On Query Algebras for Probabilistic Databases*

Christoph Koch

Department of Computer Science
Cornell University, Ithaca, NY
koch@cs.cornell.edu

Abstract

This article proposes a core query algebra for probabilistic databases. In essence, this core is part of the query languages of most probabilistic database systems proposed so far, but is sometimes hidden in complex language definitions. We give a formal definition of the algebra and illustrate it by examples. We then survey the current state of knowledge regarding the expressive power and complexity of this core.

1 Introduction

The emerging research area of probabilistic databases has attracted much interest and excitement recently. It is still quite early in the development of this field, and the community has not yet converged upon a standard set of features or use cases that probabilistic databases and their query languages should support. However, a body of foundational knowledge on query languages for probabilistic databases is forming rapidly. The aim of this article is to give a concise summary of this foundation.

We will focus on extracting and discussing a core query algebra that arguably can be found completely or mostly implemented in most probabilistic database systems developed so far, including MystiQ [9], Trio [21], MayBMS [3, 15], and MCDB [12]. This algebra is *probabilistic world-set algebra* [5, 14, 15, 16].

Agreeing on relational algebra as a core language for relational database systems was one of the foundations of their success: It has facilitated the development of a widely agreed-upon terminology which allowed the database research community to make rapid progress; but it is also the interface between query optimization and query evaluation at the very heart database systems. Part of the rationale for proposing a core algebra for probabilistic databases is, of course, the hope that it will help us replicate our previous success with relational databases in the field of probabilistic databases.

It seems proper to start the search for a such a core with the definition of *design desiderata* for probabilistic database query languages. Ours are the following:

1. Efficient query evaluation.

2. The right degree of expressive power. The language should be powerful enough to support important queries. On the other hand, it should not be too strong, because expressiveness generally comes at a price: high evaluation complexity and infeasibility of query optimization. Can a case be made that some language is in a natural way a probabilistic databases analog of the relationally complete languages (such as relational algebra) – an expressiveness yardstick?

3. Genericity. The semantics of a query language should be independent from details of how the data is represented. Queries should behave in the same way no matter how the probabilistic data is stored. This is a basic requirement that is even part of the traditional definition of what constitutes a query (cf. e.g. [1]), but it is nontrivial to achieve for probabilistic databases [5, 4]. Genericity is key to making the language applicable to many different database systems that internally represent data in different ways.

4. The ability to transform data. Queries on probabilistic databases are often interpreted quite narrowly in the literature. It is the author's view that queries in general should be compositional mappings between databases, in this case probabilistic databases. This is a property taken for granted in relational databases. It allows for the definition of clean database update languages.

5. The ability to introduce uncertainty. This may appear to be a controversial goal, since uncertainty is commonly considered undesirable, and probabilistic databases are there to deal with it by providing useful functionality *despite* uncertainty. An uncertainty-introduction operation is

*Database Principles Column. Column editor: Leonid Libkin, School of Informatics, University of Edinburgh, Edinburgh, EH8 9AB, UK. E-mail: libkin@inf.ed.ac.uk

important for compositionality, to allow the construction of an uncertain database from scratch (as part of the update language), and to support hypothetical (what-if) queries.

Probabilistic world-set algebra is a minimal extension of relational algebra that arguably satisfies all of the desiderata presented above.

Content of this article. After a short definition of the conceptual model of probabilistic databases used throughout most of the article (discrete at first) in Section 2, the algebra is formally defined in Section 3. Section 4 illustrates probabilistic world-set algebra and its SQL-like syntax by several examples. Section 5 and 6 discuss our current state of knowledge regarding the expressive power and complexity, respectively, of the algebra. Section 7 discusses extensions of the algebra (such as aggregates) and queries on probabilistic databases with continuous distributions.

2 Probabilistic Databases

Informally, our model of probabilistic databases is the following. The schema of a probabilistic database is simply a relational database schema. Given such a schema, a probabilistic database is a finite set of database instances of that schema (called possible worlds), where each world has a weight (called probability) between 0 and 1 and the weights of all worlds sum up to 1. In a subjectivist Bayesian interpretation, one of the possible worlds is "true", but we do not know which one, and the probabilities represent degrees of belief in the various possible worlds. Note that this is only the conceptual model. The physical representation of the set of possible worlds in probabilistic database management systems is quite different [9, 21, 3].

