Management of Probabilistic Data Foundations and Challenges[*]

Nilesh Dalvi
University of Washington
Seattle, WA
nilesh@cs.washington.edu

Dan Suciu
University of Washington
Seattle, WA
suciu@cs.washington.edu

ABSTRACT

Many applications today need to manage large data sets with uncertainties. In this paper we describe the foundations of managing data where the uncertainties are quantified as probabilities. We review the basic definitions of the probabilistic data model, present some fundamental theoretical result for query evaluation on probabilistic databases, and discuss several challenges, open problems, and research directions.

Categories and Subject Descriptors

F.4.1 [**Mathematical Logic**]; G.3 [**Probability and statistics**]; H.2.5 [**Heterogeneous databases**]

General Terms

Algorithms, Management, Theory

Keywords

probabilistic databases, query processing

1. UNCERTAINTIES IN THE FUTURE

We know well how to manage data that is deterministic. Databases were invented to support applications like banking, payroll, accounting, inventory, all of which require a precise semantics of the data. In the future we need to learn how to manage data that is imprecise, or uncertain, and that contains an explicit representation of the uncertainty. While data management has gone through several "paradigm shifts" in the past (NF2, OO databases, semistructured data), this one is arguably more fundamental: the foundations of the new paradigm no longer lie solely in Logic and finite model theory, but also in probabilistic inference [17], probability logic [2], and degrees of belief [8].

One can find evidence in many areas that such a paradigm shift is needed. We mention here two such areas.

Data Integration A central research topic for many years (see Halevy et al. [46]), data integration has reached the point where it needs to cope explicitly with uncertainties [82]. Precise data mappings and schema mappings are increasingly hard to build on a large scale. As an example, in a recent paper in the journal Nature Reviews Genetics, Philippi and Kohler [66] discuss the difficulties in integrating life-science databases, and the biggest obstacle they identify is imprecision in the data. Data mappings are difficult because there is no uniform conceptualization of the data, and in the databases themselves are incomplete (*e.g.* missing functional annotations), or uncertain (*e.g.* due to automatic, non-curated annotation). The common practice today is for a life-science researcher to manage uncertainties manually: they query one data source, say with a keyword, rank the answers manually based on their quality or relevance to the query, then feed the answers as queries into the next data source, ranking those results etc. To do this automatically, a system needs to represent and manipulate the uncertainties explicitly, and use them to rank the data items. Representing uncertainties explicitly is a cheap, perhaps temporary substitute for more precise integration, say one based on ontologies, a form of *pay as you go* data integration [35, 45, 61].

Novel Data We face increasingly large volumes of data generated by non-traditional means, which is almost always imprecise. Google base [84], Flickr [83], the ESP game [79], and the Mechanical Turk [85] are examples of *data generated by the masses*, who are often free to deviate from the representation guidelines suggested by the system. Large scale *information extraction* systems [33, 31, 53] extract data automatically from text, and this data is always imprecise. What may soon overtake any other kind of data in volume is *data from the physical world*, which originates from cheap sensors, cameras, RFID tags. There is emerging work to make this data available on a World-Wide Sensor Web [11, 32]. Sensor data is imprecise by its very nature and has already been modeled probabilistically [29, 30]. RFID deployments have high error rates, *e.g.* [54] reports read rates in the 60 - 70% range, and the high-level data compounded from such readings is also best modeled probabilistically [60, 14, 56].

The techniques that will allow us to manage data with uncertainties need to be grounded in a formalism that com-

[*]This research was supported in part by Suciu's NSF CAREER grant IIS-0092955, NSF grants IIS-0415193, IIS-0627585, IIS-0513877, IIS-0428168, and a gift from Microsoft.

Inference		AI	Databases
Deterministic		Theorem prover	Query evaluation
Probabilistic	Exact	Hardness [16] Algorithms [17]	PTIME for safe queries (Th. 3.9)
	Approx	Hardness [18] Algorithms [63]	PTIME for conjunctive queries (Th. 3.4)

Figure 1: Inference (deterministic and probabilistic) in AI and in Databases

bines logic with some measure of the uncertainty. The AI community has already gone through a transition from logic to logic with uncertainty, and it is natural to expect data management to follow a similar path; after all, we are both modeling data and knowledge.

A Paradigm for Managing Uncertainties

We describe here a general paradigm for data with uncertainties. We make no assumptions about the source or the nature of the uncertainties: it could be that the data in the real world is uncertain and the database simply records this uncertainty, or it could be that the real data is deterministic but our knowledge about it is uncertain for various reasons. The model simply assumes that the data is uncertain.

The AI community has considered several approaches to representing uncertainties, *e.g.* rule-based systems, Dempster-Shafer, fuzzy sets, but by 1987 the probability model has been dominant [48]. We only consider here modeling uncertainties as probabilities. Thus, in our model the data is probabilistic, where the probabilities are internal measures of the imprecision in the data. Users formulate queries using a standard query language, as if the data were precise. The system computes the answers, and for each answer computes an output probability score representing its confidence in that answer. These scores are then indicated somehow to the user, for example by returning them explicitly, or by ranking the answers in decreasing order of their scores, or by indicating the scores using some color code.

The central problem discussed in this paper is the following: given a boolean query q on a probabilistic database PDB, compute the probability of q on PDB, in notation $\mathbf{P}(q)$. We consider only Boolean queries, because the probability of an answer a to a non-Boolean query $q(x)$ can be obtained by evaluating the boolean query $q[a/x]$ (substitute a for x). Our interest is in studying the data complexity [77]: fix q and study the complexity of computing $\mathbf{P}(q)$ as a function of the size of the probabilistic database PDB. We start by establishing a simple connection to the problem of computing the probability of a boolean expression Φ, $\mathbf{P}(\Phi)$. This has been studied in the literature: it is #P-complete [75] and DNF formulas admit a FPTRAS (fully poly-time randomized approximation scheme) [55]. It follows that for every fixed query q computing $\mathbf{P}(q)$ is in #P, and for every fixed conjunctive query q, computing $\mathbf{P}(q)$ admits a FPTRAS. We then study the data complexity for specific queries q, and show that the set of conjunctive queries without self-joins has a dichotomy: q is either in PTIME (and then we call it *safe*) or is #P-hard (and we call it *unsafe*), and, moreover, deciding between the two cases can be done in PTIME in the size of q. Thus, a query processor can examine q and, either evaluate it exactly, when q is safe, or evaluate it approximatively, when it is unsafe.

While probabilistic databases have a long history, there has been no systematic study of the query evaluation problem. Early systems have avoided addressing the full query evaluation problem by either restricting the queries [9], or using an exponential-time evaluation algorithm [36], or relying on a weaker semantics based on intervals [59].

Probabilistic Inference v.s. Query Evaluation

A problem related to query evaluation studied by the AI community is inference in probabilistic networks[1]. Probabilistic inference remains very expensive to this date [58]. There is an important distinction between probabilistic inference in AI and query evaluation over probabilistic data, which mirrors the distinction between inference in knowledge bases and query evaluation in data bases. In AI there is no separation between the query and the data, and the complexity of the inference problem is formulated in terms of the size of the probabilistic network Φ: both exact inference [16] and approximate inference [18] are hard. In database we have two inputs, the query q and the probabilistic database PDB, and the system combines them at runtime to generate a probabilistic network Φ_q^{PDB}. The complexity is measured by fixing q and varying PDB: the latter is large, but usually has a simple probabilistic structure. As we saw, there are important tractable cases: for every conjunctive query approximate inference is tractable [40], and for every *safe* conjunctive query exact inference is tractable (Sec. 3). Thus, query evaluation on probabilistic databases is distinct from inference in probabilistic networks, in the same way in which query processing in relational databases is distinct from logical inference in knowledge bases, see Fig. 1.

Two Applications

We briefly illustrate how uncertainty in the data can be modeled with probabilities in two applications. We refer the reader to [38, 74] for more examples.

In *fuzzy object matching* the problem is to reconcile objects from two collections that have used different naming conventions. This is a central problem in data cleaning and data integration, and has also been called record linkage, deduplication, or merge-purge [34, 3, 13, 37, 43, 49, 15, 80, 7]. The basic approach is to compute a similarity score between pairs of objects, usually by first computing string similarity scores on their attributes, then combining these scores into a global score for the pair of objects. Next, this score is compared with a threshold, and each pair of objects is classified into a *match*, a *non-match*, and a *maybe-match* [3]. Ré et al. [68] propose to convert these scores into probabilities[2] and store them directly in a probabilistic database. There is no more need for a threshold: instead all potential matches are stored together with their probabilities, then used during query processing to rank the answers. Fig 2 (a) illustrates a small fragment of such a match table between movie titles from IMDB and reviews for DVD products from Amazon.

An *information extraction* system processes a collection of text documents and populates a user-defined schema [71]. All approaches to extraction are imprecise, and usually associate a probability to each extraction. Gupta and Sarawagi [44]

[1]The term *probabilistic network* is used in AI to denote a graph whose nodes represent random variables and whose edges represent correlations [17]. Examples are: boolean expressions, Bayesian Networks, and Markov Networks.
[2]They are often already expressed as a probability.

Reproduced from [68]:

Review	Movie	**P**
12 Monkeys	Twelve Monkeys	0.4
12 Monkeys	Twelve Monkeys (1995)	0.3
12 Monkeys	Monk	0.013
Monkey Love	Twelve Monkeys	0.35
Monkey Love	Love Story	0.27

(a)

Reproduced from [44]:

`...52 A Goregaon West Mumbai ...`

ID	House-No	Area	City	**P**
1	52	Goregaon West	Mumbai	0.1
1	52-A	Goregaon West	Mumbai	0.5
1	52	Goregaon	West Mumbai	0.2
1	52-A	Goregaon	West Mumbai	0.3
2	7	Westlake	...	...

(b)

Figure 2: Two applications of probabilistic data. Fuzzy object matching: each review matches several candidate movies (a). Information extraction: each address has several possible segmentations (b).

describe how to use a probabilistic database to store the result of text segmentation with Conditional Random Fields: Fig. 2 (b) illustrates the possible segmentations of an address, together with their probabilities. Importantly, it has been shown in [44] that the probabilities computed by CRFs correlate with the probability that the extracted instance is correct, *i.e.* if one manually inspects all extracted items that have probability between, say, 0.3 and 0.35, then one finds that approximatively $30-35\%$ of them are correct in a given instance. In other words, the probabilities have meaning.

