$\varphi = R(x), S(x, y, z) \vee S(x, y, z), T(y)$$

Level 1 (i.e. the variable x) splits γ_1 into two subcomponents $R(x)$ and $S(x, y, z)$, while level 2 splits γ_2 into two subcomponents $S(x, y, z)$ and $T(y)$. Thus, $U = \{3\}$, meaning that level 3 does not split any component. We rewrite the sentence to[15]

$$\varphi[a/3] = R(x), S(x, y, a) \vee S(x, y, a), T(y)$$

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

The new disjunctive sentence is still without separator. Note that it would have been a mistake to choose level 1 because in that case the sentence rewrites to:

$$\begin{aligned}\varphi[a/1] &= R(a), S(a,x,y) \vee S(a,y,z), T(y)\\ &= (R(a) \vee S(a,y,z), T(y)) \wedge (S(a,x,y) \vee S(a,y,z), T(y))\end{aligned}$$

and both conjuncts have a separator (the sentence is actually safe). Thus, Case 1 is necessary.

We show now why it is necessary to choose $|A| > 1$. Consider:

$$\varphi = R(x,z_1)S(x,y_1,z_1)S(x,y_2,z_2)U(x,z_2) \vee S(x,y,z)T(y) \vee R(x,z)U(x,z)$$

We also have $U = \{3\}$, because variable x disconnects the first component, and variable y disconnects the second. So we pick level 3 to rewrite. However, if we use a single constant a then the first component becomes redundant and the sentence minimizes:

$$\begin{aligned}\varphi[a/3] &= R(x,a)S(x,y_1,a)S(x,y_2,a)U(x,a) \vee S(x,y,a)T(y) \vee R(x,a)U(x,a)\\ &= S(x,y,a)T(y) \vee R(x,a)U(x,a)\end{aligned}$$

Therefore $\varphi[a/3]$ has a separator (it is actually safe). We need to choose two constants, $A = \{a,b\}$:

$$\begin{aligned}\varphi[\{a,b\}/3] = \; & R(x,a)S(x,y_1,a)S(x,y_2,b)U(x,b) \vee R(x,b)S(x,y_1,b)S(x,y_2,a)U(x,a) \vee\\ & S(x,y,a)T(y) \vee S(x,y,b)T(y) \vee R(x,a)U(x,a) \vee R(x,b)U(x,b)\end{aligned}$$

One can check that $\varphi[\{a,b\}/3]$ is minimized and has no separator.

Example E.9 Next we illustrate Case 2. Consider:

$$\begin{aligned}\varphi &= R(x,y,z)S(x,y',z') \vee R(x,y,z)T(x',y,z') \vee S(x,y,z), T(x',y',z)\\ &= \gamma_1 \vee \gamma_2 \vee \gamma_3\end{aligned}$$

Level 1 (variable x) splits the first component into two subcomponents: $R(x,y,z)$ and $S(x,y',z')$; similarly levels 2 and 3 split the second and third components. Thus $U = \emptyset$. Moreover, Case 2 applies here, because every subcomponent maps into every other component. That is, we may choose any level l to do the rewriting. We illustrate with $l = 3$. Here we must choose at least two constants: $A = \{a,b\}$ and rewrite to:

$$\begin{aligned}\varphi[\{a,b\}/3] &= \gamma_1[\{a,b\}/3] \vee \gamma_2[\{a,b\}/3] \vee S(x,y,a)T(x',y',a) \vee S(x,y,b)T(x',y',b)\\ &= (\gamma_1[\{a,b\}/3] \vee \gamma_2[\{a,b\}/3] \vee S(x,y,a) \vee S(x,y,b)) \wedge\\ &\quad (\gamma_1[\{a,b\}/3] \vee \gamma_2[\{a,b\}/3] \vee S(x,y,a) \vee T(x',y',b)) \wedge\\ &\quad (\gamma_1[\{a,b\}/3] \vee \gamma_2[\{a,b\}/3] \vee T(x',y',a) \vee S(x,y,b)) \wedge\\ &\quad (\gamma_1[\{a,b\}/3] \vee \gamma_2[\{a,b\}/3] \vee T(x',y',a) \vee T(x',y',b))\end{aligned}$$

The CNF expression is given in the last four rows, and has four conjuncts: each row is obtained by choosing, for each constant a, b, either the subcomponent $S(x, y, z)$ or the subcomponent $T(x', y', z)$, for a total of four conjuncts. Consider the second conjunct:

$$\varphi_2 = \gamma_1[\{a, b\}/3] \vee \gamma_2[\{a, b\}/3] \vee S(x, y, a) \vee T(x, y, b)$$

Expanding the first two expressions results in 8 components. Some of these are redundant, because they contain either $S(x, y, a)$ or $T(x, y, b)$. However, the following two components are not redundant:

$$\varphi_2 = \ldots R(x, y, a)S(x, y', b) \vee R(x, y, a)T(x', y, a) \ldots$$

Hence φ_2 has no separator, because neither level 1 nor level 2 is a separator (they are not root levels in the two components above).

In the example above Case 2 applied, but we were allowed to pick any level we wanted. We illustrate now how Case 2 may restrict our choice. Consider:

$$\begin{aligned}\varphi &= U(x, y', z')V(x, y'', z'') \vee R(x', y, z')S(x'', y, z'')U(x'', y, z'') \vee V(x', y', z)S(x', y', z)T(x'', y'', z) \\ &= \gamma_1 \vee \gamma_2 \vee \gamma_3\end{aligned}$$

Here the variable x splits γ_1 into two subcomponents: $U(x, y', z')$ and $V(x, y'', z'')$. There is homomrphism from the first subcomponent to γ_2: by Case 2 we should not select levels 1 or 2, but select level $l = 3$ to expand. There is also a homomorphism from the second component to γ_3: by Case 2 we should select level $l = 2$ to expand. But we should not select level 1. In other words, Case 2 allows us to select levels 2 or 3, but not level 1. To see why level 1 doesn't work, consider expanding it with a set of constants $A = \{a, b, c, \ldots\}$:

$$\begin{aligned}\varphi[A/1] &= \bigvee_{v \in A} U(v, y', z')V(v, y'', z'') \vee \gamma_2[A/1] \vee \gamma_3[A/1] \\ &= \bigwedge_k \varphi_k\end{aligned}$$

Here φ_k represent the disjuncts in the CNF expansion. Each such φ_k contains $\gamma_2[A/1] \vee \gamma_3[A/1]$ and, for each constant $v \in A$, it contains either $U(v, y', z')$ or $V(v, y'', z'')$ (thus, there are $2^{|A|}$ disjunctive sentences φ_k). However, each φ_k has a separator. Indeed, consider two terms from the expansion of $\gamma_2[A/1] \vee \gamma_3[A/1]$ that, together, have no separator:

$$\gamma_i = \ldots R(a, y, z')S(b, y, z'')U(b, y, z'') \vee V(b, y', z)S(b, y', z)T(x'', y'', z)$$

Note that we must use the same common constant, say b, in the two atoms S, to allow them to be unifiable (otherwise the two components have a separator). But γ_i contains either $U(b, y', z')$ or $V(b, y'', z'')$, making either the first or the second component above redundant. This shows that we cannot expand on level 1. Instead, if we follow Case 2, then we will expand on either level 2 or 3. Choosing $l = 3$ and a single constant

a we obtain:

$$\begin{aligned}
\varphi[a/3] &= U(x,y',a)V(x,y'',a) \vee R(x',y,a)S(x'',y,a)U(x'',y,a) \vee V(x',y',a)S(x',y',a)T(x'',y'',a) \\
&= (U(x,y',a)V(x,y'',a) \vee R(x',y,a)S(x'',y,a)U(x'',y,a) \vee V(x',y',a)S(x',y',a)) \wedge \\
&\quad (U(x,y',a)V(x,y'',a) \vee R(x',y,a)S(x'',y,a)U(x'',y,a) \vee T(x'',y'',a))
\end{aligned}$$

Neither conjunct has a separator.

Suppose φ has no separator. Our theorem follows from the following three lemmas.

Lemma E.10 (Case 1) *Suppose φ has no separator, $U \neq \emptyset$, and let $l \in U$. Let A be a set of constants s.t. $|A| \geq |Var(\gamma_i, l)|$ for all i. Then $\varphi[A/l]$ is a disjunctive sentence and has no separator.*

Proof: For any substitution $\theta \in \Theta_l(\gamma_i, A)$, $\gamma_i[\theta]$ is connected. Thus, $\varphi[A/l]$ is a disjunctive sentence. Some of the components $\gamma_i[\theta]$ may be redundant. However, by Corollary E.6, for any injective substitution $\theta_i : Vars(\gamma_i, l) \rightarrow A$, $\gamma_i[\theta]$ is not redundant.

We show now that $\varphi[A/l]$ has no separator. Suppose it had a separator V'. Construct a separator V for φ as follows. For each component γ_i, let θ be any injective substitution $\theta : Var(\gamma_i, l) \rightarrow A$. Define the separator variable x_i for γ_i to be the same separator variable for $\gamma_i[\theta]$ (the two sentences are isomorphic and "the same variable" means up to this isomorphism). Note that this definition is independent on our choice of θ. Indeed, if θ' is another injective substitution that agrees with θ on at least one variable x ($\theta(x) = \theta'(x)$) then $\gamma_i[\theta]$ and $\gamma_i[\theta']$ have two atoms that unify, hence their separator variables must unify, which means that they correspond to the same variable in γ_i. It is easy to check that the set $\{x_i \mid i = 1, \ldots\}$ is indeed a separator for φ: if γ_i and γ_j unify, then there exists substitutions θ, θ' s.t. $\gamma_i[\theta]$ unifies with $\gamma_j[\theta']$, and therefore the separator variables must unify as well. $\Box$

From now on we will assume that Case 1 does not apply. Let 1,2,3 be three arbitrary levels, and $\Gamma_1, \Gamma_2, \Gamma_3$ be as defined above. Fix $\gamma_i \in \Gamma_i$, for $i = 1, 2, 3$.

Lemma E.11 (Case 2) *Suppose there exists $\sigma_1 \in sc_1(\gamma_1)$ and a homomorphism $h : \sigma_1 \rightarrow \gamma_2$. Let A be a set of constants such that $|A| \geq |Var(\gamma_1, 3)| + |Var(\gamma_2, 3)|$ (the number of variables on the 3rd level in γ_1 plus those in γ_2). Consider the CNF expression for $\varphi[A/3] = \bigwedge \varphi_k$. Then there exists k s.t. φ_k has no separator.*

Proof: Let $\theta_2 \in \Theta_3(\gamma_2, A)$ be any injective substitution, and denote $A_2 \subseteq A$ the set of constants in $Im(\theta_2)$. Define θ_1 to be injective and s.t. forall $x \in Var(\gamma_1, 3) \cap Var(\sigma_1)$, $\theta_1(x) = \theta_2(h(x))$; that is, we use $|Var(\gamma_1, 3) - Var(\sigma_1)|$ constants in addition to those in $Im(\theta_2)$: denote A_1 the set consisting of these constants. We claim that there exists some φ_k in Eq.(9) that contains both components $\gamma_1[\theta_1]$ and $\gamma_2[\theta_2]$ (note that both are connected). Note that this proves our claim: φ_k does not have a separator, because the only root variable in $\gamma_1[\theta_1]$ is on level 1, and the only root variable in $\gamma_2[\theta_2]$ is on level 2.

Suppose that there exists a homomorphism $\sigma[\theta] \to \gamma_i[\theta_i]$, for $i = 1$ or 2, where $\sigma \in sc_3(\gamma)$. Then $\gamma \in \Gamma_3$: otherwise $\sigma = \gamma$ and we obtain a homomorphism $\gamma \to \gamma_i$, contradicting the fact that the sentence φ was minimized. Thus, assume $\gamma \in \Gamma_3$, and let z denote the variable at level 3 in γ (z is the single root variable in γ). Recall that, when defining φ_k, we may choose any $\sigma \in sc_3(\gamma)$ for each γ, θ. We consider two cases. First, when $\theta(z) \in A_2$. Then we choose σ s.t. it is incompatible with $\gamma_2[\theta_2]$: this is possible by Corollary E.5. It follows that there is no homomorphism $\sigma[\theta] \to \gamma_1[\theta_1]$ either, because the image of such a homomorphism must be in $\sigma_1[\theta_1]$ (since all constants in A_2 are here), and this would give us a homomorphism $\sigma[\theta] \to \gamma_2[\theta_2]$. Second, when $\theta(z) \in A_1$. Then we choose σ to be incompatible with $\gamma_1[\theta_1]$. Clearly there is no homomorphism $\sigma[\theta] \to \gamma_1[\theta_1]$. In addition there is no homomorphism $\sigma[\theta] \to \gamma_2[\theta_2]$ either, because every atom in $\sigma[\theta]$ contains the constant $\theta(z)$, which does not occur in $\gamma_2[\theta_2]$. $\Box$

Fix $i \neq j$, and $\gamma_i \in \Gamma_i$ and $\gamma_j \in \Gamma_j$. Since we assumed that the co-occurrence graph is connected, there is a path $\delta_0 = \gamma_i, \delta_1, \delta_2, \ldots, \delta_n = \gamma_j$ s.t. any two consecutive components δ_{k-1}, δ_k share a common relational symbol. Call n the length of the path from γ_i to γ_j. Let n_{ij} be the length of the shortest path from any component in Γ_i to any component in Γ_j

Lemma E.12 (Case 3) *Suppose* $n_{12} = \min(n_{12}, n_{13}, n_{23})$. *Let* A *be a set of constants such that* $|A| \geq \max_i(|Var(\gamma, 3)|)$, *for all components* γ *in* φ. *Then there exists a disjunctive sentence* γ_k *in the expansion (9) of* $\varphi[A/3]$ *that has no separator.*

Proof: Let $\delta_0 = \gamma_1, \delta_1, \ldots, \delta_n = \gamma_2$ be a shortest path from some component $\gamma_1 \in \Gamma_1$ to some component $\gamma_2 \in \Gamma_2$; thus, $n_{12} = n$. Note that $\delta_i \notin \Gamma_3$ for all $i = 0, n$: otherwise, we would have a shorter path from Γ_1 to Γ_3.

Define, inductively, the substitutions $\theta_i \in \Theta(\delta_i, A)$ as follows. Start by defining $\theta_0 : Var(\delta_0, 3) \to A$ to be any injective substitution. To define $\theta_k : Var(\delta_k, 3) \to A$, consider the two atoms in δ_{k-1} and δ_k that share a common relational symbol. They may share at most one variable at level 3: choose θ_k to be any injective substitution that agrees with θ_{k-1} on the common variable at level 3. Note that $\delta_i[\theta_i]$ is connected, for all $i = 0, \ldots, n$, because none of the δ_i's is in Γ_3.

We will show now that there exists a disjunctive sentence γ_k that contains all components $\delta_i[\theta_i]$: this proves our claim, since γ_k cannot have a separator, because the only root variable in δ_0 is on level 1, and the only root variable in δ_n is on level 2. Thus, we need to show that, for every component γ and every substitution $\theta \in \Theta_3(\gamma, A)$, there exists $\sigma \in sc_3(\gamma)$ s.t. $\sigma[\theta]$ is incompatible with $\delta_i[\theta_i]$ for all $i = 0, \ldots, n$.

First, notice that the only way we can have a homomorphism $\sigma[\theta] \to \delta_i[\theta_i]$ is if $\gamma \in \Gamma_3$: otherwise, $|sc_3(\gamma)| = 1$, and $\sigma = \gamma$, and a homomorphism $\gamma[\theta] \to \delta_i[\theta_i]$ implies a homomorphism $\gamma \to \delta$, which is a contradiction. So assume $\gamma \in \Gamma_3$. Then we cannot have $i = 0$ or $i = n$, because then we would be in Case 2, and we have assumed that the condition for Case 2 does not hold. Thus $i \in \{1, \ldots, n-1\}$. We consider two cases.

Case 1: there exists σ, θ such that there exists a homomorphism $\sigma[\theta] \to \delta_i[\theta_i]$, for some $i = 1, \ldots, n-1$. This gives us the path $\delta_0 = \gamma_1, \delta_1, \ldots, \delta_i, \gamma$ in the co-occurrence graph from Γ_1 to Γ_3. Since we assumed our path from Γ_1 to Γ_2 is the shortest one, we conclude $i = n-1$. Similarly, we obtain the path $\gamma, \delta_{i+1}, \ldots, \delta_n = \gamma_2$ from Γ_3 to Γ_2: we conclude that $i = 1$. Thus, $n = 2$, in other words the path from Γ_1 to Γ_2 is $\gamma_1, \delta_1, \gamma_2$. For any γ, θ, choose $\sigma \in sc_3(\gamma)$ such that $\sigma[\theta]$ is incompatible with $\delta_1[\theta_1]$: this is possible by Corollary E.5. Then the resulting φ_k contains the entire path $\gamma_1[\theta_0], \delta_1[\theta_1], \gamma_2[\theta_2]$.

Case 2: there is no homomorphism $\sigma[\theta] \to \delta_i[\theta_i]$, for any $i = 1, \ldots, n-1$. Then any φ_k contains all components $\delta_i[\theta_i]$, for $i = 0, \ldots, n$. $\Box$

Part III

Forbidden Queries are Hard

We give here the proof of the most difficult result in this paper: that all forbidden sentences are hard. We use in this chapter the term *Boolean query* or simply *query* to refer to a positive, existential FO sentence.

F Problem Setting

Let Q be a disjunctive query:

$$Q = q_1 \vee q_2 \vee q_3 \vee \ldots$$

where each q_i is a connected, conjunctive query, called a *component*. The query is 2-leveled: that means, there exists a function $l : Var(Q) \to \{1, 2\}$ s.t. forall $k \in \{1, 2\}$, $l^{-1}(k)$ is a *level*, i.e. contains exactly one variable from each atom, and is closed under unification.

Definition F.1 *The co-occurrence graph of Q is the following undirected graph $G(Q)$. The nodes are all relation symbols, and there exists an edge from S_i to S_j if S_i and S_j co-occur in a component q_k.*

Denote $l^{-1}(1) = \{x_1, x_2, \ldots\}$ and $l^{-1}(2) = \{y_1, y_2, \ldots\}$. Each atom may contain at most one x-variable and one y-variable. We call a component q_i a *left component* if no variable y occurs in all atoms; we call it a *right component* if no variable x occurs in all atoms. Any symbol that occurs in some left component is called a left symbol; any symbol occurring in some right component is a right symbol.

Definition F.2 *A 2-leveled disjunctive query Q is called a* forbidden query *if there exists a path in $G(Q)$ from a left symbol to a right symbol.*

Lemma F.3 *A 2-leveled query Q is forbidden iff it has no separator.*

Proof: Suppose that there exists a path from S to S'. We show that there is no separator. Suppose W is a separator. Let q be a left component that contains S: then W

must contain the x-variable in q. Similarly, if q' is a right component containing S', then W must contain the y-variable from q'. Consider path from S to S' in the co-occurrence graph. Each edge between two symbols S_i, S_j corresponds to a component that contains both symbols. Augment the beginning of the path with the edge S, S (corresponding to q) and the end of the path with the edge S', S' (corresponding to q'). The separator contains either an x- or a y-variable from each query along the path. Since it starts with an x-variable and ends with a y-variable, there exists two consecutive edges where the separator chooses an x-variable from q_i on the first edge, and a y-variable from q_j on the other edge; but this violates the condition that the separator be closed under unification.

