Efficient Uncertainty Tracking for Complex Queries with Attribute-level Bounds

Su Feng, Boris Glavic
Illinois Institute of Technology
Chicago, United States
sfeng14@hawk.iit.edu.bglavic@iit.edu

Aaron Huber, Oliver A. Kennedy
University at Buffalo
Buffalo, United States
ahuber@buffalo.edu,okennedy@buffalo.edu

ABSTRACT

Incomplete and probabilistic database techniques are principled methods for coping with uncertainty in data. Unfortunately, the class of queries that can be answered efficiently over such databases is severely limited, even when advanced approximation techniques are employed. We introduce *attribute-annotated uncertain databases* (AU-DBs), an uncertain data model that annotates tuples and attribute values with bounds to compactly approximate an incomplete database. AU-DBs are closed under relational algebra with aggregation using an efficient evaluation semantics. Using optimizations that trade accuracy for performance, our approach scales to complex queries and large datasets, and produces accurate results.

KEYWORDS

uncertainty, incomplete databases, annotations, aggregation

ACM Reference Format:
Su Feng, Boris Glavic and Aaron Huber, Oliver A. Kennedy. 2021. Efficient Uncertainty Tracking for Complex Queries with Attribute-level Bounds. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Uncertainty arises naturally in many application domains due to data entry errors, sensor errors and noise [39], uncertainty in information extraction and parsing [54], ambiguity from data integration [8, 35, 50], and heuristic data wrangling [15, 24, 62]. Analyzing uncertain data without accounting for its uncertainty can create hard to trace errors, with severe real world implications. Incomplete database techniques [22] have emerged as a principled way to model and manage uncertainty in data[1]. An incomplete database models uncertainty by encoding a set of possible worlds, each of which is one possible state of the real world. Under the commonly used *certain answer semantics* [5, 36], a query returns the set of answer tuples *guaranteed* to be in the result, regardless of which possible world is correct. Many computational problems are intractable over incomplete databases. Even approximations (e.g., [28, 31, 42]) are often still not efficient enough, are insufficiently expressive, or exclude useful answers [22, 25]. Thus, typical database users resort to resolving uncertainty using heuristics and then treating the result as a deterministic database [62]. In other words, this approach selects one possible world for analysis, ignoring all other possible worlds. We refer to this approach as *selected-guess* query processing (*SGQP*). SGQP is efficient, since the resulting dataset is deterministic, but discards all information about uncertainty, with the associated potential for severe negative consequences.

Example 1. *Alice is tracking the spread of COVID-19 and wants to use data extracted from the web to compare infection rates in population centers of varying size. Fig. 1a (top) shows example (unreliable) input data. Parts of this data are trustworthy, while other parts are ambiguous;* $[v_1, \ldots, v_n]$ *denotes an uncertain value (e.g., conflicting data sources) and* **null** *indicates that the value is completely unknown (i.e., any value from the attribute's domain could be correct).* $\mathcal{D}$ *encodes a set of possible worlds, each a deterministic database that represents one possible state of the real world. Alice's ETL heuristics select (e.g., based on the relative trustworthiness of each source) one possible world* D_{SG} *(Fig. 1b) by selecting a deterministic value for each ambiguous input (e.g., an infection rate of 3% for Los Angeles). Alice next computes the average rate by locale size.*

```
SELECT size, avg(rate) AS rate
FROM locales GROUP BY size
```

Querying D_{SG} *may produce misleading results, (e.g., an 18% average infection rate for cities). Conversely, querying* $\mathcal{D}$ *using* certain answer semantics *[36] produces no results at all. Although there must exist a result tuple for metros, the uncertain infection rate of Los Angeles makes it impossible to compute one certain result tuple. Furthermore, the data lacks a size for Sacramento, which can contribute to any result, rendering all* `rate` *values uncertain, even for result tuples with otherwise perfect data. An alternative is the* possible answer semantics, *which enumerates all possible results. However, the number of possible results is inordinately large (e.g., Fig. 1a, bottom). With only integer percentages there are nearly 600 possible result tuples for towns alone. Worse, enumerating either the (empty) certain or the (large) possible results is expensive (coNP-hard/NP-hard).*

Neither certain answers nor possible answer semantics are meaningful for aggregation over uncertain data (e.g., see [22] for a deeper discussion), further encouraging the (mis-)use of SGQP. One possible solution is to develop a special query semantics for aggregation, either returning hard bounds on aggregate results (e.g., [6, 13, 49]), or computing expectations (e.g., [40, 49]) when probabilities are available. Unfortunately, for such approaches, aggregate queries and non-aggregate queries return incompatible results, and thus

[1]Probabilistic databases [57] generalize incomplete databases with a probability distribution over possible worlds. We focus on contrasting with the former for simplicity, but many of the same cost and expressivity limitations also affect probabilistic databases.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

$\mathcal{D}$

locale	rate	size
Los Angeles	[3%,4%]	metro
Austin	18%	[city,metro]
Houston	14%	metro
Berlin	[1%,3%]	[town,city]
Sacramento	1%	**null**
Springfield	**null**	town

$Q(\mathcal{D})$

size	rate
village	0%
village	1%
town	0%
…	…
metro	12%

(a) X-DB

D_{SG}

locale	rate	size
Los Angeles	3%	metro
Austin	18%	city
Houston	14%	metro
Berlin	3%	town
Sacramento	1%	town
Springfield	5%	town

$Q(D_{SG})$

size	rate
metro	8.5%
city	18%
town	3%

(b) A possible world

D_{AU}

locale	rate	size	$\mathbb{N}^3$
Los Angeles	[3%/3%/4%]	metro	(1,1,1)
Austin	18%	$[city/city/metro]$	(1,1,1)
Houston	14%	metro	(1,1,1)
Berlin	[1%/3%/3%]	$[town/town/city]$	(1,1,1)
Sacramento	1%	$[village/town/metro]$	(1,1,1)
Springfield	[0%/5%/100%]	town	(1,1,1)

$Q(D_{AU})$

size	spop	$\mathbb{N}^3$
metro	[6%/8.5%/12%]	(1,1,1)
city	[7.33%/18%/18%]	(0,1,1)
town	[0.33%/4%/100%]	(1,1,1)
$[village/village/metro]$	1%	(0,0,1)

(c) Possible AU-DB Encoding (based on D_{SG})

Figure 1: Example incomplete database and query results.

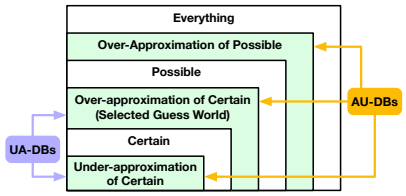


Figure 2: AU-DBs sandwich certain answers between an under-approximation and the SGW and over-approximate possible answers.

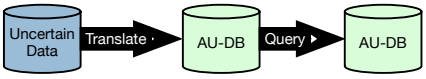


Figure 3: AU-DBs are created from uncertain data, possibly represented using incomplete or probabilistic data models.

the class of queries supported by these approaches is typically quite limited. For example, most support only a single aggregation as the last operation of a query. Worse, these approaches are often still computationally intractable. Another class of solutions represents aggregation results symbolically (e.g., [9, 27]). Evaluating queries over symbolic representations is often tractable (PTIME), but the result may be hard to interpret for a human, and extracting tangible information (e.g., expectations) from symbolic instances is again hard. In summary, prior work on processing complex queries involving aggregation over incomplete (and probabilistic) databases (i) only supports limited query types; (ii) is often expensive; and/or (iii) returns results that are hard to interpret.

We argue that for uncertain data management to be accepted by practitioners it has to be competitive with the selected-guess approach in terms of (i) performance and (ii) the class of supported queries (e.g., aggregation). In this work, we present AU-DBs, an annotated data model that approximates an incomplete database by annotating one of its possible worlds. As an extension of the recently proposed UA-DBs [25], AU-DBs generalize and subsume current standard practices (i.e., SGQP). An AU-DB is built on a selected world, supplemented with two sets of annotations: lower and upper bounds both on attributes, and on tuple multiplicities. Thus, each tuple in an AU-DB may encode a set of tuples from each possible world, each with attribute values falling within the provided bounds. In addition to being a strict generalization of SGQP, an AU-DB relation also includes enough information to bound both the certain and possible answers as illustrated in Fig. 2.

Example 2. *Fig. 1c shows an AU-DB constructed from one possible world D_{SG} of $\mathcal{D}$. We refer to this world as the* selected-guess world *(SGW). Each uncertain attribute is replaced by a 3-tuple, consisting of a lower bound, the value of the attribute in the SGW, and an upper bound, respectively. Additionally, each tuple is annotated with a 3-tuple consisting of a lower bound on its multiplicity across all possible worlds, its multiplicity in the SGW, and an upper bound on its multiplicity. For instance, Los Angeles is known to have an infection rate between 3% and 4% with a guess (e.g., based on a typical ETL approach like giving priority to a trusted source) of 3%. The query result is shown in Fig. 1c. The first row of the result indicates that there is exactly one record for metro areas (i.e., the upper and lower multiplicity bounds are both 1), with an average rate between 6% and 12% (with a selected guess of 8.5%). Similarly, the second row of the result indicates that there might (i.e., lower-bound of 0) exist one record for cities with a rate between 7.33% and 18%. This is a strict generalization of how users presently interact with uncertain data, as ignoring everything but the middle element of each 3-tuple gets us the SGW. However, the AU-DB also captures the data's uncertainty.*

As we will demonstrate, AU-DBs have several beneficial properties that make them a good fit for dealing with uncertain data:

Efficiency. Query evaluation over AU-DBs is PTIME, and by using novel optimizations that compact intermediate results to trade precision for performance, our approach scales to large datasets and complex queries. While still slower than SGQP, AU-DBs are practical, significantly outperforming alternative uncertain data management systems, especially for queries involving aggregation.

Query Expressiveness. The under- and over-approximations encoded by an AU-DB are preserved by queries from the full-relational algebra with multiple aggregations ($\mathcal{RA}^{agg}$). Thus, AU-DBs are closed under $\mathcal{RA}^{agg}$, and are (to our knowledge) the first incomplete database approach to support complex, multi-aggregate queries.

Compatibility. Like UA-DBs [25], an AU-DB can be constructed from many existing incomplete and probabilistic data models, including C-tables [36] or tuple-independent databases [57], making it possible to re-use existing approaches for exposing uncertainty in data (e.g., [12, 14, 15, 31, 42, 51, 62]). Moreover, although this paper focuses on bag semantics, our model is defined for the same class of semiring-annotated databases [32] as UA-DBs [25] which include, e.g., set semantics, security-annotations, and provenance.

Compactness. As observed elsewhere [33, 34, 47], under-approximating certain answers for non-monotone queries (like aggregates) requires over-approximating possible answers. A single AU-DB tuple can encode a large number of tuples, and can compactly approximate possible results. This over-approximation is interesting in its own right to deal with missing data in the spirit of [44, 46, 58].

Simplicity. AU-DBs use simple bounds to convey uncertainty, as opposed to the more complex symbolic formulas of m-tables [58] or tensors [9]. Representing uncertainty as ranges has been shown to lead to better decision-making [43]. AU-DBs can be integrated into uncertainty-aware user interfaces, e.g., Vizier [16, 43].

2 RELATED WORK

We build on prior research in uncertain databases, specifically, techniques for approximating certain answers and aggregation.

Approximations of Certain Answers. Queries over incomplete databases typically use certain answer semantics [5, 33, 34, 36, 47] first defined in [48]. Computing certain answers is coNP-complete [5, 36] (data complexity) for relational algebra. Several techniques for computing an under-approximation (subset) of certain answers have been proposed. Reiter [52] proposed a PTIME algorithm for positive existential queries. Guagliardo and Libkin [33, 34, 47] proposed a scheme for full relational algebra for Codd- and V-tables, and also studied bag semantics [21, 34]. Feng et. al. [25] generalized this approach to new query semantics through Green et. al.'s $\mathcal{K}$-relations [32]. m-tables [58] compactly encode large amounts of possible tuples, allowing for efficient query evaluation. However, this requires complex symbolic expressions which necessitate schemes for approximating certain answers. Consistent query answering (CQA) [12, 14] computes the certain answers to queries over all possible repairs of a database that violates a set of constraints. Variants of this problem have been studied extensively (e.g., [19, 41, 42]) and several combinations of classes of constraints and queries permit first-order rewritings [30, 31, 59, 60]. Geerts et. al. [31] study first-order under-approximations of certain answers in the context of CQA. Notably, AU-DBs build on the approach of [25] (i.e., a selected guess and lower bounds), adding an upper bound on possible answers (e.g., as in [34]) to support aggregations, and bound attribute-level uncertainty with ranges instead of nulls.