Given a schema with relation names $R_1, \ldots, R_k$. We use $sch(R_l)$ to denote the attributes of relation schema R_l. Formally, a *probabilistic database* is a *finite* set of structures

$$\mathbf{W} = \{\langle R_1^1, \ldots, R_k^1, p^{[1]}\rangle, \ldots, \langle R_1^n, \ldots, R_k^n, p^{[n]}\rangle\}$$

of relations $R_1^i, \ldots, R_k^i$ and numbers $0 < p^{[i]} \leq 1$ such that

$$\sum_{1 \leq i \leq n} p^{[i]} = 1.$$

We call an element $\langle R_1^i, \ldots, R_k^i, p^{[i]}\rangle \in \mathbf{W}$ a *possible world*, and $p^{[i]}$ its probability. We use superscripts for indexing possible worlds. To avoid confusion with exponentiation, we sometimes use bracketed superscripts $.^{[i]}$. We call a relation R *complete* or *certain* if its instantiations are the same in all possible worlds of $\mathbf{W}$, i.e., if $R^1 = \cdots = R^n$.

The definitions of the following sections are applicable if either a set- or a multiset-based semantics for relations is used. Whenever the distinction causes an subtleties, they will be made clear. Of course, when we use SQL-like syntax for queries, we automatically assume multiset semantics.

Tuple *confidence* refers to the probability of the event $\vec{t} \in R$, where R is one of the relation names of the schema, with

$$\Pr[\vec{t} \in R] = \sum_{1 \leq i \leq n:\ \vec{t} \in R^i} p^{[i]}.$$

3 Core Algebra

This section defines *probabilistic world-set algebra* (probabilistic WSA) [5, 14, 16]. Informally, probabilistic world-set algebra consists of the operations of relational algebra, an operation for computing tuple confidence conf, and the repair-key operation for *introducing* uncertainty.

- The operations of relational algebra are evaluated individually, in "parallel", in each possible world.
- The operation $\text{conf}(R)$ computes, for each tuple that occurs in relation R in at least one world, the sum of the probabilities of the worlds in which the tuple occurs. The result is a certain relation, or viewed differently, a relation that is the same in all possible worlds.
- Finally, $\text{repair-key}_{\vec{A}@P}(R)$, where $\vec{A}, P$ are attributes of R, conceptually nondeterministically chooses a maximal repair of key $\vec{A}$. This operation turns a possible world R^i into the set of worlds consisting of all possible *maximal repairs* of key $\vec{A}$. A repair of key $\vec{A}$ in relation R^i is a subset of R^i for which $\vec{A}$ is a key. It uses the numerically-valued column P for weighting the newly created alternative repairs.

We define the semantics of probabilistic world-set algebra formally using a function $[\![\cdot]\!]_{pw}$ that maps between sets of possible worlds. For a further illustration that the reader may find more intuitive, we will also provide a Monte Carlo/sampling semantics definition $[\![\cdot]\!]_{mc}$. Conceptually, $[\![Q]\!]_{mc}$ computes for query Q a sample result relation. Note that the definition of $[\![\cdot]\!]_{mc}$ is nondeterministic, an multiple invocations will yield different results. The sampling semantics also only approximates $[\![\cdot]\!]_{pw}$; we will not have space to make this precise, but will point to relevant literature.

Relational algebra. The operations of relational algebra (selection σ, projection π, product $\times$, union $\cup$, difference $-$, and attribute renaming ρ) are applied in each possible world independently.

The possible-worlds semantics of unary and binary relational algebra operations Θ on probabilistic database $\mathbf{W}$ is

$$[\![\Theta(R_l[, R_m])]\!]_{pw}(\mathbf{W}) := \{\langle R_1, \ldots, R_k, \Theta(R_l[, R_m]), p\rangle \mid \langle R_1, \ldots, R_k, p\rangle \in \mathbf{W}\}.$$

The sampling semantics $[\![\Theta(Q_1[, Q_2])]\!]_{mc}$ is simply $\Theta([\![Q_1]\!]_{mc}[, [\![Q_2]\!]_{mc}])$.

Selection conditions are Boolean combinations of atomic conditions (i.e., negation is permitted even in the positive fragment of the algebra). Arithmetic expressions may occur in atomic conditions and in the arguments of π and ρ. For example, $\rho_{A+B\to C}(R)$ in each world adds up the A and B values of each tuple of R and keeps them in a new C attribute.

Confidence. The semantics of the tuple confidence operation is

$$[\![\mathrm{conf}(R_l)]\!]_{pw}(\mathbf{W}) := \{\langle R_1, \ldots, R_k, S, p\rangle \mid \langle R_1, \ldots, R_k, p\rangle \in \mathbf{W}\}$$

where

$$S = \left\{\langle \vec{t}, \Pr[\vec{t} \in R_l]\rangle \mid \vec{t} \in \bigcup_{i=1}^{n} R_l^i\right\}.$$

The result of $\mathrm{conf}(R_l)$, the relation S, is the same in all possible worlds, i.e., it is a certain relation.

By our definition of probabilistic databases, each possible world has nonzero probability. As a consequence, conf does not return tuples with probability 0. Note also that conf implicitly eliminates duplicates.