Conversely, assume no such path exists. Then we partition the co-occurrence graph into connected components. We show how to construct a separator, by including from each component either a x or a y variable, as follows. There are three kinds of connected components: (a) those that contain at least one left-component; then it cannot contain a right component: here choose the x-variable. (b) those that contain at least one right-component; here choose the y-variable. (c) those that contain only middle queries; choose arbitrarily the x-variable. $\Box$

In this document we prove:

Theorem F.4 *For every forbidden query Q, computing $P(Q)$ on a probabilistic database is hard for FP$^{\#P}$.*

G The Signature Counting Problem

Let $\Phi = \bigvee_{(x,y)\in E} x \wedge y$ be a PP2DNF. The counting problem for PP2DNF, denoted #PP2DNF, asks for the number of assignments that make the formula true. It is known to be #P-complete [1].

Equivalently, Φ is a bipartite graph (X, Y, E), where $E \subseteq X \times Y$. Let $n_1 = |X|$, $n_2 = |Y|$, $n = |E|$. We call interchangeably xy an edge, or a conjunct in Φ. Let $m_1, m_2 = O(1)$ be two numbers, $m_1 \geq 2, m_2 \geq 2$: that is, these numbers are fixed, and independent of n_1, n_2, and are at least 2. We define the set of *left labels* and *right labels* as $LL = [m_1]$ and $RR = [m_2]$. In all definitions below we assume that the bipartite graph Φ is fixed.

Definition G.1 *A* labeling *is a pair of functions $l = (l_1, l_2)$, where $l_1 : X \to LL$ and $l_2 : Y \to RR$. There are $n_1^{m_1} \times n_2^{m_2}$ labelings.*

Definition G.2 *An* edge signature *σ^E, a* node signature on the left *σ^X and a* node signature on the right *σ^Y are functions:*

$$\begin{aligned}\sigma^E &: LL \times RR \to \{0, 1, \ldots, n\}\\ \sigma^X &: LL \to \{0, 1, \ldots, n\}\\ \sigma^Y &: LL \to \{0, 1, \ldots, n\}\end{aligned}$$

Definition G.3 *We define four types of signatures: the left end may be of type 1 or 2, and the right end may be of type 1 or 2.*

- *A signature of type 1-1 is* $\sigma = \sigma^E$.
- *A signature of type 1-2 is* $\sigma = (\sigma^E, \sigma^Y)$.
- *A signature of type 2-1 is* $\sigma = (\sigma^X, \sigma^E)$.
- *A signature of type 2-2 is* $\sigma = (\sigma^X, \sigma^E, \sigma^Y)$.

We write a signature as $\sigma = ([\sigma^X,]\sigma^E[,\sigma^Y])$ to indicate that σ^X and σ^Y are optional. In the sequel, the type of all signatures is fixed by the query Q, and is one of 1-1, 1-2, 2-1, 2-2. We will assume the type to be fixed, and denote Σ_{m_1,m_2} be the set of signatures of the fixed type, for the bipartite graph Φ. Note: $|\Sigma_{m_1,m_2}| \leq n^{m_1 m_2 + m_1 + m_2}$.

Definition G.4 *Let* $l = (l_1, l_2)$ *be a labeling. The signature of the labeling* l *is* $\sigma_l = ([\sigma_l^X,]\sigma_l^E[,\sigma_l^Y])$*, where:*

$$\begin{aligned}
\sigma_l^E(\tau_1,\tau_2) &= |\{(x,y) \in E \mid l_1(x) = \tau_1, l_2(y) = \tau_2\}| & \forall(\tau_1,\tau_2) \in LL \times RR \\
\sigma_l^X(\tau_1) &= |\{x \in X \mid l_1(x) = \tau_1\}| & \forall\tau_1 \in LL \\
\sigma_l^Y(\tau_2) &= |\{y \in Y \mid l_2(y) = \tau_2\}| & \forall\tau_2 \in RR
\end{aligned}$$

Definition G.5 *Let* $\sigma \in \Sigma_{m_1,m_2}(\Phi)$ *be a signature. The* signature count *of* σ*, denoted* $\#\sigma$*, is the number of labelings that have signature* σ*:*

$$\#\sigma = |\{l \mid \sigma_l = \sigma\}|$$

Let $\#\Sigma_{m_1,m_2}(\Phi) = \{\#\sigma \mid \sigma \in \Sigma_{m_1,m_2}(\Phi)\}$ *denote the set of all signature counts. There are* $|\#\Sigma_{m_1,m_2}(\Phi)| \leq n^{m_1 m_2 + m_1 + m_2}$ *signature counts.*

Definition G.6 *Fix* $m_1, m_2 \geq 2$*. The* m_1, m_2*-Signature-Counting (SC) problem is the following: given a bipartite graph* Φ*, compute* $\#\Sigma_{m_1,m_2}(\Phi)$*.*

Example G.7 Let $m_1 = m_2 = 2$. Assume the type 1-1. Then the 2,2-counting problem is the following. Given a bipartite graph $\Phi = (X, Y, E)$, for all tuples of 4 numbers $\sigma = \sigma^E = (n_{11}, n_{12}, n_{21}, n_{22})$ where $n_{11}, n_{12}, n_{21}, n_{22} \leq |E|$, compute $\#\sigma$ the number of possible labelings of the X-nodes and the Y-nodes s.t. exactly n_{11} edges have their endpoints labeled 1 and 1, exactly n_{12} edges have their endpoints labeled 1 and 2, etc. Note that, if $n_{11} + n_{12} + n_{21} + n_{22} \neq n$, then $\#\sigma = 0$, because no such labelings is possible: we leave such inconsistent signatures in the problem definition, for convenience. Suppose now that the type is 2-2. Then forall 8 numbers $\sigma^E = (n_{11}, n_{12}, n_{21}, n_{22})$, $\sigma^X = (n_1, n_2)$ and $\sigma^Y = (n_1', n_2')$ we need to compute the number of labelings s.t., the number of edges whose endpoints are labeled 1,1 is n_{11} etc; the number of nodes in X labeled 1 is n_1 and the number of nodes in X labeled 2 is n_2; and the number of nodes in Y labeled 1 is n_1' and the number of nodes in Y labeled 2 is n_2'.

Let $m_1 \leq m_1'$ and $m_2 \leq m_2'$. Given an m_1, m_2 signature σ, we can view it as an m_1', m_2'-signature by setting $\sigma(\tau_1, \tau_2) = 0$ whenever $\tau_1 > m_1$ or $\tau_2 > m_2$. With this convention, we have $\Sigma_{m_1,m_2}(\Phi) \subseteq \Sigma_{m_1',m_2'}(\Phi)$ and similarly $\#\Sigma_{m_1,m_2}(\Phi) \subseteq \#\Sigma_{m_1',m_2'}(\Phi)$.

Proposition G.8 *SC is hard for FP$^{\#P}$.*

Proof: By reduction from the PP2DNF problem. Assume that we have an oracle for solving the m_1, m_2-SC problem, for $m_1, m_2 \geq 2$. We can assume w.l.o.g. that the signatures are of type 1,1: if the signatures are of, say, type 2,2, then we convert signature counts $\#(\sigma^X, \sigma^E, \sigma^Y)$ into $\#\sigma^E$ by summing over all signatures σ^X and σ^Y. Thus, we assume all signatures are of type 1,1. From $\Sigma_{m_1,m_2}(\Phi)$ we obtain $\Sigma_{2,2}(\Phi)$ (since $\Sigma_{2,2}(\Phi) \subseteq \Sigma_{m_1,m_2}(\Phi)$). Next, we use an oracle for the $2,2$-SC problem to solve the #PP2DNF problem: assuming the encoding $1 = \texttt{false}$, $2 = \texttt{true}$, we obtain the number of satisfying assignments to the Boolean expression Φ by summing $\#\sigma$ over all signatures σ where $|\{(x, y) \in E \mid l_1(x) = 2, l_2(y) = 2\}| \geq 1$:

$$\#PP2DNF \quad = \quad \sum\{\#\sigma \mid \sigma \in \Sigma_{2,2}(\Phi), \sigma(2, 2) \geq 1\}$$

$\square$

The proof Theorem F.4 is the following: we will show that, for every forbidden query Q, there exists two numbers $m_1, m_2 \geq 2$ such that the m_1, m_2-SC problem can be reduced to the evaluation problem $P(Q)$ over probabilistic databases D.

H An Example

We illustrate the key elements of the proof, by proving hardness of the query:

$$H_1 \quad = \quad R(x), S(x, y) \vee S(x, y), T(y)$$

Recall that hardness of $H_0 = R(x), S(x, y), T(y)$ is very easy to prove, by direct reduction from #PP2DNF: given $\Phi = (X, Y, E)$ construct a database having tuples $R(a_i)$, $T(b_j)$, $S(a_i, b_j)$ with probabilities $1/2, 1/2, 1$, forall $a_i \in X$, $b_j \in Y$, and $(a_i, b_j) \in E$: the number of satisfying assignments for Φ is exactly the number of worlds that make H_0 true. Hence, the associated counting problem for H_0 is #P-hard.

We do not know of a similarly simple construct for H_1, or for any other forbidden query, and this forces us to use a more complex proof. We illustrate here the hardness proof for H_1. This query is of type 1-1, so all signatures below are of type 1-1. We will prove its hardness by providing a PTIME algorithm for the $2, 2$-SC problem with access to an oracle for computing $P(H_1)$: this shows that $P(H_1)$ is hard for FP$^{\#P}$. Note that the $2, 2$-SC problem is more general that the #PP2DNF (but still complete for $FP^{\#P}$). There are two parts to the hardness proof of H_1, which we call the *Algorithmic* and the *Algebra*. There is a third part, called the *Combinatorics*, for more complex queries.

H.1 Algorithmic Part

Fix $\Phi = (X, Y, E)$, $X = \{a_i \mid i = 1, n_1\}$, $Y = \{b_j \mid j = 1, n_2\}$, $E \subseteq X \times Y$, $|E| = n$. Construct a probabilistic database D as the union of blocks $D(a_i, b_j)$, one for each edge $(a_i, b_j) \in E$. Each block $D(a_i, b_j)$ is, in turn, of the union of four blocks $D_k(a_i, b_j)$, for $k = 1, 2, 3, 4$:

$$\begin{aligned}
D &= \bigcup_{(a_i,b_j)\in E} D(a_i,b_j)\\
D(a_i,b_j) &= D_1(a_i,b_j) \cup D_2(a_i,b_j) \cup D_3(a_i,b_j) \cup D_4(a_i,b_j)\\
D_k(a_i,b_j) &= \{R(a_i), S(a_i,c_{ijk}), T(c_{ijk}), S(d_{ijk},c_{ijk}), R(d_{ijk}), S(d_{ijk},b_j), T(b_j)\}
\end{aligned}$$

The only tuples shared between blocks are $R(a_i), T(b_j)$: we set their probabilities to $1/2$ and call them *endpoints*. Each block $D_k(a_i,b_j)$ contains two endpoints and five additional tuples. We fix the probabilities of four of them to some constant values (same values forall i,j,k), and for the fifth tuple we set its probability to be some variable $x_k \in (0,1)$ (same forall i,j, different for $k = 1,2,3,4$). For example, we may set $P(S(d_{ijk},c_{ijk})) = x_k$, and for all other tuples we set their probabilities to $1/2$, but this choice needs to be made carefully, as we explain below. For now it suffices to say that a single tuple in each block $D_k(a_i,b_j)$ has a variable probability.

Let $LL = RR = \{1,2\}$. Each possible world $W \subseteq D$ defines $l^W = (l_1^W, l_2^W)$ for Φ as follows:

$$l_1^W(a_i) = \begin{cases} 1 & \text{if } R(a_i) \in W \\ 2 & \text{if } R(a_i) \notin W \end{cases} \qquad l_2^W(b_j) = \begin{cases} 1 & \text{if } T(b_j) \in W \\ 2 & \text{if } T(b_j) \notin W \end{cases}$$

Fix a labeling l and consider the event "a possible world $W \subseteq D$ has labeling l", formally $l^W = l$. Clearly $P(l^W = l) = 1/2^{n_1+n_2}$. We want to compute the probability that H_1 is false conditioned on $l^W = l$: $P(\neg H_1 | l^W = l)$.

Fix an edge (a_i,b_j) and let $\tau_1 = l(a_i), \tau_2 = l(b_j)$ be the labeling of its endpoints, $\tau = (\tau_1,\tau_2)$. Denote $f_\tau(x_k)$ the probability that the query H_1 is false on the block $D_k(a_i,b_j)$, conditioned on the label τ. That is, given that $R(a_i)$ and $T(b_j)$ are true or false, as specified by τ_1 and τ_2, $f_\tau(x_k)$ denotes the probability that H_k is false on $D_k(a_i,b_j)$; for example, $f_{12}(x_k)$ denotes the probability that H_1 is false on a block $D_k(a_i,b_j)$, assuming $R(a_i)$ is true and $T(b_j)$ is false. Note that $f_\tau(x_k)$ is the same for all i and j, so we do not include the indices i,j in the expression. Thus, $f_\tau(x_k)$ is a linear polynomial in x_k:

$$f_\tau(x_k) = A_\tau + B_\tau x_k$$

The two coefficients depend on the labeling τ. The probability that H_1 is false on $D(a_i,b_j)$ conditioned on the label τ is:

$$F_\tau(\bar{x}) = f_\tau(x_1) \cdot f_\tau(x_2) \cdot f_\tau(x_3) \cdot f_\tau(x_4) \tag{10}$$

This gives us:

$$\begin{aligned}
P(\neg H_1 | l^W = l) &= \prod_{(a_i,b_j)\in E} F_{l_1(a_i),l_2(b_j)}(\bar{x}) \\
&= \prod_{\tau\in LL\times RR} F_\tau^{\sigma_l(\tau)}(\bar{x}) \\
&= F_{11}^{n_{11}}(\bar{x}) \cdot F_{12}^{n_{12}}(\bar{x}) \cdot F_{21}^{n_{21}}(\bar{x}) \cdot F_{22}^{n_{22}}(\bar{x})
\end{aligned}$$

where $\sigma_l = (n_{11}, n_{12}, n_{21}, n_{22})$, and each n_τ represents the number of edges in E whose endpoints are labeled τ. For each $\sigma \in \Sigma_{2,2}(\Phi)$ there are $\#\sigma$ labelings l s.t. $\sigma_l = \sigma$, and therefore we obtain (recall that $P(l^W = l) = 1/2^{n_1+n_2}$):

$$\begin{aligned}
P(\neg H_1) &= \sum_l P(\neg H_1 | l^W = l) P(l^W = l) \\
&= \frac{1}{2^{n_1+n_2}} \sum_{\sigma\in\Sigma_{2,2}(\Phi)} \prod_{\tau\in LL\times RR} F_\tau^{\sigma(\tau)}(\bar{x}) \cdot \#\sigma
\end{aligned}$$

We will use repeatedly an oracle for computing $P(\neg H_1)$ in order to obtain the values $\#\sigma$, forall $\sigma \in \Sigma_{2,2}(\Phi)$, thus solving the 2,2-SC problem. Since $|\Sigma_{2,2}(\Phi)| = (n+1)^4$, we have $(n+1)^4$ unknowns[16] $\#\sigma$. Choose $(n+1)^4$ different values for the four variables $\bar{x} = (x_1, x_2, x_3, x_4)$, $\bar{x} = \bar{v}_1, \bar{v}_2, \ldots, \bar{v}_{(n+1)^4}$ and invoke the $P(H_1)$-oracle for each value. We obtain a linear system with $(n+1)^4$ equations and $(n+1)^4$ unknowns, whose matrix is:

$$\begin{aligned}
M &= \Big(\prod_\tau F_\tau^{\sigma(\tau)}(\bar{v})\Big)_{\sigma\in\Sigma,\bar{v}\in V} \\
&= \big(F_{11}^{n_{11}}(\bar{v}_i) \cdot F_{12}^{n_{12}}(\bar{v}_i) \cdot F_{21}^{n_{21}}(\bar{v}_i) \cdot F_{22}^{n_{22}}(\bar{v}_i)\big)_{\substack{n_{11},n_{12},n_{21},n_{22} \le n \\ i=0,\ldots,(n+1)^4}}
\end{aligned}$$

where $V = \{\bar{v}_1, \bar{v}_2, \ldots, \bar{v}_{(n+1)^4}\}$. If M is non-singular, then we can solve the system in polynomial time, and thus obtain the solution to the 2,2-SC problem, proving hardness for the computation problem for $P(H_1)$.

H.2 Algebra Part

It remains to show that M is non-singular. This follows from three facts, which we describe below, continuing to restrict the discussion to H_1.

Fact H.1 *Let $\bar{F}(\bar{x}) = (F_1(\bar{x}), F_2(\bar{x}), F_3(\bar{x}), F_4(\bar{x}))$, where $\bar{x} = (x_1, x_2, x_3, x_4)$ be a function $\mathbf{R}^4 \to \mathbf{R}^4$, where each F_k is a multilinear polynomial. Suppose that the*

[16] For every labeling l, its signature has the property $\sum_{\tau\in LL\times RR} \#\sigma_l(\tau) = n$. If a signature does not have this property, then $\#\sigma = 0$, hence there are only $O(n^3)$ unknowns, but we prefer to treat all $\#\sigma$'s as an unknowns, for a total of $(n+1)^4$.

Jacobian $\bar{x} \to \bar{F}$ is non-zero at some point $\bar{x} \in R^4$. Define the following matrix:

$$M = \Big(F_1^{j_1}(\bar{x}_i) \cdot F_2^{j_2}(\bar{x}_i) \cdot F_3^{j_3}(\bar{x}_i) \cdot F_4^{j_4}(\bar{x}_i)\Big)_{\substack{i = 1, (n+1)^4 \\ 0 \leq j_1, j_2, j_3, j_4 \leq n}} \quad (11)$$

That is, the columns are given by four independent exponents j_1, j_2, j_3, j_4 and the rows are given by $(n+1)^4$ values for the variables $\bar{x}$. Then M is nonsingular, and, moreover, one can find in PTIME $(n+1)^4$ values $V = \{\bar{v}_1, \ldots, \bar{v}_{(n+1)^4}\}$ such that the matrix M at these values is non-singular.

We give the general statement and proof in Sec. I.1. The intuition is quite simple, however. When the values V are considered unknowns, then $det(M)$ is a multivariate polynomial in $4(n+1)^4$ variables, where each variable has degree $\leq 4(n+1)^4$. Thus, the idea in the proof is to show that one can find in PTIME values for the unknowns s.t. $det(M) \neq 0$. To give the intuition why this is possible, consider a simpler problem: given a polynomial $f(x)$ in a single variable x, of degree N, find a value v s.t. $f(v) \neq 0$. Clearly this can be done in time $O(N)$, times the time needed to evaluate f. Indeed, by trying out at most $N+1$ values for x, one can find a value v s.t. $f(v) \neq 0$. We will show that this idea extends to a proof of the claim.

Next, we turn to the Jacobian of the function $\bar{F}(\bar{x})$.

Fact H.2 *Let $f_1(x)$, $f_2(x)$, $f_3(x)$, $f_4(x)$ be four linear, non-constant polynomials in one variable x. Suppose their roots are distinct. For $k = 1, \ldots, 4$ define:*

$$F_k(x_1, x_2, x_3, x_4) = f_k(x_1) \cdot f_k(x_2) \cdot f_k(x_3) \cdot f_k(x_4)$$

Then, for any four distinct values $\bar{v} = (v_1, v_2, v_3, v_4)$, the Jacobian of $\bar{F}(\bar{x})$ is non-zero at $\bar{v}$.

We give the general statement and proof in Sec. I.2.

