EFFICIENT MANAGEMENT OF UNCERTAIN DATA

BY

SU FENG

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science
in the Graduate College of the
Illinois Institute of Technology

Approved 
Adviser

Chicago, Illinois
May 2023

TABLE OF CONTENTS

AUTHORSHIP STATEMENT

I, Su Feng (S.F.) attest that the work presented in this thesis is substantially my own.

In accordance with the disciplinary norm of Computer Science (see IIT Faculty Handbook, Appendix S), the following collaborations occurred in the thesis:

Dr. Boris Glavic (B.G.), guided and assisted the interpretation of formalisms, proves and design of experiments as is the norm for a Ph.D. supervisor (Appendix S of the IIT Faculty Handbook). Some background materials and underlying systems my formalisms and implementations relying on were the work of B.G. or with his collaborators. These backgrounds have been clearly cited in this thesis.

Dr. Oliver Kennedy (O.K.) and his Ph.D. student Aaron Huber (A.H.), of University at Buffalo, Buffalo NY, collaborated on formalism refinements, proof readings on submissions of my works and implementation of experiment: Access control semiring - mislabeling.

LIST OF FIGURES

ABSTRACT

Uncertainty arises naturally in many application domains. It can be caused by an uncertain data source (sensor errors, noise, etc.). Data preprocessing techniques (data curation, data integration, etc.) can also results in uncertainty to the data. Analyzing uncertain data without accounting for its uncertainty can create hard to trace errors, with severe real world implications. Certain answers are a principled method for coping with the uncertainty that arises in many practical data management tasks. Unfortunately, this method is expensive and may exclude useful (if uncertain) answers. Other techniques from incomplete database record and propagate more detailed uncertainty information. However, most of these approaches are either too expensive to be practical, or only focus on a narrow class of queries and only work for a specific representation. In this thesis, we investigate models and query semantics for uncertain data management and present a framework that is general and practically efficient, backed up by fundamental theoretical foundations and with formally proven correctness guarantees. We first propose Uncertainty Annotated Databases (UA-DB), which combine an under- and over-approximation of certain answers to combine the reliability of certain answers with the performance of a classical database system. We then introduce attribute-annotated uncertain databases (AU-DB), which extend the UA-DB model with attribute-level annotations that record bounds on the values of an attribute across all possible worlds. AU-DB extends UA-DBs to encode a compact over-approximation of possible answers which is necessary to support non-monotone queries including aggregation and set difference. With a further extension to AU-DB that supports ranking and windowed aggregation queries using native implementation on modern DBMS, our approaches scale to complex queries and large datasets, and produces accurate results. Furthermore, they significantly outperforms alternative methods for uncertain data management.

CHAPTER 1

INTRODUCTION

Uncertainty arises naturally in many application domains due to data entry errors, sensor errors and noise [1], uncertainty in information extraction and parsing [2], ambiguity from data integration [3–5], and heuristic data wrangling [6–8]. Analyzing uncertain data without accounting for its uncertainty can lead to hard to trace errors, with severe real world implications, such as financial damages, incorrect scientific conclusions, or even affect people's physical well-being. Incomplete database techniques [9] 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 is one possible state of the real world. Table 1.1 shows Covid infection data gathered from two different sources. The two sources may disagree with each other on some part of the data (i.e. Los Angeles has 3% infection rate in D_1 and 4% in D_2). Each possible instance of the data is a **possible world**. Naively, querying over an incomplete database can be processed by running the query on every possible world. In real world applications, the number of possible world can grow exponentially with respect to number of uncertain tuples in the database. Thus, the naive approach is not practical. Previous works developed different querying semantics for incomplete databases.

General semantics: Under the commonly used *certain answer semantics* [11, 12], a query returns the set of answer tuples *guaranteed* to be in the result, regardless of

[1]Probabilistic databases [10] generalize incomplete databases with a probability distribution over all possible worlds. We focus on contrasting with the former for simplicity, but many of the same cost and expressivity limitations also affect probabilistic databases.

Table 1.1. An incomplete database with two possible worlds

D_1			D_2		
locale	**rate**	**size**	**locale**	**rate**	**size**
Los Angeles	3%	metro	Los Angeles	4%	metro
Austin	18%	city	Austin	18%	metro
Houston	14%	metro	Houston	14%	metro
Berlin	3%	town	Berlin	1%	city
Sacramento	1%	**null**	Sacramento	1%	**null**
Springfield	**null**	town	Springfield	**null**	town

which possible world is correct. Many computational problems are intractable over incomplete databases. Even approximations (e.g., [13–15]) are often still not efficient enough, are insufficiently expressive, or exclude useful answers [9, 16]. Thus, typical database users resort to resolving uncertainty using heuristics and then treating the result as a deterministic database [6]. 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. Table 1.2 (left) shows an example of 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*

Table 1.2. Unreliable data in x-DBs [17] representation

$\mathcal{D}$

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%

the real world. Alice's ETL heuristics select (e.g., based on the relative trustworthiness of each source) one possible world D_{SG} (Table 1.3) 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 *[12] 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., Table 1.2, right). 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*

Table 1.3. A Selected Guess world (SG)

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%

is expensive (coNP-hard/NP-hard).

Aggregation semantics: Neither certain answers nor possible answer semantics are meaningful for aggregation over uncertain data (e.g., see [9] 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., [18–20]), or computing expectations (e.g., [20,21]) when probabilities are available. Unfortunately, for such approaches, aggregate queries and non-aggregate queries return incompatible results, and thus 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., [22, 23]). 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.

Order-based semantics: Uncertain versions of order-based operators like `SORT / LIMIT` (i.e., Top-K) have been studied extensively in the past [24–27]. However, the resulting semantics often lacks *closure*. That is, composing such operators with other operators typically requires a complete rethinking of the entire system [28], because the model that the operator expects its *inputs* to be encoded with differs from the model encoding the operator's *outputs*. Further more, to the best of our knowledge, there are no existing works supports uncertain windowed aggregations.

1.1 Requirements

As general goal of the thesis, we want to develop solutions to incomplete data processing that fulfill following requirements:

1. Compact, general and easy to use encoding of uncertainty information
2. Efficient semantics that propagate uncertainty information through queries correctly
3. Support wild classes of queries

We propose two uncertain data models: Uncertainty-Annotated Databases(UA-DBs) and Attribute-annotated uncertain databases (AU-DBs) that addressing these requirements.

1.2 Uncertainty-Annotated Databases (UA-DBs)

As a initial step toward modeling uncertain data, UA-DB addressed requirement 1 and 2 from Sec. 1.1. we developed an approach that generalizes to a wide range of data models, is easy to use like SGQP, is compatible with a wide range of probabilistic and incomplete data representations (e.g., tuple-independent databases [10],

C-tables [12], and x-DBs [17]) and sources of uncertainty (e.g., inconsistent databases [15, 29–33], imputation of missing values, and more), and is principled like certain answers. We address the generality requirement (1) by rethinking incomplete data management in terms of Green et. al.'s $\mathcal{K}$-database framework [34]. In this framework, each tuple is annotated with an value from a semiring $\mathcal{K}$. Choosing an appropriate semiring, $\mathcal{K}$-databases can encode a wide range of query processing semantics including classical set- and bag-semantics, as well as query processing with access control, provenance, and more. Our primary contribution here is to identify a natural, backwards-compatible generalization of certain answers to a broad class of $\mathcal{K}$-databases.

The second major contribution for UA-DB is to combine an under-approximation of certain answers with selected-guess query processing to create an *Uncertainty-Annotated Database* (*UA-DB*). A UA-DB is built around one distinguished possible world of an incomplete $\mathcal{K}$-database, for instance the "selected-guess" world that would normally be used in practice. This world serves as an *over-approximation* (upper-bound) of certain answers. Tuples from this world are labeled as either certain or uncertain to encode an *under-approximation* (lower-bound) of certain answers. As illustrated in Figure 1.2, a UA-DB bounds the certain answers between under- and over-approximations. A lightweight (extensional [10]) query evaluation semantics then propagates labels while preserving the approximation's guarantees.

Example 2. *Continuing previous example, consider query*

```
SELECT locale, size FROM locales WHERE rate >= 3%,
```

Table 1.4 shows the encoding of UA-DB and result of the query as a set UA-DB. When the UA-DB is built, one designated possible world of is selected. For this example, we select the D_{SG}. The result is based on this one designated possible world, which serves as an over-approximation of the certain answers. A subset of these tuples (tuple 3)

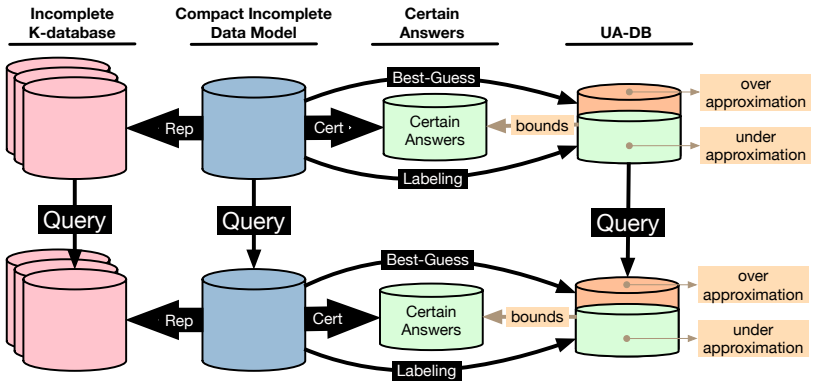


Figure 1.1. The relationship between UA-DBs, certain answers, and other incomplete data models

Table 1.4. UA-DB encoding (based on D_{SG})

D_{UA}

locale	**rate**	**size**	$\mathbb{B}$
Los Angeles	3%	metro	F
Austin	18%	city	F
Houston	14%	metro	T
Berlin	3%	town	F
Sacramento	1%	town	F
Springfield	5%	town	F

$Q(D_{UA})$

locale	**size**	$\mathbb{B}$
Los Angeles	metro	F
Austiny	city	F
Houston	metro	T
Berlin	town	F
Springfield	town	F

are explicitly labeled as certain. This is the under-approximation: A tuple might still be certain even if it is not labeled as such. We consider the remaining tuples to be "uncertain". In Figure 1.4, tuple 3 are correctly marked as certain, while tuple 1 is mis-classified as uncertain even though it appears in all worlds. We stress that even a mislabeled certain answer is still present: a UA-DB sandwiches the certain answers.

Fig. 1.1 Shows an overview of our approach. We provide *certain under-approximations* that derive a UA-DB from common incomplete data models. The resulting UA-DB bounds the certain tuples from above and below, a property pre-

served through queries. UA-DBs are both efficient and precise. We demonstrate efficiency by implementing a bag UA-DB as a query-rewriting front-end on top of a classical relational DBMS: UA-DB queries have minimal performance overhead compared to the same queries on deterministic data. We demonstrate precision both analytically and experimentally. First, under specific conditions, some of which we identify in Sec. 4.8, exactly the certain answers will be marked as certain. Second, we show experimentally that even when these conditions do not hold, the fraction of misclassified certain answers is low. Importantly, a wide range of uncertain data models can be translated into UA-DBs through simple and efficient transformations that (i) determine a selected-guess world (SGW) and (ii) obtain an under-approximation of the certain answers. We define such transformations for three popular models of incomplete data in Sec. 4.5: tuple-independent databases [10], x-DBs [17] and C-tables [12]. In classical incomplete databases, where probabilities are not available, any possible world can serve as a SGW. In probabilistic databases (or any incomplete data model that ranks possible worlds), we preferentially use the possible world with the highest probability (if computationally feasible), or an approximation thereof. We emphasize that our approach does not require enumerating (or even knowing) the full set of possible worlds. As long as some possible world can be obtained, our approach is applicable. In worst case, if no certainty information is available, our approach labels all tuples as uncertain and degrades to classical selected-guess query processing.

Furthermore, our approach is also applicable to use cases like inconsistent query answering [33] where possible worlds are defined declaratively (e.g., all repairs of an inconsistent database).

We significantly extend the state-of-the-art on under-approximating certain answers [13, 35, 36]: (1) we combine an under-approximation with best-guess query processing bounding certain answers from above and below; (2) we support sets, bags, and any other data model expressible as semiring annotations from a large class of

semirings; (3) we support translation of a wide range of incomplete and probabilistic data models into our UA-DB model; (4) in contrast to certain answers, UA-DBs are closed under queries.

1.3 Attribute-annotated uncertain databases (AU-DBs)

UA-DBs full-filled part of our requirements by providing a tuple level uncertain model that bounds certain answers using under-approximation of certain answers and SGQP. However, UA-DBs only meaningfully supports $\mathcal{RA}^+$ queries which is a subset of the query classes. In this model, we are aiming to address requirement 3 while keeping requirement 1 and requirement 2 satisfied. 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, 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 annotations (multiplicities in the case of bag semantics). 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. 1.2.

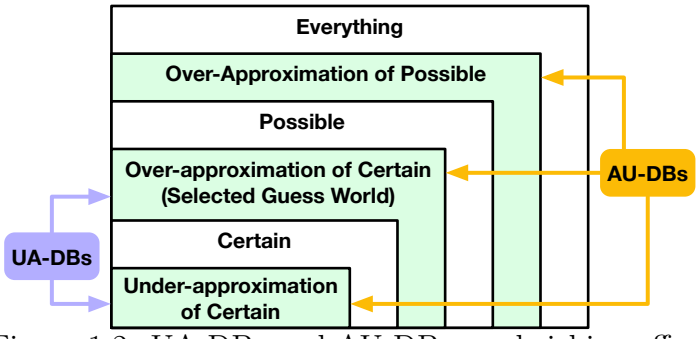


Figure 1.2. UA-DBs and AU-DBs sandwiching effect

Example 3. *Table 1.5 shows an AU-DB constructed from one possible world* D_{SG}

Table 1.5. Possible AU-DB Encoding (based on D_{SG})

$\underline{D_{AU}}$

locale	**rate**	**size**	$\underline{\mathbb{N}^3}$
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)

$\underline{Q(D_{AU})}$

size	**spop**	$\underline{\mathbb{N}^3}$
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)

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 Table 1.5. 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.

When sorting uncertain attribute values, the possible order-by attribute values of two tuples t_1 and t_2 may overlap, which leads to multiple possible sort orders. Supporting order-based operators over AU-DBs requires encoding multiple possible sort orders. Unfortunately, a dataset can only have one physical ordering. We address this limitation by introducing a **position** attribute, decoupling the *physical* order in which the tuples are stored from the set of possible *logical* orderings. With a tuple's position in a sort order encoded as a numerical attribute, operations that act on this order (i.e., `LIMIT`) can be redefined in terms of standard relational operators. In short, by virtualizing sort order into a position attribute, AU-DB model is sufficient to express the output of SQL's order-dependent operations in the presence of uncertainty.

To understand the intuition behind these operators, consider the logical sort operator, which extends each input row with a new attribute storing the row's position wrt. to ordering the input relation on a list O of order-by attributes. If the order-by attributes' values are uncertain, we have to reason about each tuple t's lowest possible position (the number of tuples that certainly precede it over all possible worlds), and highest possible position (the number of tuples that possibly precede it in at least one possible world). We can naively compute a lower (resp., upper) bound by joining every tuple t with every other tuple, counting pairs where t is certainly (resp., possibly) preceded by its pairing. We refer to this approach as the *rewrite method*, as it can be implemented in SQL. However, the rewrite approach

has quadratic runtime. Inspired by techniques for aggregation over interval-temporal databases such as [37], we propose a one-pass algorithm to compute the bounds on a tuple's position that also supports top-k queries.

Example 4 (Uncertain Sorting and Top-k)**.** *Fig. 1.3 shows a sales DB, extracted from 3 press releases. Uncertainty arises for a variety of reasons, including extraction errors (e.g., D_3 includes term 5) or missing information (e.g., only preliminary data is available for the 4th term in D_1). The task of finding the two terms with the most sales is semantically ambiguous for uncertain data. Several attempts to define semantics include (i) U-top [24] (Fig. 1.3b), which returns the most likely ranked order; (ii) U-rank [24] (Fig. 1.3b), which returns the most likely tuple at each position (term 4 is more likely than any other value for both the 1st and 2nd position); or (iii) Probabilistic threshold queries (PT-k) [38, 39], which return tuples that appear in the top-k with a probability exceeding a threshold (PT), generalizing both possible ($PT > 0$; Fig. 1.3c) and certain ($PT \geq 1$; Fig. 1.3d) answers.*

With the exception of U-Top, none of these semantics return both information about certain and possible results, making it difficult for users to gauge the (i) trustworthiness or (ii) completeness of an answer. Risk assessment on the resulting data is difficult, preventing its use for critical applications, e.g., in the medical, engineering, or financial domains. Furthermore, the outputs of uncertain ranking operators like U-Top are not valid as inputs to further uncertainty-aware queries, because they lose information about uncertainty in the source data. The AU-DB data model naturally encodes query result reliability. By providing each attribute value (and tuple multiplicity) as a range, users can quickly assess the precision of each answer.

Example 5 (AU-DB top-2 query)**.** *Table 1.6a (left) shows an AU-DB, which uses triples, consisting of a lower bound, a selected-guess value (defined shortly), and an upper bound to bound the value range of an attribute (**Term**, **Sales**) and the multi-*

Table 1.6. AU-DB result of the top-2 highest selling terms and rolling sum of sales for the current and next term on the uncertain sales database.

(a) AU-DB bounding the worlds and top-2 result produced by our approach

Term	**Sales**	$\mathbb{N}^3$
1	[2/2/3]	(1,1,1)
2	[2/3/3]	(1,1,1)
[3/3/5]	[4/7/7]	(1,1,1)
4	[4/4/7]	(1,1,1)

Term	**Sales**	`Position`	$\mathbb{N}^3$
1	[2/2/3]	[2/3/3]	(0,0,0)
2	[2/3/3]	[2/2/3]	(0,0,0)
[3/3/5]	[4/7/7]	[0/0/1]	(1,1,1)
4	[4/4/7]	[0/1/1]	(1,1,1)

(b) AU-DB windowed aggregation result produced by our approach

Term	**Sales**	**Sum**	$\mathbb{N}^3$
1	[2/2/3]	[4/5/6]	(1,1,1)
2	[2/3/3]	[6/10/10]	(1,1,1)
[3/3/5]	[4/7/7]	[4/11/14]	(1,1,1)
4	[4/4/7]	[4/4/14]	(1,1,1)

plicity of a tuple ($\mathbb{N}^3$). The AU-DB bounds *all of the possible worlds of our running example. Intuitively, each world's tuples fit into the ranges defined by the AU-DB. The selected-guess values encode one distinguished world (here, D_1) — supplementing the bounds with an educated guess about which possible world correctly reflects the real world* [2]*, providing backwards compatibility with existing systems, and a convenient reference point for users [41, 42]. Table 1.6a (right) shows the result of computing the top-2 answers sorted on* term*. The rows marked in grey encode all tuples*

[2] The process of obtaining a selected-guess world is domain-specific, but [16,40] suggest the most likely world, if it can be feasibly obtained.

that could exist in the top-2 result in some possible world. For example, the tuples $(3,4)$ *(*D_1*),* $(3,7)$ *(*D_2*), and* $(5,7)$ *(*D_3*) are all encoded by the AU-DB tuple* $([3/3/5],[4/7/7]) \rightarrow (1,1,1)$*. Results with a row multiplicity range of (0,0,0) are certainly not in the result. The AU-DB compactly represents an* under-approximation *of* certain answers *and an* over-approximation *of all the* possible answers*, e.g., for our example, the AU-DB admit additional worlds with 5 sales in term 4.*

Implementing windowed aggregation requires determining the (uncertain) membership of each window, which may be affected both by uncertainty in sort position, and in group-by attributes. Furthermore, we have to reason about which of the tuples possibly belonging to a window minimize / maximize the aggregation function result. It is possible implemented this reasoning in SQL, albeit at the cost of range self-joins on the relation (this *rewrite method* is discussed in detail in [43] and evaluated in **??**). We propose a one-pass algorithm for windowed aggregation over AU-DBs, which we will refer to as the *native method*.

The intuition behind our algorithm is to share state between multiple windows. For example, consider the window `ROWS BETWEEN 3 PRECEDING AND CURRENT ROW`. In the deterministic case, with each new window one row enters the window and one row leaves. Sum-based aggregates (`sum`, `count`, `average`) can leverage commutativity and associativity of addition, i.e., updating the window requires only constant time. Similar techniques [44] can maintain of `min`/`max` aggregates in time logarithmic in the window size.

Non-determinism in the row position makes such resource sharing problematic. First, tuples with non-deterministic positions do not necessarily leave the window in FIFO order; We need iteration over tuples sorted on both the upper- and lower-bounds of their position. Second, the number of tuples that could *possibly* belong to the window may be significantly larger than the window size. Considering all possible

rows for a k-row window (using the naive AU-DB aggregation operator [40]) results in a looser bound than if only subsets of size k are considered. For that, we need access to rows possibly in a window sorted on the bounds of the aggregation attribute values (e.g., to find the k-subset with the minimal/maximal sum) in both decreasing order of their upper bound and increasing order of their lower bound. Furthermore, we have to separate maintain tuples that certainly belong to a window (which must contribute to both bounds). To efficiently maintain sets of tuples such that they can be accessed in several sort orders efficiently, we develop a new data structure which we refer to as a *connected heap*. A connected heap is a set of heaps where an element popped from one heap can be efficiently ($O(\log n)$) removed from the other heaps even if their sort orders differ from the heap we popped the element from. This data structure allows us to efficiently maintain sufficient state for computing AU-DB results for windowed aggregation. In preliminary experiments, we demonstrated that, connected heaps significantly outperform a solution based classical heaps.

Example 6 (Windowed Aggregation)**.** *Consider the following windowed aggregation query:*

```
SELECT *, sum(Sales) OVER (ORDER BY term ASC
BETWEEN CURRENT ROW AND 1 FOLLOWING) as sum FROM R;
```

Table 1.6b shows the result of this query over our running example AU-DB. The column ***Sum*** *bounds all possible windowed aggregation results for each AU-DB tuple and the entire AU-DB relation bounds the windowed aggregation result for all possible worlds. Notice that AU-DBs ignore correlations which causes an over-approximation of ranges in the result. For example, term 1 has a maximum aggregation result value of 6 according to the AU-DB representation but the maximum possible aggregation value across all possible world is 5.*

As we will demonstrate, AU-DBs have several beneficial properties that make

them a good fit for dealing with uncertain data that satisfied our requirements:

Efficiency: Query evaluation over AU-DBs is `PTIME`, and by using novel optimizations that compact intermediate results to trade precision for performance and native implementations using sweep based algorithm to process sorting based operations, 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 and windowed 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, ranking queries and windowed aggregation queries.

Compatibility: Like UA-DBs [16], an AU-DB can be constructed from many existing incomplete and probabilistic data models, including C-tables [12] or tuple-independent databases [10], making it possible to re-use existing approaches for exposing uncertainty in data (e.g., [6, 8, 13, 15, 31, 33, 45]). Moreover, although AU-DBs focuses on bag semantics, our model is defined for the same class of semiring-annotated databases [34] as UA-DBs [16] which include, e.g., set semantics, security-annotations, and provenance.

Compactness: As observed elsewhere [35, 46, 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 [48–50].

Simplicity: AU-DBs use simple bounds to convey uncertainty, as opposed to the more complex symbolic formulas of m-tables [48] or tensors [23]. Representing uncertainty as ranges has been shown to lead to better decision-making [42]. AU-DBs can be integrated into uncertainty-aware user interfaces, e.g., Vizier [42, 51].

1.4 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 discusses related work. Chapter 3 introduces backgrounds and notations used. Chapter 4 introduces the UA-DB model and semantics. Chapter 5 introduces the AU-DB model and semantics. Chapter 6 presents and discusses the experimental results. Chapter 7 states conclusions and discusses future plans to finish this thesis. [3]

[3]Partial contents of these chapters are from S.F.'s published works [16, 40, 52] with copy rights to use in this thesis and permitted by all co-authors (B.G., O.K. and A.H.).

D_1	**Term**	**Sales**
	1	2
	2	3
	3	7
	4	4

Term	Sales	**Sum**
1	2	5
2	3	10
3	7	11
4	4	4

Term
4
3

(a) U-Top

D_2	**Term**	**Sales**
	1	3
	2	2
	3	4
	4	6

Term	Sales	**Sum**
1	3	5
2	2	6
3	4	10
4	6	6

Term
4
4

(b) U-Rank

Term
3
4
5

(c) PT(0)

D_3	**Term**	**Sales**
	1	2
	2	2
	5	4
	4	7

Term	Sales	**Sum**
1	2	4
2	2	9
5	4	4
4	7	11

Term
4

(d) PT(1)

Figure 1.3. An uncertain sales database with three possible worlds (with probability .4, .3 and .3 respectively) with top-2 highest selling terms high-lighted.

CHAPTER 2

RELATED WORK

We build on prior research in incomplete and probabilistic databases, certain answers, uncertain aggregation, uncertain top-k and uncertain sorting.

2.1 Incomplete and probabilistic data models

Uncertainty was recognized as an important problem by the database community early-on. Codd [53] extended the relational model with null values to represent missing information and proposed to use 3-valued logic to evaluate queries over databases with null values. Imielinski [12] introduced V-tables and C-tables as representations of incompleteness. C-tables are closed under full relational algebra. Reiter [36] proposed to model databases as logical theories, a model equivalent to V-tables. Abiteboul [54] defined update operations over incomplete databases. Underlying all these models is the possible world semantics. Probabilistic data models quantify the uncertainty in incomplete databases by assigning probabilities to individual possible worlds. TI-DBs [10] are a prevalent model for probabilistic data where each tuple is associated with its marginal probability and tuples are assumed to be independent. Green et al. [55] studied probabilistic versions of C-tables. Virtual C-tables generalize C-tables [6, 21] by allowing symbolic expressions as values.

2.2 Probabilistic Query Processing

Probabilistic query processing (PQP) has been a field of research for several decades (e.g., an important survey is [10]). Computing the marginal probability of a query result tuple can be reduced to weighted model counting and, thus, is `#P` in general [56]. Most practical approaches for PQP are either limited to queries which can

be answered in PTIME (so-called *safe* queries) and/or compute approximate probabilities for query answers (e.g., [57]). Systems implementing PQP include Sprout [58], Trio [17], MCDB [59], Mimir [60], MYSTIQ [61], and many others. Of particular note, Gatterbauer and Suciu [62] showed that (efficient) extensional evaluation semantics compute a lower bound on result probabilities.

2.3 Annotated Databases

Green et al. [34] introduced the semiring annotation framework that we utilize in this work. The connection between annotated databases, provenance, and uncertainty has been recognized early-on. A particular type of semiring annotations, often called Lineage, has been used for probabilistic query processing (e.g., see [10, 63]). Green et al. [34] observed that set semantics incomplete databases can be expressed as $\mathcal{K}$-relations by annotating each tuple with the set of worlds containing it. We define a more general type of incomplete databases based on $\mathcal{K}$-relations which is defined for any l-semiring. Kostylev et al. [64] investigate how to deal with dependencies among annotations from multiple domains. Similar to [64], we consider "multi-dimensional" annotations, but for a very different purpose: to extend incomplete databases beyond set semantics.

2.4 Approximations of Certain Answers

Queries over incomplete databases typically use certain answer semantics [11, 12,35,46,47] first defined in [76]. Computing certain answers is coNP-complete [11,12] (data complexity) for relational algebra. Several techniques for computing an under-approximation (subset) of certain answers have been proposed. Reiter [36] proposed a PTIME algorithm for positive existential queries. Guagliardo and Libkin [35, 46, 47] proposed a scheme for full relational algebra for Codd- and V-tables, and also studied bag semantics [46, 77]. Feng et. al. [16] generalized this approach to new query

semantics through Green et. al.'s $\mathcal{K}$-relations [34]. m-tables [48] 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) [31, 33] 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., [15,30,78]) and several combinations of classes of constraints and queries permit first-order rewritings [13,32,79,80]. Geerts et. al. [13] study first-order under-approximations of certain answers in the context of CQA. Notably, AU-DBs build on the approach of [16] (i.e., a selected guess and lower bounds), adding an upper bound on possible answers (e.g., as in [46]) to support aggregations, and bound attribute-level uncertainty with ranges instead of nulls.

2.5 Aggregation in Incomplete/Probabilistic Databases

While aggregation of uncertain data has been studied extensively (see Table 2.1 for a comparison of approaches), general solutions remain an open problem [9]. A key challenge lies in defining a meaningful semantics, as aggregates over uncertain data frequently produce empty certain answers [72]. An alternative semantics adopted for CQA and ontologies [18, 19, 65, 72, 75] returns per-attribute bounds over all possible results [18] 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 [20, 21, 28, 66, 69–71, 74]. 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 [68]).

Efforts to create a lossless symbolic encoding closed under aggregation [22, 67] 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 [23] 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 [59] queries sampled possible worlds. In principle, this approach supports arbitrary queries, but is significantly slower than SGQP [16], 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 [48, 49] 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).

2.6 Uncertain Top-k

A key challenge in uncertain top-k ranking is defining semantics that intuitively generalize the single score value of deterministic top-k; The set of tuples certainly (resp., possibly) in the top-k may have fewer (more) than k tuples. U-Topk [24] picks the top-k set with the highest probability. U-kRanks [24] assigns to each rank the tuple which is most-likely to have this rank. Global-Topk [39] first ranks tuples by probability of being in the top-k and returns the k most likely. Probabilistic threshold top-k [38] returns all tuples that have a probability of being in the top-k

that exceeds a pre-defined threshold. Expected rank [25] calculates the expected rank for each tuple across all possible worlds and picks the top-k. Ré et al. [26] proposed a multisimulation algorithm that stops when a guaranteed top-k probability can be guaranteed. Soliman et al. [81] proposed a framework that integrates tuple retrieval, grouping, aggregation, uncertainty management, and ranking in a pipelined fashion. Li et al. [27] proposed a unified ranking approach for top-k based on generating functions which use and/xor trees to reason over complex correlations. Each of these generalizations necessarily breaks some intuitions about top-k, producing more (or fewer) than k tuples, or producing results that are not the top-k in any individual world.

2.7 Uncertain Order

In contrast to uncertain top-k, which focuses on reasoning about the existence of result tuples, uncertain ordering focuses on annotating tuples with logical orders. Amarilli et. al. developed an approach based on a partially ordered relational model [82, 83]. For more general use cases where posets can not represent all possible worlds, Amarilli et. al. also develop a symbolic model of provenance [84] that can materialize uncertain sort orders in a column. Both approaches are limited to set semantics.

2.8 Temporal Aggregation

Similar to the ranges representations in AU-DBs, temporal databases also need to reason about tuples associated with partially overlapping intervals. For example, a window aggregate (both deterministic or uncertain) may be recast as an interval self-join, where one table defines the set of windows and each of its tuples is joined with the tuples in the window. Accordingly, we take inspiration from temporal databases for our own operator implementations. The first temporal aggregation algorithm was

given in [85]. Moon et al. [86] proposed a balanced tree algorithm for **count**, **sum** and **avg** aggregates, and a divide-and-conquer algorithm for **min** and **max**. Kline and Snodgrass proposed the *aggregation tree* [87], an in-memory data structure that supports incremental computation of temporal aggregates. Yang et al. [88] proposed a materialized version called the *SB-tree* that can be used as an index for incremental temporal aggregation computations. The *MVSB-tree* [89] is and extension of the SB-tree that supports predicates in the aggregation query. Piatov and Helmer [37] proposed a sweep-line based approach the reduces the space needed to compute **min** and **max** aggregates over temporal data.

