Uncertainty Annotated Databases - A Lightweight Approach for Dealing with Uncertainty

Su Feng, Aaron Huber, Boris Glavic, Oliver Kennedy


2018-02

http://www.cs.iit.edu/~dbgroup/
ABSTRACT
Incomplete and probabilistic data models have been proposed to deal with the uncertainty inherent in many real world data collection and management tasks. However, query evaluation over such models is heavy-weight - both in terms of computational complexity as well as usability. We introduce UA-databases (UA-DBs), a light-weight model of uncertainty where tuples from a single possible world are annotated with uncertainty information. UA-databases can be derived from commonly used incomplete and probabilistic data models. We present a query semantics for UA-DBs that is compatible with deterministic query processing, as well as many data models expressible as K-relations. Furthermore, we guarantee that tuples that are marked as certain in a UA-DB query result are guaranteed to be certain answers. We implement UA-DBs on top of a DBMS and experimentally demonstrate that this approach is efficient.

PVLDB Reference Format:

1. INTRODUCTION
Data uncertainty arises naturally in applications like sensing, crowd-sourcing, data exchange, distributed computing, constraint enforcement, entity resolution, and many others. Because uncertain data is hard to reason about, a variety of coping strategies have evolved to shield users from uncertainty in such applications. One approach is to compute only “certain answers” to queries: answers that are unambiguously correct, in spite of data uncertainty. However, this strategy is computationally intensive and may exclude useful results. As a consequence, it is far more common in practice to first curate uncertain source data and then to forget about the fact that the data was ever uncertain. For instance, most constraint-based data cleaning methods apply this approach. Although

Figure 1: UA-DBs relative to both Deterministic and Incomplete/Probabilistic query processing schemes.

more efficient for queries, this strategy too has its flaws. For example, people are regularly denied financial resources due to erroneous entity resolution results being treated as fact. A third alternative is to employ an expressive probabilistic or incomplete data model which precisely encodes data uncertainty. However, the complexity of query evaluation for such data models is prohibitively high and they produce sets of possible results that can be overwhelmingly large.

Example 1. Geocoders translate natural language descriptions of locations into coordinates (i.e., latitude and longitude). Consider the following example locations (drawn from [24]) in the Buffalo area, and some possible geocodings:

<table>
<thead>
<tr>
<th>id</th>
<th>address</th>
<th>geocoded</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>51 Comstock</td>
<td>(42.9380, -78.8192)</td>
</tr>
<tr>
<td>2</td>
<td>Grant at Ferguson</td>
<td>(32.2507, -110.8773)</td>
</tr>
<tr>
<td>3</td>
<td>499 Woodlawn</td>
<td>(42.9136, -78.8463)</td>
</tr>
<tr>
<td>4</td>
<td>192 Davidson</td>
<td>(42.9376, -78.8064)</td>
</tr>
</tbody>
</table>

Figure 2 illustrates the output of a spatial join that determines the neighborhood of each set of coordinates. Using standard probabilistic query semantics (Figure 2a), queries over uncertain data produce the set of all possible outputs. Each tuple is annotated with its confidence: the marginal probability that the tuple appears in the result. Conversely, the set of certain answers (Figure 2b) is the set of tuples that are guaranteed to appear in the result. In the former case, the result is large, as all possible outcomes are listed. Conversely, in the latter case, some (likely important) result records do not appear at all.
We propose a compromise between these alternatives: a family of relational data models and query evaluation semantics that we collectively refer to as Uncertainty Annotated Databases (UA-DBs). The UA-DB approach is illustrated in Figure 1. UA-DBs layer a lightweight annotation scheme over curation-based approaches to uncertainty. As we prove in this paper, these annotations provide a conservative approximation of certain answer semantics (i.e., c-soundness [23]) for RA+. UA-DBs have the performance and utility of curation-style uncertainty management, while still providing much of the reliability of certain answers.

Example 2. Continuing the prior example, Figure 2c illustrates the result of the same query as a UA-DB. The result is the set of tuples corresponding to the most likely geocodings for each location. Tuples labeled as certain (e.g., locations 1 and 4) are guaranteed to be part of the certain answer to the query. Tuples labeled as uncertain may be incorrect (e.g., location 2). Since UA-DBs are a conservative approximation of certain answers, it is possible that some tuples labeled as uncertain will actually be part of the certain answers to the query (e.g. location 3). However, discounting labels, the query result is identical to a query over a curated dataset. Hence, even if a certain answer is labeled as uncertain, it is still present in the result.

Crucially, UA-DBs can be defined for a wide range of data models. Being based on Green et. al.’s semiring annotation framework [19], UA-DBs generalize the classical set-oriented notion of incomplete data and certain answers to all data models expressible as relations annotated with elements from an l-semiring — a semiring that has a well-behaved greatest lower bound operation. Importantly, this includes not just incomplete versions of set and bag semantics, but also many extensions of the relational model such as relations annotated with provenance or trust information. UA-DBs are also backwards compatible with existing models for incomplete data. As a proof-of-concept, we define mappings from TI-(P)DBs [43] and (P)C-tables [21, 25] (and, thus, also V-tables and Codd-tables [25]) to UA-DBs.

1.1 Outline

The remainder of this paper is structured as follows.

Incomplete K-Relations. (Section 3) Before discussing UA-DBs, we lay the groundwork with a formal model for incomplete databases based on K-relations [19]. Specifically, we define a “pivoted” encoding of incomplete K-relations called KW-relations and prove that queries over such relations have possible worlds semantics. We also generalize the classical notion of certain answers to KW-relations based on the observation that certain answers represent a lower bound on the content of a possible world. It is thus natural to define certainty based on a greatest-lower-bound operation (GLB) for semiring annotations based on so-called l-semirings [30] where the GLB is well behaved. We show that certainty defined based on the GLB operation corresponds to existing definitions of certain answers for incomplete data under both set [36] and bag [23] semantics.

Approximating Certain Answers. (Section 4) We define uncertainty labelings, which are K-relations that approximate the set of certain tuples for KW-relations. An uncertainty labeling is certain- or c-sound (resp., c-complete) if it is a lower (resp., upper) bound on the set of certain tuples in a KW-relation; and c-correct if it is both. We also extend these definitions to query semantics. A query semantics preserves c-soundness if the result of the query is a c-sound labeling for the result of evaluating the query over the input KW-database from which the labeling was derived.

Creating Uncertainty Labelings. (Section 5) We next show that uncertainty labelings are backwards compatible by showing how to derive labelings for two common incomplete/probabilistic data models: tuple-independent databases (TI-DB) and C-tables. For the former we prove that the labeling is guaranteed to be c-complete, while for the latter we prove that the labeling is guaranteed to be at least c-sound.

Queries over Uncertainty Labelings. (Section 6) Since uncertainty labelings are K-relations, we can evaluate queries over such labelings. However, the question is whether this produces any meaningful results. We answer this question affirmatively by demonstrating that evaluating queries in this fashion preserves c-soundness. Furthermore, if the input labeling is derived from a TI-DB, then RA+ queries also preserve c-correctness (i.e., compute the certain answers).

UA-DBs. (Section 7) We combine uncertainty labelings with curation-based evaluation by defining annotation structures that encode both an uncertainty labeling as well as one possible world of a KW-database. The result is a UA-DB.

Implementation for Bag Semantics. (Section 8) We implement UA-DBs on top of a standard relational DBMS. We extend the schema of relations with an additional attribute to label tuples as either certain or uncertain (e.g., Figure 2c). Queries with UA-relational semantics are compiled into standard relational queries over this encoding.

Performance. Finally, in Section 9 we demonstrate experimentally that UA-DBs outperform state of the art probabilistic query processing schemes, while remaining competitive with classical deterministic (i.e., curation-based) query evaluation. We also show empirically that in at least some applications, the conditions required for UA-DBs to produce c-incomplete answers rarely arise.

2. NOTATION AND BACKGROUND

A database schema \( D = \{R_1, \ldots, R_n\} \) is a set of relation schemas. A relational schema \( R(A_1, \ldots, A_n) \) consists of a
relation name and a set of attribute names $A_1, \ldots, A_n$. The arity $\text{arity}(R)$ of a relation schema $R$ is the number of attributes in $R$. A database instance $D$ for database schema $R$ is a set of relation instances with one relation for each relation schema $R$. Assume a universal domain of attribute values $\mathbb{D}$. A tuple with schema $R$ is an element from $\mathbb{D}^{\text{arity}(R)}$. In this paper, we consider both bag and set semantics. A set semantics relation $R$ with schema $\mathbb{D}$ is a set of tuples with schema $R$, i.e., $R \subseteq \mathbb{D}^{\text{arity}(R)}$. A bag semantics relation $R$ with schema $\mathbb{D}$ is a bag (multiset) of tuples with schema $R$. We use $\text{TupDom}$ to denote the set of all tuples over domain $\mathbb{D}$.

### 2.1 Possible Worlds Semantics

Regardless of its source, uncertainty and its impact on query results can be modelled using incomplete or probabilistic databases. An incomplete database $D$ is a set of deterministic database instances $D_1, \ldots, D_n$ called possible worlds. A probabilistic database $D$ is an incomplete database paired with a probability distribution $P : \{D_i\} \rightarrow [0, 1]$ that assigns each possible world a probability from $[0, 1]$ with the requirement that $\sum_{D \in \mathbb{D}} P(D) = 1$. We write $t \in D$ to denote that a tuple $t$ appears in a specific possible world $D$, and $P(t \in D)$ to denote the marginal probability, or confidence of a given tuple appearing in any possible world:

$$P(t \in D) = \sum_{D \in \mathbb{D}} P(D)$$

**Example 3.** Continuing Example 4, Figure 3 shows the two possible worlds encoded by the instance from Figure 3. Observe that some tuples (e.g., (1, Lasalle, NY)) appear in all possible worlds — these are part of the instance no matter which possible world is chosen. Such tuples are called certain. Tuples that appear in at least one possible world (e.g., (2, Tucson, AZ)) are called possible.

Decades of research [43, 25, 21, 6, 7, 3] has focused on algorithms for efficient query processing over incomplete or probabilistic databases. These techniques commonly adopt the so-called possible worlds semantics: The result of evaluating a query $Q$ over an incomplete database is the set of relation instances resulting from evaluating $Q$ over each possible world individually using standard query semantics.

$$Q(D) = \{ Q(D') \mid D \in \mathbb{D} \}$$

The probability of a world in the result of evaluating a query $Q$ over a probabilistic database $D$ is computed as $P(D) = \sum_{D' \in \mathbb{D} \land Q(D') = D} P(D')$.