Next, we need to show that the polynomials $f_{11}, f_{12}, f_{21}, f_{22}$ have distinct roots. Recall that two multivariate polynomials f, g are called *equivalent* if there exists a number $c \neq 0$ s.t. $f = c \cdot g$. If $f(x)$, $g(x)$ are two linear polynomials in a single variable x, then they are equivalent iff their roots are equal. A multivariate polynomial is called *irreducible* if it cannot be expressed as a product of two non-constant polynomials: a classic result in algebra states that every multivariate polynomial can be uniquely decomposed as a product of irreducible polynomials, up to equivalence. Recall that we obtained the polynomials $f_{11}(x), \ldots, f_{22}(x)$ by setting other variables to constants. We prove:

Fact H.3 *Let p_1, p_2, p_3, p_4 be four multilinear polynomials. Suppose that they have some irreducible factors $p_1^0, p_2^0, p_3^0, p_4^0$, such that p_i^0 is a factor of p_i, with the following properties: the four factors are pairwise inequivalent, and they share a common variable x. Let $\bar{y}$ be all variables other than x in $p_1, \ldots, p_4$. Then there exists values $\bar{v}$ such that after substituting $\bar{v}$ for $\bar{y}$ in $p_1, \ldots, p_4$ the resulting linear polynomials $f_1(x), \ldots, f_4(x)$ are not constants, and have distinct roots.*

We give the general statement and the proof in Sec. I.3.

This brings us to the very core of the hardness proof: the connection between queries without separator and irreducible polynomials. Recall that the block $D_k(a_i, b_j)$ has five tuples: $S(a_i, c_{ijk}), T(c_{ijk}), S(d_{ijk}, c_{ijk}), R(d_{ijk}), S(d_{ijk}, b_j)$. Denote $Z_1, \ldots, Z_5$ the events that the tuples are not present in a possible world $W \subseteq D_k(a_i, b_j)$, and X, Y the events that $R(a_i)$ and $T(b_j)$ are not in W. Then, the event that H_1 is false on $D_k(a_i, b_j)$ is given by the Boolean expression:

$$\varphi \quad = \quad (X \vee Z_1)(Z_1 \vee Z_2)(Z_2 \vee Z_3)(Z_3 \vee Z_4)(Z_4 \vee Z_5)(Z_5 \vee Y)$$

Let $\varphi_{\tau_1\tau_2}$ denote the event "the query is false given that $R(a_i)$ and $T(b_j)$ are labeled τ_1 and τ_2 respectively":

$$\begin{aligned}
\varphi_{11} &= (Z_1 \vee Z_2)(Z_2 \vee Z_3)(Z_3 \vee Z_4)(Z_4 \vee Z_5) \\
\varphi_{12} &= (Z_1 \vee Z_2)(Z_2 \vee Z_3)(Z_3 \vee Z_4)Z_5 \\
\varphi_{21} &= Z_1(Z_2 \vee Z_3)(Z_3 \vee Z_4)(Z_4 \vee Z_5) \\
\varphi_{22} &= Z_1(Z_2 \vee Z_3)(Z_3 \vee Z_4)Z_5
\end{aligned}$$

Denote $p_{11}, p_{12}, p_{21}, p_{22}$ the multivariate polynomials in real variables $z_1, \ldots, z_5$ giving the probabilities of $\varphi_{11}, \varphi_{12}, \varphi_{21}, \varphi_{22}$. We claim that these polynomials factorize as follows:

$$\begin{aligned}
p_{11} &= p_{11}^0 \\
p_{12} &= p_{12}^0 \cdot z_5 \\
p_{21} &= z_1 \cdot p_{21}^0 \\
p_{22} &= z_1 \cdot p_{22}^0 \cdot z_5
\end{aligned}$$

where $p_{11}^0, p_{12}^0, p_{21}^0, p_{22}^0$ are inequivalent irreducible polynomials, having the variables z_2, z_3, z_4 in common. Indeed, start with $p_{11} = p(\varphi_{11})$. Suppose it had two non-trivial factors $p_{11} = f \cdot g$. Since p_{11} is multilinear, f and g do not share any common variables. In that case one could express φ_{11} as $\varphi_{11} = \psi \wedge \nu$, where ψ and ν are Boolean expressions that do not share any common variables. But this is not possible: the expression φ_{11} connects all variables $Z_1, \ldots, Z_5$ and cannot be written as the conjunct of two independent expressions. Thus, p_{11} is irreducible. On the other hand, p_{22} has three factors, z_1, z_5 and the multilinear polynomial in z_2, z_3, z_4 giving the probability of $(Z_2 \vee Z_3)(Z_3 \vee Z_4)$.

The four irreducible polynomials have three common variables z_2, z_3, z_4. Chose x to be any one of them, say $x = z_3$, and set the other variables to some constants to obtain for linear polynomials $f_{11}(x), \ldots, f_{22}(x)$ that have distinct roots.

It is interesting to notice that this does not work if the query Q is not forbidden. For example, consider the query $Q = R(x), S_1(x, y) \vee S_2(x, y), T(y)$ can be separated into two independent parts, and the p-polynomials factorize as:

$$p_{11} = q_1 r_1 \;\; p_{12} = q_1 r_2 \;\; p_{21} = q_2 r_1 \;\; p_{22} = q_2 r_2$$

q_i and r_j cannot have any variables in common. If we choose a variable x that occurs in r_1, r_2 only, then after setting all other variables to constants the polynomials become $a \cdot r_1(x)$, $a \cdot r_2(x)$, $b \cdot r_1(x)$, $b \cdot r_2(x)$ where a, b are constants: thus, the first and third polynomials are equivalent, and so are the second and fourth. We can actually prove a necessary and sufficient condition: Q is not a forbidden query iff a corresponding polynomial factorizes in a way that separates the left from the right.

H.3 Combinatorics Part

For queries Q more complex than H_1 we need to pre-process them, both in order to develop the algorithm, and in order to prove the algebraic properties.

I Part 1 of the Proof: Algebra

I.1 Solving for $det(M) \neq 0$

Here we formalize and generalize Fact H.1. This consists of two theorems. The first is:

Theorem I.1 *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 matrix of polynomials:*

$$M(\bar{G}) \quad = \quad (G_j(\bar{x}_i))_{i,j=1,n} \tag{12}$$

We assume $m = O(1)$. Then there exists an algorithm that runs in time $n^{O(1)}$ and either determines that $det(M) \equiv 0$ (as a multivariate polynomial in $\bar{X}$) or finds values $\bar{V}$ s.t. $det(M(\bar{G})[\bar{V}/\bar{X}]) \neq 0$.

The second theorem is:

Theorem I.2 *Let $\bar{F}(\bar{x}) = (F_1(\bar{x}), \ldots, F_m(\bar{x}))$ be m multivariate polynomials, each in m variables $\bar{x}$. For each $n > 0$, define $(n+1)^m$ polynomials $\bar{G}(\bar{x})$ as follows:*

$$\bar{G} \quad = \quad (\prod_{i=1,m} F_i^{j_i}(\bar{x}))_{0 \leq j_1, \ldots, j_m \leq n} \tag{13}$$

Thus, for each m-tuple $(j_1, j_2, \ldots, j_m) \in \{0, \ldots, n\}^m$ there is one polynomial in $\bar{G}$. Suppose that the Jacobian of the function $\bar{x} \to \bar{F}(\bar{x})$ is not identically zero. Then $det(M(\bar{G})) \not\equiv 0$, where $M(\bar{G})$ is the matrix defined in Eq.(12).

Together, these two theorems prove the following, which generalizes Fact H.1. Suppose we are given m polynomials $F_1(\bar{x}), \ldots, F_m(\bar{x})$ in m variables $\bar{x}$ with a non-zero Jacobian. Define a matrix M of dimensions $(n+1)^m \times (n+1)^m$, where each row is given by $\bar{G}$ in Eq.(13) and distinct rows differ only in the choice of the arguments $\bar{x}$. Thus, the matrix generalizes that given by Eq.(11). Then $det(M)$ is not identically zero, and one can find in PTIME $(n+1)^m$ values $\bar{V}$ for $\bar{x}$ s.t. $det(M)[\bar{V}] \neq 0$.

In the remainder of this section we prove the two theorems. We start with a theorem that we will also need later.

Theorem I.3 *For a multivariate polynomial P in m variables, denote $\mathcal{V}(P)$ the set of points in the m-dimensional complex vector space where P is zero:*

$$\mathcal{V}(P) \quad = \quad \{\bar{v} \mid v \in \mathbf{C}^m, P(\bar{v}) = 0\}$$

This is called the variety *of P. Suppose F is a multilinear polynomial s.t. $\mathcal{V}(F) \subseteq \mathcal{V}(P)$. Then there exists a polynomial G s.t. $P = F \cdot G$*

Proof: Recall Hilbert's Nullstellensatz: for any multivariate polynomials $F_1, \ldots, F_k, P$, if $\mathcal{V}(F_1) \cap \ldots \mathcal{V}(F_k) \subseteq \mathcal{V}(P)$ then there exists $n \geq 1$ and polynomials $G_1, \ldots, G_k$ s.t. $P^n = F_1 \cdot G_1 + \ldots + F_k \cdot G_k$. (In other words, P^n belongs to the ideal generated by $F_1, \ldots, F_k$.) In our case, this implies that there exists G and $n \geq 1$ s.t. $P^n = F \cdot G$. Decompose F into irreducible factors: $F = f_1 f_2 \cdots$ Since F is multilinear, no two factors share any common variables. In particular, all factors are distinct, and, since each factor divides P, it follows that P is divisible by F, proving our claim. $\Box$

To prove Theorem I.1 we need the following two lemmas.

Lemma I.4 *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 $O(n^m T)$ and either determines that $P \equiv 0$ 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. Apply induction hypothesis to this polynomial, to find a set of values $v_1, \ldots, v_{m-1}$ s.t. $Q[x_1/v_1, \ldots, x_{m-1}/v_{m-1}] \neq 0$. If we find such values, then augment them with v_m^i and return $(v_1, \ldots, v_{m-1}, v_m^i)$. If we don't find them, then we know that $Q = P[v_m^i/x_m] \equiv 0$, in other words P is divisible by $x_m - v_m^i$, by Theorem I.3. If the latter case holds for all $n+1$ values for x_m, then we know that the polynomial is identically 0. $\Box$

Recall that n multivariate polynomials $G_1, \ldots, G_n$ are called *linearly independent* if, for any constants $c_1, \ldots, c_n$, if $c_1 G_1 + \ldots + c_n G_n \equiv 0$ then $c_1 = \ldots = c_n = 0$.

Lemma I.5 *The polynomials $G_1, \ldots, G_N$ are linearly dependent iff $det(M(\bar{G})) \equiv 0$ (where M is given by Eq.(12)).*

Proof: The "if" direction follows immediately from the fact that the columns in M are linearly dependent. For the "only if" direction, we prove by induction on n that, if n polynomials $G_1, \ldots, G_n$ are linearly independent, then $det(M) \not\equiv 0$. 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, $det(M') \not\equiv 0$. Hence, there exists values $\bar{V}' = (\bar{v}_1, \ldots, \bar{v}_{n-1})$ s.t. $det(M')[\bar{V}'] \not\equiv 0$. Consider now $det(M)[\bar{V}']$: that is, we substitute $\bar{x}_1, \ldots, \bar{x}_{n-1}$ with the values $\bar{V}'$, and keep only the variables $\bar{x}_N$. Then $det(M(\bar{V}'))$ 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, hence $det(M) \not\equiv 0$, because the polynomials are linearly independent. □

We can now prove Theorem I.1.

Proof: (of Theorem I.1) We proceed by induction on n. Consider the upper left minor M' of dimensions $(n-1) \times (n-1)$. Run the algorithm on the matrix M', and this takes $(n-1)^{O(1)}$ steps. If we determine that $det(M') \equiv 0$, 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. Then $det(M)$ is a polynomial $P(\bar{x}_n)$ in $m = O(1)$ variables $\bar{x}_n$. We also have an oracle for computing P in time $O(n^3)$: given values $\bar{v}$, substitute them in last row of M, then compute $det(M)$ using Gaussian elimination. Thus, we can apply Lemma I.4 to compute a set of values $\bar{v}$ for which this polynomial is non-zero. Return the vector consisting of $\bar{V}'$ concatenated with $\bar{v}$. □

Next, we prove Theorem I.2. For this, we need to recall two notions. A Vandermonde matrix of dimensions $n \times n$ is:

$$V(y_1, \ldots, y_n) \quad = \quad \left(y_i^{j-1}\right)_{i,j=1,n}$$

The Kronecker product of two matrices $A = (a_{ij})$, $B = (b_{kl})$ of dimensions $n_1 \times n_1$ and $n_2 \times n_2$ respecitvely, is the following matrix of dimensions $n_1 n_2 \times n_1 n_2$:

$$A \otimes B \quad = \quad (a_{ij} \cdot b_{kl})_{(i,k) \in [n_1] \times [n_2]; (j,l) \in [n_1] \times [n_2]}$$

Furthermore, if $det(A) \neq 0$ and $det(B) \neq 0$ then $det(A \otimes B) \neq 0$.

Proof: (of Theorem I.2) Note that here the matrix $M(\bar{G})$ has dimension $(1+n)^m \times (1+n)^m$. Consider row i of the matrix: its entries are products of the form $F_1^{j_1} \cdots F_m^{j_m}$ for all possible values $j_1, \ldots, j_m \in \{0, \ldots, n\}$, and all over the same set of variables $\bar{x}_i$.

Since the Jacobian of $\bar{F}(\bar{x})$ is non-zero, the image of $\bar{F}$ includes a closed, m-dimensional cube C. Therefore, for each dimension $i = 1, m$ one can choose a set $Y_i = \{y_{i,0}, y_{i,1}, \ldots, y_{i,n}\}$ of $n+1$ numbers $y_{i,0} < y_{i,1} < y_{i,2} < \ldots < y_{i,n}$ such that all $(n+1)^m$ points can can be formed with these coordinates are in C, i.e. $\bar{Y} =$

$Y_1 \times \ldots \times Y_m \subseteq C$. In other words, for every $\bar{y} \in \bar{Y}$ there exists values $\bar{x}_i$ s.t. $\bar{F}(\bar{x}) = \bar{y}$. By choosing these $(n+1)^m$ values for $\bar{x}_i$ in the definition of $M(\bar{G})$ we obtain the following matrix:

$$\begin{aligned} M(\bar{G}) &= \left(y_1^{j_1} \cdots y_m^{j_m}\right)_{(y_1,\ldots,y_m)\in\bar{Y},(j_1,\ldots,j_m)\in\{0,\ldots,n\}^m} \\ &= V(y_{1,0}, y_{1,1}, \ldots, y_{1,n}) \otimes \cdots \otimes V(y_{m,0}, y_{1,1}, \ldots, y_{m,n}) \end{aligned}$$

Each Vandermonde matrix is non-singular, because the values $y_{i,0}, \ldots, y_{i,n}$ are distinct, hence M is non-singular. □

I.2 Non-zero Jacobian

Here we formalize and generalize Fact H.2.

Theorem I.6 *Let $p_i(x) = a_i x + b_i$, $i = 1, n$ be n linear polynomials in one variable x s.t. $a_i \neq 0$ forall $i = 1, n$, with distinct roots (that is, they are inequivalent polynomials). Define the following n multivariate polynomials, where $x_1, \ldots, x_n$ are n distinct variables:*

$$f_i(x_1, \ldots, x_n) = p_i(x_1) \cdot p_i(x_2) \cdots p_i(x_n)$$

Let $J(\bar{x}) = D(\bar{f})/D(\bar{x})$ be the Jacobian. Let $\bar{v} = (v_1, \ldots, v_n) \in R^n$ be any n distinct values. Then $det(J(\bar{v})) \neq 0$.

Proof: We can assume w.l.o.g. that $a_1 = \ldots = a_n = 1$. Thus, the polynomials can be written as $p_i(x) = x + y_i$, where $y_1, \ldots, y_n$ are distinct values. Then the Jacobian J, and its determinant are:

$$\begin{aligned} J &= \left(\prod_{k\neq j}(x_k + y_i)\right)_{ij} \\ det(J) &= \prod_{i<j}(x_i - x_j)(y_i - y_j) \end{aligned} \tag{14}$$

Thus, $det(J)$ is a polynomial in the variables $\bar{x}$ and $\bar{y}$, which is non-zero whenever both the x_j's are distinct and the y_i's are distinct.

We prove now the identity (14). Denote $det(J) = P(\bar{x}, \bar{y})$; obviously P is a multivariate polynomial in the variables $\bar{x}, \bar{y}$. The degree of any variable x_j is at most $n-1$; indeed, x_j occurs in each column of J with degree 1, except in column j where it does not occur at all. Next, express $det(J)$ in terms of $det(M)$ for the following matrix M:

$$\begin{aligned} M &= \left(\frac{1}{x_j + y_i}\right)_{ij} \\ P(\bar{x}, \bar{y}) &= \prod_{ij}(x_j + y_i) \cdot det(M) \end{aligned}$$

(The determinant of M is called *Cauchy's double alternant* [10].) We notice that M is symmetric in $\bar{x}$ and $\bar{y}$, which implies that the degree of every variable y_i in P is also at most $n-1$. Next, consider what happens to $det(M)$ if we set $x_i = x_j$: we obtain $det(M) = 0$, because two columns become equal. Hence, by Theorem I.3 the polynomial P is divisible by $(x_i - x_j)$, for all $i < j$. Similarly, it is divisible by $(y_i - y_j)$. It follows that $P = c\prod_{i<j}(x_i - x_j)(y_i - y_j)$. It remains to compute the constant c. For that, we compute in $det(J)$ the coefficient of the monomial $m = c \cdot x_1^0 x_2^1 \cdots x_n^{n-1} y_1^0 y_2^1 \cdots y_n^{n-1}$. In the first row, none of the columns 2, 3, ..., n contributes to m, because they have the factor $x_1 + y_1$, thus all terms will contain either x_1 or y_1. Consider now the first column in the first row, $(x_2+y_1)(x_3+y_1)\cdots(x_n+y_1)$. When viewed as a polynomial in y_1, only the free term $x_2 x_3 \cdots x_n$ contributes to m (all others contain y_1). Dividing m by $x_2 x_3 \cdots x_n$, we obtain a new monomial $c \cdot x_2^0 x_3^1 \cdots x_n^{n-2} y_2^0 y_3^1 \cdots y_n^{n-2}$, which is obtained from the lower right minor of $det(J)$; we can conclude inductively that $c = 1$. $\square$

I.3 Irreducible Polynomials

Here we formalize and generalize Fact. H.3. Recall that two multivariate polynomials p and q are called *equivalent* if there exists a constant $c \neq 0$ such that $p \equiv c \cdot q$.