Table 2.1. Comparison of approaches for aggregation over uncertain data. Features include the ability to *chain* aggregates, support *having* predicates, and support *group*ing. FD: Functional Dependency Repair, TGD: Source-Target tgds, C-Tb: C-Table, X-Tb: X-Table (aka Block-Independent), TI: Tuple-Independent, V-Tb: V-Table, IA: Tables (or horizontal partitions) are annotated to indicate whether (i) their attribute values are correct, whether they may not contain all certain tuples, and whether they contain tuples that are not certain, GLB only: under-approximates certain answers (a lower bound on every possible world), GLB+LUB: under-approximates certain answers and over-approximates possible answers (an upper bound on every possible world). [1]: Probabilistic XML analogous to C-Tables. [2]: the input is an entity resolution problem that can be represented as using X-Tables.

Approach	**Aggregates**			**Features**			**Input**	**Output**	**Complexity**
	Sum/Cnt	**Avg**	**Min/Max**	**Chain**	**Having**	**Group**			
Arenas et. al. [18]	✓	×	✓	×	×	×	FD	GLB+LUB	NP-hard
Fuxmann et. al. [65]	✓	×	✓	×	×	✓	FD	GLB+LUB	coNP-hard / PTIME
Afrati et. al. [19]	✓	✓	✓	×	×	×	TGD	GLB+LUB	NP-hard / PTIME
Fink et. al. [22]	✓	×	✓	✓	✓	✓	C-Tb	Symbolic	NP-hard
Murthy et. al. [20]	✓	✓	✓	×	×	×	X-Tb	GLB+LUB / Moments	NP-hard / PTIME
Abiteboul et. al. [66]	✓	✓	✓	×	×	×	C-Tb[1]	GLB+LUB / Moments	NP-hard
Lechtenborger et. al. [67]	✓	×	×	✓	✓	✓	C-Tb	Symbolic	NP-hard
Re et. al. [68]	⊢— `HAVING` only —⊣			✓	✓	✓	TI	Moments	NP-hard / PTIME
Soliman et. al. [28]	⊢— `TOP-K` only —⊣			×	×	✓	C-Tb	Moments	NP-hard / PTIME
Chen et. al. [69]	✓	✓	✓	×	×	✓	X-Tb	GLB+LUB	NP-hard / PTIME
Jayram et. al. [70]	✓	✓	✓	×	×	×	X-Tb	Moments	PTIME (approx)
Burdick et. al. [71]	✓	✓	×	×	×	✓	X-Tb	Moments	PTIME (approx)
Calvanese et. al [72]	✓	×	×	×	×	×	FD	GLB only	NP-hard
Kostylev et. al. [73]	⊢— `COUNT` / `DISTINCT` —⊣			×	×	×	FD	GLB only	coNP-complete
Yang et. al. [74]	⊢— Agg Constraint only —⊣			×	×	×	X-Tb	Sample of Input	coNP-complete
Jampani et. al. [59]	⊢— No restrictions —⊣			✓	✓	✓	V-Tb	Output Sample	PTIME (approx)
Kennedy et. al. [21]	✓	✓	✓	×	✓	×	C-Tb	Output Sample	PTIME (approx)
Lang et. al. [49]	✓	✓	✓	✓	✓	✓	IA	IA	Data-Indep./PTIME
Sismanis et al. [75]	✓	✓	✓	×	×	✓	X-Tb[2]	GLB+LUB	PTIME (approx)
Our apporach	✓	✓	✓	✓	✓	✓	Any	GLB+LUB	PTIME (approx)

CHAPTER 3

BACKGROUND

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

3.1 Possible Worlds Semantics

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

Example 7. *Figure 3.1 shows the two possible worlds in the result of the spatial join. Observe that some tuples (e.g.,* $(1, Lasalle, NY)$*) appear in all worlds. Such tuples are called* ***certain****. Tuples that appear in at least one possible world (e.g.,* $(2, Tuscon, AZ)$*) are called* ***possible****.*

Decades of research [10, 12, 17, 55, 61, 90] has explored query processing over incomplete databases. These techniques commonly adopt the *"possible worlds"* se-

Table 3.1. Example incomplete database $\mathcal{D} = \{D_1, D_2\}$.

(a) D_1

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

(b) D_2

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

mantics: The result of evaluating a query Q over an incomplete database is the set of relations resulting from evaluating Q over each possible world individually using deterministic semantics.

$$Q(\mathcal{D}) \coloneqq \{\ Q(D) \mid D \in \mathcal{D}\ \} \tag{3.1}$$

Example 8. *$Q_{NY} \coloneqq \sigma_{state='NY'}(\mathcal{D})$ returns locations in NY State from the database $\mathcal{D}$ shown in Figure 3.1. The result of $Q_{NY}(\mathcal{D})$ is the set of worlds computed by evaluating Q_{NY} over each world of $\mathcal{D}$ as shown in Figure 3.2. Observe that the location with id 2 appears in $Q_{NY}(D_2)$, but not $Q_{NY}(D_1)$.*

3.2 Certain and Selected-Guess Answers

An important goal of query processing over incomplete databases is to differentiate query results that are certain from ones that are merely possible. Formally, a tuple is certain if it appears in every possible world. [12, 76]:

$$\mathit{certain}(\mathcal{D}) \coloneqq \{t \mid \forall D \in \mathcal{D} : t \in D\} \tag{3.2}$$

$$\mathit{possible}(\mathcal{D}) \coloneqq \{t \mid \exists D \in \mathcal{D} : t \in D\} \tag{3.3}$$

In contrast to [12], which studies certain answers to queries, we define certainty at the instance level. These approaches are equivalent since we can compute the

Table 3.2. The result of a query Q over an incomplete database $\mathcal{D}$ is the set of results in all worlds $D \in \mathcal{D}$.

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

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

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

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

certain answers of query Q over incomplete instance $\mathcal{D}$ as $certain(Q(\mathcal{D}))$. Although computing certain answers is coNP-hard [11] in general, there exist `PTIME` under-approximations [35, 36, 47].

Selected Guess Query Processing: As mentioned in the introduction, another approach commonly used in practice is to select one possible world. Queries are evaluated solely in this world, and ambiguity is ignored or documented outside of the database. We refer to this approach as *best-guess query processing* (SGQP) [6] since typically one would like to select the possible world that is deemed most likely.

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 $\textsc{Sch}(D) = \{\text{Sch}(R_1), \ldots, \text{Sch}(R_n)\}$ is a set of relation schemas $\text{Sch}(R_i) = (A_1, \ldots, A_n)$. The arity $arity(\textsc{Sch}(R))$ of $\textsc{Sch}(R)$ is the number of attributes in $\textsc{Sch}(R)$. An instance D for database schema $\textsc{Sch}(D)$ is a set of relation instances with one relation for each relation schema in $\textsc{Sch}(D)$: $D = \{R_1, \ldots, R_n\}$. 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 $\mathbb{D}$.[4]

3.3 K-Relations

The generalization of incomplete databases we use here is based on $\mathcal{K}$-**relations** [34]. In this framework, relations are annotated with elements from the domain K of a (commutative) semiring $\mathcal{K} = (K, +_\mathcal{K}, \cdot_\mathcal{K}, \mathbb{1}_\mathcal{K}, \mathbb{0}_\mathcal{K})$, 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 $R(t)$.

The specific information encoded by an annotation depends on the choice of semiring. For instance, bag and set relations can be encoded as semirings: the natural numbers ($\mathbb{N}$) with addition and multiplication, $(\mathbb{N}, +, \times, 0, 1)$, annotates each tuple with its multiplicity; and boolean constants $\mathbb{B} = \{T, F\}$ with disjunction and conjunction, $(\mathbb{B}, \vee, \wedge, F, T)$, annotates each tuple with its set membership. Abusing notation, we often use $\mathcal{K}$ to denote both the domain and the corresponding semiring.

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

---

[4]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).

$$\textbf{Union:}\quad (R_1 \cup R_2)(t) = R_1(t) +_{\mathcal{K}} R_2(t)$$

$$\textbf{Join:}\quad (R_1 \sqcup R_2)(t) = R_1(t[\textsc{Sch}(R)_1]) \cdot_{\mathcal{K}} R_2(t[\textsc{Sch}(R)_2])$$

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

$$\textbf{Selection:}\quad (\sigma_\theta(R))(t) = R(t) \cdot_{\mathcal{K}} \theta(t)$$

For simplicity we assume in the definition above that tuples are of a compatible schema (e.g., $\textsc{Sch}(R)_1$ for a union $R_1 \cup R_2$). We use $\theta(t)$ to denote a function that returns $\mathbb{1}_{\mathcal{K}}$ iff θ evaluates to true over tuple t and $\mathbb{0}_{\mathcal{K}}$ otherwise.

A **homomorphism** is a mapping $h : \mathcal{K} \to \mathcal{K}'$ from a semiring $\mathcal{K}$ to a semiring $\mathcal{K}'$ that maps $\mathbb{0}_{\mathcal{K}}$ and $\mathbb{1}_{\mathcal{K}}$ to their counterparts in $\mathcal{K}'$ and distributes over sum and product (e.g., $h(k +_{\mathcal{K}} k') = h(k) +_{\mathcal{K}'} h(k')$). Any homomorphisms h can be lifted from semirings to $\mathcal{K}$-relations or $\mathcal{K}$-databases by applying h to the annotation of every tuple t: $h(R)(t) = h(R(t))$. We will use the same symbol for a homomorphism and its lifted variants. Importantly, queries commute with semiring homomorphisms: $h(Q(D)) = Q(h(D))$.

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 $\mathbb{N}$. Formally, $k \preceq_{\mathcal{K}} k'$ if it is possible to obtain k' by adding to k: $\exists k'' : k +_{\mathcal{K}} k'' = k'$. Semirings for which the natural order is a partial order are called *naturally ordered* [91].

$$\forall k, k' \in K : \left(k \preceq_{\mathcal{K}} k'\right) \Leftrightarrow \left(\exists k'' \in K : k +_{\mathcal{K}} k'' = k'\right) \tag{3.4}$$

3.4 Incomplete K-relations

Many incomplete data models do not support bag semantics. Our first contribution unifies set and bag semantics under a joint framework. Recall that an

incomplete database is a set of deterministic databases (possible worlds). We now generalize this idea to $\mathcal{K}$-databases.

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

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

$$Q(\mathcal{D}) \coloneqq \{\ Q(D) \mid D \in \mathcal{D}\ \} \tag{3.5}$$

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

Example 9. *Consider a $\mathbb{N}$-database $\mathcal{D}$ (bag semantics) containing a relation **LOC** with two attributes **locale** and **state**. Assume that $\mathcal{D}$ consists of the two possible worlds below:*

Table 3.3. ex. 9

LOC in D_1

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

LOC in D_2

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

Using semiring $\mathbb{N}$ *each tuple in a possible world is annotated with its multiplicity (the number of copies of the tuple that exist in the possible world). Arguably, tuples* `(Lasalle, NY)` *and* `(Tucson, AZ)` *are certain since they appear (multiplicity higher than* 0*) in both possible worlds while* `(Greenville, IN)` *is not since it is not present (its multiplicity is zero) in possible world* D_1[5]*. However, the boolean interpretation of certainty in incomplete databases is not suited to* $\mathbb{N}$*-relations (or* $\mathcal{K}$*-relations in general) because it ignores the annotations of tuples. In this particular example, tuple* `(Lasalle, NY)` *appears with multiplicity* 3 *in possible world* D_1 *and multiplicity* 2 *in possible world* D_2*. We can state with certainty that in every possible world this tuple appears* at least *twice. Thus,* 2 *is a lower bound (the greatest lower bound) for the annotation of* `(Lasalle, NY)`*. Following this logic, we will define certainty through greatest lower bounds (GLBs) on tuple annotations.*

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

To define a sensible lower bound for annotations, we need an order relation for semiring elements. We use the natural order $\preceq_{\mathcal{K}}$ as introduced in Section 3.3 to define the GLB and LUB of a set of $\mathcal{K}$-elements. For a well-defined GLB, we require

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

that $\preceq_{\mathcal{K}}$ forms a lattice over K, a property that makes $\mathcal{K}$ an *l-semiring* [64]. A lattice over a set S and with a partial order $\leq_S$ is a structure $(S, \sqcup, \sqcap)$ where $\sqcap$ (the greatest lower bound) and $\sqcup$ (the lowest upper bound) are operations over S defined for all $a, b \in S$ as:

The least upper bound $\sqcup$ is defined symmetrically.

$$a \sqcup b \coloneqq \min_{\leq_S}(\{c \mid c \in S \wedge a \leq_S c \wedge b \leq_S c\})$$

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

In a lattice, $\sqcup$ and $\sqcap$ are associative, commutative, and fulfill

$$a \sqcup (a \sqcap b) = a \qquad\qquad a \sqcap (a \sqcup b) = a$$

We will use $\sqcap_{\mathcal{K}}$ and $\sqcup_{\mathcal{K}}$ to denote the $\sqcap$ and $\sqcup$ operation of the lattice over $\preceq_{\mathcal{K}}$ for a semiring $\mathcal{K}$. Abusing notation, we will apply the $\sqcap_{\mathcal{K}}$ and $\sqcup_{\mathcal{K}}$ operations iteratively to sets of elements. e.g., $\sqcap_{\mathcal{K}}\{k_1, k_2, k_2\} = (k_1 \sqcap_{\mathcal{K}} k_2) \sqcap_{\mathcal{K}} k_3$. This is well-defined for l-semirings, since in a lattice any set of elements has a unique greatest lower bound and lowest upper bound based on the associativity and commutativity laws of lattices. That is, no matter in which order we apply $\sqcap_{\mathcal{K}}$ to the elements of a set, the result will be the same. From here on, we will limit our discussion to l-semirings. Many semirings, including the set semiring $\mathbb{B}$ and the bag semiring $\mathbb{N}$ are l-semirings. The natural order of $\mathbb{B}$ is $F \preceq_{\mathbb{B}} T$, $k_1 \sqcup_{\mathbb{B}} k_2 = k_1 \vee k_2$, and $k_1 \sqcap_{\mathbb{B}} k_2 = k_1 \wedge k_2$. The natural order of $\mathbb{N}$ is the standard order of natural numbers, $k_1 \sqcup_{\mathbb{N}} k_2 = \max(k_1, k_2)$, and $k_1 \sqcap_{\mathbb{N}} k_2 = \min(k_1, k_2)$.

We define the certain and possible annotation $\textsc{cert}_{\mathcal{K}}(\mathcal{D}, t)$ of a tuple t in an incomplete $\mathcal{K}$-database $\mathcal{D}$ by gathering the annotations of tuple t from all possible

worlds of $\mathcal{D}$ and then applying $\sqcap_{\mathcal{K}}$ to compute the greatest lower bound.

$$\textsc{cert}_{\mathcal{K}}(\vec{k}) \coloneqq \sqcap_{\mathcal{K}}(\vec{k}) \qquad \textsc{cert}_{\mathcal{K}}(\mathcal{D}, t) = \textsc{cert}_{\mathcal{K}}(\mathcal{D}(t))$$

$$\textsc{poss}_{\mathcal{K}}(\vec{k}) \coloneqq \sqcup_{\mathcal{K}}(\vec{k}) \qquad \textsc{poss}_{\mathcal{K}}(\mathcal{D}, t) = \textsc{poss}_{\mathcal{K}}(\mathcal{D}(t))$$

Importantly, GLB coincides with the standard definition of certain answers for set semantics ($\mathbb{B}$): $\textsc{cert}_{\mathbb{B}}$ returns true only when the tuple is present in all worlds. We also note that $\textsc{cert}_{\mathbb{N}} = \min$, is analogous to the definition of certain answers for bag semantics from [47]. For instance, consider the certain annotation of the first tuple from Example 9. The tuple's certain multiplicity is $\textsc{cert}_{\mathbb{N}}(\{2, 3\}) = min(2, 3) = 2$. Similarly, for the third tuple, $\textsc{cert}_{\mathbb{N}}(\{0, 5\}) = 0$. Reinterpreted under set semantics, all tuples that exist (multiplicity > 0) are annotated *true* (T) and all others $false$ (F). For the first tuple we get, $\sqcap_{\mathbb{B}}(\{T, T\}) = T \wedge T = T$ (certain). For the third tuple we get $\sqcap_{\mathbb{B}}(\{F, T\}) = F \wedge T = F$ (not certain).

$\mathcal{K}_W$-relations: For the formal exposition in the remainder of this work it will be useful to define an alternative, but equivalent, encoding of an incomplete $\mathcal{K}$-database as a single $\mathcal{K}$-database using a special class of semirings whose elements encode the annotation of a tuple across a set of possible worlds[6]. We assume a fixed set $W = \{m \mid m \in \mathbb{N} \wedge 0 < m \leq n\}$ of possible world identifiers for some number of possible worlds $n \in \mathbb{N}$. Given the domain K of a semiring $\mathcal{K}$, we write K^W to denote the set of elements from the n-way cross-product of K. We annotate tuples t with elements of K^W to store annotations of t in each possible world. We use $\vec{k}$, $\vec{k_1}$, ... to denote elements from K^W to make explicit that they are vectors.

Definition 2 (Possible World Semiring)**.** *Let $\mathcal{K} = (K, +_{\mathcal{K}}, \cdot_{\mathcal{K}}, \mathbb{0}_{\mathcal{K}}, \mathbb{1}_{\mathcal{K}})$ be an l-semiring. We define the possible world semiring $\mathcal{K}_W = (K^W, +_{\mathcal{K}_W}, \cdot_{\mathcal{K}_W}, \mathbb{0}_{\mathcal{K}_W}, \mathbb{1}_{\mathcal{K}_W})$. The opera-*

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

tions of this semiring are defined as follows

$$\forall i \in W: \quad \mathbb{0}_{\mathcal{K}_W}[i] = \mathbb{0}_{\mathcal{K}}$$

$$\forall i \in W: \quad \mathbb{1}_{\mathcal{K}_W}[i] = \mathbb{1}_{\mathcal{K}}$$

$$\forall i \in W: \quad (\vec{k_1} +_{\mathcal{K}_W} \vec{k_2})[i] = \vec{k_1}[i] +_{\mathcal{K}} \vec{k_2}[i]$$

$$\forall i \in W: \quad (\vec{k_1} \cdot_{\mathcal{K}_W} \vec{k_2})[i] = \vec{k_1}[i] \cdot_{\mathcal{K}} \vec{k_2}[i]$$

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

Example 10. *Reconsider the incomplete* $\mathbb{N}$*-relation from Example 9. The encoding of this database as a* $\mathbb{N}^2$*-relation is:*

Table 3.4. ex. 10

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

Translating between incomplete $\mathcal{K}$-databases and $\mathcal{K}_W$-databases is trivial. Given an incomplete $\mathcal{K}$-database with n possible worlds $\{D_i\}$, we create the corresponding $\mathcal{K}_W$-database by annotating each tuple t with the vector $[D_1(t), \ldots, D_n(t)]$. In the other direction, given a $\mathcal{K}_W$-database $\mathcal{D}$ with vectors of length n, we construct the corresponding incomplete $\mathcal{K}$-database by annotating each tuple t with $\mathcal{D}(t)[i]$ in possible world D_i. In addition, we will show below that queries over $\mathcal{K}_W$-databases encode possible world semantics. Thus, the following result holds and we can use incomplete $\mathcal{K}$- and $\mathcal{K}_W$-databases interchangeably.

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

Observe that $\mathcal{K}_W$ is a semiring, since we define $\mathcal{K}_W$ using the $|W|$-way version of the *product* operation of universal algebra, and products of semirings are also semirings [92].

Possible Worlds: We can extract the $\mathcal{K}$-database for a possible world (e.g., the *selected-guess world*) from a $\mathcal{K}_W$-database by projecting on one dimension of its annotations. This can be modeled as a mapping $\text{PW}_i : K^W \to K$ where $i \in W$:

$$\text{PW}_i(\vec{k}) \coloneqq \vec{k}[i] \tag{3.6}$$

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

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

Recall from Section 3.3 that a mapping between semirings commutes with queries iff it is a semiring homomorphism. Note that $\mathcal{K}_W$-relations admit a trivial extension to probabilistic data by defining a distribution $P : W \mapsto [0, 1]$. See [93] for details.

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

Proof: See Appendix A.1

Probabilistic Data: $\mathcal{K}_W$-relations admit a trivial extension to probabilistic data by defining a distribution $P : W \mapsto [0, 1]$ such that $\sum_{i \in W} P(i) = 1$. In contrast to classical frameworks for possible worlds, where the collection of worlds is a *set*, $\mathcal{K}_W$ queries preserve the same $|W|$ possible worlds[7]. Hence, the input distribution P applies, unchanged, to the $|W|$ possible query outputs.

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

$$\text{CERT}_{\mathcal{K}}(\vec{k}) = \sqcap_{\mathcal{K}}(\vec{k}) \qquad \text{CERT}_{\mathcal{K}}(\mathcal{D}, t) = \text{CERT}_{\mathcal{K}}(\mathcal{D}(t))$$

$$\text{POSS}_{\mathcal{K}}(\vec{k}) = \sqcup_{\mathcal{K}}(\vec{k}) \qquad \text{POSS}_{\mathcal{K}}(\mathcal{D}, t) = \text{POSS}_{\mathcal{K}}(\mathcal{D}(t))$$

We use possible world, certain and possible notations to show that our works bound either certain or both certain and possible answers, thus bound the incomplete database as input and the query result.

[7]Although it has no impact on our results, it is worth noting that the worlds in a $\mathcal{K}_W$ query result may not be distinct.

CHAPTER 4

UA-DBs SEMANTICS

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

4.1 UA-semirings

We define a UA-semiring as a $\mathcal{K}^2$-semiring, i.e., the direct product of a semiring $\mathcal{K}$ with itself (see Section 4.1). In the following we will write kk' instead of $k \cdot_{\mathcal{K}} k'$ if the semiring $\mathcal{K}$ is clear from the context. Recall that operations in $\mathcal{K}^2 = (K^2, +_{\mathcal{K}^2}, \cdot_{\mathcal{K}^2}, \mathbb{0}_{\mathcal{K}^2}, \mathbb{1}_{\mathcal{K}^2})$ are defined pointwise, e.g., $[k_1, k_1{}'] \cdot_{\mathcal{K}^2} [k_2, k_2{}'] = [k_1 \cdot_{\mathcal{K}} k_2, k_1{}' \cdot_{\mathcal{K}} k_2{}']$.

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

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

of semirings are semirings.

4.2 Creating UA-DBs

We now discuss how to derive UA-relations from a $\mathcal{K}_W$-database or a compact encoding of a $\mathcal{K}_W$-database using some uncertain data model like c-tables. Consider a $\mathcal{K}_W$-database $\mathcal{D}$, let D be one of its worlds and $\mathcal{D}^{\downarrow}$ a $\mathcal{K}$-database under-approximating the certain annotations of $\mathcal{D}$. We refer to $\mathcal{D}^{\downarrow}$ as a *certain lower-bound* and will study such bound in depth in Sec. 4.4 and Sec. 4.6. We cover in Sec. 4.5 how to generate a UA-DB from common uncertain data models by extracting a (best-guess) world D and an under approximation $\mathcal{D}^{\downarrow}$. We construct a UA-DB D_{UA} as an *encoding* of D and $\mathcal{D}^{\downarrow}$ by setting for every tuple t:

$$D_{UA}(t) \coloneqq [\mathcal{D}^{\downarrow}(t), D(t)]$$

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

$$h_{cert}([c,d]) \coloneqq c \quad h_{det}([c,d]) \coloneqq d$$

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

4.3 Querying UA-DBs

We now state the main result of this section: query evaluation over UA-DBs preserves the under-approximation and over-approximation of certain annotations. To prove the main result, we first show that h_{cert} and h_{det} are homomorphisms, because this implies that queries over UA-DBs are evaluated over the c and the d component of an annotation independently. Thus, we can prove the result for under- and over-approximations separately. For over-approximation we can trivially

show an even better result: By definition (Section 3.4) the possible world used as an over-approximation is preserved exactly. Hence, the over-approximation property is preserved and UA-DBs are also backwards compatible with SGQP. For under-approximations we have to show that query evaluation preserves under-approximations. This part is more involved and we will prove this result in Section 4.6.

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

Proof: See Appendix A.1

4.4 Approximations of certain (bounding)

We now define certainty apprximations, which are $\mathcal{K}$-databases whose annotations approximate certain annotations of tuples in a $\mathcal{K}_W$-database with respect to the natural order of semiring $\mathcal{K}$. A approxiation scheme is a mapping from an incomplete databases to bounds.

Definition 4 (Uncertainty approximation Scheme)**.** *Let $\mathcal{DB}_{\mathcal{K}}$ be the set of all $\mathcal{K}$-databases, $\mathcal{M}$ an incomplete/probabilistic data model, and $\mathcal{DB}_{\mathcal{M}}$ the set of all possible instances of this model. A certain approximation scheme is a function trans^{UADB} : $\mathcal{DB}_{\mathcal{M}} \rightarrow \mathcal{DB}_{\mathcal{K}}$ such that the transfer $\mathcal{D}^{\downarrow} = \mathsf{trans}^{AUDB}(\mathcal{D})$ has the schema* $\textsc{Sch}(D)$.

Ideally, we would like the annotation $\mathcal{D}^{\downarrow}(t)$ of a tuple t from an lower-bound $\mathcal{D}^{\downarrow}$ to be exactly $\textsc{cert}_{\mathcal{K}}(\mathcal{D}, t)$. Observe that an exact bound can always be computed in $O(\mathsf{W})$ time if all worlds of the incomplete database can be enumerated. However, the number of possible worlds is frequently exponential in the data size. Thus, most incomplete data models rely on factorized encodings, with size typically logarithmic in W. Ideally, we would like bounding schemes to be `PTIME` in the size of the encoding

(rather than in W). As mentioned in the introduction, computing certain answers is coNP-complete, so for tractable query semantics we must accept that $\mathcal{D}^{\downarrow}(t)$ may either over- or under-approximate $\textsc{cert}_{\mathcal{K}}(\mathcal{D}, t)$ (with respect to $\preceq_{\mathcal{K}}$). For instance, under bag semantics (semiring $\mathbb{N}$), a label n may be smaller or larger than the certain multiplicity of a tuple. We call a approximation *certain lower-bound* (no false positives) if it consistently under-approximates the certain annotation of tuples, *certain upper-bound* (no false negatives) if it consistently over-approximates certainty, and *exact certain* if it annotates every tuple with its certain annotation. We also apply this terminology to approximation schemes, e.g., a certain lower-bound scheme only produces certain lower-bound approximations. For UA-DBs we are mainly interested in certain lower-bound approximations to provide an under-approximation of certain annotations.

Definition 5. *If $\mathcal{D}^{\downarrow}$ is an uncertainty approximation for $\mathcal{D}$.*

We call $\mathcal{D}^{\downarrow}$...	***...iff for all tuples $t \in \mathcal{D}$...***
certain lower-bound	$\mathcal{D}^{\downarrow}(t) \preceq_{\mathcal{K}} \textsc{cert}_{\mathcal{K}}(\mathcal{D}, t)$
certain upper-bound	$\textsc{cert}_{\mathcal{K}}(\mathcal{D}, t) \preceq_{\mathcal{K}} \mathcal{D}^{\downarrow}(t)$
exact certain	$\textsc{cert}_{\mathcal{K}}(\mathcal{D}, t) = \mathcal{D}^{\downarrow}(t)$

A approximation is both certain lower-bound and certain upper-bound iff it is exact certain. Ideally, queries over approximations would preserve these bounding effects.

Definition 6 (Preservation of Bounds)**.** *A query semantics for uncertainty approximations preserves a property X (certain lower-bound, certain upper-bound, or exact certain) wrt. a class of queries $\mathcal{C}$, if for any incomplete database $\mathcal{D}$, approximation $\mathcal{D}^{\downarrow}$ for $\mathcal{D}$ that has property X, and query $Q \in \mathcal{C}$ we have: $Q(\mathcal{D}^{\downarrow})$ is an uncertainty approximation for $Q(\mathcal{D})$ with property X.*

4.5 Approximation Schemes

We define efficient (PTIME) apprixmation schemes for three existing incomplete data models and their probabilistic extensions: Tuple-Independent probabilistic databases [10], the disjoint-independent x-relation model from [17], and (P)C-Tables [12]. We also show how to extract a selected-guess world from an $\mathcal{K}_W$-database derived from these models. Since computing certain answers is hard in general, our PTIME approximation schemes cannot be exact certain for all models.

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

$$\mathcal{D}^{\downarrow}(t) \coloneqq T \Leftrightarrow t \text{ is not marked as optional} \tag{4.1}$$

Theorem 2 ($\mathsf{trans}^{UADB}_{\text{TI-DB}}$ is exact certain)**.** *Given a TI-DB $\mathcal{D}$, $\mathsf{trans}^{UADB}_{\text{TI-DB}}(\mathcal{D})$ is an exact certain approximation.*

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

C-tables: *C-Tables* [12] use a set Σ of variable symbols to define possible worlds. Tuples are annotated by a boolean expression over comparisons of values from $\Sigma \cup \mathbb{D}$,

called the local condition. Each variable assignment $v : \Sigma \to \mathbb{D}$ satisfying a boolean expression called the global condition defines a possible world, derived by retaining only tuples with local conditions satisfied under v. Computing certain answers for first order queries is coNP-complete [11, 94] even for Codd-tables. Since the result of any first order query over a Codd-table can be represented as a C-table and evaluating a query in this fashion is efficient, it follows that determining whether a tuple is certain in a C-table cannot be in PTIME. Instead, consider the following sufficient, but not necessary condition for a tuple t to be certain. If (1) a tuple t in a C-table contains only constants and (2) its local condition $\phi_{\mathcal{D}}(t)$ is a tautology, then the tuple is certain. To see why this is the case, recall that under the closed-world assumption, a C-table represents a set of possible worlds, one for each valuation of the variables appearing in the C-table (to constants from $\mathbb{D}$). A tuple is part of a possible world corresponding to such a valuation if the tuple's local condition is satisfied under the valuation. Thus, a tuple consisting of constants only, with a local condition that is a tautology is part of every possible world represented by the C-table. If the local condition of a tuple is in conjunctive normal form (CNF) then checking whether it is a tautology is efficient (PTIME). Our approximation for C-tables applies this sufficient condition and, thus, is a certain lower-bound. Formally, $\mathcal{D}^{\downarrow} = \mathsf{trans}^{UADB}_{\text{C-TABLE}}(\mathcal{D})$, where for a C-table $\mathcal{D}$ and any tuple $t \in \text{TupDom}$:

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

Green et. al. [55] introduced PC-tables a probabilistic version of C-tables where each variable is associated with a probability distribution over its possible values. Variables are considered independent of each other, i.e., the probability of a possible world is computed as the product of the probabilities of the individual variable assignments based on which the world was created. Our approximation scheme works for both the incomplete and probabilistic version of C-tables.

Theorem 3 ($\mathsf{trans}^{UADB}_{\text{C-TABLE}}$ is a certain lower-bound)**.** *Given an incomplete database $\mathcal{D}$ encoded as C-tables, $\mathsf{trans}^{UADB}_{\text{C-TABLE}}(\mathcal{D})$ is a certain lower-bound.*

Note that $\mathcal{D}^{\downarrow}$ is not guaranteed to be exact certain. For instance, a tuple t consisting only of constants and for which $\phi_{\mathcal{D}}(t)$ is a tautology is guaranteed to be certain, but $\mathcal{D}^{\downarrow}(t) = F$ if $\phi_{\mathcal{D}}(t)$ is not in CNF.