What these applications have in common is a need to store and manage data with a probabilistic semantics. Probabilities are already present in these applications, thus we are not concerned with where the probabilities are coming from. Instead, our goal is to allow users to ask queries over the data, and to compute the probability for each answer to the query. We give the definition of the data model next, then describe the query evaluation problem.

2. THE POSSIBLE WORLDS DATA MODEL

We review here the definition of a probabilistic database based on possible worlds, and of a disjoint-independent database. Throughout this paper we restrict our discussion to relational data over a finite domain: extensions to continuous domains [29] and to XML [50, 1, 76] have also been considered.

We fix a relational schema $\mathcal{R} = (R_1, \ldots, R_k)$, where R_i is a relation name, has a set of attributes $Attr(R_i)$, and a key $Key(R_i) \subseteq Attr(R_i)$; we explain below why we include keys in our data model. Denote D a finite domain of atomic values. For $i = 1, \ldots, k$ denote Tup_i the set of typed atomic tuples of the form $R_i(a_1, \ldots, a_k)$ where $a_1, \ldots, a_k \in D$ and $k = |Attr(R_i)|$ is the arity of R_i, and denote $Tup = \bigcup_i Tup_i$ the domain of all tuples. If $t \in Tup_i$ then $Key(t)$ represents a typed tuple of arity $|Key(R_i)|$ consisting of the key attributes in t. A database instance is any subset $I \subseteq Tup$ that satisfies all key constraints.

To illustrate, $R(\underline{A}, \underline{B}, C)$, $S(\underline{A}, \underline{D})$, $T(\underline{A}, \underline{B}, C)$ is a schema, the underlined attributes are the keys, and $t_1 = R(a, b, c)$, $t_2 = R(a, b, c')$, $t_3 = T(a, b, c)$ are three distinct atomic tuples, which we sometimes write as $R(\underline{a, b}, c)$ to emphasize the key attributes. We have $Key(t_1) = Key(t_2) \neq Key(t_3)$: the latter because we consider $Key(t)$ to be a typed tuple.

The main idea in a probabilistic database is that the state of the database, *i.e.* the instance I is not known. Instead the database can be in any one of a finite number of possible states $I_1, I_2, \ldots$, called *possible worlds*, each with some probability.

Definition 2.1. *A* probabilistic database *is a probability space* $PDB = (W, \mathbf{P})$ *where the set of outcomes is a set of* possible worlds $W = \{I_1, \ldots, I_n\}$. *In other words, it is a function* $\mathbf{P} : W \to (0, 1]$ *s.t.* $\sum_{I \in W} \mathbf{P}(I) = 1$.

We review basic notions in probability theory in the Appendix. Fig. 3 illustrates three possible worlds of a probabilistic database: the probabilities of all worlds must sum up to 1. The intuition is that we have a database with schema $R(\underline{A, B}, C, D)$, but we are not sure about the content of the database: there are several possible contents, each with a probability.

A Boolean query q defines the event $\{I \mid I \models q\}$ over a probabilistic database, and its marginal probability is $\mathbf{P}(q) = \sum_{I \in W | I \models q} \mathbf{P}(I)$. A tuple t is a special case of a Boolean query and its marginal probability is $\mathbf{P}(t) = \sum_{I \in W | t \in I} \mathbf{P}(I)$. Note that $t \neq t'$ and $Key(t) = Key(t')$ implies $\mathbf{P}(t, t') = 0$, *i.e.* t, t' are disjoint events.

Disjoint-Independent Databases In practice we cannot enumerate all possible worlds, since there are usually too many. The representation problem for probabilistic database is an active research topic [26, 10, 41, 6, 5], and Green and Tannen [41] observed a strong connection between representation systems for probabilistic databases and for incomplete database. We restrict our discussion to disjoint-independent databases, which have a simple representation.

A *possible tuple* is a tuple that occurs in at least one possible world, and we denote T the set of possible tuples. PDB is *disjoint-independent* if any set of possible tuples with distinct keys is independent: $\forall t_1, \ldots, t_n \in T$, $Key(t_i) \neq Key(t_j)$ for $i \neq j$ implies $\mathbf{P}(t_1, \ldots, t_n) = \mathbf{P}(t_1) \cdots \mathbf{P}(t_n)$. A disjoint-independent database is represented by enumerating the set of possible tuples T and their marginal probabilities $\mathbf{P}(t)$: this uniquely determines the probability $\mathbf{P}(I)$ of each instance $I \subseteq T$[3]. We denote a disjoint-independent database as $PDB = (T, \mathbf{P})$, rather than $(W, \mathbf{P})$, emphasizing that it is given by enumerating the set of possible tuples rather than the possible worlds. Its *size* is defined as the size of T. Fig. 4 shows the representation of a disjoint-independent database: it has 16 possible worlds, and three of them are precisely those shown in Fig. 3. There are seven possible tuples, each with some probability and it is convenient to group the possible tuples by their keys, A, B, to emphasize that at most one can be chosen in each group. The two tables in Fig. 2 are also disjoint-independent. From now on, throughout the paper a *database* means a *disjoint-independent* probabilistic database. We call the database

[3]$\mathbf{P}(I)$ is a product with one factor for each key k in T: if some tuple $t \in I$ has key k, then that factor is $\mathbf{P}(t)$; otherwise it is $1 - \sum_{t \in T, Key(t) = k} \mathbf{P}(t)$.

I_1

<u>A</u>	<u>B</u>	C	D
a_1	b_1	c_1	d_1
a_2	b_1	c_3	d_1
a_2	b_2	c_4	d_2

$\mathbf{P}(I_1) = 0.06$
$(= p_1 p_3 p_6)$

I_2

<u>A</u>	<u>B</u>	C	D
a_1	b_1	c_2	c_2
a_2	b_1	c_2	c_1
a_2	b_2	c_4	c_2

$\mathbf{P}(I_2) = 0.12$
$(= p_2 p_5 p_6)$

I_3

<u>A</u>	<u>B</u>	C	D
a_1	b_1	c_1	d_1
a_2	b_2	c_4	d_2

$\mathbf{P}(I_3) = 0.04$
$(= p_1(1\text{-}p_3\text{-}p_4\text{-}p_5)p_6)$

Figure 3: A probabilistic database $PDB = (\{I_1, I_2, I_3, \ldots\}, \mathbf{P})$ with schema $R(\underline{A,B},C,D)$; we show only three possible worlds.

<u>A</u>	<u>B</u>	C	D	**P**
a_1	b_1	c_1	d_1	$p_1 = 0.25$
		c_2	d_2	$p_2 = 0.75$
a_2	b_1	c_3	d_1	$p_3 = 0.3$
		c_1	d_3	$p_4 = 0.3$
		c_2	d_1	$p_5 = 0.2$
a_2	b_2	c_4	d_2	$p_6 = 0.8$
		c_5	d_2	$p_7 = 0.2$

Figure 4: Representation of a disjoint-independent probabilistic database. The seven possible tuples are grouped by their keys, for readability. There are 16 possible worlds; three are shown in Fig. 3.

independent if $Key(R_i) = Attr(R_i)$ for all relation symbols R_i, *i.e.* there are no disjoint tuples.

Uncertainty at the Attribute Level Many probabilistic database systems [9, 59, 29, 30, 70, 5] need to capture uncertainties at the attribute rather than the tuple level. For example, in a table $S(A, B)$ the A-value of some tuple is known to be a_1, but the B-value attribute can be b_1, b_2, b_3, with probabilities p_1, p_2, p_3: we must have $p_1 + p_2 + p_3 = 1$ since it is certain that a tuple of the form $(a_1, -)$ is in the database, only the value of B is uncertain. There exists a simple mapping from attribute-level uncertainties to a disjoint-independent database, as illustrated below:

A	B
a_1	$\langle b_1, p_1\rangle, \langle b_2, p_2\rangle, \langle b_3, p_3\rangle$
a_2	$\langle b_2, p_4\rangle, \langle b_9, p_5\rangle, \ldots$

$\underline{A}$	B	**P**
a_1	b_1	p_1
a_1	b_2	p_2
a_1	b_3	p_3
a_2	b_2	$\ldots$

Note that we had to make A a key to ensure that the three values for B are disjoint. The reason why we have keys in our data model is to represent attribute-level uncertainties. All the hardness results in Sec. 3 extend easily to attribute-level uncertainties.

A *p-or-set* [41] is a database where for every possible key k, $\sum_{t \in T, Key(t)=k} \mathbf{P}(t) = 1$. S above is a p-or-set.

3. FOUNDATIONS OF QUERY EVALUATION

We study the query evaluation problem: given a query q and a disjoint-independent database PDB, compute $\mathbf{P}(q)$. We first describe the connection between this problem and the probability of Boolean formulas, which has been studied in the literature; in particular it follows that for any fixed q, computing $\mathbf{P}(q)$ is in #P in the size of PDB. We then analyze the precise complexity of $\mathbf{P}(q)$ for a fixed query, and establish a dichotomy for conjunctive queries without self-joins: every query is either #P-hard or in PTIME.

From Queries to Boolean Formulas

Consider n Boolean variables $\bar{X} = \{X_1, \ldots, X_n\}$. A disjoint-independent probability space with outcomes $2^{\bar{X}}$ (the set of truth assignments) is given by a partition $\bar{X} = \bar{X}_1 \cup \ldots \cup \bar{X}_m$ and a probability function $\mathbf{P} : \bar{X} \to (0, 1]$ s.t. $\forall j = 1, m$, $\sum_{X \in \bar{X}_j} \mathbf{P}(X) \leq 1$. A truth assignment is chosen at random by independently choosing in each set $\bar{X}_j$ at most one variable X_i that will be set to *true*, while all others are set to false: namely X_i is chosen with probability $\mathbf{P}(X_i)$. We will assume that all probabilities are given as rational numbers, $\mathbf{P}(X_i) = p_i/q_i$, $i = 1, n$, where p_i, q_i are integers.

We call this probability space *independent* if $\forall j$, $|\bar{X}_j| = 1$ (*i.e.* there are no disjoint variables), and *uniform* if it is independent and $\forall i, \mathbf{P}(X_i) = 1/2$.

Let Φ be a Boolean formula over $X_1, \ldots, X_n$; the goal is to compute its probability, $\mathbf{P}(\Phi) = \sum_{\theta \in 2^{\bar{X}}:\theta(\Phi)} \mathbf{P}(\theta)$.