Aggregation in Incomplete/Probabilistic Databases. While aggregation of uncertain data has been studied extensively (see [26] for a comparison of approaches), general solutions remain an open problem [22]. A key challenge lies in defining a meaningful semantics, as aggregates over uncertain data frequently produce empty certain answers [18]. An alternative semantics adopted for CQA and ontologies [6, 13, 18, 29, 55] returns per-attribute bounds over all possible results [13] instead of a single *certain* answer. In contrast to prior work, we use bounds as a fundamental building block of our data model. Because of the complexity of aggregating uncertain data, most approaches focus on identifying tractable cases and producing statistical moments or other lossy representations [4, 17, 20, 38, 40, 49, 56, 61]. Even this simplified approach is expensive (often NP-hard, depending on the query class), and requires approximation. Statistical moments like expectation may be meaningful as final query answers, but are less useful if the result is to be subsequently queried (e.g., `HAVING` queries [53]).

Efforts to create a lossless symbolic encoding closed under aggregation [27, 45] exist, supporting complex multi-aggregate queries and a wide range of statistics (e.g, bounds, samples, or expectations). However, even factorizable encodings like aggregate semimodules [9] usually scale in the size of the aggregate input and not the far smaller aggregate output, making these schemes impractical. AU-DBs are also closed under aggregation, but replace lossless encodings of aggregate outputs with lossy, but compact *bounds*.

A third approach, exemplified by MCDB [37] queries sampled possible worlds. In principle, this approach supports arbitrary queries, but is significantly slower than SGQP [25], only works when probabilities are available, and only supports statistical measures that can be derived from samples (i.e., moments and epsilon-delta bounds).

A similarly general approach [44, 58] determines which parts of a query result over incomplete data are uncertain, and whether the result is an upper or lower bound. However, this approach tracks incompleteness coarsely (horizontal table partitions). AU-DBs are more general, combining both fine-grained uncertainty information (individual rows and attribute values) and coarse-grained information (one row in a AU-DB may encode multiple tuples).

3 NOTATION AND BACKGROUND

We now review $\mathcal{K}$-relations, a generalization of classical incomplete databases called incomplete $\mathcal{K}$-relations, and the UA-DBs model extended here. A database schema $\textsc{Sch}(D) = \{\textsc{Sch}(R_1), \ldots, \textsc{Sch}(R_n)\}$ is a set of relation schemas $\textsc{Sch}(R_i) = \langle A_1, \ldots, A_n \rangle$. The arity $arity(\textsc{Sch}(R))$ of $\textsc{Sch}(R)$ is the number of attributes in $\textsc{Sch}(R)$. Assume a universal domain of attribute values $\mathbb{D}$. A tuple with schema $\textsc{Sch}(R)$ is an element from $\mathbb{D}^{arity(\textsc{Sch}(R))}$. We assume the existence of a total order over the elements of $\mathbb{D}$.[2]

3.1 K-Relations

The generalization of incomplete databases we use here is based on **$\mathcal{K}$-relations** [32]. In this framework, relations are annotated with elements from the domain K of a (commutative) semiring $\mathcal{K} = \langle K, +_{\mathcal{K}}, \cdot_{\mathcal{K}}, \mathbb{1}_{\mathcal{K}}, \mathbb{0}_{\mathcal{K}} \rangle$, i.e., a mathematical structure with commutative and associative addition ($+_{\mathcal{K}}$) and product ($\cdot_{\mathcal{K}}$) operations where $+_{\mathcal{K}}$ distributes over $\cdot_{\mathcal{K}}$ and $k \cdot_{\mathcal{K}} \mathbb{0}_{\mathcal{K}} = \mathbb{0}_{\mathcal{K}}$ for all $k \in K$. An n-nary $\mathcal{K}$-relation is a function that maps tuples 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 $\mathbb{0}_{\mathcal{K}}$. 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 $R(t)$. In this work we are interested in bag semantics, which can be encoded using the semiring of natural numbers with standard addition and multiplication $\langle \mathbb{N}, +, \times, 0, 1 \rangle$ to annotate each tuple with its multiplicity. We discuss the applicability of our framework to a larger class of semirings in an accompanying technical report [26].

Operators of positive relational algebra ($\mathcal{RA}^+$) over $\mathbb{N}$-relations combine input annotations using + and $\cdot$.

$$\textbf{Union: } (R_1 \cup R_2)(t) = R_1(t) + R_2(t)$$

$$\textbf{Join: } (R_1 \bowtie R_2)(t) = R_1(t[\textsc{Sch}(R_1)]) \cdot R_2(t[\textsc{Sch}(R_2)])$$

$$\textbf{Projection: } (\pi_U(R))(t) = \sum_{t=t'[U]} R(t')$$

$$\textbf{Selection: } (\sigma_\theta(R))(t) = R(t) \cdot \theta(t)$$

We will make use of the so called *natural order* $\preceq_{\mathcal{K}}$ for a semiring $\mathcal{K}$ which is the standard order $\leq$ of natural numbers for $\mathbb{N}$.

3.2 Incomplete K-Relations

Instead of using the classical set-based definition of incomplete databases and certain answers, we apply the generalization from [25] called incomplete $\mathcal{K}$-relations, specifically the $\mathbb{N}$-relations that

[2] The order over $\mathbb{D}$ may be arbitrary, but range bounds are most useful when the order makes sense for the domain values (e.g., the ordinal scale of an ordinal attribute).

model bag-incomplete databases. An incomplete $\mathbb{N}$-database is a set of $\mathbb{N}$-databases $\mathcal{D} = \{D_1, \ldots, D_n\}$ called possible worlds. Queries over an incomplete $\mathbb{N}$-database use possible world semantics: The result of a query Q over an incomplete $\mathbb{N}$-database $\mathcal{D}$ is the set of possible worlds derived by evaluating Q over every world in $\mathcal{D}$.

$$Q(\mathcal{D}) \coloneqq \{\ Q(D) \mid D \in \mathcal{D}\ \} \qquad \text{(possible world semantics)}$$

Generalizing certain and possible answers, the certain (possible) multiplicity of a tuple t in an incomplete $\mathbb{N}$-database $\mathcal{D}$ is the minimum (maximum) multiplicity of the tuple across all possible worlds:

$$\textsc{cert}_{\mathbb{N}}(\mathcal{D}, t) \coloneqq \min(\{D(t) \mid D \in \mathcal{D}\})$$
$$\textsc{poss}_{\mathbb{N}}(\mathcal{D}, t) \coloneqq \max(\{D(t) \mid D \in \mathcal{D}\})$$

3.3 UA-Databases

Using $\mathcal{K}$-relations, Feng et al. [25] introduced *UA-DBs* (*uncertainty-annotated databases*) which, in the case of semiring $\mathbb{N}$, encode a tuple level under- and over-approximation of the certain multiplicity of tuples from an incomplete $\mathbb{N}$-database $\mathcal{D}$. In a bag UA-DB (semiring $\mathbb{N}$), every tuple is annotated with a pair $[c, d] \in \mathbb{N}^2$ where d is the tuple's multiplicity in a selected possible world $D_{sg} \in \mathcal{D}$ i.e., $d = D_{sg}(t)$ and c is an under-approximation of the tuple's certain multiplicity, i.e., $c \leq \textsc{cert}_{\mathbb{N}}(\mathcal{D}, t) \leq d$. The selected world D_{sg} is called the selected-guess world (SGW). Formally, these pairs $[c, d]$ are elements from a semiring $\mathbb{N}_{UA}$ which is the direct product of semiring $\mathbb{N}$ with itself ($\mathbb{N}^2$). Operations in the product semiring $\mathbb{N}^2 = \langle\ \mathbb{N}^2, +_{\mathbb{N}^2}, \cdot_{\mathbb{N}^2}, \mathbb{0}_{\mathbb{N}^2}, \mathbb{1}_{\mathbb{N}^2}\ \rangle$ are defined pointwise, e.g., $[k_1, k_1{}'] \cdot_{\mathbb{N}^2} [k_2, k_2{}'] = [k_1 \cdot k_2, k_1{}' \cdot k_2{}']$. UA-DBs are created from an incomplete or probabilistic data source by selecting a SGW D_{sg} and generating an under-approximation $\mathcal{L}$ of the certain multiplicty $\textsc{cert}_{\mathbb{N}}$ of tuples. In the UA-DB, the annotation of each tuple t is set to: $D_{UA}(t) \coloneqq [\mathcal{L}(t), D_{sg}(t)]$. UA-DBs constructed in this fashion are said to *bound* $\mathcal{D}$ through $\mathcal{L}$ and D_{sg}. Feng et al. [25] discussed how to create UA-DBs that bound C-tables, V-tables, and x-DBs. [25, Theorem 1] shows that standard $\mathbb{N}^2$-relational query semantics preserves bounds under $\mathcal{RA}^+$ queries, i.e., if the input bounds an incomplete $\mathbb{N}$-database $\mathcal{D}$, then the result bounds $Q(\mathcal{D})$.

4 OVERVIEW

Query evaluation over UA-DBs is efficient (PTIME data complexity and experimentally shown to have performance comparable to SGQP). However, UA-DBs may not be as precise and concise as possible since uncertainty is only recorded at the tuple-level. For example, the encoding of the town tuple in Fig. 1a needs just shy of 600 uncertain tuples, one for each combination of possible values of the uncertain `size` and `rate` attributes. Additionally, UA-DB query semantics does not support non-monotone operations like aggregation and set difference, as this requires an over-approximation of possible answers.

We address both shortcomings in AU-DBs through two changes relative to UA-DBs: (i) Tuple annotations include an upper bound on the tuple's possible multiplicity; and (ii) Attribute values become 3-tuples, with lower- and upper-bounds and a selected-guess (SG) value. These building blocks, range-annotated scalar expressions and $\mathbb{N}_{AU}$-relations, are formalized in Sec. 5 and 6, respectively.

Supporting both attribute-level and tuple-level uncertainty creates ambiguity in how tuples should be represented. As noted above, the tuple for towns is certain (i.e., deterministically present) and has uncertain (i.e., multiple-possible values) attributes, but could also be expressed as 600 tuples with certain attribute values whose existence is uncertain. This ambiguity makes it challenging to define what it means for an AU-DB to bound an incomplete database, a problem we resolve in Sec. 6.3 by defining *tuple matchings* that relate tuples in an AU-DB to those of a possible world. An AU-DB bounds an incomplete database if such a mapping exists for every possible world. This ambiguity is also problematic for group-by aggregation, as aggregating a relation with uncertain group-by attribute values may admit multiple, equally viable output AU-relations. We propose a specific grouping strategy in Sec. 8.1 that mirrors SGW query evaluation, and show that it behaves as expected.

5 SCALAR EXPRESSIONS

Consider a domain $\mathbb{D}$ that consists of $\mathbb{R}$ and the boolean values ($\bot$ and $\top$). Furthermore, let $\mathbb{V}$ denote a countable set of variables. For any variable $x \in \mathbb{V}$, x is an expression and for any constant $c \in \mathbb{D}$, c is an expression. If e_1, e_2 and e_3 are expressions, then …

$$e_1 \wedge e_2 \qquad e_1 \vee e_2 \qquad \neg e_1 \qquad e_1 = e_2 \qquad e_1 \neq e_2 \qquad e_1 \leq e_2$$
$$e_1 + e_2 \qquad e_1 \cdot e_2 \qquad \textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3$$

are also expressions. Given an expression e, we denote the variables of e by $\textsc{vars}(e)$. We will also use $\neq$, $\geq$, $<$, $-$, and $>$ since these operators can be defined using the expression syntax above, e.g., $e_1 > e_2 = \neg(e_1 \leq e_2)$. For an expression e, given a valuation φ that maps variables from $\textsc{vars}(e)$ to constants from $\mathbb{D}$, the expression evaluates to a constant $[\![e]\!]_\varphi$ from $\mathbb{D}$. The semantics of these expressions are standard (see [26] for explicit definitions).

5.1 Incomplete Expression Evaluation

We now define evaluation of expressions over incomplete valuations, which are sets of valuations. Each valuation in such a set, called a possible world, represents one possible input for the expression. The semantics of expression evaluation are then defined using possible worlds semantics: the result of evaluating an expression e over an incomplete valuation $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ denoted as $[\![e]\!]_\Phi$ is the set of results obtained by evaluating e over each φ_i using deterministic expression evaluation semantics: $[\![e]\!]_\Phi \coloneqq \{[\![e]\!]_\varphi \mid \varphi \in \Phi\}$. Consider an expression $e \coloneqq x + y$ and an incomplete valuation $\Phi = \{(x = 1, y = 4), (x = 2, y = 4), (x = 1, y = 5)\}$. We get $[\![e]\!]_\Phi = \{1 + 4, 2 + 4, 1 + 5\} = \{5, 6\}$.

5.2 Range-Annotated Domains

We now define *range-annotated values*, which are domain values that are annotated with an interval that bounds the value from above and below. We define an expression semantics for valuations that maps variables to range-annotated values and then prove that if the input bounds an incomplete valuation, then the range-annotated output produced by this semantics bounds the possible outcomes of the incomplete expression.