**Example 4.** Consider the following query over $D$ from Figure 3 which returns locations in NY State.

$$Q_{NY} = \sigma_{\text{state}=\text{NY}}(D)$$

The result of $Q_{NY}(D)$ is the set of instances computed by evaluating $Q_{NY}$ over each instance of $D$ as shown in Figure 4. The possible result in 4 is only produced by $Q_{NY}(D_1)$ and consequently has probability 0.8. Similarly, the result in 4 has probability 0.2. The row for location 2 appears in $Q_{NY}(D_2)$, but not $Q_{NY}(D_1)$ and so has probability 0.2.

### 2.2 Certain, Possible, and Best-Guess Answers

It is also possible to eschew probabilities, and focus on differentiating query results which are certain from ones that are merely possible. Formally, a tuple is certain (possible) if it appears in every (at least one) possible world

$$\text{certain}(D) = \{ t \mid \forall D \in \mathbb{D} : t \in D \}$$

$$\text{possible}(D) = \{ t \mid \exists D \in \mathbb{D} : t \in D \}$$

In contrast to [25], which studies certain answers to queries, we define certainty at the instance level. The two approaches are equivalent since we can compute the certain answers of query $Q$ over incomplete database $D$ as certain($Q(D)$). Although computing certain (possible) answers is $\text{coNP}$-hard (NP-complete) in general, there exist $\text{PTIME}$ approximations [35, 22, 40] that allow for false positives (tuples are returned that are not certain answers) or false negatives (some certain answers may be omitted from the query result).

Another approach commonly used in practice is to simply ignore ambiguity in the data: Manual curation, ETL workflows, or similar heuristic methods select one database instance from among the space of possible worlds. Queries are evaluated solely in this possible world, and ambiguity is ignored, or documented outside of the database if the analysts are particularly diligent. We refer to this practice as best-guess query processing (BGQP) [17], and we refer to the selected instance as the best guess world $\text{BestGuess}(D)$.

**Example 5.** For probabilistic DBs, we define the best guess world as $\text{BestGuess}(D) = \arg\max_{D \in \mathbb{D}} (P(D))$, i.e., the world with the highest probability. Hence, Figure 4 (disregarding the certain? column) shows a best guess world.

### 2.3 K-relations

Our generalization of incomplete databases, uncertainty labeling schemes, and UA-relations are all based on the $K$-relation [19] framework. In this framework, relations are annotated with elements from the domain $K$ of a commutative semiring $\mathbf{K}$. A structure $\mathbf{K} = (\mathbf{K}, \otimes, \oplus, 0, \mathbf{K})$ is a commutative semiring iff (1) addition and multiplication are associative and commutative, (2) $0$ is the neutral element of addition, (3) $1$ is the neutral element of multiplication, (4) multiplication with $0$ yields $0$, and (5) multiplication distributes over addition. For simplicity, we will refer to commutative semirings simply as semirings.

As before, we use $\mathbb{D}$ to denote a universal domain of values. An $n$-ary $K$-relation is a function that maps tuples (elements from $\mathbb{D}^n$) to elements from $K$. Tuples that are not in the relation are annotated with $0_{\mathbf{K}}$. If $\mathbb{D}$ is infinite,
then it is additionally required that only finitely many tuples are mapped to an element other than 0_K (i.e., relations must be finite). The annotations assigned to tuples serve different purposes depending on which semiring K is used.

### Encoding Sets and Bags

Consider the following two examples of commutative semirings: the natural numbers (N) with addition and multiplication (N, +, ×, 0, 1) and boolean constants B = {T, F} with disjunction (v) and conjunction (w) forming the semiring (B, v, w, T, F).

In the following, we will abuse notation and denote by N and B respectively, both the domain and the corresponding semiring structure. Green et al. [19] demonstrated that, for an appropriate choice of semiring, K-relations can be used to encode standard set and bag semantics, as well as many extensions of the relational model including various types of provenance. Specifically, B (N) encodes set (bag) semantics by annotating all tuples in the relation with true (their multiplicity) and all other tuples with false (0). Since K-relations are functions from tuples to annotations, it is customary to denote the annotation of a tuple t in relation R as R(t) (applying function R to input t).

#### Query Semantics

Operators of the positive relational K-relations are functions from tuples to annotations, it is customary to annotate all tuples in the relation with true (their multiplicity) and all other tuples with false (0). Since K-relations are functions from tuples to annotations, it is customary to denote the annotation of a tuple t in relation R as R(t) (applying function R to input t).

**Union:** \[ R_1 \cup R_2(t) = R_1(t) \oplus K R_2(t) \]

**Join:** \[ R_1 \bowtie R_2(t) = R_1(t[R_1]) \odot K R_2(t[R_2]) \]

**Projection:** \[ [\pi_U(R)](t) = \sum_{t \in U} R(t) \]

**Selection:** \[ [\sigma_{\theta}(R)](t) = R(t) \odot K \theta(t) \]

For simplicity, we assume that tuples are of a compatible schema (e.g., R_1 for a union R_1 \cup R_2). We use \( \theta(t) \) to denote a function that returns 1_K if \( t \) evaluates to true over the values from tuple t and 0_K otherwise, and \( t\cup U \) and \( t\cup R \) to denote projecting tuple t on a set of attributes U and restricting it to the schema of relation R, respectively.

**Example 6.** Figure 5 shows an example of how to encode a bag semantics database as an N-database by annotating each tuple t with its multiplicity (the number of duplicates of t that exist in the relation). Annotations are shown to the right of each tuple. Query Q_a, shown below, returns states.

\[
Q_a = \pi_{state}(Address \times Neighborhood)
\]

In the input database every tuple appears once (is annotated with 1). The annotation of an output tuple is computed by multiplying annotations of joined tuples, and summing up annotations of projected tuples. For instance, 2 NY addresses are returned.

In the following, we will make use of homomorphisms. A mapping \( h : K \rightarrow K' \) from a semirings K to a semiring K' is called a homomorphism if for any \( k, k' \in K \):

\[
h(0_K) = 0_{K'}, \quad h(1_K) = 1_{K'}
\]

As observed by Green et al. [19], any semiring homomorphism can be lifted to a homomorphism from K-relations to K'-relations by applying h to the annotation of every tuple \( t : h(R(t)) = h(R(t)) \). Importantly, it was shown that queries commute with semiring homomorphisms. That is, given a homomorphism h, query Q, and K-database D we have \( h(Q(D)) = Q(h(D)) \). We will abuse syntax and use the same function symbols (e.g., h(·)) to denote mappings between K-semirings, K-relations, as well as K-databases.

**Example 7.** Continuing with Example 6, assume we want set semantics instead. We can derive a set semantics instance (semiring B) by applying to each table a mapping \( h : N \rightarrow B \) defined as \( h(k) = T \) if \( k > 0 \) and \( h(k) = F \) otherwise. The mapping h is a semiring homomorphism (we leave the proof as an exercise for the reader). Thus, evaluating \( Q_a \) in N and then applying h to the result (i.e., \( h(Q(D)) \)) produces the same result as applying h to the input database and then evaluating \( Q_a \) in B (i.e., \( Q(h(D)) \)).

When defining upper and lower bounds for annotations in Section 5, we will make use of the fact that for many semirings K it is possible to define an order relation over K based on the addition operation of the semiring. This order relation is called the natural order. The natural order \( \leq_K \) for a semiring K is defined as: An element k precedes k’ if it is possible to obtain k’ by adding to k.

\[
\forall k, k' \in K : k \leq_K k' \iff \exists k'' \in K: k + k'' = k'
\]

Semirings for which the natural order is a partial order are called naturally ordered [18]. For naturally ordered semirings that also fulfill an additional condition we can define a monus operation \( \ominus_K \) which serves the role of subtraction. We will use this operation in Section 7.2. The condition that has to hold is: for each pair of elements k and k' from K, the set of elements \( \{k'' \mid k \leq_K k'' \leq_K k' \} \) has to have a smallest member wrt. \( \leq_K \). For semirings which have this property, \( k_1 \ominus_K k_2 \) is the smallest k_3 such that \( k_1 \ominus_K k_3 \geq_K k_2 \).

### 3. INCOMPLETE K-RELATIONS

When reasoning about incomplete databases, it is common to consider set and bag semantics independently. Our first contribution unifies both under a joint framework based on K-relations. Concretely, we show how to define incomplete (and probabilistic) variants of a large class of K-relations, including B- and N-relations. We assume a fixed set \( W = \{m \mid m \in \mathbb{N} \land m \leq n \} \) of possible world identifiers for some \( n \in \mathbb{N} \). Given the domain K of a semiring K, we write \( K^W \) to denote the set of elements from the \( |W| \) -way cross product of K. We will annotate a tuple t with an element of \( K^W \) to store the annotation of t in each possible world.

**Definition 1.** Let \( K = (K, \oplus_K, \odot_K, 0_K, 1_K) \) be a naturally ordered semiring. We define the possible world semiring...
\[ K_W = (K^W, \odot_{K_W}, \odot_{K_W}, \omega_{K_W}, 1_{K_W}) \]. The operations of this semiring are defined as follows:

\[ \forall i \in W : 0_{K_W}[i] = 0 \]
\[ \forall i \in W : 1_{K_W}[i] = 1 \]
\[ \forall i \in W : (k_1 \odot_{K_W} k_2)[i] = k_1[i] \odot_K k_2[i] \]
\[ \forall i \in W : (k_1 \odot_{K_W} k_2)[i] = k_1[i] \odot_K k_2[i] \]

The example below illustrates the use possible worlds semirings to encode set and bag semantics incomplete databases.

**Example 8.** Continuing Example 3, we can encode the two possible worlds of Figure 3 under bag semantics using a \( \mathbb{N}^2 \)-relation or under set semantics using a \( \mathbb{B}^2 \)-relation:

<table>
<thead>
<tr>
<th>id</th>
<th>locale</th>
<th>state</th>
<th>( \mathbb{N}^2 )</th>
<th>( \mathbb{B}^2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Lasalle</td>
<td>NY</td>
<td>[1, 1]</td>
<td>[T, T]</td>
</tr>
<tr>
<td>2</td>
<td>Tucson</td>
<td>AZ</td>
<td>[1, 0]</td>
<td>[T, F]</td>
</tr>
<tr>
<td>3</td>
<td>Grant Ferry</td>
<td>NY</td>
<td>[0, 1]</td>
<td>[F, T]</td>
</tr>
<tr>
<td>4</td>
<td>Kingsley</td>
<td>NY</td>
<td>[1, 1]</td>
<td>[T, T]</td>
</tr>
</tbody>
</table>

Note that the two interpretations of location 2 create two tuples which do not occur together in any possible world.

