Uncertainty Annotated Databases - A Lightweight Approach for Approximating Certain Answers

Su Feng
Illinois Institute of Technology
sfeng14@hawk.iit.edu

Aaron Huber
University at Buffalo
ahuber@buffalo.edu

Boris Glavic
Illinois Institute of Technology
bglavic@iit.edu

Oliver Kennedy
University at Buffalo
okennedy@buffalo.edu

ABSTRACT

Certain answers are a principled method for coping with uncertainty that arises in many practical data management tasks. Unfortunately, this method is expensive and may exclude useful (if uncertain) answers. Thus, users frequently resort to less principled approaches to resolve uncertainty. In this paper, we propose *Uncertainty Annotated Databases* (UA-DBs), which combine an under- and over-approximation of certain answers to achieve the reliability of certain answers, with the performance of a classical database system. Furthermore, in contrast to prior work on certain answers, UA-DBs achieve a higher utility by including some (explicitly marked) answers that are not certain. UA-DBs are based on incomplete K-relations, which we introduce to generalize the classical set-based notion of incomplete databases and certain answers to a much larger class of data models. Using an implementation of our approach, we demonstrate experimentally that it efficiently produces tight approximations of certain answers that are of high utility.

KEYWORDS

uncertain data, incomplete data, annotations

ACM Reference Format:
Su Feng, Aaron Huber, Boris Glavic, and Oliver Kennedy. 2019. Uncertainty Annotated Databases - A Lightweight Approach for Approximating Certain Answers. In *2019 International Conference on Management of Data (SIGMOD '19), June 30-July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3299869.3319887

SIGMOD '19, June 30-July 5, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-5643-5/19/06…$15.00
https://doi.org/10.1145/3299869.3319887

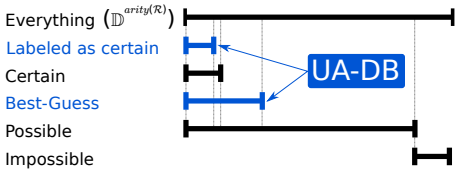


Figure 1: UA-DBs provide both an under- and over-approximation of certain answers.

1 INTRODUCTION

Data uncertainty arises naturally in applications like sensing [34], data exchange [13], distributed computing [33], data cleaning [10], and many others. Incomplete [25] and probabilistic databases [41] have emerged as a principled way to deal with uncertainty. Both types of databases consist of a set of deterministic instances called *possible worlds* that represent possible interpretations of data available about the real world. An often cited, conservative approach to uncertainty is to consider only *certain answers* [1, 25] (answers in all possible worlds). However, this approach has two problems. First, computing certain answers is expensive[1]. Furthermore, requiring answers to be certain may unnecessarily exclude useful, possible answers. Thus, users instead resort to what we term *best-guess query processing* (*BGQP*): make an educated guess about which possible world to use (i.e., how to interpret available data) and then work exclusively with this world. BGQP is more efficient than certain answers, and generally includes more useful results. However, information about uncertainty in the data is lost, and all query results produced by BGQP are consequently suspect.

Previous work has also explored approximations of certain answers [19, 36, 39]. Under the premise that missing a certain answer is better than incorrectly reporting an answer as certain, such work focuses on under-approximating certain answers. This addresses the performance problem, but under-approximations only exacerbate the problem of excluded results. Worse, these techniques are limited to specific uncertain data models such as V-tables, and with the exception of a brief discussion in [24], only support set semantics.

[1] coNP-complete [1, 25] (data complexity) for first-order queries over V-tables [25], as well as for conjunctive queries for, e.g., OR-databases [26].

ADDR

id	address	geocoded
1	51 Comstock	(42.93, -78.81)
2	Grant at Ferguson	(42.91, -78.89) **or** (32.25, -110.87)
3	499 Woodlawn	(42.91, -78.84) **or** (42.90, -78.85)
4	192 Davidson	(42.93, -78.80)

LOC

locale	state	rect
Lasalle	NY	((42.93, -78.83), (42.95, -78.81))
Tucson	AZ	((31.99, -111.045), (32.32, -110.71))
Grant Ferry	NY	((42.91, -78.91), (42.92, -78.88))
Kingsley	NY	((42.90, -78.85), (42.91, -78.84))
Kensington	NY	((42.93, -78.81), (42.96, -78.78))

Figure 2: Input data for Example 1. Tuples 2 and 3 of Table ADDR have uncertain geocoded values.

id	locale	state
1	Lasalle	NY
2	Tucson	AZ
3	Kingsley	NY
4	Kensington	NY

(a) One Possible World

id	locale	state
1	Lasalle	NY
3	Kingsley	NY
4	Kensington	NY

(b) Certain Answers

id	locale	state
1	Lasalle	NY
2	Tucson	AZ
2	Grant Ferry	NY
3	Kingsley	NY
4	Kensington	NY

(c) Possible Answers

id	locale	state	Certain?
1	Lasalle	NY	*true*
2	Tucson	AZ	*false*
3	Kingsley	NY	*false*
4	Kensington	NY	*true*

(d) Uncertainty-Annotated Database

Figure 3: Examples of query results under different evaluation semantics over uncertain data.

Example 1. *Geocoders translate natural language descriptions of locations into coordinates (i.e., latitude and longitude). Consider the **ADDR** and **LOC** relations in Figure 2. Tuples 2 and 3 of **ADDR** each have an ambiguous geocoding. This is an x-table [2], a type of incomplete data model where each tuple may have multiple alternatives. Each possible world is defined by some combination of alternatives (e.g., **ADDR** encodes 4 possible worlds). An analyst might use a spatial join with a lookup table (**LOC**) to map coordinates to geographic regions. Figure 3a shows the result of the following query in one world.*

```
SELECT a.id, l.locale, l.state
FROM ADDR a, LOC l
WHERE contains(l.rect, a.geocoded)
```

The certain answers to this query (Figure 3b) are tuples that appear in the result, regardless of which world is queried. Figure 3c shows all possible answers that could be returned for some choice of geocodings. Note also that ambiguous answers (e.g., address 2) may not be certain, but may still be useful.

Ideally, we would like an approach that (1) generalizes to a wide range of data models, (2) is easy to use like BGQP, (3) is compatible with a wide range of probabilistic and incomplete data representations (e.g., tuple-independent databases [41], C-tables [25], and x-DBs [2]) and sources of uncertainty (e.g., inconsistent databases [5, 6, 17, 29, 30, 32], imputation of missing values, and more), and (4) is principled like certain answers. We address the generality requirement (1) by rethinking incomplete data management in terms of Green et. al.'s $\mathcal{K}$-database framework [21]. In this framework, each tuple is annotated with an value from a semiring $\mathcal{K}$. Choosing an appropriate semiring, $\mathcal{K}$-databases can encode a wide range of query processing semantics including classical set- and bag-semantics, as well as query processing with access control, provenance, and more. Our primary contribution here is to identify a natural, backwards-compatible generalization of certain answers to a broad class of $\mathcal{K}$-databases.

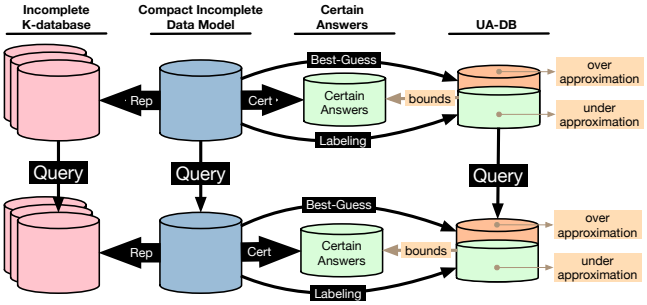


Figure 4: The relationship between UA-DBs, certain answers, and other incomplete data models

Our second major contribution is to combine an under-approximation of certain answers with best-guess query processing to create an *Uncertainty-Annotated Database* (*UA-DB*). A UA-DB is built around one distinguished possible world of an incomplete $\mathcal{K}$-database, for instance the "best-guess" world that would normally be used in practice. This world serves as an *over-approximation* of certain answers. Tuples from this world are labeled as either certain or uncertain to encode an *under-approximation* of certain answers. As illustrated in Figure 1, a UA-DB sandwiches the certain answers between under- and over-approximations. A lightweight (extensional [41]) query evaluation semantics then propagates labels while preserving the approximation's guarantees.

Example 2. *Continuing with Example 1, Figure 3d shows the result of the same query as a set UA-DB. When the UA-DB is built, one designated possible world of **ADDR** is selected, for example the highest ranked option provided by the geocoder. For this example, we select the first option for each ambiguous tuple. The result is based on this one designated possible world, which serves as an over-approximation of the certain answers. A subset of these tuples (addresses 1 and 4) are explicitly labeled as certain. This is the under-approximation: A tuple might still be certain even if it is not labeled as such. We consider the remaining tuples to be "uncertain". In Figure 3d, tuples 1 and 4 (resp., 2) are correctly marked as certain (resp., uncertain), while*

tuple 3 is mis-classified as uncertain even though it appears in all worlds. We stress that even a mislabeled certain answer is still present: a UA-DB sandwiches the certain answers.

Figure 4 overviews our approach. We provide *labeling schemes* that derive a UA-DB from common incomplete data models. The resulting UA-DB bounds the certain tuples from above and below, a property preserved through queries. UA-DBs are both efficient and precise. We demonstrate efficiency by implementing a bag UA-DB as a query-rewriting front-end on top of a classical relational DBMS: UA-DB queries have minimal performance overhead compared to the same queries on deterministic data. We demonstrate precision both analytically and experimentally. First, under specific conditions, some of which we identify in Appendix B, exactly the certain answers will be marked as certain. Second, we show experimentally that even when these conditions do not hold, the fraction of misclassified certain answers is low. Importantly, a wide range of uncertain data models can be translated into UA-DBs through simple and efficient transformations that (i) determine a best-guess world (*BGW*) and (ii) obtain an under-approximation of the certain answers. We define such transformations for three popular models of incomplete data in Section 6: tuple-independent databases [41], x-DBs [2] and C-tables [25]. In classical incomplete databases, where probabilities are not available, any possible world can serve as a BGW. In probabilistic databases (or any incomplete data model that ranks possible worlds), we preferentially use the possible world with the highest probability (if computationally feasible), or an approximation thereof. We emphasize that our approach does not require enumerating (or even knowing) the full set of possible worlds. As long as some possible world can be obtained, our approach is applicable. In worst case, if no certainty information is available, our approach labels all tuples as uncertain and degrades to classical best-guess query processing. Furthermore, our approach is also applicable to use cases like inconsistent query answering [5] where possible worlds are defined declaratively (e.g., all repairs of an inconsistent database).

We significantly extend the state-of-the-art on under-approximating certain answers [19, 36, 39]: (1) we combine an under-approximation with best-guess query processing bounding certain answers from above and below; (2) we support sets, bags, and any other data model expressible as semiring annotations from a large class of semirings; (3) we support translation of a wide range of incomplete and probabilistic data models into our UA-DB model; (4) in contrast to certain answers, UA-DBs are closed under queries.

The remainder of the paper is organized as follows.

Incomplete $\mathcal{K}$-Relations. (Section 3) We introduce incomplete $\mathcal{K}$-databases, generalizing incomplete databases to $\mathcal{K}$-relations [21]. We then define certain annotations as a natural extension of certain answers, based on the observation that certain answers are a lower bound on the content of a world. We show that certain annotations correspond to the classical notion of certain answers for set [37] and bag [24] semantics.

UA-DBs. (Section 4) We define *UA-DBs* as databases that annotate tuples with pairs of annotations from a semiring $\mathcal{K}$. The annotation of a tuple in a UA-DB bounds the certain annotation of the tuple from above and below. This is achieved by combining the annotations from one world (the over-approximation) with an under-approximation that we call an *uncertainty labeling*. Relying on results for under-approximations that we develop in the following sections, we prove that queries over UA-DBs preserve these bounds.

Under-approximating Certain Answers. (Section 5) To better understand under-approximations, we define *uncertainty labelings*, which are $\mathcal{K}$-relations that under-approximate the set of certain tuples for an incomplete $\mathcal{K}$-database.

Queries over Uncertainty Labelings. (Section 7) Since labelings are $\mathcal{K}$-relations, we can evaluate queries over such labelings. We demonstrate that evaluating queries in this fashion preserves under-approximations of certain answers, generalizing a previous result for V-tables due to Reiter [39].

Implementation for Bag Semantics. (Section 8) We implement UA-DBs on top of a relational DBMS. We extend the schema of relations to label tuples as certain or uncertain (e.g., Figure 3d). Queries with UA-relational semantics are compiled into standard relational queries over this encoding.

Performance. (Section 10) We demonstrate experimentally that UA-DBs outperform state-of-the-art incomplete and probabilistic query processing schemes, are competitive with deterministic query evaluation and other methods for under-approximating certain answers, and are also compatible with non-set semantics. We then show that our approximation is accurate for many real world datasets. Finally, we also demonstrate that best-guess answers (and hence UA-DBs), can have higher utility than certain answers.

2 NOTATION AND BACKGROUND

A database schema $\mathbf{D} = \{\mathbf{R}_1, \ldots, \mathbf{R}_n\}$ is a set of relation schemas. A relational schema $\mathbf{R}(A_1, \ldots, A_n)$ is a relation name and a set of attribute names $A_1, \ldots, A_n$. The arity $arity(\mathbf{R})$ of a relation schema $\mathbf{R}$ is the number of attributes in $\mathbf{R}$. An instance D for database schema $\mathbf{D}$ is a set of relation instances with one relation for each relation schema in $\mathbf{D}$: $D = \{R_1, \ldots, R_n\}$. Assume a universal domain of attribute values $\mathbb{D}$. A tuple with schema $\mathbf{R}$ is an element from $\mathbb{D}^{arity(\mathbf{R})}$. In this paper, we consider both bag and set semantics. A set (resp., bag) relation R with schema $\mathbf{R}$ is a set (resp., bag) of tuples with schema $\mathbf{R}$. That is, for a set, $R \subseteq \mathbb{D}^{arity(\mathbf{R})}$. We use TupDom to denote the set of all tuples over domain $\mathbb{D}$.

id	locale	state
1	Lasalle	NY
2	**Tucson**	**AZ**
3	Kingsley	NY
4	Kensington	NY

(a) D_1

id	locale	state
1	Lasalle	NY
2	**Grant Ferry**	**NY**
3	Kingsley	NY
4	Kensington	NY

(b) D_2

Figure 5: Example incomplete database $\mathcal{D} = \{D_1, D_2\}$.

id	locale	state
1	Lasalle	NY
3	Kingsley	NY
4	Kensington	NY

(a) $Q_{NY}(D_1)$

id	locale	state
1	Lasalle	NY
2	Grant Ferry	NY
3	Kingsley	NY
4	Kensington	NY

(b) $Q_{NY}(D_2)$

Figure 6: The result of a query Q over an incomplete database $\mathcal{D}$ is the set of results in all worlds $D \in \mathcal{D}$.