Example 11. *Consider a C-table consisting of two tuples $t_1 = (1, X)$ with $\phi_{\mathcal{D}}(t_1) \coloneqq (X = 1)$ and $t_2 = (1, 1)$ with $\phi_{\mathcal{D}}(t_2) \coloneqq (X \neq 1)$. $\mathsf{trans}^{UADB}_{\text{C-TABLE}}$ would mark $(1, 1)$ as uncertain, because even though this tuple exists in the C-table and it's local condition is in CNF, the local condition is not a tautology. However, tuple $(1, 1)$ is certain since either $X = 1$ and then first tuple evaluates to $(1, 1)$ or $X \neq 1$ and the second tuple is included in the possible world.*

x-DBs: An x-DB [17] is a set of x-relations, which are sets of x-tuples. An x-tuple τ is a set of tuples $\{t_1, \ldots, t_n\}$ with a label indicating whether the x-tuple is optional. Each x-tuple is assumed to be independent of the others, and its alternatives are assumed to be disjoint. Thus, a possible world of an x-relation $\leftrightarrow$ is constructed by selecting at most one alternative $t \in \tau$ for every x-tuple τ from $\leftrightarrow$ if τ is optional, or exactly one if it is not optional. The probabilistic version of x-DBs (also called a Block-Independent or BI-DB) as introduced in [17] assigns each alternative a probability and we require that $P(\tau) = \sum_{t \in \tau} P(t) \leq 1$. Thus, a tuple is optional if $P(\tau) < 1$ and there is no need to use labels to mark optional tuples. We use $|\tau|$ to denote the number of alternatives of x-tuple τ. We define a approximation scheme $\mathsf{trans}^{UADB}_{\text{x-DB}}$ for x-relations where tuple t's annotation is T iff t is the single, non-optional alternative of an x-tuple. In probabilistic x-DBs we check $P(\tau) = 1$.

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

Theorem 4 ($\mathsf{trans}^{UADB}_{\text{x-DB}}$ is c-correct)**.** *Given a database $\mathcal{D}$, $\mathsf{trans}^{UADB}_{\text{x-DB}}(\mathcal{D})$ is an exact certain approximation.*

4.5.1 Extracting selected-guess worlds. Computing some possible world is trivial for most incomplete and probabilistic data models. However, for the case of probabilistic data models we are particularly interested in the highest-probability world (the selected guess world). We now discuss in more detail how we choose the SGW D_{sg} for the data models for which we have introduced approximation schemes above.

TI-DB: For a TI-DB $\mathcal{D}$, the selected guess world consists of all tuples t such that $P(t) \geq 0.5$. To understand why this is the case recall that the probability of a world from a TI-DB is the product of the probabilities of included tuples with one minus the probability of excluded tuples. This probability is maximized by including only tuples where $P(t) \geq 0.5$. For the incomplete version of TI-DBs we have to include all non-optional tuples and can choose arbitrarily which optional tuples to include in D_{sg}.

PC-tables: For a PC-table, computing the most likely possible world reduces to answering a query over the database, which is known to be `#P` in general [10]. Specific tables (e.g., those generated by "safe" queries [10]) admit `PTIME` solutions. Alternatively, there exist a wide range of algorithms [14, 58, 62, 95] that can be used to compute an arbitrarily close approximation of the most likely world.

Disjoint-independent databases: Since the x-tuples in an x-DB are independent of each other, the probability of a possible world from an x-DB $\mathcal{D}$ is maximized by including for every x-tuple τ its alternative with the highest probability $\text{argmax}_{t \in \tau} P(t)$ or no alternative if $\max_{t \in \tau} P(t) < (1 - P(\tau))$, i.e., if the probability of not including any alternative for the x-tuple is higher than the highest probability of an alternative

for the x-tuple.

4.6 Querying over bounds

We now study whether queries over bounds produced by *approximation schemes* preserve certain lower-bound. Specifically, we demonstrate that standard $\mathcal{K}$-relational query evaluation preserves *bounding effect* for any certain lower-bound approximation scheme. Recall that a query semantics preserves bounds if a query $Q(\mathcal{D}^{\downarrow})$ evaluated on a certain approximation $\mathcal{D}^{\downarrow}$ of incomplete database $\mathcal{D}$ is a certain lower-bound for $Q(\mathcal{D})$. Our result generalizes a previous result of Reiter [36] to any type of incomplete $\mathcal{K}$-database for which we can define an efficient certain under approximation scheme. We need the following lemma, to show that the natural order of a semiring factors through addition and multiplication. This is a known result that we only state for completeness.

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

$$k_1 \preceq_{\mathcal{K}} k_3 \wedge k_2 \preceq_{\mathcal{K}} k_4 \Rightarrow k_1 +_{\mathcal{K}} k_2 \preceq_{\mathcal{K}} k_3 +_{\mathcal{K}} k_4$$

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

Proof: See Appendix A.1

4.7 Preservation of Certain Lower-bound

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

for $\text{CERT}_{\mathcal{K}}$ of the result of the query. Combining arguments, we get preservation of certain lower-bound.

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

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

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

In a nutshell, if we are given a lower-bound estimation, then evaluating any $\mathcal{RA}^+$-query over the bound using $\mathcal{K}$-relational query semantics preserves bounding effect if we can prove that $\text{CERT}_{\mathcal{K}}$ is superadditive and supermultiplicative.

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

Proof: See Appendix A.1

Using the superadditivity and -multiplicativity of $\text{CERT}_{\mathcal{K}}$, we now prove preservation of certain lower-bound. We first prove a restricted version of this result.

Lemma 4. *Let $\mathcal{D}$ be a $\mathcal{K}_W$-database and $\mathcal{D}^{\downarrow}$ be a certain answer $\mathcal{K}$-approximation for $\mathcal{D}$. $\mathcal{RA}^+$ queries over $\mathcal{D}^{\downarrow}$ preserve certain lower bound.*

The major drawback of Lemma 4 is that it is limited to exact certain input apprixmations. Next, we show that certain lower bound is still preserved even if the input approximation is only an under approximation.

Theorem 5. *Let $\mathcal{D}$ be a $\mathcal{K}_W$-database and $\mathcal{D}^{\downarrow}$ an exact certain apprximation for $\mathcal{D}$. $\mathcal{RA}^+$ queries over $\mathcal{D}^{\downarrow}$ preserve certain lower-bound.*

Proof: See Appendix A.1

In Appendix 4.8 we demonstrate that under certain circumstances, queries also preserve exact certain.

4.8 Preservation of Certain Upper-bound

TI-DBs: We now demonstrate that positive queries preserve certain upper-bound if the input is a certain approximation produced by the selected guess approximation scheme $\mathsf{trans}^{UADB}_{\text{TI-DB}}$ (Sec. 4.5). To show this, we observe that if there exists a possible world for which two $\mathcal{K}_W$-elements $\vec{k_1}$ and $\vec{k_2}$ are both minimal then $\sqcap_{\mathcal{K}}$ commutes with addition and multiplication, and standard $\mathcal{K}$-relational semantics preserves certain upper bound.

Lemma 5. *Let $\vec{k_1}, \vec{k_2} \in K^W$ for some possible world semiring $\mathcal{K}_W$. If there exists $i \in W$ such that $\sqcap_{\mathcal{K}}(\vec{k_1}) = \vec{k_1}[i]$ and $\sqcap_{\mathcal{K}}(\vec{k_2}) = \vec{k_2}[i]$, then the following holds:*

$$\sqcap_{\mathcal{K}}(\vec{k_1} +_{\mathcal{K}_W} \vec{k_2}) = \sqcap_{\mathcal{K}}(\vec{k_1}) +_{\mathcal{K}} \sqcap_{\mathcal{K}}(\vec{k_2}) = (\vec{k_1} +_{\mathcal{K}_W} \vec{k_2})[i]$$
$$\sqcap_{\mathcal{K}}(\vec{k_1} \cdot_{\mathcal{K}_W} \vec{k_2}) = \sqcap_{\mathcal{K}}(\vec{k_1}) \cdot_{\mathcal{K}} \sqcap_{\mathcal{K}}(\vec{k_2}) = (\vec{k_1} \cdot_{\mathcal{K}_W} \vec{k_2})[i]$$

Proof: Recall that PW_i is a homomorphism (Lemma 1), so $(\vec{k_1} +_{\mathcal{K}_W} \vec{k_2})[i] = \vec{k_1}[i] +_{\mathcal{K}} \vec{k_2}[i]$ and $\sqcap_{\mathcal{K}}(\vec{k_j}) = \vec{k_j}[i]$ for $j \in \{1, 2\}$. Thus, $(\vec{k_1} +_{\mathcal{K}_W} \vec{k_2})[i] = \sqcap_{\mathcal{K}}(\vec{k_1}) +_{\mathcal{K}} \sqcap_{\mathcal{K}}(\vec{k_2})$. Next, $\sqcap_{\mathcal{K}}(\vec{k_1} +_{\mathcal{K}_W} \vec{k_2}) = (\vec{k_1} +_{\mathcal{K}_W} \vec{k_2})[i]$ which holds if for any $j \neq i \in W$ we have $(\vec{k_1} +_{\mathcal{K}_W} \vec{k_2})[i] \preceq_{\mathcal{K}} (\vec{k_1} +_{\mathcal{K}_W} \vec{k_2})[j]$. Since $\vec{k_1}[i] = \sqcap_{\mathcal{K}}(\vec{k_1})$ and $\sqcap_{\mathcal{K}}$ is defined based on the natural order, we know that $\vec{k_1}[i] \preceq_{\mathcal{K}} \vec{k_1}[j]$ and analog for $\vec{k_2}$ we have $\vec{k_2}[i] \preceq_{\mathcal{K}} \vec{k_2}[j]$. Lemma 2 then implies $(\vec{k_1} +_{\mathcal{K}_W} \vec{k_2})[i] \preceq_{\mathcal{K}} (\vec{k_1} +_{\mathcal{K}_W} \vec{k_2})[j]$. The proof for multiplication is analog using Lemma 2 to show that $(\vec{k_1} \cdot_{\mathcal{K}_W} \vec{k_2})[i] \preceq_{\mathcal{K}} (\vec{k_1} \cdot_{\mathcal{K}_W} \vec{k_2})[j]$ for any $j \in W$.

To demonstrate certain-upperbound preservation for TI-DBs we have to demonstrate that the encoding of a TI-DB as a $\mathcal{K}_W$-database fulfills the precondition of Lemma 5.

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

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

Proof: Consider the possible world D defined as follows:

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

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

Lemmas 5 and 6 together imply that our apprximation approach preserves certain upper-bound if the input is a TI-DB.

Corollary 1. *Let $\mathcal{D}^{\downarrow}$ be a approximation for a TI-DB $\mathcal{D}$ computed as $\mathsf{trans}_{TI}^{UADB}(\mathcal{D})$. Then $\mathcal{RA}^{+}$ over $\mathcal{D}^{\downarrow}$ preserves upper-bound.*

x-DBs: In general, $\mathcal{RA}^{+}$ queries over approximations derived from x-DBs using our approximation $\mathsf{trans}_{\text{x-DB}}^{UADB}$ from Section 4.5 do not preserve certain upper-bound. We present a sufficient condition for a query to preserve certain upper-bound over such a apprximation. To this end, we define *x-keys*, constraints that ensure that alternatives within the scope of an x-tuple are not all identical if projected on a set of attributes A. Since our approximation scheme for x-DBs is certain upper-bound, queries preserve such bound unless a result tuple that is certain is derived from multiple *correlated*

uncertain input tuples. Since x-tuples from an x-DB are independent of each other, this can only be the case if a result tuple is derived from alternatives of an x-tuple τ from every possible world (i.e., where τ is not optional). Such a situation can be avoided if it is guaranteed that it is impossible for a result tuple to be derived from all alternatives of an x-tuple.

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

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

An x-key is a set of attributes A such that for any x-tuple τ that is not optional and has more than one alternative, there exists at least two alternatives that differ in A. The following lemma states that a superset of an x-key is also an x-key.

Lemma 7. *Let $A \subseteq B \subseteq \text{SCH}(R)$ where $\text{SCH}(R)$ is the schema of an x-relation $\leftrightarrow$. If A is an x-key for R, then so is B.*

Proof: Whether the first condition or first subcondition of the second condition of Definition 7 hold for an x-tuple is independent of the particular choice of x-key. If the second subcondition is true (which trivially implies that the first subcondition is true), then two alternatives of the x-tuple differ on A which trivially implies that they differ on a superset of A.

We prove that for any x-DB $\mathcal{D}$, if a conjunctive, self-join free query Q (a query using selection, projection, and join that accesses no relation more than once) returns at least one x-key per accessed relation, then the query preserves certain upper-bound.

Theorem 6. *Let $\mathcal{D}^{\downarrow}$ be an approximation for a x-DB $\mathcal{D}$ computed using $\mathsf{trans}_{\text{X-DB}}^{UADB}$. Consider a conjunctive query Q in canonical form $\pi_{\mathcal{A}}(\sigma_{\theta}(R_1 \times \ldots \times R_n))$ with $R_i \neq R_j$ for all $i \neq j \in \{1, \ldots, n\}$. Query Q preserves certain upper-bound if $\mathcal{A}$ contains an x-key for every relation R_i accessed by Q.*

Proof: Let $\mathcal{D} = \{R_1, \ldots, R_n\}$ be an x-database, $\mathcal{D}' = \{R'_1, \ldots, R'_n\}$ its encoding as a $\mathbb{B}_W$-database, $\mathcal{D}^{\downarrow}$ a certain upper-bound approximation for $\mathcal{D}'$ derived using $\mathsf{trans}_{\text{X-DB}}^{UADB}$, and Q be a selfjoin-free query of the form $\pi_{\mathcal{A}}(\sigma_{\theta}(R_1 \times \ldots \times R_n))$ such that $\mathcal{A}$ contains an x-key for every relation R_i for $i \in \{1, \ldots, n\}$. Any selfjoin-free $\mathcal{RA}^+$ query without union can be brought into this form. We have to show that $Q(\mathcal{D}^{\downarrow})$ is a certain upper-bound for $Q(\mathcal{D}')$. We prove this claim by contradiction. For sake of the contradiction assume that $Q(\mathcal{D}^{\downarrow})$ is not exact certain. Then there has to exist a tuple $t \in Q(\mathcal{D}')$ such that $Q(\mathcal{D}^{\downarrow})(t) = F$ and $\text{CERT}_{\mathbb{B}}(Q(\mathcal{D}')(t)) = T$. Recall that $+_{\mathbb{B}} = \vee$ and $\cdot_{\mathbb{B}} = \wedge$. Unfolding definitions of relational algebra operators over $\mathcal{K}$-relations we get:

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

$$Q(\mathcal{D}')(t) = \sum_{u : u[A]=t \wedge \forall i \in \{1,\ldots,n\} : u[R_i]=t_i} R'_1(t_1) \cdot_{\mathbb{B}_W} \ldots \cdot_{\mathbb{B}_W} R'_n(t_n) \cdot_{\mathbb{B}_W} \theta(u)$$

Note that for result tuples u of the crossproduct for which $u \not\models \theta$ we have $\theta(u) = F$ (respective $\theta(u) = \mathbb{0}_{\mathbb{B}_W}$). Thus, any monomial (product) corresponding to such a u will evaluate to F ($\mathbb{0}_{\mathbb{B}_W}$). Thus, we can equivalently write the above expressions as shown below where the j values identify monomials for which $u \models \theta$ WLOG assuming that there are $m \in \mathbb{N}$ such monomials.

$$Q(\mathcal{D}^{\downarrow})(t) = \bigvee_{\forall j \in \{1,\ldots,m\}} \left(\bigwedge_{i=1}^{n} \mathcal{D}^{\downarrow}(t_{j_i}) \right)$$

$$Q(\mathcal{D}')(t) = \sum_{\forall j \in \{1,\ldots,m\}} \prod_{i=1}^{n} R'_i(t_{j_i})$$

We use b_{j_i} to denote $\mathcal{D}^{\downarrow}(t_{j_i})$ and $\vec{k_{j_i}}$ to denote $R'_i(t_{j_i})$. Based on our assumption we know:

$$\bigvee_{\forall j \in \{1,\ldots,m\}} \left(\bigwedge_{i=1}^{n} b_{j_i} \right) = F$$

So this can only be the case if for every $j \in \{1, \ldots, m\}$ there exists $f \in \{1, \ldots, n\}$ such that $b_{j_f} = F$. For any $j \in \{1, \ldots, m\}$ let min_j denote the smallest such f, i.e.,

the first element in the j^{th} conjunct that is false and let t_{min_j} denote the corresponding tuple. Based on the fact that $\mathcal{D}^{\downarrow} = \mathsf{trans}^{UADB}_{\text{X-DB}}(\mathcal{D}')$ and that $\mathsf{trans}^{UADB}_{\text{X-DB}}$ is a certain upper bound, we know that if $\mathcal{D}^{\downarrow}(t_{min_j}) = F$ then t_{min_j} is not certain. We will use this fact to derive a contradiction with the assumption $\text{CERT}_{\mathbb{B}}(Q(\mathcal{D}')(t)) = T$. For that, we partition the set of monomials from $Q(\mathcal{D}')(t)$ into two subsets M_1 and ${M_1}^{\mathcal{C}}$ where M_1 contains the identifiers j of all monomials such that $min_j = 1$ and ${M_1}^{\mathcal{C}}$ contains all remaining monomials. We will show that

$$\text{CERT}_{\mathbb{B}}(\sum_{j \in M_1} \prod_{i=1}^{n} k_{j_i}) = F$$

, then

$$\text{CERT}_{\mathbb{B}}(\sum_{j \in {M_1}^{\mathcal{C}}} \prod_{i=1}^{n} k_{j_i}) = F$$

, and finally

$$\text{CERT}_{\mathbb{B}}(Q(\mathcal{D}')(t)) = \text{CERT}_{\mathbb{B}}(\sum_{j \in M_1} \prod_{i=1}^{n} k_{j_i} +_{\mathbb{B}_W} \sum_{j \in {M_1}^{\mathcal{C}}} \prod_{i=1}^{n} k_{j_i}) = F$$

which is the contradiction we wanted to derive.

First, consider

$$\sum_{j \in M_1} \prod_{i=1}^{n} k_{j_i}$$

Since $\cdot_{\mathbb{B}} = \wedge$ and $\cdot_{\mathbb{B}_W}$ is defined as point-wise application of $\wedge$ to a vector $\vec{k} \in \mathbb{B}_W$ we have $\vec{k} \cdot_{\mathbb{B}_W} \vec{k'} \preceq_{\mathbb{B}_W} \vec{k}$ for any $\vec{k}$ and $\vec{k'}$. Thus,

$$\sum_{j \in M_1} \prod_{i=1}^{n} k_{j_i} \preceq_{\mathbb{B}_W} \sum_{j \in M_1} k_{j_1}$$

. We show $\text{CERT}_{\mathbb{B}}(\sum_{j \in M_1} k_{j_1}) = F$, from which follows

$$\text{CERT}_{\mathbb{B}}(\sum_{j \in M_1} \prod_{i=1}^{n} k_{j_i}) = F$$

.

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

which is not certain. It follows that

$$\textsc{cert}_{\mathbb{B}}(Q(\mathcal{D}')(t)) = \textsc{cert}_{\mathbb{B}}\left(\sum_{o=1}^{n}\left(\sum_{j\in M_o}\prod_{i=1}^{n} k_{j_i}\right)\right) = F$$

which contradicts our assumption that $\textsc{cert}_{\mathbb{B}}(Q(\mathcal{D}')(t)) = T$ and thus concludes the proof.

4.9 UA-DB Implementation

We now discuss the implementation of a UA-DB as a query rewriting front-end built on top of a relational DBMS. A $\mathcal{K}_{UA}$-relation with schema $\textsc{Sch}(R)(A_1, \ldots, A_n)$ annotated with a pairs of $\mathcal{K}$-elements $[c, d]$ is encoded by $\mathcal{K}$-relation $\textsc{Sch}(R)'(A_1, \ldots, A_n, C)$ where the annotation of each tuple encodes d and attribute C stores c. We specifically implement UA-DBs for bag semantics, as this is the model used by most DBMSes. In contrast to $\mathbb{N}$-relations where the multiplicity of a tuple is stored as its annotation, relational databases represent a tuple t with multiplicity n as n copies of t. We use C as a boolean marker and mark c copies of t as certain (1) and the remaining $d - c$ copies as uncertain (0) as shown in the example in Sec. 1.2.

Our frontend rewriting engine receives queries of the form $Q(D_{UA})$ over an $\mathbb{N}_{UA}$-annotated database D_{UA} with schema $\{\ \textsc{Sch}(R)_i(A_1, \ldots, A_n)\ \}$. It rewrites these into equivalent queries $[\![Q]\!]_{UA}(D)$ over a classical bag-relational database D with schema $\{\ \textsc{Sch}(R)'_i(A_1, \ldots, A_n, C)\ \}$ where $C \in \{0, 1\}$ denotes the uncertainty label. The rewrite rules implementing $[\![\cdot]\!]_{UA}$ are given in Fig. 4.1. Implementations of the certain lower-bound and SGW-extraction schemes from Sec. 4.5 are used to make our rewriting engine directly compatible with a wide range of incomplete and probabilistic data models; Such inputs are translated inline into encoded $\mathbb{N}_{UA}$-relations. We implement our approach as a middleware over a database system through an extension of SQL. An input query is first parsed, translated into a relational algebra graph, rewritten using $[\![\cdot]\!]_{UA}$, and then converted back to SQL for execution.

$$
\begin{aligned}
[\![R]\!]_{UA} &= \text{A Labeled } \texttt{R} \text{ (see Section 4.5)} \\
[\![\sigma_\theta(Q)]\!]_{UA} &= \texttt{SELECT * FROM } [\![Q]\!]_{UA} \texttt{ WHERE } \theta \\
[\![\pi_{A_1 \ldots A_n}(Q)]\!]_{UA} &= \texttt{SELECT A1, ..., AN, C FROM } [\![Q]\!]_{UA} \\
[\![Q_1 \bowtie_\theta Q_2]\!]_{UA} &= \texttt{SELECT Q1.*, Q2.*, Q1.C*Q2.C AS C} \\
&\quad \texttt{FROM } [\![Q_1]\!]_{UA}\texttt{, } [\![Q_2]\!]_{UA} \texttt{ WHERE } \theta \\
[\![Q_1 \cup Q_2]\!]_{UA} &= [\![Q1]\!]_{UA} \texttt{ UNION ALL } [\![Q_2]\!]_{UA}
\end{aligned}
$$

Figure 4.1. Query rewrite rules

4.10 Relational Algebra Rewriting and Correctness

To prove that the rewriting defined above is correct, we first formally define the function Enc implementing the encoding of a $\mathbb{N}_{UA}$-database as an $\mathbb{N}$-database and restate $[\![]\!]_{UA}$ as relational algebra rewriting rules. Afterwards, we prove that this rewriting correctly encodes $\mathbb{N}_{UA}$ query semantics. In the following, we use $\{t \mapsto k\}$ to denote a singleton relation where tuple t is annotated with k and all other tuples are annotated with 0. Recall that $arity(R)$ denotes the arity (number of attibutes) of a relation.

Definition 8 (Multiset encoding)**.** $\text{Enc}(R)$ *is a function from* $\mathbb{N}_{UA}$*-relations to* $\mathbb{N}$*-relations. Let* R *be a* $\mathbb{N}_{UA}$*-relation with schema* $A_1, \ldots, A_n$*. Let* R' *be an* $\mathbb{N}$*-relation with schema* $A_1, \ldots, A_n, U$ *that is the result of* $\text{Enc}(R)$ *for some* R*.* Enc *and its inverse are defined as:*

$$
\begin{aligned}
\text{Enc}(R) = \bigcup_{t \in \mathbb{D}^{arity(R)}} & \{(t,1) \mapsto h_{cert}(R(t))\} \\
& \cup \{(t,0) \mapsto h_{det}(R(t)) - h_{cert}(R(t))\} \\
\text{Enc}^{-1}(R') = \bigcup_{t \in \mathbb{D}^{arity(R)}} & t \mapsto (R'(t,1), R'(t,0) + R'(t,1))
\end{aligned}
$$

We define Enc over databases as applying Enc to every relation in the data-

$$
\begin{aligned}
[\![R]\!]_{UA} &= R \\
[\![\sigma_\theta(Q)]\!]_{UA} &= \sigma_\theta([\![Q]\!]_{UA}) \\
[\![\pi_A(Q)]\!]_{UA} &= \pi_{A,C}([\![Q]\!]_{UA}) \\
[\![Q_1 \sqcup {}_\theta Q_2]\!]_{UA} &= \pi_{\mathrm{Sch}(Q_1 \sqcup Q_2), min(Q_1.C, Q_2.C) \to C}([\![Q_1]\!]_{UA} \sqcup {}_\theta [\![Q_2]\!]_{UA}) \\
[\![Q_1 \cup Q_2]\!]_{UA} &= [\![Q_1]\!]_{UA} \cup [\![Q_2]\!]_{UA}
\end{aligned}
$$

Figure 4.2. Relational algebra rewrite rules implementing $[\![Q]\!]_{UA}$

base. Note that even though we define the encoding for bag semantics here, it can be generalized to any $\mathcal{K}_{UA}$ where semiring $\mathcal{K}$ has a monus [91] by replacing $-$ with $-_{\mathcal{K}}$ (the monus operation). Next, we define the relational algebra version of our rewriting $[\![\cdot]\!]_{UA}$ that translates an input query into a query over the encoding produced by Enc. Again, the rewriting is defined through a set of rules (one per relational algebra operator). The rules are shown in Figure 4.2. Here $\mathrm{Sch}(Q)$ denotes the schema of the result of query Q and $e \to a$ used in generalized projection expressions denotes projecting on the result of evaluating expression e and calling the resulting attribute a.

Theorem 7. *Let D_{UA} be a $\mathbb{N}_{UA}$-database and Q an $\mathcal{RA}^+$ query. The following holds:*

$$Q(D_{UA}) = \textsc{Enc}^{-1}([\![Q]\!]_{UA}(\textsc{Enc}(D_{UA})))$$

Proof: Straightforward induction over the structure of queries. Base case: $Q = R$: WLOG consider a tuple t and let $R(t) = [c, d]$. We know that

$$\textsc{Enc}(R)(t, 0) = h_{det}(R(t)) - h_{cert}(R(t)) = d - c$$

and

$$\textsc{Enc}(R)(t, 1) = h_{cert}(R(t)) = c$$

Let $R'' = \textsc{Enc}^{-1}([\![Q]\!]_{UA}(\textsc{Enc}(R))) = \textsc{Enc}^{-1}(\textsc{Enc}(R))$. Then $R''(t) = [R(t,1), R(t,0)+ R(t,1)] = [c, d-c+c] = [c,d]$.

Induction Step: Assume that the claim holds for queries Q_1 and Q_2, we have to show that it also holds for applying an operator of $\mathcal{RA}^+$ to the result of these queries. We use $Q_{[\![]\!]_{UA}i} = [\![Q_i]\!]_{UA}$ and $\leftrightarrow_i = Q_i(D_{UA})$.

Selection $\sigma_\theta(Q_1)$: Note that

$$[\![\sigma_\theta(Q_1)]\!]_{UA} = \sigma_\theta([\![Q_1]\!]_{UA})$$

Consider a tuple t with $R_1(t) = [c,d]$. Let $R_1{}'' = \textsc{Enc}^{-1}(\sigma_\theta(\textsc{Enc}(R_1))$. We have $\sigma_\theta(R_1)(t) = R_1(t) \cdot_{\mathbb{N}_{UA}} \theta(t)$ and

$$\sigma_\theta(\textsc{Enc}(R_1)(t,0)) = (d-c)\cdot\theta(t,0)$$

$$\sigma_\theta(\textsc{Enc}(R_1)(t,1)) = c\cdot\theta(t,1)$$

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

$$\theta(t,0) = \theta(t,1) = 1 \Leftrightarrow \theta(t) = [1,0]$$

Applying the definition of $\textsc{Enc}^{-1}$, we get

$$R_1{}''(t) = [c\cdot\theta(t,1), (d-c)\cdot\theta(t,0) + c\cdot\theta(t,1)]$$

We now distinguish two cases: either $t \models \theta$ and $t \not\models \theta$. First consider the case where $t \models \theta$. Then, $\theta(t,0) = \theta(t,1) = 1$ and we get

$$R_1{}''(t) = [c\cdot 1, (d-c+c)\cdot 1] = [c,d] = R_1(t) = R_1(t) \cdot_{\mathbb{N}_{UA}} \theta(t)$$

Now consider the case $t \not\models \theta$. Then, $\theta(t,0) = \theta(t,1) = 0$ and we get

$$R_1{}''(t) = [c \cdot 0, (d - c + c) \cdot 0] = [0, 0] = R_1(t) \cdot_{\mathbb{N}_{UA}} \theta(t)$$

Natural Join $Q_1 \sqcup_\theta Q_2$: Let

$$\leftrightarrow'' = \textsc{Enc}^{-1}([\![Q_1 \sqcup_\theta Q_2]\!]_{UA}(\textsc{Enc}(R_1), \textsc{Enc}(R_2))$$

and consider a tuple t with $t \models \theta$ and let $t_1 = t[R_1]$, $t_2 = t[R_2]$, $R_i(t_i) = [c_i, d_i]$ for $i \in \{1, 2\}$, and $Q_{res} = [\![Q_1 \sqcup_\theta Q_2]\!]_{UA}$.

$$\begin{aligned} Q_{res} =& \pi_{\text{Sch}(Q_1 \sqcup_\theta Q_2), min(Q_1.C, Q_2.C) \to C}(Q_{join}) \\ Q_{join} =& [\![Q_1]\!]_{UA} \sqcup_\theta [\![Q_2]\!]_{UA} \end{aligned}$$

Based on the induction assumption we have
$[\![Q_i]\!]_{UA}(t_i, 0) = d_i - c_i$ and $[\![Q_i]\!]_{UA}(t_i, 1) = c_i$. Mapping Enc creates two versions of t_i, thus, there are 4 ways of joining these versions:

$$\begin{aligned} Q_{join}(t, 0, 0) &= [\![Q_1]\!]_{UA}(t_1, 0) \cdot [\![Q_2]\!]_{UA}(t_2, 0) \\ &= (d_1 - c_1) \cdot (d_2 - c_2) \\ Q_{join}(t, 0, 1) &= [\![Q_1]\!]_{UA}(t_1, 0) \cdot [\![Q_2]\!]_{UA}(t_2, 1) \\ &= (d_1 - c_1) \cdot c_2 \\ Q_{join}(t, 1, 0) &= [\![Q_1]\!]_{UA}(t_1, 1) \cdot [\![Q_2]\!]_{UA}(t_2, 0) \\ &= c_1 \cdot (d_2 - c_2) \\ Q_{join}(t, 1, 1) &= [\![Q_1]\!]_{UA}(t_1, 1) \cdot [\![Q_2]\!]_{UA}(t_2, 1) \\ &= c_1 \cdot c_2 \end{aligned}$$

The projection expression $min(Q_1.C, Q_2.C)$ maps the first three cases to $(t, 0)$ and the last case to $(t, 1)$. Thus,

$$\begin{aligned} Q_{res}(t,0) &= (d_1 - c_1) \cdot (d_2 - c_2) + (d_1 - c_1) \cdot c_2 + c_1 \cdot (d_2 - c_2) \\ &= d_1 \cdot d_2 - c_1 \cdot c_2 \\ Q_{res}(t,1) &= c_1 \cdot c_2 \end{aligned}$$

Finally, we get

$$R''(t) = [c_1 \cdot c_2, d_1 \cdot d_2 - c_1 \cdot c_2 + c_1 \cdot c_2] = [Q_1 \ \sqcup_{\theta} Q_2](t)$$

Projection $\pi_A(Q_1)$: $[\![\pi_A(Q_1)]\!]_{UA} = \pi_{A,C}([\![Q_1]\!]_{UA})$. Recall the definition of projection: $[\pi_A(R_1)](t) = \sum_{s[A]=t} R_1(s)$. Consider a tuple t and let $\{s_1, \ldots, s_n\}$ be the set of tuples with $s_i[A] = t$. Furthermore, let $s_i = [c_i, d_i]$ and

$$R_1{}'' = \textsc{Enc}^{-1}(\pi_{A,C}(\textsc{Enc}(R_1))$$

Then, $R_1{}''(t) = [k_{t,1}, k_{t,0} + k_{t,1}]$ for

$$\begin{aligned} k_{t,0} &= \sum_{(s,0)[A,U]=(t,0)} \textsc{Enc}(R_1)(s,0) = \sum_{i=1}^{n} d_i - \sum_{i=1}^{n} c_i \\ k_{t,1} &= \sum_{(s,1)[A,U]=(t,1)} \textsc{Enc}(R_1)(s,1) = \sum_{i=1}^{n} c_i \end{aligned}$$

Thus, we get

$$\begin{aligned} R_1{}''(t) &= [k_{t,1}, k_{t,0} + k_{t,1}] \\ &= \left[\sum_{i=1}^{n} d_i - \sum_{i=1}^{n} c_i + \sum_{i=1}^{n} c_i, \sum_{i=1}^{n} c_i \right] \\ &= \sum_{s[A]=t} R_1(s) \\ &= \pi_A(R_1)(t) \end{aligned}$$

Union $Q_1 \cup Q_2$: $[\![Q_1 \cup Q_2]\!]_{UA} = ([\![Q_1]\!]_{UA} \cup [\![Q_2]\!]_{UA})$. Consider a tuple t with $R_i(t) = [c_i, d_i]$. Let

$$R'' = \textsc{Enc}^{-1}(\textsc{Enc}(R_1) \cup \textsc{Enc}(R_2))$$

.