The complexity class #P consists of problems of the following form: given an NP machine, compute the number of accepting computations [64]. Let $\#\Phi$ denote the number of satisfying assignments for Φ. Valiant [75] has shown that the problem: given Φ, compute $\#\Phi$, is #P-complete.

A statement like "computing $\mathbf{P}(\Phi)$ is in #P" is technically non-sense, because computing $\mathbf{P}(\Phi)$ is not a counting problem. However, as we show in the Appendix, there exists a polynomial-time computable function F over the integers $p_1, q_1, \ldots, p_n, q_n$, s.t. computing $F \cdot \mathbf{P}(\Phi)$ is in #P. For example, in the case of a uniform distribution, take $F = 2^n$, and computing $2^n\mathbf{P}(\Phi) = \#\Phi$ is in #P. In this sense:

Theorem 3.1. *Computing* $\mathbf{P}(\Phi)$ *is in #P.*

While computing $\mathbf{P}(\Phi)$ for a DNF Φ is still #P-hard, Luby and Karp [55] have shown that it has a FPTRAS (fully polytime randomized approximation scheme). More precisely: there exists a randomized algorithm A with inputs Φ, ε, δ, which runs in polynomial time in $|\Phi|$, $1/\varepsilon$, and $1/\delta$, and returns a value $\tilde{p}$ s.t. $\mathbf{P}_A(|\tilde{p}/p - 1| > \varepsilon) < \delta$. Here $\mathbf{P}_A$ denotes the probability over the random choices of the algorithm. Grädel et al. [40] show how to extend this to independent probabilities, and we show in the Appendix how to extend it to disjoint-independent probabilities:

Theorem 3.2. *Computing* $\mathbf{P}(\Phi)$ *for a disjoint-independent probability space* $\mathbf{P}$ *and a DNF formula* Φ *has a FPTRAS.*

We now establish the connection between the probability of Boolean expressions and the probability of a query on a disjoint-independent database. Let $PDB = (T, \mathbf{P})$ be a database with possible tuples $T = \{t_1, \ldots, t_n\}$. Associate a Boolean variable X_i to each tuple t_i and define a disjoint-independent probability space on their truth assignments by partitioning the variables X_i according to their keys (X_i, X_j are in the same partition iff $Key(t_i) = Key(t_j)$), and by defining $\mathbf{P}(X_i) = \mathbf{P}(t_i)$. This creates a one-to-one correspondence between the possible worlds of PDB and the truth assignments $2^{\bar{X}}$, which preserves the probabilities.

Consider a Boolean query, q, expressed in First Order Logic (FO) over the vocabulary $\mathcal{R}$. The *intensional formula* associated to q and database PDB is a Boolean formula Φ_q^{PDB}, or simply Φ_q when PDB is understood from the context, defined inductively as follows:

$$\begin{aligned}
\Phi_{R(\bar{a})} &= \begin{cases} X_i & \text{if } R(\bar{a}) = t_i \in T \\ false & \text{if } R(\bar{a}) \notin T \end{cases} \\
\Phi_{true} &= true \quad \Phi_{false} = false \quad \Phi_{\neg q} = \neg(\Phi_q) \\
\Phi_{q_1 \wedge q_2} &= \Phi_{q_1} \wedge \Phi_{q_2} \quad \Phi_{q_1 \vee q_2} = \Phi_{q_1} \vee \Phi_{q_2} \\
\Phi_{\exists x.q(x)} &= \bigvee_{a \in D} \Phi_{q[a/x]} \quad \Phi_{\forall x.q(x)} = \bigwedge_{a \in D} \Phi_{q[a/x]}
\end{aligned}$$

Here D represents the active domain of PDB, $q[a/x]$ denotes the formula q where the variable x is substituted with a, and interpreted predicates over constants (*e.g.* $a < b$ or $a = b$) are replaced by $true$ or $false$ respectively. If q has v variables, then the size of Φ_q is $O(|q| \cdot |D|^v)$. The connection between Boolean formulas and Boolean queries is:

PROPOSITION 3.3. *For every query q and any database $PDB = (T, \mathbf{P})$, $\mathbf{P}(q) = \mathbf{P}(\Phi_q)$.*

A *Boolean conjunctive query* is restricted to the connectives $\wedge$ and $\exists$, and, as usual, we write it as:

$$q = g_1, g_2, \ldots, g_k \tag{1}$$

where each g_i is a positive relational predicate called a *subgoal.* Obviously, in this case Φ_q is a positive DNF expression. For a simple illustration, suppose $q = R(x), S(x, y)$ and that we have five possible tuples: $t_1 = R(a)$, $t_2 = R(b)$, $t_3 = S(a, c)$, $t_4 = S(a, d)$, $t_5 = S(b, d)$ to which we associate the Boolean variables X_1, X_2, Y_1, Y_2, Y_3, then $\Phi_q = X_1Y_1 \vee X_1Y_2 \vee X_2Y_3$. Our discussion implies:

THEOREM 3.4. *For a fixed a Boolean query q, computing $\mathbf{P}(q)$ on a disjoint-independent database (1) is in #P, (2) admits a FPTRAS when q if conjunctive [40].*

Grädel et al. [40] stated that computing $\mathbf{P}(q)$ is in $\mathrm{FP}^{\#P}$ (the class of functions computable by a PTIME machine with a #P oracle). They also showed that conjunctive queries have a FPTRAS over independent databases.

A Dichotomy for Queries without Self-joins

We now establish the following dichotomy for queries without self-joins: computing $\mathbf{P}(q)$ is either #P-hard or is in PTIME in the size of the database $PDB = (T, \mathbf{P})$. A query q is said to be without self-joins if each relational symbol occurs at most once in the query body [21, 20]. For example $R(x, y), R(y, z)$ has self-joins, $R(x, y), S(y, z)$ has not.

Three Hard Queries We prove in the Appendix:

THEOREM 3.5. *For each of the queries below (where $k, m \geq 1$), computing $\mathbf{P}(q)$ is #P-hard in the size of the database:*

$$\begin{aligned}
h_1 &= R(\underline{x}), S(\underline{x, y}), T(\underline{y}) \\
h_2^+ &= R_1(\underline{x}, y), \ldots, R_k(\underline{x}, y), S(\underline{y}) \\
h_3^+ &= R_1(\underline{x}, y), \ldots, R_k(\underline{x}, y), S_1(x, \underline{y}), \ldots, S_m(x, \underline{y})
\end{aligned}$$

The underlined positions represent the key attributes (see Sec. 2), thus, in h_1 the database is tuple independent, while in h_2^+, h_3^+ it is disjoint-independent. When $k = m = 1$ then we omit the + superscript and write:

$$\begin{aligned}
h_2 &= R(\underline{x}, y), S(\underline{y}) \\
h_3 &= R(\underline{x}, y), S(x, \underline{y})
\end{aligned}$$

The significance of these three (classes of) queries is that the hardness of any other conjunctive query without self-joins follows from a simple reduction from one of these three (Lemma 3.7). By contrast, the hardness of these three queries is shown directly (by reducing Positive Partitioned 2DNF [67] to h_1, and PERMANENT [75] to h_2^+, h_3^+) and these proofs are more involved.

Previously, the complexity has been studied only for independent probabilistic databases. De Rougemont [27] claimed that it is is in PTIME. Grädel at al. [27, 40] corrected this and proved that the query $R(\underline{x}), R(\underline{y}), S_1(\underline{x, z}), S_2(\underline{y, z})$ is #P-hard, by reduction from regular (non-partitioned) 2DNF: note that this query has a self-join (R occurs twice); h_1 does not have a self-join, and was first shown to be #P-hard in [21]. The only prior discussion of the complexity on disjoint-independent databases is in [20]: we announced (without proof) the hardness of h_2 and of[4] $R(\underline{x}, y), S(\underline{z}, y)$, but missed h_3 as well as the +-variants.

A PTIME Algorithm We describe here an algorithm that evaluates $\mathbf{P}(q)$ in polynomial time in the size of the database, which works for some queries, and fails for others. We need some notations. $Vars(q)$ and $Sg(q)$ are the set of variables, and the set of subgoals respectively. If $g \in Sg(q)$ then $Vars(g)$ and $KVars(g)$ denote all variables in g, and all variables in the key positions in g: *e.g.* for $g = R(\underline{x, a, y}, x, z)$, $Vars(g) = \{x, y, z\}$, $KVars(g) = \{x, y\}$. For $x \in Vars(q)$, let $sg(x) = \{g \mid g \in Sg(q), x \in KVars(g)\}$. Given a database $PDB = (T, \mathbf{P})$, D is its active domain.

Algorithm 3.1 computes $\mathbf{P}(q)$ by recursion on the structure of q. If q consists of connected components q_1, q_2, then it returns $\mathbf{P}(q_1)\mathbf{P}(q_2)$: this is correct since q has no self-joins, e.g $\mathbf{P}(R(\underline{x}), S(\underline{y}, z), T(\underline{y})) = \mathbf{P}(R(\underline{x}))\mathbf{P}(S(\underline{y}, z), T(\underline{y}))$. If some variable x occurs in a key position in all subgoals, then it applies the independent-project rule: *e.g.* $\mathbf{P}(R(\underline{x})) = 1 - \prod_{a \in D}(1 - \mathbf{P}(R(\underline{a})))$ is the probability that R is nonempty. For another example, we apply an independent project on x in $q = R(\underline{x}, y), S(\underline{x, y})$: this is correct because $q[a/x]$ and $q[b/x]$ are independent events whenever $a \neq b$. If there exists a subgoal g whose key positions are constants, then it applies a disjoint project on any variable in g: *e.g.* x is such a variable in $q = R(\underline{x}, y), S(\underline{c, d}, x)$, and any two events $q[a/x]$, $q[b/x]$ are disjoint because of the S subgoal.