Theorem I.7 *Let $p_1(\bar{x}), p_2(\bar{x}), \ldots, p_n(\bar{x})$ be n multi-linear polynomials in variables $\bar{x}$. Suppose that, for each i, p_i has an irreducible factor p_i^0 s.t. $p_1^0, \ldots, p_n^0$ are inequivalent, and all depend on a common variable x. Let $\bar{y} = \bar{x} - \{x\}$ denote all the other variables. Then there exists real values $\bar{v}$, s.t. denoting $f_i = p_i[\bar{v}/\bar{y}]$, the linear polynomials $f_1(x), \ldots, f_n(x)$ have the following properties: (a) each of them depends on x, (b) they are inequivalent (i.e. have distinct roots).*

Proof: Write $p_i^0(x) = a_i{*}x - b_i$ where a_i, b_i are polynomials in the variables $\bar{y}$. We have $a_i \not\equiv 0$ because p_i^0 depends on x. We prove that for any distinct i, k, the polynomials $a_i b_k$ and $a_k b_i$ are not identical. Suppose otherwise, i.e. $a_i b_k \equiv a_k b_i$. Then we can factorize them like this:

$$\begin{aligned} a_i &= u \cdot v \\ b_k &= w \cdot z \\ a_k &= u \cdot z \\ b_i &= v \cdot w \end{aligned}$$

Then $p_i^0(x) = u \cdot v \cdot x + v \cdot w = v \cdot (u \cdot x + w)$ and $p_k^0(x) = z \cdot (u \cdot x + w)$. We know that $u \not\equiv 0$, because $a_i \not\equiv 0$ (and $a_k \not\equiv 0$). Since both p_i^0 and p_k^0 are irreducible, we must have both v and z constant polynomials. But then p_i^0 and p_k^0 are equivalent, which is a contradiction. This proves the claim that $a_i b_k \not\equiv a_k b_i$.

Consider the following polynomial:

$$F(\bar{y}) = (\prod_i a_i) \times (\prod_{ik}(a_i{*}b_k - a_k{*}b_i))$$

Here $F(\bar{y})$ is a multivariate polynomial in variables $\bar{y}$ that is not identically zero. Hence there are values $\bar{y} = \bar{v}$ s.t. $F[\bar{v}/\bar{y}] \neq 0$. We check that these values satisfy the two conditions in the theorem. Indeed, we have $a_i[\bar{v}/\bar{y}] \neq 0$, hence $f_i(x) = a_i[\bar{v}/\bar{y}] \cdot x - b_i[\bar{v}/\bar{y}]$ depends on x, thus (a) is satisfied. To check (b) it suffices to note that, forall $i \neq k$, $a_i[\bar{v}/\bar{y}]b_k[\bar{v}/\bar{y}] \neq a_i[\bar{v}/\bar{y}]b_i[\bar{v}/\bar{y}]$. $\Box$

J Part 2 of the Proof: Combinatorics

The reduction from a Signature Counting problem to a forbidden query that we sketched in Sec. H does not work for all queries. Instead, in this section we describe several ways to simplify the query until it reaches a *normal form*, where the reduction from SC works.

For example, consider the following variation of H_1:

$$Q \quad = \quad R_1(x), R_2(x), S_1(x,y) \vee S_1(x,y), S_2(x,y) \vee S_2(x,y), T_1(y), T_2(y)$$

A naive way to attempt this reduction is to consider four left labels, corresponding to all possible truth assignments to the events $R_1(a), R_2(a)$, and similarly four right labels. But this a reduction from the 4,4-SC problem fails, because we do not obtain 16 distinct irreducible polynomials: the truth assignments $(0,1)$, $(1,0)$, and $(1,1)$ to $R_1(a), R_2(a)$ lead to the same expressions, and same for $T_1(a), T_2(a)$, and there are only four in-equivalent polynomials instead of 16. Instead, we prove hardness for Q by making all tuples $R_2(a)$ and $T_2(a)$ deterministic: then Q becomes H_1.

We assume throughout this section that the query Q is minimized. We transform a forbidden query into two ways: first we make it "long", then we "normalize" it.

Let l denote the leveling function for $Q = \bigvee_i q_i$. Denoting the variables in $l^{-1}(1)$ with $x, x_1, x_2, \ldots$ and those in $l^{-1}(2)$ with $y, y_1, y_2, \ldots$. Every atom has at most one x- and one y-variable. A variable is a *root variable* in q_i if it occurs in all atoms of q_i; q_i is *hierarchical* if it has a root variable.

We start by showing that we can discard non-hierarchical queries, by giving a direct proof that every non-hierarchical query is hard. The proof is similar to the proof given in [5] for the fact that non-hierarchical conjunctive queries are hard; we included it here for completeness.

Theorem J.1 *If Q is minimzed and has a non-hierarchical component q, then computing $p(Q)$ on probabilistic databases is hard for $\#P$.*

Proof: By reduction from the PP2DNF problem. Let $\Phi = (X, Y, E)$ be a bipartited graph representing a PP2DNF instance $\bigvee_{(a,b)\in E} a \wedge b$.

Since q is connected and non-hierarchical, there exists two variables x, y and three atoms $S_1(x, -), S_2(x, y), S_3(-, y)$ s.t. S_2 contains both x, y, S_1 contains x and possibly a second variable, and S_3 contains y and possibly a second variable. For every other variable z in Q let c_z be a fresh constant, unique for z. For every pair of constants $a \in X$ and $b \in Y$ denote $D(a,b)$ the structure consisting of all tuples in $q[a/x, b/y, \ldots c_z/z \ldots]$. Define $D = \bigcup_{(a,b)\in E} D(a,b)$. Set the probabilities of all unary tuples $S_1(a,-)$, $S_3(-)$ to 1/2, and the probabilities of all other tuples to 1. We prove that $p_D(Q)$ is precisely the probability that Φ is true when each Boolean variable is set to true with probability 1/2.

Let $W \subseteq D$ be a world. W corresponds uniquely to a truth assigment of the Boolean variables in Φ, namely a is true iff $S_1(a,-) \in W$ and similarly b is true iff $S_3(-,b) \in W$. We prove that $W \models Q$ iff the corresponding truth assignment makes Φ true. Indeed, let θ be a valuation from some component q_1 in Q to W. Compose this valuation with the homomorphism $W \to q$ that maps a to x, b to y, and each constant c_z to the variable z: we obtain a homomorphism $q_1 \to q$. Thus, q_1 is q and the homomorphism $q_1 \to q$ is an isomorphism. Thus, the valuation θ must contain both tuples $S_1(a,-)$ and $S_3(-,b)$ where $(a,b) \in E$, and therefore it corresponds to an assigment making Φ true. The converse is straightforward: if the assignment θ makes Φ true then there exists $(a,b) \in E$ s.t. both $S_1(a,-)$ and $S_3(-,b)$ are in W, hence the mapping $x \mapsto a$ and $y \mapsto b$ is a valuation from q to W. $\Box$

Thus, from now on we will assume w.l.o.g. that Q is hierarchical. Thus, each component has a root variable, which is either x, or y, or both, and we have:

Definition J.2 *Let q_i be a component in Q.*

- *q_i is a* left component *if it has an x-root but no y-root variable.*
- *q_i is a* right component *if it has a y-root but no x-root variable.*
- *q_i is a* center component *if it has both an x-root and a y-root variable.*

A center component has exactly two variables, x, y, both root variables.

Definition J.3 *A relational symbol S (binary or unary) is called a* left symbol *if it occurs in a left component; it is called a* right symbol *if it occurs in a right component. A symbol that is neither left nor right is called a* center symbol*.*

Definition J.4 *Let q be a component of Q. A* subcomponent *of q is a maximal subset of atoms that contain exactly the same set of variables.*

For example, consider the component q:

$$q = R(x), S_1(x,y_1), S_2(x,y_1), S_1(x,y_2), S_3(x,y_2)$$

it has three subcomponents s_1, s_2, s_3:

$$s_1 = R(x) \quad s_2 = S_1(x,y_1), S_2(x,y_1) \quad s_3 = S_1(x,y_2), S_3(x,y_2)$$

J.1 Long Forbidden Queries

Definition J.5 *The forbidden query Q is called* long *if no component contains both a left and a right symbol.*

Example J.6

$$\begin{aligned}
H_1 &= R(x), S(x,y) \vee S(x,y), T(y) \\
H_2 &= R(x), S_1(x,y) \vee S_1(x,y), S_2(x,y) \vee S_2(x,y), T(y) \\
H_3 &= R(x), S_1(x,y) \vee S_1(x,y), S_2(x,y) \vee S_2(x,y), S_3(x,y) \vee S_3(x,y), T(y) \\
& \ldots
\end{aligned}$$

Every H_k for $k \geq 3$ is long; H_1, H_2 are short.

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

Query Q a forbidden query because both S_1, S_2 are both left and right symbols; but Q is short.

Our reduction from the SC-Problem only works when Q is long. In this section we prove the following:

Proposition J.7 *For every hierarchical forbidden query Q there exists a long forbidden query Q^m such that the evaluation problem of Q^m on 2-leveled structures can be reduced to the evaluation problem of Q on 2-leveled structures.*

Proof: Let Q be a (short) forbidden query. Let n_x be the maximum number of x_i variables, and n_y the maximum number of y_i variables in any component q. Fix a number $m \geq 2$: for the claim in the proposition it suffices to set $m = 2$, but we describe the proof for general $m \geq 2$.

Given two constants d, c, construct the following 2-leveled structure $G(d, c)$, which we call the *template*:

- The level 1 constants are: $X = X_1 \cup X_2 \cup \ldots \cup X_m$ where $X_1 = \{d\}$ and forall $i = 2, m$, $|X_i| = n_x$.
- The level 2 constants are: $Y = Y_1 \cup Y_2 \cup \ldots \cup Y_m$ where forall $j = 1, m-1$, $|Y_j| = n_y$, and $Y_m = \{c\}$.
- For every left unary symbol $R(x)$, the structure contains all tuples of the form $R(u)$ for $u \in X$.
- For every right unary symbol $T(y)$, the structure contains all tuples of the form $T(v)$ for $v \in Y$.
- For every binary symbol $S(x, y)$, the structure contains all tuples for the form $S(u, v)$ where $u \in X_i$ and $v \in Y_i$, or $u \in X_{i+1}$ and $v \in Y_i$.

One can think of the template $G(d, c)$ as a bipartite graph, where both the left nodes and the right nodes are partitioned into m sets. The subgraph restricted to X_i, Y_i is complete; the subgraph restricted to X_{i+1}, Y_i is also complete; and there are no other edges. We call d and c the *distinguished nodes* of the template.

We will define Q^m over a new vocabulary, with the following property. For every 2-leveled probabilistic database D^m over the vocabulary of Q^m, there exits a 2-leveled probabilistic database D over Q's vocabulary such that $p_D(Q) = p_{D^m}(Q^m)$.

We describe the new vocabulary for Q^m together with the translation $D^m \mapsto D$. Recall that D^m must be leveled: : let $A = l^{-1}(1)$ and $B = l^{-1}(2)$. Then D consists of the union of structures $G(a, b)$ forall $a \in A, b \in B$. More precisely, for every pair of constants a, b and every template node $u \in X \cup Y$, D contains a distinct constant $u^{a,b}$ *except* that we equate $c^{a,b} = a$ and $d^{a,b} = b$. Thus, two different substructures $G(a, b)$ and $G(a, b')$ share the node a and nothing else; similarly $G(a, b)$ and $G(a', b)$ share b and nothing else. Now we describe the new vocabulary for Q^m and the translation $D^m \mapsto D$:

- For every left unary symbol $R(x)$ for Q, there is a unary symbol $R(x)$ for Q^m. Every tuple $R(a)$ in D^m is copied to a tuple $R(a)$ in D. Moreover, for every non-distinguish node $u \in X$, there is a binary symbol $R^u(x, y)$ in the vocabulary for Q^m. Every tuple $R^u(a, b)$ in D^m is copied to a tuple $R(u^{a,b})$ in D.

- Similarly for a right unary unary symbol $T(y)$ for Q: there is a unary symbol $T(y)$ for Q^m and there are binary symbols $T^v(x, y)$, one for each non-distinguish node $v \in Y$.

- For every tuple $S(u, v)$ in the template $G(c, d)$ there exists a binary symbol $S^{u,v}(x, y)$ in the vocabulary for Q^m. Every tuple $S^{u,v}(a, b)$ is copied to a tuple $S(u^{a,b}, v^{a,b})$ in D. Here we apply the convention that, if $u = d$ then $u^{a,b} = a$, and if $v = c$ then $v^{a,b} = b$.

We define now the translated query Q^m. Let q be a component in Q, and let $\eta : q \to G(d, c)$ be any valuation from q to the template $G(d, c)$. We define q^η to be the following component in Q^m:

- Suppose q has a root variable x (thus, it is a left component or a center component) and $\eta(x) = d$ (the distinguished node). In that case we copy each left unary atom $R(x)$ in q to q^η (for all left unary symbols R), and for each binary atom $S(x, y_i)$ in q we insert a binary atom $S^{d,\eta(y_i)}(x, y_i)$ in q^η.

- Suppose q has a root variable y and $\eta(y) = c$ (the distinguished node). The we proceed similarly to the case above.

- Otherwise, q^η will have exactly two variables x, y. For each left unary atom $R(x_i)$ in q we create a binary atom $R^{\eta(x_i)}(x, y)$; for each binary atom $S(x_i, y_j)$ we create a binary atom $S^{\eta(x_i),\eta(y_j)}(x, y)$; and for each right unary atom $T(y_j)$ we create a binary atom $T^{\eta(y_j)}(x, y)$ in q^η.

Finally, define $Q^m = \bigcup_{q,\eta} q^\eta$.

Note that every component q^η has either a root variable x, or a root variable y, or both. Thus, Q^m is hierarchical.

We first prove that Q^m is a long query. For that we notice that a left component in Q consists of only the unary symbols $R(x)$, and of binary symbols S^{uv}, where

$(u, v) \in X_1 \times Y_1$; thus, these are the only left symbols. Similarly the right symbols are T, and all symbols S^{uv} where $(u, v) \in X_m \times Y_m$. On the other hand, every component contains symbols whose level differs by at most one. That is, if q^η contains both symbols $S_1^{u_1 v_1}$ and $S_2^{u_2 v_2}$ and $(u_1, v_l) \in X_{i_1} \times Y_{j_1}$, $(u_2, v_2) \in X_{i_2} \times Y_{j_2}$, then $i_1 = i_2$ and $|j_1 - j_2| \leq 1$, or $|i_1 - i_2| \leq 1$ and $j_1 = j_2|$. It follows that, if $m \geq 2$, then no component q^η may contain both a left symbol and a right symbol.

Next we prove that Q^m is forbidden. This follows from the following four claims.

Claim 1: if q is a left component in Q, S is a left symbol in q, and $(c, u) \in X_1 \times Y_1$ is an edge, then there exists η s.t. q^η is a left component, contains the symbol S^{cu}, and is non-redundant in Q. Indeed, let $S(x, y_i)$ be any atom in q containing S. Then any injective mapping η from the variables in q to the nodes of the template $G(d, c)$ that maps y_i to u satisfies the conditions. To see that this is a left component, notice that q either has two distinct y-variables, or has a unary predicate $R(x)$, and these properties are preserved in q^η. To see that it is non-redundant, we notice that any homomorphism $q_1^{\eta_1} \rightarrow q^\eta$ can be extended to a homomorphism $q_1 \rightarrow q$, contradicting the fact that Q is minimal.

Claim 2: symmetrically, if q is a right component in Q, S is a right symbol in q, and $(u, d) \in X_m \times Y_m$ is an edge, then there exists η s.t. q^η is a right component, contains the symbol S^{ud}, and is non-redundant in Q. The proof is similar to claim 1 and omitted.

Claim 3: if q is a central component in Q, S a symbol in q, and (u, v) an edge in the template, then there exists a non-redudant component q^η that contains the symbol S^{uv}. This follows immediately from the mapping $\eta(x) = u$, $\eta(y) = v$, and all atoms $S(x, y)$ in q become $S^{uv}(x, y)$ (same u, v for all atoms).

Claim 4: if q is a left component in Q, then for any two consecutive edges $(u, v) \in X_i \times Y_{i-1}$ and $(u, w) \in X_i \times Y_i$, and every two (not necessarily distinct) binary symbols S_1, S_2 in q, the symbols S_1^{uv} and S_2^{uw} are connected in the co-occurrence graph. The proof considers two cases. The first is when there exists two atoms with the relation symbols S_1, S_2 that have distinct y-variables: $S_1(x, y_1)$, $S_2(x, y_2)$. Then let η be $x \mapsto u$, $y_1 \mapsto v$, $y_2 \mapsto w$, and every other variable y_i is mapped to another distinct constant in either Y_{i-1} or Y_i. The resulting query q^η has the form $S_1^{uv}(x, y), S_2^{uw}(x, y) \ldots S_j^{uz_i}, (x, y) \ldots$. We show that it is non-redudant. If there were a homomorphism $q_1^{\eta_1} \rightarrow q^\eta$ then superscripts of the relation name in $q_1^{\eta_1}$ must have the same shape $uv, uw, \ldots uz_i \ldots$ Whenever we have a distinct node z_i, there is a distinct variable y_i in q_1. This allows us to construct a homomorphism $q_1 \rightarrow q$, contradicting the fact that Q is minimal. The second case is when $S_1(x, y), S_2(x, y)$ have the same y variable. If there exists a unary predicate $R(x)$ in q, then we consider two mappings η: the first maps x, y to u, v, and hence maps $R(x), S_1(x, y)$ to $R^u(x, y), S_1^{uv}(x, y)$; the second maps x, y to u, w and hence maps the atoms $R(x), S_2(x, y)$ to $R^u(x, y), S_2^{uw}(x, y)$. Thus, S_1^{uv} and S_2^{uw} are connected in the co-occurrence graph. If there is no unary symbol, then there is another binary symbol using a different variable y: $S_3(x, y_3)$. Here we consider three mappings: $x, y, y_3 \mapsto u, v, w$, resulting in the symbols $S_1^{uv}, S_2^{uv}, S_3^{uw}$, then $x, y, y_3 \mapsto u, w, v$, resulting in the symbols $S_1^{uw}, S_2^{uw}, S_3^{uv}$. To connect them, we consider a third mapping η_3: $x, y, y_3 \mapsto u, v, v$, resulting in $S_1^{uv}(x, y), S_2^{uv}(x, y), S_3^{uv}(x, y)$. If q^{η_3} is non-redundant, then we are done. Otherwise, there exists a homomorphism $r^{\eta'} \rightarrow q^{\eta_3}$. Then $r^{\eta'}$ forms the needed connection.

Claim 5: symmetrically, if q is a right component in Q, then for any two consecutive edges $(u, v) \in X_i \times Y_i$ and $(w, v) \in X_{i+1} \times Y_i$, and every two (not necessarily distinct) binary symbols S_1, S_2 in q, S_1^{uv} and S_2^{wv} are connected in the co-occurrence graph of Q^m.

Combining claims 1-4 allows us to prove that there exists a path in the co-occurrence graph of Q^m from a left to a right symbol. Consider a path in Q from a left symbol S_1 to a right symbol S_k. Start in Q^m with a left component that contains the left symbol S_1^{du}, for any $u \in Y_1$ (claim 1). Using claim 3, we argue that it is connected to S_k^{cu}; using claim 5 we connect this to S_k^{wu}, where $w \in X_2$; using claim 3 we connect it to S_1^{wu}, then claim 1 to connect it to S_1^{wv}, for $v \in Y_2$, etc. Continuing this way we end at S_k^{uc} where $c \in X_m$.

Finally, we prove that $p_D(Q) = p_{D^m}(Q^m)$. For that, we claim the following. Let $W^m \subseteq D^m$ be a possible world, and $W \subseteq D$ be its corresponding possible world in D. Then $W \models q$ iff there exists η s.t. $W^m \models q^\eta$: this implies that $p_D(Q) = p_{D^m}(Q^m)$. It remains to prove the claim.

Suppose $W \models q$. Then there exists a valuation θ from $Var(q)$ to the active domain of W s.t. all atoms in q are mapped to tuples in W. We will define η by considering three cases, and prove that $W^m \models q^\eta$.