Observe that \( K_W \) is a semiring, since we define \( K_W \) using the \( |W| \)-way version of the product operation of universal algebra and products of semirings are also semirings 8.

**Possible Worlds.** We can extract a possible world (a \( K \)-database) from a \( K_W \)-database by selecting one dimension from each of the vectors that annotates tuples. This can be modelled as a mapping \( \text{pw}_i : K^W \rightarrow K \) where \( i \in W \):

\[ \text{pw}_i(k) = k[i] \quad (5) \]

Recall that under possible world semantics, the result of a query \( Q \) is the set of possible worlds computed by evaluating \( Q \) over each world of the input. As a basic sanity check, we would like to ensure that query processing over \( K_W \)-relations matches this definition. Observe that we can state possible world query semantics equivalently as follows: the content of a possible world in the query result \((\text{pw}_i(Q(D)))\) is the result of evaluating query \( Q \) over this possible world in the input \((Q(\text{pw}_i(D)))\). That is, \( K_W \)-relations have possible worlds semantics iff \( \text{pw}_i \) commutes with queries:

\[ \forall i \in W : \text{pw}_i(Q(D)) = Q(\text{pw}_i(D)) \]

Recall from Section 2.3 that a mapping between semirings commutes with queries iff it is a semiring homomorphism. Thus, Lemma 1 shown below implies that queries over \( K_W \)-relations have possible worlds semantics.

**Lemma 1.** For any semiring \( K \) and possible world \( i \in W \), mapping \( \text{pw}_i \) is a semiring homomorphism.

**Proof.** Trivially proven by substitution of definitions.

\[ \text{pw}_i(0_{K_W}) = 0_{K}[i] = 0_K \]
\[ \text{pw}_i(1_{K_W}) = 1_{K_W}[i] = 1_K \]
\[ \text{pw}_i(k_1 \odot_{K_W} k_2)[i] = (k_1 \odot_{K_W} k_2)[i] = k_1[i] \odot_K k_2[i] \]
\[ = \text{pw}_i(k_1) \odot_K \text{pw}_i(k_2) \]
\[ \text{pw}_i(k_1 \odot_{K_W} k_2)[i] = (k_1 \odot_{K_W} k_2)[i] = k_1[i] \odot_K k_2[i] \]
\[ = \text{pw}_i(k_1) \odot_K \text{pw}_i(k_2) \]

\[ \square \]

**Probabilistic Data.** \( K_W \)-relations admit a trivial extension to probabilistic data by defining a distribution \( P : W \mapsto [0, 1] \) such that \( \sum_{i \in W} P(i) = 1 \). In contrast to classical framework for possible worlds, where the collection of worlds is a set, \( K_W \) queries preserve the same \( W \) possible worlds\(^2\).

Hence, the input distribution \( P \) applies, unchanged, to the \( W \) possible query outputs.

**Certain and Possible Tuples.** Under set semantics, a tuple from an incomplete or probabilistic database is called certain if it appears in all possible worlds and possible if it appears in at least one possible world. Intuitively, certain (possible) denotes a lower (upper) bound on the content of any given possible world. We adopt this intuition to generalize these notions to \( K_W \)-databases.

We utilize the natural order \( \preceq_K \) as introduced in Section 2.3 to define the greatest lower bound and least upper bound of a set of \( K \)-elements for this purpose. For that to be well-defined we require that \( \preceq_K \) forms a lattice over \( K \).

Using the terminology from \footnote{Although it has no impact on our results, it is worth noting that the worlds in a \( K_W \) query result may not be distinct.}, we will refer to semirings which enjoy this property as \( l \)-semirings. A lattice over a set \( S \) and with a partial order \( \preceq_S \) is a structure \((S, \preceq_S)\) where \( \preceq \) (the least upper bound) and \( \sqcap \) (the greatest lower bound) are operations over \( S \) defined for all \( a, b \in S \) as:

\[ a \sqcup b = \min\{c : c \in S \land a \preceq_S c \land b \preceq_S c\} \quad \quad (\leq_S) \]
\[ a \sqcap b = \max\{c : c \in S \land c \preceq_S a \land c \preceq_S b\} \]

For \( S \) to be a lattice, \( \sqcup \) and \( \sqcap \) have to be associative, commutative, and fulfill the following absorption laws:

\[ a \sqcup (a \sqcap b) = a \quad a \sqcap (a \sqcup b) = a \]

In the following we will use \( \sqcup_K \) and \( \sqcap_K \) to denote the \( \sqcup \) and \( \sqcap \) operations of the lattice over \( \preceq_K \) for a semiring \( K \). Abusing notation, we will apply the \( \sqcup_K \) and \( \sqcap_K \) operations to elements from \( K_W \) with the understanding that they will be applied to the set of elements from such a vector. This is well-defined for \( l \)-semirings, since in a lattice any set of elements has a unique greatest lower bound and lowest upper bound.

From here on, we will limit our discussion to \( l \)-semirings. Many semirings, including the set semiring \( B \) and the bag semiring \( N \) are \( l \)-semirings. The natural order of \( B \) is \( F \preceq_S T \), \( k_1 \sqcup_B k_2 = k_1 \lor_k k_2 \), and \( k_1 \sqcap_B k_2 = k_1 \land_k k_2 \). The natural order of \( N \) is the standard order of natural numbers, \( k_1 \sqcup_B k_2 = \max(k_1, k_2) \), and \( k_1 \sqcap_B k_2 = \min(k_1, k_2) \).

Based on \( \sqcup_K \) and \( \sqcap_K \), we define the certain and possible annotations for \( K_W \)-elements and \( K_W \)-databases.

\[ \text{CERT}_K(\bar{k}) = \sqcap_K(\bar{k}) \quad \text{CERT}_K(D, t) = \text{CERT}_K(D(t)) \]
\[ \text{POSS}_K(\bar{k}) = \sqcup_K(\bar{k}) \quad \text{POSS}_K(D, t) = \text{POSS}_K(D(t)) \]

As a sanity check, we observe that our definition coincides with the standard definition of certain answers for set semantics (\( B \)), because \( \text{CERT}_B \) corresponds to \( \land \) (returns true if the tuple is present in all worlds) and \( \text{POSS}_B \) to \( \lor \) (the tuple appears in at least one world). Furthermore, \( \text{CERT}_N \) is min and \( \text{CERT}_N \) is max. This is also analogous to the definition for bag semantics given by Guagliardo and Libkin \footnote{Although it has no impact on our results, it is worth noting that the worlds in a \( K_W \) query result may not be distinct.} which defines the certain (possible) multiplicity of a tuple as the minimum (maximum) multiplicity of the tuple across all worlds. For
instance, consider how to compute the certain annotation of the first tuple from Example 1. \(\text{CERT}_{\Delta}(\{(T, T)\}) = T \land T = T\). That is, this tuple is certain. For the second tuple from this example we get \(\text{CERT}_{\Delta}(\{(T, F)\}) = T \land F = F\). As expected, this tuple is not certain. Under bag semantics the certain annotation of this tuple is \(\text{CERT}_{\Delta}(\{1, 0\}) = \min(1, 0) = 0\), i.e., its certain multiplicity is 0.

4. UNCERTAINTY LABELINGS

We now formally define uncertainty labelings, which approximate certain annotations of tuples in a \(\mathcal{K}_W\)-database. Here, by approximate we mean that labelings may over- or under-estimate the certain annotation of a tuple with respect to the natural order of semiring \(K\). We do not want to prescribe a particular representation for labelings. We only require that a labeling comes with a function \(\text{cert}\) that allows us to extract the estimate for the certain annotation of a tuple \(t\) encoded by the labeling. Furthermore, it should be possible to derive a labeling from a \(\mathcal{K}_W\)-database.

**Definition 2** (Uncertainty Labeling Scheme). An uncertainty labeling scheme is a tuple \(\text{LAB} = (\mathcal{L}, \text{cert}, \text{label})\) where \(\mathcal{L}\) is a set of possible labelings, \(\text{cert} : \mathcal{L} \times \text{TupleDOM} \to K\) is a function that extracts a certainty label \(k \in K\) for a tuple \(t\) from a labeling \(\mathcal{L}\), and \(\text{label} : \mathcal{K}_W\text{-database} \to \mathcal{L}\) is a function that creates a labeling for an input database \(\mathcal{D}\).

Ideally, we would like the label assigned to a tuple \(t\) by an uncertainty labeling to be equal to \(\text{CERT}_K(\mathcal{D}, t)\). However, as mentioned in the introduction, to define a tractable query semantics we will have to accept that a label \(\text{CERT}(\mathcal{L}, t)\) may either overestimate or underestimate \(\text{CERT}_K(\mathcal{D}, t)\) with respect to the natural order \(\preceq\). For instance, under bag semantics (semiring \(\mathbb{N}\)), a label \(n\) may be smaller or larger than the certain multiplicity of a tuple. A labeling is \(c\)-sound, i.e., does not allow for false positives, if it consistently underestimates the certain annotation of tuples, \(c\)-complete if it consistently overestimates certainty, and \(c\)-correct if it annotates every tuple with its certain annotation. We also apply this terminology to labeling schemes, e.g., a \(c\)-sound labeling scheme only produces \(c\)-sound labelings.

**Definition 3** (Correctness). Let \(\mathcal{L}\) be an uncertainty labeling for a \(\mathcal{K}_W\text{-database } \mathcal{D}\).

- **\(c\)**-sound iff for all tuples \(t \in \mathcal{D}\)
  - \(\text{CERT}(\mathcal{L}, t) \preceq \text{CERT}_K(\mathcal{D}, t)\)
- **\(c\)**-complete
  - \(\text{CERT}_K(\mathcal{D}, t) \preceq \text{CERT}(\mathcal{L}, t)\)
- **\(c\)**-correct
  - \(\text{CERT}(\mathcal{L}, t) = \text{CERT}_K(\mathcal{D}, t)\)

Observe that a labeling is both \(c\)-sound and \(c\)-complete iff it is \(c\)-correct. A query semantics over labelings should preserve the correctness properties of an input labeling if possible. That is, the result of evaluating a query \(Q\) over an uncertainty labeling \(\mathcal{L}\) for a \(\mathcal{K}_W\text{-database } \mathcal{D}\) should be a labeling for the result of evaluating the query over \(\mathcal{D}\).