We illustrate the algorithm on the query below:

$$\begin{aligned}
q &= R(\underline{x}), S(\underline{x, y}), T(\underline{y}), U(\underline{u}, y), V(\underline{a}, u) \\
\mathbf{P}(q) &= \sum_{b \in D} \mathbf{P}(R(\underline{x}), S(\underline{x, y}), T(\underline{y}), U(\underline{b}, y), V(\underline{a}, b)) \\
&= \sum_{b \in D} \mathbf{P}(R(\underline{x}), S(\underline{x, y}), T(\underline{y}), U(\underline{b}, y))\mathbf{P}(V(\underline{a}, b)) \\
&= \sum_{b \in D} \sum_{c \in D} \mathbf{P}(R(\underline{x}), S(\underline{x, c}), T(\underline{c}), U(\underline{b}, c))\mathbf{P}(V(\underline{a}, b)) \\
&= \sum_{b \in D} \sum_{c \in D} \mathbf{P}(R(\underline{x}), S(\underline{x, c}))\mathbf{P}(T(\underline{c}))\mathbf{P}(U(\underline{b}, c))\mathbf{P}(V(\underline{a}, b)) \\
&= \sum_{b \in D} \sum_{c \in D} (1 - \prod_{d \in D}(1 - \mathbf{P}(R(\underline{d}))\mathbf{P}(S(\underline{d, c})))) \cdot \\
&\qquad \mathbf{P}(T(\underline{c}))\mathbf{P}(U(\underline{b}, c))\mathbf{P}(V(\underline{a}, b))
\end{aligned}$$

We call a query *safe* if algorithm `Safe-Eval` terminates successfully; otherwise we call it *unsafe*. Safety is a property

[4]This query *rewrites* $\Rightarrow^*$ h_2 hence is hard by Lemma 3.7.

Algorithm 3.1 `Safe-Eval`
Input: query q and database $PDB = (T, \mathbf{P})$
Output: $\mathbf{P}(q)$

```
1: Base Case: if $q = R(\bar{a})$
     return if $R(\bar{a}) \in T$ then $\mathbf{P}(R(\bar{a}))$ else 0
2: Join: if $q = q_1, q_2$ and $Vars(q_1) \cap Vars(q_2) = \emptyset$
     return $\mathbf{P}(q_1)\mathbf{P}(q_2)$
3: Independent project: if $sg(x) = Sg(q)$
     return $1 - \prod_{a \in D}(1 - \mathbf{P}(q[a/x]))$
4: Disjoint project: if $\exists g(x \in Vars(g), KVars(g) = \emptyset)$
     return $\sum_{a \in D} \mathbf{P}(q[a/x])$
5: Otherwise: FAIL
```

that depends only on the query q, not on the database PDB, and it can be checked in PTIME in the size of q by simply running the algorithm over an active domain of size 1, $D = \{a\}$. Based on our previous discussion, if the query is safe then the algorithm computes the probability correctly:

Proposition 3.6. *For any safe query q, the algorithm computes correctly $\mathbf{P}(q)$ and runs in time $O(|q| \cdot |D|^{|Vars(q)|})$.*

We first described `Safe-Eval` in [20], in a format more suitable for an implementation, by translating q into an algebra plan using joins, independent projects, and disjoint projects, and stated without proof the dichotomy property. Andritsos et al. [4] describe a query evaluation algorithm for a more restricted class of queries.

The Dichotomy Property We define below a rewrite rule $q \Rightarrow q'$ between two queries, where g, g' denote subgoals:

$$\begin{array}{rcll}
q & \Rightarrow & q[a/x] & \text{if } x \in Vars(q), a \in D \\
q & \Rightarrow & q_1 & \text{if } q = q_1, q_2, Vars(q_1) \cap Vars(q_2) = \emptyset \\
q & \Rightarrow & q[y/x] & \text{if } \exists g \in Sg(q), x, y \in Vars(g) \\
q, g & \Rightarrow & q & \text{if } KVars(g) = Vars(g) \\
q, g & \Rightarrow & q, g' & \text{if } KVars(g') = KVars(g), \\
 & & & Vars(g') = Vars(g), arity(g') < arity(g)
\end{array}$$

The intuition is that if $q \Rightarrow q'$ then evaluating $\mathbf{P}(q')$ can be reduced in polynomial time to evaluating $\mathbf{P}(q)$; the reduction is quite easy (see the Appendix) and implies:

Lemma 3.7. *If $q \Rightarrow^* q'$ and q' is #P-hard, then q is #P-hard.*

Thus, $\Rightarrow$ gives us a convenient tool for checking if a query is hard, by trying to rewrite it to one of the known hard queries. For example, consider the queries q and q' below: `Safe-Eval` fails immediately on both queries, *i.e.* none of its cases apply. We show that both are hard by rewriting them to h_1 and h_3^+ respectively. By abuse of notations we reuse the same relation name during the rewriting, *e.g.* S, T, in the second and third line denote different relations:

$$\begin{array}{rcll}
q & = & R(\underline{x}), R'(\underline{x}), S(\underline{x, y}, y), T(\underline{y, z, b}) & \\
 & \Rightarrow & R(\underline{x}), S(\underline{x, y}, y), T(\underline{y, z, b}) & \\
 & \Rightarrow^* & R(\underline{x}), S(\underline{x, y}), T(\underline{y}) & = h_1 \\
q' & = & R(\underline{x}, y), S(\underline{y}, z), T(\underline{z}, x), U(\underline{y}, x) & \\
 & \Rightarrow & R(\underline{x}, y), S(\underline{y}, x), T(\underline{x}, x), U(\underline{y}, x) & \\
 & \Rightarrow^* & R(\underline{x}, y), S(\underline{y}, x), U(\underline{y}, x) & = h_3^+
\end{array}$$

Call a query q *final* if it is unsafe, and $\forall q'$, if $q \Rightarrow q'$ then q' is safe. Clearly every unsafe query rewrites to a final query: simply apply $\Rightarrow$ repeatedly until all rewritings are to safe queries. We prove in the Appendix:

Lemma 3.8. *h_1, h_2^+, h_3^+ are the only final queries.*

This implies immediately the dichotomy property:

Theorem 3.9. *Let q be a query without self-joins. Then one of the following holds:*

- *q is unsafe and q rewrites to one of h_1, h_2^+, h_3^+. In particular, q is #P-hard.*
- *q is safe. In particular, it is in PTIME.*

How restrictive is the assumption that the query has no self-joins ? It is used both in `Join` and in `Independent project`. We illustrate on $q = R(\underline{x, y}), R(\underline{y, z})$ how, by dropping the assumption, independent projects become incorrect. Although y occurs in all subgoals, we cannot apply an independent project because the two queries $q[a/y] = R(\underline{x, a}), R(\underline{a, z})$ and $q[b/y] = R(\underline{x, b}), R(\underline{b, z})$ are not independent: both $\Phi_{q[a/y]}$ and $\Phi_{q[b/y]}$ depend on the tuple $R(a, b)$ (and also on $R(b, a)$). In fact q is #P-hard [23]. The restriction to queries without self-joins is thus significant. We have recently proved that a dichotomy property holds for arbitrary conjunctive queries, but only over independent probabilistic databases [23], and that result is significantly more complex. The case of independent-disjoint databases is open.

The Complexity of the Complexity We complete our analysis by studying the following problem: given a relational schema $\mathcal{R}$ and conjunctive query q without self-joins over $\mathcal{R}$, decide whether q is safe[5]. We have seen that this problem is in PTIME; here we establish tighter bounds.

In the case of *independent databases*, the key in each relation R consists of all the attributes, $Key(R) = Attr(R)$, hence $sg(x)$ becomes: $sg(x) = \{g \mid x \in Vars(g)\}$.

Definition 3.10. *A conjunctive query is* hierarchical *if for any two variables x, y, either $sg(x) \cap sg(y) = \emptyset$, or $sg(x) \subseteq sg(y)$, or $sg(y) \subseteq sg(x)$.*

As an example, the query[6] $q = R(x), S(x, y)$ is hierarchical because $sg(x) = \{R, S\}$, $sg(y) = \{S\}$, while $h_1 = R(x), S(x, y), T(y)$ is not hierarchical because $sg(x) = \{R, S\}$ and $sg(y) = \{S, T\}$. `SAFE-EVAL` works as follows on independent databases. When the hierarchy $\{sg(x) \mid x \in Vars(q)\}$ has a root variable x, then it applies an independent project on x; when it has multiple connected components, then it applies joins. One can check easily that a query is unsafe iff it contains a sub-pattern:

$$R(x, \ldots), S(x, y, \ldots), T(y, \ldots)$$

Proposition 3.11. *Let SG be a binary relation name. We represent a pair $\mathcal{R}$, q, where $\mathcal{R}$ is a relational schema for an independent database and q a conjunctive query without self-joins, as an instance over SG, as follows[7]. The*

[5]For a fixed $\mathcal{R}$ there are only finitely many queries without self-joins: this is the reason why $\mathcal{R}$ is part of the input.
[6]Since all attributes are keys we don't underline them.
[7]This representation is lossy, because it ignores both the positions where the variables occur in the subgoals in q, and it also ignores all constants in q.

constants are $\mathcal{R} \cup Vars(q)$, and for each subgoal R of q and each variable $x \in Vars(R)$, there is a tuple $SG(R, x)$. Then the property "given $\mathcal{R}$, q, q is unsafe" can be expressed in FO over the vocabulary SG.

In fact, it is expressed by the following conjunctive query with negations, with variables R, S, T, x, y:

$$SG(R, x), \neg SG(R, y), SG(S, x), SG(S, y), SG(T, y), \neg SG(T, x)$$

In the case of *disjoint-independent databases* we will show that checking safety is PTIME complete. Recall the Alternating Graph Accessibility Problem (AGAP): given a directed graph where the nodes are partitioned into two sets called AND-nodes and OR-nodes, decide if all nodes are accessible. An AND-node is accessible if all its parents are; an OR node is accessible if at least one of its parents is. AGAP is PTIME-complete [42]. We prove in the Appendix:

PROPOSITION 3.12. *AGAP is reducible in LOGSPACE to the following problem: given a schema $\mathcal{R}$ and a query q without self-joins, check if q is safe. In particular, the latter is PTIME-hard.*

4. CHALLENGES AND OPEN PROBLEMS

We discuss here some challenges, open problems, and future research directions in probabilistic databases.

Open Problems in Query Evaluation

We enumerate here briefly a few open problems:

Extend the Dichotomy to interpreted predicates ($\neq, <, \leq$), unions, self-joins. Unions and self-joins seem interrelated because of $\mathbf{P}(q_1 \cup q_2) = \mathbf{P}(q_1) + \mathbf{P}(q_2) - \mathbf{P}(q_1 q_2)$. Queries with self-joins have been recently shown to admit a dichotomy [23], but only for independent database; the dichotomy for disjoint-independent databases is open.

Add partial determinism to the database. For example if T is deterministic then $R(\underline{x}), S(\underline{x, y}), T(\underline{y})$ is in PTIME. At a finer granularity, we may know that only certain tuples in a table are deterministic, and would like to exploit that during query processing.