2.1 Possible Worlds Semantics

Incomplete and probabilistic databases model uncertainty and its impact on query results. An *incomplete database* $\mathcal{D}$ is a set of deterministic database instances $D_1, \ldots, D_n$ of schema **D**, called *possible worlds*. We write $t \in D$ to denote that a tuple t appears in a specific possible world D.

Example 3. *Continuing Example 1, Figure 5 shows the two possible worlds in the result of the spatial join. Observe that some tuples (e.g.,* ⟨ 1, Lasalle, NY ⟩*) appear in all worlds. Such tuples are called* ***certain****. Tuples that appear in at least one possible world (e.g.,* ⟨ 2, Tuscon, AZ ⟩*) are called* ***possible****.*

Decades of research [2, 4, 7, 22, 25, 41] has explored query processing over incomplete databases. These techniques commonly adopt the *"possible worlds"* semantics: The result of evaluating a query Q over an incomplete database is the set of relations resulting from evaluating Q over each possible world individually using deterministic semantics.

$$Q(\mathcal{D}) := \{\, Q(D) \mid D \in \mathcal{D} \,\} \tag{1}$$

Example 4. $Q_{NY} := \sigma_{state='NY'}(\mathcal{D})$ *returns locations in NY State from the database $\mathcal{D}$ shown in Figure 5. The result of $Q_{NY}(\mathcal{D})$ is the set of worlds computed by evaluating Q_{NY} over each world of $\mathcal{D}$ as shown in Figure 6. Observe that the location with id 2 appears in $Q_{NY}(D_2)$, but not $Q_{NY}(D_1)$.*

2.2 Certain and Best-Guess Answers

An important goal of query processing over incomplete databases is to differentiate query results that are certain from ones that are merely possible. Formally, a tuple is certain if it appears in every possible world. [25, 37]:

$$certain(\mathcal{D}) := \{t \mid \forall D \in \mathcal{D} : t \in D\} \tag{2}$$

In contrast to [25], which studies certain answers to queries, we define certainty at the instance level. These approaches are equivalent since we can compute the certain answers of query Q over incomplete instance $\mathcal{D}$ as $certain(Q(\mathcal{D}))$.

id	address	ℓ	$\mathbb{N}$	$\mathbb{B}$
1	51 Co…	L_1	1	T
2	Grant …	L_2	1	T
3	499 W…	L_4	1	T

(a) Address

ℓ	locale	state	$\mathbb{N}$	$\mathbb{B}$
L_1	L…	NY	1	T
L_2	T…	AZ	1	T
L_3	G…	NY	1	T
L_4	K…	NY	1	T
L_5	W…	IL	1	T

(b) Neighborhood

state	$\mathbb{N}$	$\mathbb{B}$
NY	$2 = (1 \cdot 1) + (1 \cdot 1)$	$T = (T \wedge T) \vee (T \wedge T)$
AZ	$1 = (1 \cdot 1)$	$T = (T \wedge T)$
IL	$0 = (0 \cdot 1)$	$F = (F \wedge T)$

(c) Result of Q_a

Figure 7: $\mathbb{N}$- and $\mathbb{B}$-relation examples

Although computing certain answers is coNP-hard [1] in general, there exist PTIME under-approximations [23, 36, 39].

Best Guess Query Processing. As mentioned in the introduction, another approach commonly used in practice is to select one possible world. Queries are evaluated solely in this world, and ambiguity is ignored or documented outside of the database. We refer to this approach as *best-guess query processing* (BGQP) [45] since typically one would like to select the possible world that is deemed most likely.

2.3 K-relations

Our generalization of incomplete databases is based on the **$\mathcal{K}$-relation** [21] framework. In this framework, relations are annotated with elements from the domain K of a (commutative) semiring $\mathcal{K}$. A commutative semiring is a structure $\mathcal{K} = \langle K, \oplus_{\mathcal{K}}, \otimes_{\mathcal{K}}, \mathbb{1}_{\mathcal{K}}, \mathbb{0}_{\mathcal{K}} \rangle$ with commutative and associative addition ($\oplus_{\mathcal{K}}$) and product ($\otimes_{\mathcal{K}}$) operations where $\otimes_{\mathcal{K}}$ distributes over $\oplus_{\mathcal{K}}$. As before, $\mathbb{D}$ denotes a universal domain. An n-ary $\mathcal{K}$-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 $\mathbb{0}_{\mathcal{K}}$. Only finitely many tuples may be mapped to an element other than $\mathbb{0}_{\mathcal{K}}$ (i.e., relations must be finite). Since $\mathcal{K}$-relations are functions from tuples to annotations, it is customary to denote the annotation of a tuple t in relation R as $R(t)$. The specific information encoded by an annotation depends on the choice of semiring.

Encoding Sets and Bags. Green et al. [21] demonstrated that bag and set relations can be encoded as commutative semirings: the natural numbers ($\mathbb{N}$) with addition and multiplication, $\langle \mathbb{N}, +, \times, 0, 1 \rangle$, annotates each tuple with its multiplicity; and boolean constants $\mathbb{B} = \{T, F\}$ with disjunction and conjunction, $\langle \mathbb{B}, \vee, \wedge, F, T \rangle$, annotates each tuple with its set membership. Abusing notation, we denote by $\mathbb{N}$ and $\mathbb{B}$ both the domain and the corresponding semiring.

Query Semantics. Operators of the positive relational algebra ($\mathcal{RA}^+$) over $\mathcal{K}$-relations are defined by combining input annotations using operations $\oplus_{\mathcal{K}}$ and $\otimes_{\mathcal{K}}$.

$$\textbf{Union: } [R_1 \cup R_2](t) = R_1(t) \oplus_{\mathcal{K}} R_2(t)$$
$$\textbf{Join: } [R_1 \bowtie R_2](t) = R_1(t[\mathbf{R}_1]) \otimes_{\mathcal{K}} R_2(t[\mathbf{R}_2])$$
$$\textbf{Projection: } [\pi_U(R)](t) = \sum_{t=t'[U]} R(t')$$
$$\textbf{Selection: } [\sigma_\theta(R)](t) = R(t) \otimes_{\mathcal{K}} \theta(t)$$

Example 5. *Figure 7 shows a $\mathbb{N}$- (i.e., bag-) database, with each tuple t annotated with its multiplicity (the copies of t in the relation). Annotations appear beside each tuple. The query $Q_a := \pi_{state}$ (Address $\bowtie$ Neighborhood) computes states. Every input tuple appears once (is annotated with 1). The output tuple annotation is computed by multiplying annotations of joined tuples, and summing annotations projected onto the same result tuple. For instance, 2 NY addresses are returned.*

In the following, we will make use of homomorphisms. A mapping $h : \mathcal{K} \to \mathcal{K}'$ from a semiring $\mathcal{K}$ to a semiring $\mathcal{K}'$ is a called a **homomorphism** if it maps $\mathbb{0}_{\mathcal{K}}$ and $\mathbb{1}_{\mathcal{K}}$ to their counterparts in $\mathcal{K}'$ and distributes over sum and product (e.g., $h(k \oplus_{\mathcal{K}} k') = h(k) \oplus_{\mathcal{K}'} h(k')$). As observed by Green et al. [21], any semiring homomorphism h can be lifted to a homomorphism from $\mathcal{K}$-relations to $\mathcal{K}'$-relations by applying h to the annotation of every tuple t: $h(R)(t) = h(R(t))$. Importantly, queries commute with semiring homomorphisms. That is, given a homomorphism h, query Q, and $\mathcal{K}$-database D we have $h(Q(D)) = Q(h(D))$. We will abuse syntax and use the same function symbols (e.g., $h(\cdot)$) to denote mappings between semirings, $\mathcal{K}$-relations, as well as $\mathcal{K}$-databases.

Example 6. *Continuing Example 5, we can derive a set instance through a mapping $h : \mathbb{N} \to \mathbb{B}$ defined as $h(k) = T$ if $k > 0$ and $h(k) = F$ otherwise. h is a semiring homomorphism, so evaluating Q_a in $\mathbb{N}$ first and then applying h (i.e., $h(Q(D))$) is equivalent to applying h first, and then evaluating Q_a.*

When defining bounds for annotations in Section 3, we make use of the so called *natural order* $\preceq_{\mathcal{K}}$ for a semiring $\mathcal{K}$, defined as an element k preceding k' if it is possible to obtain k' by adding to k. Semirings for which the natural order is a partial order are called *naturally ordered* [20].

$$\forall k, k' \in K : \left(k \preceq_{\mathcal{K}} k'\right) \Leftrightarrow \left(\exists k'' \in K : k \oplus_{\mathcal{K}} k'' = k'\right) \quad (3)$$

3 INCOMPLETE K-RELATIONS

Many incomplete data models do not support bag semantics. Our first contribution unifies set and bag semantics under a joint framework. Recall that an incomplete database is a set of deterministic databases (possible worlds). We now generalize this idea to $\mathcal{K}$-databases.

Definition 1 (Incomplete $\mathcal{K}$-database). *Let $\mathcal{K}$ be a semiring. An* incomplete $\mathcal{K}$-database $\mathcal{D}$ *is a set of possible worlds $\mathcal{D} = \{D_1, \ldots, D_n\}$ where each D_i is a $\mathcal{K}$-database.*

Like classical incomplete databases, queries over an incomplete $\mathcal{K}$-database use possible world semantics, i.e., the result of evaluating a query Q over an incomplete $\mathcal{K}$-database $\mathcal{D}$ is the set of all possible worlds derived by evaluating Q over every possible world $D \in \mathcal{D}$ (i.e., $Q(\mathcal{D}) = \{Q(D_1), \ldots, Q(D_n)\}$).

3.1 Certain Annotations

While possible worlds semantics are directly compatible with incomplete $\mathcal{K}$-databases, the same does not hold for the concepts of certain tuples, as we will show in the following. First off, we have to define what precisely do we mean by certain answers over possible worlds that are $\mathcal{K}$-databases.

Example 7. *Consider a $\mathbb{N}$-database $\mathcal{D}$ (bag semantics) containing a relation **LOC** with two attributes **locale** and **state**. Assume that $\mathcal{D}$ consists of the two possible worlds below:*

***LOC** in D_1*

locale	***state***	$\mathbb{N}$
Lasalle	*NY*	***3***
Tucson	*AZ*	***2***

***LOC** in D_2*

locale	***state***	$\mathbb{N}$
Lasalle	*NY*	***2***
Tucson	*AZ*	***1***
Greenville	*IN*	***5***

Using semiring $\mathbb{N}$ each tuple in a possible world is annotated with its multiplicity (the number of copies of the tuple that exist in the possible world). Arguably, tuples `(Lasalle, NY)` *and* `(Tucson, AZ)` *are certain since they appear (multiplicity higher than* 0*) in both possible worlds while* `(Greenville, IN)` *is not since it is not present (its multiplicity is zero) in possible world D_1*[2]*. However, the boolean interpretation of certainty in incomplete databases is not suited to $\mathbb{N}$-relations (or $\mathcal{K}$-relations in general) because it ignores the annotations of tuples. In this particular example, tuple* `(Lasalle, NY)` *appears with multiplicity* 3 *in possible world D_1 and multiplicity* 2 *in possible world D_2. We can state with certainty that in every possible world this tuple appears* at least *twice. Thus,* 2 *is a lower bound (the greatest lower bound) for the annotation of* `(Lasalle, NY)`*. Following this logic, we will define certainty through greatest lower bounds (GLBs) on tuple annotations.*

To further justify defining certain answers as lower bounds on annotations, consider classical (i.e., set) incomplete databases. Here, a tuple is *certain* if it appears in all possible worlds. Like the bag semantics example above, certainty is a lower bound on a tuple's annotation across all worlds. Consider the the order $false < true$. If a tuple exists in every possible world (is always annotated *true*), then intuitively, the GLB of its annotation across all worlds is *true*. Otherwise, the tuple is not certain (is annotated *false* in at least one world), and the GLB is $false$.

To define a sensible lower bound for annotations, we need an order relation for semiring elements. We use the natural order $\preceq_{\mathcal{K}}$ as introduced in Section 2.3 to define the GLB of a set of $\mathcal{K}$-elements. For a well-defined GLB, we require that

[2] All tuples not shown in the tables are assumed to be annotated with zero.

$\preceq_{\mathcal{K}}$ forms a lattice over K, a property that makes $\mathcal{K}$ an *l-semiring* [31]. A lattice over a set S according to a partial order $\preceq_S$ over S is a structure $(S, \sqcup, \sqcap)$ where $\sqcap$ (the greatest lower bound) is an operation over S defined for all $a, b \in S$:

$$a \sqcap b \coloneqq \max_{\preceq_S}(\{c \mid c \in S \wedge c \preceq_S a \wedge c \preceq_S b\})$$

We will use $\sqcap_{\mathcal{K}}$ to denote the $\sqcap$ operation of the lattice over $\preceq_{\mathcal{K}}$ for a semiring $\mathcal{K}$. Abusing notation, we will apply the $\sqcap_{\mathcal{K}}$ operation iteratively to sets of elements. e.g., $\sqcap_{\mathcal{K}}\{k_1, k_2, k_2\} = (k_1 \sqcap_{\mathcal{K}} k_2) \sqcap_{\mathcal{K}} k_3$. From here on, we will limit our discussion to l-semirings. Many semirings, including the set semiring $\mathbb{B}$ and the bag semiring $\mathbb{N}$ are l-semirings. The natural order of $\mathbb{B}$ is $F \preceq_{\mathbb{B}} T$ and $k_1 \sqcap_{\mathbb{B}} k_2 = k_1 \wedge k_2$. The natural order of $\mathbb{N}$ is the standard order of natural numbers and $k_1 \sqcap_{\mathbb{N}} k_2 = \min(k_1, k_2)$.

We define the certain annotation $\textsc{cert}_{\mathcal{K}}(\mathcal{D}, t)$ of a tuple t in an incomplete $\mathcal{K}$-database $\mathcal{D}$ by gathering the annotations of tuple t from all possible worlds of $\mathcal{D}$ and then applying $\sqcap_{\mathcal{K}}$ to compute the greatest lower bound.

$$\textsc{cert}_{\mathcal{K}}(\mathcal{D}, t) \coloneqq \sqcap_{\mathcal{K}}(\{D(t) \mid D \in \mathcal{D}\})$$

Importantly, GLB coincides with the standard definition of certain answers for set semantics ($\mathbb{B}$): $\textsc{cert}_{\mathbb{B}}$ returns true only when the tuple is present in all worlds. We also note that $\textsc{cert}_{\mathbb{N}} = \min$, is analogous to the definition of certain answers for bag semantics from [23]. For instance, consider the certain annotation of the first tuple from Example 7. The tuple's certain multiplicity is $\textsc{cert}_{\mathbb{N}}(\{2, 3\}) = min(2, 3) = 2$. Similarly, for the third tuple, $\textsc{cert}_{\mathbb{N}}(\{0, 5\}) = 0$. Reinterpreted under set semantics, all tuples that exist (multiplicity > 0) are annotated $true$ (T) and all others $false$ (F). For the first tuple we get, $\sqcap_{\mathbb{B}}(\{T, T\}) = T \wedge T = T$ (certain). For the third tuple we get $\sqcap_{\mathbb{B}}(\{F, T\}) = F \wedge T = F$ (not certain).