We have $[R_1 \cup R_2](t) = [c_1 + c_2, d_1 + d_2]$ and

$$[\textsc{Enc}(R_1) \cup \textsc{Enc}(R_2)](t, 0)) = (d_1 - c_1) + (d_2 - c_2)$$
$$[\textsc{Enc}(R_1) \cup \textsc{Enc}(R_2)](t, 1)) = c_1 + c_2$$

Based on this we get

$$\begin{aligned} R_1''(t) &= [c_1 + c_2, (d_1 - c_1) + (d_2 - c_2) + (c_1 + c_2)] \\ &= [c_1 + c_2, d_1 + d_2] \\ &= [c_1, d_1] +_{\mathbb{N}_{UA}} [c_2, d_2] = [R_1 \cup R_2](t) \end{aligned}$$

4.10.1 SQL Implementations of Approximation Schemes. We now show SQL implementations of our methods for extracting selected guess worlds and certain lower-bounds from Sec. 4.5.

TI-DBs: Consider a TI-DB relation $R(A_1, \ldots, A_n)$ which is stored as a relation $R'(A_1, \ldots, A_n, P)$ where attribute P stores the probabilities of tuples. Recall that we include all tuples t where $P(t) \geq 0.5$ in the selected guess world and certain lower-bound for TI-DBs annotates tuples t with T (certain) if $P(t) = 1$. In SQL this is expressed as

```
SELECT A1, ... An,
       CASE WHEN P = 1
            THEN 1
            ELSE 0
```

```
        END AS C
FROM R
WHERE P >= 0.5
```

We expect the user to specify the name of the attribute storing the probability for any relation that is marked to be a TI-DB relation. The example shown below illustrates how to mark a relation R which stores probabilities in attribute p as a TI-DB relation.

```
SELECT * FROM R IS TI WITH PROBABILITY (p)
```

x-DBs: For an x-relation $R(A_1, \ldots, A_n)$ which is stored as a relation $R'(X_{id}, Alt_{id}, A_1, \ldots, A_n, P)$ where X_{id} stores identifiers for x-tuples and Alt_{id} store an identifier for alternatives that is unqiue within the scope of an x-tuple. For each x-tuple τ we pick the alternative with the highest probability if the total probability mass of the x-tuple is larger or equal to 0.5. We only mark alternatives of x-tuples as certain if $P(\tau) = 1$ and $|\tau| = 1$. In the SQL implementation we make extensive use of analytical functions (SQL's `OVER`-clause).

```
SELECT A1, ..., An
        CASE WHEN P = 1
             THEN 1
             ELSE 0
        END AS C
FROM R
WHERE Aid = FIRST_VALUE(Aid) OVER w1
      AND 1 - (sum(P) OVER w2)
          >= max(P) OVER w2
WINDOW w1 AS (PARTITION BY Xid ORDER BY P DESC),
       w2 AS (PARTITION BY Xid)
```

When an input relation is identified as an x-relation, we require that the user specifies which attributes stores x-tuple identifies, alternative identifiers, and probabilities. For example, consider the SQL snipplet shown below.

```
SELECT *
FROM R IS X WITH XID (tid)
                 ALTID (aid)
                 PROBABILITY (p)
```

C-tables: For a C-table $R(A_1, \ldots, A_n)$ which is stored as a relation $R'(A_1, \ldots, A_n, V_1, \ldots, V_n, LC)$ where LC stores the local condition $\phi_{\mathcal{D}}(t)$ (as a string) and V_i stores a variable name if $A_i = v$ for some variable v and `NULL` otherwise. The SQL implementation of the certain lower-bound and selected guess world computation for C-tables assumes the existence of a UDF `isTautology` that implements the tautology check as described in Sec. 4.5.

```
SELECT A1, ..., An
       CASE WHEN isTautology(LC)
            THEN 1
            ELSE 0
       END AS C
FROM R
WHERE V1 IS NULL AND ... AND Vn IS NULL
```

To mark an input as a C-table the user has to specify which attributes store the V_i's and local condition.

```
SELECT *
FROM R IS CTABLE WITH VARIABLES (V1, ..., Vn)
                       LOCAL CONDITION (lc)
```

CHAPTER 5

AU-DBs SEMANTICS

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 Table 1.2 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. 5.1 and 5.2, 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. 5.2.3 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. 5.5.3 that mirrors SGW query evaluation, and show that it behaves as expected.

5.1 Scalar Expressions

Recall that $\mathbb{D}$ denotes a domain of values. For this section we assume that $\mathbb{D}$ that consists of $\mathbb{R}$ and the boolean values ($\bot$ and $\top$). Furthermore, let $\mathbb{V}$ denote a countable set of variables.

Definition 9 (Expression Syntax)**.** *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 ...*

$$e_1 \wedge e_2 \qquad e_1 \vee e_2 \qquad \neg e_1 \qquad e_1 = e_2 \qquad e_1 \neq e_2 \qquad e_1 \leq e_2$$

$$e_1 + e_2 \qquad e_1 \cdot e_2 \qquad \frac{1}{e_1} \qquad \textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3$$

are also expressions. Given an expression e, we denote the variables in 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., $e_1 > e_2 = \neg\,(e_1 \leq e_2)$. Assuming that $\mathbb{D}$ contains negative numbers, subtraction can be expressed using addition and multiplication. 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 from $\mathbb{D}$. The semantics of expression evaluation is defined below.

Definition 10 (Expression Semantics)**.** *Let e be an expression. Given a valuation $\varphi : \textsc{vars}(e) \to \mathbb{D}$, the result of evaluating expression e over φ is denoted as $[\![e]\!]_\varphi$. Note that $[\![\frac{1}{e}]\!]_\varphi$ is undefined if $[\![e]\!]_\varphi = 0$. The semantics of expression is defined as shown below:*

$$[\![x]\!]_\varphi \coloneqq \varphi(x) \qquad [\![c]\!]_\varphi \coloneqq c \qquad [\![\neg e_1]\!]_\varphi \coloneqq \neg [\![e_1]\!]_\varphi$$

$$[\![e_1 \wedge e_2]\!]_\varphi \coloneqq [\![e_1]\!]_\varphi \wedge [\![e_2]\!]_\varphi \qquad [\![e_1 \vee e_2]\!]_\varphi \coloneqq [\![e_1]\!]_\varphi \vee [\![e_2]\!]_\varphi$$

$$[\![e_1 + e_2]\!]_\varphi \coloneqq [\![e_1]\!]_\varphi + [\![e_2]\!]_\varphi \qquad [\![e_1 \cdot e_2]\!]_\varphi \coloneqq [\![e_1]\!]_\varphi \cdot [\![e_2]\!]_\varphi$$

$$[\![\frac{1}{e_1}]\!]_\varphi \coloneqq \frac{1}{[\![e_1]\!]_\varphi}$$

$$[\![e_1 = e_2]\!]_\varphi \coloneqq [\![e_1]\!]_\varphi = [\![e_2]\!]_\varphi \qquad [\![e_1 \leq e_2]\!]_\varphi \coloneqq [\![e_1]\!]_\varphi \leq [\![e_2]\!]_\varphi$$

$$[\![\textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3]\!]_\varphi \coloneqq \begin{cases} [\![e_2]\!]_\varphi & \textit{if } [\![e_1]\!]_\varphi \\ [\![e_3]\!]_\varphi & \textit{otherwise} \end{cases}$$

5.1.1 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 $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ is the set of results obtained by evaluating e over each φ_i using the deterministic expression evaluation semantics defined above.

Definition 11 (Incomplete Expression Semantics)**.** *An incomplete valuation Φ is a set $\{\varphi_1, \ldots, \varphi_n\}$ where each φ_i is a valuation. The result of evaluating an expression e over Φ denoted as $[\![e]\!]_\Phi$ is:*

$$[\![e]\!]_\Phi \coloneqq \{[\![e]\!]_\varphi \mid \varphi \in \Phi\}$$

Example 12. *Consider an expression $e \coloneqq x + y$ and an incomplete valuation with possible bindings $\Phi = \{(x = 1, y = 4), (x = 2, y = 4), (x = 1, y = 5)\}$. Applying deterministic evaluation semantics for each of the three valuations from Φ we get $1 + 4 = 5$,$2 + 5 = 6$, and $1 + 5 = 6$. Thus, the possible outcomes of this expression under this valuation are: $[\![e]\!]_\Phi = \{5, 6\}$.*

5.1.2 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 12. *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 $c^{\downarrow} \leq c^{sg} \leq c^{\uparrow}$.

Example 13. *For the boolean domain $\mathbb{D} = \{\bot, \top\}$ with order $\bot < \top$, the corresponding range annotated domain is:*

$$\mathbb{D}_I = \{[\top/\top/\top], [\bot/\top/\top], [\bot/\bot/\top], [\bot/\bot/\bot]\}$$

We use valuations that map the variables of an expression to elements from $\mathbb{D}_I$ to bound incomplete valuations.

Definition 13 (Range-annotated valuation)**.** *Let e be an expression. A* range-annotated valuation $\tilde{\varphi}$ *for e is a mapping* $\text{VARS}(e) \to \mathbb{D}_I$.

Definition 14. *Given an incomplete valuation Φ and a range-annotated valuation $\tilde{\varphi}$ for e, we say that $\tilde{\varphi}$ bounds Φ iff*

$$\forall x \in \text{VARS}(e) : \forall \varphi \in \Phi : \tilde{\varphi}(x)^{\downarrow} \leq \varphi(x) \leq \tilde{\varphi}(x)^{\uparrow}$$

$$\exists \varphi \in \Phi : \forall x \in \text{VARS}(e) : \varphi(x) = \tilde{\varphi}(x)^{sg}$$

Example 14. *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.*

5.1.3 Range-annotated Expression Evaluation. We now define a semantics for evaluating expressions over range-annotated valuations. We then demonstrate that this semantics preserves bounds.

Definition 15. *[Range-annotated expression evaluation] Let* e *be an expression. Given a range valuation* $\tilde{\varphi} : \text{VARS}(e) \to \mathbb{D}_I$, *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:*

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

Note that $[\![\frac{1}{e}]\!]_{\tilde{\varphi}}$ *is undefined if* $[\![e]\!]_{\tilde{\varphi}}{}^{\downarrow} \leq 0$ *and* $[\![e]\!]_{\tilde{\varphi}}{}^{\uparrow} \geq 0$, *because then* $\tilde{\varphi}$ *may bound a valuation* φ *where* $[\![e]\!]_{\varphi} = 0$. *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* $[\![e_3]\!]_{\tilde{\varphi}} = c$. *Then,*

$$[\![e_1 \wedge e_2]\!]_{\tilde{\varphi}}{}^{\downarrow} \coloneqq a^{\downarrow} \wedge b^{\downarrow} \qquad [\![e_1 \wedge e_2]\!]_{\tilde{\varphi}}{}^{\uparrow} \coloneqq a^{\uparrow} \wedge b^{\uparrow}$$

$$[\![e_1 \vee e_2]\!]_{\tilde{\varphi}}{}^{\downarrow} \coloneqq a^{\downarrow} \vee b^{\downarrow} \qquad [\![e_1 \vee e_2]\!]_{\tilde{\varphi}}{}^{\uparrow} \coloneqq a^{\uparrow} \vee b^{\uparrow}$$

$$[\![\neg e_1]\!]_{\tilde{\varphi}}{}^{\downarrow} \coloneqq \neg\, a^{\uparrow} \qquad [\![\neg e_1]\!]_{\tilde{\varphi}}{}^{\uparrow} \coloneqq \neg\, a^{\downarrow}$$

$$[\![e_1 + e_2]\!]_{\tilde{\varphi}}{}^{\downarrow} \coloneqq a^{\downarrow} + b^{\downarrow} \qquad [\![e_1 + e_2]\!]_{\tilde{\varphi}}{}^{\uparrow} \coloneqq a^{\uparrow} + b^{\uparrow}$$

$$[\![e_1 \cdot e_2]\!]_{\tilde{\varphi}}{}^{\downarrow} \coloneqq \min(a^{\uparrow} \cdot b^{\uparrow}, a^{\uparrow} \cdot b^{\downarrow}, a^{\downarrow} \cdot b^{\uparrow}, a^{\downarrow} \cdot b^{\downarrow})$$

$$[\![e_1 \cdot e_2]\!]_{\tilde{\varphi}}{}^{\uparrow} \coloneqq \max(a^{\uparrow} \cdot b^{\uparrow}, a^{\uparrow} \cdot b^{\downarrow}, a^{\downarrow} \cdot b^{\uparrow}, a^{\downarrow} \cdot b^{\downarrow})$$

$$[\![\frac{1}{e_1}]\!]_{\tilde{\varphi}}{}^{\downarrow} \coloneqq \frac{1}{a^{\uparrow}})$$

$$[\![\frac{1}{e_1}]\!]_{\tilde{\varphi}}{}^{\uparrow} \coloneqq \frac{1}{a^{\downarrow}}$$

$$[\![a \leq b]\!]_{\tilde{\varphi}}{}^{\downarrow} \coloneqq a^{\uparrow} \leq b^{\downarrow} \qquad [\![a \leq b]\!]_{\tilde{\varphi}}{}^{\uparrow} \coloneqq a^{\downarrow} \leq b^{\uparrow}$$

$$[\![a = b]\!]_{\tilde{\varphi}}{}^{\downarrow} \coloneqq (a^{\uparrow} = b^{\downarrow} \wedge b^{\uparrow} = a^{\downarrow}) \qquad [\![a = b]\!]_{\tilde{\varphi}}{}^{\uparrow} \coloneqq a^{\downarrow} \leq b^{\uparrow} \wedge b^{\downarrow} \leq a^{\uparrow}$$

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

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

5.1.4 Preservation of Bounds. 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 16. *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 8. *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: We prove this theorem through induction over the structure of an expression under the assumption that $\tilde{\varphi}$ bounds Φ.

Base case: If $e \coloneqq c$ for a constant c, then $e^{\downarrow} = e^{sg} = e^{\uparrow} = c$ which is also the result of e in any possible world of Φ. If $e \coloneqq x$ for a variable x, then since $\tilde{\varphi}$ bounds Φ, the value of x in any possible world is bounded by $\tilde{\varphi}(x)$.

Induction step: Assume that for expressions e_1, e_2, and e_3, we have that their results under Φ are bounded by their result under $\tilde{\varphi}$:

$$\forall i \in \{1,2,3\} : \forall c \in [\![e_i]\!]_\Phi : [\![e_i]\!]_{\tilde{\varphi}}{}^{\downarrow} \leq c \leq [\![e_i]\!]_{\tilde{\varphi}}{}^{\uparrow}$$

$$\exists \varphi \in \Phi : \forall i \in \{1,2,3\} : [\![e_i]\!]_{\tilde{\varphi}}{}^{sg} = [\![e_i]\!]_{\varphi}$$

Note that the second condition trivially holds since $[\![e]\!]_{\tilde{\varphi}}{}^{sg}$ was defined as applying deterministic expression semantics to $\tilde{\varphi}^{sg}$. We, thus, only have to prove that the lower and upper bounds are preserved for all expressions e that combine these expressions using one of the scalar, conditional, or logical operators.

$\underline{e \coloneqq e_1 + e_2}$: Inequalities are preserved under addition. Thus, for any $\varphi \in \Phi$ we have $[\![e_1]\!]_{\tilde{\varphi}}{}^{\downarrow} + [\![e_2]\!]_{\tilde{\varphi}}{}^{\downarrow} \leq [\![e_1]\!]_{\varphi} + [\![e_2]\!]_{\varphi} \leq [\![e_1]\!]_{\tilde{\varphi}}{}^{\uparrow} + [\![e_2]\!]_{\tilde{\varphi}}{}^{\uparrow}$.

$\underline{e \coloneqq e_1 \cdot e_2}$: We distinguish sixteen cases based on which of $[\![e_1]\!]_{\tilde{\varphi}}{}^{\downarrow}$, $[\![e_2]\!]_{\tilde{\varphi}}{}^{\downarrow}$, $[\![e_1]\!]_{\tilde{\varphi}}{}^{\uparrow}$, and $[\![e_2]\!]_{\tilde{\varphi}}{}^{\downarrow}$ are negative. For instance, if all numbers are positive then clearly $[\![e_1]\!]_{\tilde{\varphi}}{}^{\downarrow} \cdot [\![e_2]\!]_{\tilde{\varphi}}{}^{\downarrow} \leq [\![e_1]\!]_{\varphi} \cdot [\![e_2]\!]_{\varphi}$. While there are sixteen cases, there are only four possible combinations of lower and upper bounds we have to consider. Thus, if we take the minimal (maximal) value across all these cases, we get a lower (upper) bound on e.

$\underline{e \coloneqq \frac{1}{e_1}}$: For any pair of numbers c_1 and c_2 that are either both positive or both negative, we have $c_1 \leq c_2$ implies $\frac{1}{c_1} \geq \frac{1}{c_2}$. Thus, $\frac{1}{a^{\uparrow}}$ is an upper bound on $\frac{1}{c}$ for any c bound by a. Analog, $\frac{1}{a^{\downarrow}}$ is an upper bound.

$\underline{e \coloneqq e_1 \wedge e_2}$ and $\underline{e \coloneqq e_1 \vee e_2}$: Both $\vee$ and $\wedge$ are monotone in their arguments wrt. the order $F \preceq_{\mathbb{B}} T$. Thus, applying these operations to combine lower (upper) bounds preserves these bounds.

$\underline{e \coloneqq \neg\, e_1}$: We distinguish three cases: (i) $[\![e_1]\!]_{\varphi} = \bot$ for all $\varphi \in \Phi$; (ii)$[\![e_1]\!]_{\varphi} = \top$ for some $\varphi \in \Phi$ and $[\![e_1]\!]_{\varphi} = \bot$ for some $\varphi \in \Phi$; and (iii) $[\![e_1]\!]_{\varphi} = \bot$ for all $\varphi \in \Phi$. In case (i) for $\tilde{\varphi}$ to bound the input either $[\![e_1]\!]_{\tilde{\varphi}} = [\bot/\bot/\bot]$ in which case $[\![e]\!]_{\tilde{\varphi}} = [\top/\top/\top]$

or $[\![e_1]\!]_{\tilde{\varphi}} = [\bot/\bot/\top]$ and $[\![r]\!]_{\tilde{\varphi}} = [\bot/\top/\top]$. We have $[\![e]\!]_{\varphi} = \top$ for all $\varphi \in \Phi$ and, thus, in either case $[\![e]\!]_{\tilde{\varphi}}$ bounds $[\![e]\!]_{\Phi}$. In case (ii), $[\![e]\!]_{\tilde{\varphi}}{}^{\downarrow} = \bot$ and $[\![e]\!]_{\tilde{\varphi}}{}^{\uparrow} = \top$ which trivially bound $[\![e]\!]_{\Phi}$. The last case is symmetric to (i).

$e \coloneqq e_1 \leq e_2$: Recall that $\bot \leq \top$. $e_1 \leq e_2$ is guaranteed to evaluate to true in every possible world if the upper bound of e_1 is lower than or equal to the lower bound of e_2. In this case it is safe to set $[\![e]\!]_{\tilde{\varphi}}{}^{\downarrow} = \top$. Otherwise, there may exist a possible world where $e_1 \leq e_2$ evaluates to false and we have to set $[\![e]\!]_{\tilde{\varphi}}{}^{\downarrow} = \bot$. Similarly, if the lower bound of e_1 is larger than the upper bound of e_2 then $e_1 \leq e_2$ evaluates to false in every possible world and $[\![e]\!]_{\tilde{\varphi}}{}^{\uparrow} = \bot$ is an upper bound. Otherwise, there may exist a world where $e_1 \leq e_2$ holds and we have to set $[\![e]\!]_{\tilde{\varphi}}{}^{\uparrow} = \top$.

if e_1 **then** e_2 **else** e_3: When e_1 is certainly true ($[\![e_1]\!]_{\tilde{\varphi}}{}^{\downarrow} = [\![e_1]\!]_{\tilde{\varphi}}{}^{\uparrow} = \top$) or certainly false ($[\![e_1]\!]_{\tilde{\varphi}}{}^{\downarrow} = [\![e_1]\!]_{\tilde{\varphi}}{}^{\uparrow} = \bot$) then the bounds e_2 (certainly true) or e_3 (certainly false) are bounds for e. Otherwise, e may evaluate to e_2 in some worlds and to e_3 in others. Taking the minimum (maximum) of the bounds for e_2 and e_3 is guaranteed to bound e from below (above) in any possible world.

We conclude that the result of range-annotated expression evaluation under $\tilde{\varphi}$ which bounds an incomplete valuation Φ bounds the result of incomplete expression evaluation for any expression e.

Conditional range-annotated expressions are bound-preserving for any totally-ordered domain. We can add support for additional scalar operations as long as a bound-preserving evaluation semantics can be defined. Important classes of scalar operations are operations that preserve the order over the domain, i.e., for any domain values $a < b$ and $c < d$ which $a \diamond b$. For instance, this is the case for addition over natural numbers. For any such operation, point-wise application to range-annotated domain values is bound preserving. For categorical values where no sensible order

can be defined, we impose an arbitrary order. Note that in the worst-case, we can just annotate a value with the range covering the whole domain to indicate that it is completely uncertain.

5.2 Attribute-Annotated Uncertain Databases

We define attribute-annotated uncertain databases (AU-DBs) as a special type of $\mathcal{K}$-relations over range-annotated domains and demonstrate how to bound an incomplete $\mathcal{K}$-relation using this model. Afterwards, define a metric for how precise the bounds of an incomplete $\mathcal{K}$-database encoded by a AU-DB are and proceed to define a query semantics for AU-DBs and prove that this query semantics preserves bounds. Tuple annotations of AU-DBs are triples of elements from a semiring $\mathcal{K}$. These triples form a semiring structure $\mathcal{K}_{AU}$. The construction underlying $\mathcal{K}_{AU}$ is well-defined if $\mathcal{K}$ is an l-semiring, i.e., a semiring where the natural order forms a lattice over the elements of the semiring. Importantly, $\mathbb{N}$ (bag semantics), $\mathbb{B}$ (set semantics), and many provenance semirings are l-semirings.

5.2.1 AU-DBs. 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 tuples. Thus, instead of using annotations from $\mathcal{K}^2$, we use $\mathcal{K}^3$ to encode three annotations for each tuple: a lower bound on the certain annotation of the tuple, the annotation of the tuple in the SGW, and an over-approximation of the tuple's possible annotation.

Definition 17 (Tuple-level Annotations)**.** *Let $\mathcal{K}$ be an l-semiring and let $\preceq_{\mathcal{K}}$ denote its natural order. Then the tuple level range-annotated domain K_{AU} is defined as:*

$$\{(k^{\downarrow}, k, k^{\uparrow}) \mid k, k^{\downarrow}, k^{\uparrow} \in \mathcal{K} \wedge k^{\downarrow} \preceq_{\mathcal{K}} k \preceq_{\mathcal{K}} k^{\uparrow}\}$$

We use $\mathcal{K}_{AU}$ to denote semiring $\mathcal{K}^3$ restricted to elements from K_{AU}.

Similar to the range-annotated domain, a value (k_1, k_2, k_3) from $\mathcal{K}_{AU}$ encodes a semiring element from $\mathcal{K}$ and two elements (k_1 and k_3) that bound the element from below and above. Given an $\mathcal{K}_{AU}$-element $k = (k_1, k_2, k_3)$ we define $k^{\downarrow} = k_1$, $k^{sg} = k_2$, and $k^{\uparrow} = k_3$. Note that $\mathcal{K}_{AU}$ is a semiring since when combining two elements of $\mathcal{K}_{AU}$ with $+_{\mathcal{K}^3}$ and $\cdot_{\mathcal{K}^3}$, the result (k_1, k_2, k_3) fulfills the requirement $k_1 \preceq_{\mathcal{K}} k_2 \preceq_{\mathcal{K}} k_3$. This is the case because semiring addition and multiplication preserves the natural order of $\mathcal{K}$ and these operations in $\mathcal{K}^3$ are defined as pointwise application of $+_{\mathcal{K}}$ and $\cdot_{\mathcal{K}}$, e.g., $(k^{\downarrow}, k, k^{\uparrow}) +_{\mathcal{K}^3} (l^{\downarrow}, l, l^{\uparrow}) = (l^{\downarrow} +_{\mathcal{K}} l^{\downarrow}, k +_{\mathcal{K}} l, k^{\uparrow} +_{\mathcal{K}} l^{\uparrow})$ and $k_1 \preceq_{\mathcal{K}} k_2 \wedge k_3 \preceq_{\mathcal{K}} k_4 \Rightarrow k_1 +_{\mathcal{K}} k_3 \preceq_{\mathcal{K}} k_2 \preceq_{\mathcal{K}} k_4$ for any $k_1, k_2, k_3, k_4 \in \mathcal{K}$.

Definition 18 ($\mathcal{K}_{AU}$-relations)**.** *Given a range-annotated data domain $\mathbb{D}_I$ and l-semiring $\mathcal{K}$, an $\mathcal{K}_{AU}$-relation of arity n is a function $R : {\mathbb{D}_I}^n \to \mathcal{K}_{AU}$.*

As a notational convenience we show certain values, i.e., values $c \in \mathbb{D}_I$ where $c^{\downarrow} = c^{sg} = c^{\uparrow} = c'$, as the deterministic value c' they encode.

5.2.2 Extracting Selected-Guess Worlds. Note that the same tuple t may appear more than once in a $\mathcal{K}_{AU}$-relation albeit with different value annotations. We can extract the selected-guess world encoded by a $\mathcal{K}_{AU}$-relation by grouping tuples by the SG of their attribute values and then summing up their tuple-level SG annotation.

Definition 19. *We lift function sg from values to tuples: $sg : {\mathbb{D}_I}^n \to \mathbb{D}^n$, i.e., given an AU-DB tuple $\mathbf{t} = (v_1, \ldots, v_n)$, $\mathbf{t}^{sg} \coloneqq ({v_1}^{sg}, \ldots, {v_n}^{sg})$. For a $\mathcal{K}_{AU}$-relation $\mathbf{R}$, $\mathbf{R}^{sg}$, the SGW encoded by $\mathbf{R}$, is then defined as:*

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

Example 15. *Figure 5.1a 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 bound, selected-guess and upper bound of the annotation value. Since this is a $\mathbb{N}_{AU}$ relation, the annotations encode*

Table 5.1. Example AU-DB relation and the SGW it encodes

(a) Example AU-DB instance

A	**B**	$\underline{\mathbb{N}^3}$
[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)

(b) selected-guess world

A	**B**	$\underline{\mathbb{N}}$
1	1	5
2	3	1

multiplicities of tuples. For example, the first tuple represents a tuple $(1, 1)$ *that appears at least twice in every possible world (its lower bound annotation is* 2*), appears twice in the SGW, and may appear in any possible world at most thrice. Figure 5.1b shows the SGW encoded by the AU-DB produced by summing up the annotations of tuples with identical SG values. For instance, the first two tuples both represent tuple* $(1, 1)$ *and their annotations sum up to* 5*, i.e., the tuple* $(1, 1)$ *appears five times in the chosen SGW.*

5.2.3 Encoding Bounds. We now formally define what it means for an AU-DB to bound a an incomplete $\mathcal{K}$-relation from above and below. For that we first define bounding of deterministic tuples by range-annotated tuples.

Definition 20 (Tuple Bounding)**.** *Let* $\mathbf{t}$ *be a range-annotated tuple with schema* $(a_1, \ldots, a_n)$ *and* t *be a tuple with same schema as* $\mathbf{t}$*. We say that* $\mathbf{t}$ *bounds* t *written as* $t \sqsubseteq \mathbf{t}$ *iff*

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

Obviously, one AU-DB tuple can bound multiple different conventional tuples and vice versa. We introduce *tuple matchings* as a way to match the annotations of tuples of a $\mathcal{K}_{AU}$-database (or relation) with that of one possible world of an incomplete

$\mathcal{K}$-database (or relation). Based on tuple matchings we then define how to bound possible worlds.

Definition 21 (Tuple matching)**.** *Let n-ary AU-relation* $\mathbf{R}$ *and an n-ary database* R*. A tuple matching* $\mathcal{TM}$ *for* $\mathbf{R}$ *and* R *is a function* $(\mathbb{D}_I)^n \times \mathbb{D}^n \to \mathcal{K}$*. s.t.*

$$\forall \mathbf{t} \in {\mathbb{D}_I}^n : \forall t \not\sqsubseteq \mathbf{t} : \mathcal{TM}(\mathbf{t}, t) = \mathbb{0}_{\mathcal{K}}$$

and

$$\forall t \in \mathbb{D}^n : \sum_{\mathbf{t} \in {\mathbb{D}_I}^n} \mathcal{TM}(\mathbf{t}, t) = R(t)$$

Intuitively, a tuple matching distributes the annotation of a tuple from R over one or more matching tuples from $\mathbf{R}$. That is, multiple tuples from a UA-DB may encode the same tuple from an incomplete database. This is possible when the multidimensional rectangles of their attribute-level range annotations overlap. For instance, range-annotated tuples ([1/2/3]) and ([2/3/5]) both match the tuple (2).

Definition 22 (Bounding Possible Worlds)**.** *Given an n-ary AU-DB relation* $\mathbf{R}$ *and a n-ary deterministic relation* R *(a possible world of an incomplete* $\mathcal{K}$*-relation), relation* $\mathbf{R}$ *is a lower bound for* R *iff there exists a tuple matching* $\mathcal{TM}$ *for* $\mathbf{R}$ *and* R *s.t.*

$$\forall \mathbf{t} \in {\mathbb{D}_I}^n : \sum_{t \in \mathbb{D}^n} \mathcal{TM}(\mathbf{t}, t) \succeq_{\mathcal{K}} \mathbf{R}(\mathbf{t})^{\downarrow} \tag{5.1}$$

and is upper bounded by $\mathbf{R}$ *iff there exists a tuple matching* $\mathcal{TM}$ *for* $\mathbf{R}$ *and* R *s.t.*

$$\forall \mathbf{t} \in {\mathbb{D}_I}^n : \sum_{t \in \mathbb{D}^n} \mathcal{TM}(\mathbf{t}, t) \preceq_{\mathcal{K}} \mathbf{R}(\mathbf{t})^{\uparrow} \tag{5.2}$$

A AU-relation $\mathbf{R}$ *bounds a relation* R *written as* $R \sqsubset \mathbf{R}$ *iff there exists a tuple matching* $\mathcal{TM}$ *for* $\mathbf{R}$ *and* R *that fulfills both Equations* (5.1) *and* (5.2)*.*

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

Definition 23 (Bounding Incomplete Relations)**.** *Given an incomplete $\mathcal{K}$-relation $\mathcal{R}$ and a AU-relation $\mathbf{R}$, we say that $\mathbf{R}$ bounds R, written as $\mathcal{R} \sqsubset \mathbf{R}$ iff*

$$\forall R \in \mathcal{R} : R \sqsubset \mathbf{R} \tag{5.3}$$

$$\exists R \in \mathcal{R} : \mathbf{R}^{sg} = R \tag{5.4}$$

Note that all bounds we define for relations are extended to databases in the obvious way.