Example 3.1 On probabilistic database

R^1	A	B
	a	b
	b	c

$p^{[1]} = .3$

R^2	A	B
	a	b
	c	d

$p^{[2]} = .7$

$\mathrm{conf}(R)$ computes

conf(R)	A	B	P
	a	b	1
	b	c	.3
	c	d	.7

i.e., for each possible tuple, the sum of the weights of the possible worlds in which it occurs. □

Let $S_1, \ldots, S_m$ be samples from $[\![Q]\!]_{mc}$, that is, the results of m separate invocations of $[\![Q]\!]_{mc}$. Then we compute $[\![\mathrm{conf}(Q)]\!]_{pw}$ as

$$\left\{\langle \vec{t}, |\{i : \vec{t} \in S_i\}|/m\rangle : \vec{t} \in \bigcup_{i=1}^{m} S_i\right\}.$$

Repair-key. The uncertainty-introducing operation *repair-key* can be thought of as sampling a maximum repair of a key for a relation. Repairing a key of a complete relation R means to compute, as possible worlds, all subset-maximal relations obtainable from R by removing tuples such that a key constraint is satisfied. We will use this as a method for constructing probabilistic databases, with probabilities derived from relative weights attached to the tuples of R.

We say that relation R' is a *maximal repair* of a functional dependency (fd, cf. [1]) for relation R if R' is a maximal subset of R which satisfies that functional dependency, i.e., a subset $R' \subseteq R$ that satisfies the fd such that there is no relation R'' with $R' \subset R'' \subseteq R$ that satisfies the fd.

Let $\vec{A}, B \in sch(R_l)$. For each possible world $\langle R_1, \ldots, R_k, p\rangle \in \mathbf{W}$, let column B of R contain only numerical values greater than 0 and let R_l satisfy the fd $(sch(R_l) - B) \to sch(R_l)$. Then,

$$[\![\text{repair-key}_{\vec{A}@B}(R_l)]\!]_{pw}(\mathbf{W}) := \Big\{\langle R_1, \ldots, R_k, \pi_{sch(R_l)-B}(\hat{R}_l), \hat{p}\rangle \mid \langle R_1, \ldots, R_k, p\rangle \in \mathbf{W},$$
$$\hat{R}_l \text{ is a maximal repair of fd } \vec{A} \to sch(R_l),$$
$$\hat{p} = p \cdot \prod_{\vec{t}\in\hat{R}_l} \frac{\vec{t}.B}{\sum_{\vec{s}\in R_l : \vec{s}.\vec{A} = \vec{t}.\vec{A}} \vec{s}.B}\Big\}$$

Such a repair operation, apart from its usefulness for the purpose implicit in its name, is a powerful way of constructing probabilistic databases from complete relations.

The sampling semantics makes this operation more intuitive: Conceptually, given a sample R from $[\![Q]\!]_{mc}$, we group the tuples of R by the columns $\vec{A}$: for each distinct $\vec{a}$ in $\pi_{\vec{A}}(R)$, we independently sample exactly one tuple $\vec{t}$ from group $G_{\vec{a}} = \sigma_{\vec{A}=\vec{a}}(R)$ with the probability distribution given by (normalized) column B,

$$\Pr[\text{choose } \vec{t} \text{ from group } \vec{a}] = \vec{t}.B / \sum_{\vec{t}' \in G_{\vec{a}}} \vec{t}'.B.$$

Example 3.2 Consider the example of tossing a biased coin twice. We start with a certain database

R	Toss	Face	FProb
	1	H	.4
	1	T	.6
	2	H	.4
	2	T	.6

$p = 1$

that represents the possible outcomes of tossing the coin twice. We turn this into a probabilistic database that represents this information using alternative possible worlds for the four outcomes using the query $S :=$ repair-key$_{\mathrm{Toss@FProb}}(R)$. The resulting possible worlds are

S^1	Toss	Face
	1	H
	2	H

S^2	Toss	Face
	1	H
	2	T

S^3	Toss	Face
	1	T
	2	H

S^4	Toss	Face
	1	T
	2	T

with probabilities $p^{[1]} = p \cdot \frac{.4}{.4+.6} \cdot \frac{.4}{.4+.6} = .16$, $p^{[2]} = p^{[3]} = .24$, and $p^{[4]} = .36$. □

Coins	Type	Count
	fair	2
	2headed	1

Faces	Type	Face	FProb
	fair	H	.5
	fair	T	.5
	2headed	H	1

Tosses	Toss
	1
	2

R^f	Type
	fair

R^{dh}	Type
	2headed

$S^{f.HH}$	Type	Toss	Face
	fair	1	H
	fair	2	H

$p^{f.HH} = 1/6$

$S^{f.HT}$	Type	Toss	Face
	fair	1	H
	fair	2	T

$p^{f.HT} = 1/6$

$S^{f.TH}$	Type	Toss	Face
	fair	1	T
	fair	2	H

$p^{f.TH} = 1/6$

$S^{f.TT}$	Type	Toss	Face
	fair	1	T
	fair	2	T

$p^{f.TT} = 1/6$

S^{dh}	Type	Toss	Face
	2headed	1	H
	2headed	2	H

$p^{dh} = 1/3$

Ev	Toss	Face
	1	H
	2	H

Q	Type	P
	fair	$(1/6)/(1/2) = 1/3$
	2headed	$(1/3)/(1/2) = 2/3$

Figure 1: Tables of Example 4.1.