3.2 $\mathcal{K}_W$-relations

For the formal exposition in the remainder of this work it will be useful to define an alternative, but equivalent, encoding of an incomplete $\mathcal{K}$-database as a single $\mathcal{K}$-database using a special class of semirings whose elements encode the annotation of a tuple across a set of possible worlds[3]. We assume a fixed set $W = \{m \mid m \in \mathbb{N} \wedge 0 < m \leq n\}$ of possible world identifiers for some number of possible worlds $n \in \mathbb{N}$. Given the domain K of a semiring $\mathcal{K}$, we write K^W to denote the set of elements from the n-way cross-product of K. We annotate tuples t with elements of K^W to store annotations of t in each possible world. We use $\vec{k}$, $\vec{k_1}$, ... to denote elements from K^W to make explicit that they are vectors.

Definition 2 (Possible World Semiring). *Let $\mathcal{K} = (K, \oplus_{\mathcal{K}}, \otimes_{\mathcal{K}}, \mathbb{0}_{\mathcal{K}}, \mathbb{1}_{\mathcal{K}})$ be an l-semiring. We define the possible world semiring $\mathcal{K}_W = (K^W, \oplus_{\mathcal{K}_W}, \otimes_{\mathcal{K}_W}, \mathbb{0}_{\mathcal{K}_W}, \mathbb{1}_{\mathcal{K}_W})$. The operations of this semiring are defined as follows $\forall i \in W$:*

$$\mathbb{0}_{\mathcal{K}_W}[i] \coloneqq \mathbb{0}_{\mathcal{K}} \qquad (\vec{k_1} \oplus_{\mathcal{K}_W} \vec{k_2})[i] \coloneqq \vec{k_1}[i] \oplus_{\mathcal{K}} \vec{k_2}[i]$$

$$\mathbb{1}_{\mathcal{K}_W}[i] \coloneqq \mathbb{1}_{\mathcal{K}} \qquad (\vec{k_1} \otimes_{\mathcal{K}_W} \vec{k_2})[i] \coloneqq \vec{k_1}[i] \otimes_{\mathcal{K}} \vec{k_2}[i]$$

Thus, a $\mathcal{K}_W$-database is simply a pivoted representation of an incomplete $\mathcal{K}$-database.

Example 8. *Reconsider the incomplete $\mathbb{N}$-relation from Example 7. The encoding of this database as a $\mathbb{N}^2$-relation is:*

locale	state	$\underline{\mathbb{N}^2}$
Lasalle	NY	**[3,2]**
Tucson	AZ	**[2,1]**
Greenville	IN	**[0,5]**

Observe that $\mathcal{K}_W$ is a semiring, since we define $\mathcal{K}_W$ 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 the $\mathcal{K}$-database for a possible world (e.g., the *best-guess world*) from a $\mathcal{K}_W$-database by projecting on one dimension of its annotations. This can be modeled as a mapping $\textsc{pw}_i : K^W \rightarrow K$ where $i \in W$:

$$\textsc{pw}_i(\vec{k}) \coloneqq \vec{k}[i] \tag{4}$$

Recall that under possible world semantics, the result of a query Q is the set of worlds computed by evaluating Q over each world of the input. As a sanity check, we would like to ensure that query processing over $\mathcal{K}_W$-relations matches this definition. We can state possible world semantics equivalently as follows: the content of a possible world in the query result ($\textsc{pw}_i(Q(\mathcal{D}))$) is the result of evaluating query Q over this possible world in the input ($Q(\textsc{pw}_i(\mathcal{D}))$):

$$\forall i \in W : \textsc{pw}_i(Q(\mathcal{D})) = Q(\textsc{pw}_i(\mathcal{D}))$$

Recall from Section 2.3 that a mapping between semirings commutes with queries iff it is a semiring homomorphism.

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

Proof. See Appendix A □

A useful consequence of Lemma 1 is that $\mathcal{K}$-databases and $\mathcal{K}_W$-databases are equivalent and interchangeable.

Proposition 1. *Incomplete $\mathcal{K}$-databases and $\mathcal{K}_W$-databases are isomorphic wrt. possible worlds semantics for $\mathcal{RA}^+$ queries.*

Certain Annotations. Since the annotation of a tuple t in a $\mathcal{K}_W$-database is a vector recording t's annotations in all worlds, certain annotations for incomplete $\mathcal{K}$-databases are computed by applying $\sqcap_{\mathcal{K}}$ to the set of annotations contained in the vector. Thus, the certain annotation of a tuple t from a $\mathcal{K}_W$-DB $\mathcal{D}$ is computed as: $\textsc{cert}_{\mathcal{K}}(\mathcal{D}, t) = \sqcap_{\mathcal{K}}(\mathcal{D}(t))$

[3] This encoding is a technical device that allows us to adopt results from the theory of $\mathcal{K}$-relations directly to our problem. It is *not* materialized.

4 UA-DATABASES

We now introduce *UA-DBs* (uncertainty-annotated databases) which encode both under- and over-approximations of the certain annotations of an incomplete $\mathcal{K}$-database $\mathcal{D}$. This is achieved by annotating every tuple with a pair $[c, d] \in K^2$ where d records the tuple's annotation in an arbitrary possible world $D_{bg} \in \mathcal{D}$ i.e., $d = \text{PW}_{bg}(\mathcal{D})(t)$ and c stores the under-approximation of the tuple's certain annotation (i.e., $c \preceq_{\mathcal{K}} \text{CERT}_{\mathcal{K}}(\mathcal{D}, t) \preceq_{\mathcal{K}} d$). We call the world selected as D_{bg} the best-guess world (BGW). Both under- and over-approximations of certain annotations assign tuples annotations from $\mathcal{K}$, making them $\mathcal{K}$-databases. Every possible world is by definition a superset of the certain tuples, so a UA-DB contains all certain answers, even though the certainty of some answers may be underestimated. We start by formally defining the annotation domains of UA-DBs and mappings that extract the two components of an annotation. Afterwards, we state the main result of this section: queries over UA-DBs preserve the under- and over-approximation of certain annotations.

4.1 UA-semirings

We define a UA-semiring as a $\mathcal{K}^2$-semiring, i.e., the direct product of a semiring $\mathcal{K}$ with itself (see Section 4.1). Recall that operations in $\mathcal{K}^2 = (K^2, \oplus_{\mathcal{K}^2}, \otimes_{\mathcal{K}^2}, \mathbb{0}_{\mathcal{K}^2}, \mathbb{1}_{\mathcal{K}^2})$ are defined pointwise, e.g., $[k_1, k_1{}'] \otimes_{\mathcal{K}^2} [k_2, k_2{}'] = [k_1 \otimes_{\mathcal{K}} k_2, k_1{}' \otimes_{\mathcal{K}} k_2{}']$.

Definition 3 (UA-semiring). *Let $\mathcal{K}$ be a semiring. We define the corresponding UA-semiring $\mathcal{K}_{UA} := \mathcal{K}^2$*

Note that for any $\mathcal{K}$, $\mathcal{K}_{UA}$ is a semiring, because, as mentioned earlier, products of semirings are semirings.

4.2 Creating UA-DBs

We now discuss how to derive UA-relations from a $\mathcal{K}_W$-database or a compact encoding of a $\mathcal{K}_W$-database using some uncertain data model like c-tables. Consider a $\mathcal{K}_W$-database $\mathcal{D}$, let D be one of its worlds and $\mathcal{L}$ a $\mathcal{K}$-database under-approximating the certain annotations of $\mathcal{D}$. We refer to $\mathcal{L}$ as a *labeling* and will study such labelings in depth in Section 5 and 7. We cover in Section 6 how to generate a UA-DB from common uncertain data models by extracting a (best-guess) world D and a labeling $\mathcal{L}$. We construct a UA-DB D_{UA} as an *encoding* of D and $\mathcal{L}$ by setting for every tuple t:

$$D_{UA}(t) := [\mathcal{L}(t), D(t)]$$

For a UA-DB D_{UA} constructed in this fashion we say that D_{UA} *approximates* $\mathcal{D}$ by encoding $(\mathcal{L}, D)$. Given a UA-DB D_{UA}, we would like to be able to restore $\mathcal{L}$ and D from D_{UA}. For that we define two morphisms $\mathcal{K}^2 \to \mathcal{K}$:

$$h_{cert}([c, d]) := c \qquad h_{det}([c, d]) := d$$

Note that by construction, if an UA-DB D_{UA} is an *encoding* of a possible world D and a labeling $\mathcal{L}$ of a $\mathcal{K}_W$-database $\mathcal{D}$ then: $h_{det}(D_{UA}) = D$ **and** $h_{cert}(D_{UA}) = \mathcal{L}$.

4.3 Querying UA-DBs

We now state the main result of this section: query evaluation over UA-DBs preserves the under-approximation and over-approximation of certain annotations.

Theorem 1 (Queries Preserve Bounds). *Let $\mathcal{D}$ be a $\mathcal{K}_W$-database, $\mathcal{L}$ a labeling for $\mathcal{K}_W$, D one of its possible worlds, and D_{UA} be the UA-DB encoding the pair $(\mathcal{L}, D)$. Clearly D_{UA} approximates $\mathcal{D}$. Then $Q(D_{UA})$ is an approximation for $Q(\mathcal{D})$ encoding the pair $(Q(\mathcal{L}), Q(D))$.*

Proof Sketch. We decouple c and d by showing that h_{cert} and h_{det} are homomorphisms. Computing d is exactly BGQP. Then, we show in Section 7 that queries over h_{cert} preserve the lower bound. *(See Appendix A for the full proof)*

5 UNCERTAINTY LABELINGS

We now define uncertainty labelings, which are $\mathcal{K}$-databases whose annotations over- or under-approximate certain annotations of tuples in a $\mathcal{K}_W$-database with respect to the natural order of semiring $\mathcal{K}$. A labeling scheme is a mapping from an incomplete databases to labelings.

Definition 4 (Uncertainty Labeling Scheme). *9Let $\mathcal{DB}_{\mathcal{K}}$ be the set of all $\mathcal{K}$-databases, $\mathcal{M}$ an incomplete/probabilistic data model, and $\mathcal{DB}_{\mathcal{M}}$ the set of all possible instances of this model. An uncertainty labeling scheme is a function label : $\mathcal{DB}_{\mathcal{M}} \to \mathcal{DB}_{\mathcal{K}}$ such that the labeling $\mathcal{L}$ = label($\mathcal{D}$) has the schema* **D**.

Ideally, we would like the label (annotation) $\mathcal{L}(t)$ of a tuple t from an uncertainty labeling $\mathcal{L}$ to be exactly $\text{CERT}_{\mathcal{K}}(\mathcal{D}, t)$. Observe that an exact labeling can always be computed in $O(|W|)$ time if all worlds of the incomplete database can be enumerated. However, the number of possible worlds is frequently exponential in the data size. Thus, most incomplete data models rely on factorized encodings, with size typically logarithmic in $|W|$. Ideally, we would like labeling schemes to be PTIME in the size of the encoding (rather than in $|W|$). As mentioned in the introduction, computing certain answers is coNP-complete, so for tractable query semantics we must accept that $\mathcal{L}(t)$ may either over- or under-approximate $\text{CERT}_{\mathcal{K}}(\mathcal{D}, t)$ (with respect to $\preceq_{\mathcal{K}}$). For instance, under bag semantics (semiring $\mathbb{N}$), a label n may be smaller or larger than the certain multiplicity of a tuple. We call a labeling *c-sound* (no false positives) if it consistently under-approximates the certain annotation of tuples, *c-complete* (no false negatives) if it consistently over-approximates 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. For UA-DBs we are mainly interested in c-sound labeling schemes to provide an under-approximation of certain annotations.

Definition 5. *If $\mathcal{L}$ is an uncertainty labeling for $\mathcal{D}$.*

We call $\mathcal{L}$...	***... iff for all tuples*** $t \in \mathcal{D}$...
c-sound	$\mathcal{L}(t) \preceq_{\mathcal{K}} \textit{CERT}_{\mathcal{K}}(\mathcal{D}, t)$
c-complete	$\textit{CERT}_{\mathcal{K}}(\mathcal{D}, t) \preceq_{\mathcal{K}} \mathcal{L}(t)$
c-correct	$\textit{CERT}_{\mathcal{K}}(\mathcal{D}, t) = \mathcal{L}(t)$

A labeling is both c-sound and c-complete iff it is c-correct. Ideally, queries over labelings would preserve these bounds.

Definition 6 (Preservation of Bounds). *A query semantics for uncertainty labelings preserves a property X (c-soundness, c-completeness, or c-correctness) wrt. a class of queries C, if for any incomplete database $\mathcal{D}$, labeling $\mathcal{L}$ for $\mathcal{D}$ that has property X, and query $Q \in C$ we have: $Q(\mathcal{L})$ is an uncertainty labeling for $Q(\mathcal{D})$ with property X.*

6 LABELING SCHEMES

We define efficient (PTIME) labeling schemes for three incomplete data models and their probabilistic extensions: Tuple-Independent (probabilistic) databases [41], (P)C-Tables [22, 25], and x-DBs [2]. For further details and examples see [14]. We also discuss methods for selecting a BGW for each model.