Definition 1. *Let $\mathbb{D}$ be a domain and let $\leq$ denote a total order over its elements. Then the* range-annotated domain *$\mathbb{D}_I$ is defined as:*

$$\left\{[c^{\downarrow}/c^{sg}/c^{\uparrow}] \mid c^{\downarrow}, c^{sg}, c^{\uparrow} \in \mathbb{D} \wedge c^{\downarrow} \leq c^{sg} \leq c^{\uparrow}\right\}$$

A value $c = [c^{\downarrow}/c^{sg}/c^{\uparrow}]$ from $\mathbb{D}_I$ encodes a value $c^{sg} \in \mathbb{D}$ and two values ($c^{\downarrow}$ and $c^{\uparrow}$) that bound c^{sg} from below and above. We call a value $c \in \mathbb{D}_I$ *certain* if $c^{\downarrow} = c^{sg} = c^{\uparrow}$. Observe, that the definition requires that for any $c \in \mathbb{D}_I$ we have $c^{\downarrow} \leq c^{sg} \leq c^{\uparrow}$. We use valuations that map the variables of an expression to elements from $\mathbb{D}_I$ to bound incomplete valuations.

Definition 2. *A* range-annotated valuation $\tilde{\varphi}$ *for an expression* e *is a mapping* $\textsc{vars}(e) \to \mathbb{D}_I$. *Given an incomplete valuation* Φ *and a range-annotated valuation* $\tilde{\varphi}$ *for* e, *we say that* $\tilde{\varphi}$ *bounds* Φ *iff*

$$\forall x \in \textsc{vars}(e) : \forall \varphi \in \Phi : \tilde{\varphi}(x)^{\downarrow} \leq \varphi(x) \leq \tilde{\varphi}(x)^{\uparrow}$$
$$\exists \varphi \in \Phi : \forall x \in \textsc{vars}(e) : \varphi(x) = \tilde{\varphi}(x)^{sg}$$

Consider the incomplete valuation $\Phi = \{(x = 1), (x = 2), (x = 3)\}$. The range-annotated valuation $x = [0/2/3]$ is a bound for Φ, while $x = [0/2/2]$ is not a bound. We now define a semantics for evaluating expressions over range-annotated valuations. We then demonstrate that this semantics preserves bounds.

Definition 3. *Let* e *be an expression. Given a range valuation* $\tilde{\varphi} : \textsc{vars}(e) \to \mathbb{D}_I$, *we define* $\tilde{\varphi}^{sg}(x) \coloneqq \tilde{\varphi}(x)^{sg}$. *The result of expression* e *over* $\tilde{\varphi}$ *denoted as* $[\![e]\!]_{\tilde{\varphi}}$ *is defined as:*

$$[\![x]\!]_{\tilde{\varphi}} \coloneqq [\tilde{\varphi}(x)^{\downarrow}/\tilde{\varphi}(x)^{sg}/\tilde{\varphi}(x)^{\uparrow}] \qquad [\![c]\!]_{\tilde{\varphi}} \coloneqq [c/c/c]$$

For any of the following expressions we define $[\![e]\!]_{\tilde{\varphi}}{}^{sg} \coloneqq [\![e]\!]_{\tilde{\varphi}^{sg}}$. *Let* $[\![e_1]\!]_{\tilde{\varphi}} = a$, $[\![e_2]\!]_{\tilde{\varphi}} = b$, *and* $[\![e_3]\!]_{\tilde{\varphi}} = c$. *All expressions omitted below are defined point-wise (e.g.,* $[\![e_1 + e_2]\!]_{\tilde{\varphi}}{}^{\downarrow} \coloneqq a^{\downarrow} + b^{\downarrow}$*).*

$$[\![\neg e_1]\!]_{\tilde{\varphi}}{}^{\downarrow} \coloneqq \neg\, a^{\uparrow} \qquad [\![\neg e_1]\!]_{\tilde{\varphi}}{}^{\uparrow} \coloneqq \neg\, a^{\downarrow}$$
$$[\![e_1 \cdot e_2]\!]_{\tilde{\varphi}}{}^{\downarrow} \coloneqq \min(\ a^{\uparrow} \cdot b^{\uparrow},\ a^{\uparrow} \cdot b^{\downarrow},\ a^{\downarrow} \cdot b^{\uparrow},\ a^{\downarrow} \cdot b^{\downarrow}\ )$$
$$[\![e_1 \cdot e_2]\!]_{\tilde{\varphi}}{}^{\uparrow} \coloneqq \max(\ a^{\uparrow} \cdot b^{\uparrow},\ a^{\uparrow} \cdot b^{\downarrow},\ a^{\downarrow} \cdot b^{\uparrow},\ a^{\downarrow} \cdot b^{\downarrow}\ )$$
$$[\![e_1 \leq e_2]\!]_{\tilde{\varphi}}{}^{\downarrow} \coloneqq a^{\uparrow} \leq b^{\downarrow} \qquad [\![e_1 = e_2]\!]_{\tilde{\varphi}}{}^{\downarrow} \coloneqq a^{\uparrow} = b^{\downarrow} \wedge b^{\uparrow} = a^{\downarrow}$$
$$[\![e_1 \leq e_2]\!]_{\tilde{\varphi}}{}^{\uparrow} \coloneqq a^{\downarrow} \leq b^{\uparrow} \qquad [\![e_1 = e_2]\!]_{\tilde{\varphi}}{}^{\uparrow} \coloneqq a^{\downarrow} \leq b^{\uparrow} \wedge b^{\downarrow} \leq a^{\uparrow}$$
$$[\![\textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3]\!]_{\tilde{\varphi}}{}^{\downarrow} \coloneqq \begin{cases} b^{\downarrow} & \text{if } a^{\downarrow} = a^{\uparrow} = \top \\ c^{\downarrow} & \text{if } a^{\downarrow} = a^{\uparrow} = \bot \\ \min(b^{\downarrow}, c^{\downarrow}) & \text{else} \end{cases}$$
$$[\![\textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3]\!]_{\tilde{\varphi}}{}^{\uparrow} \coloneqq \begin{cases} b^{\uparrow} & \text{if } a^{\downarrow} = a^{\uparrow} = \top \\ c^{\uparrow} & \text{if } a^{\downarrow} = a^{\uparrow} = \bot \\ \max(b^{\uparrow}, c^{\uparrow}) & \text{else} \end{cases}$$

Assuming that an input range-annotated valuation bounds an incomplete valuation, we need to prove that the output of range-annotated expression evaluation also bounds the possible outcomes.

Definition 4. *A value* $c \in \mathbb{D}_I$ *bounds a set of values* $S \subseteq \mathbb{D}$ *if:*

$$\forall c_i \in S : c^{\downarrow} \leq c_i \leq c^{\uparrow} \qquad \exists c_i \in S : c_i = c^{sg}$$

Theorem 1. *Let* $\tilde{\varphi}$ *be a range-annotated valuation that bounds an incomplete valuation* e *for an expression* e, *then* $[\![e]\!]_{\tilde{\varphi}}$ *bounds* $[\![e]\!]_{\Phi}$.

Proof Sketch. Proven by straightforward induction over the structure of formulas. We present the full proof in [26]. □

Conditional range-annotated expressions are bound-preserving for any totally-ordered domain. For categorical values where no sensible order can be defined, we impose an arbitrary order.

6 AU-DBS

We define attribute-annotated uncertain databases (AU-DBs) as a special type of $\mathcal{K}$-relation over range-annotated domains and demonstrate how to bound an incomplete $\mathcal{K}$-relation using this model. Afterwards, we define a metric for how precise the bounds of an incomplete $\mathcal{K}$-database encoded by an AU-DB are, and define a query semantics for AU-DBs that preserves bounds. We limit the discussion to semiring $\mathbb{N}$ (bag semantics). See [26] for the generalization to other semirings including $\mathbb{B}$ (set semantics).

6.1 $\mathbb{N}_{AU}$-relations

In addition to allowing for range-annotated values, AU-DBs also differ from UA-DBs in that they encode an upper bound of the possible annotation of each tuple. We use $\mathbb{N}^3$ annotations to encode for each tuple: a lower bound on the certain multiplicity of the tuple, the multiplicity of the tuple in the SGW, and an over-approximation of the tuple's possible multiplicity.

Definition 5. *The range-annotated multiplicity domain* $\mathbb{N}_{AU}$ *is defined as:*

$$\{(k^{\downarrow}, k, k^{\uparrow}) \mid k^{\downarrow}, k, k^{\uparrow} \in \mathbb{N} \wedge k^{\downarrow} \leq k \leq k^{\uparrow}\}$$

We use $\mathbb{N}_{AU}$ *to denote semiring* $\mathbb{N}^3$ *restricted to elements from* $\mathbb{N}_{AU}$.

Note that $\mathbb{N}_{AU}$ is a semiring since when combining two elements of $\mathbb{N}_{AU}$ with $+_{\mathbb{N}^3}$ and $\cdot_{\mathbb{N}^3}$, the result $(k^{\downarrow}, k^{sg}, k^{\uparrow})$ fulfills the requirement $k^{\downarrow} \leq k^{sg} \leq k^{\uparrow}$. This is the case because addition and multiplication in $\mathbb{N}$ preserve order [25], and these operations in $\mathbb{N}^3$ are pointwise application of $+$ and $\cdot$.

Definition 6. *A* $\mathbb{N}_{AU}$*-relation of arity* n *is a function* $\mathbf{R} : \mathbb{D}_I{}^n \to \mathbb{N}_{AU}$ *with finite support* $\{\boldsymbol{t} \mid \mathbf{R}(\boldsymbol{t}) \neq (0,0,0)\}$.

6.2 Extracting Selected-Guess Worlds

$\mathbb{N}_{AU}$-relations permit multiple tuples with identical values in the selected-guess world (SGW). We extract the selected-guess world encoded by a $\mathbb{N}_{AU}$-relation by grouping tuples based on their SG values and summing up their SG multiplicities.

Definition 7. *We lift operator sg from values to tuples:* $sg : \mathbb{D}_I{}^n \to \mathbb{D}^n$, *i.e., given an AU-DB tuple* $\boldsymbol{t} = \langle\, c_1, \ldots, c_n \,\rangle$, $\boldsymbol{t}^{sg} \coloneqq \langle\, c_1{}^{sg}, \ldots, c_n{}^{sg} \,\rangle$. *For a* $\mathbb{N}_{AU}$*-relation* $\mathbf{R}$, $\mathbf{R}^{sg}$, *the SGW encoded by* $\mathbf{R}$, *is then defined as:*

$$\mathbf{R}^{sg}(t) \coloneqq \sum_{\boldsymbol{t}^{sg} = t} (\mathbf{R}(\boldsymbol{t}))^{sg}$$

Example 3. *Fig. 4a shows an instance of a* $\mathbb{N}_{AU}$*-relation* R *where each attribute is a triple showing the lower bound, selected-guess, and upper bound of the value. Each tuple is annotated by a triple showing the lower multiplicity bound, selected-guess multiplicity, and the upper multiplicity bound of the tuple. For example, the first tuple represents a tuple* $\langle\, 1, 1 \,\rangle$ *that appears at least twice in every possible world (its lower multiplicity bound is* 2*), appears twice in the SGW, and appears no more than thrice in any world. Fig. 4b shows the SGW encoded by this AU-DB which is computed by summing up the multiplicities of tuples with identical SG values. For instance, the first two tuples both represent tuple* $\langle\, 1, 1 \,\rangle$ *in the SGW and their annotations sum up to* 5 *(i.e., the tuple* $\langle\, 1, 1 \,\rangle$ *appears five times). Conversely, the first two tuples of* D_2 *match different AU-DB tuples.*

A	B	$\mathbb{N}^3$
[1/1/1]	[1/1/1]	(2,2,3)
[1/1/1]	[1/1/3]	(2,3,3)
[1/2/2]	[3/3/3]	(1,1,1)

(a) Example AU-DB instance

	D_1			$\mathbb{N}$
	A	**B**		
t_1	1	1		5
t_2	2	3		1

	D_2		
	A	**B**	$\mathbb{N}$
t_3	1	1	2
t_4	1	3	2
t_5	2	3	1

(b) Worlds D_1 (SGW) and D_2

Figure 4: Example AU-DB relation and two worlds it bounds

6.3 Encoding Bounds

We now define what it means for an AU-DB to bound an incomplete $\mathbb{N}$-relation. For that, we first define bounding of deterministic tuples by range-annotated tuples.

Definition 8. *Let $\boldsymbol{t}$ be a range-annotated tuple with schema $\langle a_1, \ldots, a_n \rangle$ and t be a tuple with the same schema as $\boldsymbol{t}$. We say that $\boldsymbol{t}$ bounds t written as $t \sqsubseteq \boldsymbol{t}$ iff*

$$\forall i \in \{1, \ldots, n\} : \boldsymbol{t}.a_i{}^{\downarrow} \leq t.a_i \leq \boldsymbol{t}.a_i{}^{\uparrow}$$

One AU-DB tuple may bound multiple conventional tuples and vice versa. We define *tuple matchings* as a way to match the multiplicities of tuples of a $\mathbb{N}_{AU}$-database (or relation) with that of one possible world of an incomplete $\mathbb{N}$-database (or relation). Based on tuple matchings we then define how to bound possible worlds.