Example 16. *Consider the AU-DB from Ex. 15 and the two possible world shown below.*

Table 5.2. ex. 16

D_1

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

D_2

	A	**B**	$\underline{\mathbb{N}}$
t_3	1	1	2
t_4	1	3	2
t_5	2	4	1

This AU-DB bounds these worlds, since there exist tuple matchings that provides both a lower and an upper bound for the annotations of the tuples of these worlds. For instance, denoting the tuples from this example as

$$\mathbf{t}_1 \coloneqq ([1/1/1], [1/1/1])$$

$$\mathbf{t}_2 \coloneqq ([1/1/1], [1/1/3])$$

$$\mathbf{t}_3 \coloneqq ([1/2/2], [3/3/3])$$

tuple matchings $\mathcal{TM}_1$ and $\mathcal{TM}_2$ shown below to bound D_1 and D_2.

$$\mathcal{TM}_1(\mathbf{t}_1, t_1) = 2 \qquad \mathcal{TM}_1(\mathbf{t}_2, t_1) = 3 \qquad \mathcal{TM}_1(\mathbf{t}_3, t_1) = 0$$
$$\mathcal{TM}_1(\mathbf{t}_1, t_2) = 0 \qquad \mathcal{TM}_1(\mathbf{t}_2, t_2) = 0 \qquad \mathcal{TM}_1(\mathbf{t}_3, t_2) = 1$$
$$\mathcal{TM}_2(\mathbf{t}_1, t_3) = 2 \qquad \mathcal{TM}_2(\mathbf{t}_2, t_3) = 0 \qquad \mathcal{TM}_2(\mathbf{t}_3, t_3) = 0$$
$$\mathcal{TM}_2(\mathbf{t}_1, t_4) = 0 \qquad \mathcal{TM}_2(\mathbf{t}_2, t_4) = 2 \qquad \mathcal{TM}_2(\mathbf{t}_3, t_4) = 0$$
$$\mathcal{TM}_2(\mathbf{t}_1, t_5) = 0 \qquad \mathcal{TM}_2(\mathbf{t}_2, t_5) = 0 \qquad \mathcal{TM}_2(\mathbf{t}_3, t_5) = 1$$

5.2.4 Tightness of Bounds. Def. 23 defines what it means for an AU-DB to bound an incomplete databases. However, given an incomplete database, there may be many possible AU-DBs that bound it that differ in how tight the bounds are. For instance, both $t_1 \coloneqq ([1/15/100])$ and $t_2 \coloneqq ([13/14/15])$ bound tuple (15), but intuitively the bounds provided by the second tuple are tighter. In this section we develop a metric for the tightness of the approximation provided by an AU-DB and prove that finding a AU-DB that maximizes tightness is intractable. Intuitively, given two AU-DBs $\mathbf{D}$ and $\mathbf{D}'$ that both bound an incomplete $\mathcal{K}$-database D, $\mathbf{D}$ is a tighter bound than $\mathbf{D}'$ if the set of deterministic databases bound by $\mathbf{D}$ is a subset of the set of deterministic databases bound by $\mathbf{D}'$. As a sanity check, consider $\mathbf{D}_1 \coloneqq \{t_1\}$ and $\mathbf{D}_2 \coloneqq \{t_2\}$ using t_1 and t_2 from above and assume that $\mathbb{D} = \mathbb{N} \cup \mathbb{B}$. Then $\mathbf{D}_2$ is a tighter bound than $\mathbf{D}_1$ since the three deterministic databases it bounds $\{(13)\}$, $\{(14)\}$ and $\{(15)\}$ are also bound by $\mathbf{D}_1$, but $\mathbf{D}_1$ bounds additional databases, e.g., $\{(2)\}$ that are not bound by $\mathbf{D}_2$.

Definition 24 (Bound Tightness)**.** *Consider two $\mathcal{K}_{AU}$-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 $\mathcal{K}$-databases D with schema S we have:*

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

We say that $\mathbf{D}$ *is a 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* $\mathcal{K}$*-database* $\mathcal{D}$ *if:*

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

Note that the notion of tightness is well-defined even if the data domain $\mathbb{D}$ is infinite. For instance, if we use the reals $\mathbb{R}$ instead of natural numbers as the domain in the example above, then still $\mathbf{D}_1 \succ_I \mathbf{D}_2$. In general AU-DBs that are tighter bounds are preferable. However, computing a maximally tight bound is intractable.

Theorem 9 (Finding Maximally Tight Bounds)**.** *Let* $\mathcal{D}$ *be an incomplete* $\mathbb{N}$*-database encoded as a C-table [12]. Computing a maximally tight bound* $\mathbf{D}$ *for* $\mathcal{D}$ *is NP-hard.*

Proof: Note that obviously, C-tables which apply set semantics cannot encode every possible incomplete $\mathbb{N}$-database. However, the class of all $\mathbb{N}$-databases where no tuples appear more than once can be encoded using C-tables. To prove the hardness of computing maximally tight bounds it suffices to prove the hardness of finding bounds for this subset of all $\mathbb{N}$-databases. We prove the claim through a reduction from the NP-complete 3-colorability decision problem. A graph $G = (V, E)$ is 3-colorable if each node n can be assigned a color $C(n) \in \{r, g, b\}$ (red, green, and blue) such that for every edge $e = (v_1, v_2)$ we have $C(v_1) \neq C(v_2)$. Given such a graph, we will construct a C-table $\mathcal{R}$ encoding an incomplete $\mathbb{B}$-relation (C-tables use set semantics) with a single tuple and show that the tight upper bound on the annotation of the tuple is $\top$ iff the graph G is 3-colorable. We now briefly review C-tables for readers not familiar with this model. Consider a set of variables Σ. A C-table [12] $\mathcal{R} = (R, \phi, \Phi)$ is a relation R paired with (i) a global condition Φ which is also a logical condition over Σ and (ii) a function ϕ that assigns to each tuple $t \in R$ a logical condition over Σ. Given a valuation μ that assigns to each variable from

Σ a value, the global condition and all local conditions evaluate to either $\top$ or $\bot$. The incomplete database represented by a C-table $\mathcal{R}$ is the set of all relations R such that there exists a valuation μ for which $\mu(\Phi)$ is true and $R = \{t \mid \mu(\phi(t))\}$, i.e., R contains all tuples for which the local condition evaluates to true. Given an input graph G, we associate a variable x_v with each vertex $v \in V$. Each possible world of the C-table we construct encodes one possible assignment of colors to the nodes of the graph. This will be ensured through the global condition which is a conjunction of conditions of the form $(x_v = r \vee x_v = g \vee x_v = b)$ for each node $v \in V$. The C-table contains a single tuple $t_{one} = (1)$ whose local condition tests whether the assignment of nodes to colors is a valid 3-coloring of the input graph. That is, the local condition is a conjunction of conditions of the form $x_{v_1} \neq x_{v_2}$ for every edge $e = (v_1, v_2)$. Thus, the C-table (R, ϕ, Φ) we construct for G is:

$$R = \{t_{one}\} \text{ for } t_{one} = (1)$$

$$\Phi = \bigwedge_{v \in V} (x_v = r \vee x_v = g \vee x_v = b)$$

$$\phi(t_{one}) = \bigwedge_{(v_1, v_2) \in E} x_{v_1} \neq x_{v_2}$$

Note that in any possible world R' represented by $\mathcal{R}$, each x_v is assigned one of the valid colors, because otherwise the global condition would not hold. For each such coloring, the tuple $t_{one} = (1)$ exists $R'(t_{one}) = \top$ if no adjacent vertices have the same color, i.e., the graph is 3-colorable. Thus, if G is not 3-colorable, then $R'(t_{one}) = \bot$ in every possible world and if G is 3-colorable, then $R'(t_{one}) = \top$ in at least one possible world. Thus, the tight upper bound on t_{one}'s annotation is $\top$ iff G is 3-colorable.

In the light of this result, any efficient methods for translating incomplete and probabilistic databases into AU-DBs can not guarantee tight bounds. Nonetheless, comparing the tightness of AU-DBs is useful for evaluating how tight bounds are in practice as we will do in Chapter 6. Furthermore, note that even if we were able to

compute tight bounds for an input incomplete database, preserving the bounds under queries is computationally hard. This follows from hardness results for computing tight bounds for the results of an aggregation query over incomplete databases (e.g., see [18]).

5.3 AU-DB Query Semantics

In this section we first 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 $\mathcal{K}$-database $\mathcal{D}$, then the output bounds $Q(\mathcal{D})$. Conveniently, it turns out that the standard query semantics for $\mathcal{K}$-relations with a slight extension to deal with uncertain boolean values in conditions is sufficient for this purpose. Recall from Sec. 5.1 that conditions (or more generally scalar expressions) over range-annotated values evaluate to triples of boolean values, e.g., $[F/F/T]$ would mean that the condition is false in some worlds, is false in the SGW, and may be true in some worlds. Recall the standard semantics for evaluating selection conditions over $\mathcal{K}$-relations. For a selection $\sigma_\theta(R)$ the annotation of a tuple t in annotation of t in the result of the selection is computed by multiplying $R(t)$ with $\theta(t)$ which is defined as a function $\mathbb{B} \to \{\mathbb{0}_\mathcal{K}, \mathbb{1}_\mathcal{K}\}$ that returns $\mathbb{1}_\mathcal{K}$ if θ evaluates to true on t and $\mathbb{0}_\mathcal{K}$ otherwise. In $\mathcal{K}_{AU}$-relations tuple t is a tuple of range-annotated values and, thus, θ evaluates to an range-annotated Boolean value as described above. Using the range-annotated semantics for expressions from Sec. 5.1, a selection condition evaluates to a triple of boolean values $\mathbb{B}^3$. We need to map such a triple to a corresponding $\mathcal{K}_{AU}$-element to define a semantics for selection that is compatible with $\mathcal{K}$-relational query semantics.

Definition 25 (Boolean to Semiring Mapping)**.** *Let $\mathcal{K}$ be a semiring. We define*

function $\mathcal{M}_{\mathcal{K}} : \mathbb{B}^3 \to \mathcal{K}^3$ *as:*

$$\mathcal{M}_{\mathcal{K}}(b_1, b_2, b_3) \coloneqq (k_1, k_2, k_3) \text{ where}$$

$$\forall i \in \{1,2,3\} : k_i \coloneqq \begin{cases} \mathbb{1}_{\mathcal{K}} & \text{if } b_i = true \\ \mathbb{0}_{\mathcal{K}} & \text{otherwise} \end{cases}$$

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

Definition 26 (Conditions over Range-annotated Tuples)**.** *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* $\theta(\mathbf{t})$*, the result of the condition* θ *applied to* $\mathbf{t}$ *as:*

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

Example 17. *Consider the example* $\mathbb{N}_{AU}$*-relation* R *shown below. The single tuple* $\mathbf{t}$ *of this relation exists at least once in every possible world, twice in the SGW, and no possible world contains more than* 3 *tuples bound by this tuple.*

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

To evaluate query $Q \coloneqq \sigma_{A=2}(R)$ *over this relations, 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 possible world of the query result, appears twice in the SGW query result, and occurs at most three times in any possible world.

5.3.1 Preservation of Bounds. For this query semantics to be useful, we need to prove that it preserves bounds. Intuitively, this is true because expressions are evaluated using our range-annotated expression semantics which preserves bounds on values and queries are evaluated in a direct-product semiring $\mathbb{N}_{AU}$ for which semiring operations are defined point-wise. Furthermore, we utilize a result we have proven in [93, Lemma 2]: the operations of l-semirings preserve the natural order, e.g., if $k_1 \preceq_{\mathcal{K}} k_2$ and $k_3 \preceq_{\mathcal{K}} k_4$ then $k_1 +_{\mathcal{K}} k_3 \preceq_{\mathcal{K}} k_2 +_{\mathcal{K}} k_4$.

Theorem 10 ($\mathcal{RA}^+$Queries Preserve Bounds)**.** *Let $\mathcal{D}$ be an incomplete $\mathbb{N}$-database, Q be a $\mathcal{RA}^+$ query, and $\mathbf{D}$ be an $\mathbb{N}_{AU}$-database that bounds D. Then $Q(\mathbf{D})$ bounds $Q(\mathcal{D})$.*

Proof: We prove this lemma using induction over the structure of a relational algebra expression under the assumption that $\mathbf{D}$ bounds the input $\mathcal{D}$.

Base case: The query Q consists of a single relation access R. The result is bounded following from $\mathbf{D} \sqsubset \mathcal{D}$.

Induction step: Let $\mathbf{R}$ and $\mathbf{S}$ bound n-ary relation R and m-ary relation S. Consider $D \in \mathcal{D}$ and let $\mathcal{TM}_R$ and $\mathcal{TM}_S$ be two tuple matchings based on which these bounds can be established for D. We will demonstrate how to construct a tuple matching $\mathcal{TM}_Q$ based on which $Q(\mathbf{D})$ bounds $Q(D)$. From this then immediately follows that $Q(\mathcal{D}) \sqsubset Q(\mathbf{D})$. Note that by definition of $\mathcal{K}_{AU}$ as the 3-way direct product of $\mathcal{K}$ with itself, semiring operations are point-wise, e.g., $k_1 +_{\mathcal{K}_{AU}} k_2{}^{\downarrow} = k_1{}^{\downarrow} +_{\mathcal{K}} k_2{}^{\downarrow}$. Practically, this means that queries are evaluated over each dimension individually. We will make use of this fact in the following. We only prove that $\mathcal{TM}_Q$ is a lower bound since the proof for $\mathcal{TM}_Q$ being an upper bound is symmetric.

$\underline{\pi_U(\mathbf{R})}$: Recall that for $\mathcal{TM}$ to be a tuple matching, two conditions have to hold: (i) $\mathcal{TM}(\mathbf{t}, t) = \mathbb{0}_\mathcal{K}$ if $t \not\sqsubseteq \mathbf{t}$ and (ii) $\sum_{\mathbf{t} \in \mathbb{D}_I{}^n} \mathcal{TM}(\mathbf{t}, t) = R(t)$. Consider an U-tuple t. Applying the definition of projection for $\mathcal{K}$-relations we have:

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

Since $\mathcal{TM}_R$ is a tuple matching based on which $\mathbf{R}$ bounds R, we know that by the definition of tuple matching the sum of annotations assigned to a tuple t by the tuple matching is equal to the annotation of the tuple in R):

$$\sum_{t=t'[U]} R(t) = \sum_{t=t'[U]} \sum_{\mathbf{t} \in \mathbb{D}_I{}^n} \mathcal{TM}_R(\mathbf{t}, t') \tag{5.5}$$

By definition for any tuple matching $\mathcal{TM}$ we have $\mathcal{TM}(\mathbf{t}, t) = \mathbb{0}_\mathcal{K}$ if $t \not\sqsubseteq \mathbf{t}$. Thus, Equation (5.5) can be rewritten as:

$$= \sum_{t=t'[U]} \sum_{t' \sqsubseteq \mathbf{t}} \mathcal{TM}_R(\mathbf{t}, t') \tag{5.6}$$

Observe that for any n-ary range-annotated $\mathbf{t}$ and n-ary tuple t it is the case that $t \sqsubseteq \mathbf{t}$ implies $t[U] \sqsubseteq \mathbf{t}[U]$ (if $\mathbf{t}$ matches t on all attributes, then clearly it matches t on a subset of attributes). For pair t and $\mathbf{t}$ such that $t[U] \sqsubseteq \mathbf{t}[U]$, but $t \not\sqsubseteq \mathbf{t}$ we know that $TM_R(\mathbf{t}, t) = \mathbb{0}_\mathcal{K}$. Thus,

$$= \sum_{t=t'[U]} \sum_{\mathbf{t}=\mathbf{t}'[U] \wedge t \sqsubseteq \mathbf{t}} \mathcal{TM}_R(\mathbf{t}', t') \tag{5.7}$$

So far we have established that:

$$\pi_U(R)(t) = \sum_{t=t'[U]} \sum_{\mathbf{t}=\mathbf{t}'[U] \wedge t \sqsubseteq \mathbf{t}} \mathcal{TM}_R(\mathbf{t}', t') \tag{5.8}$$

We now define $\mathcal{TM}_Q$ as shown below:

$$\mathcal{TM}_Q(\mathbf{t}, t) \coloneqq \sum_{\forall \mathbf{t}', t : \mathbf{t}'[U]=\mathbf{t} \wedge t'[U]=t} \mathcal{TM}_R(\mathbf{t}', t') \tag{5.9}$$

$\mathcal{TM}_Q$ is a tuple matching since Equation (5.8) ensures that $\forall t \in \mathbb{D}^n : \sum_{\mathbf{t} \in {\mathbb{D}_I}^n} \mathcal{TM}(\mathbf{t}, t) = R(t)$ (second condition in the definition) and we defined $\mathcal{TM}_Q$ such that $\mathcal{TM}_Q(\mathbf{t}, t) = \mathbb{0}_\mathcal{K}$ if $t \not\sqsubseteq \mathbf{t}$. What remains to be shown is that $\pi_U(\mathbf{R})$ bounds $\pi_U(R)$ based on $\mathcal{TM}_Q$. Let $|U| = m$, we have to show that

$$\forall \mathbf{t} \in {\mathbb{D}_I}^m : \pi_U(\mathbf{R})(\mathbf{t})^\downarrow \preceq_\mathcal{K} \sum_{t \in \mathbb{D}^m} \mathcal{TM}_Q(\mathbf{t}, t)$$

Since addition in $\mathcal{K}_{AU}$ is pointwise application of $+_\mathcal{K}$, using the definition of projection over $\mathcal{K}$-relations we have

$$\pi_U(\mathbf{R})(\mathbf{t})^\downarrow = \sum_{\mathbf{t}'[U]=\mathbf{t}} R(\mathbf{t}')$$

Furthermore, since $\mathcal{TM}_R$ is a tuple matching based on which $\mathbf{R}$ bounds R,

$$= \sum_{\mathbf{t}'[U]=\mathbf{t} \wedge t \in \mathbb{D}^n} \mathcal{TM}_R(\mathbf{t}, t)$$

Using again the fact that $t \not\sqsubseteq \mathbf{t}$ implies $\mathcal{TM}_R(\mathbf{t}, t) = \mathbb{0}_\mathcal{K}$,

$$\begin{aligned}
&= \sum_{\forall t', \mathbf{t}' : \mathbf{t}'[U]=\mathbf{t} \wedge t' \sqsubseteq \mathbf{t}'} \mathcal{TM}_R(\mathbf{t}', t') \\
&= \sum_{t \sqsubseteq \mathbf{t}} \sum_{\forall t', \mathbf{t}' : \mathbf{t}'[U]=\mathbf{t} \wedge t'[U]=t} \mathcal{TM}_R(\mathbf{t}', t') \\
&= \sum_{t \sqsubseteq \mathbf{t}} \mathcal{TM}_Q(\mathbf{t}, t) = \sum_{t \in \mathbb{D}^m} \mathcal{TM}_Q(\mathbf{t}, t)
\end{aligned}$$

Since we have established that $\pi_U(\mathbf{R})(\mathbf{t})^{\downarrow} \preceq_{\mathcal{K}} \sum_{t \in \mathbb{D}^m} \mathcal{TM}_Q(\mathbf{t}, t)$, $\pi_U(\mathbf{R})$ lower bounds $\pi_U(R)$ via $\mathcal{TM}_Q$.

$\sigma_\theta(\mathbf{R})$: By definition of selection and based on (i) and (ii) as in the proof of projection we have

$$Q(\mathbf{R})(\mathbf{t})^{\downarrow} = \mathbf{R}(\mathbf{t})^{\downarrow} \cdot_{\mathcal{K}} \theta(\mathbf{t})^{\downarrow} \preceq_{\mathcal{K}} \sum_{t \in \mathbb{D}^n} \mathcal{TM}_R(\mathbf{t}, t) \cdot_{\mathcal{K}} \theta(\mathbf{t})^{\downarrow}$$

Assume that for a tuple t we have $t \not\sqsubseteq \mathbf{t}$, then by Def. 21 it follows that $\mathcal{TM}_{rel}(\mathbf{t}, t) = \mathbb{0}_{\mathcal{K}}$. In this case we get $Q(\mathbf{R})(\mathbf{t})^{\downarrow} = \mathbf{R}(\mathbf{t}) \cdot_{\mathcal{K}} \mathbb{0}_{\mathcal{K}} = \mathbb{0}_{\mathcal{K}}$. Since $\mathbb{0}_{\mathcal{K}} +_{\mathcal{K}} k = k$ for any $k \in \mathcal{K}$, we get

$$\sum_{t \in \mathbb{D}^n} \mathcal{TM}_R(\mathbf{t}, t) \cdot_{\mathcal{K}} \theta(\mathbf{t})^{\downarrow} = \sum_{t \sqsubseteq \mathbf{t}} \mathcal{TM}_R(\mathbf{t}, t) \cdot_{\mathcal{K}} \theta(\mathbf{t})^{\downarrow}$$

Note that based on Thm. 8, we have $\theta(\mathbf{t})^{\downarrow} \preceq_{\mathcal{K}} \theta(t)$ since $t \sqsubseteq \mathbf{t}$ from which follows that: $\sum_{t \sqsubseteq \mathbf{t}} \mathcal{TM}_R(\mathbf{t}, t) \cdot_{\mathcal{K}} \theta(\mathbf{t})^{\downarrow} \preceq_{\mathcal{K}} \sum_{t \sqsubseteq \mathbf{t}} \mathcal{TM}_R(\mathbf{t}, t) \cdot_{\mathcal{K}} \theta(t)^{\downarrow}$. It follows that $Q(\mathbf{R})$ lower bounds $Q(R)$ through $\mathcal{TM}(\mathbf{t}, t) \coloneqq \mathcal{TM}_R(\mathbf{t}, t) \cdot_{\mathcal{K}} \theta(t)^{\downarrow}$.

$\mathbf{R} \times \mathbf{S}$: Based on the definition of cross product for $\mathcal{K}$-relations, (i) from above, and that semiring multiplication preserves natural order we get $(\mathbf{R} \times \mathbf{S})(\mathbf{t})^{\downarrow} = \mathbf{R}(\mathbf{t}[R])^{\downarrow} \cdot_{\mathcal{K}} \mathbf{S}(\mathbf{t}[S])^{\downarrow} \preceq_{\mathcal{K}} \sum_{t \in \mathbb{D}^n} \mathcal{TM}_R(\mathbf{t}[R], t) \cdot_{\mathcal{K}} \sum_{t' \in \mathbb{D}^m} \mathcal{TM}_S(\mathbf{t}[S], t')$. Thus, $\mathbf{R} \times \mathbf{S}$ lower bounds $R \times S$ via $\mathcal{TM}_Q(\mathbf{t}, t) \coloneqq \sum_{t \in \mathbb{D}^n} \mathcal{TM}_R(t, \mathbf{t}[R]) \cdot_{\mathcal{K}} \sum_{t \in \mathbb{D}^m} \mathcal{TM}_S(t, \mathbf{t}[S])$.

$\mathbf{R} \cup \mathbf{S}$: Assume that R and S are n-ary relations. Substituting the definition of union and by (i) and (ii) from above we get: $(\mathbf{R} \cup \mathbf{S})(\mathbf{t})^{\downarrow} = \mathbf{R}(\mathbf{t})^{\downarrow} +_{\mathcal{K}} \mathbf{S}(\mathbf{t})^{\downarrow} \preceq_{\mathcal{K}} \sum_{t \in \mathbb{D}^n} \mathcal{TM}_R(\mathbf{t}, t) +_{\mathcal{K}} \sum_{t \in \mathbb{D}^n} \mathcal{TM}_S(\mathbf{t}, t)$. Thus, $\mathbf{R} \cup \mathbf{S}$ lower bounds $R \cup S$ via $\mathcal{TM}_Q(\mathbf{t}, t) \coloneqq \sum_{t \in \mathbb{D}^n} \mathcal{TM}_R(\mathbf{t}, t) +_{\mathcal{K}} \sum_{t \in \mathbb{D}^n} \mathcal{TM}_S(\mathbf{t}, t)$.

5.4 Set Difference

In this section, we discuss the evaluation of queries with set difference over AU-DBs.

5.4.1 Selected-Guess Combiner. In this section we introduce an auxiliary operator for defining set difference over $\mathcal{K}_{AU}$-relations that merges tuples that have the same values in the SGW. The purpose of this operator is to ensure that a tuple in the SGW is encoded as a single tuple in the AU-DB.

The main purpose for using the merge operator is to prevent tuples from over-reducing or over counting. And make sure we can still extract SGW from the non-monotone query result.

Definition 27 (SG-Combiner)**.** *Given a AU-DB relation* $\mathbf{R}$*, the combine operator* Ψ *yields a AU-DB relation by grouping tuples with the same SGW attribute values:*

$$\Psi(\mathbf{R})(\mathbf{t}) \coloneqq \begin{cases} \sum_{\mathbf{t}' : \mathbf{t}^{sg} = \mathbf{t}'^{sg}} \mathbf{R}(\mathbf{t}') & \text{if } \mathbf{t} = Comb(\mathbf{R}, \mathbf{t}^{sg}) \\ 0_k & \text{else} \end{cases}$$

where $Comb(\mathbf{R}, t)$ *defined below computes the mimimum bounding box for the ranges of all tuples from* $\mathbf{R}$ *that have the same SGW values as* t *and are not annotated with* $\mathbb{0}_{\mathcal{K}_{AU}}$*. Let* a *be an attribute from the schema of* $\mathbf{R}$*, then*

$$Comb(\mathbf{R}, t).a^{\downarrow} = \min_{\mathbf{t}^{sg} = t \wedge \mathbf{R}(\mathbf{t}) \neq \mathbb{0}_{\mathcal{K}_{AU}}} \mathbf{t}.a^{\downarrow}$$

$$Comb(\mathbf{R}, t).a^{sg} = \mathbf{t}.a^{sg}$$

$$Comb(\mathbf{R}, t).a^{\uparrow} = \max_{\mathbf{t}^{sg} = t \wedge \mathbf{R}(\mathbf{t}) \neq \mathbb{0}_{\mathcal{K}_{AU}}} \mathbf{t}.a^{\uparrow}$$

The SG-combiner merges all tuples with the same SG attribute values are combined by merging their attribute ranges and summing up their annotations. For instance, consider a relation $\mathbf{R}$ with two tuples $([1/2/2], [1/3/5])$ and $([2/2/4], [3/3/4])$ which are annotated with $(1, 2, 2)$ and $(3, 3, 4)$, respectively. Applying SG-combiner to this relation the two tuples are combined (they have the same SGW values) into a tuple $([1/2/4], [1/3/5])$ annotated with $(1 + 3, 2 + 3, 2 + 4) = (4, 5, 6)$. Before moving on and discussing semantics for set difference and aggrgeation we first establish that the SG-combiner preserves bounds.

Lemma 8. *Let* $\mathbf{R}$ *by a* $\mathcal{K}_{AU}$*-relation that bounds an n-nary* $\mathcal{K}$*-relation* R*. Then* $\Psi(\mathbf{R})$ *bounds* R*.*

Proof: Consider a tuple $t \in \mathbb{D}^n$ and let $supp(\mathbf{R}, t)$ denote the set $\{\mathbf{t} \mid \mathbf{R}(\mathbf{t}) \neq \mathbb{0}_{\mathcal{K}_{AU}} \wedge \mathbf{t}^{sg} = t\}$. Observe that $Comb(\mathbf{R}, t)$ merges the range annotations $supp(\mathbf{R}, t)$. Let $\mathbf{t}$ be the result of $Comb(\mathbf{R}, t)$. Then

$$\Psi(\mathbf{R})(\mathbf{t})^{\downarrow} = \sum_{\mathbf{t}^{sg}=\mathbf{t'}^{sg}} \mathbf{R}(\mathbf{t'}) \preceq_{\mathcal{K}} \sum_{\mathbf{t}^{sg}=\mathbf{t'}^{sg}} \sum_{t \in \mathbb{D}^n} \mathcal{TM}(\mathbf{t}, t)$$

Thus, $\Psi(\mathbf{R})$ bounds R through $\mathcal{TM}'(\mathbf{t}, t) = \sum_{\mathbf{t}^{sg}=\mathbf{t'}^{sg}} \sum_{t \in \mathbb{D}^n} \mathcal{TM}(\mathbf{t'}), t$.

5.4.2 Set Difference. Geerts [91] did extend $\mathcal{K}$-relations to support set difference through m-semirings which are semirings equipped with a monus operation that is used to define difference. The monus operation $-_{\mathcal{K}}$ is defined based on the natural order of semirings as $k_1 -_{\mathcal{K}} k_2 = k_3$ where k_3 is the smallest element from $\mathcal{K}$ *s.t.* $k_2 +_{\mathcal{K}} k_3 \succeq_{\mathcal{K}} k_1$. For instance, the monus of semiring $\mathbb{N}$ is truncating subtraction: $k_1 -_{\mathbb{N}} k_2 = max(0, k_1 - k_2)$. The monus construction for a semiring $\mathcal{K}$ can be lifted through point-wise application to $\mathcal{K}^3$ since $\mathcal{K}^3$-semirings are direct products. We get

$$(k_1, k_2, k_3) -_{\mathcal{K}_{AU}} (l_1, l_2, l_3) = (k_1 -_{\mathcal{K}} l_1, k_2 -_{\mathcal{K}} l_2, k_3 -_{\mathcal{K}} l_3)$$

However, the result of $k -_{\mathcal{K}^3} k'$ for $k, k' \in \mathcal{K}_{AU}$ is not necessarily in $\mathcal{K}_{AU}$, i.e., this semantics for set difference does not preserve bounds even if we disallow range-annotated values. For instance, consider an incomplete $\mathbb{N}$-relations with two possible worlds: $D_1 = \{R(1) \mapsto 2, S(2) \mapsto 1\}$ and $D_2 = \{R(1) \mapsto 1, R(2) \mapsto 1, S(1) \mapsto 3\}$. Here we use $t \mapsto k$ to denote that tuple t is annotated with k. Without using range-annotations, i.e., we can bound these worlds using $\mathbb{N}_{AU}$-database $\mathbf{D}_1$: $\mathbf{D} \coloneqq \{\mathbf{R}(1) \mapsto (1,2,2), \mathbf{R}(2) \mapsto (0,0,1), \mathbf{S}(1) \mapsto (0,0,3), \mathbf{S}(2) \mapsto (0,1,1)\}$. Consider the query $R - S$. Applying the definition of set difference from Geerts [91] which is $(R-S)(t) \coloneqq R(t) -_{\mathcal{K}} S(t)$, for tuple $\mathbf{t} \coloneqq (1)$ we get the annotation $\mathbf{R}(\mathbf{t}) -_{\mathbb{N}_{AU}} \mathbf{S}(\mathbf{t}) =$

$(1,2,2) -_{\mathbb{N}_{AU}} (0,0,3) = (\max(1-0,0), \max(2-0,0), \max(2-3,0) = (1,2,0)$. However, $(1,2,0)^{\downarrow} = 1$ is not a lower bound on the certain annotation of $\mathbf{t}$, since $\mathbf{t}$ is not in the result of the query in D_2 ($max(1-3,0) = 0$). This failure of the point-wise semantics to preserve bounds is not all surprising if we consider the following observation from [46]: because of the negation in set difference, a lower bound on certain answers can turn into an upper bound. To calculate an lower (upper) bound for the result one has to combine a lower bound for the LHS input of the set difference with an upper bound of the RHS. Thus, we can define

$$(\mathbf{R} - \mathbf{S})(\mathbf{t}) \coloneqq (\mathbf{R}(\mathbf{t})^{\downarrow} -_{\mathcal{K}} \mathbf{S}(\mathbf{t})^{\uparrow}, \mathbf{R}(\mathbf{t})^{sg} -_{\mathcal{K}} \mathbf{S}(\mathbf{t})^{sg}, \mathbf{R}(\mathbf{t})^{\uparrow} -_{\mathcal{K}} \mathbf{S}(\mathbf{t})^{\downarrow})$$

to get a result that preserves bounds. For instance, for $\mathbf{t} \coloneqq (1)$ we get $(\max(1-3,0), \max(2-0,0), \max(2-0,0)) = (0,2,2)$.