Case 1 The image of θ does not contain any constants $a \in A$ or $b \in B$. Then $Im(\theta)$ is contained in a single template $G(a, b)$. Every variable x_i is mapped to some constant $u^{a,b} \in X$ and every y_j is mapped to $v^{a,b} \in Y$. Define $\eta(x_i) = u$ and $\eta(y_j) = v$. Consider a binary atom $S^{uv}(x, y)$ in q^η. This means that there exists an atom $S(x_i, y_j)$ in q s.t. $u(x_i) = u$ and $u(y_j) = v$. This means that there exists a tuple $S(u^{a,b}, v^{a,b})$ in W: the corresponding tuple $S^{uv}(a, b)$ must be in W^m. Similarly, consider a binary atom $R^u(x, y)$ in q^η. This means that there exists an atom $R(x_i)$ in q s.t. $\eta(x_i) = u$; hence there exists a tuple $R(u^{a,b})$ in W; therefore the corresponding tuple $R^u(a, b)$ must exists in W^m. Similarly for a binary atom $T^u(x, y)$.

Case 2 The image of θ contains some constant a. Then it cannot contain a constant b, and hence $Im(\theta)$ is included in $D(a, b_1) \cup D(a, b_2) \cup \ldots$ In this case q must be a left or center component, hence it has a unique variable x, and $\theta(x) = a$ For each variable y_i, assume $\theta(y_i) = v^{a,b_j}$. Define $\eta(y_i) = v$. First, any unary predicate $R(x)$ in q is the same unary predicate in q^η, and since $R(a) \in W$ we know that it is also in W^m. Now consider a binary predicate $S(x, y_i)$. This is mapped to $S(a, v^{a,b}) \in W$. The corresponding atom in q^η is $S^{dv}(x, y)$; and this corresponds to the tuple $S^{dv}(a, b)$ in W^m, which we know must be present since it corresponds to $S^(a, v^{ab}) \in W$.

Case 3 The image of θ contains some constant b. This case is similar to the previous one and is omitted.

Conversely, suppose that $W^m \models q^\eta$ for some η, thus there exists a valuation θ^m from q^η to W^m. We prove that $W \models q$. Since q^η is hierarchical, it has either a root variable x or a root variable y. Assume the former. We define the valuation θ as follows. For a binary symbol $S(x, y_i)$, consider the corresponding binary symbol

$S^{uv}(x, y_i)$ in q^m: this is mapped to $S^{uv}(a, b_j)$ in W^m. We define θ to map $S(x, y_i)$ to $S((\eta(x))^{a,b_j}, \eta(y_i)^{a,b_j})$. For a unary symbol $R(x)$, then we either map it to $R(a)$ (if $\eta(x) = d$) or to $R((\eta(x))^{a,b_j})$ (if $\eta(x)$ is a non-distinguished node). $\Box$

The construction in the proof looks similar to the blocks $D_k(a_i, b_j)$ that we illustrated in Sec. H, but they are in fact quite different. For example, here we change the vocabulary, while in the construction in Sec. H we keep the same vocabulary. These two constructions are different, and are used at different places in the proof.

Example J.8 Consider H_1, which is not a long query. Taking $m = 2$ (as in the proof of Prop.J.7) we transform it into a long query:

$$\begin{aligned} H_1 &= R(x), S(x,y) \vee S(x,y), T(y) \\ H_1^2 &= R(x), S^{11}(x,y) \vee S^{11}(x,y), T^1(x,y) \vee \\ &\quad R^2(x,y), S^{21}(x,y) \vee S^{21}(x,y), T^1(x,y) \vee \\ &\quad R^2(x,y), S^{22}(x,y) \vee S^{22}(x,y), T(y) \end{aligned}$$

The left symbols are R and S^{11}; the right symbols are S^{22} and T. They do not co-occur, hence the query is long. The following is a path in the co-occurrence graph, proving that the query is forbidden:

$$S^{11}, T^1, S^{21}, R^2, S^{22}$$

Consider now:

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

Take $m = 2$ and define $X_1 = \{1\}$, $X_2 = \{2, 2'\}$. Similarly $Y_1 = \{1, 1'\}$, $Y_2 = \{2\}$. Then:

$$
\begin{aligned}
Q^2 \quad = \quad & \bigvee_{u,v \in Y_1} S_1^{1u}(x,y_1) S_2^{1v}(x,y_2) \\
& \vee \bigvee_{u \in Y_1} S_1^{1u}(x,y) S_2^{1u}(x,y) \;\; /\!* \text{ redundant } *\!/ \\
& \vee \bigvee_{u \in X_2, v \in Y_1} S_1^{1v}(x,y), S_2^{uv}(x,y) \\
& \vee \bigvee_{u \in X_2, v \in Y_1} S_1^{uv}(x,y), S_2^{1v}(x,y) \\
& \vee \bigvee_{u \in X_2, v \in Y_1} S_1^{uv}(x,y), S_2^{uv}(x,y) \\
& \vee \bigvee_{u \in X_2, v \in Y_1} S_1^{uv}(x,y), S_2^{u2}(x,y) \\
& \vee \bigvee_{u \in X_2, v \in Y_1} S_1^{u2}(x,y), S_2^{uv}(x,y) \\
& \vee \bigvee_{u \in X_2} S_1^{u2}(x,y), S_2^{u2}(x,y) \;\; /\!* \text{ redundant } *\!/ \\
& \vee \bigvee_{u,v \in X_2} S_1^{u2}(x_1,y), S_2^{v2}(x_2,y)
\end{aligned}
$$

The left symbols are S_k^{11}, $S_k^{11'}$ for $k = 1,2$; the right symbols are S_k^{22}, $S_k^{2'2}$, for $k = 1,2$. They do not co-occur in any component, thus the query Q^2 is long.

From now on we will assume wlog that the query is long, and will we prove hardness only for long queries.

As a final remark, we note that the construction in the proof of Prop. J.7 also works for some non-hierarchical queries. For example, consider H_0: we can transform it into a long query by taking $m = 3$:

$$
\begin{aligned}
H_0 &= R(x), S(x,y), T(y) \\
H_1^3 &= R(x), S^{11}(x,y), T^1(x,y) \\
& \quad R^2(x,y), S^{21}(x,y), T^1(x,y) \\
& \quad R^2(x,y), S^{22}(x,y), T^2(x,y) \\
& \quad R^3(x,y), S^{32}(x,y), T^2(x,y) \\
& \quad R^3(x,y), S^{33}(x,y), T(y)
\end{aligned}
$$

However, if applied to arbitrary non-hierarchical queries Q^m then it doesn't work. An example is:

$$\begin{aligned} Q &= S_1(x_1,y_1),S_2(x_1,y_2),S_3(x_2,y_1),S_4(x_2,y_2) \vee \\ &\quad S_1(x,y),S_2(x,y) \vee S_1(x,y),S_3(x,y) \vee S_1(x,y),S_4(x,y) \vee \\ &\quad S_2(x,y),S_3(x,y) \vee S_2(x,y),S_4(x,y) \vee S_3(x,y),S_4(x,y) \end{aligned}$$

This is a forbidden query because the first component is non-hierarchical (hence all its symbols are both left and right). If we compute Q^m as in Prop. J.7, then the resulting query is no longer forbidden. This is because whenever we equate either $x_1 = x_2$ or $y_1 = y_2$ in the first component, it becomes redundant.

J.2 Normalizing Forbidden Queries

We further simplify the query by repeatedly applying three simple rewrite rules; we call the resulting query *normalized*. In this section we establish several properties of normalized queries, which we later use to show that certain multivariate polynomials are irreducible.

The three rewrite rules are the following. Given a conjunctive query q, denote $q[S = 0]$ to be q if it doesn't contain the symbol S, and $\emptyset$ if it contains S. Denote $q[S = 1]$ to be the query obtained from q by removing all atoms that contain S, then minimizing the resulting query. If Q is a disjunctive query, then we write $Q[S = 0]$ and $Q[S = 1]$ the query obtained by replacing each component q with $q[S = 0]$ or $q[S = 1]$ respectively, then minimizing the query.

Definition J.9 *Let Q, Q' be two 2-leveled queries, over two different vocabularies. We say that Q rewrites to Q', in notation $Q \to Q'$, if Q' can be obtained from Q by one of the following steps.*

Empty rewriting *Set a unary or binary symbol S_i to 0 (empty), thus $Q' \equiv Q[S_i = 0]$. The vocabulary of Q' is that of Q less S_i.*

Deterministic rewriting *Set a unary or binary symbol S_i to 1 (deterministic), thus $Q' \equiv Q[S_i = 1]$. The vocabulary of Q' is that of Q less S_i.*

Left unary rewriting *Let $c_1, \ldots, c_k$ be fresh constants. For each subcomponent $s(x, y)$ that contains only left symbols, replace it with $s(x, y) \vee s(x, c_1) \vee \ldots \vee s(x, c_k)$, then replace each $S_i(x, c_j)$ with a new unary relation name $R_{ij}(x)$. After this step we need to convert Q' to DNF, then minimize it. The vocabulary of Q' consist of that of Q, plus all new unary symbols R_{ij}.*

Right unary rewriting *Symmetrically.*

The left unary rewriting differs from the first rewriting in Def. 6.3 in that it allows the subcomponent $s(x, y)$ to exists. In fact, because of that, the rewriting always results in a query that is equivalent to Q: however, before we minimize the rewritten query, we will apply an empty rewriting, and thus remove some of the subcomponents $s(x, y)$.

Proposition J.10 *If $Q \to Q'$ then the query evaluation problem for Q' can be reduced to that for Q.*

Proof: Let D' be a probabilistic database instance for Q'. For an empty rewriting, add to D' a new relation symbol S_i which is empty (i.e. it has no tuples; equivalently, all its tuples have probability 0). For a deterministic rewriting, add a new relation symbol S_i with all tuples $S_i(a, b)$ over the active domain of D', and with probability 1. For a unary rewriting, replace each tuple $R_{ij}(a)$ with $S_i(a, c_j)$. Call D the new probabilistic database. In all three cases, the probability of Q over D is equal to the probability of Q' over D'. $\Box$

Thus, if Q' is hard, then Q is hard. For example, recall the query at the beginning of this section:

$$Q \quad = \quad R_1(x), R_2(x), S_1(x, y) \vee S_1(x, y), S_2(x, y) \vee S_2(x, y), T_1(y), T_2(y)$$

By rewriting $Q[R_2 = 1]$ we obtain a query that is isomorphic to H_1, and, therefore, Q is hard.

Notice that there are infinitely long rewritings, because of the unary rewriting. To avoid this, we introduce a simple restriction:

Definition J.11 *We say that Q* strictly rewrites *to Q', $Q \Rightarrow Q'$, if $Q \to Q'$ using an empty or a deterministic rewriting, or if there exists Q'' s.t. $Q \to Q''$ with a unary rewriting, and $Q'' \to Q'$ using an empty rewriting on a binary symbol.*

Consider the transitive closure $\overset{*}{\Rightarrow}$; it is always terminating, because every rewriting either results in strictly fewer binary symbols, or has the same number of binary symbols and strictly fewer unary symbols.

Definition J.12 *A forbidden query Q is in* normal form *if there is no forbidden query Q' such that $Q \Rightarrow Q'$.*

Obviously, every forbidden query rewrites to some (not necessarily unique) forbidden query in normal form.

We describe now some key properties of normalized queries. Recall that a query Q is forbidden if the co-occurrence graph contains a path $S_0, S_1, S_2, \ldots, S_n$ where S_0 is a left symbol, S_n is a right symbol, and any pair of symbols S_{i-1}, S_i co-occur in some component q_i. Equivalently, consider the dual graph: the nodes are components q_i, and two components are connected by an edge if they share a common symbol.

Definition J.13 *A* left-to-right *path, or LR-path, from a left symbol S to a right symbol S' is a path $P = (q_1, q_2, \ldots, q_n)$ where q_1 contains S and q_n contains S'.*

Obviously, Q is forbidden iff there exists a left-to-right path. We call an LR-path *strict* if no subpath is an LR-path:

Definition J.14 *$P = (q_1, \ldots, q_n)$ is a* strict *LR path if q_1 contains a left symbol, q_n contains a right symbol, and for all $i = 2, n-1$, q_i contains only central symbols.*

If the query Q is long, then $n \geq 2$ because no left and right symbols co-occur.

Proposition J.15 *If Q is a forbidden query then there exists a strict LR-path.*

Proof: Any LR path of minimal length is a strict LR path. □

Definition J.16 *A component q_i is called* useful *if there exists a strict LR path containing q; otherwise it is called* useless*, or* superfluous*. A symbol S is called useful if it occurs in some useful component; otherwise it is called useless, or superfluous.*

Example J.17 Consider the following query in normal form:

$$\begin{aligned} K &= S(x,y_1), S_1(x,y_1), S(x,y_2), S_2(x,y_2), \\ &\vee \; S_1(x,y), S_2(x,y), C(x,y) \\ &\vee \; C(x,y), S_1'(x,y), S_2'(x,y) \\ &\vee \; S'(x_1,y), S_1'(x_1,y), S'(x_2,y), S_2'(x_2,y) \end{aligned} \tag{15}$$

The left symbols are S, S_1, S_2; the right symbols are S_1', S_2', S'; symbol C is a center symbol. Components 2 and 3 are useful; components 1 and 4 are superfluous. Symbols S_1, S_2, C, S_1', S_2' are useful; symbols S, S' are superfluous.

Consider the following query in normal form:

$$\begin{aligned} K' &= S(x,y_1), S_1(x,y_1), S(x,y_2), S_2(x,y_2), \\ &\vee \; S(x,y), S_1(x,y), D(x,y) \\ &\vee \; S_1(x,y), S_2(x,y), C(x,y), D(x,y) \\ &\vee \; C(x,y), S'(x,y) \\ &\vee \; S'(x,y), T(y) \end{aligned}$$

S, S_1, S_2 are left symbols; S', T are right symbols; D, C are center symbols. Components 3,4 are useful; components 1,2,5 are superfluous; symbols S_1, S_2, C, D, S' are useful; symbols S, T are superfluous.

All left components, and all right components are superfluous, and all unary symbols are superfluous; the examples above show that the converse fails.

Proposition J.18 *Let Q be a normalized query and $P = (q_1, q_2, \ldots, q_n)$ be any strict LR-path. Then every central symbol C occurs in P, i.e. there exists i s.t. q_i contains C. In particular, every central symbol is useful.*

For example, in K' above both central symbols C, D are useful.

Proof: Let C be a central symbol that does not occur in the path. The by setting $C = 0$ we rewrite the query to $Q[C = 0]$ which is still a forbidden query because it continues to have the path P; this contradicts the fact that Q is normalized. □

Let UL denote the set of left components that contain some useful left symbol, and similarly UR is the set of right components containing some useful right symbol. Thus, every component in UL is useless, and contains a useful symbol; similarly for UR.

Proposition J.19 (Superfluous is Ubiquitous) *If Q is normalized and S is a superfluous left symbol, then every component $q \in UL$ contains S.*

Proof: Suppose not: set the symbol to empty, $S = 0$. Let $q \in UL$ be a left, useful component in Q that does not contain S. Then Q continues to exists in $Q[S = 0]$. Since $q \in UL$, it contains a useful left symbol, S_1. Hence, there exists a strict path $P = (q_1, q_2, \ldots)$ in Q from S_1 to a right symbol. None of q_i contains S, otherwise S would be useful. Thus, the same path exists in $Q[S = 0]$, so $Q[S = 0]$ is still forbidden, contradicting the fact that Q is in normal form. $\Box$

Definition J.20 *Let S be a symbol and q be a component. Denote $s_q(S)$ the set of all subcomponents of q that contain S. (If q does not contain S, then $s_q(S) = \emptyset$.)*

Definition J.21 *Let S be a symbol. Denote $c(S)$ the set of useful components that contain S. In particular, S is useful iff $c(S) \neq \emptyset$.*

The following is a key technical lemma.

Lemma J.22 *Let Q be a normalized query. Let S, S_1 be two left symbols, and q any left component. Then one of the following must hold: $s_q(S_1) \subseteq s_q(S)$ or $c(S_1) \subseteq c(S)$.*

We illustrate before giving the proof with:

$$S(x, y_1)S_2(x, y_1), S_1(x, y_2) \vee S_1(x, y)S_2(x, y)C(x, y) \vee C(x, y)S'(x, y) \vee S'(x, y)T(y)$$

Here S and S_1 violate the lemma. S_1 is useful, $c(S_1) = \{q_2\}$ (where $q_2 =$ the second component), S is useless, $c(S) = \emptyset$, hence $c(S_1) \not\subseteq c(S)$. On the other hand in q_1, S appears in the first subcomponent, and S_1 in the second, hence we have $s_{q_1}(S_1) \not\subseteq s_{q_1}(S)$. This violates the lemma. The lemma says that we can get rid of such queries by normalizing, using a unary rewriting. That is, consider only databases where all tuples for S have the form $S(x, c)$, for a fixed constant c. The query rewrites to:

$$\begin{aligned} & S(x, c)S_2(x, c), S_1(x, y_2) \vee S_1(x, y)S_2(x, y)C(x, y) \vee C(x, y)S'(x, y) \vee S'(x, y)T(y) \\ \equiv \quad & R(x), R_2(x), S_1(x, y_2) \vee S_1(x, y)S_2(x, y)C(x, y) \vee C(x, y)S'(x, y) \vee S'(x, y)T(y) \end{aligned}$$

Where $R(x)$ and $R_2(x)$ are new relation symbols representing $S(x, c)$ and $S_2(x, c)$ respectively.

Proof: Suppose otherwise. Let's write q to expose its subcomponents:

$$q = [s_0(x),]s_1(x, y_1), s_2(x, y_2), \ldots, s_k(x, y_k)$$

q may, or may not have unary symbols, hence $s_0(x)$ is optional. By assumption there exists a component s_i in q that contains $S_1(x, y_i)$ but does not contain S. (It is possible that q does not contain S at all.) Also, since $c(S_1) \not\subseteq c(S)$, there exists a strict LR-path $P = (q_1, q_2, \ldots, q_n)$ s.t. q_1 contains S_1 but does not contain S. Perform a left unary rewriting with m constants, followed by the empty rewriting $S = 0$. This means that, for each binary symbol $S'(x, y)$ we create k fresh unary symbols $S'(x, c_1), \ldots, S'(x, c_k)$, and rewrite the query by replacing each subcomponent $s(x, y)$ with $s(x, y) \vee s(x, c_1) \vee \ldots \vee s(x, c_k)$: then we rewrite the query in DNF, minimize, then substitute $S(x, y)$ with empty, $S = 0$. Thus, $Q \Rightarrow Q'$. Let's examine the structure of the resulting query Q':

- Each component r in Q that does not contain S is also in Q'. This is because, when we expand in DNF, one of the components contains exactly all the original subcomponents $s(x, y)$ in r. (On the other hand if r contains S, then the subcomponents that contain S no longer exists after rewriting; only their unary rewritings $s(x, c_i)$ survive.) In particular, the strict LR-path P continues to exists in Q', since it doesn't contain S.

- From the component q, we obtain the following component q' (among others): $[s_0], s'_1, s'_2, \ldots, s'_k$, where s'_i is $s_i(x, y)$ if s_i does not contain S, and is $s_i(x, c_i)$ if it contains S. In other words, we provide each subcomponent containing S with a fresh constant, and leave other subcomponents untouched. Notice that at least one subcomponent containing $S_1(x, y)$ remains (since by assumption $s_q(S_1) \not\subseteq s_q(S)$), and therefore this is still a left component. It remains to check that it is not redundant. This follows from the fact that we have chosen enough constants $c_1, \ldots, c_k$: if there exists a homomorphism $h : r' \to q'$, then, by renaming the constants c_i to variables y_i we obtain a homomorphism $h : r \to q$, contradicting the fact that Q was minimal.