Definition 9. *Let $\mathbf{R}$ be an n-ary AU-relation and R an n-ary relation. A* tuple matching $\mathcal{TM}$ *for $\mathbf{R}$ and R is a function $(\mathbb{D}_I)^n \times \mathbb{D}^n \to \mathbb{N}$. s.t.the following conditions hold:*

$$\forall \boldsymbol{t} \in \mathbb{D}_I{}^n : \forall t \not\sqsubseteq \boldsymbol{t} : \mathcal{TM}(\boldsymbol{t}, t) = 0 \quad \forall t \in \mathbb{D}^n : \sum_{\boldsymbol{t} \in \mathbb{D}_I{}^n} \mathcal{TM}(\boldsymbol{t}, t) = R(t)$$

A tuple matching distributes the multiplicity of a tuple from R over one or more matching tuples from $\mathbf{R}$. Multiple tuples from an AU-DB may encode the same tuple when the multidimensional rectangles of their attribute-level range annotations overlap, as with the first two tuples of the AU-DB in Fig. 4 and the SGW.

Definition 10. *Given an n-ary AU-DB relation $\mathbf{R}$ and an n-ary deterministic $\mathbb{N}$-relation R (a possible world), relation $\mathbf{R}$ bounds R (denoted $R \sqsubset \mathbf{R}$) iff there exists a tuple matching $\mathcal{TM}$ for $\mathbf{R}$ and R s.t.*

$$\forall \boldsymbol{t} \in \mathbb{D}_I{}^n : \sum_{t \in \mathbb{D}^n} \mathcal{TM}(\boldsymbol{t}, t) \geq \mathbf{R}(\boldsymbol{t})^{\downarrow} \textbf{ and } \sum_{t \in \mathbb{D}^n} \mathcal{TM}(\boldsymbol{t}, t) \leq \mathbf{R}(\boldsymbol{t})^{\uparrow}$$

Having defined when a possible world is bound by a $\mathbb{N}_{AU}$-relation, we are ready to define bounding of incomplete $\mathbb{N}$-relations.

Definition 11. *Given an incomplete $\mathbb{N}$-relation $\mathcal{R}$ and a $\mathbb{N}_{AU}$-relation $\mathbf{R}$, we say that $\mathbf{R}$ bounds $\mathcal{R}$, written as $\mathcal{R} \sqsubset \mathbf{R}$ iff*

$$\forall R \in \mathcal{R} : R \sqsubset \mathbf{R} \qquad \exists R \in \mathcal{R} : R = \mathbf{R}^{sg}$$

These definitions are extended to databases in the obvious way.

Example 4. *Consider the AU-DB of Ex. 3 and the worlds of Fig. 4b. The AU-DB bounds these worlds, since there exist tuple matchings that provide a lower and an upper bound for the annotations of the tuples of each world. For instance, denoting the tuples from this example as*

$$\boldsymbol{t}_1 \coloneqq \langle\, [1/1/1], [1/1/1] \,\rangle \qquad \boldsymbol{t}_2 \coloneqq \langle\, [1/1/1], [1/1/3] \,\rangle$$
$$\boldsymbol{t}_3 \coloneqq \langle\, [1/2/2], [3/3/3] \,\rangle$$

Tuple matching $\mathcal{TM}_1$ (shown below) bounds D_1.

$$\mathcal{TM}_1(\boldsymbol{t}_1, t_1) = 2 \quad \mathcal{TM}_1(\boldsymbol{t}_2, t_1) = 3 \quad \mathcal{TM}_1(\boldsymbol{t}_3, t_1) = 0$$
$$\mathcal{TM}_1(\boldsymbol{t}_1, t_2) = 0 \quad \mathcal{TM}_1(\boldsymbol{t}_2, t_2) = 0 \quad \mathcal{TM}_1(\boldsymbol{t}_3, t_2) = 1$$

6.4 Tightness of Bounds

Given an incomplete database, there are many possible AU-DBs that bound it of varying tightness. For instance, both $\mathbf{t}_1 \coloneqq \langle\, [1/15/100] \,\rangle$ and $\mathbf{t}_2 \coloneqq \langle\, [13/14/15] \,\rangle$ bound tuple $\langle\, 15 \,\rangle$, but intuitively the bounds provided by $\mathbf{t}_2$ are tighter. We develop a metric for the tightness of the approximation provided by an AU-DB and prove that finding an AU-DB that maximizes tightness is intractable. Given two AU-DBs $\mathbf{D}$ and $\mathbf{D}'$ that both bound an incomplete $\mathbb{N}$-database $\mathcal{D}$, $\mathbf{D}$ is a tighter bound than $\mathbf{D}'$ if the set of worlds bound by $\mathbf{D}$ is a subset of the set of worlds bound by $\mathbf{D}'$.

Definition 12. *Consider two $\mathbb{N}_{AU}$-databases $\mathbf{D}$ and $\mathbf{D}'$ over the same schema S. We say that $\mathbf{D}$ is at least as tight as $\mathbf{D}'$, written as $\mathbf{D} \preceq_I \mathbf{D}'$, if for all $\mathbb{N}$-databases D with schema S we have:*

$$D \sqsubset \mathbf{D} \rightarrow D \sqsubset \mathbf{D}'$$

We say that $\mathbf{D}$ is strictly tighter than $\mathbf{D}'$, written as $\mathbf{D} \prec_I \mathbf{D}'$ if $\mathbf{D} \preceq_I \mathbf{D}'$ and there exists $D \sqsubset \mathbf{D}'$ with $D \not\sqsubset \mathbf{D}$. Furthermore, we call $\mathbf{D}$ a maximally tight bound for an incomplete $\mathbb{N}$-database $\mathcal{D}$ if:

$$\mathcal{D} \sqsubset \mathbf{D} \qquad \nexists \mathbf{D}' : \mathcal{D} \sqsubset \mathbf{D}' \wedge \mathbf{D}' \prec_I \mathbf{D}$$

This notion of tightness is well-defined even if the data domain $\mathbb{D}$ is infinite. In general AU-DBs that are tighter bounds are preferable. However, computing a maximally tight bound is intractable.

Theorem 2. *Let $\mathcal{D}$ be an incomplete $\mathbb{N}$-database encoded as a C-table [36]. Computing a maximally tight bound $\mathbf{D}$ for $\mathcal{D}$ is NP-hard.*

Proof Sketch. Proven by reduction from 3-colorability [26]. We construct a C-table with one tuple and encode the 3-colorability of the input graph G in the local condition of the tuple, i.e., the tuple is possible iff G is 3-colorable. The maximally tight upper multiplicity bound for this tuple is 1 iff graph G is 3-colorable. □

7 AU-DB QUERY SEMANTICS

We now introduce a semantics for $\mathcal{RA}^+$ queries over AU-DBs that preserves bounds, i.e., if the input of a query Q bounds an incomplete $\mathbb{N}$-database $\mathcal{D}$, then the output bounds $Q(\mathcal{D})$. AU-DBs follow the standard K-relational semantics for all $\mathcal{RA}^+$ operations except selection. The revised selection operator (described below) remains linear in the size of its input, thus implying PTIME query evaluation [10, 26]. Recall the standard semantics for evaluating selection conditions over $\mathbb{N}$-relations: The multiplicity of a tuple t in the result of a selection $\sigma_\theta(R)$ is computed by multiplying $R(t)$ with $\theta(t)$. $\theta(t)$ is defined as a function $\mathbb{B} \to \{0, 1\}$ that returns 1 if θ evaluates to true on t and 0 otherwise. In $\mathbb{N}_{AU}$-relations, t is a tuple of range-annotated values. Using the range-annotated semantics for expressions from Sec. 5, selection conditions evaluate to triples of boolean values $\mathbb{B}^3$. For example, $[F/F/T]$ means that the condition may be false in some worlds, is false in the SGW, and may be true in some worlds. To define selection semantics compatible with $\mathbb{N}$-relational query semantics, these triples must be translated to elements of $\mathbb{N}_{AU}$.

Definition 13. *We define function* $\mathcal{M}_{\mathbb{N}} : \mathbb{B}^3 \to \mathbb{N}^3$ *as:*

$$\mathcal{M}_{\mathbb{N}}([b_1/b_2/b_3]) := (k_1, k_2, k_3) \text{ where } k_i := \begin{cases} 1 & \text{if } b_i = true \\ 0 & \text{otherwise} \end{cases}$$

We use the mapping of range-annotated boolean values to $\mathbb{N}_{AU}$ elements to define evaluation of selection conditions.

Definition 14. *Let* $\mathbf{t}$ *be a range-annotated tuple, and* θ *be a boolean condition over variables representing attributes from* $\mathbf{t}$*. Furthermore, let* $\tilde{\varphi}_{\mathbf{t}}$ *denote the range-annotated valuation that maps each variable to the corresponding value from* $\mathbf{t}$*. We define* $\theta(\mathbf{t})$ *as:*

$$\theta(\mathbf{t}) := \mathcal{M}_{\mathbb{N}}([\![\theta]\!]_{\tilde{\varphi}_{\mathbf{t}}})$$

Example 5. *Consider the example* $\mathbb{N}_{AU}$*-relation* R *shown below. The single tuple* $\mathbf{t}$ *of this relation binds at least one tuple in every possible world, two in the SGW, and no more than three in any world.*

A	B	$\mathbb{N}_{AU}$
[1/2/3]	2	(1, 2, 3)

To evaluate query $Q := \sigma_{A=2}(R)$ *over this relation, we first evaluate the expression* $A = 2$ *using range-annotated expression evaluation semantics. We get* $[1/2/3] = [2/2/2]$ *which evaluates to* $[F/T/T]$*. Using* $\mathcal{M}_{\mathbb{N}}$*, this value is mapped to* $(0, 1, 1)$*. To calculate the annotation of the tuple in the result of the selection we then multiply these values with the tuple's annotation in* R *and get:*

$$R(\mathbf{t}) \cdot_{\mathbb{N}_{AU}} \theta(\mathbf{t}) = (1, 2, 3) \cdot (0, 1, 1) = (0, 2, 3)$$

Thus, the tuple may not exist in every world of the query result, appears twice in the SGW, and occurs at most thrice.

For this query semantics to be useful, it has to preserve bounds. Formally, a query semantics is *bound preserving* for a class of queries C if for every query $Q \in C$, incomplete $\mathbb{N}$-database $\mathcal{D}$ and $\mathbb{N}_{AU}$-database $\mathbf{D}$ that bounds $\mathcal{D}$ ($\mathcal{D} \sqsubset \mathbf{D}$), we have $Q(\mathcal{D}) \sqsubset Q(\mathbf{D})$.

Theorem 3. $\mathbb{N}_{AU}$*-relational semantics preserves bounds for* $\mathcal{RA}^+$.

Proof Sketch. Intuitively, this is true because expressions are evaluated using range-annotated expression semantics, which preserves bounds on values, and queries are evaluated in a direct-product semiring $\mathbb{N}_{AU}$ whose operations are defined point-wise. Concretely, we use [25, Lemma 2]: addition and multiplication preserve order in $\mathbb{N}$ (e.g., if $k_1 \leq k_2$ and $k_3 \leq k_4$ then $k_1 + k_3 \leq k_2 + k_4$), and induction over the structure of a query under the assumption that $\mathbf{D}$ bounds $\mathcal{D}$ to prove the theorem. (full proof in [26]) □

7.1 Join Optimizations

One performance bottleneck of query evaluation over AU-DBs is that equi-joins may degenerate into cross products if the bounds of join attribute values are loose. For example, this happens when the join attribute bounds of every tuple from the LHS and RHS overlap. We now develop an optimization for joins that splits each input relation $\mathbf{R}$ of the join into two parts: $\text{split}^{sg}(\mathbf{R})$ and $\text{split}^{\uparrow}(\mathbf{R})$ as follows. If the equijoin is on attributes $\langle a_1, \ldots, a_n \rangle$:

$$\text{split}^{sg}(\mathbf{R})(\mathbf{t})^{\downarrow} := \begin{cases} \mathbf{R}(\mathbf{t})^{\downarrow} & \text{if } \forall i : \mathbf{t}.a_i{}^{\downarrow} = \mathbf{t}.a_i{}^{sg} = \mathbf{t}.a_i{}^{\uparrow} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{split}^{sg}(\mathbf{R})(\mathbf{t})^{sg} := \text{split}^{sg}(\mathbf{R})(\mathbf{t})^{\uparrow} := \mathbf{R}(\mathbf{t})^{sg}$$

$$\text{split}^{\uparrow}(\mathbf{R})(\mathbf{t})^{\downarrow} := \text{split}^{\uparrow}(\mathbf{R})(\mathbf{t})^{sg} := 0$$

$$\text{split}^{\uparrow}(\mathbf{R})(\mathbf{t})^{\uparrow} := \mathbf{R}(\mathbf{t})^{\uparrow}$$

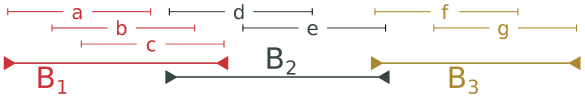


Figure 5: An example of join compression with a compression size of 3: Tuples $a - g$ are compressed into bins $B_1 - B_3$.