The fragment of probabilistic WSA which excludes the difference operation is called *positive* probabilistic WSA.

Computing possible and certain tuples is redundant with conf:

$$\begin{aligned} \text{poss}(R) &:= \pi_{sch(R)}(\text{conf}(R)) \\ \text{cert}(R) &:= \pi_{sch(R)}(\sigma_{P=1}(\text{conf}(R))) \end{aligned}$$

4 Examples

4.1 Adding Evidence

Example 4.1 A bag of coins contains two fair coins and one double-headed coin. We take one coin out of the bag but do not look at its two faces to determine its type (fair or double-headed) for certain. Instead we toss the coin twice to collect evidence about its type.

We start with a complete database (i.e., a relational database, or a probabilistic database with one possible world of probability 1) consisting of three relations, Coins, Faces, and Tosses (see Figure 1 for all tables used in this example). We first pick a coin from the bag and model that the coin be either fair or double-headed. In probabilistic WSA this is expressed as

$$R := \text{repair-key}_{\emptyset@\text{Count}}(\text{Coins}).$$

This results in a probabilistic database of two possible worlds, $\langle \text{Coins}, \text{Faces}, \text{Tosses}, R^f, p^f = 2/3\rangle$ and $\langle \text{Coins}, \text{Faces}, \text{Tosses}, R^{dh}, p^{dh} = 1/3\rangle$.

The possible outcomes of tossing the coin twice can be modeled as

$$S := \text{repair-key}_{\text{Toss@FProb}}(R \bowtie \text{Faces} \times \text{Tosses}).$$

This turns the two possible worlds into five, since there are four possible outcomes of tossing the fair coin twice, and only one for the double-headed coin.

Let $T := \pi_{\text{Toss,Face}}(S)$. The posterior probability that a coin of type x was picked, given the *evidence* Ev (see Figure 1) that both tosses result in H, is

$$\Pr[x \in R \mid T = Ev] = \frac{\Pr[x \in R \wedge T = Ev]}{\Pr[T = Ev]}.$$

Let A be a relational algebra expression for the Boolean query $T = Ev$. Then we can compute a table of pairs $\langle x, \Pr[x \in R \mid T = Ev]\rangle$ as

$$\begin{aligned} Q := \pi_{\text{Type},P_1/P_2 \to P}(\\ \rho_{P\to P_1}(\text{conf}(R \times A)) \times \rho_{P\to P_2}(\text{conf}(A))). \end{aligned}$$

The prior probability that the chosen coin was fair was $2/3$; after taking the evidence from two coin tosses into account, the posterior probability Pr[the coin is fair | both tosses result in H] is only $1/3$. Given the evidence from the coin tosses, the coin is now more likely to be double-headed. □

4.2 Hypothetical Queries: Skills Management

For the second example, we use an SQL-like syntax for probabilistic WSA. The mapping is in strict analogy with that from relational algebra to SQL. Repair-key is a new operation whose syntax should be intuitive. Confidence computation (or strictly speaking, the combination of confidence computation and projection, $\text{conf}(\pi_{\vec{A}}(R))$) has become an aggregate, which conveys the intuition that duplicates are eliminated: for each group, only one tuple with a probability is returned.

Example 4.2 Given a relational database representing companies, employees, and their skills such as the following.

CE	CID	EID
	LEH	Bob
	LEH	Joe
	MER	Dan
	MER	Bill
	MER	Fred

ES	EID	Skill
	Bob	subprime mortgage
	Joe	subprime mortgage
	Dan	junk bonds
	Dan	subprime mortgage
	Bill	risk management
	Fred	junk bonds

We now want to ask the following hypothetical query: Suppose I buy one of the companies and exactly one employee leaves. Which skills do I gain for

certain? Note that this query starts on a traditional relational database (without uncertainty) and returns a certain table. We will create a probabilistic database for intermediate results.

We first choose one company to by and one employee who will leave and compute the employees that will remain in my company.

```
create table RemainingEmployees as
select CE.cid, CE.eid
from   CE,
       (repair key (dummy)
        in (select 1 as dummy, * from CE)) Choice
where CE.cid = Choice.cid
and   CE.eid <> Choice.eid;
```

No probabilities are available, so we will make our choice uniformly. Since we only ask for certain answers in this example, the probabilities actually do not matter.