Tuple-Independent Databases. A tuple-independent database (TI-DB) $\mathcal{D}$ is a database where each tuple t is marked as optional or not. The incomplete database represented by a TI-DB $\mathcal{D}$ is the set of instances that include all non-optional tuples and some subset of the optional tuples. That is, the existence of a tuple t is independent of the existence of any other tuple t'. In the probabilistic version of TI-DBs each tuple is associated with its marginal probability. The probability of a possible world is then the product of the probability of all tuples included in the world multiplied by the product of $1 - P(t)$ for all tuples from $\mathcal{D}$ that are not part of the possible world. We define a labeling function $\text{label}_{\text{TI-DB}}$ for TI-DBs that returns a $\mathbb{B}$-labeling $\mathcal{L}$ that annotates a tuple with T (certain) iff it is not optional. For probabilistic TI-DBs we label tuples as certain if their marginal probability is 1.

$$\mathcal{L}(t) := T \Leftrightarrow t \text{ is not marked as optional}$$

To create a BGW from an incomplete TI-DB $\mathcal{D}$, we can choose any subset of $\mathcal{D}$ that is a superset of the tuples we labeled as certain. For probabilistic TI-DBs, we choose the world with the highest probability, which can be computed efficiently as follows: $D_{bg} := \{t \mid t \in \mathcal{D} \wedge P(t) \geq 0.5\}$

Theorem 2 ($\text{label}_{\text{TI-DB}}$ is c-correct). *Given a TI-DB $\mathcal{D}$, $\textit{label}_{\textit{TI-DB}}(\mathcal{D})$ is a c-correct labeling.*

Proof. Trivially holds. An incomplete (probabilistic) database tuple is certain iff it is not optional (if $P(t) = 1$). □

(P)C-tables. Tuples in a *C-table* [25] consist of constants and variables from a set Σ. Tuples are annotated by a boolean expression $\phi_{\mathcal{D}}(t)$ over comparisons of values from $\Sigma \cup \mathbb{D}$, called the local condition. Each variable assignment $v : \Sigma \to \mathbb{D}$ defines a possible world derived by retaining only tuples with local conditions satisfied under v. Determining certainty of tuples based on a C-table is expensive. Thus, we cannot hope for an efficient c-correct labeling scheme for C-tables. Instead, consider the following sufficient condition for certainty. If (1) a tuple t in a C-table contains only constants and (2) its local condition $\phi_{\mathcal{D}}(t)$ is a tautology, then the tuple is certain. Our labeling scheme for C-tables applies this sufficient condition and, thus, is c-sound. Formally, $\mathcal{L} = \text{label}_{\text{C-TABLE}}(\mathcal{D})$, where for a C-table $\mathcal{D}$ and any tuple $t \in \text{TupDom}$:

$$\mathcal{L}(t) = T \Leftrightarrow \phi_{\mathcal{D}}(t) \text{ is in CNF } \wedge (\models \phi_{\mathcal{D}}(t))$$

Green et. al. [22] extended C-tables by assigning each variable to a probability distribution over its possible values. In the resulting "PC-tables," variables are treated as independent, i.e., the probability of a possible world is the product of probabilities of its defining variable assignments. Our labeling scheme works for both C-tables and PC-tables. To compute a BGW for a C-table, we can randomly choose an assignment for each variable in Σ. For a PC-table, computing the most likely possible world reduces to answering a query over a TI-DB (#P in general [41]). Thus, we heuristically select the highest probability value for each variable independently. Any such assignment must contain all certain tuples, since their local conditions are tautologies. Alternatively, a wide range of algorithms [15, 16, 18, 35] can compute an arbitrarily close approximation of the most likely world.

Theorem 3 ($\text{label}_{\text{C-TABLE}}$ is c-sound). *Given an incomplete/probabilistic database $\mathcal{D}$ encoded as a C-table or PC-table, $\textit{label}_{\textit{C-TABLE}}(\mathcal{D})$ is c-sound.*

x-DBs. An x-DB [2] is a set of x-relations, which are sets of x-tuples. An x-tuple τ is a set of tuples $\{t_1, \ldots, t_n\}$ with a label indicating whether the x-tuple is optional. Each x-tuple is assumed to be independent of the others, and its alternatives are assumed to be disjoint. Thus, a possible world of an x-relation R is constructed by selecting at most one alternative $t \in \tau$ for every x-tuple τ from R if τ is optional, or exactly one if it is not optional. The probabilistic version of x-DBs (also called a Block-Independent or BI-DB) as introduced in [2] assigns each alternative a probability and we require that $P(\tau) = \sum_{t \in \tau} P(t) \leq 1$. Thus, a tuple is optional if $P(\tau) < 1$ and there is no need to use labels to mark optional tuples. We use $|\tau|$ to denote the number of alternatives of x-tuple τ. We define a labeling scheme $\text{label}_{\text{x-DB}}$ for x-relations where tuple

t's annotation is T iff t is the single, non-optional alternative of an x-tuple. In probabilistic x-DBs we check $P(\tau) = 1$.

$$\mathcal{L}(t) \coloneqq T \Leftrightarrow \exists \tau \in \mathcal{D} : |\tau| = 1 \wedge \tau \text{ is not optional}$$

Theorem 4 (label$_{\text{x-DB}}$ is c-correct). *Given a database $\mathcal{D}$, label$_{x\text{-}DB}(\mathcal{D})$ is a c-correct labeling.*

For probabilistic x-DBs, the possible world with highest probability can be efficiently computed and used as the BGW. Since the x-tuples in an x-DB are independent, the probability of a possible world from an x-DB $\mathcal{D}$ is maximized by including the highest probability alternative for each x-tuple τ (i.e., $\text{argmax}_{t \in \tau} P(t)$), or no tuple if that is the highest probability option (i.e., $\max_{t \in \tau} P(t) < (1 - P(\tau))$). For incomplete x-DBs, we choose a random alternative for each x-tuple.

7 QUERYING LABELINGS

We now study whether queries over labelings produced by *labeling schemes* such as the ones described in Section 6 preserve c-soundness. Specifically, we demonstrate that standard $\mathcal{K}$-relational query evaluation preserves *c-soundness* for any c-sound labeling scheme. Recall that a query semantics for labelings preserves c-soundness if a query $Q(\mathcal{L})$ evaluated on a c-sound labeling $\mathcal{L}$ of incomplete database $\mathcal{D}$ is a c-sound labeling for $Q(\mathcal{D})$. Our result generalizes a previous result of Reiter [39] to any type of incomplete $\mathcal{K}$-database for which we can define an efficient c-sound labeling scheme. We need the following lemma, to show that the natural order of a semiring factors through addition and multiplication. This is a known result that we only state for completeness.

Lemma 2. *Let $\mathcal{K}$ be a naturally ordered semiring. For all $k_1, k_2, k_3, k_4 \in \mathcal{K}$ we have:*

$$k_1 \preceq_{\mathcal{K}} k_3 \wedge k_2 \preceq_{\mathcal{K}} k_4 \Rightarrow k_1 \oplus_{\mathcal{K}} k_2 \preceq_{\mathcal{K}} k_3 \oplus_{\mathcal{K}} k_4$$
$$k_1 \preceq_{\mathcal{K}} k_3 \wedge k_2 \preceq_{\mathcal{K}} k_4 \Rightarrow k_1 \otimes_{\mathcal{K}} k_2 \preceq_{\mathcal{K}} k_3 \otimes_{\mathcal{K}} k_4$$

Proof. See Appendix A □

7.1 Preservation of C-Soundness

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

Functions that have the property mentioned in (a) 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 $a_1, a_2 \in A$:

$$f(a_1 + a_2) \geq_B f(a_1) + f(a_2) \qquad \text{(superadditive)}$$
$$f(a_1 \times a_2) \geq_B f(a_1) \times f(a_2) \qquad \text{(supermultiplicative)}$$

In a nutshell, if we are given a c-sound $\mathcal{K}$-labeling, then evaluating any $\mathcal{RA}^+$-query over the labeling using $\mathcal{K}$-relational query semantics preserves c-soundness if we can prove that cert$_{\mathcal{K}}$ is superadditive and supermultiplicative.

Lemma 3. *Let $\mathcal{K}$ be a semiring. cert$_{\mathcal{K}}$ is superadditive and supermultiplicative wrt. the natural order $\preceq_{\mathcal{K}}$.*

Proof. See Appendix A □

Using the superadditivity and -multiplicativity of cert$_{\mathcal{K}}$, we now prove preservation of c-soundness.

Theorem 5. *Let $\mathcal{D}$ be a $\mathcal{K}_W$-database and $\mathcal{L}$ a c-sound labeling for $\mathcal{D}$. $\mathcal{RA}^+$ queries over $\mathcal{L}$ preserve c-soundness.*

Proof. See Appendix A □

In Appendix B we demonstrate that under certain circumstances, queries also preserve c-completeness.

8 IMPLEMENTATION

We now discuss the implementation of a UA-DB as a query rewriting front-end built on top of a relational DBMS. A $\mathcal{K}_{UA}$-relation with schema $\mathbf{R}(A_1, \ldots, A_n)$ annotated with a pairs of $\mathcal{K}$-elements $[c, d]$ is encoded by $\mathcal{K}$-relation $\mathbf{R}'(A_1, \ldots, A_n, C)$ where the annotation of each tuple encodes d and attribute C stores c. We specifically implement UA-DBs for bag semantics, as this is the model used by most DBMSes. In contrast to $\mathbb{N}$-relations where the multiplicity of a tuple is stored as its annotation, relational databases represent a tuple t with multiplicity n as n copies of t. We use C as a boolean marker and mark c copies of t as certain (1) and the remaining $d - c$ copies as uncertain (0) as shown in the example in Section 1.

Our frontend rewriting engine receives queries of the form $Q(D_{UA})$ over an $\mathbb{N}_{UA}$-annotated database D_{UA} with schema $\{ \mathbf{R}_i(A_1, \ldots, A_n) \}$. It rewrites these into equivalent queries $[\![Q]\!]_{UA}(D)$ over a classical bag-relational database D with schema $\{ \mathbf{R}'_i(A_1, \ldots, A_n, C) \}$ where $C \in \{0, 1\}$ denotes the uncertainty label. The rewrite rules implementing $[\![\cdot]\!]_{UA}$ are given in Figure 8. Implementations of the labeling and BGW-extraction schemes from Section 6 are used to make our rewriting engine directly compatible with a wide range of incomplete and probabilistic data models; Such inputs are translated inline into encoded $\mathbb{N}_{UA}$-relations. We implement our approach as a middleware over a database system through an extension of SQL. An input query is first parsed,

$$
\begin{aligned}
[\![R]\!]_{UA} &= \text{A Labeled R (see Section 6)} \\
[\![\sigma_\theta(Q)]\!]_{UA} &= \texttt{SELECT * FROM } [\![Q]\!]_{UA} \texttt{ WHERE } \theta \\
[\![\pi_{A_1 \ldots A_n}(Q)]\!]_{UA} &= \texttt{SELECT A1, ..., AN, C FROM } [\![Q]\!]_{UA} \\
[\![Q_1 \bowtie_\theta Q_2]\!]_{UA} &= \texttt{SELECT Q1.*, Q2.*, Q1.C*Q2.C AS C} \\
&\quad \texttt{FROM } [\![Q_1]\!]_{UA}, [\![Q_2]\!]_{UA} \texttt{ WHERE } \theta \\
[\![Q_1 \cup Q_2]\!]_{UA} &= [\![Q1]\!]_{UA} \texttt{ UNION ALL } [\![Q_2]\!]_{UA}
\end{aligned}
$$

Figure 8: Query rewrite rules

translated into a relational algebra graph, rewritten using $[\![\cdot]\!]_{UA}$, and then converted back to SQL for execution. We formally prove the correctness of our rewriting and show SQL implementations of our labeling schemas from Section 6 in [14].

9 RELATED WORK

Incomplete and probabilistic data models. Uncertainty was recognized as an important problem by the database community early-on. Codd [11] 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. C-tables are closed under full relational algebra. Reiter [39] proposed to model databases as logical theories, a model equivalent to V-tables. Probabilistic data models quantify the uncertainty in incomplete databases by assigning probabilities to individual possible worlds. TI-DBs [41] are a prevalent model for probabilistic data where each tuple is associated with its marginal probability and tuples are assumed to be independent. Green et al. [22] studied probabilistic versions of C-tables. Virtual C-tables generalize C-tables [28, 45] by allowing symbolic expressions as values. Probabilistic query processing (PQP) has been studied for several decades (e.g., an important survey is [41]). Of particular note, Gatterbauer and Suciu [18] showed that (efficient) extensional evaluation semantics compute a lower bound on result probabilities.

Certain Answers. Many approaches for answering queries over incomplete databases employ certain answer semantics [1, 23–25, 36]. The foundational work by Lipski [37] defined certain answers analogously to our approach, but using minima instead of GLBs. Computing certain answers is coNP-complete [1, 25] (data complexity) for first order queries, even for restricted data models such as Codd-tables. This hardness result even holds for conjunctive queries over more complex uncertain data models (e.g., OR-databases [26]). Thus, it is not surprising that approaches for approximating the set of certain answers have been proposed. Reiter [39] 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, 24, 36] propose a query semantics that preserves c-soundness for

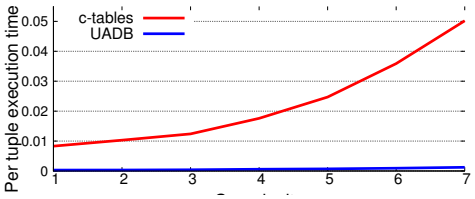


Figure 9: Certain answers over C-tables

full relational algebra (first order queries) for Codd- and V-tables. Then, [24] defined certain and possible multiplicities for bag semantics, and presented initial thoughts on how to extend [23] for bag semantics. Our approach works with a wider range of data models and models of uncertainty than [23, 24], at the cost of being a slightly weaker approximation. Furthermore, unlike this approach, UA-DBs are closed under query evaluation. Sundarmurthy et al. [42] introduced m-tables, which can represent not just uncertainty, but also model information about missing tuples. [42] also introduced the terms c-soundness/-correctness. Consistent query answering [5, 6] (CQA) is computing certain answers to a query over the incomplete database consisting of all repairs for a database that violates a set of constraints. The complexity of variants of this problem has been studied extensively (e.g., [9, 29, 32]) and several combinations of classes of constraints and queries have been identified that permit first-order rewritings [17, 19, 43, 44]. Geerts et al. [19] study first order under-approximations of certain answers in the context of CQA.

Annotated Databases. Green et al. [21] introduced the semiring annotation framework that we utilize in this work. The connection between annotated databases, provenance, and uncertainty has been recognized early-on. A particular type of semiring annotations, often called Lineage, has been used for probabilistic query processing (e.g., see [40, 41]). Green et al. [21] observed that set semantics incomplete databases can be expressed as $\mathcal{K}$-relations by annotating each tuple with the set of worlds containing it. We define a more general type of incomplete databases based on $\mathcal{K}$-relations which is defined for any l-semiring. Kostylev et al. [31] investigate how to deal with dependencies among annotations from multiple domains. Similar to [31], we consider "multi-dimensional" annotations, but for a very different purpose: to extend incomplete databases beyond set semantics.

10 EXPERIMENTS

We evaluate the performance of queries over UA-DBs implemented on a commercial DBMS[4]. We compare **UA-DB**s with (1) **Det**: Deterministic BGQP, (2) **Libkin**: An alternate under-approximation of certain answers [23, 36], (3) **MayBMS**: We

[4]The DBMS is not identified due to license restrictions

use MayBMS to compute the full set of possible answers[5], and (4) **MCDB**: We use MCDB-style [27] database sampling (10 samples) to over-approximate the certain answers. All experiments are run on a machine with 2×6 core AMD Opteron 4238 CPUs, 128GB RAM, 4×1TB 7.2K HDs (RAID 5). We report the average running time of 5 runs. We also evaluate false negative (i.e., a misclassified certain answer) rates for UA-DBs and both false negative and false positive rates for other systems. Furthermore, we demonstrate that BGQP and UA-DBs produce answers that are of higher utility (more similar to a ground truth result) than certain answers.