This operation ensures (i) that any relation R bound by $\mathbf{R}$ is also bound by both $\text{split}^{\uparrow}(\mathbf{R})$ and $\text{split}^{sg}(\mathbf{R}) \cup \text{split}^{\uparrow}(\mathbf{R})$, (ii) that the join attributes of $\text{split}^{sg}(\mathbf{R})$ are all certain, and (iii) that $\text{split}^{sg}(\mathbf{R})^{sg} = \mathbf{R}^{sg}$. Properties (i) and (iii) allow us to compute a bounding AU-relation for the join $\mathbf{R}_1 \bowtie \mathbf{R}_2$ as

$$(\text{split}^{sg}(\mathbf{R}_1) \bowtie \text{split}^{sg}(\mathbf{R}_2)) \cup (\text{split}^{\uparrow}(\mathbf{R}_1) \bowtie \text{split}^{\uparrow}(\mathbf{R}_2))$$

Thanks to property (ii), the first join is a classical relational join and can be evaluated efficiently. The latter join remains expensive, but tuples in this join can be compressed: Based on a join compression size CT and join attribute A, we construct CT buckets $[b_i{}^{\downarrow}, b_i{}^{\uparrow}]$ over the domain of A such that (i) for every tuple in $\mathbf{t} \in \text{split}^{\uparrow}(\mathbf{R}_i)$, there exists an i such that $b_i{}^{\downarrow} \leq \mathbf{t}.A^{\downarrow} \leq \mathbf{t}.A^{\uparrow} \leq b_i{}^{\uparrow}$, and (ii) the number of tuples contained in this way by each bucket is roughly uniform. The compressed $\text{split}^{\uparrow}(\mathbf{R}_i)$ relation has exactly CT tuples (one for each bucket) with attribute bounds that cover all tuples assigned to the bucket, lower and selected-guess multiplicities of 0, and an upper-bound multiplicity of the number of tuples assigned to the bucket. For example, Fig. 5 shows the overlapping ranges of the join attributes of seven tuples ($a - g$). To create a compressed representation of size 3 (i.e., $CT = 3$), these tuples are partitioned into 3 roughly-equally-sized bins (B_1, B_2, B_3), and the resulting attribute bounds are selected as the maximum upper and lower bounds of the component tuples. The compression size allows a trade-off between faster query evaluation and tighter bounds. We prove that the resulting join operation is correct in [26].

7.2 Creating AU-DBs

One advantage that AU-DBs inherit from UA-DBs is that other models for incomplete and probabilistic databases can be translated into this model such that the generated AU-DB bounds the input uncertain database, albeit with reduced precision and no probabilities. Alternatively, lenses [16, 62] can be used to directly generate AU-DBs that encode the uncertainty introduced by curation heuristics. We present translation schemes for tuple-independent databases, x-DBs, and V-table in [26] and prove that they preserve bounds.

8 AGGREGATION

We now introduce a bound preserving semantics for aggregation over $\mathbb{N}_{AU}$-relations that is based on results from [9] that we review below. We leave a generalization to other semirings to future work. See [26] for a discussion of the challenges involved with that. Importantly, our semantics has PTIME data complexity.

Aggregation Monoids. The semantics for aggregation queries over $\mathcal{K}$-relations from [9] deals with aggregation functions that are commutative monoids $\langle M, +_M, \mathbb{0}_M \rangle$, i.e., where the values from M that are the input to aggregation are combined through an operation $+_M$ which has a neutral element $\mathbb{0}_M$. Abusing notation, we will use M to both denote the monoid and its domain. Standard aggregation functions **sum**, **min**, **max**, and **count** are monoids. **avg** can be derived from **sum** and **count**. As an example, consider the monoids

for **sum** and **min**: $\mathsf{SUM} := \langle\, \mathbb{R}, +, 0 \,\rangle$ and $\mathsf{MIN} := \langle\, \mathbb{R}, \min, \infty \,\rangle$. For $M \in \{\mathsf{SUM}, \mathsf{MIN}, \mathsf{MAX}\}$ (**count** uses SUM), we define a corresponding monoid M_I using range-annotated expression semantics (Sec. 5). Note that this gives us aggregation functions which can be applied to range-annotated values and are bound preserving, i.e., the result of the aggregation function bounds all possible results for any set of values bound by the inputs. For example, **min** is expressed as $\min(v, w) := \textbf{if } v \leq w \textbf{ then } v \textbf{ else } w$.

Lemma 1. *SUM_I, MIN_I, MAX_I are monoids.*

Proof Sketch. Proven by unfolding of definitions to show that the addition operations of these monoids is pointwise application of the operation of the monoids they originated from. □

Semimodules. Aggregation over $\mathcal{K}$-relations has to take the annotations of tuples into account for computing aggregation results. For instance, consider an $\mathbb{N}$-relation $R(A)$ with tuples $\langle\, 30 \,\rangle \mapsto 2$ and $\langle\, 40 \,\rangle \mapsto 3$, (i.e., respectively 2 and 3 duplicates). The sum over A should be $30 \cdot 2 + 40 \cdot 3 = 180$. More generally speaking, we need an operation $*_M : \mathbb{N} \times M \to M$ that combines the tuple multiplicities with domain values. As observed in [9] this operation has to fulfill the algebraic laws of semimodules. Note that $*$ is well-defined for $\mathbb{N}$ and all of the monoids we consider:

$$k *_{\mathsf{SUM}} m = k \cdot m \qquad k *_{\mathsf{MIN}} m = k *_{\mathsf{MAX}} m = \begin{cases} m & \text{if } k \neq 0 \\ 0 & \text{else} \end{cases}$$

Unfortunately, as the following lemma shows, it is not possible to use semimodules for aggregation over $\mathbb{N}_{AU}$-relations, because such semimodules, if they exist, cannot be bound preserving.

Lemma 2. *$*_{\mathbb{N}_{AU},\mathsf{SUM}}$, if it is well-defined, is not bound preserving.*

Proof Sketch. Assume that this semimodule exists and is bound preserving. We lead this assumption to a contradiction by deriving two different results for the expression $(1, 1, 2) *_{\mathbb{N}_{AU},\mathsf{SUM}} [0/0/0]$ using semimodule laws: (i) the obvious $[0/0/0]$ and (ii) $[-1/0/1] = (1, 1, 2) *_{\mathbb{N}_{AU},\mathsf{SUM}} ([-1/-1/-1] + [1/1/1])$ (full proof in [26]). □

In spite of this negative result, not everything is lost. Observe that it not necessary for the operation that combines semiring elements (tuple annotations) with elements of the aggregation monoid to follow semimodule laws. After all, what we care about is that the operation is bound-preserving. Below we define operations $\circledast_M$ that are not semimodules, but are bound-preserving. To achieve bound-preservation we can rely on the bound-preserving expression semantics we have defined in Sec. 5. For example, since $*_{\mathsf{SUM}}$ is multiplication, we can define $\circledast_{\mathsf{SUM}}$ using multiplication for range-annotated values. This approach of computing the bounds as the minimum and maximum over all pair-wise combinations of value and tuple-annotation bounds also works for MIN and MAX. In [26] we prove that $\circledast_M$ is in fact bound preserving.

Definition 15. *Consider a monoid $M \in \{\mathsf{SUM}, \mathsf{MIN}, \mathsf{MAX}\}$. Let $[m^{\downarrow}/m/m^{\uparrow}]$ be a range-annotated value from $\mathbb{D}_I$ and $(k^{\downarrow}, k, k^{\uparrow}) \in \mathbb{N}_{AU}$. We define $(k^{\downarrow}, k, k^{\uparrow}) \circledast_M [m^{\downarrow}/m/m^{\uparrow}] =$*

$$\begin{aligned} &[min(k^{\downarrow} *_M m^{\downarrow}, k^{\downarrow} *_M m^{\uparrow}, k^{\uparrow} *_M m^{\downarrow}, k^{\uparrow} *_M m^{\uparrow}),\\ &\ k *_M m,\\ &\ max(k^{\downarrow} *_M m^{\downarrow}, k^{\downarrow} *_M m^{\uparrow}, k^{\uparrow} *_M m^{\downarrow}, k^{\uparrow} *_M m^{\uparrow})] \end{aligned}$$

8.1 Bound-Preserving Aggregation

We now define a bound preserving aggregation semantics based on the $\circledast_M$ operations. Since AU-DBs can be used to encode an arbitrary number of groups as a single tuple, we need to decide how to trade conciseness of the representation for accuracy. Furthermore, we need to ensure that the aggregation result in the SGW is encoded by the result. There are many possible strategies for how to group possible aggregation results. In this paper, we limit the discussion to a default strategy that we introduce next (see [26] for a general treatment). Afterwards, we demonstrate how to calculate group-by attribute and aggregation result ranges for output tuples.

Grouping Strategy. Our *grouping strategy* takes as input an n-ary $\mathbb{N}_{AU}$-relation $\mathbf{R}$ and a list of group-by attributes G and returns a pair $(\mathcal{G}, \alpha)$ where $\mathcal{G}(G, \mathbf{R})$ is a set of output tuples — one for every SG group (an input tuple's group-by values in the SGW), and α assigns each input tuple to one output based on its SG group-by values. Even if the SG annotation of an input tuple is 0, we still use its SG values to assign it to an output tuple. Only tuples that are not possible (annotated with $\mathbb{0}_{\mathbb{N}_{AU}} = (0, 0, 0)$) are not considered. Since output tuples are identified by their SG group-by values, we will use these values to identify elements from $\mathcal{G}$.

Definition 16. *Consider a query $Q := \gamma_{G,f(A)}(\mathbf{R})$. Let $\mathbf{t} \in \mathbb{D}_I{}^n$. The default grouping strategy $\mathbb{G}_{def} := (\mathcal{G}, \alpha)$ is defined as:*

$$\mathcal{G}(G, \mathbf{R}) := \{t.G \mid \exists \mathbf{t} : \mathbf{t}^{sg} = t \wedge \mathbf{R}(\mathbf{t}) \neq \mathbb{0}_{\mathbb{N}_{AU}}\} \qquad \alpha(\mathbf{t}) := \mathbf{t}.G^{sg}$$

For instance, consider three tuples $\mathbf{t}_1 := \langle\, [1/2/2] \,\rangle$ and $\mathbf{t}_2 := \langle\, [2/2/4] \,\rangle$ and $\mathbf{t}_3 := \langle\, [2/3/4] \,\rangle$ of a relation $\mathbf{R}(A)$. Furthermore, assume that $\mathbf{R}(\mathbf{t}_1) = (1, 1, 1)$, $\mathbf{R}(\mathbf{t}_2) = (0, 0, 1)$, and $\mathbf{R}(\mathbf{t}_3) = (0, 0, 3)$. Grouping on A, the default strategy will generate two output groups g_1 for SG group (2) and g_2 for SG group (3). Based on their SG group-by values, α assigns $\mathbf{t}_1$ and $\mathbf{t}_2$ to g_1 and $\mathbf{t}_3$ to g_2.

Aggregation Semantics. We now introduce an aggregation semantics based on this grouping strategy. For simplicity we define aggregation without group-by as a special case of aggregation with group-by (the only difference is how tuple annotations are handled). We first define how to construct a result tuple $\mathbf{t}_g$ for each output group g returned by the grouping strategy and then present how to calculate tuple annotations. The construction of an output tuple is divided into two steps: (i) determine range annotations for the group-by attributes and (ii) determine range annotations for the aggregation function result attributes. To ensure that all possible groups that an input tuple $\mathbf{t}$ with $\alpha(\mathbf{t}) = g$ belongs to are contained in $\mathbf{t}_g.G$ we have to merge the group-by attribute bounds of all of these tuples. Furthermore, we use the unique SG group-by values common to all input tuples assigned to $\mathbf{t}_g$ (i.e., $\mathbf{t}_g.G^{sg} = g$) as the output's SG group-by value.