This semantics is however still not sufficient if we consider range-annotated values. For instance, consider the following $\mathbb{N}_{AU}$-database $\mathbf{D}_2$ that also bounds our example incomplete $\mathbb{N}$-database : $\{\mathbf{R}(1) \mapsto (1,1,1), \mathbf{R}([1/1/2]) \mapsto (1,1,1), \mathbf{S}([1/1/2]) \mapsto (1,1,3)\}$. Observe that tuple (1) from the SGW (D_1) is encoded as two tuples in $\mathbf{D}_2$. To calculate the annotation of this tuple in the SGW we need to sum up the annotations of all such tuples in the LHS and RHS. To calculate lower bound annotations, we need to also use the sum of annotations of all tuples representing the tuple and then compute the monus of this sum with the sum of all annotations of tuples from the RHS that could be equal to this tuple in some world. Two range-annotated tuples may represent the same tuple in some world if all of their attribute values overlap. Conversely, to calculate an upper bound it is sufficient to use annotations of RHS tuples if both tuples are certain (they are the same in every possible world). We use the SG-combiner operator define above to merge tuples with the same SG values and then apply the monus using the appropriate set of tuples from the RHS.

Definition 28 (Set Difference)**.** *Let* $\mathbf{t}$ *and* $\mathbf{t}'$ *be n-ary range-annotated tuples with*

schema $(a_1, \ldots, a_n)$. *We define a predicate* $\mathbf{t} \equiv \mathbf{t}'$ *that evaluates to true iff* $\mathbf{t} = \mathbf{t}'$ *and both* $\mathbf{t}$ *and* $\mathbf{t}'$ *are certain and a predicate* $\mathbf{t} \simeq \mathbf{t}'$ *that evaluates to true iff* $\forall i \in \{1, \ldots, n\} : \mathbf{t}.a_i{}^{\downarrow} \preceq_{\mathcal{K}} \mathbf{t}'.a_i{}^{\downarrow} \preceq_{\mathcal{K}} \mathbf{t}.a_i{}^{\uparrow} \vee \mathbf{t}.a_1{}^{\downarrow} \preceq_{\mathcal{K}} \mathbf{t}'.a_i{}^{\uparrow} \preceq_{\mathcal{K}} \mathbf{t}.a_i{}^{\uparrow}$. *Using these predicates we define set difference as shown below.*

$$(\mathbf{R}_1 - \mathbf{R}_2)(\mathbf{t})^{\downarrow} \coloneqq \Psi(\mathbf{R}_1)(\mathbf{t})^{\downarrow} -_{\mathcal{K}} \sum_{\mathbf{t} \simeq \mathbf{t}'} \mathbf{R}_2(\mathbf{t}')^{\uparrow}$$

$$(\mathbf{R}_1 - \mathbf{R}_2)(\mathbf{t})^{sg} \coloneqq \Psi(\mathbf{R}_1)(\mathbf{t})^{sg} -_{\mathcal{K}} \sum_{\mathbf{t}^{sg} = \mathbf{t}'^{sg}} \mathbf{R}_2(\mathbf{t}')^{sg}$$

$$(\mathbf{R}_1 - \mathbf{R}_2)(\mathbf{t})^{\uparrow} \coloneqq \Psi(\mathbf{R}_1)(\mathbf{t})^{\uparrow} -_{\mathcal{K}} \sum_{\mathbf{t} \equiv \mathbf{t}'} \mathbf{R}_2(\mathbf{t}')^{\downarrow}$$

5.4.3 Bound Preservation. We now demonstrate that the semantics we have defined for set difference preserves bounds.

Theorem 11 (Set Difference Preserves Bounds)**.** *Let* $Q \coloneqq R - S$*, R and S be incomplete $\mathcal{K}$-relations, and* $\mathbf{R}$ *and* $\mathbf{S}$ *be $\mathcal{K}_{AU}$-relations that bound R and S. Then* $Q(\mathbf{R}, \mathbf{S})$ *bounds* $Q(R, S)$*.*

Proof: Given all input tuples are bounded, we first prove that the lower bound of the query semantics reserves the bound. We assume relation $\mathbf{R}$ is pre-combined s.t. $\mathbf{R} = \Psi(\mathbf{R})$ and R preserves the bound.

For lower bounds $(\mathbf{R} - \mathbf{S})(\mathbf{t})^{\downarrow}$, on the L.H.S. of $-_{\mathcal{K}}$ we have

$$\mathbf{R}(\mathbf{t})^{\downarrow} \preceq_{\mathcal{K}} \sum_{t \in \mathbb{D}^n} \mathcal{TM}(\mathbf{t}, t)$$

On the R.H.S. we have

$$\sum_{\mathbf{t} \simeq \mathbf{t}'} \mathbf{S}(\mathbf{t}')^{\uparrow} \succeq_{\mathcal{K}} \sum_{\mathbf{t} \simeq \mathbf{t}'} \sum_{t' \in \mathbb{D}^n} \mathcal{TM}(\mathbf{t}', t')$$

thus

$$
\begin{aligned}
(\mathbf{R} - \mathbf{S})(\mathbf{t})^{\downarrow} :=& \mathbf{R})(\mathbf{t})^{\downarrow} -_{\mathcal{K}} \sum_{\mathbf{t} \simeq \mathbf{t}'} \mathbf{S}(\mathbf{t}')^{\uparrow} \\
&\preceq_{\mathcal{K}} \sum_{t \in \mathbb{D}^n} \mathcal{TM}(\mathbf{t}, t) -_{\mathcal{K}} \sum_{\mathbf{t} \simeq \mathbf{t}'} \sum_{t' \in \mathbb{D}^n} \mathcal{TM}(\mathbf{t}', t') \\
&= \sum_{t \in \mathbb{D}^n} (\mathcal{TM}(\mathbf{t}, t) -_{\mathcal{K}} \sum_{t' \in \mathbb{D}^n} \mathcal{TM}(\mathbf{t}', t'))
\end{aligned}
$$

So the lower bounds is bounded by tuple-matching $\forall_{\mathbf{t} \in \mathbf{R}} : \mathcal{TM}(\mathbf{t}, t) = \mathcal{TM}(\mathbf{t}, t) -_{\mathcal{K}} \sum_{t' \in \mathbb{D}^n} \mathcal{TM}(\mathbf{t}', t')$.

5.5 Aggregation

We now introduce a semantics for aggregation over AU-DBs that preserves bounds. We leave a generalization to other semirings to future work. See [43] for a discussion of the challenges involved with that. Importantly, our semantics has `PTIME` data complexity. One major challenge in defining aggregation over $\mathcal{K}$-relations which also applies to our problem setting is that one has to take the annotations of tuples into account when calculating aggregation function results. For instance, under bag semantics (semiring $\mathbb{N}$) the multiplicity of a tuple affects the result of `SUM` aggregation. We based our semantics for aggregation on earlier results from [23]. For AU-DBs we have to overcome two major new challenges: (i) since the values of group-by attributes may be uncertain, a tuple's group membership may be uncertain too and (ii) we are aggregating over range-bounded values. To address (ii) we utilitze our expression semantics for range-bounded values from Sec. 5.1. However, additional complications arise when taking the $\mathbb{N}_{AU}$-annotations of tuples into account. For (i) we will reason about all possible group memberships of range-annotated tuples to calculate bounds on group-by values, aggregation function results, and number of result groups.

5.5.1 Aggregation Monoids. Amsterdamer et al. [23] introduced a semantics for aggregation queries over $\mathcal{K}$-relations that commutes with homomorphisms and

under which aggregation results can be encoded with polynomial space. Contrast this with the aggregation semantics for c-tables from [67] where aggregation results may be of size exponential in the input size. [23] deals with aggregation functions that are commutative monoids $(M, +_M, \mathbb{0}_M)$, 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. A monoid is a mathematical structure $(M, +_M, \mathbb{0}_M)$ where $+_M$ is a commutative and associative binary operation over M, and $\mathbb{0}_M$ is the neutral element of M. For instance, $\mathsf{SUM} := (\mathbb{R}, +, 0)$, i.e., addition over the reals can be used for sum aggregation. Most standard aggregation functions (**sum**, **min**, **max**, and **count**) can be expressed as monoids or, in the case of **avg**, can be derived from multiple monoids (count and sum). As an example, consider the monoids for **sum** and **min**: $\mathsf{SUM} := (\mathbb{R}, +, 0)$ and $\mathsf{MIN} := (\mathbb{R}, \min, \infty)$. 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. 5.1). 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 9. SUM_I, MIN_I, MAX_I *are monoids.*

Proof: Addition in $\mathbb{D}_I$ is applied point-wise. Thus, addition in $\mathbb{D}_I$ is commutative and associative and has neutral element $[0/0/0]$. Thus, SUM_I is a monoid. For MIN_I if we substitute the definition of $\textbf{if } v \leq w \textbf{ then } v \textbf{ else } w$ and $v \leq w$, if simplifying the resulting expression we get

$$\begin{aligned}&\min([a_1/a_2/a_3], [b_1/b_2/b_3])\\ =&[\min(a_1, b_1)/\min(a_2, b_2)/\min(a_3, b_3)]\end{aligned}$$

That is, the operation is again applied pointwise and commutativity, associativity,

$$k *_M (m_1 +_M m_2) = k *_{\mathcal{K}} m_1 +_M k *_{\mathcal{K}} m_2 \tag{5.10}$$

$$(k_1 +_{\mathcal{K}} k_2) *_M m = k_1 *_{\mathcal{K}} m +_M k_2 *_{\mathcal{K}} m \tag{5.11}$$

$$\mathbb{1}_{\mathcal{K}} *_M m = m \tag{5.12}$$

$$\mathbb{0}_{\mathcal{K}} *_M m = \mathbb{0}_M \tag{5.13}$$

$$k *_M \mathbb{0}_M = \mathbb{0}_M \tag{5.14}$$

$$(k_1 \cdot_{\mathcal{K}} k_2) *_M m = k_1 *_M (k_2 *_M m) \tag{5.15}$$

Figure 5.1. Semimodule laws (semiring $\mathcal{K}$ paired with monoid M)

and identity of the neutral element ($[\infty/\infty/\infty]$) follow from the fact that MIN is a monoid. The proof for max is symmetric.

Based on Lem. 9, aggregation functions over range-annotated values preserve bounds.

Corollary 2 (Aggregation Functions Preserve Bounds)**.** *Let $\mathcal{S} = \{c_1, \ldots, c_n\} \subseteq \mathbb{D}_I$ be a set of range-annotated values and $S = \{d_1, \ldots d_n\} \subseteq \mathbb{D}$ be a set of values such that c_i bounds d_i, and $M_I \in \{\mathsf{SUM}_I, \mathsf{MAX}_I, \mathsf{MIN}_I\}$, then using the addition operation of M_I (M) we have that $\sum \mathcal{S}$ bounds $\sum S$.*

Proof: The corollary follows immediately from Thm. 8 and Lem. 9.

Semimodules: One challenge of supporting aggregation over $\mathcal{K}$-relations is that the annotations of tuples have to be factored into the aggregation computation. For instance, consider an $\mathbb{N}$-relation $R(A)$ with two tuples $(30) \mapsto 2$ and $(40) \mapsto 3$, i.e., there are two duplicates of tuple (30) and 3 duplicates of tuple (40). Computing the sum over A we expect to get $30 \cdot 2 + 40 \cdot 3 = 180$. More generally speaking, we need an operation $*_M : \mathbb{N} \times M \to M$ that combines semiring elements with values from the aggregation function's domain. As observed in [23] this operation has to be a

semimodule, i.e., it has to fulfill a set of equational laws, all of which are shown for $\mathbb{N}$ in Fig. 5.1. Note that in the example above we made use of the fact that $*_{\mathbb{N},\mathsf{SUM}}$ is $\cdot$ to get $30 *_{\mathbb{N}} 2 = 30 \cdot 2 = 60$. Operation $*$ is not well-defined for all semirings, but it is defined for $\mathbb{N}$ and all of the monoids we consider. We show the definition for $*_{\mathbb{N},M}$ for all considered monoids below:

$$k *_{\mathbb{N},\mathsf{SUM}} m = k \cdot m$$

$$k *_{\mathbb{N},\mathsf{MIN}} m = k *_{\mathsf{MAX}} m = \begin{cases} m & \text{if } k \neq 0 \\ 0 & \text{else} \end{cases}$$

The Tensor Construction: Amsterdamer et al [23]. demonstrated that there is no meaningful way to define semimodules for all combinations of semirings and standard aggregation function monoids. To be more precise, if aggregation function results are concrete values from the aggregation monoid, then it is not possible to retain the important property that queries commute with homomorphisms. Intuitively, that is the case because applying a homomorphism to the input may change the aggregation function result. Hence, it is necessary to delay the computation of concrete aggregation results by keeping the computation symbolic. For instance, consider an $\mathbb{N}[X]$-relation $R(A)$ (provenance polynomials) with a single tuple $(30) \mapsto x_1$. If we compute the sum over A, then under a homomorphism $h_1 : x_1 \to 1$ we get a result of $30 \cdot 1 = 30$ while under a homomorphism $h_2 : x_1 \to 2$ we get $30 \cdot 2 = 60$. The solution presented in [23] uses monoids whose elements are symbolic expressions that pair semiring values with monoid elements. Such monoids are compatible with a larger class of semirings including, e.g., the provenance polynomial semiring. Given a semiring $\mathcal{K}$ and aggregation monoid M, the symbolic commutative monoid has as domain bags of elements from $\mathcal{K} \times M$ with bag union as addition (denoted as $+_{\mathcal{K}\otimes M}$) and the emptyset as neutral element. This structure is then extended to a $\mathcal{K}$-semimodule $\mathcal{K} \otimes M$ by defining $k *_{\mathcal{K}\otimes M} \sum k_i \otimes m_i \coloneqq \sum(k \cdot_{\mathcal{K}} k_i) \otimes m_i$ and taking the quotient

(the structure whose elements are equivalent classes) wrt. the semimodule laws. For some semirings, e.g., $\mathbb{N}$ and $\mathbb{B}$, the symbolic expressions from $\mathcal{K} \otimes M$ correspond to concrete aggregation result values from M.[8] However, this is not the case for every semiring and aggregation monoid. For instance, for most provenance semirings these expressions cannot be reduced to concrete values. Only by applying homomorphisms to semirings for which this construction is isomorphic to the aggregation monoid is it possible to map such symbolic expressions back to concrete values. For instance, computing the sum over the $\mathbb{N}[X]$-relation $R(A)$ with tuples $(30) \mapsto x_1$ and $(20) \mapsto x_2$ yields the symbolic expression $30 \otimes x_1 +_{\mathbb{N}[X] \otimes \mathsf{SUM}} 20 \otimes x_2$. If the input tuple annotated with x_1 occurs with multiplicity 2 and the input tuple annotated with x_2 occurs with multiplicity 4 then this can be expressed by applying a homomorphism $h : \mathbb{N}[X] \to \mathbb{N}$ defined as $h(x_1) = 2$ and $h(x_2) = 4$. Applying this homomorphism to the symbolic aggregation expression $30 \otimes x_1 +_{\mathbb{N}[X] \otimes \mathsf{SUM}} 20 \otimes x_2$, we get the expected result $30 \cdot 2 + 20 \cdot 4 = 140$. If we want to support aggregation for $\mathcal{K}_{AU}$-relations with this level of generality then we would have to generalize range-annotated values to be symbolic expressions from $\mathcal{K} \otimes M$ and would have to investigate how to define an order over such values to be able to use them as bounds in range-annotated values. For instance, intuitively we may bound $(x_1 + x_2) \otimes 3 +_{\mathbb{N}[X] \otimes \mathsf{SUM}} x_3 \otimes 2$ from below using $x_1 \otimes 3 +_{\mathbb{N}[X] \otimes \mathsf{SUM}} x_3 \otimes 1$ since $x_1 \preceq_{\mathbb{N}[X]} x_1 + x_2$ and $2 < 3$. Then we would have to show that aggregation computations preserve such bounds to show that queries with aggregation with this semantics preserve bounds. We trade generality for simplicity by limiting the discussion to semirings where $\mathcal{K} \otimes M$ is isomorphic to M. This still covers the important cases of bag semantics and set semantics ($\mathbb{N}$ and $\mathbb{B}$), but has the advantage that we are not burdening the user with interpreting bounds that are complex symbolic expressions. For instance, consider an aggregation without group-by over a relation with millions of rows. The resulting bound expressions for

[8] That is the case when M and $\mathcal{K} \otimes M$ are isomorphic.

the aggregation result value may contain millions of terms which would render them completely useless for human consumption. Additionally, while query evaluation is still PTIME when using $\mathcal{K} \otimes M$, certain operations like joins on aggregation results are inefficient.[9]

5.5.2 Applying Semimodules to $\mathbb{N}_{AU}$-Relations. As we will demonstrate in the following, even though it may be possible to define $\mathcal{K}_{AU}$-semimodules, such semimodules cannot be bound preserving and, thus, would be useless for our purpose. We then demonstrate that it is possible to define bound preserving operations that combine $\mathbb{N}_{AU}$ elements with $\mathbb{D}_I$ elements and that this is sufficient for defining a bound preserving semantics for aggregation.

Lemma 10 (Bound preserving $\mathbb{N}_{AU}$-semimodules are impossible)**.** *The semimodule for $\mathbb{N}_{AU}$ and* SUM*, if it exists, cannot be bound preserving.*

Proof: For sake of contadiction assume that this semimodule exists and is bound preserving. Consider $k = (1, 1, 2)$ and $m = [0/0/0]$. Then by semimodule law 5.14 we have $k \circledast_{\mathsf{SUM}} m = m = [0/0/0]$. Now observe that for $m_1 = [-1/-1/-1]$ and $m_2 = [1/1/1]$ we have $m = m_1 + m_2$. Let $m_1' = k \circledast_{\mathsf{SUM}} m_1$. We know that $m_1' = [l_1/-1/u_1]$ for some l_1 and u_1. Since the semimodule is assumed to be bound preserving we know that $l_1 \leq -2$ and $u_1 \geq -1$ ($-1 \cdot 2 = -2$ and $-1 \cdot 1 = -1$). Analog, let $m_2' = [l_2/1/u_2] = k \circledast_{\mathsf{SUM}} m_2$. By the same argument we get $l_2 \leq 1$ and $u_2 \geq 2$. Applying semimodule law 5.10 we get $k \circledast_{\mathsf{SUM}} m = k \circledast_{\mathsf{SUM}} (m_1 + m_2) = m_1' + m_2' = [l_1 + l_2/0/u_1 + u_2]$. Let $l' = l_1 + l_2$, $u' = u_1 + u_2$ and $m'' = [l'/0/u']$. Based on the inequalities constraining l_i and u_i we know that $l_1 + l_2 \leq -1$ and $u_1 + u_2 \geq 1$. Thus,

[9]Comparing symbolic expressions requires an extension of annotations to treat these comparisons symbolically. The reason is that since an aggregation result cannot be mapped to a concrete value, it is also not possible to determine whether such a values are equal. The net result is that joins on such values may degenerate to cross products.

we have the contradiction $k \circledast_{\mathsf{SUM}} m = [0/0/0] \neq [l'/0/u'] = k \circledast_{\mathsf{SUM}} m$.

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. 5.1. For example, since $*_{\mathbb{N},\mathsf{SUM}}$ is multiplication, we can define $*_{\mathbb{N}_{AU},\mathsf{SUM}}$ using our definition of multiplication for range-annotated expression evaluation. It turns out that 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:

Definition 29. *Consider an aggregation monoid M such that $*_{\mathbb{N},M}$ is well defined. 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 $\circledast_M$ as shown below.*

$$(k^{\downarrow}, k, k^{\uparrow}) \circledast_M [m^{\downarrow}/m/m^{\uparrow}] =$$

$$\begin{aligned}
&(min(k^{\downarrow} *_{\mathbb{N},M} m^{\downarrow}, k^{\downarrow} *_{\mathbb{N},M} m^{\uparrow}, k^{\uparrow} *_{\mathbb{N},M} m^{\downarrow}, k^{\uparrow} *_{\mathbb{N},M} m^{\uparrow}),\\
&k *_{\mathbb{N},M} m,\\
&max(k^{\downarrow} *_{\mathbb{N},M} m^{\downarrow}, k^{\downarrow} *_{\mathbb{N},M} m^{\uparrow}, k^{\uparrow} *_{\mathbb{N},M} m^{\downarrow}, k^{\uparrow} *_{\mathbb{N},M} m^{\uparrow}))
\end{aligned}$$

As the following theorem demonstrates $*_{\mathbb{N}_{AU},M}$ is in fact bound preserving.

Theorem 12. *Let $M \in \{\mathsf{MIN}, \mathsf{MAX}\}$ and $\mathcal{K} \in \{\mathbb{B}, \mathbb{N}\}$ or $M = \mathsf{SUM}$ and $\mathcal{K} = \mathbb{N}$. Then $\circledast_M$ preserves bounds.*

Proof: We first prove the theorem for $\mathbb{N}$. We have to show for all $M \in \{\mathsf{MIN}, \mathsf{MAX}, \mathsf{SUM}\}$ that for any $\vec{k} = (k^{\downarrow}, k^{sg}, k^{\uparrow}) \in \mathbb{N}_{AU}$ and $\vec{m} = [m^{\downarrow}/m^{sg}/m^{\uparrow}] \in \mathbb{D}_I$ we have that $\vec{k} \circledast_M \vec{m}$

bounds $k *_{\mathbb{N},M} m$ for any k bound by $\vec{k}$ and m bound by $\vec{m}$. We prove the theorem for each $M \in \{\mathsf{MIN}, \mathsf{MAX}, \mathsf{SUM}\}$.

$\underline{M = \mathsf{SUM}:}$ We have $k *_{\mathbb{N},\mathsf{SUM}} m \coloneqq k \cdot m$. We distinguish four cases:

$\underline{m^{\downarrow} < 0,\ m^{\uparrow} < 0:}$ We have that $k^{\uparrow} \cdot m^{\downarrow} = min(k^{\downarrow} \cdot m^{\downarrow}, k^{\downarrow} \cdot m^{\uparrow}, k^{\uparrow} \cdot m^{\downarrow}, k^{\uparrow} \cdot m^{\uparrow})$ and $k^{\downarrow} \cdot m^{\uparrow} = max(k^{\downarrow} \cdot m^{\downarrow}, k^{\downarrow} \cdot m^{\uparrow}, k^{\uparrow} \cdot m^{\downarrow}, k^{\uparrow} \cdot m^{\uparrow})$. Thus,

$$\vec{k} \circledast_M \vec{m} = [k^{\uparrow} \cdot m^{\downarrow} / k^{sg} \cdot m^{sg} / k^{\downarrow} \cdot m^{\uparrow}]$$

Now for any k bound by $\vec{k}$ and m bound by $\vec{m}$ we have: $k^{\uparrow} \cdot m^{\downarrow} \leq k \cdot m$ because m is a negative number and $k \leq k^{\uparrow}$. Analog, $k \cdot m \leq k^{\downarrow} \cdot m^{\uparrow}$, because m is negative and $m^{\uparrow} \geq m$ and $k^{\downarrow} \leq k$. Thus, $\vec{k} \circledast_M \vec{m}$ bounds $k *_{\mathbb{N},M} m$.

$\underline{m^{\downarrow} \geq 0,\ m^{\uparrow} \geq 0:}$ We have that $k^{\downarrow} \cdot m^{\downarrow} = min(k^{\downarrow} \cdot m^{\downarrow}, k^{\downarrow} \cdot m^{\uparrow}, k^{\uparrow} \cdot m^{\downarrow}, k^{\uparrow} \cdot m^{\uparrow})$ and $k^{\uparrow} \cdot m^{\uparrow} = max(k^{\downarrow} \cdot m^{\downarrow}, k^{\downarrow} \cdot m^{\uparrow}, k^{\uparrow} \cdot m^{\downarrow}, k^{\uparrow} \cdot m^{\uparrow})$. Thus,

$$\vec{k} \circledast_M \vec{m} = [k^{\downarrow} \cdot m^{\downarrow} / k^{sg} \cdot m^{sg} / k^{\uparrow} \cdot m^{\uparrow}]$$

Now for any k bound by $\vec{k}$ and m bound by $\vec{m}$ we have: $k^{\uparrow} \cdot m^{\downarrow} \leq k \cdot m$ because m is a positive number and $k \geq k^{\downarrow}$. Analog, $k \cdot m \leq k^{\uparrow} \cdot m^{\uparrow}$, because m is negative and $m^{\uparrow} \geq m$ and $k^{\uparrow} \geq k$. Thus, $\vec{k} \circledast_M \vec{m}$ bounds $k *_{\mathbb{N},M} m$.

$\underline{m^{\downarrow} < 0,\ m^{\uparrow} \geq 0:}$ We have that $k^{\uparrow} \cdot m^{\downarrow} = min(k^{\downarrow} \cdot m^{\downarrow}, k^{\downarrow} \cdot m^{\uparrow}, k^{\uparrow} \cdot m^{\downarrow}, k^{\uparrow} \cdot m^{\uparrow})$ and $k^{\uparrow} \cdot m^{\uparrow} = max(k^{\downarrow} \cdot m^{\downarrow}, k^{\downarrow} \cdot m^{\uparrow}, k^{\uparrow} \cdot m^{\downarrow}, k^{\uparrow} \cdot m^{\uparrow})$. Thus,

$$\vec{k} \circledast_M \vec{m} = [k^{\uparrow} \cdot m^{\downarrow} / k^{sg} \cdot m^{sg} / k^{\uparrow} \cdot m^{\uparrow}]$$

Now consider some k bound by $\vec{k}$ and m bound by $\vec{m}$. If m is positive, then trivially $k^{\downarrow} \cdot m^{\downarrow}$ bounds $k \cdot m$ from below since $m^{\downarrow}$ is negative. Otherwise, the lower bound holds using the argument for the case of $m^{\downarrow} < 0$, $m^{\uparrow} < 0$. If m is negative, then trivally $k^{\uparrow} \cdot m^{\uparrow}$ bounds $k \cdot m$ from above since $m^{\uparrow}$ is positive. Otherwise, the

upper bound holds using the argument for the case of $m^{\downarrow} \geq 0$, $m^{\uparrow} \geq 0$. Thus, $\vec{k} \circledast_M \vec{m}$ bounds $k *_{\mathbb{N},M} m$.

$\underline{M = \mathsf{MIN}}$: We have

$$k *_{\mathbb{N},\mathsf{MIN}} m \coloneqq \begin{cases} 0 & \text{if } k = 0 \\ m & \text{otherwise} \end{cases}$$

We distinguish three cases.

$\underline{k^{\downarrow} \geq 0}$: If $k^{\downarrow} \geq 0$, then $k *_{\mathbb{N},\mathsf{MIN}} m$ returns m and $\vec{k} \circledast_{\mathsf{MIN}} \vec{m}$ returns $\vec{m}$. Since $\vec{m}$ bounds m also $\vec{k} \circledast_{\mathsf{MIN}} \vec{m}$ bounds $k *_{\mathbb{N},\mathsf{MIN}} m$.

$\underline{k^{\downarrow} = 0,\ k^{\uparrow} \geq 0}$: Now consider the remaining case: $k^{\downarrow} = 0$. Then the result of $\vec{k} \circledast_{\mathsf{MIN}} \vec{m}$ simplifies to $[min(0, m^{\downarrow})/k^{sg} \cdot m^{sg}/max(0, m^{\uparrow})]$. Now consider some k bound by $\vec{k}$ and m bound by $\vec{m}$. If $k \neq 0$ then $k *_{\mathbb{N},\mathsf{MIN}} m = m$ and the claim to be proven holds. Otherwise, $k *_{\mathbb{N},\mathsf{MIN}} m = 0$ which is bound by $[min(0, m^{\downarrow})/k^{sg} \cdot m^{sg}/max(0, m^{\uparrow})]$.

$\underline{k^{\downarrow} = 0,\ k^{\uparrow} = 0}$: In this case $\vec{k} \circledast_{\mathsf{MIN}} \vec{m} = [0/0/0]$ and since $k = 0$ because $\vec{k}$ bounds k we have $k *_{\mathbb{N},\mathsf{MIN}} m = 0$ which is trivally bound by $[0/0/0]$.

$\underline{M = \mathsf{MAX}}$: We have

$$k *_{\mathbb{N},\mathsf{MAX}} m \coloneqq \begin{cases} 0 & \text{if } k = 0 \\ m & \text{otherwise} \end{cases}$$

The proof for MAX is analog to the proof for MIN.

For semiring $\mathbb{B}$ and $M = \mathsf{MIN}$ or $M = \mathsf{MAX}$ observe that $k *_{\mathbb{B},M} m$ is the identity for m if $k \neq \mathbb{0}_{\mathbb{B}}$ and $k *_{\mathbb{B},M} m = 0$ otherwise. Thus, the proof is analog to the proof for semiring $\mathbb{N}$.

5.5.3 Bound-Preserving Aggregation. We now define a bound preserving aggregation semantics based on the $\circledast_M$ operations. As mentioned above, the main

challenge we have to overcome is to deal with the uncertainty of group memberships plus the resulting uncertainty in the number of groups and of which inputs contribute to a group's aggregation function result values. In general, the number of possible groups encoded by an input AU-DB-relation may be very large. Thus, enumerating all concrete groups is not a viable option. While 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. We, thus, formalize grouping strategies and define a semantics for aggregation that preserves bounds for any such grouping semantics. Additionally, we present a reaonsable default strategy. We define our aggregation semantics in three steps: (i) we introduce grouping strategies and our default grouping strategy that matches SG and possible input groups to output tuples (each output tuple will represent exactly one group in the SGW and one or more possible groups); (ii) we calculate group-by attribute ranges for output tuples based on the assignment of input tuples to output tuples; (iii) we calculate the multiplicities (annotations) and bounds for aggregation function results for each output tuple.

5.5.4 Grouping Strategies. A *grouping strategy* $\mathbb{G}$ is a function that takes as input an n-ary $\mathcal{K}_{AU}$-relation $\mathbf{R}$ and list of group-by attributes G and returns a triple $(\mathcal{G}, \psi, \alpha)$ where $\mathcal{G}$ is a set of output groups, ψ is a function associating each input tuple $\mathbf{t}$ from $\mathbf{R}$ where $\mathbf{R}(\mathbf{t})^{sg} \neq \mathbb{0}_{\mathcal{K}}$ with an output from $\mathcal{G}$, and α is a function associating each input tuple $\mathbf{t}$ from $\mathbf{R}$ where $\mathbf{R}(\mathbf{t}) \neq \mathbb{0}_{\mathcal{K}_{AU}}$ with an output from $\mathcal{G}$. Note that the elements of $\mathcal{G}$ are just unique identifiers for output tuples. The actual range-annotated output tuples returned by an aggregation operator are not returned by the grouping strategy directly but are constructed by our aggregation semantics based on the information returned by a grouping strategy. Intuitively, ψ takes care of the association of groups in the SGW with an output while α does the same for

all possible groups. For $\mathbb{G}$ to be a grouping strategy we require that for any input relation $\mathbf{R}$ and list of group-by attributes G we have:

$$\forall \mathbf{t}, \mathbf{t}' \in {\mathbb{D}_I}^n : \mathbf{R}(\mathbf{t}) \neq \mathbb{0}_{\mathcal{K}_{AU}} \wedge \mathbf{R}(\mathbf{t}') \neq \mathbb{0}_{\mathcal{K}_{AU}} \wedge \mathbf{t}.G^{sg} = \mathbf{t}'.G^{sg}$$
$$\rightarrow \psi(\mathbf{t}) = \psi(\mathbf{t}')$$

This condition ensures that for every tuple that exists in the SGW, all inputs that exists in the SGW and belong this group are associated with a single output. Our aggregation semantics relies on this property to produce the correct result in the SGW. We use function α to ensure that every possible group is accounted for by the $\mathcal{K}_{AU}$-relation returned as the result of aggregation. Inuitively, every range-annotated input tuple may correspond to several possible groups based on the range annotations of its group-by attribute values. Our aggregation semantics ensures that an output tuple's group-by ranges bound the group-by attribute ranges of every input associated to it by α.