**Definition 4** (Correctness Preservation). A query semantics for uncertainty labelings preserves a correctness property \(X\) (\(c\)-soundness, \(c\)-completeness, or \(c\)-correctness) wrt. a class of queries \(\mathcal{Q}\), if for any \(\mathcal{K}_W\text{-database } \mathcal{D}\), \(\mathcal{L}\) for \(\mathcal{D}\) that has property \(X\), and query \(Q \in \mathcal{Q}\) we have: \(Q(\mathcal{L})\) is an uncertainty labeling for \(Q(\mathcal{D})\) which also has property \(X\).

As mentioned in the introduction (and observed elsewhere [35, 22]), false negatives are less of a concern than false positives. Thus, for cases where we cannot preserve \(c\)-correctness, we would at least opt for \(c\)-soundness. That is, we prefer labelings that underestimate the “real” \(\text{CERT}_K(\mathcal{D}, t)\) annotation wrt. the natural order \(\preceq\).

5. FROM INCOMPLETE TO UNCERTAIN

In this section, we define efficient (\(\text{PTIME}\)) uncertainty labelings for two existing probabilistic and incomplete data models: Tuple-Independent databases [43] and (P)C-Tables [25, 21]. Let \(\mathcal{D}_K\) be the set of all \(K\)-databases. Both labeling schemes are of the form \(\mathcal{D}_K\)-label, \(\text{cert}_{\Delta}\), i.e., they use a \(K\)-database to encode a labeling, and use the annotation of a tuple in such a database its label: \(\text{cert}_{\Delta}(\mathcal{L}, t) = \mathcal{L}(t)\). In the following, we refer to such labeling as \(K\)-labelings. Since computing certain answers is hard in general, our \(\text{PTIME}\) labeling schemes cannot be \(c\)-correct in all cases.

**Tuple Independent Probabilistic Databases.** A tuple independent probabilistic database (TI-DB) \(\mathcal{D}\) is a database instance where each tuple \(t\) is annotated with its marginal probability \(\gamma(t) \leq 0 < P(t \in \mathcal{D}) \leq 1\). The probabilistic database represented by a TI-DB \(\mathcal{D}\) is the power-set of all of its tuples: For each tuple we decide whether the tuple should be part of the possible world or not. As the name indicates, the existence of tuples in TI-DBs is assumed to be independent of each other. Thus, the probability of a possible world \(D\) is the product of probabilities of tuples that do appear, and one minus the probability of tuples that do not. For a specific TI-DB \(\mathcal{D}\), we define a labeling function \(\text{label}_{\mathcal{D}}(t)\) that returns a \(\mathcal{B}\)-database labeling \(\mathcal{L}\), which annotates a tuple with \(T\) (certain) if its marginal probability is 1.

\[\mathcal{L}(t) = T \iff P(t \in \mathcal{D}) = 1\]

**Theorem 1** (\(\text{label}_{\mathcal{D}}\) is \(c\)-correct). Given a TI database \(\mathcal{D}\), \(\text{label}_{\mathcal{D}}(t)\) is a \(c\)-correct labeling.

**Proof.** Trivially holds, since in any probabilistic database a tuple is certain iff \(P(t) = 1\).

**C-tables.** C-Tables [24] enrich the active domain of the database with a set \(\Sigma\) of variable symbols, often called labeled nulls, that are used to represent unknown values. The set of possible worlds defined by a C-table \(\mathcal{D}\) is the set of instances that are derived from \(\mathcal{D}\) by replacing each variable symbol from \(\Sigma\) according to a valuation \(v\) that assigns variables to constants from \(\mathcal{D}\). Allowable valuations may be restricted by defining a global condition, a boolean expression over symbols from \(\Sigma\) that a valuation must satisfy. Finally, tuples are annotated by a local condition, a boolean expression over comparisons of values from \(\Sigma \cup \mathcal{D}\). A tuple appears only in worlds where its local condition is satisfied. When the global condition is replaced by a joint probability distribution over allowable valuations, then the resulting structure is called a Probabilistic-, or PC-Table [21]. Both C-Tables and PC-Tables are closed under relational algebra, and can be extended for closure under generalized projection [28] and aggregation [26]. C-tables generalize several other incomplete database models. For instance, a V-table is a C-table without local and global conditions (all conditions are constant true) and a Codd-table is a V-table where no variable appears more than once.

While determining a \(c\)-correct labeling for a TI-DB is straightforward and efficient, the same is not true for C-tables. Under the closed world assumption, C-tables are
closed under full relational algebra (first order queries). It was shown early on that computing certain answers for first order queries is coNP-complete [10] even for Codd-tables. Since we can represent the result of any first order query as a C-table and evaluating a query in this fashion is efficient, it follows that determining whether a tuple is certain in a C-table cannot be in PTIME.

Thus, we cannot hope for an efficient c-correct labeling scheme for C-tables. Instead, consider the following sufficient, but not necessary condition for a tuple to be certain. If (1) a tuple \( t \) in a C-table contains only constants and (2) its local condition \( \phi(t) \) is a tautology, then the tuple is certain. To see why this is the case, recall that under the closed-world assumption, a C-table represents a set of possible worlds, one for each valuation of the variables appearing in the C-table (to constants from \( \mathbb{D} \)). A tuple is part of a possible world corresponding to such a valuation if the tuple’s local condition is satisfied under the valuation. Thus, a tuple consisting of constants only, with a local condition that is a tautology is part of every possible world represented by the C-table. If the local condition of a tuple is in conjunctive normal form (CNF) then checking whether it is a tautology is efficient (PTIME). Our labeling scheme for C-tables applies this sufficient condition and, thus, is c-sound. Formally, \( \mathcal{L} = \text{label}_{C-table}(D) \), where for a C-table \( D \) and any tuple \( t \in \text{TupDOM} \):

\[
\mathcal{L}(t) = \exists \phi(t) \text{ in CNF} \land (\models \phi(t))
\]

**Theorem 2** (\text{label}_{C-table} is c-sound). Given an incomplete database \( D \) encoded as C-tables, \text{label}_{C-table}(D) is c-sound.

**Proof.** Let \( \mathcal{L} = \text{label}_{C-table}(D) \). By definition of \( \mathcal{L} \), a tuple \( t \) is labeled as certain iff \( \phi(t) \) is in CNF and \( \models \phi(t) \), which means the expression \( \phi(t) \) is a tautology. By definition of C-tables, a tuple \( t \) exists in a possible world if \( \phi(t) \) evaluates to true in that possible world. Thus \( t \) must exist in all possible worlds if \( \phi(t) \) is a tautology. Thus, \( \mathcal{L} \) is c-sound.

Note that \( \mathcal{L} \) is not guaranteed to be c-correct. For instance, a tuple where \( \phi(t) \) is a tautology is guaranteed to be certain, but \( \mathcal{L}(t) = F \) if \( \phi(t) \) is not in CNF.

**Example 9.** Consider a C-table consisting of two tuples
\[
t_1 = (1, X) \quad \text{with} \quad \phi(t_1) = (X = 1)
\]
and
\[
t_2 = (1, 1) \quad \text{with} \quad \phi(t_2) = (X \neq 1).
\]
\text{label}_{C-table} would mark \((1, 1)\) as uncertain, because even though this tuple exists in the C-table and its local condition is in CNF, the local condition is not a tautology. However, tuple \((1, 1)\) is certain since either \( X = 1 \) and then first tuple evaluates to \((1, 1)\) or \( X \neq 1 \) and the second tuple is included in the possible world.

### 6. QUERYING LABELINGS

Motivated by the effectiveness of K-labeling schemes as we described in Section 3, we study whether queries over K-labelings preserve c-soundness and c-completeness. Specifically, we demonstrate that standard K-relational query evaluation preserves c-soundness for K-labeling schemes. Recall that a query semantics for labelings preserves c-soundness if for any K\(_W\)-database \( D \) and a c-sound labeling \( \mathcal{L} \) for \( D \), the output of any query \( Q \) evaluated over \( \mathcal{L} \) is a c-sound labeling for \( Q(D) \). Our result generalizes a previous result of Reiter [40] to any type of K\(_W\)-database for which we can define an efficient c-sound labeling scheme. Furthermore, we show in Section 6.2 that positive relational algebra (i.e., SPJU) queries also preserve c-completeness if the input is a labeling for a TI-DB. We will make use of the following lemma which demonstrates that the natural order of a semiring factors through addition and multiplication. This is a known result which we only state for completeness.

**Lemma 2.** Let \( K \) be a naturally ordered semiring. For all \( k_1, k_2, k_3, k_4 \in K \) we have:

\[
k_1 \preceq_K k_3 \land k_2 \preceq_K k_4 \Rightarrow k_1 \oplus_K k_2 \preceq_K k_3 \oplus_K k_4
\]

**Proof.** \( \oplus_K \): Based on the definition of \( \preceq_K \), if \( k_1 \preceq_K k' \) then there exists \( k'' \) such that \( k_1 \oplus_K k'' = k' \). Thus, \( k_3 = k_1 \oplus_K k'' \) and \( k_4 = k_3 \oplus_K k'' \) for some \( k'' \). Also, \((k_1 \oplus_K k_3) \preceq_K (k_1 \oplus_K k_3) \oplus_K k'' \) for any \( k'' \) and we get:

\[
k_1 \oplus_K k_3 \preceq_K (k_1 \oplus_K k_3) \oplus_K (k_1 \oplus_K k_3) \preceq_K (k_1 \oplus_K k_3) \oplus_K (k_1 \oplus_K k_3) = k_2 \oplus_K k_4
\]

\( \otimes_K \): The proof for multiplication \( \otimes_K \) is similar.

\[
(k_1 \otimes_K k_3) = (k_1 \otimes_K k_3) \otimes_K (k_1 \otimes_K k_3) \otimes_K (k_1 \otimes_K k_3) = (k_1 \otimes_K k_3) = k_2 \otimes_K k_4
\]

6.1 Preservation of C-Soundness

We now prove that \( R,A^+ \) over K-labelings preserves c-soundness. Since queries over both K\(_W\)-databases and K-labelings have K-relational query semantics, we can make use of the fact that \( R,A^+ \) over K-relations is defined using \( \otimes_K \) and \( \otimes_K \). At a high level, the argument is as follows: (1) we show that \text{CERT}\(_K\) applied to the result of an addition (or multiplication) of two K\(_W\)-elements \( k_1 \) and \( k_2 \) yields a larger (wrt. \( \preceq_K \)) result than adding (or multiplying) the result of applying \text{CERT}\(_K\) to \( k_1 \) and \( k_2 \); (2) Since c-sound labelings for an input provide a lower bound on \text{CERT}\(_K\)(\( k_i \)), we can apply Lemma 2 to show that the query result in the labeling is a lower bound for the result of the query over a c-correct labeling. Combining the two arguments we get c-soundness.