Next we compute a table of probabilities, for companies and skills, (p1) that I gain the the skill and buy the company, (p2) that I buy the company, and (p1/p2) the conditional probability that I gain the skill if I buy the company.

```
create table Skills as
select Q1.cid, Q1.skill, p1, p2, p1/p2 as p
from (select R.cid, ES.skill, conf() as p1
      from RemainingEmployees R, ES
      where R.eid = ES.eid
      group by R.cid, ES.skill) Q1,
     (select cid, conf() as p2
      from RemainingEmployees
      group by cid) Q2
where Q1.cid = Q2.cid;
```

For the database given above, this results in the table

Skills	CID	Skill	p1	p2	p
	LEH	subprime mortgage	2/5	2/5	1
	MER	junk bonds	3/5	3/5	1
	MER	subprime mortgage	2/5	3/5	2/3
	MER	risk management	2/5	3/5	2/3

The query

```
select cid, skill from Skills where p=1;
```

yields the desired answer. □

5 Expressiveness

The repair-key operation admits an interesting class of queries: Like in Example 4.1, we can start with a probabilistic database of prior probabilities, add further evidence (in Example 4.1, the result of the coin tosses) and then compute interesting posterior probabilities. The adding of further evidence may require extending the hypothesis space first. For this, the repair-key operation is essential. Even though our goal is not to update the database, we have to be able to introduce uncertainty just to be able to model new evidence – say, experimental data. Many natural and important probabilistic database queries cannot be expressed without the repair-key operation. The coin tossing example was admittedly a toy example (though hopefully easy to understand). Real applications such as diagnosis or processing scientific data involve technically similar questions.

Regarding our desiderata, it is quite straightforward to see that probabilistic WSA is generic (3): see also [5]. It is clearly a data transformation query language (4) that supports powerful queries for defining databases. The repair-key operation is our construct for uncertainty introduction (5). The evaluation efficiency (1) of probabilistic WSA is studied in Section 6. The expressiveness desideratum (2) is discussed next.

An expressiveness yardstick. In [5] a non-probabilistic version of WSA is introduced. It replaces the confidence operation with an operation $\text{poss}_{\vec{A}}(Q)$, where $\vec{A}$ is a set of column names of Q, for computing possible tuples. Compared to the poss operation described above, the operation of [5] is more powerful. The operation partitions the set of possible worlds into the groups of those worlds that agree on $\pi_{\vec{A}}(Q)$. The result in each world is the set of tuples possible in Q within the world's group. Thus, this operation supports the grouping of possible worlds just like the group-by construct in SQL supports the grouping of tuples.

The main focus of [5] is to study the fragment of (non-probabilistic) WSA in which repair-key is replaced by the choice-of operation, definable as $\text{choice-of}_{\vec{A}@P}(R) := R \bowtie \text{repair-key}_{\emptyset@P}(\pi_{\vec{A},P}(R))$. The choice-of operation introduces uncertainty like the repair-key operation, but can only cause a polynomial, rather than exponential, increase of the number of possible worlds. This restricted language still allows to express interesting queries; for instance, Example 4.2 is expressible despite the absence of probabilities and the conf operation. This language has the property that query evaluation on enumerative representations of possible worlds is in PTIME (see Section 6 for more on this). Moreover, it is *conservative* over relational algebra in the sense that any query that starts with a certain database (a classical relational database) and produces a certain database is equivalent to a relational algebra query and can be efficiently rewritten into relational algebra. This is a nontrivial result, because in this language we can produce uncertain intermediate results consisting of many possible worlds using the choice-of operator. This allows us to express and efficiently answer hypothetical (what-if) queries.

(Full non-probabilistic) WSA consists of the relational algebra operations, repair-key, and $\text{poss}_{\vec{A}}$. In [16], it is shown that WSA precisely captures second-

order logic. Leaving aside inessential details about interpreting second-order logic over uncertain databases – it can be done in a clean way – this result shows that a query is expressible in WSA if and only if it is expressible in second-order logic. WSA seems to be the first algebraic (i.e., variable and quantifier-free) language known to have exactly the same expressive power as second-order logic.

It can be argued that this establishes WSA as the natural analog of relational algebra for uncertain databases. Indeed, while it is well known that useful queries (such as transitive closure or counting queries, cf. [1]) cannot be expressed in it, relational algebra is a very popular expressiveness yardstick for relational query languages (and query languages that are as expressive as relational algebra are called *relationally complete*). Relational algebra is also exactly as expressive as the *relational calculus* [1]. Second-order logic is just first-order logic extended by (existential) quantification over relations ("Does there exist a relation R such that ϕ holds?", where ϕ is a formula). This is the essence of (what-if) reasoning over uncertain data. For example, the query of Example 4.1 employed what-if reasoning over relations twice via the repair-key operation, first considering alternative choices of coin and then alternative outcomes to coin tossing experiments.