10.1 Performance Comparison

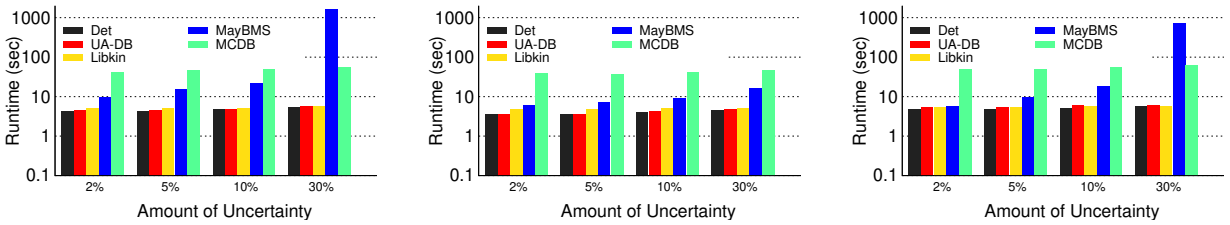
We first use PDBench [3], a modified TPC-H data generator [12] that introduces uncertainty by generating random possible values for randomly selected cells (attributes). The generator produces a columnar encoding optimized for MayBMS, with tables as pairs of tuple identifiers and attribute values. Ambiguity arises from having multiple values for the same tuple identifier. We directly run MayBMS queries (omitting probability computations) on these columnar tables. For MCDB, we simulate the tuple bundle query using 10 samples. We also apply Libkin by constructing a database instance with nulls from the PDBench tables and applying queries generated by the optimized rewriting described in [23]. We run deterministic queries and queries generated by our approach on one possible world that is selected by randomly choosing a value for each uncertain cell. For our approach, we treat the input as an x-DB and mark tuples with at least one uncertain cell as uncertain. The three PDBench queries roughly correspond to TPC-H queries Q3, Q6 and Q7.

Amount of uncertainty. Using a scale factor 1 database (~1GB of data per possible world), we evaluate scalability with respect to amount of uncertainty. Using PDBench, we vary the percentage of uncertain cells in the range 2%, 5%, 10% and 30%. Each uncertain cell has up to 8 possible values. Figure 10 shows the runtime results for the three PDBench queries. As expected, runtimes for UA-DBs and Libkin are similar to deterministic query processing. The slight overheads arise from propagating uncertainty annotations and dealing with nulls, respectively. Furthermore, UA-DBs have to output additional tuples that belong to the best-guess world, but are not certain. Libkin slightly outperforms UA-DBs for query Q3 at levels of uncertainty above 10%, since the query's join only returns certain tuples and, thus, there is no overhead for dealing with nulls. For queries Q1 and Q2, UA-DBs slightly outperform Libkin as the overhead of comparing labeled nulls outweighs the overhead for returning a larger result and propagating uncertainty annotations. MCDB effectively needs to evaluate queries once for each sample, and so runs more than 10 times slower than deterministic query processing. MayBMS has a reasonable, but still noticeable overhead at lower levels of uncertainty. As uncertainty increases, the query output size in MayBMS increases roughly cubically for Q1 and Q3, and it begins to perform several orders of magnitude (the plots use log scale) slower than UA-DBs. MayBMS performs better for the simple selection query Q2. To better understand the performance of MayBMS, we show result sizes (number of tuples) for each query varying amounts of uncertainty in Figure 11. Our approach produces the same number of results as deterministic processing. Conversely, MayBMS returns the full set of possible answers and, thus the result size increases dramatically as uncertainty increases. We also show the percentage of certain answers for each query per input uncertainty level in Figure 12. The unexpected increase of result size for Q1 over UA-DBs is caused by a shift in the correlation between attributes `o_orderkey` and `l_shipdate` that affects the number of tuples passing Q1's selection condition resulting from PDBench choosing values for uncertain cells independently.

Dataset size. To evaluate scalability, we use datasets with scale factors (SF) 0.1 (100MB), 1 (1GB) and 10 (10GB) and fix the uncertainty percentage (2%). The results are shown in Figure 13. Again UA-DBs and Libkin exhibit performance similar to deterministic queries as well as certain answers and MCDB is again significantly slower. MayBMS's relative overhead over deterministic processing increases with data set size. For instance, for Q1 the overhead is ~ 60% for SF 0.1 and ~500% for SF 10.

Certain Answers over C-tables. As an example of a more complex incomplete data model, we evaluate the performance of UA-DBs against computing certain answers over C-tables. We create a synthetic table with 8 attributes. For each tuple we randomly chose half of its attributes to be variables and the other half to be floating point constants. We construct random queries by assembling a number of randomly chosen self-joins, projections, or selections. We measure query execution time using UA-DBs. The exact certain answers for a query over the C-tables are computed by instrumenting the query to calculate a local condition for every result tuple and running the Z3 constraint solver (https://github.com/Z3Prover/z3) over the resulting boolean expression. An answer is certain iff its local condition is a tautology. Figure 9 shows the average runtime per result tuple for both C-tables and UA-DBs averaged over all randomly generated queries. The x-axis is the number of operators (i.e., selection, projection or join) in the source query. Overhead for C-tables increases super-linearly in query complexity from about 27× to over 40×.

[5]Times listed for MayBMS do include only computing possible answers and not computing probabilities, unless stated otherwise.



(a) PDBench - Q1 (b) PDBench - Q2 (c) PDBench - Q3

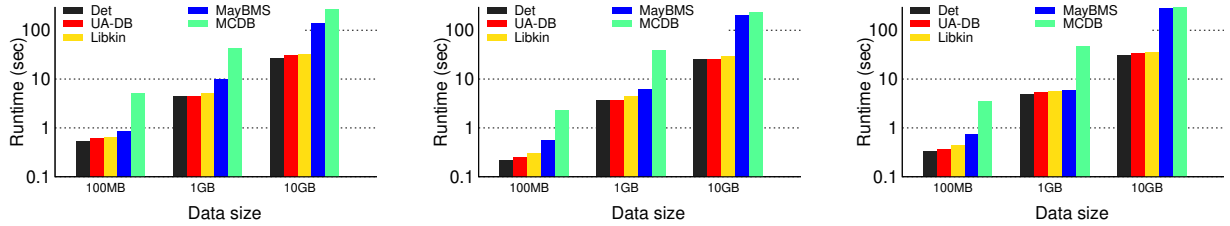
Figure 10: Performance of PDBench queries - varying the amount of uncertainty for scale factor 1

	UA-DB			MayBMS		
	$Q1$	$Q2$	$Q3$	$Q1$	$Q2$	$Q3$
2%	14,260	152,583	9,016	113,966	210,996	15,108
5%	34,041	152,432	8,619	501,114	327,052	32,438
10%	61,800	152,389	8,794	2,392,916	618,199	97,454
30%	130,581	152,885	7,994	134,054,635	3,941,554	4,351,782

Figure 11: Query result sizes (#rows)

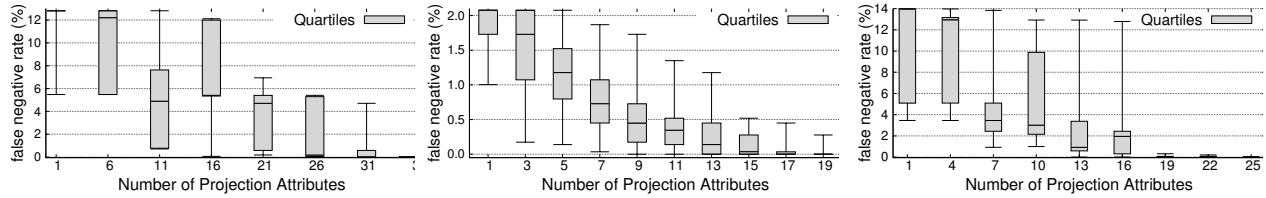
	$Q1$	$Q2$	$Q3$
2%	0 (0%)	143,618 (94%)	7,861 (87%)
5%	1 (0%)	130,594 (86%)	6,023 (70%)
10%	4 (0%)	111,120 (73%)	3,979 (45%)
30%	1 (0%)	52,724 (34%)	586 (7%)

Figure 12: Result certain answer %

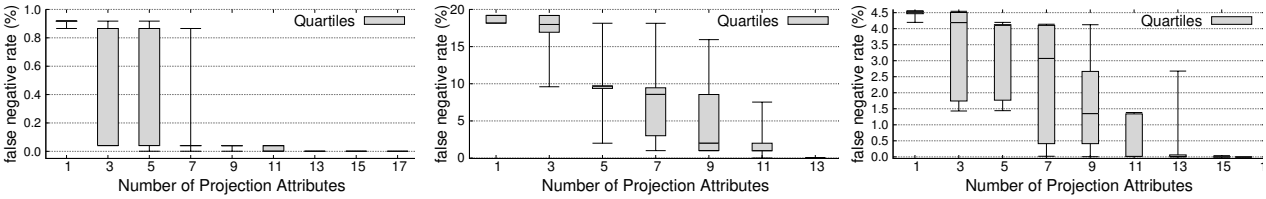


(a) PDBench - Q1 (b) PDBench - Q2 (c) PDBench - Q3

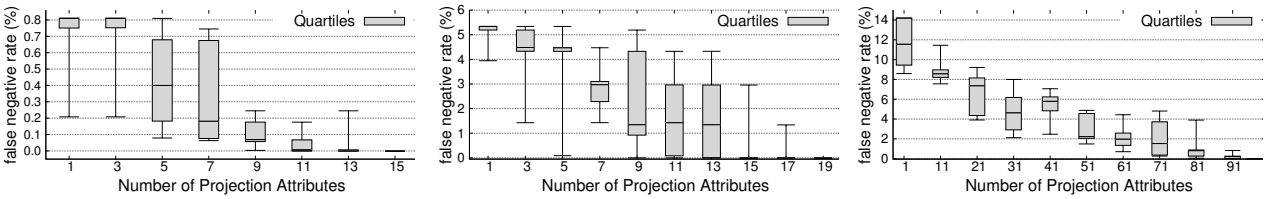
Figure 13: Performance of PDBench queries - varying database size for 2% uncertainty



(a) Building Violations (b) Shootings in Buffalo (c) Business Liscenses



(d) Chicago Crime (e) Contracts (f) Food Inspections



(g) Graffiti Removal (h) Building Permits (i) Public Library Survy

Figure 14: Measuring incompleteness as the fraction of certain answers that were misclassified as uncertain

Dataset	Rows	Cols	U_{Attr}	U_{Row}
Building Violations	1.3M	35	0.82%	12.8%
Shootings in Buffalo	2.9K	21	0.24%	2.1%
Business Licenses	63K	25	1.39%	14.0%
Chicago Crime	6.6M	17	0.21%	0.9%
Contracts	94K	13	1.50%	19.2%
Food Inspections	169K	16	0.34%	4.6%
Graffiti Removal	985K	15	0.09%	0.8%
Building Permits	198K	19	0.42%	5.3%
Public Library Survy	9.2K	99	1.19%	14.2%

Figure 15: Real World Datasets

	$Q1$	$Q2$	$Q3$	$Q4$	$Q5$
Overhead	2.28%	1.81%	1.32%	2.88%	3.51%
Error Rate	0.55%	0.37%	0%	0.92%	0.29%

Figure 16: Real Query Results

10.2 Real world datasets

We use multiple real world datasets (Appendix C) from a wide variety of domains to evaluate how our approach performs for real world data. We use SparkML to impute missing values in the datasets, treating alternative imputations as a source of uncertainty. The resulting dataset, represented as an x-DB, was converted to a UA-DB using $\text{label}_{\text{x-DB}}$ (6), which marks all tuples with at least one uncertain attribute as uncertain. Figure 15 shows basic statistics for the cleaned datasets: the #rows, #attributes, the percentage of attribute values that are uncertain (U_{attr}), and the percentage of rows marked as uncertain by our c-complete labeling scheme (U_{row}).

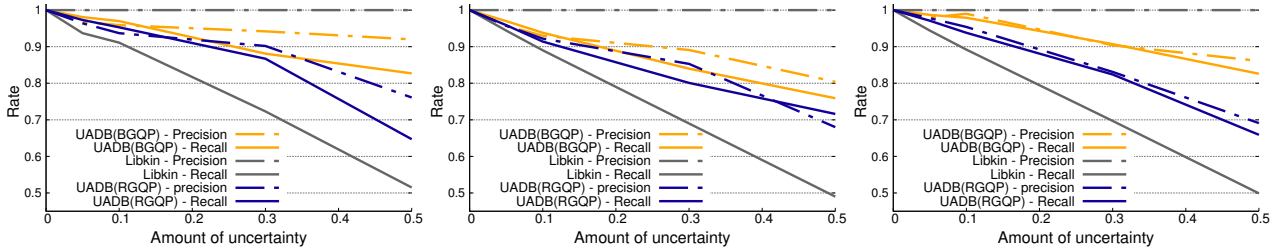
Incompleteness. To measure the false negative rate (fraction of answers that are misclassified as uncertain) of our approach, we use queries that project on a randomly chosen set of attributes. The rationale for this is that projecting an uncertain tuple onto a subset of its attributes that are certain causes the tuple to produce a certain answer. This is the primary situation in which UA-DBs mis-classify results, so this experiment represents a worst case scenario for UA-DBs. We evaluate queries which project on a randomly chosen set of attributes and measure the false negative rate (*FNR*). Figure 14a to 14i show the distribution of the FNR (min, 25-percentile, median, 75-percentile, max) for queries with a fixed number of projection attributes. As expected, the FNR decreases as the number of projection attributes grows, but is low in general (less than 20% in the worst case).For most datasets, the median FNR is below 5% when at least half of the attributes are involved in the projection. Note that selection and join do not produce any "new" false negative results (see proof of Theorem 6 in Appendix B). This shows that for real world datasets with correlated errors, the FNR is typically low.

Real Queries. We next evaluate the effectiveness of our approach on five queries over the real world datasets (we present the SQL code and descriptions of these queries in [14]). Most of our real world datasets are from open data portals that associate analyses (e.g., visualizations) with datasets. Test queries are reverse engineered from these analyses. We measure the performance overhead and false negative rate of UA-DBs. Performance overhead is measured as the slowdown relative to deterministic query processing. As Figure 16 shows, our approach introduces a slight (less than 4%) overhead for these queries. The worst case (4%) is Q5, which involves a join operator. All other queries, which contain only selections and projections have under 3% overhead. In each case, we saw a 1% false negative rate or lower. Notably, Q3 returns no misclassified results due to its small result size.

Probabilistic databases. We next compare the performance and accuracy of UA-DBs against MayBMS. For this experiment, we use a BI-DB (an x-DB with probabilities), varying the number of alternatives for each block and use three queries Q_{P1}, Q_{P2} and Q_{P3} of varying complexity described in [14]. For MayBMS, we treat tuples with probability $p \geq 1$ as certain[6]. Figure 18 shows both runtime and error rate for both systems, with 2, 5, 10, or 20 alternatives. For MayBMS we show the result for exact probability computation and for approximation using the scheme from [38] with an error bound of 0.3 (shown in parentheses). Note that query processing in a UA-DB is independent of the number of possible worlds. Only a single alternative is used for each block. We observe that MayBMS's results include both false positives and false negatives. Because results are computed by summing floating point numbers, even MayBMS' exact probability computations exhibits a small amount of rounding error that is more noticeable for larger number of alternatives (e.g., MB-20). Although approximating probabilities can improve performance especially for complex queries, MayBMS is still orders of magnitude slower than UA-DBs. Q_{P3} includes a self-join which further slows MayBMS down due to the increase in possible worlds and expression complexity.