Functions that have the property mentioned in (1) are called superadditive and supermultiplicative. Formally, a function \( f : A \rightarrow B \) where \( A \) and \( B \) are closed under addition and multiplication, and \( B \) is ordered (order \( \leq_B \)) is superadditive (supermultiplicative) iff for all \( a_1, a_2 \in A \):

\[
f(a_1 + a_2) \geq_B f(a_1) + f(a_2) \quad \text{superadditive}
\]
\[
f(a_1 \times a_2) \geq_B f(a_1) \times f(a_2) \quad \text{supermultiplicative}
\]

In a nutshell, if we are given a c-correct K-labeling, then evaluating any \( R,A^+ \)-query over the labeling using K-relational query semantics preserves c-soundness if we can prove that \text{CERT}\(_K\) is superadditive and supermultiplicative.

**Lemma 3.** Let \( K \) be a semiring. \text{CERT}\(_K\) is superadditive and supermultiplicative wrt. the natural order \( \preceq_K \).

**Proof.** Recall that \( \otimes_K \) and \( \otimes_K \) are defined element-wise and that \( \text{CERT}\(_K\)(\( k_i \)) = \text{CERT}\(_K\)(\( k_i \))). Furthermore, \( k_1 \preceq_K k_2 \) iff \( \forall \mathcal{L} : k_1 \otimes_K k' = k_2 \). Consider an arbitrary \( k_1, k_2 \in K\(_W\). Let \( k_{\text{glb}} = \text{CERT}\(_K\)(\( k_1 \)) \) and \( k_{\text{glb}} = \text{CERT}\(_K\)(\( k_2 \)). Based on the definition of \( \text{CERT}\(_K\) \) this implies that for any \( i, k_{\text{glb}} \preceq_K k_i[i] \) which in
Let $k_{glb} = \Gamma_K(\tilde{k}_1 \sqcup_K \tilde{k}_2)$. We are going to prove that $k_{glb_1} \sqcup_K k_{glb_2}$ is a lower bound for $(\tilde{k}_1 \sqcup_K \tilde{k}_2)[i]$, i.e., that $\forall i \in W: k_{glb_1} \sqcup_K k_{glb_2} \preceq_K (\tilde{k}_1 \sqcup_K \tilde{k}_2)[i]$. Since, $k_{glb}$ is the greatest lower bound this implies that $k_{glb_1} \sqcup_K k_{glb_2} \preceq_K k_{glb}$. Consider an arbitrary $i \in W$. Based on the discussion above we have:

\[(\tilde{k}_1 \sqcup_K \tilde{k}_2)[i] = k_1[i] \preceq (k_{glb_1} \sqcup_K k_{glb_2}) \preceq_K k_{glb} \preceq_K k_{glb_1} \sqcup_K k_{glb_2},\]

Thus, $k_{glb_1} \sqcup_K k_{glb_2}$ is a lower bound and since $k_{glb_1} = \text{cert}_K(\tilde{k}_1)$ and $k_{glb_2} = \text{cert}_K(\tilde{k}_2)$ it follows that $\text{cert}_K$ is superadditive:

\[\text{cert}_K(\tilde{k}_1) \sqcup_K \text{cert}_K(\tilde{k}_2) \preceq_K \text{cert}_K(\tilde{k}_1 \sqcup_K \tilde{k}_2)\]

Supermultiplicativity: We use an analogous argument to prove supermultiplicativity. Let $k_{glb} = \text{cert}_K(\tilde{k}_1 \sqcup \tilde{k}_2)$. We will prove that $k_{glb_1} \sqcup_K k_{glb_2}$ is a lower bound for $(\tilde{k}_1 \sqcup_K \tilde{k}_2)[i]$ which implies supermultiplicativity. Consider $i \in W$:

\[
\begin{align*}
(\tilde{k}_1 \sqcup_K \tilde{k}_2)[i] &= (k_{glb_1} \sqcup_K k_{glb_2}) \sqcup_K (k_{glb_1} \sqcup_K k_{glb_2})^\prime \\
&= (k_{glb_1} \sqcup_K k_{glb_2}) \sqcup_K (k_{glb_1} \sqcup_K k_{glb_2})^\prime \\
&\preceq (k_{glb_1} \sqcup_K k_{glb_2}) \\
&\preceq_K (k_{glb_1} \sqcup_K k_{glb_2})
\end{align*}
\]

Using the superadditivity and -multiplicativity of $\text{cert}_K$, we now prove preservation of c-soundness. We first prove a restricted version of this result.

**Lemma 4.** Let $\mathcal{D}$ be a $K_W$-database and $\mathcal{L}$ be a $c$-correct $K$-labeling for $\mathcal{D}$. $\mathcal{R}A^+$ queries over $\mathcal{D}$ preserve c-soundness.

**Proof.** Consider an $\mathcal{R}A^+$ query $Q$ and $\mathcal{D}$ an $K_W$-database. To prove preservation of c-soundness, we have to show that the result of $Q(\mathcal{L})$ is a c-sound labeling for $Q(\mathcal{D})$, i.e., that for any tuple $t$ we have $Q(\mathcal{L})(t) \preceq_K \text{cert}_K(Q(\mathcal{D}))(t)$. Recall that $\mathcal{R}A^+$ queries over $K_W$-relations and queries over $K$-labelings are defined using the semiring addition and multiplication operations. Hence, the claim

\[Q(\mathcal{L})(t) \preceq_K \text{cert}_K(Q(\mathcal{D}))(t)\]

follows immediately from the superadditivity and supermultiplicativity of $\text{cert}_K$ (Lemma 3) and the fact that $\mathcal{L}$ is a $c$-correct labeling.

The major drawback of Lemma 4 is that it is limited to $c$-correct input labelings. Next, we show that c-soundness is still preserved even if the input labeling is only c-sound.

**Theorem 3.** Let $\mathcal{D}$ be a $K_W$-database, $\mathcal{L}$ a c-sound $K$-labeling for $\mathcal{D}$. $\mathcal{R}A^+$ queries over $\mathcal{L}$ preserve c-soundness.

**Proof.** Since $\mathcal{L}$ is a c-sound labeling for any tuple $t$ we have $L(t) \preceq_K \text{cert}_K(\mathcal{D}, t)$. We have to prove that for any tuple $t$ we have $Q(\mathcal{L})(t) \preceq_K \text{cert}_K(\mathcal{Q}(\mathcal{D}))(t)$.

For that we show that for any $k_1, k_2 \in K$ and $\tilde{k}_1, \tilde{k}_2 \in K_W$ such that $k_1 \preceq_K \text{cert}_K(\tilde{k}_3)$ and $k_2 \preceq_K \text{cert}_K(\tilde{k}_4)$, we have $(k_1 \sqcup_K k_2) \preceq_K \text{cert}_K(\tilde{k}_1 \sqcup_K \tilde{k}_2)$ and $k_1 \sqcup_K k_2 \preceq_K \text{cert}_K(\tilde{k}_1 \sqcup_K \tilde{k}_2)$.

Since by assumption the input labeling is c-sound, we have $\text{cert}(\mathcal{L}, t) \preceq_K \text{cert}(\mathcal{D}, t)$ for any tuple $t$. Thus, based on the property we have just proven and the fact the K-relational query semantics is defined based on the operations of semirings only, this implies that for any tuple $t$: $\text{cert}(Q(\mathcal{L}), t) = Q(\mathcal{L})(t) \preceq_K \text{cert}(Q(\mathcal{D}), t)$.

**6.2 Preservation of C-Completeness**

We now demonstrate that positive queries preserve c-completeness if the input is a $K$-labeling produced by our labeling scheme for TI-DBs. Recall that we already proven that the labeling scheme for TI-DBs presented in Section 3 is c-complete. The proof of preservation of c-completeness is based on the following observation: If two $K_W$-elements $k_1$ and $k_2$ are minimal in the same possible world, i.e., there exists a possible world $i$ such that $\Gamma_k(\tilde{k}_1) = k_1[i]$ and $\Gamma_k(\tilde{k}_2) = k_2[i]$, then $\Gamma_k$ commutes with addition and multiplication. Thus, standard $K$-relational query evaluation semantics over c-complete labelings preserves c-completeness if the aforementioned property holds. We then prove that every $K_W$ encoding of a TI-DB database has this property.

**Lemma 5.** Let $k_1, k_2 \in K_W$ for some possible world semiring $K_W$. If there exists $i \in W$ such that $\Gamma_k(\tilde{k}_1) = k_1[i]$ and $\Gamma_k(\tilde{k}_2) = k_2[i]$, then the following holds:

\[
\Gamma_k(\tilde{k}_1 \sqcup_K \tilde{k}_2)[i] = \Gamma_k(\tilde{k}_1) \sqcup_K \Gamma_k(\tilde{k}_2) = (k_1 \sqcup_K k_2)[i]
\]

**Proof.** Recall that $\mathcal{P}$, is a homomorphism (Lemma 1), so $(k_1 \sqcup_K k_2)[i] = k_1[i] \sqcup_K k_2[i]$ and $\Gamma_k(\tilde{k}_1) = k_1[i]$ for $j \in \{1, 2\}$. Thus, $\tilde{k}_1[i] = \Gamma_k(\tilde{k}_1) \sqcup_K \Gamma_k(\tilde{k}_2)$. It remains to be shown that $\Gamma_k(\tilde{k}_1 \sqcup_K \tilde{k}_2) = (k_1 \sqcup_K k_2)[i]$ which holds for any $j \neq i \in W$ we have $(k_1 \sqcup_K k_2)[i] \preceq (\tilde{k}_1 \sqcup_K \tilde{k}_2)[j]$. Since $k_1[i] = \Gamma_k(\tilde{k}_1)$ and $\Gamma_k$ is defined based on the natural order, we know that $k_1[i] \preceq k_1[j]$ and analog for $k_2$ we have $k_2[i] \preceq k_2[j]$. Based on Lemma 2 this implies $(k_1 \sqcup_K k_2)[i] \preceq (\tilde{k}_1 \sqcup_K \tilde{k}_2)[j]$. The proof for multiplication is analogous using Lemma 3 to show that $(k_1 \sqcup_K k_2)[i] \preceq (\tilde{k}_1 \sqcup_K \tilde{k}_2)[j]$ for any $j \in W$.