Thus, the new query Q' is still forbidden, because the same path P is still a strict LR-path, contradicting the fact that Q was normalized. $\Box$

Since this is the most difficult lemma involving rewritings, we illustrate with a few examples. By a *unary rewriting for S* we mean a unary rewriting, followed by $S = 0$.

Example J.23 Consider:

$$Q = S(x, y_1)S_1(x, y_2) \vee S(x, y_1)S_2(x, y_2) \vee S_1(x, y)S_2(x, y)C(x, y) \ldots$$

The left unary rewriting for S is:

$$\begin{aligned}
Q \quad \to \quad & (S(x, y_1) \vee S(x, c))(S_1(x, y_2) \vee S_1(x, c)) \\
& \vee (S(x, y_1) \vee S(x, c))(S_2(x, y_2) \vee S_2(x, c)) \\
& \vee S_1(x, y)S_2(x, y)C(x, y) \ldots \\
\to \quad & S(x, c), S_1(x, y_2) \vee S(x, c), S_1(x, c) \vee S(x, c), S_2(x, y_2) \vee S(x, c), S_2(x, c) \vee \ldots \\
= \quad & R(x), S_1(x, y) \vee R(x), R_1(x) \vee R(x), S_2(x, y) \vee R(x), R_2(x) \vee \ldots
\end{aligned}$$

In the first step we replaced every symbol $S_i(x, y)$ with $S_i(x, y) \vee S_i(x, c)$. Notice that there is no $C(x, c)$: by definition of the unary rewriting only the left symbols are rewritten. (Thus, the definition only makes sense if no symbol is both left and right.) In the second step we made $S(x, y) = 0$. The last line replaces every $S_i(x, c)$ with unary symbols $R_i(x)$.

The important thing to see here is that we have started with the left component $S(x, y_1)S_1(x, y_2)$ and rewritten it into a left component $R(x)S_1(x, y)$, and eliminated completely the binary symbol $S(x, y)$. The query above further reduces, by setting $R_1 = 0$.

Example J.24 This shows why we need more constants. Consider a query with two left components:

$$\begin{aligned} Q \quad = \quad & S(x,y_1)S_1(x,y_1), S(x,y_2)S_2(x,y_2)S_3(x,y_2), S_4(x,y_3) \vee \quad /{*}q{*}/ \\ & S(x,y_1)S_1(x,y_1)S_2(x,y_1), S(x,y_2), S_3(x,y_2) \quad /{*}q'{*}/ \\ & \ldots \end{aligned}$$

Suppose we do a unary rewriting for S. If we use a single constant c, then the first component q rewrites to:

$$\begin{aligned} Q' \quad = \quad & S(x,c)S_1(x,c), S_2(x,c), S_3(x,c), S_4(x,y_3) \vee \quad /{*}q{*}/ \\ & S(x,c)S_1(x,c)S_2(x,c), S(x,c), S_3(x,c) \quad /{*}q'{*}/ \end{aligned}$$

The problem is that now q is redudant: there exists a homomorphism from the second component. However, if we use two constants c_1, c_2 (since there are two occurrences of S in q) then one of the rewritings of q is:

$$\begin{aligned} Q' \quad = \quad & S(x,c_1)S_1(x,c_1), S(x,c_2), S_2(x,c_2), S_3(x,c_2), S_4(x,y_3) \vee \\ & S(x,c_1)S_1(x,c_1)S_2(x,c_1), S(x,c_2), S_3(x,c_2) \vee \ldots \end{aligned}$$

(each component must contain all four combinations $c_1c_1, c_1c_2, c_2c_1, c_2c_2$; only c_1c_2 is shown). Each of the two components shown above is isomorphic to their image in Q, up to remaining of c_1, c_2 as y_1, y_2. Therefore no new homomorphisms are introduced, and both are non-redudnat.

Example J.25 This shows that the unary rewriting for q may wipe out some other symbol S_1:

$$q \quad = \quad S(x,y_1)S_1(x,y_1), S_2(x,y_2)$$

After the unary rewriting for S, the component becomes:

$$q' \quad = \quad S(x,c)S_1(x,c), S_2(x,y_2)$$

That is, we lost $S_1(x, y)$. This may be important, if S_1 is the only link to the right side. This is why in the lemma we only do a unary rewriting for S when we know that there exists a useful symbol S_1 s.t. $s_q(S_1) \not\subseteq s_q(S)$. In this example $s_q(S_1) = s_q(S)$, and we cannot apply the rewriting.

Intuitively, Lemma J.22 implies that, if S is supefluous (i.e. $c(S) = \emptyset$), then $s_q(S)$ must be very big. We make this precise.

Definition J.26 *Call a binary left S symbol* ubiquitous *if, for every $q \in UL$ and every binary subcomponent s of q, s contains S. That is, $s_q(S)$ is the set of all binary subcomponents of q. Similarly for right ubiquitous symbols.*

For an illustration, consider the query K in Example J.17: the symbol S is ubiquitous because it occurs in both subcomponents of $S(x, y_1)S_1(x, y_1)S(x, y_2)S_2(x, y_2)$.

Corollary J.27 *If Q is in normal form, then all superfluous symbols are ubiquitous.*

Proof: Let S be a superfluous symbol. For any useful symbol S_1 we have $c(S) = \emptyset$ and $c(S_1) \neq \emptyset$, which implies $c(S_1) \not\subseteq c(S)$. Thus, by Lemma J.22, for any left component $q \in UL$ we have $s_q(S_1) \subseteq s_q(S)$. Let:

$$s_q(*) \quad = \quad \bigcup \{s_q(S_1) \mid S_1 \text{ is useful }\}$$

Thus, $s_q(*)$ is the set of all subcomponents in q that contain some useful symbol. For every superfluous symbol S, $s_q(*) \subseteq s_q(S)$ because $s_q(S_1) \subseteq s_q(S)$ for any useful symbol S_1. Supose S is a superfluous symbol that is not ubiquituous, i.e. there exists a left component $q \in UL$ s.t. $s_q(S)$ is not the set of all binary subcomponents; let s_i be a binary subcomponent s.t. $s_i \notin s_q(S)$. Then $s_i \notin s_q(*)$, hence s_i consists only of binary, superfluous symbols S'. Since $q \in UL$ it contains some useful symbol S_1; let s_j be a subcomponent that contains S_1. In addition, s_j contains all superfluous symbols S', because $s_q(S_1) \subseteq s_q(S')$. This implies that there exists a homomorphisms $s_i \to s_j$, contradicting the fact that Q is minimized. $\Box$

Corollary J.28 *If Q is in normal form then every left component contains a useful symbol. In other words, every left component is in UL.*

Proof: Otherwise some left component q contains only supefluous components. Let q_1 be any component in UL. We prove that there exists a homomorphism $q \to q_1$, contradicting the fact that Q is minimal. Consider first any unary symbol R in q. Since R is superfluous (all unary symbols are superfluous), it follows that R occurs in all component in UL: thus, R is in q_1. Consider a binary symbol S in q. Since S is superfluous (by assumption), it is ubiquitous, hence it occurs in all binary components of q_1. Therefore there exists a homomorphism $q \to q_1$. $\Box$

The converse to Corollary J.27 fails in general. For example in:

$$H_3 \quad = \quad R(x), S_1(x, y) \vee S_1(x, y), S_2(x, y) \vee S_2(x, y), S_3(x, y) \vee S_3(x, y), T(y)$$

S_1 is both ubiquitous and useful. However, the converse does hold under a restriction:

Corollary J.29 *If Q is in normal form and has at least one non-ubiquitous binary symbol, then all left ubiquitous symbols are superfluous.*

Proof: Let S be a left ubiquitous symbol, and consider the rewriting $S = 1$. We prove that the resulting query $Q[S = 1]$ is also forbidden. First, we show that every left component q remains non-redundant after the rewriting: this is because every homomorphism $h : r[S = 1] \rightarrow q[S = 1]$ extends to a homomorphism $h : r \rightarrow q$, because S appears in all binary subcomponents of q. Also, $q[S = 1]$ continues to be a left component (i.e. doesn't become a central component), because it has exactly the same number of subcomponents as q. Next, assuming S is useful, consider any strict LR path $P = (q_1, q_2, \ldots)$ where q_1 contains S. Let S_1 be any non-ubiquitous binary symbol. (There exists at least, by the assumption of the corollary.) Since $s_q(S) \not\subseteq s_q(S_1)$ for any left query q, we have $c(S) \subseteq c(S_1)$, hence S_1 occurs in q_1. Thus, the path P is also a path from S_1 to the right, so if we show that all components in P remain in $Q[S = 1]$ then we have proven that $Q[S = 1]$ is a forbidden query. Suppose there exists a homomorphism $h : r[S = 1] \rightarrow q_i[S = 1]$. Clearly r must contain S. There are two cases. Case 1: $i > 1$ then q_i contains only central or right symbols: therefore all symbols in $r[S = 1]$ must be center symbols (since S doesn't co-occur with a right symbol). r cannot be a left component, because then $r[S = 1]$ contains at least one left symbol, so r must be a central component. r is also a useful component, because the path $r, q_i, q_{i+1}, \ldots$ is a strict, LR path connecting the left symbol S to the right. Thus, $r \in c(S)$, hence it also contains S_1, which is a contradiction because q_i does not contain S_1. Consider case 2, $i = 1$. Then q_1 contains S, and we can extend the homomorphism $r[S = 1] \rightarrow q_1[S = 1]$ to a homomorphism $r \rightarrow q_1$. Thus, we have proven that $Q[S = 1]$ is also a forbidden query, contradicting the fact that it was normalized. □

J.3 Classification of Forbidden Queries

In addition to the properties of normal queries described so far, we give now a classification of their endpoings. We show that their left end is of one of two types, and similarly their right end; thus, there are four types of forbidden queries.

Definition J.30 (Type 1) *If Q has a left unary symbol R, then we say that its left end is of Type 1. Similarly for the right end.*

Theorem J.31 *Suppose Q is in normal form and has the left-end of type 1. Then the following hold:*

- *Q has exactly one left unary symbol.*
- *Every left binary symbol is useful.*
- *Every left component has exactly two variables x, y.*

Similarly for a right-end of Type 1.

The second item is stronger than Corollary J.28.

Definition J.32 (Type 2) *If Q has no left unary symbols, then we say that its left end is of Type 2. Similarly for the right end.*

Theorem J.33 *Suppose Q is in normal form and has the left-end of type 2. Then the following hold:*

- *There exists q such that $q \in c(S_1)$ for all left useful symbols S_1.*
- *There exists at least one superfluous symbol S.*

Similarly for the right.

Thus, there are for types of normalized forbidden queries: 1-1, 1-2, 2-1, or 2-2.
In the rest of the section we prove the two theorems, through a sequence of lemmas.

Lemma J.34 *If Q is in normal form and the left end is of Type 1, then it has exactly one unary symbol R.*

Proof: Let $R_1, R_2, \ldots, R_m$ be all left unary symbols, $m \geq 1$ (because the query is of Type 1). Since all area superfluous, every component $q \in UL$ contains all symbols R_i (Prop. J.19). Suppose $m > 1$. Then choose any symbol R_j and rewrite Q by making R_j deterministic: $R_j = 1$. No component becomes redundant, hence the new query is still forbidden, contradicting the assumption that Q was normalized. $\Box$

Next, we show that Q has no superfluous left binary symbols. For example:

$$R(x), S(x,y), S_1(x,y) \vee S_1(x,y), C(x,y) \vee \ldots$$

contains the superfluous left symbol S: simply set $S = 1$ and we remove it. On the other hand, consider:

$$R(x), S(x,y), S_1(x,y) \vee S(x,y), S_1(x,y), D(x,y) \vee S_1(x,y), D(x,y), C(x,y) \\ \vee C(x,y), S_1'(x,y) \vee S_1'(x,y), T(y)$$

Here S is still superfluous, but we cannot set $S = 1$ because we make the S_1DC component redundant, disconnecting the left from the right. Instead, we set $S_1 = 1$: now S becomes a useful symbol.

Lemma J.35 *If Q is in normal form and the left end is of Type 1, then every left symbol S is useful.*

Proof: Let S_1 be any useful left symbol, and consider a strict LR-path P from S_1 to a right symbol, $P = (q_1, q_2, \ldots)$. Let S be a binary, left superfluous symbol; by Corollary J.27 it, is ubiquitous, i.e. occurs in all binary subcomponents of all components in UL.

Define R the set of all components r with the following properties: r contains S and there exists a homomorphism $h : r[S = 1] \to q_1$ (the first component in P). We consider two cases.

First, $R = \emptyset$. Then rewrite Q by setting $S = 1$. None of the components on the path P becomes redundant in $Q[S = 1]$: indeed if there exists a homomorphism $r[S = 1] \rightarrow q_i$ then we must have $i \geq 2$. Then r cannot be a left component, because it would contain the symbol $R(x)$. Hence r is a central component. Moreover, $r[S = 1]$ consists only of central symbols, because it maps to q_i and $i \geq 2$: but in that case $(r, q_i, q_{i+1}, \ldots)$ is an LR path from S to a right symbol contradicting the assumption that S is superfluous. Thus, the path P continues to exists in $Q[S = 1]$. Let $q \in UL$ be a left component containing S_1. Since S is ubiquitous, $q[S = 1]$ is not redundant because any homomorphism $q'[S = 1] \rightarrow q[S = 1]$ extends to $q' \rightarrow q$, and is still a left component (contains the left unary symbol $R(x)$, and also $S_1(x, y)$). Thus, $Q[S = 1]$ still has the strict LR path P.

Second, suppose $R \neq \emptyset$. Rewrite Q by setting $S_1 = 1$: we prove that in $Q[S_1 = 1]$ there exists a LR path from the left symbol S to a right symbol, contradicting the fact that Q is normalized. At least one component $r[S_1 = 1]$ for $r \in R$ is not redundant. This is because any homomorphism $r'[S_1 = 1] \rightarrow r[S_1 = 1]$ implies a homomorphism $r'[S_1 = 1, S = 1] \rightarrow r[S_1 = 1, S = 1] \rightarrow q_1[S_1 = 1]$ (since, by definition, there exists a homomorphism $r[S = 1] \rightarrow q_1$), which can be extended to a homomorphism $r'[S = 1] \rightarrow q_1$, meaning $r' \in R$. Thus, any component in R that becomes redundant does so because of some other component also in R, and at least one $r[S_1 = 1]$ will remain non-redundant in $Q[S_1 = 1]$, for some $r \in R$. Obviously $q_1[S_1 = 1]$ also exists in $Q[S_1 = 1]$, because q_1 already contains S_1 and cannot become redundant by setting it to 1. Thus, in $Q[S_1 = 1]$ we have a path from S to a right symbol.

It remains to prove S is a left symbol in $Q[S_1 = 1]$, and for that we will show that at least one component $q \in UL$ remains in $Q[S_1 = 1]$, i.e. it does not become redundant. Then, $q[S_1 = 1]$ still contains the symbol S (since it is ubiquitous), and also contains the unary symbol R, and hence it is a left component. To prove the claim, denote $UL(S_1)$ the set of components in UL that contain the symbol S_1. We will show that, for every $q \in UL(S_1)$ if there exists a homomorphism $r[S_1 = 1] \rightarrow q[S_1 = 1]$, then $r \in UL(S_1)$: this immediately implies that at least one $q[S_1 = 1]$ remains nonredundant. Indeed, suppose $r \notin UL(S_1)$. Clearly r contains S_1 (otherwise we get a homomorphism $r \rightarrow q$), hence the only possibility is for r to be a central component. It cannot have any central symbols (since q doesn't have central symbols), and it is a single subcomponent: $r = S_1(x, y), S_2(x, y), S_3(x, y) \ldots$ The homomorphism must map $r[S_1 = 1]$ to a single subcomponent of q, that contains $S_2, S_3, \ldots$ but does not contain S_1 (otherwise we have a homomorphism $r \rightarrow q$). Fix $i \geq 2$. We have $s_q(S_1) \not\subseteq s_q(S_i)$ and $s_q(S_i) \not\subseteq s_q(S_1)$. By Lemma J.22 $c(S_1) = c(S_i)$; furthermore, this holds for any i. In this case, however, every starting point q_1 of a LR path from S_1 to a right symbol is redundant: indeed, $q_1 \in c(S_1) = c(S_2) = \ldots$, hence q_1 contains all symbols $S_2, S_3, \ldots$, hence there exists a homomrphism $r \rightarrow q_1$, contradicting that Q is minimal. $\Box$

Lemma J.36 *If Q is in normal form and the left end is of Type 1, then all left components have exactly two variables x, y.*

Proof: Recall that all left components are in UL (Corollary J.28). Let $q_0 \in UL$ violate our condition. Thus $q_0 = R(x), s_1(x, y_1), \ldots, s_k(x, y_k)$, with $k \geq 2$. Consider any two subcomponents s_1, s_2. Let S_1 be a symbol contained in the first but not the second, and S_2 be a symbol in the second but not the first. Thus, $s_{q_0}(S_1) \not\subseteq s_{q_0}(S_2)$ and $s_{q_0}(S_2) \not\subseteq s_{q_0}(S_1)$; Lemma J.22 implies $c(S_1) = c(S_2)$. Both S_1, S_2 are useful symbols, because if S_1 were superfluous then it would be ubiquitous, which is a contradiction.

Let $P = (q_1, q_2, \ldots)$ be a LR-strict path from S_1 to a right symbol; q_1 also contains S_2 (since $c(S_1) = c(S_2)$). Consider the deterministic rewriting, $S_1 = 1$: clearly $q_0[S_1 = 1]$ is still a left component, since it still has $R(x)$ and at least one binary symbol, namely $S_2(x, y)$. We will prove that in the new query $Q[S_1 = 1]$ the path P still exists, i.e. none of the components $q_i[S_1 = 1]$ is redundant: this shows that $Q[S_1 = 1]$ is still a forbidden query, contradicting the fact that Q was normalized. Suppose otherwise: there exists a homomorphism $h : r[S_1 = 1] \to q_i[S_1 = 1]$. We must have $i > 1$, because q_1 contains S_1, hence any homomorphism h extends to $r \to q_1$, contradiction. If $i > 1$, then q_i contains only center and right symbols, $q_i[S_1 = 1] = q_i$. There are two cases. First, if r is a center component. Then r is also useful, because the path $r, q_i, q_{i+1}, \ldots$ is a strict LR path connecting the left symbol S_1 to something on the right. Thus, $r \in c(S_1)$, and also $r \in c(S_2)$, in other words $r[S_1 = 1]$ contains S_2: this makes a homomorphism $r[S_1 = 1] \to q_i$ impossible. Second, if r is a left component, then it must have contain the unary symbol $R(x)$, and the homomorphism is also not possible. $\Box$

This completes the proof of Theorem J.31. We turn now to the proof of Theorem J.33, which follows from the following two lemmas.