Beyond Set Semantics. In this experiment we evaluate the FNR of our approach using bag semantics (semiring $\mathbb{N}$) and the access control semiring $\mathbb{A}$ [21]. For the bag semantics experiment we evaluate projections under bag semantics over some of the real world datasets from Figure 15. The results for this experiment are shown in Figure 19. Observe that the FNR is similar to the set semantics case. The access control semiring annotates each tuple with an access control level (one of **0** - "nobody can access the data', **T** is "top secret", **S** is "secret", **C** is "confidential", and **P** is "public") to determine what clearance-level is necessary to view the tuple. Addition (multiplication) is max (min) according to the following order over the elements $\mathbf{0} < \mathbf{T} < \mathbf{S} < \mathbf{C} < \mathbf{P}$. For this experiment, we emulate a scenario where private information in a dataset is heuristically detected and secured with an $\mathbb{A}$ annotation.

[6]MayBMS may report prob. > 1 due to rounding/approximation errors.



(a) Income Survey **(b) Buffalo News** **(c) Business License**

Figure 17: Utility - varying the amount of uncertainty

		UADB	MB-02	MB-05	MB-10	MB-20
Q_{P1}	time (ms)	3.1	4.0 (4.1)	22.7 (22.3)	308.5 (305.6)	4.8k (4.7k)
	error	0%	0% (0%)	0% (0%)	0% (0%)	0% (0%)
Q_{P2}	time (ms)	4.4	6.8 (6.8)	28.4 (28.5)	374.5 (367.0)	8.8k (7.0k)
	error	1.6%	0% (0%)	0%(0%)	0% (0.5%)	0.5% (1.1%)
Q_{P3}	time (ms)	7.6	54.0 (20.3)	17.0k (10.8k)	289.7k (118.6k)	3.5m (1.1m)
	error	3.0%	0% (0.1%)	0.1% (0.1%)	0.2% (0.3%)	0.6% (1.1%)

Figure 18: Probabilistic database

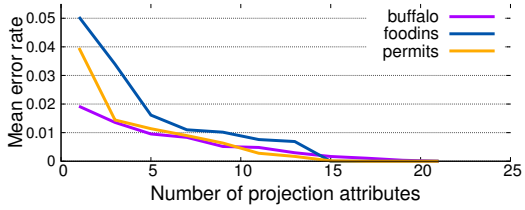


Figure 19: Bag semantics - mislabelings

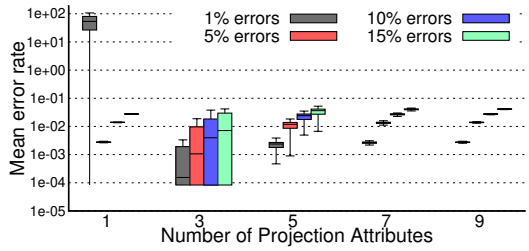


Figure 20: Access control semiring - mislabelings

Using 5 real world datasets from Figure 15, we randomly assigned access control labels to each tuple, and then created multiple labelings with 1%, 2%, 5%, and 10% of misclassified tuples. We evaluated random projection queries over these datasets and measured the amount of misclassified query results weighted by the distance between the certain annotation and the labeling, e.g., the distance of **C** and **T** is $\frac{2}{5} = 0.4$. In Figure 20, we vary the number of projection attributes and show the distribution of the amount of misclassified query results over 9 randomly selected projection queries for 5 datasets. The FNR increases when the input error rate is increased, but is quite low in most cases.

10.3 Utility of Query Answers

We claimed that BGQP and, thus also UA-DBs, have better utility than certain answers, as additional, useful possible answers are included in the result. The next experiment supports this claim quantitatively by contrasting the under-approximation of Libkin with UA-DBs evaluating two methods for extracting a best-guess world. To start, we create an incomplete database for which we have the ground truth (i.e., a "correct" possible world). This world (denoted as D_{ground}) is created by processing a source dataset to remove all rows with nulls. We next use D_{ground} to create an incomplete database $\mathcal{D}$ by replacing a random set of attribute values with nulls, varying the fraction of attributes replaced from 0% (deterministic input), to 50%. Then, we derive a best guess world D_{clean} from $\mathcal{D}$ by either using a standard missing value imputation algorithm (we refer to this method as **BGQP**) or randomly pick a replacement value (random-guess query processing or **RGQP**). We evaluate queries over $\mathcal{D}$ and D_{clean} using Libkin and UA-DBs respectively, and compare the result with the ground truth D_{ground}. Figure 17 shows both precision (fraction of results in D_{ground}) and recall (fraction of D_{ground} in the results) as we vary the level of uncertainty. Libkin's method always under-approximates, guaranteeing 100% precision. However, recall is much lower than for UA-DBs and drops rapidly when the amount of uncertainty is increased. In contrast, the precision and recall achieved by UA-DBs remains between 80-90% for BGQP, even when half of all attribute values are uncertain. This supports our conjecture that certain answers are less similar to actual answers than answers obtained over a best-guess world. Compared with BGQP, RGQP is less accurate and less complete, but still has a higher recall than Libkin.

11 CONCLUSIONS AND FUTURE WORK

We proposed UA-DBs as a novel and efficient way to represent uncertainty as bounds on certain answers. Being based on $\mathcal{K}$-relations, our approach applies to the incomplete version of any data model that can be encoded as $\mathcal{K}$-relations including set and bag semantics. UA-DBs are backward compatible with many uncertain data models such as tuple-independent databases, x-DBs and C-tables. In future work, we plan add attribute level annotations, to encode certainty at finer granularity, and to support larger classes of queries,

e.g., queries involving negation and aggregation. Furthermore, we plan to explore uncertain versions of more complex semirings in the context of new use cases such as inconsistent query answering and querying the result of data exchange.

ACKNOWLEDGMENTS

This work is supported in part by NSF Awards OAC-1640864 and IIS-175046. The conclusions and opinions in this work are solely those of the authors and do not represent the views of the National Science Foundation.

REFERENCES

[1] Serge Abiteboul, Paris C. Kanellakis, and Gösta Grahne. 1991. On the Representation and Querying of Sets of Possible Worlds. *Theor. Comput. Sci.* 78, 1 (1991), 158–187.

[2] Parag Agrawal, Omar Benjelloun, Anish Das Sarma, Chris Hayworth, Shubha U. Nabar, Tomoe Sugihara, and Jennifer Widom. 2006. Trio: A System for Data, Uncertainty, and Lineage. In *VLDB*.

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

[4] L. Antova, C. Koch, and D. Olteanu. 2007. MayBMS: Managing Incomplete Information with Probabilistic World-Set Decompositions. In *ICDE*.

[5] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. 1999. Consistent Query Answers in Inconsistent Databases. In *PODS*.

[6] Leopoldo E. Bertossi. 2011. *Database Repairing and Consistent Query Answering*. Morgan & Claypool Publishers.

[7] Jihad Boulos, Nilesh N. Dalvi, Bhushan Mandhani, Shobhit Mathur, Christopher Ré, and Dan Suciu. 2005. MYSTIQ: a system for finding more answers by using probabilities. In *SIGMOD*.

[8] Stanley Burris and H.P. Sankappanavar. 2012. *A Course in Universal Algebra*. Springer-Verlag.

[9] Andrea Calì, Domenico Lembo, and Riccardo Rosati. 2003. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *PODS*.

[10] Xu Chu, John Morcos, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. 2015. KATARA: A Data Cleaning System Powered by Knowledge Bases and Crowdsourcing. In *SIGMOD*.

[11] E. F. Codd. 1979. Extending the Database Relational Model to Capture More Meaning. *TODS* 4, 4 (1979), 397–434.

[12] Transaction Processing Performance Council. [n. d.]. TPC-H specification. http://www.tpc.org/tpch/.

[13] Ronald Fagin, Benny Kimelfeld, and Phokion G. Kolaitis. 2011. Probabilistic Data Exchange. *J. ACM* 58, 4 (2011), 15:1–15:55.

[14] Su Feng, Aaron Huber, Boris Glavic, and Oliver Kennedy. 2019. *Uncertainty Annotated Databases - A Lightweight Approach for Approximating Certain Answers (extended version)*. Technical Report CoRR. https://arxiv.org/pdf/1904.00234.

[15] Robert Fink, Andrew Hogue, Dan Olteanu, and Swaroop Rath. 2011. SPROUT^2: a squared query engine for uncertain web data. In *SIGMOD*. 1299–1302.

[16] Robert Fink, Jiewen Huang, and Dan Olteanu. 2013. Anytime approximation in probabilistic databases. *VLDB J.* 22, 6 (2013), 823–848.

[17] Ariel D Fuxman and Renée J Miller. 2005. First-order query rewriting for inconsistent databases. In *ICDT*.

[18] Wolfgang Gatterbauer and Dan Suciu. 2017. Dissociation and propagation for approximate lifted inference with standard relational database management systems. *VLDBJ* 26, 1 (2017), 5–30.

[19] Floris Geerts, Fabian Pijcke, and Jef Wijsen. 2017. First-order underapproximations of consistent query answers. *International Journal of Approximate Reasoning* 83 (2017), 337–355.

[20] Floris Geerts and Antonella Poggi. 2010. On database query languages for K-relations. *J. Applied Logic* 8, 2 (2010), 173–185.

[21] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. 2007. Provenance Semirings. In *PODS*.

[22] Todd J. Green and Val Tannen. 2006. Models for Incomplete and Probabilistic Information. *IEEE Data Eng. Bull.* 29, 1 (2006), 17–24.

[23] Paolo Guagliardo and Leonid Libkin. 2016. Making SQL Queries Correct on Incomplete Databases: A Feasibility Study. In *PODS*.

[24] Paolo Guagliardo and Leonid Libkin. 2017. Correctness of SQL Queries on Databases with Nulls. *SIGMOD Record* 46, 3 (2017), 5–16.

[25] Tomasz Imielinski and Witold Lipski Jr. 1984. Incomplete Information in Relational Databases. *J. ACM* 31, 4 (1984), 761–791.

[26] Tomasz Imielinski, Ron van der Meyden, and Kumar V. Vadaparty. 1995. Complexity Tailored Design: A New Design Methodology for Databases With Incomplete Information. *J. Comput. Syst. Sci.* 51, 3 (1995), 405–432.

[27] Ravi Jampani, Fei Xu, Mingxi Wu, Luis Leopoldo Perez, Christopher Jermaine, and Peter J Haas. 2008. MCDB: a monte carlo approach to managing uncertain data. In *SIGMOD*.

[28] O. Kennedy and C. Koch. 2010. PIP: A database system for great and small expectations. In *ICDE*.

[29] Phokion G. Kolaitis and Enela Pema. 2012. A dichotomy in the complexity of consistent query answering for queries with two atoms. *Inf. Process. Lett.* 112, 3 (2012), 77–85.

[30] Phokion G. Kolaitis, Enela Pema, and Wang-Chiew Tan. 2013. Efficient Querying of Inconsistent Databases with Binary Integer Programming. *PVLDB* 6, 6 (2013), 397–408.

[31] Egor V. Kostylev and Peter Buneman. 2012. Combining dependent annotations for relational algebra. In *ICDT*.

[32] Paraschos Koutris and Jef Wijsen. 2018. Consistent Query Answering for Primary Keys and Conjunctive Queries with Negated Atoms. In *PODS*.

[33] Willis Lang, Rimma V. Nehme, Eric Robinson, and Jeffrey F. Naughton. 2014. Partial results in database systems. In *SIGMOD*.

[34] Julie Letchner, Christopher Ré, Magdalena Balazinska, and Matthai Philipose. 2009. Access Methods for Markovian Streams. In *ICDE*.

[35] Jian Li, Barna Saha, and Amol Deshpande. 2011. A unified approach to ranking in probabilistic databases. *VLDBJ.* 20, 2 (2011), 249–275.

[36] Leonid Libkin. 2016. SQL's Three-Valued Logic and Certain Answers. *TODS* 41, 1 (2016), 1:1–1:28.

[37] Witold Lipski. 1979. On Semantic Issues Connected with Incomplete Information Databases. *TODS* 4, 3 (1979), 262–296.

[38] Dan Olteanu, Jiewen Huang, and Christoph Koch. 2010. Approximate confidence computation in probabilistic databases. In *ICDE*.

[39] Raymond Reiter. 1986. A sound and sometimes complete query evaluation algorithm for relational databases with null values. *J. ACM* 33, 2 (1986), 349–370.

[40] Anish Das Sarma, Martin Theobald, and Jennifer Widom. 2008. Exploiting Lineage for Confidence Computation in Uncertain and Probabilistic Databases. In *ICDE*.

[41] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. 2011. Probabilistic databases. *Synthesis Lectures on Data Management* 3, 2 (2011), 1–180.

[42] Bruhathi Sundarmurthy, Paraschos Koutris, Willis Lang, Jeffrey F. Naughton, and Val Tannen. 2017. m-tables: Representing Missing Data. In *ICDT*.

[43] Jef Wijsen. 2010. On the first-order expressibility of computing certain answers to conjunctive queries over uncertain databases. In *PODS*.

[44] Jef Wijsen. 2012. Certain conjunctive query answering in first-order logic. *TODS* 37, 2 (2012), 9:1–9:35.
[45] Ying Yang, Niccolò Meneghetti, Ronny Fehling, Zhen Hua Liu, and Oliver Kennedy. 2015. Lenses: An On-demand Approach to ETL. *PVLDB* 8, 12 (2015), 1578–1589.