To demonstrate c-completeness preservation for TI-DBs we have to demonstrate that the encoding of a TI-DB as a $K_W$-database fulfills the precondition of Lemma 5.

**Lemma 6.** Let $\mathcal{D}$ be a $K_W$-database that represents a TI-DB. Then there exists $i \in W$ such that for any tuple $t$: 

\[
\text{cert}(\mathcal{L}, t) \preceq_K \text{cert}(\mathcal{D}, t)
\]


\[ \forall t, (D(t)) = D(t)[i]. \]

**Proof.** Consider the possible world \( D \) defined as follows:

\[
D(t) = \begin{cases} 
\bigcap_{K} (D(t)) & \text{if } P(t) = 1 \\
0_K & \text{otherwise}
\end{cases}
\]

This world exists, because in a TI-DB all tuples with probability equals to 1 carry the same annotation in every possible world \( i \). Furthermore, since the tuples are assumed to be independent events there exists a possible world that does contain none of the tuples with probability less than one. Let \( i \) denote the identifier of \( D \) in \( D \), i.e., \( pw_i(D) = D \). We claim that \( \bigcap_{K} (D(t)) = D(t)[i] \) for all \( t \in \text{TUPDOM} \). We have to distinguish two cases. Either \( P(t) = 1 \) and \( D(t)[j] = D(t)[i] \) for any \( j \in W \) and, thus, \( \bigcap_{K} (D(t)) = (D(t)[i]) \). Otherwise, \( P(t) < 1 \) and \( D(t) = D(t)[i] = 0_K \). Since, \( 0_K \) is the smallest element in \( K \) wrt. \( \preceq_K \) from this follows that \( \bigcap_{K} (D(t)) = 0_K = D(t)[i] \).

\( \square \)

Lemmas 3 and 4 together imply that our labeling approach preserves c-completeness if the input is a TI-DB.

**Corollary 1.** Let \( L \) be a labeling for a TI-DB \( D \) computed as \( label_1(D) \). Then \( RA^+ \) over \( L \) preserves c-completeness.

**7. UA-DATABASES**

We now introduce \( UA-DBs \), which are databases that simultaneously encode (1) a single possible world from a \( KU \)-database \( D \), and (2) an uncertainty labeling for \( D \). As motivated in the introduction, by annotating a possible world with uncertainty information we get the best of certain answer semantics and deterministic query processing. We first define \( UA \)-semirings, a \( KU \)-semiring semantics for \( UA-DBs \) and to prove that this semantics preserves c-soundness of the labeling encoded by a \( UA-DB \).

**Theorem 4.** If \( K \) is a commutative semiring, then \( K_{UA} \) is a commutative semiring.

**Proof.** A structure is a commutative semiring if it fulfills the equational laws of commutative semirings: addition and multiplication are commutative and associative, multiplication distributes over addition, multiplication by \( 0_{K} \) yields \( 0_{K} \), and \( 0_{K} \) and \( 1_{K} \) are the identity elements of addition and multiplication, respectively. For any \( k_1 = (u_1, c_1), k_2 = (u_2, c_2) \), and \( k_3 = (u_3, c_3) \) we have:

**Commutativity:**

\[
k_1 \otimes_{K_{UA}} k_2 = [u_1 \otimes_{K} u_2, c_1 \otimes_{K} c_2] = [u_2 \otimes_{K} u_1, c_2 \otimes_{K} c_1] = k_2 \otimes_{K_{UA}} k_1
\]

**Associativity:**

\[
k_1 \otimes_{K_{UA}} (k_2 \otimes_{K_{UA}} k_3) = [u_1 \otimes_{K} (u_2 \otimes_{K} u_3), c_1 \otimes_{K} (c_2 \otimes_{K} c_3)] = [(u_1 \otimes_{K} u_2) \otimes_{K} u_3, (c_1 \otimes_{K} c_2) \otimes_{K} c_3] = (k_1 \otimes_{K_{UA}} k_2) \otimes_{K_{UA}} k_3
\]

**Identity element:**

In the following we will write \( kk' \) instead of \( k \otimes_{K} k' \) if the semiring \( K \) is clear from the context.

**Definition 5 (UA-semiring).** Let \( K \) be a semiring, we define the corresponding \( UA \)-semiring

\[ K_{UA} = (K^2, \oplus_{K_{UA}}, \otimes_{K_{UA}}, 0_{K_{UA}}, 1_{K_{UA}}) \]

where for any \( [u_1, c_1], [u_2, c_2] \in K^2 \):

\[
[u_1, c_1] \oplus_{K_{UA}} [u_2, c_2] = [u_1 \oplus_{K} u_2, c_1 \oplus_{K} c_2]
\]

\[
[u_1, c_1] \otimes_{K_{UA}} [u_2, c_2] = [u_1 u_2 \oplus_{K} u_1 c_2 \oplus_{K} u_2 c_1, c_1 c_2]
\]

\[
0_{K_{UA}} = [0_K, 0_K]
\]

\[
1_{K_{UA}} = [1_K, 1_K]
\]

Before introducing how to derive a \( UA-DB \) as an uncertainty labeling for a possible world, we first establish that for any semiring \( K \), the \( UA \)-semiring \( K_{UA} \) is also a semiring. We will use this fact to utilize standard \( K \)-relational query semantics for \( UA-DBs \) and to prove that this semantics preserves c-soundness of the labeling encoded by a \( UA-DB \).
$$k_1 \oplus \kappa_{UA}(0,0) = [u_1 \oplus \kappa 0, c_1 \oplus \kappa 0] = k_1$$

$$k_1 \oplus \kappa_{UA}(0,1) = [u_1 \oplus \kappa 0, c_1 \oplus \kappa c_1] = k_1$$

**Annihilation by zero:**

$$k_1 \oplus \kappa_{UA}(0,0) = [u_1 \oplus \kappa 0, c_1 \oplus \kappa 0] = 0_{\kappa_{UA}}$$

**Distributivity:**

$$k_1 \oplus \kappa_{UA}(k_2 \oplus \kappa_{UA}k_3) = (k_1 \oplus \kappa_{UA}k_2) \oplus \kappa_{UA}k_3$$

**Proof.**

Given a UA-DB \(D_{UA}\) derived from a possible world \(D\), we would like to be able to restore \(D\) from \(D_{UA}\). Similarly, we would like to be able to recover the labeling \(L\) we started from. If this is possible then we can confidently claim that \(D_{UA}\) encodes the possible world \(D\) and labeling \(L\). Furthermore, we would like to determine which part of an annotation is uncertain. For that we define three morphisms \(K^2 \rightarrow K\) which for any \([u, c] \in K^2\) are defined as:

$$h_{uncert}[u, c] = u \quad h_{cert}[u, c] = c \quad h_{det}[u, c] = u \oplus \kappa c$$

We say a UA-DB \(D_{UA}\) is an encoding of a possible world \(D\) and a labeling \(L\) of a \(K\)-database \(D\) if

$$h_{det}(D_{UA}) = D \quad \forall t : h_{cert}(D_{UA}(t)) = cert(L, t)$$

A UA-DB created from a possible world and uncertainty labeling \(L\) can be treated as an uncertainty labeling itself by defining \(cert_{UA}\) as follows:

$$cert_{UA}(D_{UA}, t) = h_{cert}(D_{UA}(t))$$

The definitions above allow us to create a UA-DB from a possible world and uncertainty labeling. We create uncertainty labelings using the labeling schemes introduced in Section 5. Deriving some possible world is trivial for most incomplete and probabilistic data models. However, for the case of probabilistic data models we are particularly interested in the highest-probability world (the best guess world).

**TI-DB.** For TI-DB the best guess world is the set of tuples from \(D\) with probability larger than or equal to 0.5. To understand why this is the case recall that the probability of a possible world TI-DB database is the product of the probabilities of included tuples with one minus the probability of excluded tuples. This probability can be maximized by including only tuples where \(P(t) \geq 0.5\).

**PC-tables.** For a PC-table, computing the most likely possible world reduces to answering a query over the database, which is known to be \#P in general. Specific tables (e.g., those generated by “safe” queries) admit PTIME solutions. Alternatively, there exist a wide range of algorithms that can be used to compute an arbitrarily close approximation of the most likely world.

### 7.3 Preservation of C-soundness and Worlds

In this section we prove that query evaluation over UA-DBs that are uncertainty labelings preserves c-soundness and preserves encodings of possible worlds.

**Theorem 5.** (Preservation of Encodings). Let \(D_{UA}\) be a UA-database that is an encoding for a possible world \(D\) from a \(K\)-database \(D\). Then \(Q(D_{UA})\) is an encoding for \(Q(D)\), i.e.,

$$h_{det}(Q(D_{UA})) = Q(h_{det}(D_{UA}))$$

**Proof.** Note that \(h_{det}(Q(D_{UA})) = Q(h_{det}(D_{UA}))\) holds if \(h_{det}\) is a homomorphism. Thus, it suffices to show that \(h_{det}\) is a homomorphism. Let \(k_1 = (u_1, c_1)\) and \(k_2 = (u_2, c_2)\).

$$h_{det}(0_{\kappa_{UA}}) = (0_{\kappa} \oplus \kappa 0_{\kappa}) = 0_{\kappa}$$

$$h_{det}(1_{\kappa_{UA}}) = (0_{\kappa} \oplus \kappa 0_{\kappa}) = 1_{\kappa}$$

$$h_{det}(k_1 \oplus \kappa_{UA}k_2) = h_{det}([u_1 \oplus \kappa u_2, c_1 \oplus \kappa c_2]) = u_1 \oplus \kappa u_2 \oplus \kappa c_1 \oplus \kappa c_2$$

$$= h_{det}(k_1 \oplus \kappa_{UA}h_{det}(k_2))$$

Recall that a query semantics for labelings preserves c-soundness if for any probabilistic database \(D\) and one of its possible worlds \(D\) and a c-sound labeling \(L\) for \(D\), the output any query \(Q\) is a c-sound labeling for \(Q(D)\).
Proof. Mapping \( h_{cert}(D_{UA}) \) derives a simple labeling from a \( D_{UA} \) by extracting the uncertainty labeling that was encoded in the second component of tuple annotations in \( D_{UA} \). Note that \( h_{cert} \) is a homomorphism from \( K_{UA} \) to \( K_A \), and, thus, computes with queries. Now we have \( h_{cert}(Q(D_{UA})) = Q(h_{cert}(D_{UA})) \), i.e., the uncertainty labeling generated by evaluating \( Q \) over \( D_{UA} \) is equivalent to the result of evaluating the query over the simple labeling corresponding to \( D_{UA} \). Recall that we have already demonstrated that \( RA^+ \) queries over simple labelings preserve c-soundness. Hence, \( RA^+ \) over UA-DBs preserves c-soundness. \( \Box \)

8. IMPLEMENTATION

The direct implementation of UA-DBs inside a DBMS would require extensions in the database kernel to support annotated relations and queries with UA-DB semantics. We opt instead to define a bag semantics encoding of \( N_{UA} \)-relations to be able to store such relations in a standard DBMS. Recall that \( N_{UA} \) is the UA-semiring corresponding to the bag semantics semiring \( N \). UA-DB query semantics is then realized over this encoding through a set of rewriting rules which translate a query with \( N_{UA} \)-semantics into a bag semantics relational query.