Lemma J.37 *If Q is in normal form and the left end is of Type 2, then there exists a useful central component q that contains every useful symbol S. In other words, $q \in c(S)$ forall left useful symbols S.*

Proof: Let $s_*(S)$ denote the set of all subcomponents of all left components that contain the useful symbol S. In other words, $s_*(S)$ is the union of $s_q(S)$ for all left components q. Since a useful symbol S cannot be ubiquitous, $s_*(S)$ does not contain all subcomponents. On the other hand, every subcomponent s (of every left component q) must contain at least one useful symbol: otherwise, if it contains only superfluous symbols then it can be mapped to any other subcomponent, since all superfluous symbols are ubiquitous, contradicting the fact that q is minimized. Order the sets $s_*(S)$ by set inclusion. There are at least two maximal sets. Indeed, if $s_*(S_1)$ is any maximal set, then let s be any subcomponent that is not included in $s_*(S_1)$, and let S be any useful symbol in s. Let $s_*(S_2)$ be any maximal set that contains $s_*(S)$: it is incomparable with $s_*(S_1)$. Since $s_*(S_1) \not\subseteq s_*(S_2)$ and $s_*(S_2) \not\subseteq s_*(S_1)$ it follows that $c(S_1) = c(S_2)$. Let $q \in c(S_1)$. We will show that q contains all useful left symbols. Let S be a useful left symbol, and let $s_*(S')$ be any maximal set containing $s_*(S)$: if $s_*(S)$ is already maximal, then we take $S' = S$, otherwise we have $s_*(S) \subset s_*(S')$. Consider the set $s_*(S')$: it may either be equal to $s_*(S_1)$, or to $s_*(S_2)$, or may be different from both. In all cases $s_*(S')$ is different from either $s_*(S_1)$ or $s_*(S_2)$: suppose it is different from $s_*(S_1)$. Then $s_*(S') \not\subseteq s_*(S_1)$ and $s_*(S_1) \not\subseteq s_*(S')$, implying $c(S_1) = c(S')$. Returning to S, we either have $S = S'$, implying that $c(S_1) = c(S)$

hence q contains S, or $s_*(S) \subset s_*(S_1)$, implying $c(S_1) \subseteq c(S)$, and we also conclude that q contains S. $\Box$

Lemma J.38 *If Q is in normal form and the left end is of Type 2, then there exists at least one superfluous symbol S.*

Proof: Indeed, let q be the component given by the previous lemma, i.e. $q \in c(S_1)$ forall left useful symbols S_1. In other words, q contains all left useful symbols S_1. Let q_0 be any left component with a useful symbol. If q_0 contains no superfluous symbols, then there exists a homomorphism $q_0 \to q$, contradicting the fact that Q is minimal. This proves the third item. $\Box$

K Part 3 of the Proof: Algorithm

Let Q be a forbidden query. We assume throughout this section that Q is minimized, long, and normalized. We describe in this section PTIME algorithm for the m_1, m_2-Signature-Counting problem with an oracle for computing $P(Q)$ over tuple-independent databases. Here $m_1, m_2 \geq 2$ and depende on the query Q.

K.1 The Labels

We define two sets of labels LL and RR for the left end, and for the right end of Q.

Definition K.1 *If the left end of Q is of Type 1, then $LL = \{\tau_1, \tau_2\}$, where τ_1, τ_2 are the following queries:*

$$\begin{aligned} \tau_1 &= \neg R(x) \\ \tau_2 &= R(x) \end{aligned}$$

Here R is the unique left unary symbol in Q.

Similarly, if the right end of Q is of Type 1, then $RR = \{\tau_1, \tau_2\}$ where:

$$\begin{aligned} \tau_1 &= \neg T(x) \\ \tau_2 &= T(x) \end{aligned}$$

Here T is the unique right unary symbol in Q.

Assume that the left end of Q is of type 2. Call a *pre-label* a disjunction of conjunctive queries of the form $S_1(x, y), S_2(x, y), \ldots, S_k(x, y)$, for $k \geq 1$.

Definition K.2 *Let Q have a left end of Type 2, and let $UL = \{q_1, \ldots, q_k\}$. Let a be a constant. The* base labels *are the prelabels $\tau_1, \ldots, \tau_{m_1}$ such that:*

$$\tau_1[a/x] \wedge \ldots \wedge \tau_{m_1}[a/x] \quad \equiv \quad q_1[a/x] \vee \ldots \vee q_k[a/x]$$

and the CNF expression above is not redundant. The set LL is defined as the closure under $\vee$ of all base labels.

In particular, it follows that for any left component $q \in UL$ and every left label $\tau \in LL$, $q \Rightarrow \tau$.

Example K.3 We illustrate here by showing only the left components of Q. Consider first a query with one left component:

$$Q = S(x,y_1)S_1(x,y_1), S(x,y_2)S_2(x,y_2), S(x,y_3)S_3(x,y_3) \vee \ldots$$

The labels are:

$$\begin{aligned}
\tau_1 &= S(x,y), S_1(x,y) \\
\tau_2 &= S(x,y), S_2(x,y) \\
\tau_3 &= S(x,y), S_3(x,y)
\end{aligned}$$

and their closure under disjunctions. Thus, LL has 7 elements.

Consider a query with two left components:

$$\begin{aligned}
Q = {} & S(x,y_1), S_1(x,y_1), S(x,y_2), S_2(x,y_2) \vee \\
& S(x,y_1), S_1(x,y_1), S(x,y_3), S_3(x,y_3) \vee \ldots
\end{aligned}$$

The labels are:

$$\begin{aligned}
\tau_1 &= S(x,y), S_1(x,y) \\
\tau_2 &= S(x,y), S_2(x,y) \vee S(x,y), S_3(x,y)
\end{aligned}$$

and $\tau_1 \vee \tau_2$. Thus, LL has 3 elements.

Thus, for every query Q we have two sets of labels LL and RR. Denote $m_1 = |LL|$, $m_2 = |RR|$. It should be clear that $m_1 \geq 2$ and $m_2 \geq 2$.

K.2 The Expansion Formula

For a left label $\tau_1 \in LL$ and constant $a \in X$, denote $\tau[a/x]$ the query obtained by substituting x with the constant a. Its negation, $\neg\tau[a/x]$, is a formula that is universally quantified in the variable y. Similarly we denote $\tau[b/y]$ when $\tau \in RR$ and $b \in Y$.

Lemma K.4 *Let W be a world (i.e. a deterministic database) s.t. Q is false on W. Then, for every constant in its active domain, $a \in Adom(W)$ there exists a label $\tau \in LL$ s.t. $\tau[a/x]$ is false on W. In notation:*

$$W \models \neg Q \quad \Rightarrow \quad \forall a \in Adom(W) \exists \tau \in LL \; : \; W \models \neg\tau[a/x]$$

Proof: If Q's left end is of type 1, then the lemma is trivial: for every $a \in W$ either $R(a)$ is false or $\neg R(a)$ is false. Suppose Q's left end is of type 2. We proof by contradiction. If the statement is false, then for each $i = 1, \ldots, m_1$ there exists a constant b_i s.t. $W \models \tau_i[a/x, b_i/y]$. In other words:

$$W \models \bigwedge_{i=1,m_1} \tau_i[a/x]$$

hence $W \models \bigvee_{q \in UL} q[a/x]$. But this implies $W \models Q$, contradiction. $\square$

We write $W(a, b)$ to denote a deterministic structure with two distinguished constants, s.t. $W(a, b)$ does not share any constants other than a, b with other structures. When we don't care naming b, then we write $W(a, *)$; similarly $W(*, b)$.

If the left end of Q is of Type 1, then a label $\tau \in LL$ is either $\neg R(x)$ or $R(x)$. Define $\varepsilon^\tau \in \{0, 1\}$ as follows:

$$\begin{aligned} \varepsilon^{\neg R(x)} &= 0 \\ \varepsilon^{R(x)} &= 1 \end{aligned}$$

Recall that $Q[R = 0]$ means the query Q obtained by setting $R(x) \equiv$ `false`; and $Q[R = 1]$ means the query Q obtained by setting $R(x) \equiv$ `true`. Thus, $(\tau[a/x] \vee Q)[R = \neg\varepsilon^\tau]$ is equivalent to $Q[R = \neg\varepsilon^\tau]$, because $\tau[a/b]$ is `false` after the substitution $R(a) = \neg\varepsilon^\tau$. On the other hand if the left end of Q is of Type 2, then $(\tau[a/x] \vee Q)[R = \neg\varepsilon^\tau]$ simply denotes $\tau[a/x] \vee Q$, since in this case there is no symbol R. Similarly for the right end.

For fixed $a \in X, b \in Y$, define the following queries:

$$\begin{aligned} Q^{a,b}_{\tau_1,\tau_2} &= (\tau_1[a/x] \vee Q \vee \tau_2[b/y])[R = \neg\varepsilon^{\tau_1}, T = \neg\varepsilon^{\tau_2}] \\ Q^{a}_{\tau_1,*} &= (\tau_1[a/x] \vee Q)[R = \neg\varepsilon^{\tau_1}] \\ Q^{b}_{*,\tau_2} &= (Q \vee \tau_2[b/y])[T = \neg\varepsilon^{\tau_2}] \end{aligned}$$

Fix an instance of the m_1, m_2-SC problem: $\Phi = (X, Y, E)$, where $X = \{a_i \mid i = 1, n_1\}$, $Y = \{b_j \mid j = 1, n_2\}$, $E \subseteq X \times Y$, $|E| = n$. The type of the signature to be the same as the type of the left and right endings of the query (i.e. type 1,1 or 1,2 or 2,1 or 2,2). (See Sec. G for the definitions.)

We associate to Φ a probabilistic database D that is the union of three types of blocks:

- One block $D(a, b)$ for each edge $(a, b) \in E$.
- If Q's left end is of type 2, then one block $D(a, *)$ for each node $a \in X$.
- If Q's right end is of type 2, then one block $D(*, b)$ for each node $b \in Y$.
- If the left end is of type 1, then we set $p(R(a)) = 1/2$ forall $a \in X \cup \{*\}$
- If the right end is of type 1, then we set $p(T(b)) = 1/2$ forall $b \in Y \cup \{*\}$.

Blocks do not share any constants, except the constants in X and Y. All blocks $D(a, b)$ isomorphic; all blocks $D(a, *)$ are isomorphic; all blocks $D(*, b)$ are isomorphic. All probabilities other than those of the unary tuples are left unspecified: we will set them later. However, we require the tuple probabilities to be preserved by the isomorphisms: that is, we only need to specify the probabilities for one block $D(a, b)$, one block $D(a, *)$, and one block $D(*, b)$; the probabilities in all other blocks will be equal.

Definition K.5 *Given a world W and labeling $l = (l_1, l_2)$, $l_1 \in LL^X$, $l_2 \in RR^Y$, we say that W satisfies the labeling, $W \models \neg l$, if:*

$$\begin{array}{ll} \forall a \in X: & W \models \neg l_1(a)[a/x] \\ \forall b \in Y: & W \models \neg l_2(b)[b/y] \end{array}$$

Lemma K.4 implies that, whenever $W \models \neg Q$ then there exists a labeling l such that $W \models l$.

We start with a key technical lemma:

Lemma K.6 *Let $W = \bigcup_{(a,b)\in E} W(a,b) \cup \bigcup_{a\in X} W(a,*) \cup \bigcup_{b\in Y} W(*,b)$ be a world (deterministic database) that is a union of blocks. The blocks do not share any constants other than those in X and Y.*

$$W \models \neg Q \quad \equiv \quad \bigvee_{\substack{l_1 \in LL^X \\ l_2 \in RR^Y}} \left(\begin{array}{l} \bigwedge_{(a,b)\in E} W(a,b) \models Q^{a,b}_{l_1(a),l_2(b)} \wedge (R(a) = \varepsilon^{l_1(a)}) \wedge (T(b) = \varepsilon^{l_2(b)}) \\ Q^{a}_{l_1(a),*} \wedge (R(a) = \varepsilon^{l_1(a)}) \\ \\ Q^{b}_{*,l_2(b)} \wedge (T(b) = \varepsilon^{l_2(b)}) \end{array} \right) \tag{16}$$

Proof: The "only if" implication follows from the previous lemma. We prove the "if" implication. Suppose the contrary, that $W \models Q$, and let q be a component in Q that is true in W. If q is a center component, then it has only two variables x, y and they must both be mapped to the same block $W(a,b)$, or $W(a,*)$ or $W(b,*)$: assuming the former, we have $W(a,b) \models Q$. If q is a left component with variables $x, y_1, \ldots, y_k$ then we consider two cases. If x is mapped to a constant $a \in X$, then we use the fact that $q \Rightarrow l_1(a)$, thus $q[a/x] \Rightarrow l_1(a)[a/x]$, and therefore $W \models q[a/x]$ implies $W \models l_1(a)[a/x]$. If x mapped to a constant that is not in X, then all y_i's must be mapped to a common block, say $W(a,b)$, and it follows $W(a,b) \models q$, hence $W(a,b) \models Q$, contradiction. $\Box$

Denote $p_{D(a,b)}(-)$ the probability space defined by the probabilistic database $D(a,b)$; and similarly for $D(a,*)$ and $D(*,b)$. We define:

$$\begin{array}{rcl} F_{\tau_1\tau_2} & = & p_{D(a,b)}(\neg Q^{a,b}_{\tau_1,\tau_2}) \\ F_{\tau_1,*} & = & p_{D(a,*)}(\neg Q^{a}_{\tau_1,*}) \\ F_{*,\tau_2} & = & p_{D(*,b)}(\neg Q^{b}_{*,\tau_2}) \end{array}$$

These three functions do not depend on a and b, because the blocks are isomorphic.

The key result connecting the probability $p(\neg Q)$ to the SC-problem is given in the next three theorems, one for each type of query 1-1, 1-2, or 2-2.

Theorem K.7 *Suppose both left and right ends of Q are of type 1. Let $D = \bigcup_{(a,b)\in E} D(a,b)$. Then:*

$$p(\neg Q) \quad = \quad \frac{1}{2^{n_1+n_2}} \sum_{\sigma\in\Sigma_{2,2}(\Phi)} \#\sigma \cdot \prod_{\tau_1\in LL,\tau_2\in RR} F^{\sigma(\tau_1,\tau_2)}_{\tau_1,\tau_2}$$

Proof: Each possible world $W \subseteq D$ fixes the truth values of $R(a), a \in X$ and $T(b), b \in Y$. We associate to W the following labeling $l^W = (l_1^W, l_2^W)$:

$$\begin{aligned}
\forall a \in X: \ l_1^W(a) &= \tau, \text{ s.t. } \tau \in LL \text{ and } W \models \neg\tau[a/x] \\
\forall b \in Y: \ l_2^W(b) &= \tau, \text{ s.t. } \tau \in RR \text{ and } W \models \neg\tau[b/y]
\end{aligned}$$

Fix a labeling $l = (l_1, l_2)$ and denote $l^W = l$ the event that a random world W defines the label l. We will compute the conditional probability $p(\neg Q | l^W = l)$, using Eq. (16): in our case we only have the conjunct $\bigwedge_{(a,b)\in E}$, because there are no blocks of the form $D(a,*)$ or $D(*,b)$. Consider two constants a, b and consider the event that, for a random world W, Q is false on $W(a,b) \subseteq D(a,b)$. Recall that we assumed that W satisfies l, and therefore:

$$W(a,b) \models \neg Q \quad \equiv \quad W(a,b) \models \neg Q^{a,b}_{l_1(a),l_2(b)}$$

Let a, b vary and consider the family of events above: this set of events is independent, because each event depends only on the probabilistic tuples in the block $D(a,b)$, which are not shared with any other block. Moreover, the probability of this event is:

$$p_{D(a,b)}(Q^{a,b}_{l_1(a),l_2(b)}) \quad = \quad F_{l_1(a),l_2(b)}$$

Thus, by Eq. (16), $P(\neg Q | l^W = l)$ is their product:

$$\begin{aligned}
p(\neg Q | l^W = L) &= \prod_{(a,b)\in E} F_{l_1(a),l_2(a)} \\
&= \prod_{\tau_1 \in LL, \tau_2 \in RR} F^{\sigma_l(\tau_1,\tau_2)}_{\tau_1,\tau_2}
\end{aligned}$$

We use the fact that $p(l^W = l) = 1/2^{n_1+n_2}$ and derive:

$$\begin{aligned}
p(\neg Q) &= \frac{1}{2^{n_1+n_2}} \sum_l \prod_{\tau_1 \in LL, \tau_2 \in RR} F^{\sigma_l(\tau_1,\tau_2)}_{\tau_1,\tau_2} \\
&= \frac{1}{2^{n_1+n_2}} \sum_\sigma \#\sigma \prod_{\tau_1 \in LL, \tau_2 \in RR} F^{\sigma(\tau_1,\tau_2)}_{\tau_1,\tau_2}
\end{aligned}$$

$\square$

Next we turn to queries of type 2-1. Here we order the set LL by: $\tau_1 \leq \tau_2$ if $\tau_2 \Rightarrow \tau_1$. LL becomes a meet semi-lattice, where the meet operation is query disjunction $\tau_1 \vee \tau_2$; we complete it to the lattice $\overline{LL} = LL \cup \{\hat{1}\}$. Recall that a signature of type 2-1 is $\sigma = (\sigma^X, \sigma^E)$.

Theorem K.8 *Suppose that the left of Q is of type 2, and the right end of type 1. Let $D = \bigcup_{(a,b)\in E} D(a,b) \cup \bigcup_{a\in X} D(a,*)$. Then:*

$$\begin{aligned}
p(\neg Q) \quad &= \quad \frac{(-1)^{n_1+1}}{2^{n_2}} \sum_{\sigma\in\Sigma_{m_1,2}(\Phi)} \#\sigma \cdot \prod_{\tau_1\in LL,\tau_2\in RR} F_{\tau_1,\tau_2}^{\sigma^E(\tau_1,\tau_2)} \\
&\qquad \prod_{\tau_1\in LL} (\mu_{LL}(\tau_1,\hat{1})\cdot F_{\tau_1,*})^{\sigma^X(\tau_1)}
\end{aligned}$$

Proof: Each possible world $W \subseteq D$ defines a Y-labeling l_2^W as follows:

$$\forall b \in Y. l_2^W(b) \quad = \quad \tau, \text{ s.t. } \tau \in RR \text{ and } W \models \neg\tau[b/y]$$

Conversely, for each Y-labeling l_2, we consider the event that the labeling defined by a random world W agrees with l, i.e. $l_2^W = l_2$. For a fixed l_2, compute the conditional probability $p(\neg Q|l_2^W = l_2)$, by using Eq. 16: in our case we only have the conjuncts $\bigwedge_{(a,b)\in E}$ and $\bigwedge_{a\in X}$, because there are no blocks of the form or $D(*,b)$. We claim that, conditioned on $l_2^W = l_2$, the following is an independent set of events:

$$\begin{aligned}
&\{W(a,b) \not\models l_1(a)[a/x] \vee Q \vee l_2(b)[b/y] \mid (a,b)\in E\} \cup \\
&\{W(a,*) \not\models l_1(a)[a/x] \vee Q \mid a \in X\}
\end{aligned}$$

Indeed, the truth values of all $T(b)$'s are fixed, and there is no unary predicate on a, so all events above depend only on the probabilistic tuples inside their specific block, $D(a,b)$ or $D(a,*)$, which are not shared with any other block. Moreover, the probability of each such event is:

$$\begin{aligned}
p_{D(a,b)}(\neg l_1(a)[a/x] \wedge \neg Q \wedge \neg l_2(b)[b/y]|\neg l_2(b)[b/y]) \quad &= \quad p_{D(a,b)}(\neg Q^{a,b}_{\tau_1,\tau_2}) = F_{l_1(a),l_2(b)} \\
p_{D(a,*)}(\neg l_1(a)[a/x] \wedge \neg Q \wedge \neg l_2(b)[b/y]|\neg l_2(b)[b/y]) \quad &= \quad p_{D(a,*)}(\neg Q^{a}_{\tau_1,*}) = F_{l_1(a),*}
\end{aligned}$$

Therefore, we compute $p(\neg Q|l_2^W = l_2)$ by staring from Eq. 16, then using Mobius' inversion formula. We abbreviate $p(-|l_2^W = l_2)$ with $p^{l_2}(-)$:

$$\begin{aligned}
&p^{l_2}(\neg Q) = \\
&\qquad p^{l_2}(\bigvee_{l_1\in LL^X} \bigwedge_{(a,b)\in E} W(a,b) \not\models l_1(a)[a/x] \vee Q \vee l_2(b)[b/y] \wedge \bigwedge_{a\in X} W(a,*) \not\models l_1(a)[a/x] \vee Q) \\
&\quad = \quad \sum_{l_1\in LL^X} \mu_{\overline{LL^X}}(l_1,\hat{1}) \prod_{(a,b)\in E} F_{l_1(a),l_2(b)} \times \prod_{a\in X} F_{l_1(a),*} \\
&\quad = \quad (-1)^{n_1+1} \sum_{l_1\in LL^X} \prod_{(a,b)\in E} F_{l_1(a),l_2(b)} \times \prod_{a\in X} (\mu_{LL}(l_1(a),\hat{1}) F_{l_1(a),*})
\end{aligned}$$

In the last line above we used Lemma A.3, which states that the Mobius function in the lattice LL^X is the product of Mobius functions in the lattice LL. The claim of the theorem follows from the fact that $p(l_2^W = l_2) = 1/2^{n_1}$. $\Box$

Finally, when both left and right ends are of type 2 then we have:

Theorem K.9 *Suppose both that both left and right ends of Q are of type 2. Let $D = \bigcup_{(a,b)\in E} D(a,b) \cup \bigcup_{a\in X} D(a,*) \cup \bigcup_{b\in Y} D(*,b)$. Then:*

$$\begin{aligned} p(\neg Q) &= (-1)^{n_1+n_2} \sum_{\sigma\in\Sigma_{m_1,m_2}(\Phi)} \#\sigma \cdot \prod_{\tau_1\in LL,\tau_2\in RR} F_{\tau_1,\tau_2}^{\sigma^E(\tau_1,\tau_2)} \\ & \prod_{\tau_1\in LL} (\mu_{LL}(\tau_1,\hat{1})\cdot F_{\tau_1,*})^{\sigma^X(\tau_1)} \cdot \\ & \prod_{\tau_2\in RR} (\mu_{RR}(\tau_2,\hat{1})\cdot F_{*,\tau_1})^{\sigma^Y(\tau_1)} \end{aligned}$$

The proof is similar to the previous theorem and omitted.