A PROOFS

Proof of Lemma 1. Proven by substitution of definitions.

$$\text{PW}_i(\mathbb{0}_{\mathcal{K}_W}) = \mathbb{0}_{\mathcal{K}_W}[i] = \mathbb{0}_{\mathcal{K}} \quad \text{PW}_i(\mathbb{1}_{\mathcal{K}_W}) = \mathbb{1}_{\mathcal{K}_W}[i] = \mathbb{1}_{\mathcal{K}}$$

$$\begin{aligned}\text{PW}_i(\vec{k_1} \oplus_{\mathcal{K}_W} \vec{k_2}) &= (\vec{k_1} \oplus_{\mathcal{K}_W} \vec{k_2})[i] = \vec{k_1}[i] \oplus_{\mathcal{K}} \vec{k_2}[i] \\ &= \text{PW}_i(\vec{k_1}) \oplus_{\mathcal{K}} \text{PW}_i(\vec{k_2}) \\ \text{PW}_i(\vec{k_1} \otimes_{\mathcal{K}_W} \vec{k_2}) &= (\vec{k_1} \otimes_{\mathcal{K}_W} \vec{k_2})[i] = \vec{k_1}[i] \otimes_{\mathcal{K}} \vec{k_2}[i] \\ &= \text{PW}_i(\vec{k_1}) \otimes_{\mathcal{K}} \text{PW}_i(\vec{k_2})\end{aligned}$$ □

Proof of Theorem 1. We first prove that the possible world $D = \text{PW}_i(\mathcal{D})$ for some i encoded by D_{UA} is preserved by queries. We have to show that for any query Q we have $h_{det}(Q(D_{UA})) = \text{PW}_i(Q(\mathcal{D}))$. Since a UA-DB is the direct product of two semirings, h_{det} is a homomorphism. Also by construction we have $h_{det}(D_{UA}) = D$. Using these facts and Lemma 1 we get:

$$\begin{aligned}h_{det}(Q(D_{UA})) =& Q(h_{det}(D_{UA})) = Q(D) \\ =& Q(\text{PW}_i(\mathcal{D})) = \text{PW}_i(Q(\mathcal{D}))\end{aligned}$$

For the same argument as above, h_{cert} is a homomorphism, so $Q(h_{cert}(D_{UA})) = h_{cert}(Q(D_{UA}))$. Since according to Theorem 5 queries over labelings preserve the under-approximation of certain annotations this implies the theorem. □

Proof of Lemma 2. $\underline{\oplus_{\mathcal{K}}}$: Based on the definition of $\preceq_{\mathcal{K}}$, if $k \preceq_{\mathcal{K}} k'$ then there exists k'' such that $k \oplus_{\mathcal{K}} k'' = k'$. Thus, $k_3 = k_1 \oplus_{\mathcal{K}} k_1{}'$ and $k_4 = k_2 \oplus_{\mathcal{K}} k_2{}'$ for some $k_1{}'$ and $k_2{}'$. Also, $(k_1 \oplus_{\mathcal{K}} k_2) \preceq_{\mathcal{K}} (k_1 \oplus_{\mathcal{K}} k_2) \oplus_{\mathcal{K}} k''$ for any k'' and we get:

$$k_1 \oplus_{\mathcal{K}} k_2 \preceq_{\mathcal{K}} (k_1 \oplus_{\mathcal{K}} k_2) \oplus_{\mathcal{K}} (k_1{}' \oplus_{\mathcal{K}} k_2{}') = k_3 \oplus_{\mathcal{K}} k_4$$

$\underline{\otimes_{\mathcal{K}}}$: The proof for multiplication $\otimes_{\mathcal{K}}$ is similar.

$$\begin{aligned}&(k_1 \otimes_{\mathcal{K}} k_2) \\ \preceq_{\mathcal{K}} &(k_1 \otimes_{\mathcal{K}} k_2) \oplus_{\mathcal{K}} (k_1 \otimes_{\mathcal{K}} k_2{}') \oplus_{\mathcal{K}} (k_1{}' \otimes_{\mathcal{K}} k_2) \oplus_{\mathcal{K}} (k_1{}' \otimes_{\mathcal{K}} k_2{}') \\ =&(k_1 \oplus_{\mathcal{K}} k_1{}') \otimes_{\mathcal{K}} (k_2 \oplus_{\mathcal{K}} k_2{}') = k_3 \otimes_{\mathcal{K}} k_4\end{aligned}$$ □

Proof of Lemma 3. Recall that $\oplus_{\mathcal{K}_W}$ and $\otimes_{\mathcal{K}_W}$ are defined element-wise and that $\text{CERT}_{\mathcal{K}}(\vec{k}) = \sqcap_{\mathcal{K}}(\vec{k})$. Furthermore, $k_1 \preceq_{\mathcal{K}} k_2$ iff $\exists k' : k_1 \oplus_{\mathcal{K}} k' = k_2$. Consider an arbitrary $\vec{k_1}, \vec{k_2} \in K^W$. Let $k_{glb_1} = \sqcap_{\mathcal{K}}(\vec{k_1})$ and $k_{glb_2} = \sqcap_{\mathcal{K}}(\vec{k_2})$. Based on the definition of $\sqcap_{\mathcal{K}}$ this implies that for any i, $k_{glb_1} \preceq_{\mathcal{K}} \vec{k_1}[i]$ which in turn implies that $\vec{k_1}[i] = k_{glb_1} \oplus_{\mathcal{K}} k'$ for some k'. Analog, we can find a k'' such that $\vec{k_2}[i] = k_{glb_2} \oplus_{\mathcal{K}} k''$.

Superadditivity: Let $k_{glb} = \sqcap_{\mathcal{K}}(\vec{k_1} \oplus_{\mathcal{K}_W} \vec{k_2})$. We are going to prove that $k_{glb_1} \oplus_{\mathcal{K}} k_{glb_2}$ is a lower bound for $(\vec{k_1} \oplus_{\mathcal{K}_W} \vec{k_2})$, i.e., that $\forall i \in W : k_{glb_1} \oplus_{\mathcal{K}} k_{glb_2} \preceq_{\mathcal{K}} (\vec{k_1} \oplus_{\mathcal{K}_W} \vec{k_2})[i]$. Since, k_{glb} is the greatest lower bound this implies that $k_{glb_1} \oplus_{\mathcal{K}} k_{glb_2} \preceq_{\mathcal{K}} k_{glb}$. Consider an arbitrary $i \in W$. Based on the discussion above we have:

$$\begin{aligned}&(\vec{k_1} \oplus_{\mathcal{K}_W} \vec{k_2})[i] = \vec{k_1}[i] \oplus_{\mathcal{K}} \vec{k_2}[i] = k_{glb_1} \oplus_{\mathcal{K}} k' \oplus_{\mathcal{K}} k_{glb_2} \oplus_{\mathcal{K}} k'' \\ =&(k_{glb_1} \oplus_{\mathcal{K}} k_{glb_2}) \oplus_{\mathcal{K}} k' \oplus_{\mathcal{K}} k'' \succeq_{\mathcal{K}} k_{glb_1} \oplus_{\mathcal{K}} k_{glb_2}\end{aligned}$$

Thus, $k_{glb_1} \oplus_{\mathcal{K}} k_{glb_2}$ is a lower bound and since $k_{glb_1} = \text{CERT}_{\mathcal{K}}(\vec{k_1})$ and $k_{glb_2} = \text{CERT}_{\mathcal{K}}(\vec{k_2})$ it follows that $\text{CERT}_{\mathcal{K}}$ is superadditive:

$$\text{CERT}_{\mathcal{K}}(\vec{k_1}) \oplus_{\mathcal{K}} \text{CERT}_{\mathcal{K}}(\vec{k_2}) \preceq_{\mathcal{K}} \text{CERT}_{\mathcal{K}}(\vec{k_1} \oplus_{\mathcal{K}_W} \vec{k_2})$$

Supermultiplicativity: We use an analogous argument to prove supermultiplicativity. Let $k_{glb} = \text{CERT}_{\mathcal{K}}(\vec{k_1} \otimes_{\mathcal{K}} \vec{k_2})$. We will prove that $k_{glb_1} \otimes_{\mathcal{K}} k_{glb_2}$ is a lower bound for $(\vec{k_1} \otimes_{\mathcal{K}_W} \vec{k_2})$ which implies supermultiplicativity. Consider $i \in W$:

$$\begin{aligned}&(\vec{k_1} \otimes_{\mathcal{K}_W} \vec{k_2})[i] = (k_{glb_1} \oplus_{\mathcal{K}} k') \otimes_{\mathcal{K}} (k_{glb_2} \oplus_{\mathcal{K}} k'') \\ =&(k_{glb_1} \otimes_{\mathcal{K}} k_{glb_2}) \oplus_{\mathcal{K}} (k_{glb_1} \otimes_{\mathcal{K}} k'') \oplus_{\mathcal{K}} (k' \otimes_{\mathcal{K}} k_{glb_2}) \oplus_{\mathcal{K}} (k' \otimes_{\mathcal{K}} k'') \\ \succeq_{\mathcal{K}} &(k_{glb_1} \otimes_{\mathcal{K}} k_{glb_2})\end{aligned}$$ □

Proof of Theorem 5. Since $\mathcal{L}$ is a c-sound labeling, for any tuple t we have $\mathcal{L}(t) \preceq_{\mathcal{K}} \text{CERT}_{\mathcal{K}}(\mathcal{D}, t)$. We have to prove that for any t we have $Q(\mathcal{L})(t) \preceq_{\mathcal{K}} \text{CERT}_{\mathcal{K}}(Q(\mathcal{D}), t)$. For that we show that for any $k_1, k_2 \in \mathcal{K}$ and $\vec{k_3}, \vec{k_4} \in K^W$ such that $k_1 \preceq_{\mathcal{K}} \text{CERT}_{\mathcal{K}}(\vec{k_3})$ and $k_2 \preceq_{\mathcal{K}} \text{CERT}_{\mathcal{K}}(\vec{k_4})$, we have $(k_1 \oplus_{\mathcal{K}} k_2) \preceq_{\mathcal{K}} \text{CERT}_{\mathcal{K}}(\vec{k_3} \oplus_{\mathcal{K}_W} \vec{k_4})$ and $k_1 \otimes_{\mathcal{K}} k_2 \preceq_{\mathcal{K}} \text{CERT}_{\mathcal{K}}(\vec{k_3} \otimes_{\mathcal{K}_W} \vec{k_4})$.

$$\begin{aligned}k_1 \oplus_{\mathcal{K}} k_2 &\preceq_{\mathcal{K}} \text{CERT}_{\mathcal{K}}(\vec{k_3}) \oplus_{\mathcal{K}} \text{CERT}_{\mathcal{K}}(\vec{k_4}) && \text{(by Lemma 2)} \\ &\preceq_{\mathcal{K}} \text{CERT}_{\mathcal{K}}(\vec{k_3} \oplus_{\mathcal{K}_W} \vec{k_4}) && \text{(by Lemma 3)} \\ k_1 \otimes_{\mathcal{K}} k_2 &\preceq_{\mathcal{K}} \text{CERT}_{\mathcal{K}}(\vec{k_3}) \otimes_{\mathcal{K}} \text{CERT}_{\mathcal{K}}(\vec{k_4}) && \text{(by Lemma 2)} \\ &\preceq_{\mathcal{K}} \text{CERT}_{\mathcal{K}}(\vec{k_3} \otimes_{\mathcal{K}_W} \vec{k_4}) && \text{(by Lemma 3)}\end{aligned}$$

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

Proof of Theorem 3. Let $\mathcal{L} = \text{label}_{\text{C-TABLE}}(\mathcal{D})$. A tuple t is labeled as certain iff $\phi_{\mathcal{D}}(t)$ is in CNF and $\models \phi_{\mathcal{D}}(t)$, which means the expression $\phi_{\mathcal{D}}$ is a tautology. By definition of C-tables, a tuple t exists in a possible world if $\phi_{\mathcal{D}}(t)$ evaluates to true in that possible world. Thus, t must exist in all possible worlds if $\phi_{\mathcal{D}}(t)$ is a tautology and $\mathcal{L}$ is c-sound. □

Proof of Theorem 4. Trivially holds, since a tuple is certain iff it is not optional and has only one alternative. Even though multiple x-tuples may share an alternative, the independence of x-tuples guarantees that this does not lead to additional certain tuples. □

B PRESERVATION OF C-COMPLETENESS

TI-DBs. We now demonstrate that positive queries preserve c-completeness if the input is a labeling produced by the c-complete labeling scheme $\text{label}_{\text{TI-DB}}$ (Section 6). To show this, we observe that if there exists possible a world for which two $\mathcal{K}_W$-elements $\vec{k}_1$ and $\vec{k}_2$ are both minimal then $\sqcap_{\mathcal{K}}$ commutes with addition and multiplication, and standard $\mathcal{K}$-relational semantics preserves c-completeness.

Lemma 4. *Let $\vec{k}_1, \vec{k}_2 \in K^W$ for some possible world semiring $\mathcal{K}_W$. If there exists $i \in W$ such that $\sqcap_{\mathcal{K}}(\vec{k}_1) = \vec{k}_1[i]$ and $\sqcap_{\mathcal{K}}(\vec{k}_2) = \vec{k}_2[i]$, then the following holds:*

$$\sqcap_{\mathcal{K}}(\vec{k}_1 \oplus_{\mathcal{K}_W} \vec{k}_2) = \sqcap_{\mathcal{K}}(\vec{k}_1) \oplus_{\mathcal{K}} \sqcap_{\mathcal{K}}(\vec{k}_2) = (\vec{k}_1 \oplus_{\mathcal{K}_W} \vec{k}_2)[i]$$

$$\sqcap_{\mathcal{K}}(\vec{k}_1 \otimes_{\mathcal{K}_W} \vec{k}_2) = \sqcap_{\mathcal{K}}(\vec{k}_1) \otimes_{\mathcal{K}} \sqcap_{\mathcal{K}}(\vec{k}_2) = (\vec{k}_1 \otimes_{\mathcal{K}_W} \vec{k}_2)[i]$$

Proof. Recall that PW_i is a homomorphism (Lemma 1), so $(\vec{k}_1 \oplus_{\mathcal{K}_W} \vec{k}_2)[i] = \vec{k}_1[i] \oplus_{\mathcal{K}} \vec{k}_2[i]$ and $\sqcap_{\mathcal{K}}(\vec{k}_j) = \vec{k}_j[i]$ for $j \in \{1, 2\}$. Thus, $(\vec{k}_1 \oplus_{\mathcal{K}_W} \vec{k}_2)[i] = \sqcap_{\mathcal{K}}(\vec{k}_1) \oplus_{\mathcal{K}} \sqcap_{\mathcal{K}}(\vec{k}_2)$. Next, $\sqcap_{\mathcal{K}}(\vec{k}_1 \oplus_{\mathcal{K}_W} \vec{k}_2) = (\vec{k}_1 \oplus_{\mathcal{K}_W} \vec{k}_2)[i]$ which holds if for any $j \neq i \in W$ we have $(\vec{k}_1 \oplus_{\mathcal{K}_W} \vec{k}_2)[i] \preceq_{\mathcal{K}} (\vec{k}_1 \oplus_{\mathcal{K}_W} \vec{k}_2)[j]$. Since $\vec{k}_1[i] = \sqcap_{\mathcal{K}}(\vec{k}_1)$ and $\sqcap_{\mathcal{K}}$ is defined based on the natural order, we know that $\vec{k}_1[i] \preceq_{\mathcal{K}} \vec{k}_1[j]$ and analog for $\vec{k}_2$ we have $\vec{k}_2[i] \preceq_{\mathcal{K}} \vec{k}_2[j]$. Lemma 2 then implies $(\vec{k}_1 \oplus_{\mathcal{K}_W} \vec{k}_2)[i] \preceq_{\mathcal{K}} (\vec{k}_1 \oplus_{\mathcal{K}_W} \vec{k}_2)[j]$. The proof for multiplication is analogous. □

To demonstrate c-completeness preservation for TI-DBs we have to demonstrate that the encoding of a TI-DB as a $\mathcal{K}_W$-database fulfills the precondition of Lemma 4.