Definition 17. *Consider a result group $g \in \mathcal{G}(G, \mathbf{R})$ for an aggregation with group-by attributes G over a $\mathbb{N}_{AU}$-relation $\mathbf{R}$. The bounds for the group-by attributes values of $\mathbf{t}_g$ are defined as shown below. For all $a \in G$ we define:*

$$\mathbf{t}_g.a^{\downarrow} = \min_{\mathbf{t}:\alpha(\mathbf{t})=g} \mathbf{t}.a^{\downarrow} \qquad \mathbf{t}_g.a^{sg} = g.a \qquad \mathbf{t}_g.a^{\uparrow} = \max_{\mathbf{t}:\alpha(\mathbf{t})=g} \mathbf{t}.a^{\uparrow}$$

Note that in the definition above, min and max are the minimum and maximum wrt. to the order over the data domain $\mathbb{D}$ which we

used to define range-annotated values. Reconsider the three example tuples and two result groups from above. The group-by range annotation for output tuple $\mathbf{t}_{g_1}$ is $[\min(1,2)/2/\max(2,4)] = [1/2/4]$. Observe that $[1/2/4]$ bounds every group $\mathbf{t}_1$ and $\mathbf{t}_2$ may belong to in some possible world. To calculate bounds on the result of an aggregation function for one group, we have to reason about the minimum and maximum possible aggregation function result based on the bounds of aggregation function input values, their associated row annotations, and their possible and guaranteed group memberships. To calculate a conservative lower bound of the aggregation function result for an output tuple $\mathbf{t}_g$, we use $\circledast_M$ to pair the aggregation function value of each tuple $\mathbf{t}$ for which $\alpha(\mathbf{t}) = g$ holds with the tuple's annotation and then extract the lower bound from the resulting range-annotated value. The group membership of a contributing tuple is uncertain if either its group-by values are uncertain or if it need not exist in all possible worlds (its certain multiplicity is 0). We thus take the minimum of the neutral element of the aggregation monoid and the result of $\circledast_M$ for such tuples. We introduce an *uncertain group* predicate $\text{UG}(G, \mathbf{R}, \mathbf{t})$ for this purpose:

$$\text{UG}(G, \mathbf{R}, \mathbf{t}) := (\exists a \in G : \mathbf{t}.a^{\downarrow} \neq \mathbf{t}.a^{\uparrow}) \vee \mathbf{R}(\mathbf{t})^{\downarrow} = 0$$

We then sum up the resulting values in the aggregation monoid. Note that here summation is addition in M. The upper bound calculation is analogous (using the upper bound and maximum instead). The SG result is calculated using standard $\mathbb{N}$-relational semantics. We use $\mathbf{t} \sqcap \mathbf{t}'$ to denote that the range annotations of tuples $\mathbf{t}$ and $\mathbf{t}'$ with the same schema $(A_1, \ldots, A_n)$ overlap on each attribute A_i, i.e.,

$$\mathbf{t} \sqcap \mathbf{t}' := \bigwedge_{i \in \{1,\ldots,n\}} [\mathbf{t}.A_i^{\downarrow}, \mathbf{t}.A_i^{\uparrow}] \cap [\mathbf{t}'.A_i^{\downarrow}, \mathbf{t}'.A_i^{\uparrow}] \neq \emptyset$$

Definition 18. *Consider input* $\mathbf{R}$*, set of group-by attributes* G*, an output* $g \in \mathcal{G}(G, \mathbf{R})$*, and aggregation function* $f(A)$ *with monoid* M*. We use* $\eth(g)$ *to denote the set of inputs with group-by attribute bounds overlaping* $\mathbf{t}_g.G$*, i.e., belonging to a group represented by* $\mathbf{t}_g$*:*

$$\eth(g) := \{\mathbf{t} \mid \mathbf{R}(\mathbf{t}) \neq \mathbb{0}_{\mathbb{N}_{AU}} \wedge \mathbf{t}.G \sqcap \mathbf{t}_g.G\}$$

The aggregation function result bounds for tuple $\mathbf{t}_g$ *are defined as:*

$$\mathbf{t}_g.f(A)^{\downarrow} = \sum_{\mathbf{t} \in \eth(g)} \mathit{LBAGG}(\mathbf{t}) \qquad \mathbf{t}_g.f(A)^{\uparrow} = \sum_{\mathbf{t} \in \eth(g)} \mathit{UBAGG}(\mathbf{t})$$

$$\mathbf{t}_g.f(A)^{sg} = \sum_{\mathbf{t} \in \eth(g)} (\mathbf{R}(\mathbf{t}) \circledast_M \mathbf{t}.A)^{sg}$$

$$\mathit{LBAGG}(\mathbf{t}) = \begin{cases} \mathit{min}(\mathbb{0}_M, (\mathbf{R}(\mathbf{t}) \circledast_M \mathbf{t}.A)^{\downarrow}) & \text{if } \text{UG}(G, \mathbf{R}, \mathbf{t}) \\ (\mathbf{R}(\mathbf{t}) \circledast_M \mathbf{t}.A)^{\downarrow} & \text{otherwise} \end{cases}$$

$$\mathit{UBAGG}(\mathbf{t}) = \begin{cases} \mathit{max}(\mathbb{0}_M, (\mathbf{R}(\mathbf{t}) \circledast_M \mathbf{t}.A)^{\uparrow}) & \text{if } \text{UG}(G, \mathbf{R}, \mathbf{t}) \\ (\mathbf{R}(\mathbf{t}) \circledast_M \mathbf{t}.A)^{\uparrow} & \text{otherwise} \end{cases}$$

Example 6. *For instance, consider calculating the sum of A grouping on B for a relation* $R(A, B)$*, which consists of two tuples* $\mathbf{t}_1 := \langle\ [3/5/10], [3/3/3]\ \rangle$ *and* $\mathbf{t}_2 := \langle\ [-4/-3/-3], [2/3/4]\ \rangle$ *that are both annotated with (1,2,2) (appear certainly once and may appear twice). Consider calculating the aggregation function result bounds for the result tuple* $\mathbf{t}_g$ *for the output group* $g := \langle\ 3\ \rangle$*. The lower bound* $\sum_{\mathbf{t} \in \eth(g)} \mathit{LBAGG}(\mathbf{t})$ *on* `sum(A)` *is calculated (Def. 15) as:*

$$= ((1,2,2) \cdot [3/5/10])^{\downarrow} + \mathit{min}(0, ((1,2,2) \cdot [-4/-3/-3])^{\downarrow})$$

$$= [3/10/20]^{\downarrow} + \mathit{min}(0, [-8/-6/-3]^{\downarrow}) = 3 + \mathit{min}(0, -8) = -5$$

The aggregation result is guaranteed to be greater than or equal to -5 *since* $\mathbf{t}_1$ *certainly belongs to* g *(no minimum operation), because its group-by attribute value* $[3/3/3]$ *is certain and the tuple certainly exists (*$(1,2,1)^{\downarrow} > 0$*). This tuple contributes* 3 *to the sum and* $\mathbf{t}_2$ *contributions at least* -8*. While it is possible that* $\mathbf{t}_2$ *does not belong to* g *this can only increase the final result (*$3 + 0 > 3 + -8$*).*

We still need to calculate the multiplicity annotation for each result tuple. For aggregation without group-by, there is always exactly one result tuple. In this case there exists a single possible SG output group (the empty tuple $\langle\ \rangle$) and all input tuples are assigned to it through α. Let $\mathbf{t}_{\langle\ \rangle}$ denote this single output tuple. Recalling that all remaining tuples have multiplicity 0, we define:

$$\gamma_{f(A)}(\mathbf{R})(\mathbf{t})^{\downarrow} = \gamma_{f(A)}(\mathbf{R})(\mathbf{t})^{sg} = \gamma_{f(A)}(\mathbf{R})(\mathbf{t})^{\uparrow} := \begin{cases} 1 & \text{if } \mathbf{t} = \mathbf{t}_{\langle\ \rangle} \\ 0 & \text{otherwise} \end{cases}$$

In order to calculate the upper bound on the possible multiplicity for a result tuple of a group-by aggregation, we have to determine the maximum number of distinct groups this output tuple could correspond to. We compute the bound for an output $\mathbf{t}_g$ based on $\mathcal{G}$ making the worst-case assumption that (i) each input tuple $\mathbf{t}$ with $\alpha(\mathbf{t}) = g$ has the maximal multiplicity possible ($\mathbf{R}(\mathbf{t})^{\uparrow}$); (ii) each tuple t encoded by $\mathbf{t}$ is in a separate group; and (iii) groups produced from two inputs $\mathbf{t}$ and $\mathbf{t}'$ do not overlap. We can improve this bound by partitioning the input into two sets: tuples with uncertain group-by values and tuples with certain group membership. When calculating the maximum number of groups for an output $\mathbf{t}_g$, the set of input tuples with certain group-by values that fall into the group-by bounds of $\mathbf{t}_g$ only contribute the number of distinct SG group-by values from this set to the bound. For the first set we still apply the worst-case assumption. To determine the lower bound on the certain annotation of a tuple we have to reason about which input tuples certainly belong to a group. These are inputs whose group-by attributes are certain. For such tuples we sum up their tuple annotation lower bounds. We then need to derive the annotation of a result tuple from relevant input tuple annotations. [9] extended semirings with a duplicate elimination operator $\delta_{\mathbb{N}}$ for this purpose which is defined as $\delta_{\mathbb{N}}(k) = 0$ if $k = 0$ and $\delta_{\mathbb{N}}(k) = 1$ otherwise.

Definition 19. *Let* $Q := \gamma_{G,f(A)}(\mathbf{R})$ *and* $\mathbb{G}_{def}(\mathbf{R}, G) = (\mathcal{G}, \alpha)$*. Consider a tuple* $\mathbf{t}$ *such that* $\exists g \in \mathcal{G}$ *with* $\mathbf{t} = \mathbf{t}_g$*. Then,*

$$\gamma_{G,f(A)}(\mathbf{R})(\mathbf{t})^{\downarrow} := \delta_{\mathbb{N}}\left(\sum_{\mathbf{t}' : \alpha(\mathbf{t}') = g \wedge \neg\, \text{UG}(G,\mathbf{R},\mathbf{t}')} \mathbf{R}(\mathbf{t}')^{\downarrow}\right)$$

$$\gamma_{G,f(A)}(\mathbf{R})(\mathbf{t})^{sg} := \delta_{\mathbb{N}}\left(\sum_{\mathbf{t}' : \alpha(\mathbf{t}') = g} \mathbf{R}(\mathbf{t}')^{sg}\right)$$

$$\gamma_{G,f(A)}(\mathbf{R})(\mathbf{t})^{\uparrow} := \sum_{\mathbf{t}' : \alpha(\mathbf{t}') = g} \mathbf{R}(\mathbf{t}')^{\uparrow}$$

For any tuple $\mathbf{t}$ *such that* $\neg\exists g \in \mathcal{G}$ *with* $\mathbf{t} = \mathbf{t}_g$*, we define*

$$\gamma_{G,f(A)}(\mathbf{R})(\mathbf{t})^{\downarrow} = \gamma_{G,f(A)}(\mathbf{R})(\mathbf{t})^{sg} = \gamma_{G,f(A)}(\mathbf{R})(\mathbf{t})^{\uparrow} = 0$$

The following example illustrates the application of the aggregation semantics we have defined in this section.

Example 7. *Consider the relation shown in Fig. 6 which records addresses (street, street number, number of inhabitants). Following [62],*

street	number	#inhab	$\mathbb{N}_{AU}$
Canal	[165/165/165]	[1/1/1]	(1,1,2)
Canal	[154/153/156]	[1/2/2]	(1,1,1)
State	[623/623/629]	[2/2/2]	(2,2,3)
Monroe	[3574/3550/3585]	[2/3/4]	(0,0,1)

(a) Input relation address

```
SELECT street,
       count(*) AS cnt
FROM address
GROUP BY street;
```

street	cnt	$\mathbb{N}_{AU}$
Canal	[1/2/3]	(1,1,2)
State	[2/2/4]	(1,1,1)
Monroe	[1/1/2]	(0,0,1)

(b) Aggregation with Group-by

Figure 6: Aggregation over AU-DBs

uncertain text fields are marked in red to indicate that their bound encompasses the whole domain. Street values v in black are certain, i.e., $v^{\downarrow} = v^{sg} = v^{\uparrow}$. We are uncertain about particular street numbers, number of inhabitants at certain addresses, and one street name. Furthermore, several tuples may represent more than one address. For the query shown in Fig. 6b consider the second result tuple (group State). This tuple certainly exists since the 3rd tuple in the input appears twice in every world and its group-by value is certain. Thus, the count for group State is at least two. The second input tuple could also belong to this group and, thus, the count could be 4 (the upper bound). Note also the multiplicity of the first output tuple (group Canal): [1/1/2]. In the SGW the first two input tuples belong to this group. However, the second input tuple need not be in this group in all worlds, and in fact may not belong to any existing group. Thus, in some possible worlds this one AU-DB tuple may represent 2 distinct output tuples.

We are ready to state the main result of this section: our aggregation semantics for AU-DBs is bound-preserving.