5.5.5 Default Grouping Strategy. Our *default grouping strategy* takes as input a n-ary $\mathbb{N}_{AU}$-relation $\mathbf{R}$ and list of group-by attributes G and returns a pair $(\mathcal{G}, \alpha)$ where $\mathcal{G}$ is a set of output tuples — one for every SG group, i.e., an input tuple's group-by values in the SGW. α assigns each input tuple to one output tuple based on its SG group-by values. Note that 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 30 (Default Grouping Strategy)**.** *Consider a query* $Q \coloneqq \gamma_{G,f(A)}(\mathbf{R})$*. Let* $\mathbf{t} \in {\mathbb{D}_I}^n$ *such that* $\mathbf{R}(\mathbf{t}) \neq \mathbb{0}_{\mathbb{N}_{AU}}$ *and* $\mathbf{t}' \in {\mathbb{D}_I}^n$ *such that* $\mathbf{R}(\mathbf{t}')^{sg} \neq 0$*. The default*

grouping strategy $\mathbb{G}_{def} \coloneqq (\mathcal{G}, \alpha)$ *is defined as shown below.*

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

For instance, consider three tuples $\mathbf{t}_1 \coloneqq ([1/2/2])$ and $\mathbf{t}_2 \coloneqq ([2/2/4])$ and $\mathbf{t}_3 \coloneqq ([2/3/4])$ over schema $\leftrightarrow (A)$. Furthermore, assume that $\mathbf{R}(\mathbf{t}_1) = (1,1,1)$, $\mathbf{t}(\mathbf{t}_2) = (0,0,1)$, and $\mathbf{t}(\mathbf{t}_3) = (0,0,3)$. 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, the possible grouping function α assigns $\mathbf{t}_1$ and $\mathbf{t}_2$ to g_1 and $\mathbf{t}_3$ to g_2.

5.5.6 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.

Group-by Bounds: To ensure that all possible groups an input tuple $\mathbf{t}$ with $\alpha(\mathbf{t}) = \mathbf{t}_g$ 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 set $\mathbf{t}_g.G^{sg} = \mathbf{t}_g$, i.e., we use the unique SG group-by values of 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 31 (Range-bounded Groups)**.** *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. Let* g *be the unique*

element from the set $\{\mathbf{t}.G^{sg} \mid \mathbf{R}(\mathbf{t})^{sg} \neq 0 \wedge \alpha(\mathbf{t}) = \mathbf{t}_g\}$. *For all* $a \in G$ *we define:*

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

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 any group $\mathbf{t}_1$ and $\mathbf{t}_2$ may belong to in some possible world.

Aggregation Function Bounds: 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 row annotations, and their possible and guaranteed group memberships (even when a value of the aggregation function input attribute is certain the group membership of the tuple it belongs too may be uncertain). 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 $\mathbf{t}$ with $\alpha(\mathbf{t}) = g$) with the tuple's annotation and then extract the lower bound from the resulting range-annotated value. For some tuples their group membership is uncertain because either their group-by values are uncertain or they may not exist in all possible worlds (their certain multiplicity is 0). We take this into account by taking the minimum of the neutral element of the aggregation monoid and the result of $\circledast_M$ for such tuples. Towards this goal we introduce a predicate $\textsc{ug}(G, \mathbf{R}, \mathbf{t})$ that is defined as shown below.

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

We then sum up the resulting values in the aggregation monoid. Note that here summation is assumed to use addition in M. The upper bound calculation is analog

(using the upper bound and maximum instead). The SG result is calculated using standard $\mathcal{K}$-relational semantics. In the definition we will 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.,

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

Definition 32 (Aggregation Function Result Bounds)**.** *Consider an output $g \in \mathcal{G}$, input $\mathbf{R}$, group-by attributes G, and aggregation function $f(A)$ with monoid M. We use $\eth(g)$ to denote the set of input tuples whose group-by attribute bounds overlap with $\mathbf{t}_g.G$, i.e., they may be belong to a group represented by $\mathbf{t}_g$:*

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

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

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

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

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

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

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

Example 18. *For instance, consider calculating the sum of A grouping on B for a relation $R(A, B)$ which consists of two tuples $\mathbf{t}_3 \coloneqq ([3/5/10], [3/3/3])$ and $\mathbf{t}_4 \coloneqq ([-4/-3/-3], [2/3/4])$ which 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 *which corresponds to SG group* $g \coloneqq (3)$. *The lower bound on* `sum(A)` *is calculated as shown below:*

$$\begin{aligned}
&\sum_{\mathbf{t} \in \eth(g)} \text{LBAGG}(\mathbf{t}) \\
=&((1,2,2) \cdot [3/5/10])^{\downarrow} + min(0, ((1,2,2) \cdot [-4/-3/-3])^{\downarrow}) \\
=&[3/10/20]^{\downarrow} + min(0, [-8/-6/-3]^{\downarrow}) \\
=&3 + min(0, -8) = -5
\end{aligned}$$

The aggregation result is guaranteed to be greater than or equal to -5 *since* $\mathbf{t}_3$ *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* $\mathbf{t}_4$ *contributions at least* -8. *While it is possible that* $\mathbf{t}_4$ *does not belong to* g *this can only increase the final result (*$3 + 0 > 3 + -8$*).*

Aggregation Without Group-by: Having defined how each output tuple of aggregation is constructed we still need to calculate the row annotation for each result tuple. For aggregation without group-by there will be exactly one result tuple independent of what the input is. In this case there exists a single possible SG output group (the empty tuple ()) and all input tuples are assigned to it through α. Let $\mathbf{t}_{()}$ denote this single output tuple. Recalling that all remaining tuples have multiplicity 0, we define:

Definition 33 (Aggregation Without Group-By)**.** *Consider a query* $Q \coloneqq \gamma_{f(A)}(\mathbf{R})$. *Let* $\mathbb{G}_{def}(\emptyset, \mathbf{R}) = (\mathcal{G}, \alpha)$, *and* $\mathbf{t}$ *be a range-annotated tuple with the same schema as* Q *and* g *denote the single output group in* $\mathcal{G}$. *Then*

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

Aggregation With Group-by: For aggregation with group-by 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 each 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 $\mathbf{t}$ from $\mathcal{G}(g)$ occurs with the maximal multiplicity possible ($\mathbf{R}(\mathbf{t})^{\uparrow}$) and that each tuple t encoded by $\mathbf{t}$ belongs to a separate group and (ii) that the sets of 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 attribute values and tuple's whose group membership is certain. For the latter we can compute the maximum number of groups for an output $\mathbf{t}_g$ by simplying counting the number of groups using SG values for each input tuple that overlaps with the group-by bounds of $\mathbf{t}_g$. 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 the annotations of the relevant input tuples. For this purpose, [23] did extend semirings with a duplicate elimination operator $\delta_{\mathbb{N}}$ defined as $\delta_{\mathbb{N}}(k) = 0$ if $k = 0$ and $\delta_{\mathbb{N}}(k) = 1$ otherwise.

Definition 34 (Aggregation With Group-By)**.** *Consider a query* $Q := \gamma_{G,f(A)}(\mathbf{R})$. *Let* $\mathbb{G}_{def}(\mathbf{R}, G) = (\mathcal{G}, \alpha)$. *Consider a tuple* $\mathbf{t}$ *such that* $\exists g \in \mathcal{G}$ *with* $\mathbf{t} = \mathbf{t}_g$. *Then,*

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

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

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

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 19 (Aggregation)**.** *Consider the relation shown in Fig. 5.2 which records addresses (street, street number, number of inhabitants). For the street attribute, instead of showing range annotations we mark values in red to indicate that their bound encompass as the whole domain of the street attribute. Street values* v *in black are certain, i.e.,* $v^{\downarrow} = v^{sg} = v^{\uparrow}$*. In this example, we are uncertain about particular street numbers and the number of inhabitants at certain addresses. Furthermore, several tuples may represent more than one address. Finally, we are uncertain about the street for the address represented by the second tuple. Consider the aggregation query without group-by shown in Fig. 5.2b. We are calculating the number of inhabitants. In the SGW there are 7 inhabitants (*$1 \cdot 1 + 2 \cdot 1 + 2 \cdot 2$*). As another example consider, the query shown in Fig. 5.2c. Consider the second result tuple (group State). This tuple certainly exists since the 3rd tuple in the input appears twice in every possible world and its group-by value is certain. Thus, the count for group State is at least two. Possibly, the second input tuple could also belong to this group and, thus, the count could be* 3 *(the upper bound on the aggregation result).*

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 sum(#inhab) AS pop FROM address;
```

pop	$\mathbb{N}_{AU}$
[6/7/14]	(1,1,1)

(b) Aggregation without Group-by

```
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)

(c) Aggregation with Group-by

Figure 5.2. Aggregation over AU-DBs

5.5.7 Preservation of bounds. We now demonstrate that our aggregation semantics for AU-DBs is bound-preserving. In the proof of this fact, we will make use of two auxiliary lemmas.

Lemma 11. *For* $M \in \{\mathsf{SUM}, \mathsf{MIN}, \mathsf{MAX}\}$ *we have for all* $k_1, k_2 \in \mathbb{N}_{AU}$ *and* $m \in M^3$*:* $(k_1 +_{\mathbb{N}_{AU}} k_2) \circledast_M m = k_1 \circledast_M m +_{M_I} k_2 \circledast_M m$

Proof: Consider $k = k_1 + k_2$ and $m \in \mathbb{D}_I$. Recall the definition of $\circledast_M$:

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

MIN: Consider $k \circledast_{\mathsf{MIN}} m^{\downarrow}$. $*_{\mathsf{MIN}}$ is the identify on $\mathbb{D}$ except for $k = 0$. Furthermore, $\mathbb{0}_{\mathsf{MIN}} = \infty$ and $+_{\mathsf{MIN}} = \min$. We distinguish three cases: $k^{\downarrow} = k^{\uparrow} = 0$, $k^{\downarrow} = 0 \wedge k^{\uparrow} > 0$ and $k^{\downarrow} > 0$.

If $k^{\downarrow} = k^{\uparrow} = 0$, then $(k \circledast_{\mathsf{MIN}} m)^{\downarrow} = (k_1 \circledast_M m)^{\downarrow} +_{\mathsf{MIN}} (k_2 \circledast_M m)^{\downarrow} = (k \circledast_{\mathsf{MIN}} m)^{\uparrow} = (k_1 \circledast_M m)^{\uparrow} +_{\mathsf{MIN}} (k_2 \circledast_M m)^{\uparrow} = \mathbb{0}_{\mathsf{MIN}}$.

If $k^{\downarrow} = 0 \wedge k^{\uparrow} > 0$, then

$$\begin{aligned} (k \circledast_{\mathsf{MIN}} m)^{\downarrow} &= \min(k^{\downarrow} *_{\mathsf{MIN}} m^{\downarrow}, k^{\downarrow} *_{\mathsf{MIN}} m^{\uparrow}, \\ &\qquad k^{\uparrow} *_{\mathsf{MIN}} m^{\downarrow}, k^{\uparrow} *_{\mathsf{MIN}} m^{\uparrow}) \\ &= k^{\uparrow} *_{\mathsf{MIN}} m^{\downarrow} \\ &= (k_1 + k_2)^{\uparrow} *_{\mathsf{MIN}} m^{\downarrow} \\ &= m^{\downarrow} \end{aligned}$$

Since $k^{\uparrow} > 0$, at least one of ${k_1}^{\uparrow}$ and ${k_2}^{\uparrow}$ is larger than 0. WLOG ${k_1}^{\uparrow} > 0$, then $(k_1 *_{\mathsf{MIN}} m)^{\downarrow} = m^{\downarrow}$. $(k_2 *_{\mathsf{MIN}} m)^{\downarrow}$ is either $m^{\downarrow}$ or $\mathbb{0}_{\mathsf{MIN}}$. Since min is idempotent, in either case we get:

$$
\begin{aligned}
& = (k_1 \circledast_M m)^{\uparrow} +_{\mathsf{MIN}} (k_2 \circledast_M m)^{\uparrow} \\
(k \circledast_{\mathsf{MIN}} m)^{\uparrow} &= \max(k^{\downarrow} *_{\mathsf{MIN}} m^{\downarrow}, k^{\downarrow} *_{\mathsf{MIN}} m^{\uparrow}, \\
& \qquad k^{\uparrow} *_{\mathsf{MIN}} m^{\downarrow}, k^{\uparrow} *_{\mathsf{MIN}} m^{\uparrow}) \\
&= k^{\downarrow} *_{\mathsf{MIN}} m^{\uparrow} \\
&= (k_1 + k_2)^{\downarrow} *_{\mathsf{MIN}} m^{\uparrow} \\
&= ({k_1}^{\downarrow} *_{\mathsf{MIN}} m^{\uparrow}) +_{\mathsf{MIN}} ({k_2}^{\downarrow} *_{\mathsf{MIN}} m^{\uparrow}) \\
&= (k_1 \circledast_M m)^{\uparrow} +_{\mathsf{MIN}} (k_2 \circledast_M m)^{\uparrow}
\end{aligned}
$$

If $k^{\downarrow} > 0$, then

$$
\begin{aligned}
(k \circledast_{\mathsf{MIN}} m)^{\downarrow} &= \min(k^{\downarrow} *_{\mathsf{MIN}} m^{\downarrow}, k^{\downarrow} *_{\mathsf{MIN}} m^{\uparrow}, \\
& \qquad k^{\uparrow} *_{\mathsf{MIN}} m^{\downarrow}, k^{\uparrow} *_{\mathsf{MIN}} m^{\uparrow}) \\
&= k^{\downarrow} *_{\mathsf{MIN}} m^{\downarrow} \\
&= (k_1 + k_2)^{\downarrow} *_{\mathsf{MIN}} m^{\downarrow} \\
&= m^{\downarrow}
\end{aligned}
$$

Since $k^{\downarrow} > 0$, at least one of ${k_1}^{\downarrow}$ and ${k_2}^{\downarrow}$ is larger than 0. WLOG ${k_1}^{\downarrow} > 0$, then

$(k_1 *_{\mathsf{MIN}} m)^{\downarrow} = m^{\downarrow}$. Applying the same argument as above, we get:

$$\begin{aligned} &= (k_1 \circledast_M m)^{\downarrow} +_{\mathsf{MIN}} (k_2 \circledast_M m)^{\downarrow} \\ (k \circledast_{\mathsf{MIN}} m)^{\uparrow} &= \max(k^{\downarrow} *_{\mathsf{MIN}} m^{\downarrow}, k^{\downarrow} *_{\mathsf{MIN}} m^{\uparrow}, \\ &\qquad k^{\uparrow} *_{\mathsf{MIN}} m^{\downarrow}, k^{\uparrow} *_{\mathsf{MIN}} m^{\uparrow}) \\ &= k^{\uparrow} *_{\mathsf{MIN}} m^{\uparrow} \\ &= (k_1 + k_2)^{\uparrow} *_{\mathsf{MIN}} m^{\uparrow} \end{aligned}$$

Since $k^{\uparrow} > 0$, at least one of ${k_1}^{\uparrow}$ and ${k_2}^{\uparrow}$ is larger than 0. WLOG ${k_1}^{\uparrow} > 0$, then $(k_1 *_{\mathsf{MIN}} m)^{\uparrow} = m^{\uparrow}$. Applying the same argument as above, we get:

$$= (k_1 \circledast_M m)^{\uparrow} +_{\mathsf{MIN}} (k_2 \circledast_M m)^{\uparrow}$$

MAX: The proof is analog to the proof for MIN.

SUM: Consider $k \circledast_{\mathsf{SUM}} m^{\downarrow}$. We first address that case $m^{\downarrow} < 0$.

$$(k \circledast_{\mathsf{SUM}} m)^{\downarrow} = \min(k^{\downarrow} \cdot m^{\downarrow}, k^{\downarrow} \cdot m^{\uparrow}, k^{\uparrow} \cdot m^{\downarrow}, k^{\uparrow} \cdot m^{\uparrow})$$

Since $m^{\downarrow} < 0$:

$$\begin{aligned} &= k^{\uparrow} \cdot m^{\downarrow} \\ &= (k_1 + k_2)^{\uparrow} \cdot m^{\downarrow} \\ &= ({k_1}^{\uparrow} + {k_2}^{\uparrow}) \cdot m^{\downarrow} \\ &= ({k_1}^{\uparrow} \cdot m^{\downarrow}) + ({k_2}^{\uparrow} \cdot m^{\downarrow}) \\ &= (k_1 \circledast_{\mathsf{SUM}} m)^{\downarrow} +_{\mathsf{SUM}} (k_2 \circledast_{\mathsf{SUM}} m)^{\downarrow} \end{aligned}$$

Now consider the case $m^{\downarrow} > 0$.

$$\begin{aligned}
&(k \circledast_{\mathsf{SUM}} m)^{\downarrow} \\
=&k^{\downarrow} \cdot m^{\downarrow} \\
=&(k_1 + k_2)^{\downarrow} \cdot m^{\downarrow} \\
=&({k_1}^{\downarrow} \cdot m^{\downarrow}) + ({k_2}^{\downarrow} \cdot m^{\downarrow}) \\
=&(k_1 \circledast_{\mathsf{SUM}} m)^{\downarrow} +_{\mathsf{SUM}} (k_2 \circledast_{\mathsf{SUM}} m)^{\downarrow}
\end{aligned}$$

We now prove that $k \circledast_{\mathsf{SUM}} m^{\uparrow} = (k_1 \circledast_{\mathsf{SUM}} m)^{\uparrow} +_{\mathsf{SUM}} (k_2 \circledast_{\mathsf{SUM}} m)^{\uparrow}$.

$$(k \circledast_{\mathsf{SUM}} m)^{\uparrow} = \max(k^{\downarrow} \cdot m^{\downarrow}, k^{\downarrow} \cdot m^{\uparrow}, k^{\uparrow} \cdot m^{\downarrow}, k^{\uparrow} \cdot m^{\uparrow})$$

First consider $m^{\uparrow} < 0$.

$$\begin{aligned}
&(k \circledast_{\mathsf{SUM}} m)^{\uparrow} \\
=&k^{\downarrow} \cdot m^{\uparrow} \\
=&(k_1 + k_2)^{\downarrow} \cdot m^{\uparrow} \\
=&({k_1}^{\downarrow} \cdot m^{\uparrow}) + ({k_2}^{\downarrow} \cdot m^{\uparrow}) \\
=&(k_1 \circledast_{\mathsf{SUM}} m)^{\uparrow} +_{\mathsf{SUM}} (k_2 \circledast_{\mathsf{SUM}} m)^{\uparrow}
\end{aligned}$$

Now consider $m^{\uparrow} \geq 0$.

$$\begin{aligned}
&(k \circledast_{\mathsf{SUM}} m)^{\uparrow} \\
=&k^{\uparrow} \cdot m^{\uparrow} \\
=&(k_1 + k_2)^{\uparrow} \cdot m^{\uparrow} \\
=&({k_1}^{\uparrow} \cdot m^{\uparrow}) + ({k_2}^{\uparrow} \cdot m^{\uparrow}) \\
=&(k_1 \circledast_{\mathsf{SUM}} m)^{\uparrow} +_{\mathsf{SUM}} (k_2 \circledast_{\mathsf{SUM}} m)^{\uparrow}
\end{aligned}$$

In addition we will prove that for $M \in \{\mathsf{SUM}, \mathsf{MIN}, \mathsf{MAX}\}$ and for all $m_1, m_2, m_3, m_4 \in M$ such that $m_1 \leq m_2$ and $m_3 \leq m_4$ (here $<$ is the order of $\mathbb{D}$), we have

$m_1 +_M m_2 \leq m_3 +_M m_4$. This is implies as a special case $m +_M m' \leq m +_M \mathbb{0}_M$ for $m, m' \in M$ and $m' \leq \mathbb{0}_M$.

Lemma 12. *Let* $M \in \{\mathsf{SUM}, \mathsf{MIN}, \mathsf{MAX}\}$. $\forall m_1, m_2, m_3, m_4 \in M$:

$$m_1 \leq m_2 \wedge m_3 \leq m_4 \Rightarrow m_1 +_M m_2 \leq m_3 +_M m_4$$

Proof: MIN: WLOG assume that $m_1 \leq m_2$ and $m_3 \leq m_4$ (the other cases are analog).

$$\min(m_1, m_2) = m_1 \leq m_3 = \min(m_3, m_4)$$

SUM: Since addition preserves inequalities, we get

$$m_1 +_{\mathsf{SUM}} m_2 = m_1 + m_2 \leq m_3 + m_4 = m_3 +_{\mathsf{SUM}} m_4$$

MAX: WLOG assume that $m_1 \leq m_2$ and $m_3 \leq m_4$ (the other cases are analog).

$$\max(m_1, m_2) = m_2 \leq m_4 = \max(m_3, m_4)$$

Having proven this lemma, we are ready to proof that aggregation preserves bounds.

Theorem 13. *Let* $Q := \gamma_{G,f(A)}(R)$ *or* $Q := \gamma_{f(A)}(R)$ *and* $\mathcal{R}$ *be an incomplete* $\mathcal{K}$*-relation that is bound by an* $\mathcal{K}_{AU}$*-relation* $\mathbf{R}$*. Then* $Q(\mathbf{R})$ *bounds* $Q(\mathcal{R})$*.*

Proof: We first consider the case of aggregation with group-by, i.e., $Q := \gamma_{G,f(A)}(R)$. Let $\mathbf{t}_g$ be the output tuple corresponding to $g \in \mathcal{G}$. Abusing notation, we will understand $\alpha(\mathbf{t}) = \mathbf{t}_g$ to mean $\alpha(\mathbf{t}) = g$. Consider one possible world $R \in \mathcal{R}$ and let $\mathcal{TM}_R$ be a tuple matching based on which $\mathbf{R}$ bounds R. We will prove the existence of a tuple matching $\mathcal{TM}_Q$ between $Q(R)$ and $Q(\mathbf{R})$ and demonstrate that $Q(\mathbf{R})$ bounds $Q(R)$ based on this tuple matching. For that we first prove that for each result tuple

$t \in Q(D)$ the set $\mathbf{S}_t = \{\mathbf{t} \mid t \sqsubseteq \mathbf{t} \wedge Q(\mathbf{R})(\mathbf{t}) \neq \mathbb{0}_{\mathbb{N}_{AU}}\}$ is non-empty. Intuitively, the set $\mathbf{S}_t$ contains potential candidates for which we can set $\mathcal{TM}_Q(\mathbf{t}, t)$ to a non-zero value, because only tuples that bound t can be associated with t in a tuple matching. Because, for $\mathcal{TM}_Q$ to be a tuple matching we have to assign that annotation $Q(R)(t)$ to a set of tuples such that $\sum_{\mathbf{t}} \mathcal{TM}_Q(\mathbf{t}, t) = Q(R)(t)$. Note that since each aggregation result in $Q(R)$ is annotated with 1, this boils down to assigning t to exactly one $\mathbf{t} \in \mathbf{S}_t$.

Afterwards, we show that for each $\mathbf{t} \in Q(\mathbf{R})$ it is possible to set $\mathcal{TM}_Q(\mathbf{t}, t)$ for all $t \in Q(D)$ for which $t \sqsubseteq \mathbf{t}$ such that (1) $Q(\mathbf{R})(\mathbf{t})^{\downarrow} \preceq_{\mathbb{N}} \sum_{t:t\sqsubseteq\mathbf{t}} Q(D)(t) \preceq_{\mathbb{N}} Q(\mathbf{R})(\mathbf{t})^{\uparrow}$ and (2) for all $t \in Q(D)$ we have $\sum_{\mathbf{t}:t\sqsubseteq\mathbf{t}} \mathcal{TM}_Q(\mathbf{t}, t) = Q(D)(t)$. The consequence of these two steps and Ex. 16 is that $Q(\mathbf{R})$ bounds $Q(R)$ based on $\mathcal{TM}_Q$.

We will make use of the following notation. Let $\mathcal{G} = \{t.G \mid R(t) \neq 0\}$, i.e., the set of groups in the possible world R. For a group $g \in \mathcal{G}$, we define

$$T_g = \{t \mid t.G = g \wedge R(t) \neq 0\}$$
$$\mathbf{S}_g = \{\mathbf{t} \mid \exists t : R(t) \neq 0 \wedge t.G = g \wedge \mathcal{TM}_R(\mathbf{t}, t) \neq 0\}$$
$$\mathbf{O}_g = \{\mathbf{t} \mid \mathbf{t} \in Q(\mathbf{R}) \wedge \exists \mathbf{t}' \in \mathbf{S}_g : \alpha(\mathbf{t}') = \mathbf{t}\}$$

Furthermore, for any $\mathbf{o} \in \mathbf{O}_g$, we define

$$\mathbf{N}_{\mathbf{o}} = \{\mathbf{t} \mid \alpha(\mathbf{t}) = \mathbf{o} \wedge \mathbf{t} \notin \mathbf{S}_g\}$$

Consider a group $g \in \mathcal{G}$ and let t_g denote the result tuple in $Q(R)$ corresponding to g. There is at least on $t \in T_g$, otherwise g would not be in $\mathcal{G}$. Consider an arbitrary $\mathbf{o} \in \mathbf{O}_g$. At least one such $\mathbf{o}$ exists since $\mathcal{TM}(\mathbf{t}, t) \neq 0$ for one or more $\mathbf{t}$ with $\mathbf{R}(\mathbf{t}) \neq \mathbb{0}_{\mathbb{N}_{AU}}$ and $\mathbf{t}$ has to be associated with at least one output $\mathbf{o}$ by α. Let $\mathbf{S}_{\mathbf{o}} = \{\mathbf{t} \mid \mathbf{t} \in \mathbf{S}_g \wedge \alpha(\mathbf{t}) = \mathbf{o}\}$. We will show that $t_g \sqsubseteq \mathbf{o}$.

$t_g.G \sqsubseteq \mathbf{o}.G$: For all $\mathbf{t} \in \mathbf{S_o}$ we know that $g \sqsubseteq \mathbf{t}.G$ because for $\mathbf{t}$ to be in $\mathbf{S}_g$ it has to be the case there exists t with $t.G = g$ such that $\mathcal{TM}_{\leftrightarrow}(\mathbf{t}, t) \neq 0$. This implies $t \sqsubseteq \mathbf{t}$ which in turn implies $g = t.G \sqsubseteq \mathbf{t}$. Since $\alpha(\mathbf{t}) = \mathbf{o}$ and since by Def. 31 the range annotations of $\mathbf{o}.G$ are defined as the union of the range annotations of all $\mathbf{t} \in \mathbf{S_o}$ (and any other $\mathbf{t}$ with $\alpha(\mathbf{t})$). Thus, $g \sqsubseteq \mathbf{o}.G$.

$t_g.f(A) \sqsubseteq \mathbf{o}.f(A)$: Based on the definition of aggregation over $\mathbb{N}$-relation, we have:

$$t_g.f(A) = \sum_{t \in T_g} R(t) *_M t.A \tag{5.16}$$

Let $f_g = t_g.f(A)$. Note that based on Def. 32, $\mathbf{o}.f(A)$ is calculated over all tuples from $\mathbf{S}_g$ and $\mathbf{N_o}$. Observe that if $\mathbf{t} \in \mathbf{N_o}$ then either $\textsc{ug}(G, \mathbf{R}, \mathbf{t})$ or $\mathbf{R}(\mathbf{t})^{\downarrow} = 0$. To see why this has to be the case consider that if $\mathbf{t}.G$ is certain and $\mathbf{t}$ exists in every possible world ($\mathbf{R}(\mathbf{t})^{\downarrow} > 0$ then for $\mathcal{TM}_R$ to be a tuple matching based on which $\mathbf{R}$ bounds R there has to exist some $t \in T_g$ for which $\mathcal{TM}_R(\mathbf{t}, t) \neq 0$ which would lead to the contradiction $\mathbf{t} \in \mathbf{S}_g$. Define

$$\mathbf{S}_g^{uncertain} = \{\mathbf{t} \mid \mathbf{t} \in \mathbf{S}_g \wedge \textsc{ug}(G, \mathbf{R}, \mathbf{t})\}$$
$$\mathbf{S}_g^{certain} = \mathbf{S}_g - \mathbf{S}_g^{uncertain}$$

We have to show that $\mathbf{o}.f(A)^{\downarrow} \leq f_g \leq \mathbf{o}.f(A)^{\uparrow}$.