Aggregates: the expected values of some aggregates s.a. `count(*)` can be computed easily; `avg` it is harder see [12, 52]; the complexity of computing the probability of a `HAVING` predicate like `count(*)` > 25 is open.

Improve hardness results It is open whether h_2^+, h_3^+ are hard for a wider ranges of probabilities. Similarly for h_1, when the probabilities in all three tables R, S, T are 1/2.

Optimizations with Safe Subplans

Even when a query is hard, it may have subplans that are safe. In this case the query processor can start by evaluating the safe subplans efficiently, then run a FPTRAS on a much smaller problem. Experiments in [68] report significant speedups with this approach. A general theory of such optimizations techniques remains to be developed. We note here that it does not suffice for the subquery to be safe, its answer must also be a disjoint-independent table. For illustration, consider the query $q = R(\underline{x}, y), S(\underline{y}, z), T(\underline{y, z, u})$: it is hard, because $q \Rightarrow R(\underline{x}, y), S(\underline{y}, z) \Rightarrow^* R(\underline{x}, y), S'(\underline{y})$ which is h_2. The subquery $sq(y) = S(\underline{y}, z), T(\underline{y, z, u})$ is safe, more precisely for every constant a the query $sq[a/y]$ is safe, hence can be evaluated efficiently. Moreover, for any two constants a, b, the events $sq[a/y]$ and $sq[b/y]$ are independent, hence the answers to sq can be stored in a temporary, independent table, $SQ(\underline{y})$. Now the query becomes: $q = R(\underline{x}, y), SQ(\underline{y})$. This is still hard, but smaller than the original one and the FPTRAS runs much more efficiently.

Ranking

In several applications of uncertain data it makes sense to hide completely the probabilities from the user and only use them to rank the query answers. For example, if the real data is deterministic but the system has incomplete knowledge of the data and uses probabilities only to quantify its degree of confidence in the data, then output probabilities are not meaningful to the user, only the ranking is meaningful. This justifies the following:

PROBLEM 1. *The ranking problem is the following. Given two queries q_1, q_2 and a probabilistic database $PDB = (T, \mathbf{P})$, decide whether $\mathbf{P}(q_1) \geq \mathbf{P}(q_2)$.*

The complexity class PP consists of problems of the following form: given an NP machine, check if at least 1/2 of its computations are accepting. The problem: "given a Boolean formula Φ check if at least half of the truth assignments make Φ true" is PP-complete. Ranking is PP-hard (just take $\mathbf{P}(q_2) = 1/2$), and it can be shown to be in PP. Beyond that, nothing is known about the ranking problem. Of particular interest is to study the case where $q_1 = q[a/x]$ and $q_2 = q[b/x]$, where $q(x)$ is a non-boolean query, *i.e.* rank the outputs a, b to a query. What is the class of queries q for which this is tractable ? Another aspect is top-k ranking: if all possible answers to $q(x)$ are $a_1, a_2, \ldots, a_n$, compute the top k answers, see [68].

Probabilistic Inference for Query Evaluation

There exists a rich literature on probabilistic inference in Bayesian Networks [25] and Markov Networks [17]. Boolean expressions are special cases of a Bayesian networks [81]. An important research direction is to adapt these techniques to the query evaluation problem.

We illustrate here with a technique described[8] by Zabiyaka and Darwiche [81]. A *decomposition tree*, dtree, for a DNF formula Φ is a binary tree whose leaves are in one-to-one correspondence with the disjuncts, and whose nodes are labeled with sets of propositional variables s.t. (1) each leaf is labeled with the propositional variables of its disjunct, and (2) for each variable X_i the set of nodes labeled with X_i forms a connected component. The width of the dtree is the maximum number of labels on any node minus 1. Zabiyaka and Darwiche show that, given a dtree of width w, the probability $\mathbf{P}(\Phi)$ can be computed in time $O(|\Phi| 2^w)$. The main idea is this. Each node in the *dtree* represents a formula $\Psi = \Psi_1 \vee \Psi_2$. If the two expressions Ψ_1, Ψ_2 are independent, then $\mathbf{P}(\Psi) = 1 - (1 - \mathbf{P}(\Psi_1))(1 - \mathbf{P}(\Psi_2))$. If they share one common variable X_i, write $\Psi = (X_i \wedge \Psi) \vee (\neg X_i \wedge \Psi)$ and, denoting $\Phi^0 = \Phi[false/X_i]$ and $\Phi^1 = \Phi[true/X_i]$ for any formula Φ, we have $\mathbf{P}(\Psi) = \mathbf{P}(\Psi^0) + \mathbf{P}(\Psi^1)$. Moreover, $\Psi^0 = \Psi_1^0 \vee \Psi_2^0$, and the two expressions are now independent, and similarly for Ψ^1. In general, if Ψ_1, Ψ_2 share w common variables, then we need to expand Ψ as a disjunction of 2^w cases, considering all truth assignments to the w common variables, resulting in a $O(n 2^w)$ time algorithm.

[8]We adapt the original presentation from CNF to DNF

It follows that, for any fixed parameter w, given Φ and a dtree of width $\leq w$ for Φ, computing $\mathbf{P}(\Phi)$ is in PTIME. Note that this does not immediately lead to a practical algorithm because computing a dtree of width $\leq w$ for Φ is hard. It is interesting to note that these tractable classes are orthogonal to those defined by safe queries. For example, consider the safe query $q = R(x), S(y)$. The class of Boolean formulas Φ_q^{PDB} is tractable but does not have a bounded treewidth because $\Phi_q^{PDB} = \bigvee_{i,j} X_i Y_j$, and its treewidth is $\Omega(n)$, when $n = |\bar{X}| = |\bar{Y}|$. Conversely, one can sometimes impose constraints on the database PDB that result in a bounded tree-width: for example consider the constraint *the representation of S is one-to-one*, then the query $h_1 = R(\underline{x}), S(\underline{x,y}), T(\underline{y})$ is in PTIME because Φ_q^{PDB} has tree width $O(1)$.

Expressing Correlations through Views

Graphical models are formalisms that capture local correlations between random variables and derive from here a concise representation of their joint distribution. A treatment of these models is beyond the scope of this paper: the interested reader may find a good primer targeted to a database audience in [28], and a broad discussion and motivation in [65]. Graphical models are also used in some probabilistic databases [39, 72].

There is a very interesting relationship between graphical models and database views that has been little explored. Verma and Pearl [78] show a connection between Embedded Multivalued Dependencies and a certain class of graphical models. We hypothesize that one can express any probabilistic database $PDB = (W, \mathbf{P})$ having complex correlations between attributes and/or tuples as a view v over a disjoint-independent database $PDB' = (T', \mathbf{P}')$, such that the complexity of the view and/or the disjoint-independent database PDB' depends on the structure of the graphical model describing PDB. This is significant, since it allows us to extend a query processor on disjoint-independent databases to arbitrary probabilistic databases: to evaluate q over PDB simply expand the view definition v and evaluate the rewritten query over PDB'.

Problem 2. *Explore the connection between probabilistic databases described as graphical models on one hand, and views over disjoint-independent databases on the other hand.*

Any probabilistic database is a view over a disjoint-independent database. We illustrate this with one relation $R(\underline{A, B, C})$. Any probabilistic database $PDB = (\{W_1, \ldots, W_n\}, \mathbf{P})$ is obtained as the view $R(x, y, z) = S(w), T(w, x, y, z)$ over a disjoint-independent database $PDB' = (T', \mathbf{P}')$ with schema $S(W)$ $(Key(S) = \emptyset)$, $T(\underline{W, A, B, C})$, defined as follows. S has n distinct identifiers $w_1, \ldots, w_n$ (one for each world) $\mathbf{P}'(w_j) = \mathbf{P}(W_j)$; T is deterministic and contains all tuples $T(w_j, a, b, c)$ for $R(a, b, c) \in W_j$.

Constraints

When the data is uncertain, constraints can be used to increase the quality of the data, and hence they are an important tool in managing data with uncertainties. Currently, there is no generally accepted formal definition of constraints over probabilistic databases. We explore here informally the space of alternatives.

In one interpretation a constraint Γ is a sentence that must hold on the possible worlds of a probabilistic database.

	Deterministic database	Probabilistic database
Deterministic constraint	Repairing data [4]	Cleaning probabilistic data $\mathbf{P}(q \mid \Gamma)$
Probabilistic constraint	Probabilistic mappings	Probabilistic data exchange

Figure 5: Applications of constraints

Query evaluation is now a conditional probability $\mathbf{P}(q|\Gamma) = \mathbf{P}(q, \Gamma)/\mathbf{P}(\Gamma)$, thus we need to study the problem of evaluating probabilities of the form $\mathbf{P}(q, \Gamma)$, for various constraint languages. For example, if Γ is a functional dependency, then the problem reduces to conjunctive queries with self-joins and $\neq$: if, say $A \rightarrow B$ in $R(A, B, C)$, then $\Gamma = \neg q_0$, where $q_0 = R(x, y, z), R(x, y', z'), y \neq y'$, hence $\mathbf{P}(q, \Gamma) = \mathbf{P}(q) - \mathbf{P}(q, q_0)$.

A second interpretation is that of *soft constraints* [51, 73], for example: "each employee has a unique name with probability 0.85"; or "most people buy what their friends buy", which correspond to statements like $\mathbf{P}(\Gamma) = p$, or $\mathbf{P}(\Gamma) > p$. A formal semantics, proposed in knowledge representation [8, 47], is to consider all probability spaces $\mathbf{P}$ that satisfy the constraints, choose that with maximum entropy, and let the domain grow to infinity; query evaluation becomes undecidable. We added the requirement that the expected cardinality of each relation remains fixed [22]; query evaluation becomes decidable for conjunctive Γ's.

A particularly interesting application of soft constraints is to probabilistic mappings, generalizing mappings in peer data exchange [57]. Here the constraint is between a source schema and a target schema. For example if the source and target are `S(name, phone)` and `T(name, homePhone, officePhone)` then Γ may map `phone` to `homePhone` with probability 0.3 and to `officePhone` with probability 0.7.

Fig. 5, reproduced from [69], summarizes our discussion.

A Logic for Information Leakage

Alice has a database instance I and she wants to send Bob a view v. However, she wants to ensure that a certain query q remains secret, *i.e.* Bob will not learn the secret $q(I)$ from the view $v(I)$. The question is how do we determine if a view v leaks information about a query q ?