Our encoding represents a tuple \( t = (a_1, \ldots, a_n) \) from a relation \( R(A_1, \ldots, A_n) \) annotated with a \( N_{UA} \)-element \([u, c]\) as a bag of \( u + c \) tuples over a schema \( R'(A_1, \ldots, A_n, U) \). The boolean attribute \( U \) is used to mark a tuple as uncertain (0) or certain (1). The bag of tuples representing \( t \) annotated with \([u, c]\) will contain \( u \) duplicates of a tuple \((a_1, \ldots, a_n, 0)\) and \( c \) duplicates of a tuple \((a_1, \ldots, a_n, 1)\). In the introduction we already gave an example for such an encoding in Figure 2c (using uncertain and certain instead of 0 and 1). In the following, we use \( t \mapsto k \) to denote a singleton relation where tuple \( t \) is annotated with \( k \) and all other tuples are annotated with 0. Recall that \( arity(R) \) denotes the arity (number of attributes) of a relation.

Definition 6 (Multiset encoding). \( ENC(R) \) is a function from \( N_{UA} \)-relations to \( N \)-relations. Let \( R \) be a \( N_{UA} \)-relation with schema \( A_1, \ldots, A_n \). Let \( R' \) be an \( N \)-relation with schema \( A_1, \ldots, A_n, U \) that is the result of \( ENC(R) \) for some \( R \). \( ENC \) and its inverse are defined as:

\[
ENC(R) = \bigcup_{t \in \text{Derrity}(R)} \{(t, 1) \mapsto \text{cert}_{UA}(R, t)\} \\
\cup \{(t, 0) \mapsto \text{huncert}(R(t))\}
\]

\[
ENC^{-1}(R') = \bigcup_{t \in \text{Derrity}(R)} t \mapsto (R'(t, 0), R'(t, 1))
\]

We define \( ENC \) over databases as applying \( ENC \) to every relation in the database. Next, we define a rewriting \( uRewr \) that translates an input query into a query over the encoding produced by \( ENC \) which faithfully simulates the semantics of \( N_{UA} \)-relational queries over this encoding. The rewriting \( uRewr \) is defined through a set of rules (one per relational algebra operator). Here \( D(Q) \) denotes the schema of the result of query \( Q \) and \( e \rightarrow a \) used in generalized projection expressions denotes projecting on the result of evaluating expression \( e \) and calling the resulting attribute \( a \).

\[
uRewr(R) = R
\]

\[
uRewr(\sigma_\theta(Q)) = \sigma_\theta(uRewr(Q))
\]

\[
uRewr(\pi_A(Q)) = \pi_A(uRewr(Q))
\]

\[
uRewr(Q_1 \bullet_\varphi Q_2) = \pi_{\text{Scm}(Q_1 \bullet_\varphi Q_2, \text{min}(Q_1, U, Q_2, U)} \rightarrow_U (uRewr(Q_1) \bullet_\varphi uRewr(Q_2))
\]

\[
uRewr(Q_1 \cup Q_2) = uRewr(Q_1) \cup uRewr(Q_2)
\]

We implement our approach as a middleware over a database system through an extension of SQL. An input query is first parsed, translated into a relational algebra graph, rewritten using \( uRewr \), and then converted back to SQL for execution.

Theorem 7. Let \( D_{UA} \) be a \( N_{UA} \)-database and \( Q \) an \( RA^+ \) query. Let \( Q_{uRewr} = uRewr(Q) \). The following holds:

\[
Q(D_{UA}) = ENC^{-1}(Q_{uRewr}(ENC(D_{UA})))
\]

Proof. Straightforward induction over the structure of queries.

Base case: \( Q = R \). Wlog consider a tuple \( t \) and let \( R(t) = [u, c] \). We know that

\[
ENC(R)(t, 0) = \text{huncert}(R(t)) = \text{huncert}([u, c]) = u
\]

and

\[
ENC(R)(t, 1) = \text{cert}_{UA}(R, t) = c
\]

Let \( R' = ENC^{-1}(Q_{uRewr}(ENC(R))) = ENC^{-1}(ENC(R)) \).

Then \( R'(t) = [R(t, 0), R(t, 1)] = [u, c] \).

Induction Step: Assume that the claim holds for queries \( Q_1 \) and \( Q_2 \), we have to show that it also holds for applying an operator of \( RA^+ \) to the result of these queries. We use \( Q_{uRewr} = uRewr(Q) \) and \( R_1 = Q_1(D_{UA}) \).

Selection \( \sigma_\theta(Q_1) \): Note that \( uRewr(\sigma_\theta(query)) = \sigma_\theta(uRewr(Q_1)) \). Consider a tuple \( t \) with \( R_1(t) = [u, c] \). Let \( R_1' = ENC^{-1}(\sigma_\theta(ENC(R_1))) \).

We have \( \sigma_\theta(ENC(R_1))(t) = R_1(t) \otimes_{N_{UA}} \theta(t) \) and

\[
\sigma_\theta(ENC(R_1)(t, 0)) = u \cdot \theta(t, 0)
\]

\[
\sigma_\theta(ENC(R_1)(t, 1)) = c \cdot \theta(t, 1)
\]

Since the selection condition does not access attribute \( U \), we have

\[
\theta(t, 0) = \theta(t, 1) = 1 \iff \theta(t) = [0, 1]
\]

Applying the definition of \( ENC^{-1} \), we get

\[
R_1'(t) = [u \cdot \theta(t, 0), c \cdot \theta(t, 1)]
\]

We now distinguish two cases: either \( t \models \theta \) and \( t \nmid \theta \). First consider the case where \( t \models \theta \). Then, \( \theta(t, 0) = \theta(t, 1) = 1 \) and we get

\[
R_1'(t) = [u \cdot 1, c \cdot 1] = [u, c] = R_1(t) = R_1(t) \otimes_{N_{UA}} \theta(t)
\]

Now consider the case \( t \nmid \theta \). Then, \( \theta(t, 0) = \theta(t, 1) = 0 \) and we get
Let $Q$ and $t$ be a tuple and consider a tuple $t$. Then, $t = t[R_1] = t[R_2] = t[R_3]$.

Consider $Q$ and $Q'$ as representations of incompleteness that are closed under full relational algebra. Reiter [40] extended the relational model with null values to represent missing information and proposed to use 3-valued logic to evaluate queries over databases with null values. Imielinski [25] introduced V-tables and C-tables as representations of incompleteness, which can represent not just uncertainty, but also model inconsistencies (it is neither c-sound nor c-complete). They propose certain answers semantics and proposed to use 3-valued logic to evaluate queries with null values can produce both false negatives and false positives. Green et al. [21] studied probabilistic versions of C-tables, relating the terms c-soundness and c-completeness to individual possible worlds. TI-DBs [43] are a prevalent model for probabilistic data where each tuple is associated with its marginal probability. Green et al. [21] studied probabilistic versions of C-tables, relating them to [41]. Virtual C-tables generalize C-tables [24, 47] by allowing symbolic expressions as values.

9. RELATED WORK

Incomplete and probabilistic data models. Uncertainty has been recognized as an important problem by the database community early on. Codd [10] extended the relational model with null values to represent missing information and proposed to use 3-valued logic to evaluate queries over databases with null values. Imielinski [25] introduced V-tables and C-tables as representations of incompleteness. However, computing certain answers is coNP-complete [2, 29] (data complexity) for first order queries, even for restricted data models such as Codd-tables. Thus, it is not surprising that approaches for estimating the set of certain answers have been proposed. Reiter [40] proposed a PTIME algorithm that returns a subset of the certain answers (c-sound) for positive existential queries (and a limited form of universal queries). Guagliardo and Libkin [23, 35, 22] demonstrated that SQL’s semantics for dealing with null values can produce both false negatives and false positives (it is neither c-sound nor c-complete). They propose an alternative evaluation scheme that preserves c-soundness for full relational algebra (first order queries) over Codd and V-tables. [22] defined certain and possible multiplicities for tuples under bag semantics and presents some initial thoughts on how extend the approach from [22] for bag semantics. Sundaramurthy et al. [44] introduced m-tables which can represent not just uncertainty, but also model information about missing tuples. This approach works for both set and bag semantics. The terms c-soundness and c-correctness which we utilize here were coined in this work. Apart from [44] and [22], to the best of our knowledge we are the first to generalize certain answers and incomplete databases beyond set semantics. In fact, we generalize these
concepts to a large class of semirings, namely, all l-semirings (semirings where the natural order forms a lattice structure). Additionally, what distinguishes our approach from prior work is that we split the computation of certain answers into two subproblems: 1) creating an uncertainty labeling from an incomplete database and 2) evaluating queries over uncertainty labelings. This has the distinct advantage, that we can apply our techniques to a wide-range of incomplete and probabilistic data models.

Annotated Databases. Green et al. [19] introduced the semiring annotation framework that we utilize in this work and demonstrated how extensions of the relational data model can be modelled as $K$-relations. This initial work also defined positive relational algebra ($RA^+$) over $K$-relations using the addition and multiplication operations of semirings. The connection between annotated databases, provenance, and uncertainty has been recognized early-on. A particular type of semiring annotations, called Lineage in the probabilistic database literature, have been used extensively for probabilistic query processing (e.g., see [43, 42]). Green et al. [19] observed that set semantics incomplete databases can be expressed as $K$-relations by annotating each tuple with the set of possible worlds that contain this tuple. We define a more general type of incomplete databases based on $K$-relations which is defined for any l-semiring. Geerts et al. [18] used the standard construction of a monus operation for naturally ordered semiring defining the relational algebra difference operator for $K$-relations. Kostylev et al. [30] investigate how to deal with dependencies among annotations from multiple domains. Similar to [30] we consider “multi-dimensional” annotations and similar to [18] we utilize the concept of the natural order and monus, however, for a very different purpose: to extend the concept of certain answers to $K_\mu$-relations and to define an annotation structure that encodes both a possible world and uncertainty labeling.