6 Complexity

The core of the algebra, positive relational algebra, can be efficiently evaluated on c-tables. In [3], a version of c-tables called U-relations was developed on which all expressions of this algebra fragment can be evaluated purely in relational algebra.

Properties of the relational-algebra reduction. The relational algebra rewriting down to positive relational algebra on U-relations has a number of nice properties. First, since relational algebra has PTIME (even AC_0) data complexity, the query language of positive relational algebra, repair-key, and poss on probabilistic databases represented by U-relations has the same. The rewriting is in fact a *parsimonious translation*: The number of algebra operations does not increase and each of the operations selection, projection, join, and union remains of the same kind. Query plans are hardly more complicated than the input queries. As a consequence, we were able to observe that off-the-shelf relational database query optimizers do well in practice [3].

Thus, for all but two operations of probabilistic world-set algebra, there is a very efficient solution that builds on relational database technology. The remaining operations are confidence computation and relational algebra difference.

Approximate confidence computation. To compute the confidence in a tuple of data values occurring possibly in several tuples of a U-relation, we have to compute the probability of the disjunction of the local conditions of all these tuples. We have to eliminate duplicate tuples because we are interested in the probability of the data tuples rather than some abstract notion of tuple identity that is really an artifact of our representation. That is, we have to compute the probability of a DNF, i.e., the sum of the weights of the worlds identified with valuations θ of the random variables such that the DNF becomes true under θ. This problem is #P-complete [11, 9]. The result is not the sum of the probabilities of the individual conjunctive local conditions, because they may, intuitively, "overlap".

Confidence computation can be efficiently approximated by Monte Carlo simulation [11, 9, 14]. The technique is based on the Karp-Luby fully polynomial-time randomized approximation scheme (FPRAS) for counting the number of solutions to a DNF formula [13, 8]. There is an efficiently computable unbiased estimator that in expectation returns the probability p of a DNF of n clauses (i.e., the local condition tuples of a Boolean U-relation) such that computing the average of a polynomial number of such Monte Carlo steps (= calls to the Karp-Luby unbiased estimator) is an (ϵ, δ)-approximation for the probability: If the average $\hat{p}$ is taken over at least $\lceil 3 \cdot n \cdot \log(2/\delta)/\epsilon^2 \rceil$ Monte Carlo steps, then $\Pr\big[|p - \hat{p}| \geq \epsilon \cdot p\big] \leq \delta$. The paper [8] improves upon this by determining smaller numbers (within a constant factor from optimal) of necessary iterations to achieve an (ϵ, δ)-approximation.

Avoiding the difference operation. Difference $R - S$ is conceptually simple on c-tables. Without loss of generality, assume that S does not contain tuples $\langle \vec{a}, \psi_1 \rangle, \ldots, \langle \vec{a}, \psi_n \rangle$ that are duplicates if the local conditions are disregarded. (Otherwise, we replace them by $\langle \vec{a}, \psi_1 \vee \cdots \vee \psi_n \rangle$.) For each tuple $\langle \vec{a}, \phi \rangle$ of R, if $\langle \vec{a}, \psi \rangle$ is in S then output $\langle \vec{a}, \phi \wedge \neg\psi \rangle$; otherwise, output $\langle \vec{a}, \phi \rangle$. Testing whether a tuple is possible in the result of a query involving difference is already NP-hard [2]. For U-relations, we in addition have to turn $\phi \wedge \neg\psi$ into a DNF to represent the result as a U-relation. This may lead to an exponentially large output.

In many practical applications, the difference operation can be avoided. Difference is only hard on uncertain relations. On such relations, it can only lead to displayable query results in queries that close the possible worlds semantics using conf, computing a single certain relation. Probably the most important application of the difference operation is for encoding universal constraints, for example in data cleaning. But if the confidence operation is applied on top of a universal query, there is a trick that will often allow to rewrite the query into an existential one (which can be expressed in positive relational algebra plus conf, without difference) [14].

Suppose we compute a conditional probability $\Pr[\phi \mid \psi] = \Pr[\phi \wedge \psi]/\Pr[\psi]$. Here ϕ is existential (expressible in positive relational algebra) and ψ is an equality-generating dependency (i.e., a special universal query) [1]. The trick is to turn relational difference into the subtraction of probabilities, $\Pr[\phi \wedge \psi] =$

Language Fragment	Complexity
On non-succinct representations:	
RA + conf + possible + choice-of	in PTIME (SQL) [14]
RA + possible + repair-key	NP-&coNP-hard [5], in P^{NP} [16]
RA + possible$_Q$ + repair-key	PHIER-compl. [16]
On U-relations:	
Pos.RA + repair-key + possible	in AC0 [3]
RA + possible	co-NP-hard [2]
Conjunctive queries + conf	#P-hard [9]
Probabilistic WSA	in $P^{\#P}$ [14]
Pos.RA + repair-key + possible + approx.conf + egds	in PTIME [14]

Figure 2: Complexity results for (probabilistic) world-set algebra. RA denotes relational algebra.