Theorem 4. *Let $Q := \gamma_{G,f(A)}(R)$ or $Q := \gamma_{f(A)}(R)$. $\mathbb{N}_{AU}$-relational query semantics preserves bounds for Q.*

Proof Sketch. The claim is proven by demonstrating that for every possible world R of the input incomplete $\mathbb{N}$-relation $\mathcal{R}$ and a tuple matching $\mathcal{TM}$ based on which $\mathbf{R}$ bounds R, we can construct a tuple matching based on which $Q(\mathbf{R})$ bounds $Q(R)$. For each aggregation result t in R, we show that there is a result in $Q(\mathbf{R})$ that bounds the values of t, because our aggregation semantics computes aggregation function results based on the minimum and maximum contribution of each possible input tuple, and group-by attribute bounds are generated by conservatively merging the group-by bounds of input tuples. See [26] for the full proof. □

Our main technical result follows from Thm. 4, [26, Thm. 3] (bound preservation for set difference) and Thm. 3:

Corollary 1. *$\mathbb{N}_{AU}$-semantics preserves bounds for $\mathcal{RA}^{agg}$.*

9 EXPERIMENTS

We compare AU-DBs (**AU-DB**) implemented on Postgres against (1) **Det**: Deterministic SGQP; (2) **Libkin**: An under-approximation of certain answers [33, 47]; (3) **UA-DB**: An under-approximation of certain answers combined with SGQP [25]; (4) **MayBMS**: MayBMS used to compute all possible answers[3]; (5) **MCDB**: Database sampling (10 samples) in the spirit of MCDB [37] to over-approximate certain answers; (6) **Trio**: A probabilistic DB with bounds for aggregation [7]; and (7) **Symb**: An SMT solver (Z3) calculating aggregation result bounds based on the symbolic representation from [9]. All experiments are run on a 2×6 core AMD Opteron 4238 CPUs, 128GB RAM, 4×1TB 7.2K HDs (RAID 5). We report the average of 10 runs.

[3]Times listed for MayBMS and MCDB include only computing possible answers and not computing probabilities.

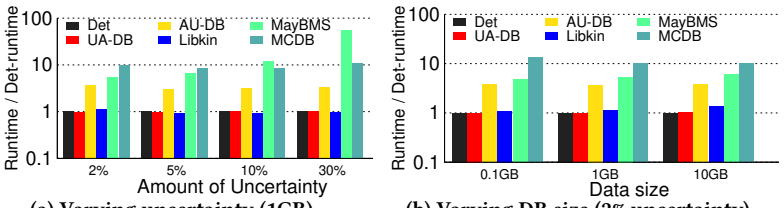
9.1 Uncertain TPC-H (PDBench)

We use PDBench [11], a modified TPC-H data generator [23] that creates an x-DB (block-independent database) with attribute-level uncertainty by replacing random attributes with multiple randomly selected possible alternatives. We directly run MayBMS queries (without probability computations) on its native columnar data representation. For MCDB, we approximate tuple bundles with 10 samples. We apply Libkin on a database with labeled nulls for uncertain attributes using the optimized rewriting from [33]. We run Det on one randomly selected world — this world is also used as the SGW for UA-DB and AU-DB. We construct an AU-DB instance by annotating each cell in this world with the minimum and maximum possible values for this cell across all worlds. For UA-DB we mark all tuples with at least one uncertain value as uncertain.

PDBench Queries. To evaluate the overhead of our approach compared to UA-DBs for queries supported by this model, we reproduce the experimental setup from [25], which uses the queries of PDBench (simple SPJ queries). With a scale factor 1 (SF1) database (~1GB per world), we evaluate scalability relative to amount of uncertainty. Using PDBench, we vary the percentage of uncertain cells: 2%, 5%, 10% and 30%. Each uncertain cell has up to 8 possible values picked uniformly at random over the whole domain, resulting in large ranges, a worst-case scenario for AU-DBs and a best-case scenario for MayBMS. As Fig. 7a shows, our approach has constant overhead (a factor of ~ 5), resulting from the many possible tuples created by joins on attributes with ranges across the entire domain. To evaluate scalability, we use 100MB, 1GB, and 10GB datasets (SF 0.1, 1, and 10) and fix the uncertainty percentage (2%). As evident from Fig. 7b, AU-DBs scale linearly in the SF for such queries.

TPC-H queries. We now evaluate actual TPC-H queries on PDBench data. These queries contain aggregation with uncertain group-by attributes (only supported by AU-DB and MCDB). Results are shown in Fig. 9. For most queries, AU-DB has an overhead factor of between 3-7 over Det. This overhead is mainly due to additional columns and scalar expressions. Compared to MCDB, AU-DB is up to 570% faster, while producing hard bounds instead of an estimation.

Simple Aggregation. We use a simple aggregation query with certain group-by attributes on an SF0.1 instance to compare against a wider range of approaches, varying the number of aggregation operators (#agg-ops). For systems that do not support subqueries like Trio, operator outputs are materialized as tables. In this experiment, Trio produces incorrect answers, as its representation of aggregation results (bounds) is not closed under queries; We are only interested in its performance. Fig. 8 shows the runtime of our technique compared to Trio which is significantly slower, and Symb (only competitive for low #agg-ops values).



(a) Varying uncertainty (1GB) (b) Varying DB size (2% uncertainty)

Figure 7: PDbench Queries

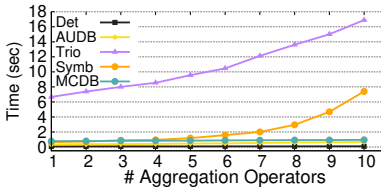


Figure 8: Simple aggregation over TPC-H data

Queries		2%/SF0.1	2%/SF1	5%/SF1	10%/SF1	30%/SF1
Q1	AU-DB	1.607	15.636	15.746	15.811	16.021
	Det	0.560	1.833	1.884	1.882	1.883
	MCDB	5.152	19.107	18.938	19.063	19.279
Q3	AU-DB	0.713	7.830	8.170	8.530	7.972
	Det	0.394	1.017	1.058	1.092	1.175
	MCDB	4.112	11.138	11.222	10.936	11.454
Q5	AU-DB	0.846	8.877	8.803	8.839	8.925
	Det	0.247	0.999	1.012	1.123	1.117
	MCDB	2.599	10.152	10.981	11.527	11.909
Q7	AU-DB	0.791	7.484	7.537	7.303	7.259
	Det	0.145	0.977	0.985	0.989	1.044
	MCDB	1.472	10.123	10.277	10.749	10.900
Q10	AU-DB	0.745	7.377	7.283	7.715	8.012
	Det	0.263	1.024	0.993	1.004	1.015
	MCDB	2.691	10.743	10.937	11.826	11.697

Figure 9: TPC-H query performance (runtime in sec)

9.2 Micro-benchmarks

We use a synthetic table with 100 attributes with uniform random values to evaluate the performance and accuracy of our approach.

Varying number of group-by attributes. We use an **sum** aggregation with 1 to 99 group-by attributes on a table with 35k rows and 5% uncertainty. Our implementation applies an aggregate analog of the join optimization described in Sec. 7.1: possible groups are compressed before being joined with the output groups (see [26]). This improves performance when there are fewer result groups. As Fig. 10a shows, overhead over `Det` is up to a factor of 6 to 7.

Varying number of aggregates. Using a similar query and dataset, and 1 group-by attribute, we vary the number of aggregation functions from 1 to 99. As Fig. 10b shows, the overhead of our approach compared to `Det` varies between a factor of 5 to 6.

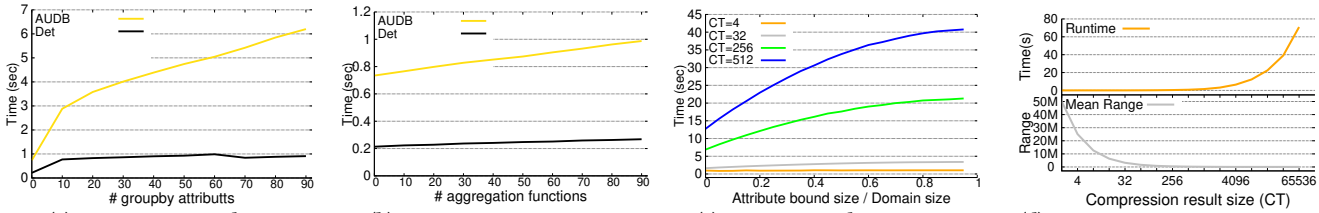
Compression Trade-off for Aggregation. We evaluate the trade-offs between tightness and compression for aggregation using **sum** aggregation with group-by. Fig. 10d shows the runtime overhead of our approach over `Det` when increasing the number of tuples in the compressed pre-aggregation result. The input table has 10% uncertainty and 10k rows. For tightness we calculate tight bounds for the aggregation function results for each possible group (a group that exists in at least one world). We then measure for each such group the relative size of our approximate bounds compared to the maximally tight bounds and report the average of this number.

Attribute Bound Size. Next, we vary the average size of attribute-level bounds (same query as above). We generate tables with 35k rows each and 5% uncertainty, varying the range of uncertain attribute values from 0% to 100% of the attribute's domain. We measure runtime, varying the number of tuples in the compressed result (CT) for the pre-aggregation step. For more aggressive compression (Fig. 10c), the runtime of our approach is only slightly affected by the size of attribute-level bounds. We also measure how the size of attribute ranges affects precision. We generate x-DBs with 2%, 3%, and 5% of uncertain tuples (10 alternatives per uncertain tuple) varying attribute ranges from 1% to 10% of the entire value domain. We create an AU-DB from the x-DB ([26] details how this is achieved). Fig. 12a and 12b show the percentage of over-grouping for `AU-DB` (increase in group size, because of over-estimation of possible group-by attribute values) and relative factor of aggregation result range over-estimation. The range over-estimation grows faster than over-grouping, as it is affected by uncertainty in aggregation function inputs as well as the over-grouping.

Join Optimizations. Next, we evaluate the impact of our join optimization. Fig. 11a shows the runtime for a single equi-join (log-scale) varying the size of both input relations from 5k to 20k rows containing 3% uncertain values ranging over 2% of the value domain. The optimized version is between ~ 1 and ~ 2 orders of magnitude faster depending on the compression rate (i.e., CT). As a simple accuracy measure, Fig. 11a shows the number of possible tuples in the join result. Next, we join tables of 4k rows with 3% or 10% uncertainty and vary the number of joins (1 to 4 chained equality joins, i.e., no overlap of join attributes between joins). As shown in Fig. 13, joins without optimization are up to 4 orders of magnitude more expensive, because of the nested loop joins that are needed for interval-overlap joins and resulting large result relations.

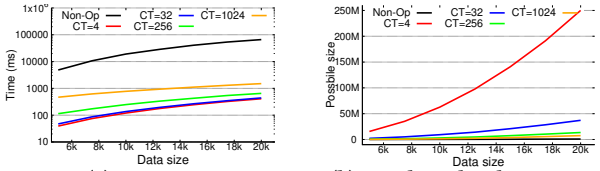
9.3 Real World Data

For this experiment, we repaired key violations for real world datasets (references shown in Fig. 14). To repair key violations, we group tuples by their key attributes so that each group represents all possibilities of a single tuple with the corresponding key value. For each group, we randomly pick one tuple for the SGW and use all tuples in the group to determine its attribute bounds as the minimum (maximum) value within the group. Fig. 14 shows for each dataset the percentage of tuples with uncertain values and for all such tuples the average number of possibilities. We generated SPJ (**SPJ**) and simple aggregation queries with group-by (**GB**) for each of these datasets (query types are shown in Fig. 14, see [26] for additional details). Fig. 14 shows the runtime for these queries comparing `AU-DB` with `MCDB`, `Trio` and `UA-DB`. `AU-DB` is significantly faster than `Trio` and consistently outperforms `MCDB`. As a comparison point and to calculate our quality metrics, for each query we calculated the precise set of certain and possible tuples and exact bounds for attribute-level uncertainty in the query result. We execute those queries in each system and report the recall of certain and possible tuples it returns versus the exact result. Note that for possible tuple recall, we report two metrics. The first ignores attribute-level uncertainty. Possible tuples are grouped by their key (or group-by values for aggregation queries) and we measure the



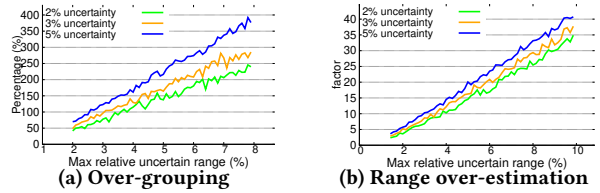
(a) Varying #group by (b) Varying #aggregation (c) Varying attribute range (d) Varying compression rate