Following [62] we assume that the adversary knows a probability distribution $\mathbf{P}$ for the instance I, and, hence, can compute the *a prior* probability of the secret, $\mathbf{P}(q)$. Once Alice publishes v, then Bob can revise this probability, and compute the *a posteriori* probability $\mathbf{P}(q \mid v)$. To prevent leakage we want $\mathbf{P}(q \mid v) \approx \mathbf{P}(q)$. Our goal is to be able to decide security (is q secure w.r.t. v ?), and also to make inferences, like, for example, testing for collusions: given that q is secure w.r.t. v_1 and that q is secure w.r.t. v_2, is q secure w.r.t. to the pair of views v_1, v_2 ? Formally: $\mathbf{P}(q \mid v_1, v_2) \approx \mathbf{P}(q)$.

The difficulty with this approach is choosing the adversary's prior distribution $\mathbf{P}$. Miklau and Suciu [62] consider a powerful adversary by allowing $\mathbf{P}$ to be *any* independent distribution: a query q is *perfectly secure* w.r.t. v if $\mathbf{P}(q \mid v) = \mathbf{P}(q)$ for all independent distributions $\mathbf{P}$. But this definition is too restrictive: it classifies as un-secure many query-view pairs that are considered secure in practice. It also fails to model collusions, because $\mathbf{P}(q|v_1) = \mathbf{P}(q|v_2) = \mathbf{P}(q)$ implies $\mathbf{P}(q \mid v_1, v_2) = \mathbf{P}(q)$: collusions are known to exist in practice, hence this model is too restrictive. Miklau, Dalvi, and Suciu [19] relaxed this definition and assumed that the probability of each tuple is small, while the domain of tuples is large, such that the expected size of the table is some constant. This gives rise to a particular distribution, $\mathbf{P}$, which corresponds to a certain random graph. A query q is *practically secure* w.r.t. v if $\lim_{n\to\infty} \mathbf{P}(q|v) = \lim_{n\to\infty} \mathbf{P}(q)$, where n is the size of the domain. In other words, this definition requires that $\mathbf{P}(q|v) \approx \mathbf{P}(q)$ holds over a large domain, but allows it to fail over small domains. This models query-view security closer to common practice, and also captures collusion [24], *i.e.* there are queries q, v_1, v_2 for which $\lim \mathbf{P}(q|v_1) = \lim \mathbf{P}(q|v_2) = 0$, yet $\lim \mathbf{P}(q|v_1, v_2) = 1$.

In probability logic [2] the *material conditioning*, $v \rightarrow q \equiv \neg v \vee q$ is replaced with *probabilistic conditioning* $v \Rightarrow q \equiv (\mathbf{P}(q|v) \approx 1)$, which leads to a form of non-monotonic reasoning [65]. We propose as research problem the design of a non-monotonic logic based on probabilistic conditioning for reasoning about information leakage, ideally by making minimal assumptions on the adversary's prior. We note that the probabilistic conditioning in the random graph is already non-monotone [24]: we can have $\lim \mathbf{P}(q \mid v) = 1$ (significant disclosure), yet $\lim \mathbf{P}(q \mid v, v') = 0$ (practical security), *i.e.* based on the evidence v the adversary has high confidence that q is true, but after also seeing the evidence v' the adversary finds a different explanation for v and his confidence in q drops.

5. CONCLUSIONS

The cost of enforcing a precise semantics is increasing with the scale of the data and with its degree of heterogeneity. By making uncertainties first class citizens we can reduce that cost, or (more likely) enable applications that were otherwise prohibitive. We have described here a model in which uncertainties are represented as probabilities, and the system computes and output probability for each answer to a query. We studied some formal aspects of the query evaluation problem, and discussed number of open problems in probabilistic data management.

Acknowledgments We thank Amol Deshpande, Luna Dong, Lise Getoor, Alon Halevy, Christoph Koch, Phokion Kolaitis, Reneé Miller, Christopher Ré, Sunita Sarawagi, Val Tannen, and Victor Vianu for their comments and advice.

6. REFERENCES

[1] S. Abiteboul and P. Senellart. Querying and updating probabilistic information in XML. In *EDBT*, pages 1059–1068, 2006.

[2] Ernest Adams. *A Primer of Probability Logic*. CSLI Publications, Stanford, California, 1998.

[3] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *VLDB*, pages 586–597, 2002.

[4] P. Andritsos, A. Fuxman, and R. J. Miller. Clean answers over dirty databases. In *ICDE*, 2006.

[5] L. Antova, C. Koch, and D. Olteanu. 10^(10^6) worlds and beyond: Efficient representation and processing of incomplete information. In *ICDE*, 2007.

[6] L. Antova, C. Koch, and D. Olteanu. World-set decompositions: Expressiveness and efficient algorithms. In *ICDT*, pages 194–208, 2007.

[7] A. Arasu, V. Ganti, and R. Kaushik. Efficient exact set-similarity joins. In *VLDB*, pages 918–929, 2006.

[8] F. Bacchus, A. Grove, J. Halpern, and D. Koller. From statistical knowledge bases to degrees of belief. *Artificial Intelligence*, 87(1-2):75–143, 1996.

[9] D. Barbara, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *IEEE Trans. Knowl. Data Eng.*, 4(5):487–502, 1992.

[10] O. Benjelloun, A. Das Sarma, A. Halevy, and J. Widom. ULDBs: Databases with uncertainty and lineage. In *VLDB*, pages 953–964, 2006.

[11] G. Borriello and F. Zhao. World-Wide Sensor Web: 2006 UW-MSR Summer Institute Semiahmoo Resort, Blaine, WA, 2006. www.cs.washington.edu/mssi/2006/schedule.html.

[12] D. Burdick, P. Deshpande, T. S. Jayram, R. Ramakrishnan, and S. Vaithyanathan. Efficient allocation algorithms for olap over imprecise data. In *VLDB*, pages 391–402, 2006.

[13] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *ACM SIGMOD*, San Diego, CA, 2003.

[14] T. Choudhury, M. Philipose, D. Wyatt, and J. Lester. Towards activity databases: Using sensors and statistical models to summarize people's lives. *IEEE Data Eng. Bull*, 29(1):49–58, March 2006.

[15] W. Cohen, P. Ravikumar, and S. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, pages 73–78, 2003.

[16] G. Cooper. Computational complexity of probabilistic inference using bayesian belief networks (research note). *Artificial Intelligence*, 42:393–405, 1990.

[17] R. Cowell, P. Dawid, S. Lauritzen, and D. Spiegelhalter, editors. *Probabilistic Networks and Expert Systems*. Springer, 1999.

[18] P. Dagum and M. Luby. Approximating probabilistic inference in bayesian belief networks is NP-hard. *Artificial Intelligence*, 60:141–153, 1993.

[19] N. Dalvi, G. Miklau, and D. Suciu. Asymptotic conditional probabilities for conjunctive queries. In *ICDT*, 2005.

[20] N. Dalvi, Chris Re, and D. Suciu. Query evaluation on probabilistic databases. *IEEE Data Engineering Bulletin*, 29(1):25–31, 2006.

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

[22] N. Dalvi and D. Suciu. Answering queries from statistics and probabilistic views. In *VLDB*, 2005.

[23] N. Dalvi and D. Suciu. The dichotomy of conjunctive queries on random structures. In *PODS*, 2007.

[24] Nilesh Dalvi. Query evaluation on a database given by a random graph. In *ICDT*, pages 149–163, 2007.

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

[26] A. Das Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working models for uncertain data. In *ICDE*, 2006.

[27] Michel de Rougemont. The reliability of queries. In *PODS*, pages 286–291, 1995.

[28] A. Deshpande, M. Garofalakis, and R. Rastogi. Independence is good: Dependency-based histogram synopses for high-dimensional data. In *SIGMOD*, pages 199–210, 2001.

[29] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, pages 588–599, 2004.

[30] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Using probabilistic models for data management in acquisitional environments. In *CIDR*, pages 317–328, 2005.

[31] A. Doan, R. Ramakrishnan, F. Chen, P. DeRose, Y. Lee, R. McCann, M. Sayyadian, and W. Shen. Community information management. *IEEE Data Engineering Bulletin, Special Issue on Probabilistic Data Management*, 29(1):64–72, March 2006.

[32] M. Balazinska et al. Data management in the world-wide sensor web. *IEEE Pervasive Computing*, 2007. To appear.

[33] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A.M. Popescu, T. Shaked, S. Soderland, D. Weld, and A. Yates. Web-scale information extraction in KnowItAll: (preliminary results). In *WWW*, pages 100–110, 2004.

[34] Ivan Felligi and Alan Sunter. A theory for record linkage. *Journal of the American Statistical Society*, 64:1183–1210, 1969.

[35] M. Franklin, A. Halevy, and D. Maier. From databases to dataspaces: a new abstraction for information management. *SIGMOD Record*, 34(4):27–33, 2005.

[36] Norbert Fuhr and Thomas Roelleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.*, 15(1):32–66, 1997.

[37] Helena Galhardas, Daniela Florescu, Dennis Shasha, Eric Simon, and Cristian-Augustin Saita. Declarative data cleaning: Language, model, and algorithms. In *VLDB*, pages 371–380, 2001.

[38] Minos Garofalakis and Dan Suciu. Special issue on probabilistic data management. *IEEE Data Engineering Bulletin*, pages 1–72, 2006.

[39] Lise Getoor. An introduction to probabilistic graphical models for relational data. *IEEE Data Engineering Bulletin, Special Issue on Probabilistic Data Management*, 29(1):32–40, March 2006.

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

[41] T. Green and V. Tannen. Models for incomplete and probabilistic information. *IEEE Data Engineering Bulletin*, 29(1):17–24, March 2006.

[42] R. Greenlaw, J. Hoover, and W. Ruzzo. *Limits to Parallel Computation. P-Completeness Theory*. Oxford University Press, New York, Oxford, 1995.

[43] L. Gu, R. Baxter, D. Vickers, and C. Rainsford. Record linkage: Current practice and future directions. In *CMIS Technical Report No. 03/83*, 2003.

[44] R. Gupta and S. Sarawagi. Creating probabilistic databases from information extraction models. In *VLDB*, pages 965–976, 2006.

[45] A. Halevy, M. Franklin, and D. Maier. Principles of dataspace systems. In *PODS*, pages 1–9, 2006.

[46] A. Halevy, A. Rajaraman, and J. Ordille. Data integration: The teenage years. In *VLDB*, pages 9–16, 2006.

[47] J. Halpern. From statistical knowledge bases to degrees of belief: an overview. In *PODS*, pages 110–113, 2006.