$\Pr[\phi] - \Pr[\phi \wedge \neg\psi]$ and $\Pr[\psi] = 1 - \Pr[\neg\psi]$, where $\neg\psi$ is existential (with inequalities). Thus $\neg\psi$ and $\phi \wedge \neg\psi$ are expressible in positive relational algebra. This works for a considerable superset of the equality-generating dependencies [14], which in turn subsume useful data cleaning constraints, such as *conditional functional dependencies* [7].

Complexity Overview. Figure 2 gives an overview over the known complexity results for the various fragments of probabilistic WSA. Two different representations are considered, non-succinct representations that basically consist of enumerations of the possible worlds [5] and succinct representations: U-relational databases. In the non-succinct case, only the repair-key operation, which may cause an exponential explosion in the number of possible worlds, makes queries hard. All other operations, including confidence computation, are easy. In fact, we may add much of SQL – for instance, aggregations – to the language and it still can be processed efficiently, even by a reduction of the query to an SQL query on a suitable non-succinct relational representation.

When U-relations are used as representation system, the succinctness causes both difference [2] and confidence computation [9] independently to make queries NP-hard. Full probabilistic world-set algebra is essentially not harder than the language of [9], even though it is substantially more expressive.

It is worth noting that repair-key by itself, despite the blowup of possible worlds, does not make queries hard. For the language consisting of positive relational algebra, repair-key, and poss, we have shown by construction that it has PTIME complexity: We have given a positive relational algebra rewriting to queries on the representations earlier in this section. Thus queries are even in the highly parallelizable complexity class AC_0.

The final result in Figure 2 concerns the language consisting of the positive relational algebra operations, repair-key, (ϵ, δ)-approximation of confidence computation, and the generalized equality generating dependencies of [14] for which we can rewrite difference of uncertain relations to difference of confidence values. The result is that queries of that language fragment are in PTIME overall. In [14], a stronger result than just the claim that each of the operations of such a query is individually in PTIME is proven. It is shown that, leaving aside a few pitfalls, global approximation guarantees can be achieved in polynomial time, i.e., results of entire queries in this language can be approximated arbitrarily closely in polynomial time.

This is a non-obvious result because the query language is compositional and selections can be made based on approximated confidence values. In a query $\sigma_{P=0.5}(\text{approx.conf}(R))$, an approximated P value will almost always be slightly off, even if the exact P value is indeed 0.5, and the selection of tuples made based on whether P is 0.5 is nearly completely arbitrary. In [14, 10], it is shown that this is essentially an unsurmountable problem. All we can tell is that if P is very different from 0.5, then the probability that the tuple should be in the answer is very small. If atomic selection conditions on (approximated) probabilities usually admit ranges such as $P < 0.5$ or $0.4 < P < 0.6$, then query approximation will nevertheless be meaningful: we are able to approximate query results unless probability values are very close or equal to the constants used as interval bounds. (These special points are called *singularities* in [14].)

The results of [14] have been obtained for powerful conditions that may use arithmetics over several approximated attributes, which is important if conditional probabilities have to be checked in selection conditions or if several probabilities have to be compared. The algorithm that gives overall (ϵ, δ)-approximation guarantees in polynomial time is not strikingly practical. Further progress on this has been made in [10], but more work is needed.

7 Further Remarks

The continuous case. In general, in the continuous case, we have to rely on Monte Carlo simulation for evaluating queries (cf. e.g. [20, 12]). In fact, our sampling semantics $[\![\cdot]\!]_{mc}$ immediately applies in the continuous case. Apart from that, we need other ways of introducing uncertainty and defining probabilistic databases which are more powerful than the repair-key operation. Observe that the sampling semantics of repair-key suggests that repair-key$_{\vec{A}@B}(R)$ on a relation of schema $R(\vec{A}, B, \vec{C})$ could be thought of as an aggregate operation

select $\vec{A}$, choose($\vec{C}$; B) from R group by $\vec{A}$;

that nondeterministically chooses one tuple from each group with probability given by the weights B. (But note that the result of the choose aggregate is a $\vec{C}$-tuple, rather than a single value.) This can be generalized to uncertainty introduction aggregate functions

that return *relations* of dependent uncertain values (random variables) that are unnested into the result relation. These are essentially the variable generation (VG) functions of MCDB [12]. Some functions that do not need relation-typed input but only need a tuple of parameters can be implemented to resemble simple (rather than aggregate SQL functions), e.g. the function normal$(\cdot,\cdot)$

select mu, sigma, normal(mu, sigma) from R;

which extends R by a column of independent normally distributed values (random variables) whose parameters are given by R.