Figure 10: Aggregation Microbenchmarks - Performance and Accuracy



(a) Runtime (b) Attribute-level Accuracy

Figure 11: Join Optimizations - Performance and Accuracy



(a) Over-grouping (b) Range over-estimation

Figure 12: Aggregation - varying attribute range

Comp. Size		1 join	2 joins	3 joins	4 joins
4	3%	0.004	0.006	0.009	0.015
	10%	0.004	0.007	0.010	0.015
16	3%	0.005	0.008	0.012	0.017
	10%	0.005	0.009	0.012	0.017
64	3%	0.009	0.027	0.47	0.069
	10%	0.009	0.029	0.049	0.070
256	3%	0.036	0.308	0.627	0.969
	10%	0.043	0.337	0.660	1.019
No Comp.	3%	0.216	1.351	6.269	29.639
	10%	0.213	2.565	29.379	333.695

Figure 13: Join query performance (runtime in sec)

Datasets & Queries			Time (sec)	cert. tup.	attr. bounds min	attr. bounds max	pos.tup. by id	pos.tup. by val
Netflix [3] (1.9%, 2.1)	$Q_{n,1}$ **SPJ**	AU-DB	**0.011**	100%	1	1	100%	100%
		Trio	0.900	100%	1	1	100%	100%
		MCDB	0.049	N.A.	1	1	99.6%	98.5%
		UA-DB	0.006	100%	N.A.	N.A.	99.1%	97.3%
	$Q_{n,2}$ **GB**	AU-DB	**0.082**	100%	1	4	100%	100%
		Trio	1.700	100%	1	1	98.8%	98.0%
		MCDB	0.118	N.A.	1	1	99.9%	97.9%
		UA-DB	0.009	0%	N.A.	N.A.	99.3%	95.7%
Crimes [2] (0.1%, 3.2)	$Q_{c,1}$ **SPJ**	AU-DB	**1.58**	100%	1	1	100%	100%
		Trio	59.0	100%	1	1	100%	100%
		MCDB	6.91	N.A.	0.6	1	99.9%	92.1%
		UA-DB	0.63	100%	N.A.	N.A.	99.9%	87.5%
	$Q_{c,2}$ **GB**	AU-DB	**2.09**	100%	1	1.01	100%	100%
		Trio	103.1	100%	1	1	100%	100%
		MCDB	5.24	N.A.	0.99	0	100%	~ 0%
		UA-DB	0.47	0%	N.A.	N.A.	100%	~ 0%
Healthcare [1] (1.0%, 2.7)	$Q_{h,1}$ **SPJ**	AU-DB	**0.179**	99.5%	1	1	100%	100%
		Trio	20.6	100%	1	1	100%	100%
		MCDB	0.501	N.A.	0.4	1	99.9%	87.6%
		UA-DB	0.042	98.2%	N.A.	N.A.	99.3%	65.4%
	$Q_{h,2}$ **GB**	AU-DB	**0.859**	100%	1	45	100%	100%
		Trio	29.2	100%	1	1	100%	100%
		MCDB	2.31	N.A.	0.78	1	100%	~ 0%
		UA-DB	0.235	0%	N.A.	N.A.	100%	~ 0%

Figure 14: Real world data - performance and accuracy

percentage of returned groups (a group is "covered" if at least one possible tuple from the group is returned). The second metric just measures the percentage of all possible tuples (without grouping) that are returned. We also measure the tightness of attribute-level bounds for certain rows by measuring for each tuple the average size of its attribute-level bounds relative to exact bounds. Fig. 14 shows the minimum and maximum of this metric across all certain result tuples. Since MCDB relies on samples, it (i) may not return all possible tuples and (ii) calculating bounds for attributes values from the sample, we get bounds that may not cover all possible values. Furthermore, MCDB cannot distinguish between certain and possible tuples. For Trio the bounds on aggregation results are tight, but Trio does not support uncertainty in group-by attributes (no result is returned for a group with uncertain group-by values). As shown in Fig. 14, our attribute-level bounds are close to the tight bounds produced by Trio for most of the certain result tuples. MCDB does not return all possible aggregation result values (the ones not covered by the samples). Furthermore, we never miss possible tuples like both Trio and MCDB, and seldomly report a certain tuple as uncertain, while MCDB cannot distinguish certain from possible. UA-DB has performance close to conventional (SGQP) query processing and outperforms all other methods. However, UA-DBs provide no attribute level uncertainty and only contain tuples from the SGW and, thus, miss most possible tuples. Furthermore, aggregates over UA-DBs will not return any certain answers, as doing so requires having a bound on all possible input tuples for the aggregate and often additionally requires attribute-level uncertainty (the group exists certainly in the result, but the aggregation function result for this group is uncertain). For aggregates over UA-DBs, the range of the attribute bounds is significantly affected by the attribute domain and the aggregation functions used. $Q_{n,2}$ and $Q_{c,2}$ use **max** and **count**, which return a relatively small over-estimation of the actual bounds. $Q_{h,2}$ uses **sum**, where the larger domain for the attribute over which we are aggregating over, and the combined effect of over-grouping and over-estimation of possible attribute values results in a larger over-estimation.

10 CONCLUSIONS

We present attribute-annotated uncertain databases (AU-DBs), an efficient scheme for approximating certain and possible answers for full relational algebra and aggregation. Our approach stands out in that it is (i) more general in terms of supported queries than most past work, (ii) has guaranteed PTIME data complexity, and (iii) compactly encodes over-approximations of incomplete databases. In future work, we will investigate extensions of this model for queries with ordering (top-k queries and window functions). We will also explore how to manage non-ordinal categorical attributes.

REFERENCES

[1] https://data.medicare.gov/data/hospital-compare. Medicare Hospital Dataset. (https://data.medicare.gov/data/hospital-compare).

[2] https://www.kaggle.com/currie32/crimes-in-chicago. Chicago Crimes Dataset. (https://www.kaggle.com/currie32/crimes-in-chicago).

[3] https://www.kaggle.com/shivamb/netflix-shows. Netflix Dataset. (https://www.kaggle.com/shivamb/netflix-shows).

[4] Serge Abiteboul, T.-H. Hubert Chan, Evgeny Kharlamov, Werner Nutt, and Pierre Senellart. 2010. Aggregate queries for discrete and continuous probabilistic XML. In *ICDT*. 50–61.

[5] 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.

[6] Foto N. Afrati and Phokion G. Kolaitis. 2008. Answering aggregate queries in data exchange. In *PODS*. 129–138.

[7] 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*.

[8] Parag Agrawal, Anish Das Sarma, Jeffrey Ullman, and Jennifer Widom. 2010. Foundations of uncertain-data integration. *PVLDB* 3, 1-2 (2010), 1080–1090.

[9] Yael Amsterdamer, Daniel Deutch, and Val Tannen. 2011. Provenance for Aggregate Queries. In *PODS*. 153–164.

[10] Yael Amsterdamer, Daniel Deutch, and Val Tannen. 2011. Provenance for aggregate queries. In *PODS*. 153–164.

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

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

[13] Marcelo Arenas, Leopoldo E. Bertossi, Jan Chomicki, Xin He, Vijay Raghavan, and Jeremy P. Spinrad. 2003. Scalar aggregation in inconsistent databases. *Theor. Comput. Sci.* 296, 3 (2003), 405–434.

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

[15] George Beskales, Ihab F. Ilyas, Lukasz Golab, and Artur Galiullin. 2014. Sampling from Repairs of Conditional Functional Dependency Violations. *VLDBJ* 23, 1 (2014), 103–128.

[16] Mike Brachmann, Carlos Bautista, Sonia Castelo, Su Feng, Juliana Freire, Boris Glavic, Oliver Kennedy, Heiko Müller, Rémi Rampin, William Spoth, and Ying Yang. 2019. Data Debugging and Exploration with Vizier. In *SIGMOD*.

[17] Douglas Burdick, Prasad M. Deshpande, T. S. Jayram, Raghu Ramakrishnan, and Shivakumar Vaithyanathan. 2007. OLAP over uncertain and imprecise data. *VLDBJ* 16, 1 (2007), 123–144.

[18] Diego Calvanese, Evgeny Kharlamov, Werner Nutt, and Camilo Thorne. 2008. Aggregate queries over ontologies. In *International Workshop on Ontologies and Information Systems for the Semantic Web (ONISW)*. 97–104.

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

[20] Arbee L. P. Chen, Jui-Shang Chiu, and Frank Shou-Cheng Tseng. 1996. Evaluating Aggregate Operations Over Imprecise Data. *IEEE Trans. Knowl. Data Eng.* 8, 2 (1996), 273–284.

[21] Marco Console, Paolo Guagliardo, and Leonid Libkin. 2019. Fragments of Bag Relational Algebra: Expressiveness and Certain Answers. In *ICDT*. 8:1–8:16.

[22] Marco Console, Paolo Guagliardo, Leonid Libkin, and Etienne Toussaint. 2020. Coping with Incomplete Data: Recent Advances. In *PODS*. ACM, 33–47.

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

[24] Wenfei Fan. 2008. Dependencies revisited for improving data quality. In *PODS*. 159–170.

[25] Su Feng, Aaron Huber, Boris Glavic, and Oliver Kennedy. 2019. Uncertainty Annotated Databases - A Lightweight Approach for Approximating Certain Answers. In *SIGMOD*.

[26] Su Feng, Aaron Huber, Boris Glavic, and Oliver Kennedy. 2021. Efficient Uncertainty Tracking for Complex Queries with Attribute-Level Bounds (extended version). (2021). arXiv:2102.11796 [cs.DB]

[27] Robert Fink, Larisa Han, and Dan Olteanu. 2012. Aggregation in Probabilistic Databases via Knowledge Compilation. *PVLDB* 5, 5 (2012), 490–501.

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

[29] A. Fuxman, E. Fazli, and R.J. Miller. 2005. Conquer: Efficient management of inconsistent databases. In *SIGMOD*. 155–166.

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

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

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

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

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

[35] Alon Halevy, Anand Rajaraman, and Joann Ordille. 2006. Data integration: the teenage years. In *VLDB*. 9–16.

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

[37] 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*.

[38] T. S. Jayram, Satyen Kale, and Erik Vee. 2007. Efficient aggregation algorithms for probabilistic data. In *SODA*. 346–355.

[39] Shawn R. Jeffery, Gustavo Alonso, Michael J. Franklin, Wei Hong, and Jennifer Widom. 2006. Declarative Support for Sensor Data Cleaning. In *PERVASIVE*. 83–100.

[40] O. Kennedy and C. Koch. 2010. PIP: A database system for great and small expectations. In *ICDE*. 157–168.

[41] 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.

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

[43] Poonam Kumari, Said Achmiz, and Oliver Kennedy. 2016. Communicating Data Quality in On-Demand Curation. In *QDB*.

[44] Willis Lang, Rimma V. Nehme, Eric Robinson, and Jeffrey F. Naughton. 2014. Partial results in database systems. In *SIGMOD*. 1275–1286.

[45] Jens Lechtenbörger, Hua Shu, and Gottfried Vossen. 2002. Aggregate Queries Over Conditional Tables. *J. Intell. Inf. Syst.* 19, 3 (2002), 343–362.

[46] Xi Liang, Zechao Shang, Sanjay Krishnan, Aaron J. Elmore, and Michael J. Franklin. 2020. Fast and Reliable Missing Data Contingency Analysis with Predicate-Constraints. In *SIGMOD*. 285–295.

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

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

[49] Raghotham Murthy, Robert Ikeda, and Jennifer Widom. 2011. Making Aggregation Work in Uncertain and Probabilistic Databases. *IEEE Trans. Knowl. Data Eng.* 23, 8 (2011), 1261–1273.

[50] Dan Olteanu, Lampros Papageorgiou, and Sebastiaan J van Schaik. 2013. Pigora: An Integration System for Probabilistic Data. In *ICDE*. 1324–1327.

[51] Alexander J. Ratner, Stephen H. Bach, Henry R. Ehrenberg, and Christopher Ré. 2017. Snorkel: Fast Training Set Generation for Information Extraction. In *SIGMOD*. 1683–1686.

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

[53] Christopher Ré and Dan Suciu. 2009. The trichotomy of HAVING queries on a probabilistic database. *VLDBJ* 18, 5 (2009), 1091–1116.

[54] Sunita Sarawagi et al. 2008. Information extraction. *Foundations and Trends® in Databases* 1, 3 (2008), 261–377.

[55] Yannis Sismanis, Ling Wang, Ariel Fuxman, Peter J. Haas, and Berthold Reinwald. 2009. Resolution-Aware Query Answering for Business Intelligence. In *ICDE*. 976–987.

[56] Mohamed A. Soliman, Ihab F. Ilyas, and Kevin Chen-Chuan Chang. 2008. Probabilistic top-k and ranking-aggregate queries. *TODS* 33, 3 (2008), 13:1–13:54.

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

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

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

[60] Jef Wijsen. 2012. Certain conjunctive query answering in first-order logic. *TODS* 37, 2 (2012), 9:1–9:35.

[61] Mohan Yang, Haixun Wang, Haiquan Chen, and Wei-Shinn Ku. 2011. Querying uncertain data with aggregate constraints. In *SIGMOD*. 817–828.

[62] 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.