[48] D. Heckerman. Tutorial on graphical models, June 2002.

[49] M. Hernandez and S. Stolfo. The merge/purge problem for large databases. In *SIGMOD*, pages 127–138, 1995.

[50] E. Hung, L. Getoor, and V.S. Subrahmanian. PXML: A probabilistic semistructured data model and algebra. In *ICDE*, 2003.

[51] I.F. Ilyas, V. Markl, P.J. Haas, P. Brown, and A. Aboulnaga. Cords: Automatic discovery of correlations and soft functional dependencies. In *SIGMOD*, pages 647–658, 2004.

[52] T.S. Jayram, S. Kale, and E. Vee. Efficient aggregation algorithms for probabilistic data. In *SODA*, 2007.

[53] T.S. Jayram, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu. Avatar information extraction system. *IEEE Data Engineering Bulletin*, 29(1):40–48, 2006.

[54] S. Jeffery, M. Garofalakis, and M. Franklin. Adaptive cleaning for RFID data streams. In *VLDB*, pages 163–174, 2006.

[55] R. Karp and M. Luby. Monte-Carlo algorithms for enumeration and reliability problems. In *Proceedings of the annual ACM symposium on Theory of computing*, 1983.

[56] N. Khoussainova, M. Balazinska, and D. Suciu. Towards correcting input data errors probabilistically using integrity constraints. In *MobiDB*, pages 43–50, 2006.

[57] P. Kolaitis. Schema mappings, data exchange, and metadata management. In *PODS*, pages 61–75, 2005.

[58] D. Koller. Representation, reasoning, learning. Computers and Thought 2001 Award talk.

[59] L. Lakshmanan, N. Leone, R. Ross, and V.S. Subrahmanian. Probview: A flexible probabilistic database system. *ACM Trans. Database Syst.*, 22(3), 1997.

[60] J. Lester, T. Choudhury, N. Kern, G. Borriello, and B. Hannaford. A hybrid discriminative/generative approach for modeling human activities. In *IJCAI*, pages 766–772, 2005.

[61] J. Madhavan, S. Cohen, X. Dong, A. Halevy, S. Jeffery, D. Ko, and C. Yu. Web-scale data integration: You can afford to pay as you go. In *CIDR*, pages 342–350, 2007.

[62] G. Miklau and D. Suciu. A formal analysis of information disclosure in data exchange. In *SIGMOD*, 2004.

[63] Radford Neal. Probabilistic inference using Markov Chain Monte Carlo methods. Technical Report CRG-TR-93-1, Univ. of Toronto, 1993.

[64] Christos Papadimitriou. *Computational Complexity*. Addison Wesley Publishing Company, 1994.

[65] Judea Pearl. *Probabilistic reasoning in intelligent systems*. Morgan Kaufmann, 1988.

[66] S. Philippi and J. Kohler. Addressing the problems with life-science databases for traditional uses and systems biology. *Nature Reviews Genetics*, 7:481–488, June 2006.

[67] J. S. Provan and M. O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput.*, 12(4):777–788, 1983.

[68] C. Re, N. Dalvi, and D. Suciu. Efficient Top-k query evaluation on probabilistic data. In *ICDE*, 2007.

[69] Christopher Ré. Applications of probabilistic constraints. Technical Reprot TR2007-03-03, University of Washington, Seattle, Washington, March 2007.

[70] R. Ross, V.S. Subrahmanian, and J. Grant. Aggregate operators in probabilistic databases. *JACM*, 52(1), 2005.

[71] Sunita Sarawagi. Automation in information extraction and data integration. Tutorial presented at VLDB'2002.

[72] Prithviraj Sen and Amol Deshpande. Representing and querying correlated tuples in probabilistic databases. In *ICDE*, 2007.

[73] W. Shen, X. Li, and A. Doan. Constraint-based entity matching. In *AAAI*, pages 862–867, 2005.

[74] D. Suciu and N. Dalvi. Tutorial: Foundations to probabilistic answers to queries. In *SIGMOD*, 2005. Available from `www.cs.washington.edu/homes/suciu`.

[75] L. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8:410–421, 1979.

[76] M. van Keulen, A. de Keijzer, and W. Alink. A probabilistic XML approach to data integration. In *ICDE*, pages 459–470, 2005.

[77] M. Y. Vardi. The complexity of relational query languages. In *Proceedings of 14th ACM SIGACT Symposium on the Theory of Computing*, pages 137–146, San Francisco, California, 1982.

[78] T. Verma and J. Pearl. Causal networks: Semantics and expressiveness. *Uncertainty in Artificial Intelligence*, 4:69–76, 1990.

[79] L. von Ahn and L. Dabbish. Labeling images with a computer game. In *CHI*, pages 319–326, 2004.

[80] William Winkler. The state of record linkage and current research problems. Technical report, Statistical Research Division, U.S. Bureau of the Census, 1999.

[81] Y. Zabiyaka and A. Darwiche. Functional treewidth: Bounding complexity in the presence of functional dependencies. In *SAT*, pages 116–129, 2006.

[82] `alonhalevy.blogspot.com/2007_01_01_archive.html`.

[83] `www.flickr.com`.

[84] `base.google.com`.

[85] `http://www.mturk.com/mturk/welcome`.

APPENDIX

A finite probability space is $(\Omega, \mathbf{P})$ where Ω is a finite *set of outcomes*, and $\mathbf{P} : \Omega \to [0,1]$ s.t. $\sum_{\omega\in\Omega} \mathbf{P}(\omega) = 1$. An *event* is $E \subseteq \Omega$, it's *marginal probability* is $\mathbf{P}(E) = \sum_{\omega\in E} \mathbf{P}(\omega)$. If $\mathbf{P}(E) > 0$, the *conditional probability* of E' is $\mathbf{P}(E'|E) = \mathbf{P}(E'E)/\mathbf{P}(E)$ and the *conditional probability space* is $(E, \mathbf{P}')$, $\mathbf{P}'(\omega) = \mathbf{P}(\omega|E)$.

A set $\{E_1, \ldots, E_m\}$ is *independent* if $\mathbf{P}(E_1, \ldots, E_m) = \mathbf{P}(E_1)\cdots\mathbf{P}(E_m)$. E_1, E_2 are *disjoint* if $\mathbf{P}(E_1, E_2) = 0$.

The *product space* [41] of n probability spaces $(\Omega_i, \mathbf{P}_i)$, $i = 1, n$ is $(\Omega, \mathbf{P})$ where $\Omega = \prod_i \Omega_i$ and $\mathbf{P}(\omega_1, \ldots, \omega_n) = \prod_i \mathbf{P}(\omega_i)$. Let $(\Omega, \mathbf{P})$ be a finite probability space, Ω' a set, and $f : \Omega \to \Omega'$ a function. The *image space* [41] is $(\Omega', \mathbf{P}')$ where $\mathbf{P}'(\omega') = \sum_{\omega\in\Omega: f(\omega)=\omega'} \mathbf{P}(\omega)$.

Lemma A.1. *(1) Let $\mathbf{P}$ be a disjoint probability space on the Boolean variables $Y_1, \ldots, Y_n$, $\mathbf{P}(Y_1) + \ldots + \mathbf{P}(Y_n) \leq 1$. Then $\mathbf{P}$ is the image of an independent space over $X_1, \ldots, X_n$:*

$$\begin{aligned} Y_i &= (\bigwedge_{j<i} \neg X_j) \wedge X_i, i = 1, n \\ \mathbf{P}(X_i) &= \frac{\mathbf{P}(Y_i)}{1 - \sum_{j<i} \mathbf{P}(Y_j)}, i = 1, n \end{aligned}$$

(2) Let $\mathbf{P}$ be an independent space on the Boolean variables $Y_1, \ldots, Y_n$, s.t. $\forall i = 1, n$, $\mathbf{P}(Y_i)$ is a rational number p_i/q_i and p_i, q_i can be represented using at most k bits. Then $\mathbf{P}$ is isomorphic to a conditional space of a uniform space over kn variables $X_1, \ldots, X_{kn}$ (i.e. $\mathbf{P}(X_i) = 1/2$).

Proof. (1) direct calculation. (2) use $(X_{ij})_j$ to encode the bits of the number(s) p_i, q_i, define $Y_i =$ "$(X_{ij})_j < p_i$" and condition on the event $\bigwedge_i$ "$(X_{ij})_j < q_i$", see [40]. □

The lemma implies the following Corollary by direct calculations, and this proves Theorems 3.1 and 3.2:

Corollary A.2. *There exists a poly-time computable function $F(k, n, m, n_1, \ldots, n_m, p_1, q_1, \ldots, p_n, q_n)$ (where $\sum_j n_j = n$, $\forall i.p_i, q_i < 2^k$), s.t.: for every disjoint-independent probability space $\mathbf{P}$ over $Y_1, \ldots, Y_n$ defined by a partition $\bar{Y}_j$, $j = 1, m$ ($n_j = |\bar{Y}_j|$) and $\mathbf{P}(Y_i) = p_i/q_i$ ($p_i, q_i < 2^k$) there exists a formula Ψ over kn variables, computable in poly-time s.t, $\mathbf{P}(\Phi) = \#\Psi/F$.*

A *perfect matching* in a bipartite graph $E \subseteq \bar{X} \times \bar{Y}$ where $|\bar{X}| = |\bar{Y}| = n$ is a subset $E_0 \subseteq E$ of size n s.t. each node occurs exactly once in E_0. The number of perfect matching is equal to permanent of the 0-1 adjacency matrix for G, hence this problem is denoted PERMANENT. Valiant showed that it is #P-hard [75].