Language extensions. The focus of this brief article was on a query algebra in the spirit of relational algebra, but with the extensions needed to manage probabilistic databases. The probably most important language feature not covered by probabilistic WSA are aggregates. Aggregates have been studied by several researchers [19, 12]. The most relevant fact is probably that aggregates can be dealt with quite well by a Monte-Carlo approach, even in the continuous setting [12]. But we are referring to the computation of expectations and moments of aggregates here, closing the possible worlds semantics. Compositionally defining aggregates on representations of probabilistic databases (as is done for relational algebra in e.g. [3]) leads to exponential blowup in the size of any representations that have been studied so far.

Apart from that, Trio [21] has extended its probabilistic database query language by support for processing data provenance [6]. Top-k queries [18] are expressible in probabilistic WSA, but there are no special language constructs to hook efficient implementations of top-k queries to. The paper [17] introduces a language construct for conditioning a probabilistic database, i.e., to essentially remove possible worlds that do not satisfy a given set of constraints. Updates have been studied in [5, 15]: Given a compositional base language such as probabilistic WSA, defining update operations is quite clean and straightforward. APIs and programming language access to probabilistic databases are studied in [4].

Acknowledgments

The author thanks his collaborators on some of the work discussed in this paper, Lyublena Antova, Michaela Götz, and Dan Olteanu, and the NSF for support under grant IIS-0812272.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] S. Abiteboul, P. Kanellakis, and G. Grahne. "On the Representation and Querying of Sets of Possible Worlds". *Theor. Comput. Sci.*, **78**(1):158–187, 1991.

[3] L. Antova, T. Jansen, C. Koch, and D. Olteanu. "Fast and Simple Relational Processing of Uncertain Data". In *Proc. ICDE*, 2008.

[4] L. Antova and C. Koch. "On APIs for Probabilistic Databases". In *Proc. 2nd International Workshop on Management of Uncertain Data*, Auckland, New Zealand, 2008.

[5] L. Antova, C. Koch, and D. Olteanu. "From Complete to Incomplete Information and Back". In *Proc. SIGMOD*, 2007.

[6] O. Benjelloun, A. D. Sarma, A. Halevy, and J. Widom. "ULDBs: Databases with Uncertainty and Lineage". In *Proc. VLDB*, 2006.

[7] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. "Conditional Functional Dependencies for Data Cleaning". In *Proc. ICDE*, 2007.

[8] P. Dagum, R. M. Karp, M. Luby, and S. M. Ross. "An Optimal Algorithm for Monte Carlo Estimation". *SIAM J. Comput.*, **29**(5):1484–1496, 2000.

[9] N. Dalvi and D. Suciu. "Efficient query evaluation on probabilistic databases". *VLDB Journal*, **16**(4):523–544, 2007.

[10] M. Goetz and C. Koch. "A Compositional Framework for Complex Queries over Uncertain Data". In *Proc. ICDT*, 2009. to appear.

[11] E. Grädel, Y. Gurevich, and C. Hirsch. "The Complexity of Query Reliability". In *Proc. PODS*, pages 227–234, 1998.

[12] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. M. Jermaine, and P. J. Haas. "MCDB: A Monte Carlo approach to managing uncertain data". In *Proc. ACM SIGMOD Conference*, pages 687–700, 2008.

[13] R. M. Karp, M. Luby, and N. Madras. "Monte-Carlo Approximation Algorithms for Enumeration Problems". *J. Algorithms*, **10**(3):429–448, 1989.

[14] C. Koch. "Approximating Predicates and Expressive Queries on Probabilistic Databases". In *Proc. PODS*, 2008.

[15] C. Koch. "MayBMS: A system for managing large uncertain and probabilistic databases". In C. Aggarwal, editor, *Managing and Mining Uncertain Data*, chapter 6. Springer-Verlag, 2008. To appear.

[16] C. Koch. "A Compositional Query Algebra for Second-Order Logic and Uncertain Databases". In *Proc. ICDT*, 2009. to appear.

[17] C. Koch and D. Olteanu. "Conditioning Probabilistic Databases". In *Proc. VLDB*, 2008.

[18] C. Re, N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *Proc. ICDE*, pages 886–895, 2007.

[19] C. Ré and D. Suciu. "Efficient Evaluation of HAVING Queries on a Probabilistic Database". In *Proc. DBPL*, pages 186–200, 2007.

[20] S. Singh, C. Mayfield, S. Mittal, S. Prabhakar, S. E. Hambrusch, and R. Shah. "Orion 2.0: native support for uncertain data". In *Proc. ACM SIGMOD Conference*, pages 1239–1242, 2008.

[21] J. Widom. "Trio: a system for data, uncertainty, and lineage". In C. Aggarwal, editor, *Managing and Mining Uncertain Data*. Springer-Verlag, 2008. To appear.