K.3 Irreducible Polynomials

Consider a positive CNF formula $\varphi = \bigwedge_i C_i$, where each clause $C_i = X_1 \vee X_2 \vee \ldots$ is a disjunction of Boolean variables. We assume φ is non-redundant, i.e. no clause C_i is contained in any other clause C_j.

Definition K.10 *The* co-occurrence graph *of φ is the following: the nodes are the Boolean variables, and there is an (undirected) edge from X_i to X_j if they co-occur in a clause.*

Theorem K.11 *Let $f(x_1,\ldots,x_n)$ be the multilinear polynomial corresponding to φ. Write it as a product of irreducible factors: $f = g_1 g_2 \ldots g_m$: each variable x_i occurs in at most one irreducible (since f is multilinear). Then, forall i,j the variables x_i, x_j occur in the same irreducible iff X_i, X_j are connected in the co-occurrence graph.*

Proof: Assume X_i, X_j are disconnected in the co-occurrence graph. Hence $\varphi = \varphi_1 \wedge \varphi_2$ where φ_1 contains X_i, φ_2 contains X_j, and φ_1, φ_2 do not share any common variables. Then $f = f_1 f_2$, where x_i occurs in f_1 and x_2 occurs in f_2.

Assume x_i, x_j occur in different irreducible factors. Write $f = f_1 f_2$, and let φ_1, φ_2 be the (minimal) CNF formulas corresponding to f_1 and f_2. Obviously $\varphi \equiv \varphi_1 \wedge \varphi_2$. $\Box$

The theorem also extends to non-monotone CNF, but in that case there is no notion of "reduced" formula, so stating the theorem requires more care. We don't need non-monotone CNF.

Let D be a probabilistic database, defining a probability distribution on possible worlds $W \subseteq D$.

Definition K.12 *The* colineage *of a query $Q = \bigvee_i q_i$ over a database D is the following CNF formula φ_D^Q.*

- *There is a Boolean variable X_t for each tuple $t \in D$; $X_t = 1$ iff $t \notin W$.*
- *For each conjunctive query q_i and each valuation θ that maps q_i to the tuples $t_1, t_2, \ldots$, there is a clause $X_{t_1} \vee X_{t_2} \vee \ldots$ in $\tilde{F}$.*
- *Thus:*

$$\varphi_D^Q = \bigwedge_{q_i,\theta} (X_{t_1} \vee X_{t_2} \vee \ldots)$$

Note that the colineage says that the query is false:

$$p(\neg Q) = p(\varphi_D^Q)$$

It follows that the irreducibles of $p(\neg Q)$ are the same as the irreducibles of $p(\varphi_D^Q)$. We drop the superscript and/or the subscript from φ_D^Q when it is clear from the context.

K.4 Constructing the Blocks

We now describe how to construct the blocks $D(a, b)$. This construction is similar to that in the proof of Prop. J.7.

Given two constants u, v denote $B(u, v)$ the canonical database for Q over the two constants u and v, i.e. it consists of the following tuples:

- The tuple $R(u)$ (if the left end of Q is of type 1).
- All tuples $S(u, v)$, for all binary symbols S.
- The tuple $T(v)$ (if the right end of Q is of type 2).

Given two sets of constants U, V, denote:

$$B(U, V) = \bigcup_{u \in U, v \in V} B(u, v)$$

Let n_x be the maximum number of x_i variables, and n_y the maximum number of y_i variables in any component q. Let U, V be two sets of constants s.t. $|U| = n_x$, $|V| = n_y$. We assume that these sets are disjoint from X, Y (the nodes in the bipartite graph Φ). Fix two constants $a \in X, b \in Y$, and define the following blocks, which we call the *generic databases*:

$$\begin{aligned}
G(a, b) &= B(a, V) \cup B(U, V) \cup B(U, b) \\
G(a, *) &= B(a, V) \cup B(U, V) \cup B(U, *) \\
G(*, b) &= B(*, V) \cup B(U, V) \cup B(U, b)
\end{aligned}$$

For each tuple in $G(a, b)$ denote $x \in [0, 1]$ its probability. Let $\bar{x}$ denote the probabilities of all tuples in $G(a, b)$; we view $\bar{x}$ as a set of unknowns. Similarly, $\bar{y}, \bar{z}$ denote the probabilities of all tuples in $G(a, *), G(*, b)$. Let:

$$\begin{aligned} f_{\tau_1,\tau_2}(\bar{x}) &= p_{G(a,b)}(Q^{a,b}_{\tau_1,\tau_2}) \\ f_{\tau_1,*}(\bar{y}) &= p_{G(a,b)}(Q^{a}_{\tau_1,*}) \\ f_{*,\tau_2}(\bar{z}) &= p_{G(a,b)}(Q^{b}_{*,\tau_2}) \end{aligned}$$

Let $m = m_1 m_2$, and define:

- Forall $k = 1, m$: $D_k(a, b)$ is an isomorphic copy of $G(a, b)$ and $\bar{x}_k$ are its probabilities.
- Forall $k = 1, m_1$: $D_k(a, *)$ is an isomorphic copy of $G(a, *)$ and $\bar{y}_k$ are its probabilities.
- Forall $k = 1, m_2$: $D_k(*, b)$ is an isomorphic copy of $G(*, b)$ and $\bar{z}_k$ are its probabilities.

Finally, define:

$$\begin{aligned} D(a, b) &= \bigcup_k D_k(a, b) \\ D(a, *) &= \bigcup_k D_k(a, *) \\ D(*, b) &= \bigcup_k D_k(*, b) \end{aligned}$$

Lemma K.13

$$\begin{aligned} F_{\tau_1,\tau_2}(\bar{x}_1, \ldots, \bar{x}_m) &= \prod_{k=1,m} f_{\tau_1,\tau_2}(\bar{x}_k) \\ F_{\tau_1,*}(\bar{y}_1, \ldots, \bar{y}_{m_1}) &= \prod_{k=1,m_1} f_{\tau_1,*}(\bar{y}_k) \\ F_{*,\tau_2}(\bar{z}_1, \ldots, \bar{z}_{m_2}) &= \prod_{k=1,m_2} f_{*,\tau_2}(\bar{z}_k) \end{aligned}$$

Proof: Consider the following event: for a random world $W \subseteq D(a, b)$, $W \models \neg Q^{a,b}_{\tau_1,\tau_2}$. This is the conjunction of the following independent events: $W \cap D_k(a, b) \models Q^{a,b}_{\tau_1,\tau_2}$, since the query $Q^{a,b}_{\tau_1,\tau_2}$ only depends on binary tuples (any unary tuples $R(x), T(y)$ are set to either `false` or `true` in $Q^{a,b}_{\tau_1,\tau_2}$). Hence, these events are independent, which implies the claim in the lemma. $\square$

Definition K.14 *Let q be a conjunctive query whose set of variables is $X \cup Y$, and let U, V be two sets of constants. Denote*

$$q[U, V] = \bigvee_{\theta_x : X \to U, \theta_y : Y \to V} \theta(q)$$

Thus, $q[U, V]$ is a ground query, where all atoms are grounded.

We will now study the irreducible factors in the multilinear polynomials f_{τ_1,τ_2}, $f_{\tau_1,*}$, f_{*,τ_2}. Let:

$$Q = QL \vee QC \vee QR$$

where QL contains all left components, QC contains all center components, and QR contains all right components. Consider the generic database $G(a, b) = B(a, V) \cup B(U, V) \cup B(U, b)$.

$$\begin{aligned}
\tilde{Q}_{\tau_1,\tau_2} = {} & QL[a, V] \vee QC[a, V] \vee QR[a \cup U, V] \\
& \vee QC[U, V] \vee \\
& QL[U, V \cup b] \vee QC[U, b] \vee QR[U, b]
\end{aligned}$$

This is a grounded query, where all atoms are grounded atoms (i.e. have no variables). We minimize this query: some of the grounded components are redundant and thus are eliminated during minimization. However:

- If q is a left component, then every grounding in $q[a, V]$ or in $q[U, V]$ that uses distinct constants v_i for distinct variables y_i is non-redundant in $\tilde{Q}$.
- If q is center component, then every grounding in $q[a, V]$, $a[U, V]$, $q[U, b]$ is non-redundant in $\tilde{Q}$.
- If q is a right component, then every grounding in $q[U, V]$ or in $q[U, b]$ that uses distinct constants u_i for distinct variables x_i is non-redundant in $\tilde{Q}$.

In particular, this implies that the connection graph of $\tilde{Q}$ has a path from left atom $S(a, v)$ to any right atom $S'(u, b)$.

For every $\tau_1 \in LL, \tau_2 \in RR$ define:

$$\begin{aligned}
\tilde{Q}_{\tau_1,\tau_2} &= (\tau_1[a, V] \vee \tilde{Q} \vee \tau_2[U, b])[R(a) = \varepsilon^{\tau_1}, T(b) = \varepsilon^{\tau_2}] \\
\tilde{Q}_{\tau_1,*} &= (\tau_1[a, V] \vee \tilde{Q})[R(a) = \varepsilon^{\tau_1}] \\
\tilde{Q}_{*,\tau_2} &= (\tilde{Q} \vee \tau_2[U, b])[T(b) = \varepsilon^{\tau_2}]
\end{aligned}$$

It follows that:

$$\begin{aligned} f_{\tau_1,\tau_2}(\bar{x}) &= p_{G(a,b)}(\tilde{Q}_{\tau_1,\tau_2}) \\ f_{\tau_1,*}(\bar{y}) &= p_{G(a,b)}(\tilde{Q}_{\tau_1,*}) \\ f_{*,\tau_2}(\bar{z}) &= p_{G(a,b)}(\tilde{Q}_{*,\tau_2}) \end{aligned}$$

Recall that the variables of f_{τ_1,τ_2} correspond to tuples in $D(a,b)$. Therefore, a subset of these variables correspond to tuples in $QC[U,V]$. Similarly, a subset of the variables of $f_{\tau_1,*}$ correspond to tuples of $QC[U,V]$, and similarly for f_{*,τ_2}.

Proposition K.15 *Let x be any variable of f_{τ_1,τ_2} corresponding to a tuple in $QC[U,V]$. Denote $f^0_{\tau_1,\tau_2}$ the irreducible factor containing x. Then these factors are inequivalent for distinct labels τ_1,τ_2. More precisely: if $f^0_{\tau_1,\tau_2} \equiv f^0_{\tau_1',\tau_2'}$ then $\tau_1 = \tau_1'$ and $\tau_2 = \tau_2'$.*

Let y be any variable of $f_{\tau_1,*}$ corresponding to a tuple in $QC[U,V]$ and $f^0_{\tau_1,*}$ is the irreducible factor containing y. Then $f^0_{\tau_1,*} \equiv f^0_{\tau_1',*}$ implies $\tau_1 = \tau_1'$.

Let z be any variable of f_{*,τ_2} corresponding to a tuple in $QC[U,V]$ and $f^0_{\tau_1,*}$ is the irreducible factor containing z. Then $f^0_{*,\tau_2} \equiv f^0_{*,\tau_2'}$ implies $\tau_2 = \tau_2'$.

Proof: We prove the first statement only, the other two are similar. Suppose the left end of Q is of type 1. Then $\tau_1[a,V][R(a) = \varepsilon^{\tau_1}] \equiv \texttt{false}$. Thus, τ_1 affects only $QL[a,V][R(a) = \varepsilon^{\tau_1}]$, as follows. When $\tau_1 = R(x)$, then it sets $R(a) = 0$ and thus removes QL completely. In this case the connection graph of $\tilde{Q}_{\tau_1,\tau_2}$ remains connected on the left. Thus, $f^0_{R(x),\tau_2}$ contains all variables corresponding to all tuples in $Q[a,V] \vee QC[U,V]$ (and possibly more). When $\tau_1 = \neg R(x)$ then it sets $R(a) = 1$, and some of the center components $QC[a,V]$ may become redundant. Note that the grounded queries in $QC[U,V]$ are not affected, i.e. $f^0_{\neg R(x),\tau_2}$ contains all variables corresponding to the tuples in $QC[U,V]$. Moreover, $\tilde{Q}_{R(x),\tau_2} \not\equiv Q_{\neg R(x),\tau_2}$, because $\tilde{Q}$ depends on $R(a)$, and therefore $f_{R(x),\tau_2} \not\equiv f_{\neg R(x),\tau_2}$.

Suppose now that the left end of Q is of type 2. Then the assignments $[R(a) = 0]$ or $[R(a) = 1]$ are vacuous, and we will omit them. Here $\tau_1[a,V] \vee QL[a,V] \equiv \tau_1[a,V]$. That is, all queries in $QL[a,V]$ are redundant; some queries in $QC[a,V]$ may also become redudant. However, recall that there exists a useless left symbol S and there exists a center component $q \in QC$ that contains all useful symbols S_1. Any instance $q[a,v]$ is non-redudant, because every conjunctive query in $\tau_1[a,V]$ contains the useless symbol S, and therefore cannot map to $q[a,v]$. Thus, all symbols in $QC[U,V]$ remain connected to $q[a,v]$, and therefore all symbols in $QL[a,V], QC[a,V], QR[a,V], QC[U,V]$ are connected. Therefore, $f^0_{\tau_1,\tau_2}$ contains all their corresponding variables. Furthermore, for $\tau_1 \not\equiv \tau_1'$ we have $\tilde{Q}_{\tau_1,\tau_2} \not\equiv \tilde{Q}_{\tau_1',\tau_2}$, and therefore $f^0_{\tau_1,\tau_2} \not\equiv \tilde{f}^0_{\tau_1',\tau_2}$. $\Box$

Consider the generic database $G(a,b)$, and fix any tuple in $QC[U,V]$: call it the distinguished tuple, and let x be its variable. From Theorem I.7 we conclude that there are values $\bar{v}$ of all other tuple variables in $G(a,b)$ s.t. after substituting those variables with $\bar{v}$, the resulting linear polynomials $f_{\tau_1,\tau_2}(x)$ in the single variable x are inequivalent. We use the same values $\bar{v}$ for all non-distinguished tuples in all copies $D_k(a,b)$ of the generic block $G(a,b)$: that is the probabilities of all tuples in $D_k(a,b)$

are set according to $\bar{v}$. The distinguished tuples will have distinct probabilities, set to some unknown value $x_k \in [0,1]$, for $k = 1, m$. We repeat the same argument to choose m_1 distinguished tuples in $D(a,*)$, and similarly for m_2 distinguished tuples in $D(*,b)$. It follows that:

$$\begin{array}{rcl} F_{\tau_1,\tau_2} & = & f_{\tau_1,\tau_2}(x_1)\cdots f_{\tau_1,\tau_2}(x_m) \\ F_{\tau_1,*} & = & f_{\tau_1,*}(y_1)\cdots f_{\tau_1,*}(y_{m_1}) \\ F_{*,\tau_2} & = & f_{*,\tau_2}(z_1)\cdots f_{*,\tau_2}(z_{m_2}) \end{array}$$

K.5 Completing the Proof

Let Q be a normalized, minimized forbidden query; it can be of type 1-1, or 2-1, or 2-2 (the case 1-2 is similar to 2-1). We use an oracle for computing $P(Q)$ to solve an instance Φ of the m_1, m_2-Signature Counting problem of the same type. First we construct a database D consisting of blocks $D(a,b)$ (and possibly $D(a,*)$ and $D(*,b)$, depending on the type of Q). This database has $m = m_1 m_2$ unknown probabilities $\bar{x} = (x_1, \ldots, x_m)$, (and possibly m_1 unknown probabilities $\bar{y}$ and m_2 probabilities $\bar{z}$). For that, we use Theorem I.1 to find n^m values for the variables $\bar{x}$ (or $n^m n_1^{m_1}$ values for $\bar{x}, \bar{y}$, or $n^m n_1^{m_1} n_2^{m_2}$ values for $\bar{x}, \bar{y}, \bar{z}$) such that the matrix M of the system of equations given by the formula in Theorem K.7 (or Theorem K.8 or Theorem K.9) is non-singular.

We must prove that such values exists, and for that we use Theorem I.2. In turn, this requires us to check that the following Jaboians are non-zero:

$$\begin{array}{rcl} (x_1, \ldots, x_m) & \to & ((F_{\tau_1,\tau_2})_{\tau_1 \in LL, \tau_2 \in RR}) \\ (x_1, \ldots, x_m, y_1, \ldots, y_{m_1}) & \to & ((F_{\tau_1,\tau_2})_{\tau_1 \in LL, \tau_2 \in RR}, (F_{\tau_1,*})_{\tau_1 \in LL}) \\ (x_1, \ldots, x_m, y_1, \ldots, y_{m_1}, z_1, \ldots, z_{m_2}) & \to & ((F_{\tau_1,\tau_2})_{\tau_1 \in LL, \tau_2 \in RR}, (F_{\tau_1,*})_{\tau_1 \in LL}, (F_{*,\tau_2})_{\tau_2 \in RR}) \end{array}$$

The first follows immediately from Theorem I.6 and the fact that we have chosen the distinguished tuple in $G(a,b)$ for the variable x and the values of the other variables such that the conditions in Theorem I.7 are met. The same argument implies that the mapping $\bar{y} \to (F_{\tau_1,*})_{\tau_1 \in LL}$ has a non-zero Jacobian, hence the combined mapping given in line 2 above has a non-zero Jacobian. Same for the third line.

This completes the proof.