Proof. (of Theorem 3.5) The proof of h_1 is given in [21] by reduction from partitioned, positive 2DNF [67]. For h_2, h_3 we use reductions from PERMANENT. We will show how to count the number of perfect matches in $E \subseteq \bar{X} \times \bar{Y}$ using an oracle for computing $\mathbf{P}(h_i)$, for each $i = 2, 3$. Both reductions share the same construction for R: the possible tuples are E, and $\mathbf{P}(R(\underline{x}, y)) = 1/\mathtt{degree}(x)$ (R is a p-or-set). Call an $\bar{X}$-match a subset of E that contains each node in $\bar{X}$ exactly once: the possible instances of R are precisely the $\bar{X}$-matches, and every possible $\bar{X}$-match has the same probability $P = 1/\prod_{x\in\bar{X}} \mathtt{degree}(x)$. Consider now the query h_2: let the possible tuples in S be $\bar{Y}$, and $\mathbf{P}(S(y)) = 1 - p$, for a fixed $p \in (0, 1)$. Let E_i denote the event that the $\bar{X}$-match given by R uses exactly i nodes in $\bar{Y}$, and M_i be the number of such $\bar{X}$-matches: thus, $\mathbf{P}(E_i) = M_i P$. M_n is the number of perfect matches, and $\sum_{i=0,n} M_i = \prod_{x\in\bar{X}} \mathtt{degree}(x)$: denote the latter with M. We have $\mathbf{P}(h_2|E_i) = 1 - p^i$, because once we fix i target nodes in $\bar{Y}$, this is the probability that we include at least one of them in S. Thus, $\mathbf{P}(h_2) = F(p) = \sum_i \mathbf{P}(h_2|E_i)\mathbf{P}(E_i) = \sum_i M_i P(1-p^i) = MP - \sum_i p^i M_i$. Use the oracle for computing $F(p)$ on n distinct values $p_1, \ldots, p_n$: this leads to a linear system of equations with unknowns M_i's whose matrix is Vandermonde, allowing us to compute all M_i's, and, in particular, M_n.

For h_3 we use a different reduction: the possible tuples in S are all pairs $(x, y) \in \bar{X} \times \bar{Y}$, and $\mathbf{P}(S(x, \underline{y})) = p$ for some $p \in (0, 1/n)$. The *type* of an $\bar{X}$-match is $\bar{k} = (k_1, \ldots, k_n)$ where k_i = number of edges incident to y_i. Let $E_{\bar{k}}$ be the event that the R-instance has type $\bar{k}$, and let $M_{\bar{k}}$ be the number of $\bar{X}$-matches of type $\bar{k}$; thus $\mathbf{P}(E_{\bar{k}}) = PM_{\bar{k}}$ and the number of perfect matches is $M_{\bar{1}}$, where $\bar{1} = (1, 1, \ldots, 1)$. Now we have $\mathbf{P}(\neg h_3|E_{\bar{k}}) = f_{\bar{k}}(p)$, where $f_{\bar{k}}(p) = \prod_i (1 - k_i p)$. This is because in S we must choose at most one edge for each y_i, and we cannot choose any of the k_i incoming edges, hence we are left with the choice of the remaining edges or none at all, for a total probability of $1 - k_i p$. For the type $\bar{1}$, $f_{\bar{1}} = (1-p)^n$ is a polynomial of degree n in p, while for every other type $f_{\bar{k}}$ is a polynomial of degree $< n$, because any type other than $\bar{1}$ has at least one $k_i = 0$. We have $\mathbf{P}(\neg h_3) = F(p) = \sum_{\bar{k}} PM_{\bar{k}} f_{\bar{k}}(p)$. This is a polynomial in p of degree n and we want to compute the leading coefficient, a_n, *i.e.* the coefficient of p^n, because $M_{\bar{1}} = (-1)^n/a_n$. Fix a small $h > 0$ and define $\Delta g(p) = g(p+h) - g(p)$ for a polynomial g: if $degree(g) = k$, then $degree(\Delta g) = k - 1$, and the leading coefficient is multiplied by kh. Letting $\Delta^{(1)} = \Delta$, $\Delta^{(i+1)} = \Delta(\Delta^{(i)})$, we have $\Delta^{(n)} f(p) = n!\, h^n a_n$ (the degree is 0), and it can be computed from the values of f on $n+1$ equidistant points p.

Finally we show that h_2^+, h_3^+ are hard by induction. Let q be a query that we know is hard, and q' be obtained by adding one more copy of R' or S': $R'(\underline{x}, y)$ or $S'(x, \underline{y})$. We cannot reduce q to q' by making R' deterministic because it violates the key constraints. Instead we exploit the special structure of the hardness for q (which holds on the base cases h_2, h_3 then follows inductively). Case 1: the probabilities of another copy $\mathbf{P}(R(\underline{x}, y))$ can either be made arbitrarily low (like for $S(x, \underline{y})$ in h_3). Define $\mathbf{P}(R'(\underline{x}, y)) = 1/n$ (*i.e.* R' becomes a p-or-set): this has the effect of multiplying $\mathbf{P}(R(\underline{x}, y))$ with $1/n$, and the query is still hard because we can make these probabilities small. Case 2: R is a p-or-set, *i.e.* its worlds are X-matches. $\mathbf{P}(R'(\underline{x}, y)) = p/n$ for a small p, and denote q_i the event *the number of edges $R(a, b)$ that satisfy the query q is exactly i*, hence $\mathbf{P}(q) = \sum_{i=1,n} \mathbf{P}(q_i)$. We have $\mathbf{P}(\neg q'|q_i) = (1 - p/n)^i$ because R' must avoid all of those i edges to fail the query q'. Hence $\mathbf{P}(\neg q') = \sum_{i=1,n} (1 - p/n)^i \mathbf{P}(q_i)$. This allows us to compute all the $\mathbf{P}(q_i)$'s using a Vandermonde matrix. $\square$

PROOF. (of Lemma 3.7) The reductions here are very simple, and we only sketch them. We have an oracle for computing $\mathbf{P}'(q)$ for any $PDB' = (T', \mathbf{P}')$ and need to compute $\mathbf{P}(q')$ on a given $PDB = (T, \mathbf{P})$, where $q \Rightarrow q'$. We derive PDB' from PB s.t. $\mathbf{P}(q') = \mathbf{P}'(q)$. When $q' = q[a/x]$, then remove all possible tuples from PDB that have a value other than a on an x-position; similarly for $q' = q[y/x]$. When q' is a connected component of q, *i.e.* $q = q', q''$, then make all relations in q'' deterministic ($\mathbf{P}' = 1$), and fill them with a canonical database for q'' (hence q'' is true in any possible world). When q' is obtained by removing a subgoal g then make that table deterministic (this is possible because $KVars(g) = Vars(g)$). Finally, if q' is obtained by replacing a subgoal g with g' and $g \rightarrow g'$, then compute the g-table in PDB' from the g'-table in PDB in the way dictated by the relationship between g and g': copy values across attributes, and/or set new attributes to constants. Details omitted. $\square$

PROOF. (of Lemma 3.8) Let q be final. None of the rules of `Safe-Eval` (join, independent project, disjoint project) applies to q, otherwise q would be safe (since it is final). We can also assume wlog that q has at least two variables, otherwise one can show it is safe. Let $Vars(q) = \{x_1, \ldots, x_n\}$, and define the sets of subgoals $S_i = sg(x_i)$, $T_i = Sg(q) - sg(x_i)$, $i = 1, n$. We have $S_i \neq Sg(q)$: otherwise an independent project applies to x_i. Whenever two variables x_i, x_j occur together in a subgoal then $S_i \cup S_j = Sg(q)$: this is because q rewrites to $q[x_j/x_i]$, the latter is safe, and the independent project on x_i is the only rule that may apply without applying to q, *i.e.* $S_i \cup S_j = Sg(q)$. We now show:

$$\forall i \neq j, S_i \cup S_j \quad = \quad Sg(q) \qquad (2)$$

Suppose x_i, x_j do not co-occur in any subgoal. There exists x_k that co-occurs with x_i (otherwise x_i forms a connected component and we can apply a join), hence $S_i \cup S_k = Sg(q)$, hence x_k co-occurs with x_j, hence $S_j \cup S_k = Sg(q)$, implying $S_k = Sq(q)$, contradiction. Eq.(2) holds, and in particular $\forall i, S_i \neq \emptyset$ (otherwise $S_j = Sg(q)$ for $j \neq i$). We also have $\forall i \neq j, T_i \cap T_j = \emptyset$ (equivalently $T_i \subseteq S_j$). For every subgoal g, all variables occur on its key positions, except perhaps one (namely that x_i for which $g \in T_i$). We now prove that there are at most two variables x_1, x_2. Suppose there is a third, and note that $q' = q[a/x_3]$ is safe. Since each subgoal in q has at least two variables in key positions we cannot apply a disjoint project in q'; moreover the subgoals in q' are still connected since $Sg(q') = S_1 \cup S_2$ and $T_3 \subseteq S_1 \cap S_2 \neq \emptyset$ hence a join doesn't apply either, contradiction. It follows that there are exactly two variables x_1, x_2. Case 1: there exists $g \in S_1 \cap S_2$, hence $g = R(\underline{x_1, x_2})$ (a subgoal like $R(\underline{x_1, x_2}, x_1)$ rewrites to $R(\underline{x_1, x_2})$). Rewrite q by removing the subgoal g: this query is safe, hence it must be disconnected, hence q is $S(\underline{x_1}), R(\underline{x_1, x_2}), T(\underline{x_2})$ which is h_1. Case 2: $S_1 \cap S_2 = \emptyset$. If there exists a subgoal $g \in S_1$ s.t. $Vars(g) = \{x_1\}$, *i.e.* $g = R(\underline{x_1})$, then $S_1 = \{g\}$ because by removing g we must obtain a safe query; similarly for $g' \in S_2$ s.t. $Vars(g') = \{x_2\}$; denote $g' = S(\underline{x_2})$. q cannot be g, g' since that is safe, so there are three cases: g does not exists, g' exists, then $q = R_1(\underline{x}, y), \ldots, R_k(\underline{x}, y), S(\underline{y}) = h_2^+$, or g exists and g' does not exists (here q is also equivalent to a h_2^+), or neither g nor g' exists, hence $q = R_1(\underline{x}, y), \ldots, R_k(\underline{x}, y), S_1(x, \underline{y}), \ldots, S_m(x, \underline{y})$, which is h_3^+. $\square$

PROOF. (of Proposition 3.12) Let G be an AND/OR graph, we construct the following relational schema $\mathcal{R}$ and query q. There will be one variable in q for every node in G, plus one extra variable c; the invariant is: a node is accessible in the graph iff the corresponding variable can be made constant by `SAFE-EVAL`. For every AND node x with parents $y, z, \ldots$ we introduce a new relational symbol R and a subgoal $R(\underline{y, z, \ldots}, x)$ in q (a disjoint project applies to x iff all its parents are constants). For every OR node x with parents $y, z, \ldots$ we introduce a new subgoal for each parent $S_y(\underline{y}, z), \ldots$ (a disjoint project applies to z iff one of its parents is constant). Finally, for each node $x, y, z, \ldots$ we create a new subgoal $T_x(\underline{c}, x)$, $T_y(\underline{c}, y)$: these prevent an independent-project until all variables became constants. $\square$