10. EXPERIMENTS
We evaluate the performance of query evaluation over UA-DBs on a commercial DBMS for which the licensing restrictions prevent us from identifying the system here. We compare against deterministic query processing and state-of-the-art PQP methods (MayBMS and MCDB). All experiments are run on a machine with 2 x 6 cores AMD Opteron 4238 CPUs, 128GB RAM, 4 x 1TB 7.2K HDs (RAID 5). We report the average running time of 5 runs. Furthermore, we also evaluate the impact of c-incompleteness by measuring the false negative rate, i.e., the fraction of certain answers that are misclassified as uncertain by our approach.

10.1 Synthetic dataset
We use PDBench [5] which uses a modified TPC-H data generator [11] that introduces uncertainty by generating random possible values for randomly selected cells (attribute values). The generator produces a columnar encoding of the TPC-H tables as pairs of tuple identifiers and attribute values. An uncertain cell is represented by having multiple attribute values. We run deterministically queries and queries generated by our approach on one possible world selected by randomly choosing a value for each uncertain cell and mark any tuple with a cell with more than one possible value as uncertain. The three PDBench queries used here do roughly correspond to TPC-H queries 3, 6 and 7.

Amount of uncertainty. To evaluate scalability as a function of the amount of uncertainty in the data, we use a scale factor 1 database (~1GB of data per possible world) and generate versions of this dataset with an uncertainty percentage of 2%, 5%, 10% and 30% (percentage of cells that are uncertain). Each cell has up to 8 possible values. Figure 6 shows the runtime results for PDBench queries 1, 2 and 3. We observe that the runtime for UA-DBs is close to deterministic query processing as expected. The slight overhead of UA-DBs is due to propagating uncertainty annotations. MCDB runs more than 10 times slower than deterministic query processing with sample size of 10 on our testing queries. Again this is expected, since we essentially have to evaluate each query multiple times. MayBMS has reasonable but still noticeable overhead for lower amounts of uncertainty. For larger uncertainty percentages, the overhead over UA-DBs can be several orders of magnitude (notice the logarithmic scale of the figure), especially for Q1 and Q3 which involve expensive joins. MayBMS performs better for Q2, because Q2 is a simple selection query.

To better understand the performance of MayBMS, we show result sizes (number of tuples) for each query and varying amounts of uncertainty in Figure 7. Our approach produces the same number of results as deterministic processing if applied to the same possible world. MayBMS returns the full set of possible answers and, thus its results size increases dramatically (combinatorial explosion) with increasing amount of uncertainty leading to its poor scalability in the amount of uncertainty. We also show the percentage of certain answers for each query per input uncertainty level in Figure 8. PDBench’s uncertainty generation seems to affect the correlation among values of attributes, because the number of tuples that fulfill the selection condition of Q1 increases dramatically when increasing the amount of uncertainty. We verified that this was due to uncertain values being chosen independently of each other which affects the correlation between attributes $o$_orderkey and $l$_shipdate. This in turn makes it much more likely for tuples with uncertain values to fulfill the selection condition of query Q1 than tuples generated by vanilla TPC-H.

Dataset size. To evaluate scalability in terms of data size, we use datasets with scale factors (SF) 0.1 (100MB), 1 (1GB) and 10 (10GB) and fix the uncertainty percentage (2%). Figure 9 shows the results for PDBench queries 1, 2 and 3. Again UA-DBs exhibit performance similar to deterministic queries and MCDB is again roughly 10 times slower (we again used a sample size of 10). MayBMS’s relative overhead over deterministic processing increases with data set size. For instance, for Q1 the overhead is $\sim 60\%$ for SF 0.1 and $\sim 500\%$ for SF 10.

Incompleteness. We evaluate the incompleteness of query results over UA-DBs by measuring the false negative rate, i.e., the fraction of certain answers we incorrectly classify as uncertain. The three PDBench queries used here do roughly correspond to TPC-H queries 3, 6 and 7.

Figure 10 shows the rate of queries over UA-DBs. Figure 10a shows the mean false
Figure 6: Performance of PDBench queries - varying the amount of uncertainty

<table>
<thead>
<tr>
<th></th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>UA-DB</th>
<th>MayBMS</th>
<th>MCDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>5%</td>
<td>34.041</td>
<td>152.432</td>
<td>8.619</td>
<td>501.114</td>
<td>327.052</td>
<td>32.438</td>
</tr>
<tr>
<td>10%</td>
<td>61.800</td>
<td>152.389</td>
<td>8.794</td>
<td>2,392,916</td>
<td>618,199</td>
<td>97,454</td>
</tr>
<tr>
<td>30%</td>
<td>130.581</td>
<td>152.885</td>
<td>7.994</td>
<td>134,054,635</td>
<td>3,941,554</td>
<td>4,351,782</td>
</tr>
</tbody>
</table>

Figure 7: Query result sizes (#rows)

<table>
<thead>
<tr>
<th></th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>UA-DB</th>
<th>MayBMS</th>
<th>MCDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>2%</td>
<td>0 (0%)</td>
<td>143,618 (94%)</td>
<td>7,861 (87%)</td>
<td>113,966</td>
<td>210.996</td>
<td>15.108</td>
</tr>
<tr>
<td>5%</td>
<td>1 (0%)</td>
<td>130,594 (86%)</td>
<td>6,023 (70%)</td>
<td>501.114</td>
<td>327.052</td>
<td>32.438</td>
</tr>
<tr>
<td>10%</td>
<td>4 (0%)</td>
<td>111,120 (73%)</td>
<td>3,979 (45%)</td>
<td>2,392,916</td>
<td>618,199</td>
<td>97,454</td>
</tr>
<tr>
<td>30%</td>
<td>1 (0%)</td>
<td>52,724 (34%)</td>
<td>586 (7%)</td>
<td>134,054,635</td>
<td>3,941,554</td>
<td>4,351,782</td>
</tr>
</tbody>
</table>

Figure 8: Result certain answer %

<table>
<thead>
<tr>
<th></th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>UA-DB</th>
<th>MayBMS</th>
<th>MCDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>2%</td>
<td>0 (0%)</td>
<td>143,618 (94%)</td>
<td>7,861 (87%)</td>
<td>113,966</td>
<td>210.996</td>
<td>15.108</td>
</tr>
<tr>
<td>5%</td>
<td>1 (0%)</td>
<td>130,594 (86%)</td>
<td>6,023 (70%)</td>
<td>501.114</td>
<td>327.052</td>
<td>32.438</td>
</tr>
<tr>
<td>10%</td>
<td>4 (0%)</td>
<td>111,120 (73%)</td>
<td>3,979 (45%)</td>
<td>2,392,916</td>
<td>618,199</td>
<td>97,454</td>
</tr>
<tr>
<td>30%</td>
<td>1 (0%)</td>
<td>52,724 (34%)</td>
<td>586 (7%)</td>
<td>134,054,635</td>
<td>3,941,554</td>
<td>4,351,782</td>
</tr>
</tbody>
</table>

Figure 9: Performance of PDBench queries - varying database size

Figure 10: Measuring incompleteness as the fraction of certain answers that were misclassified as uncertain
negative rate versus the number of attributes projected for uncertainty percentages 2%, 5%, 10% and 30%. Since a tuple is labeled uncertain if any of its attributes are uncertain, the fraction of tuples labeled as uncertain is already more than 70% for 10% attribute-level uncertainty as the cells which are made uncertain are chosen uniformly random by PDBench. Note that our approach misclassifies a tuple as uncertain for such projection queries, if the attributes of the tuple which are uncertain (differ across possible worlds) are projected away. Thus, the false negative rate decreases as more attributes are used in the projection since projecting on more attributes is likely to return tuples with uncertain values (which we correctly classify as uncertain). When the uncertainty percentage is high (30%), the false negative rate actually decreases at a faster rate in the number of attributes than for lower uncertainty percentages. This is because the per-column attribute uncertainty is high enough so that it is more likely that a tuples in the query result contains uncertain values (there are less certain answers in general).

10.2 Real world dataset

The false negative rate for the synthetic dataset is quite high, because errors (uncertain attributes) are independent of each other which is not realistic. We use a dataset storing shootings recorded by Buffalo news [21] to evaluate how our approach performs for real world data. The data consists of ~3,000 rows and 25 columns and contains NULLs and other data quality problems. We introduce a worst case scenario involving test cases that creates false negatives. We use Mimir [42] to clean the Buffalo news dataset using data cleaning techniques that introduce attribute-level uncertainty. Since currently we only support tuple-level uncertainty, we model attribute-level uncertainty as tuple uncertainty by creating tuples which are correlated (exclusive) across possible worlds. Our labeling schemes mark such tuples as uncertain (~2% of tuples were marked as uncertain).

A projection over certain attributes of those tuples produces a misclassified certain result. We evaluate queries which project on a randomly chosen set of attributes and measure the false negative rate. Figure 10a shows the distribution of the false negative rate for queries with a fixed number of projection attributes (10 random projections for each). As expected, the false negative rate decreases with increasing number of projection attributes. However, in contrast to the result over the synthetic dataset, the false negative rate is quite low in general (less than 3% in the worst case). This demonstrates that, while in the worst case the false positive rate can be quite high, for real world datasets with correlated errors it is quite low.

11. CONCLUSIONS AND FUTURE WORK

We propose UA-DBs as a novel way to represent uncertainty as annotations on the tuples of a possible world. Being based on K-relations, our approach applies to incomplete versions of any type of data model that can be encoded as K-relations including set and bag semantics. We achieve efficient query evaluation over UA-DBs by sacrificing completeness (we may mark certain tuples as uncertain), but not soundness (we never mark uncertain tuples as certain). In future work, we plan to extend our approach with attribute level annotations to encode certainty at finer granularity and to support larger classes of queries, e.g., queries involving negation and aggregation.

12. REFERENCES

2007.


[29] Oliver Kennedy. Stop the truthiness and just be wrong. CIDR, 2017.


[45] USA Federal Trade Commission. In FTC study, five percent of consumers had errors on their credit reports that could result in less favorable terms for loans. https://www.ftc.gov/news-events/press-releases/2013/02/ftc-study-five-percent-consumers-had-errors-their-credit-reports.