Lemma 5. *Let $\mathcal{D}$ be a $\mathcal{K}_W$-database that represents a TI-DB. Then there exists $i \in W$ such that for any tuple t:*

$$\sqcap_{\mathcal{K}}(\mathcal{D}(t)) = \mathcal{D}(t)[i].$$

Proof. Consider the possible world D defined as follows:

$$D(t) = \begin{cases} \sqcap_{\mathcal{K}}(\mathcal{D}(t)) & \text{if } P(t) = 1 \\ \mathbb{0}_{\mathcal{K}} & \text{otherwise} \end{cases}$$

This world exists, because in a TI-DB all tuples with probability $p = 1$ have annotation $\mathbb{1}_{\mathbb{B}}$ in all worlds. Furthermore, since the tuples are independent events, there must exist one world containing no tuples with probability $p < 1$. Let i denote the identifier of this world and denote by $D = \text{PW}_i(\mathcal{D})$. **(Case 1)** $P(t) = 1$ and so $\forall j \in W : \mathcal{D}(t)[i] = \mathcal{D}(t)[j]$. **(Case 2)** $P(t) < 1$ and $D(t) = \mathcal{D}(t)[i] = \mathbb{0}_{\mathcal{K}}$. Because $\forall k \in K : \mathbb{0}_{\mathcal{K}} \preceq_{\mathcal{K}} k$, it follows that $\sqcap_{\mathcal{K}}(\mathcal{D}(t)) = \mathbb{0}_{\mathcal{K}} = \mathcal{D}(t)[i]$. As a result, $\forall t \in \text{TupDom} : \sqcap_{\mathcal{K}}(\mathcal{D}(t)) = D(t) = \mathcal{D}(t)[i]$ □

Lemmas 4 and 5 together imply that our labeling approach preserves c-completeness if the input is a TI-DB.

Corollary 1. *Let $\mathcal{L}$ be a labeling for a TI-DB $\mathcal{D}$ computed as $\text{label}_{TI}(\mathcal{D})$. Then $\mathcal{RA}^+$ over $\mathcal{L}$ preserves c-completeness.*

x-DBs. In general, $\mathcal{RA}^+$ queries over labelings derived from x-DBs using our labeling scheme $\text{label}_{\text{x-DB}}$ from Section 6 do not preserve c-completeness. We present a sufficient condition for a query to preserve c-completeness over such a labeling. To this end, we define *x-keys*, constraints that ensure that alternatives within the scope of an x-tuple are not all identical if projected on a set of attributes A. Since our labeling scheme for x-DBs is c-complete, queries preserve c-completeness unless a result tuple that is certain is derived from multiple *correlated* uncertain input tuples. Since x-tuples from an x-DB are independent of each other, this can only be the case if a result tuple is derived from alternatives of an x-tuple τ from every possible world (i.e., where τ is not optional). Such a situation can be avoided if it is guaranteed that it is impossible for a result tuple to be derived from all alternatives of an x-tuple.

Definition 7 (x-key). *Let R be an x-relation with schema $\mathbf{R}$. A set of attributes $A \subseteq \mathbf{R}$ is called an x-key for R iff*

$$\forall \tau \in R : (\tau \text{ is optional}) \vee |\tau| = 1 \vee (\exists t_1, t_2 \in \tau : t_1[A] \neq t_2[A])$$

An x-key is a set of attributes A such that for any x-tuple τ that is not optional and has more than one alternative, there exists at least two alternatives that differ in A. We prove that for any x-DB $\mathcal{D}$, if a conjunctive, self-join free query Q (a query using selection, projection, and join that accesses no relation more than once) returns at least one x-key per accessed relation, then the query preserves c-completeness.

Theorem 6. *Let $\mathcal{L}$ be a labeling for an x-DB $\mathcal{D}$ computed using $\text{label}_{\text{x-DB}}$. Consider a conjunctive query Q in canonical form $\pi_{\mathcal{A}}(\sigma_\theta(R_1 \times \ldots \times R_n))$ with $R_i \neq R_j$ for all $i \neq j \in \{1, \ldots, n\}$. Query Q preserves c-completeness if $\mathcal{A}$ contains an x-key for every relation R_i accessed by Q.*

Proof. Let $\mathcal{D} = \{R_1, \ldots, R_n\}$ be an x-database, $\mathcal{D}' = \{R'_1, \ldots, R'_n\}$ its encoding as a $\mathbb{B}_W$-database, $\mathcal{L}$ a c-complete labeling for $\mathcal{D}'$ derived using $\text{label}_{\text{x-DB}}$, and Q be a selfjoin-free query of the form $\pi_{\mathcal{A}}(\sigma_\theta(R_1 \times \ldots \times R_n))$ such that $\mathcal{A}$ contains an x-key for every relation R_i for $i \in \{1, \ldots, n\}$. Any selfjoin-free $\mathcal{RA}^+$ query without union can be brought into this form. We have to show that $Q(\mathcal{L})$ is a c-complete labeling for $Q(\mathcal{D}')$. We prove this claim by contradiction. For sake of the contradiction assume that $Q(\mathcal{L})$ is not a c-complete labeling. Then there has to exist a tuple $t \in Q(\mathcal{D}')$

such that $Q(\mathcal{L})(t) = F$ and $\text{CERT}_{\mathbb{B}}(Q(\mathcal{D}')(t)) = T$. Recall that $\oplus_{\mathbb{B}} = \vee$ and $\otimes_{\mathbb{B}} = \wedge$. Unfolding definitions of relational algebra operators over $\mathcal{K}$-relations we get:

$$Q(\mathcal{L})(t) = \bigvee_{u:u[A]=t \wedge \forall i \in \{1,\ldots,n\}: u[R_i]=t_i} \left(\bigwedge_{i=1}^{n} \mathcal{L}(t_i) \right) \wedge \theta(u)$$

$$Q(\mathcal{D}')(t) = \sum_{u:u[A]=t \wedge \forall i \in \{1,\ldots,n\}: u[R_i]=t_i} R_1'(t_1) \otimes_{\mathbb{B}_W} \ldots \otimes_{\mathbb{B}_W} R_n'(t_n) \otimes_{\mathbb{B}_W} \theta(u)$$

Note that for result tuples u of the crossproduct for which $u \not\models \theta$ we have $\theta(u) = F$ (respective $\theta(u) = \mathbb{0}_{\mathbb{B}_W}$). Thus, any monomial (product) corresponding to such a u will evaluate to F ($\mathbb{0}_{\mathbb{B}_W}$). Thus, we can equivalently write the above expressions as shown below where the j values identify monomials for which $u \models \theta$ WLOG assuming that there are $m \in \mathbb{N}$ such monomials. We get: $Q(\mathcal{L})(t) = \bigvee_{\forall j \in \{1,\ldots,m\}} \left(\bigwedge_{i=1}^{n} \mathcal{L}(t_{j_i}) \right)$ and $Q(\mathcal{D}')(t) = \sum_{\forall j \in \{1,\ldots,m\}} \prod_{i=1}^{n} R_i'(t_{j_i})$. We use b_{j_i} to denote $\mathcal{L}(t_{j_i})$ and $\vec{k_{j_i}}$ to denote $R_i'(t_{j_i})$. Based on our assumption we know: $\bigvee_{\forall j \in \{1,\ldots,m\}} \left(\bigwedge_{i=1}^{n} b_{j_i} \right) = F$. So this can only be the case if for every $j \in \{1,\ldots,m\}$ there exists $f \in \{1,\ldots,n\}$ such that $b_{j_f} = F$. For any $j \in \{1,\ldots,m\}$ let min_j denote the smallest such f, i.e., the first element in the j^{th} conjunct that is false and let t_{min_j} denote the corresponding tuple. Based on the fact that $\mathcal{L} = \text{label}_{\text{x-DB}}(\mathcal{D}')$ and that $\text{label}_{\text{x-DB}}$ is c-complete, we know that if $\mathcal{L}(t_{min_j}) = F$ then t_{min_j} is not certain. We will use this fact to derive a contradiction with the assumption $\text{CERT}_{\mathbb{B}}(Q(\mathcal{D}')(t)) = T$. For that, we partition the set of monomials from $Q(\mathcal{D}')(t)$ into two subsets M_1 and ${M_1}^C$ where M_1 contains the identifiers j of all monomials such that $min_j = 1$ and ${M_1}^C$ contains all remaining monomials. We will show that $\text{CERT}_{\mathbb{B}}(\sum_{j \in M_1} \prod_{i=1}^{n} k_{j_i}) = F$, then $\text{CERT}_{\mathbb{B}}(\sum_{j \in {M_1}^C} \prod_{i=1}^{n} k_{j_i}) = F$, and finally $\text{CERT}_{\mathbb{B}}(Q(\mathcal{D}')(t)) = \text{CERT}_{\mathbb{B}}(\sum_{j \in M_1} \prod_{i=1}^{n} k_{j_i} \oplus_{\mathbb{B}_W} \sum_{j \in {M_1}^C} \prod_{i=1}^{n} k_{j_i}) = F$ which is the contradiction we wanted to derive.

First, consider $\sum_{j \in M_1} \prod_{i=1}^{n} k_{j_i}$. Since $\otimes_{\mathbb{B}} = \wedge$ and $\otimes_{\mathbb{B}_W}$ is defined as point-wise application of $\wedge$ to a vector $\vec{k} \in \mathbb{B}_W$ we have $\vec{k} \otimes_{\mathbb{B}_W} \vec{k'} \preceq_{\mathbb{B}_W} \vec{k}$ for any $\vec{k}$ and $\vec{k'}$. Thus, $\sum_{j \in M_1} \prod_{i=1}^{n} k_{j_i} \preceq_{\mathbb{B}_W} \sum_{j \in M_1} k_{j_1}$. We show $\text{CERT}_{\mathbb{B}}(\sum_{j \in M_1} k_{j_1}) = F$ from which follows $\text{CERT}_{\mathbb{B}}(\sum_{j \in M_1} \prod_{i=1}^{n} k_{j_i}) = F$.

By construction we have that t_{j_1} is not certain for all j in M_1. Now consider the set of x-tuples from R_1 for which the tuples t_{j_1} are alternatives. WLOG let $\tau_1, \ldots, \tau_l$ be these x-tuples. Now consider an arbitrary x-tuple τ from this set and let $s_1, \ldots, s_o$ be its alternatives that are present in M_1. We know that none of the s_i are certain based on the fact that alternatives are disjoint events and x-tuples are independent of each other. We distinguish 2 cases: either τ is optional or τ is not optional. In the latter case based on the fact that the query result contains an x-key for R_1 we know that there exists at least one alterative s of τ that is neither in M_1 nor in ${M_1}^C$. To see why this is the case observe that its presence in M_1 would violate the x-key while by construction ${M_1}^C$ only contains tuples t from R_1' which are certain. Next we construct a possible world $w \in W$ from $\mathcal{D}$ which does not contain any of the t_{j_1} which means that $k_{j_1}[w] = F$. In turn, this implies that $\sum_{j \in M_1} k_{j_1} = F$. We construct w as follows: for every x-tuple τ from $\tau_1, \ldots, \tau_l$ we either include no alternative of τ if the x-tuple is optional or an alternative that is not present in M_1. Now further partition ${M_1}^C$ into two subsets: M_2 which contains all monomials for which $min_j = 2$ and ${M_2}^C$ for all remaining monomials. Then using an argument symmetric to the one given for M_1 above we can construct a possible world for which $\sum_{j \in M_2} \prod_{i=1}^{n} t_{j_i}[w] = F$ and, thus, $\text{CERT}_{\mathbb{B}}(\sum_{j \in M_2} \prod_{i=1}^{n} k_{j_i}) = F$. Because the x-tuples from M_1 and M_2 are from different relations there is no overlap between these sets of x-tuples. Based on the independence of x-tuples in x-DBs this implies that we can also construct a possible world w where $\sum_{j \in M_1} \prod_{i=1}^{n} k_{j_i}[w] \oplus_{\mathbb{B}_W} \sum_{j \in M_2} \prod_{i=1}^{n} k_{j_i}[w] = F$ and, thus, $\text{CERT}_{\mathbb{B}}(\sum_{j \in M_1} \prod_{i=1}^{n} k_{j_i} \oplus_{\mathbb{B}_W} \sum_{j \in M_2} \prod_{i=1}^{n} k_{j_i}) = F$. We can now continue this construction to include M_3, M_4, and so on. Note that we are guaranteed that M_n contains all monomials that will be left over at this point, because we started from the observation that at least one k in every monomial corresponds to a tuple t which is not certain. It follows that $\text{CERT}_{\mathbb{B}}(Q(\mathcal{D}')(t)) = \text{CERT}_{\mathbb{B}}\left(\sum_{o=1}^{n} \left(\sum_{j \in M_o} \prod_{i=1}^{n} k_{j_i}\right)\right) = F$ which contradicts our assumption that $\text{CERT}_{\mathbb{B}}(Q(\mathcal{D}')(t)) = T$ and thus concludes the proof. □

C DATASETS

Building Violations: Building violations issued by Chicago's Department of Buildings from 2006 to the present. [7] **Shootings in Buffalo:** Shootings in Buffalo during the year 2016. [8] **Business Licenses:** Current and active business licenses issued by the Department of Business Affairs and Consumer Protection. [9] **Chicago Crime:** Reported incidents of crime occurring from 2001 to present. [10] **Contracts:** Contracts and modifications awarded by the City of Chicago since 1993. [11] **Food Inspections:** Inspections of restaurants and other food establishments in Chicago from January 1, 2010 to the present. [12] **Graffiti Removal:** All graffiti removal requests,

[7] https://data.cityofchicago.org/Buildings/Building-Violations/22u3-xenr
[8] http://projects.buffalonews.com/charts/shootings/index.html
[9] https://data.cityofchicago.org/Community-Economic-Development/Business-Licenses-Current-Active/uupf-x98q
[10] https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-present/ijzp-q8t2
[11] https://data.cityofchicago.org/Administration-Finance/Contracts/rsxa-ify5
[12] https://data.cityofchicago.org/Health-Human-Services/Food-Inspections/4ijn-s7e5

open and closed, since January 1, 2011. [13] **Building Permits:** All types of structural permits in San Francisco from Jan 1, 2013-Feb 25th 2018. [14] **Public Library Survey:** Over 25 years worth of research publications about the Public Libraries Survey. [15] **NHANES:** Family level information on income sources, monthly income, and family cash assets. [16]

[13] https://data.cityofchicago.org/Service-Requests/311-Service-Requests-Graffiti-Removal/hec5-y4x5
[14] https://www.kaggle.com/aparnashastry/building-permit-applications-data/data
[15] https://www.imls.gov/research-evaluation/data-collection/public-libraries-survey/explore-pls-data/pls-data
[16] https://wwwn.cdc.gov/Nchs/Nhanes/2013-2014/INQ_H.htm