$\mathbf{o}.f(A)^{\downarrow} \leq f_g$: Substituting Def. 32 we get for $\mathbf{o}.f(A)^{\downarrow}$:

$$\mathbf{o}.f(A)^{\downarrow} = \sum_{\mathbf{t}\in\mathbf{S}_g^{certain}} (\mathbf{R}(\mathbf{t}) \circledast_M \mathbf{t}.A)^{\downarrow} +_M \sum_{\mathbf{t}\in\mathbf{S}_g^{uncertain}} \min((\mathbf{R}(\mathbf{t}) \circledast_M \mathbf{t}.A)^{\downarrow}, \mathbb{0}_M) +_M \sum_{\mathbf{t}\in\mathbf{N}_g} \min((\mathbf{R}(\mathbf{t}) \circledast_M \mathbf{t}.A)^{\downarrow}, \mathbb{0}_M) \quad (5.17)$$

Using Lem. 12, we know that

$$\sum_{\mathbf{t}\in\mathbf{N}_g} \min((\mathbf{R}(\mathbf{t}) \circledast_M \mathbf{t}.A)^{\downarrow}, \mathbb{0}_M) \leq \sum_{\mathbf{t}\in\mathbf{N}_g} \mathbb{0}_M = \mathbb{0}_M$$

Thus, we can bound Equation (5.17) from above:

$$\leq \sum_{\mathbf{t}\in\mathbf{S}_g^{certain}} (\mathbf{R}(\mathbf{t}) \circledast_M \mathbf{t}.A)^{\downarrow} +_M \sum_{\mathbf{t}\in\mathbf{S}_g^{uncertain}} \min((\mathbf{R}(\mathbf{t}) \circledast_M \mathbf{t}.A)^{\downarrow}, \mathbb{0}_M) \quad (5.18)$$

We next will relate Equation (5.18) to f_g through $\mathcal{TM}_R$. Towards this goal for any $\mathbf{t} \in \mathbf{S}_g$ we define $T_{\mathbf{t}} = \{t \mid \mathcal{TM}_R(\mathbf{t}, t) > 0\}$. We know that for any $\mathbf{t} \in \mathbf{S}_g$, we have $\mathbf{R}(\mathbf{t})^{\downarrow} \leq \sum_{t\in T_{\mathbf{t}}} \mathcal{TM}_R(\mathbf{t}, t)\mathbf{R}(\mathbf{t})^{\uparrow}$ because $\mathcal{TM}_R$ is a tuple matching based on which $\mathbf{R}$ bounds R. Consider the sum in Equation (5.18) which ranges over $\mathbf{S}_g^{certain}$ first. Since $\mathbf{R}(\mathbf{t})^{\downarrow} \leq \sum_{t\in T_{\mathbf{t}}} \mathcal{TM}_R(\mathbf{t}, t) \leq \mathbf{R}(\mathbf{t})^{\uparrow}$, based on Thm. 12 we have that $(\sum_{t\in T_{\mathbf{t}}} \mathcal{TM}_R(\mathbf{t}, t)) *_M \mathbf{t}.A^{\downarrow}$ is bound from below by $(\mathbf{R}(\mathbf{t}) \circledast_M \mathbf{t}.A)^{\downarrow}$. Thus,

$$\sum_{\mathbf{t}\in\mathbf{S}_g^{certain}} (\mathbf{R}(\mathbf{t}) \circledast_M \mathbf{t}.A)^{\downarrow}$$

$$\leq \sum_{\mathbf{t}\in\mathbf{S}_g^{certain}} \left(\sum_{t:t\in T_{\mathbf{t}}} \mathcal{TM}_R(\mathbf{t},t) \right) *_M \mathbf{t}.A^{\downarrow} \tag{5.19}$$

Note that for any $\mathbf{t} \in \mathbf{S}_g^{certain}$, $t \notin T_g \Rightarrow t \not\sqsubseteq \mathbf{t}$ since $\mathbf{t}.G$ is certain.

$$= \sum_{\mathbf{t}\in\mathbf{S}_g^{certain}} \left(\sum_{t\in T_g} \mathcal{TM}_R(\mathbf{t},t) \right) *_M \mathbf{t}.A^{\downarrow}$$

For any semimodule and thus also every $\mathbb{N}$-semimodule the law $(k_1 + k_2) \circledast_M m = k_1 *_M m +_M k_2 *_M m$ holds. Applying this law we can factor out the inner sum:

$$= \sum_{\mathbf{t}\in\mathbf{S}_g^{certain}} \sum_{t\in T_g} \mathcal{TM}_R(\mathbf{t},t) *_M \mathbf{t}.A^{\downarrow}$$

Using commutativity and associativity of $+_M$, we commute the two sums:

$$= \sum_{t\in T_g} \sum_{\mathbf{t}\in\mathbf{S}_g^{certain}} \mathcal{TM}_R(\mathbf{t},t) *_M \mathbf{t}.A^{\downarrow}$$

Since, $t \sqsubseteq \mathbf{t}$ for any $t \in T_{\mathbf{t}}$, we have $\mathbf{t}.A < t.A$ from which follows that:

$$< \sum_{t\in T_g} \sum_{\mathbf{t}\in\mathbf{S}_g^{certain}} \mathcal{TM}_R(\mathbf{t},t) *_M t.A$$

To recap so far we have shown that

$$\sum_{\mathbf{t}\in\mathbf{S}_g^{certain}} (\mathbf{R}(\mathbf{t}) \circledast_M \mathbf{t}.A)^{\downarrow}$$

$$< \sum_{t\in T_g} \sum_{\mathbf{t}\in\mathbf{S}_g^{certain}} \mathcal{TM}_R(\mathbf{t}, t) *_M t.A \tag{5.20}$$

Next we will bound the second sum from Equation (5.18) which ranges over $\mathbf{S}_g^{uncertain}$ in a similar fashion and then show that f_g is lower bound by the sum of these bounds. For $\mathbf{t} \in \mathbf{S}_g^{uncertain}$ since $\mathbf{t}.G$ is uncertain, some $t \in T_\mathbf{t}$ may belong to a group $g' \neq g$. We will have to treat this case differently in the following. For that we define $T_\mathbf{t}^+ = \{t \mid t \in T_\mathbf{t} \wedge t.G = g\}$ and $T_\mathbf{t}^- = \{t \mid t \in T_\mathbf{t} \wedge t.G \neq g\}$. Let $k_\mathbf{t}^+ = \sum_{t\in T_\mathbf{t}^+} \mathcal{TM}_R(\mathbf{t}, t)$ and $k_\mathbf{t}^- = \sum_{t\in T_\mathbf{t}^-} \mathcal{TM}_R(\mathbf{t}, t)$. Using the same argument as for Equation (5.19), we get:

$$\sum_{\mathbf{t}\in\mathbf{S}_g^{uncertain}} \min((\mathbf{R}(\mathbf{t}) \circledast_M \mathbf{t}.A)^{\downarrow}, \mathbb{0}_M)$$

$$\leq \sum_{\mathbf{t}\in\mathbf{S}_g^{uncertain}} \min\left(\left(k_\mathbf{t}^+ + k_\mathbf{t}^-\right) \circledast_M \mathbf{t}.A^{\downarrow}, \mathbb{0}_M\right)$$

We consider two cases: (i) $\mathbf{R}(\mathbf{t})^{\downarrow} \leq k_\mathbf{t}^+ \leq \mathbf{R}(\mathbf{t})^{\uparrow}$ and (ii) $k_\mathbf{t}^+ < \mathbf{R}(\mathbf{t})^{\downarrow} \leq \mathbf{R}(\mathbf{t})^{\uparrow}$. For case (i) first consider that based on Lem. 12, we have

$$\leq \sum_{\mathbf{t}\in\mathbf{S}_g^{uncertain}} \mathbf{R}(\mathbf{t}) \circledast_M \mathbf{t}.A^{\downarrow}$$

From (i) follows that $\sum_{t\in T_\mathbf{t}} \mathcal{TM}_R(\mathbf{t}, t) *_M \mathbf{t}^{\downarrow}$ is bound from below by $\mathbf{R}(\mathbf{t}) \circledast_M \mathbf{t}.A^{\downarrow}$ for any $\mathbf{t} \in \mathbf{S}_g^{uncertain}$. Applying the same steps as for $\mathbf{S}_g^{certain}$, we get:

$$\leq \sum_{t \in T_g} \sum_{\mathbf{t} \in \mathbf{S}_g^{uncertain}} \mathcal{TM}_R(\mathbf{t}, t) *_M t.A \tag{5.21}$$

Now we have to prove the same for (ii), i.e., when $k_{\mathbf{t}}^+ < \mathbf{R}(\mathbf{t})^{\downarrow}$. Let $min_A = \min(\{t.A \mid t \in T_{\mathbf{t}}^+\})$. We now prove for each $M \in \{\mathsf{SUM}, \mathsf{MIN}, \mathsf{MAX}\}$ that under assumption (ii) for any $\mathbf{t} \in \mathbf{S}_g^{uncertain}$ the following holds:

$$\begin{aligned} &\min((k_{\mathbf{t}}^+ + k_{\mathbf{t}}^-) *_M \mathbf{t}.A^{\downarrow}, \mathbb{0}_M) \\ \leq & k_{\mathbf{t}}^+ *_M min_a \\ = & \left(\sum_{t \in T_{\mathbf{t}}^+} \mathcal{TM}_R(\mathbf{t}, t) \right) *_M min_a \end{aligned} \tag{5.22}$$

SUM: Recall that $*_{\mathsf{SUM}}$ is multiplication and $\mathbb{0}_{\mathsf{SUM}} = 0$. We distinguish two cases: $min_a \leq 0$ and $\min_a > 0$. If $min_a \leq 0$, since $\mathbf{t}.A^{\downarrow} \leq min_a$, it follows that $\min((k_{\mathbf{t}}^+ + k_{\mathbf{t}}^-) *_{\mathsf{SUM}} \mathbf{t}.A^{\downarrow}, \mathbb{0}_M) = \min((k_{\mathbf{t}}^+ + k_{\mathbf{t}}^-) \cdot \mathbf{t}.A^{\downarrow}, 0) = (k_{\mathbf{t}}^+ + k_{\mathbf{t}}^-) \cdot \mathbf{t}.A^{\downarrow} < k_{\mathbf{t}}^+ \cdot \mathbf{t}.A^{\downarrow} < k_{\mathbf{t}}^+ \cdot min_a$. If $min_a > 0$, then $k_{\mathbf{t}}^+ \cdot min_a \geq 0$ and since $\min(m, 0) \leq 0$ for any m, we get $\min((k_{\mathbf{t}}^+ + k_{\mathbf{t}}^-) *_{\mathsf{SUM}} \mathbf{t}.A^{\downarrow}, 0) \leq k_{\mathbf{t}}^+ \cdot min_a$.

MIN: Since $*_{\mathsf{MIN}}$ is the identity on MIN except for when $k = 0$ and because $\mathbb{0}_{\mathsf{MAX}} = \infty$, we get $\min((k_{\mathbf{t}}^+ + k_{\mathbf{t}}^-) *_{\mathsf{MIN}} \mathbf{t}.A^{\downarrow}, \mathbb{0}_M) = (k_{\mathbf{t}}^+ + k_{\mathbf{t}}^-) *_{\mathsf{MIN}} \mathbf{t}.A^{\downarrow}$. Distinguish two cases. If $k_{\mathbf{t}}^+ = 0$, then $k_{\mathbf{t}}^+ *_{\mathsf{MIN}} \mathbf{t}.A^{\downarrow} = \infty > m$ for any m including $(k_{\mathbf{t}}^+ + k_{\mathbf{t}}^-) *_{\mathsf{MIN}} \mathbf{t}.A^{\downarrow}$. Otherwise, since $*_{\mathsf{MIN}}$ is the identify on MIN if $k \neq 0$, we have $(k_{\mathbf{t}}^+ + k_{\mathbf{t}}^-) *_{\mathsf{MIN}} \mathbf{t}.A^{\downarrow} = k_{\mathbf{t}}^+ *_{\mathsf{MIN}} \mathbf{t}.A^{\downarrow}$.

MAX: Since $\mathbb{0}_{\mathsf{MAX}} = -\infty$, we get $\min((k_{\mathbf{t}}^+ + k_{\mathbf{t}}^-) *_{\mathsf{MAX}} \mathbf{t}.A^{\downarrow}, \mathbb{0}_M) = -\infty \leq k_{\mathbf{t}}^+ *_{\mathsf{MAX}} min_a$.

Using Equation (5.22) proven above, we can apply the same steps as in the proof of Equation (5.20) to deduce that:

$$
\begin{aligned}
&\sum_{\mathbf{t}\in\mathbf{S}_g^{uncertain}} \min(\mathbf{R}(\mathbf{t}) \circledast_M \mathbf{t}.A^{\downarrow}, \mathbb{0}_M) \\
\leq &\sum_{\mathbf{t}\in\mathbf{S}_g^{uncertain}} \left(\sum_{t\in T_{\mathbf{t}}^{+}} \mathcal{TM}_R(\mathbf{t},t) \right) \circledast_M min_a \\
= &\sum_{\mathbf{t}\in\mathbf{S}_g^{uncertain}} \left(\sum_{t\in T_g} \mathcal{TM}_R(\mathbf{t},t) \right) \circledast_M min_a \\
= &\sum_{\mathbf{t}\in\mathbf{S}_g^{uncertain}} \sum_{t\in T_g} \mathcal{TM}_R(\mathbf{t},t) \circledast_M min_a \\
\leq &\sum_{\mathbf{t}\in\mathbf{S}_g^{uncertain}} \sum_{t\in T_g} \mathcal{TM}_R(\mathbf{t},t) \circledast_M t.A \\
= &\sum_{t\in T_g} \sum_{\mathbf{t}\in\mathbf{S}_g^{uncertain}} \mathcal{TM}_R(\mathbf{t},t) \circledast_M t.A
\end{aligned} \tag{5.23}
$$

Combining Equation (5.20) with Equation (5.21) and Equation (5.23) and using Lem. 12 we get

$$
\begin{aligned}
\mathbf{o}.f(A)^{\downarrow} \leq &\sum_{t\in T_g} \sum_{\mathbf{t}\in\mathbf{S}_g^{certain}} \mathcal{TM}_R(\mathbf{t},t) *_M t.A \\
&+_M \sum_{t\in T_g} \sum_{\mathbf{t}\in\mathbf{S}_g^{uncertain}} \mathcal{TM}_R(\mathbf{t},t) *_M t.A \\
= &\sum_{t\in T_g} \sum_{\mathbf{t}\in\mathbf{S}_g} \mathcal{TM}_R(\mathbf{t},t) *_M t.A \\
= &f_g
\end{aligned}
$$

$\underline{\mathbf{o}.f(A)^{\uparrow} \geq f_g}$: We still need to prove that $\mathbf{o}.f(A)^{\uparrow} \geq f_g$.

$$\mathbf{o}.f(A)^{\uparrow} = \sum_{\mathbf{t} \in \mathbf{S}_g^{certain}} (\mathbf{R}(\mathbf{o}) \circledast_M \mathbf{t}.A)^{\uparrow} +_M \sum_{\mathbf{t} \in \mathbf{S}_g^{uncertain}} \max((\mathbf{R}(\mathbf{o}) \circledast_M \mathbf{t}.A)^{\uparrow}, \mathbb{0}_M) +_M \sum_{\mathbf{t} \in \mathbf{N}_g} \max((\mathbf{R}(\mathbf{o}) \circledast_M \mathbf{t}.A)^{\uparrow}, \mathbb{0}_M)$$

This prove is analog to the prove for $\mathbf{o}.f(A)^{\downarrow} \leq f_g$ except that it is always the case that $k_{\mathbf{t}}^{+} \leq \mathbf{R}(\mathbf{t})^{\uparrow}$ which simplifies the case for $\mathbf{S}_g^{uncertain}$.

$Q(\mathbf{R})(\mathbf{o})^{\downarrow} \leq \sum_{t \sqsubseteq \mathbf{o}} \mathcal{TM}_Q(\mathbf{o}, t) \leq Q(\mathbf{R})(\mathbf{o})^{\uparrow}$: So far we have established that for any $\mathbf{o} \in \mathbf{O}_g$ we have $t_g \sqsubseteq \mathbf{o}$. For that follows that when constructing a tuple matching $\mathcal{TM}_Q$ based on which $Q(\mathbf{t})$ bounds $Q(R)$ we can associate t_g with any subset of $\mathbf{O}_g$. It remains to be shown that we can find such a tuple matching such that $Q(\mathbf{R})(\mathbf{o})^{\downarrow} \leq \sum_{t \sqsubseteq \mathbf{o}} \mathcal{TM}_Q(\mathbf{o}, t) \leq Q(\mathbf{R})(\mathbf{o})^{\uparrow}$. Since each aggregation result in $Q(R)$ appears exactly once, this boils down to proving that $Q(\mathbf{R})(\mathbf{o})^{\downarrow} \leq |\{t \mid \mathcal{TM}_Q(\mathbf{o}, t) \neq 0\}| \leq Q(\mathbf{R})(\mathbf{o})^{\uparrow}$. We will make use of the following notation:

$$\mathbf{Tup}_R = \{t \mid R(t) \neq 0\}$$

$$\mathbf{Tup}_{\mathbf{R}} = \{\mathbf{t} \mid \mathbf{R}(\mathbf{t}) \neq 0\}$$

$$\mathbf{Tup}_{output} = \{\mathbf{o} \mid Q(\mathbf{R})(\mathbf{o}) \neq 0\}$$

Recall that $\mathcal{G}$ denotes the set of groups in R. For the construction of $\mathcal{TM}_Q$ we will use a mapping $gcover : \mathcal{G} \rightarrow \mathbf{Tup}_R \times \mathbf{Tup}_{\mathbf{R}} \times \mathbf{Tup}_{output}$ such that for any $g \in \mathcal{G}$ for which $gcover(g) = (t, \mathbf{t}, \mathbf{o})$ the following conditions hold:

$$t \in T_g \qquad \mathcal{TM}_R(\mathbf{t}, t) \neq 0 \qquad \alpha(\mathbf{t}) = \mathbf{o}$$

We will refer to such a mapping as a group cover. The purpose of a group cover *gcover* is to assign each group g in the possible world to an output $\mathbf{o}$ which represents this group and to justify this assignment through an input $\mathbf{t}$ that is associated by TM_R with at least one tuple from group g and is assigned by the grouping strategy to the AU-DB output tuple $\mathbf{o}$. We will first prove that at least one group cover exists and then prove that a group cover induces a tuple matching $\mathcal{TM}_Q$ for which the condition we want to prove ($Q(\mathbf{R})(\mathbf{o})^{\downarrow} \leq \sum_{t \sqsubseteq \mathbf{o}} \mathcal{TM}_Q(\mathbf{o}, t) \leq Q(\mathbf{R})(\mathbf{o})^{\uparrow}$) holds for all $\mathbf{o} \in \mathbf{Tup}_{output}$.

Group cover exists: To prove the existence of a group cover, we will show how to construct such a group cover for any aggregation query Q, input R that is bound by a $\mathbb{N}_{AU}$-relation $\mathbf{R}$. Consider a group $g \in \mathbf{G}$ and pick an arbitrary tuple $t \in T_g$ and $\mathbf{t} \in \mathbf{S}_g$ such that $\mathcal{TM}_R(\mathbf{t}, t) \neq$. At least one such t has to exist for g to be a group in the result of $Q(R)$. Furthermore, since $\mathcal{TM}_R$ is a tuple matching based on which $\mathbf{R}$ bounds R, there has to exist at least one such $\mathbf{t}$. Now recall that α associates each tuple $\mathbf{t}$ for which $\mathbf{R}(\mathbf{t}) \neq \mathbb{0}_{\mathbb{N}_{AU}}$ with one output in $Q(\mathbf{R})$. WLOG let $o = \alpha(\mathbf{R})$. We set $gcover(g) = (t, \mathbf{t}, \mathbf{o})$. By construction *gcover* is a group cover.

$Q(\mathbf{R})(\mathbf{o})^{\downarrow} \leq \sum_{t \sqsubseteq \mathbf{o}} \mathcal{TM}_Q(\mathbf{o}, t) \leq Q(\mathbf{R})(\mathbf{o})^{\uparrow}$: It remains to be shown that it is possible to construct a tuple matching $\mathcal{TM}_Q$ such that for any $\mathbf{o}$ in $Q(\mathbf{R})$ we have

$$Q(\mathbf{R})(\mathbf{o})^{\downarrow} \leq \sum_{t \sqsubseteq \mathbf{o}} \mathcal{TM}_Q(\mathbf{o}, t) \leq Q(\mathbf{R})(\mathbf{o})^{\uparrow}$$

which implies that $Q(\mathbf{R})$ bounds $Q(R)$ based on $\mathcal{TM}_Q$. Since aggregation returns a single result tuple t_g for each group g, we know that that $Q(R)(t_g) = 1$. Using *gcover*, we construct $\mathcal{TM}_Q$ as shown below:

$$\mathcal{TM}_Q(t_g, \mathbf{o}) = \begin{cases} 1 & \text{if } \exists t, \mathbf{t} : gcover(g) = (t, \mathbf{t}, \mathbf{o}) \\ 0 & \text{otherwise} \end{cases}$$

Obviously, $\sum_{\mathbf{t}} \mathcal{TM}_Q(t_g, \mathbf{t}) = 1 = Q(R(t_g))$. Thus, $\mathcal{TM}_Q$ is a tuple matching. It remains to be shown that for each $\mathbf{o}$ in $Q(\mathbf{R})$ we have $Q(\mathbf{R})(\mathbf{o})^{\downarrow} \leq \sum_t \mathcal{TM}_Q(t, \mathbf{R}) \leq Q(\mathbf{R})(\mathbf{o})^{\uparrow}$. Observe that based on how we have constructed *gcover*, the following holds for any such $\mathbf{o}$:

$$\sum_{t_g} \mathcal{TM}_Q(\mathbf{o}, t_g) = \sum_{g \in \mathbf{G}: \exists t, \mathbf{t}: gcover(g) = (t, \mathbf{t}, \mathbf{o})} 1$$

For any group cover if $gcover(g) = (t, \mathbf{t}, \mathbf{o})$ then $\mathcal{TM}_R(\mathbf{t}, t) \neq 0$. Then,

$$\leq \sum_{g \in \mathbf{G}: \exists t, \mathbf{t}: gcover(g) = (t, \mathbf{t}, \mathbf{o})} \mathbf{R}(\mathbf{t})^{\uparrow}$$

Since α may assign to $\mathbf{o}$ additional tuples which do not co-occur with $\mathbf{o}$ in *gcover*, i.e., where $\neg \exists g, \mathbf{t}, tup : gcover(g) = (t, \mathbf{t}, \mathbf{o})$, we get:

$$\leq \sum_{\mathbf{t}: \alpha(\mathbf{t}) = \mathbf{o}} \mathbf{R}(\mathbf{t})^{\uparrow} = \qquad\qquad Q(\mathbf{R})(\mathbf{o})^{\uparrow}$$

It remains to be shown that $Q(\mathbf{R})(\mathbf{o})^{\downarrow} \leq \sum_{t \sqsubseteq \mathbf{o}} \mathcal{TM}_Q(\mathbf{o}, t)$. From the construction of $\mathcal{TM}_Q$ follows that:

$$\sum_{t \sqsubseteq \mathbf{o}} \mathcal{TM}_Q(\mathbf{o}, t)$$
$$= \sum_{g \in \mathbf{G}: \exists t, \mathbf{t}: gcover(g) = (t, \mathbf{t}, \mathbf{o})} 1$$

Based on Def. 34,

$$Q(\mathbf{R})(\mathbf{o})^{\downarrow} = \delta_{\mathbb{N}} \left(\sum_{\mathbf{t}': \alpha(\mathbf{t}') = g} \mathbf{R}(\mathbf{t}')^{\downarrow} \right) \tag{5.24}$$

First consider the case where the set $\{\mathbf{t} \mid \alpha(\mathbf{t}) = \mathbf{o}\}$ is empty. It follows that $Q(\mathbf{R})(\mathbf{o})^{\downarrow} = 0$ and the claim trivially holds.

If the set is non empty, then $\sum_{t \sqsubseteq \mathbf{o}} \mathcal{TM}_Q(\mathbf{o}, t) \geq 1$. Also

$$\delta_{\mathbb{N}} \left(\sum_{\mathbf{t}': \alpha(\mathbf{t}') = \mathbf{o}} \mathbf{R}(\mathbf{t}')^{\downarrow} \right) \leq 1$$

, because for any $k \in \mathbb{N}$, $\delta_{\mathbb{N}}(k) \leq 1$ if $k \neq 0$. Thus,

$$\leq 1 \qquad\qquad \leq \sum_{t \sqsubseteq \mathbf{o}} \mathcal{TM}_Q(\mathbf{o}, t)$$

Thus, we have shown that $Q(\mathbf{R})(\mathbf{o})^{\downarrow} \leq \sum_{t \sqsubseteq \mathbf{o}}$ which together with $\sum_{t \sqsubseteq \mathbf{o}} \leq Q(\mathbf{R})(\mathbf{o})^{\uparrow}$ and the fact that TM_Q only assigns non-zero annotations to t and $\mathbf{o}$ if $t \sqsubseteq \mathbf{o}$ that we have proven above implies that $Q(\mathbf{R})$ bounds $Q(R)$ based on $\mathcal{TM}_Q$.

Aggregation without group-by: The proof for aggregation without group-by is analog except for that both $Q(R)$ and $Q(\mathbf{R})$ contain a single result tuple annotated with 1 and $(1, 1, 1)$ respectively. Let t and $\mathbf{o}$ denote this single result tuple. Then we trivially define $\mathcal{TM}_Q(\mathbf{t}, t) = 1$ and $\mathcal{TM}_Q(t', \mathbf{t}') = 0$ if either $t' \neq t$ or $\mathbf{t} \neq \mathbf{t}'$). Then, $Q(\mathbf{R})(\mathbf{o})^{\downarrow} \leq \sum_{t \sqsubseteq \mathbf{o}} \mathcal{TM}_Q(\mathbf{o}, t) \leq Q(\mathbf{R})(\mathbf{o})^{\uparrow}$. The proof of $t \sqsubseteq \mathbf{o}$ is analog to the proof for group-by aggregation.

From Thm. 13, Thm. 11, and Thm. 10 follows our main technical result: Our query semantics for $\mathcal{RA}^{agg}$ queries preserves bounds.

Corollary 3 (Preservation of bounds for $\mathcal{RA}^{agg}$)**.** *Let Q be an $\mathcal{RA}^{agg}$ query and $\mathcal{D}$ an incomplete $\mathbb{N}$-database that is bound by an $\mathbb{N}_{AU}$-database $\mathbf{D}$. Then $Q(\mathbf{D})$ bounds $Q(\mathcal{D})$.*

$$\mathcal{D} \sqsubset \mathbf{D} \Rightarrow Q(\mathcal{D}) \sqsubset Q(\mathbf{D})$$

Note that our semantics for $\mathcal{RA}^{agg}$ queries over $\mathbb{N}_{AU}$-relations has `PTIME` data complexity.

Theorem 14 (Data Complexity of $\mathcal{RA}^{agg}$ Queries)**.** *Evaluation of $\mathcal{RA}^{agg}$ queries over $\mathbb{N}_{AU}$-relations has `PTIME` data complexity.*

Proof: Query evaluation for $\mathcal{RA}^{+}$ over $\mathcal{K}$-relations is known to be in `PTIME`. For $\mathcal{RA}^{+}$, our semantics only differs in the evaluation of expressions which adds an overhead that is independent on the size of the input database. For set difference, the

semantics according to Def. 28 is in worst-case accessing the annotation of every tuple in the right-hand side input to calculate the annotation of a result tuple. Since each result tuple belongs to the left-hand side input, the complexity is certainly in $O(n^2)$ which is PTIME. Finally, for aggregation, the number of result tuples is at most linear in the input size and even a naive implementation just has to test for each input whether it contributes to a particular output. Thus, aggregation is certainly in $O(n^2)$ and we get an overall PTIME data complexity for evaluation of $\mathcal{RA}^{agg}$ queries over $\mathbb{N}_{AU}$-relations.

We then introduce the semantics for sorting and windowed aggregations. In order to formalize these semantics in AU-DBs, we first define deterministic sorting and windowed aggregation semantics. Note that, our formalization focuses on *logical* order; the objective of these operators is to materialize sort positions of rows *as data*.

5.6 Deterministic Top-k and Windowed Aggregation Semantics

Sort order: Assume a total order $<$ for the domains of all attributes. For simplicity, we only consider sorting in ascending order. The extension for supporting both ascending and descending order is straightforward. For any two tuples t and t' with schema $(A_1, \ldots, A_n)$ and sort attributes $O = (A_{i_1}, \ldots, A_{i_m})$ we define:

$$t <_O t' \Leftrightarrow \exists j \in \{1, \ldots, m\} : \\ \forall k \in \{1, \ldots, j-1\} : t.A_{i_k} = t'.A_{i_k} \wedge t'.A_{i_j} < t.A_{i_j}$$

The less-than or equals comparison operator $\leq_O$ generalizes this definition in the usual way. Note that SQL sorting (ORDER BY) and some window bounds (ROWS BETWEEN ...) may be non-deterministic. For instance, consider a relation R with schema (A, B) with two rows $t_1 = (1, 1)$ and $t_2 = (1, 2)$ each with multiplicity 1; Sorting this relation on attribute A (the tuples are indistinguishable on this attribute), can return the tuples in either order. Without loss of generality, we ensure a fully deterministic

semantics (up to tuple equivalence) by extending the ordering on attributes O, using the remaining attributes of the relation as a tiebreaker: The total order $t <_O^{total} t'$ for tuples from a relation R is defined as $t <_{O,\text{Sch}(R)-O} t'$ (assuming some arbitrary order of the attributes in $\text{Sch}(R)$).

Example 20 (Sorting)**.** *Consider the relation R shown on the left below. The multiplicity from $\mathbb{N}$ assigned to each tuple is shown on the right. The result of sorting the relation on attribute A using our deterministic semantics and storing the sort positions in column* `pos` *is shown below on the right. Note the order of tuples $t_1 = (3, 15)$ and $t_2 = (1, 1)$ is made deterministic by $t_2 <_A^{total} t_1$, because $t_2.B < t_1.B$. Note also that the two copies of t_2 are each assigned a different position.*

Table 5.3. ex. 20

A	B	$\mathbb{N}$
3	15	1
1	1	2

A	B	pos	$\mathbb{N}$
1	1	0	1
1	1	1	1
3	15	4	1

We first introduce operators for windowed aggregation, because sorting can be defined as a special case of windowed aggregation.

5.6.1 Windowed Aggregation. A windowed aggregate is defined by an aggregate function, a sort order (`ORDER BY`), and a window bound specification. A window boundary is relative to the defining tuple, by the order-by attribute value (`RANGE BETWEEN...`), or by position (`ROWS BETWEEN`). In the interest of space, we will limit our discussion to row-based windows, as range-based windows are strictly simpler. A window includes every tuple within a specified interval of the defining tuple. Windowed aggregation extends each input tuple with the aggregate value computed

over the tuple's window. If a `PARTITION BY` clause is present, then window boundaries are evaluated within a tuple's partition. In SQL, a single query may define a separate window for each aggregate function (SQL's `OVER` clause). This can be modeled by applying multiple window operators in sequence.

Example 21 (Row-Based Windows)**.** *Consider the bag relation below and consider the windowed aggregation* $sum(B)$ *sorting on* A *with bounds* $[-2, 0]$ *(including the two preceding tuples and the tuple itself). The window for the first duplicate of* $t_1 = (a, 5, 3)$ *contains tuple* t_1 *with multiplicity* 1*, the window for the second duplicate of* t_1 *contains* t_1 *with multiplicity* 2 *and so on. Because each duplicate of* t_1 *ends up in a different window, there are three result tuples produced for* t_1*, each with a different* $sum(B)$ *value. Furthermore, tuples* $t_2 = (b, 3, 1)$ *and* $t_3 = (b, 3, 4)$ *have the same position in the sort order, demonstrating the need to use* $<_O^{total}$ *to avoid non-determinism in what their windows are. We have* $t_2 <_O^{total} t_3$ *and, thus, the window for* t_2 *contains* t_2 *with multiplicity* 1 *and* t_1 *with multiplicity* 2 *while the window for* t_3 *contains* t_1*,* t_2 *and* t_3 *each with multiplicity* 1*.*

Table 5.4. ex. 21

A	**B**	**C**	$\mathbb{N}$
a	5	3	3
b	3	1	1
b	3	4	1

A	**B**	**C**	**sum(B)**	$\mathbb{N}$
a	5	3	5	1
a	5	3	10	1
a	5	3	15	1
b	3	1	13	1
b	3	4	11	1

The semantics of the row-based windowed aggregation operator ω is shown in Fig. 5.3. The parameters of ω are partition-by attributes G, order-by attributes O, an aggregate function $f(A)$ with $A \subseteq \text{Sch}(R)$, and an interval $[l, u]$. For simplicity,

we hide some arguments (G,O,l,u) in the definitions and assume they passed to intermediate definitions where needed. The operator outputs a relation with schema $\mathrm{Sch}(R) \circ X$.

The heavy lifting occurs in the definition of relation $\mathcal{R}(R)$, which "explodes" relation R, adding an attribute i to replace each tuple of multiplicity n with n distinct tuples. $\mathcal{R}(R)$ computes the windowed aggregate over the window defined for the pair (t, i), denoted as $\mathcal{W}_{R,t,i}(t')$. To construct this window, we define the multiplicity of tuple t' in the partition for tuple t (denoted as $\mathcal{P}_{R,t}(t')$), the range of sort positions the tuple t covers ($cover(R, t)$), and the range of positions in its window ($bounds(R, t, i)$). The multiplicity of tuple t' in the window for the i^{th} duplicate of t is the size of the overlap between the bounds $bounds(R, t, i)$, and the cover of t'.

$$\Omega^{[l,u]}_{f(A)\to X;\, G;\, O}(R)(t) = \begin{cases} R(t') & \textbf{if } t = t' \circ f(\pi_A(\mathcal{W}^{groups}_{G,O,l,u,R,t'})) \\ 0 & \textbf{otherwise} \end{cases}$$

$$\mathcal{W}^{groups}_{G,O,l,u,R,t}(t') = \begin{cases} R(t') & \textbf{if } (\mathsf{rank}(\mathcal{P}_{R,G,t}, O, t) \\ & \qquad -\mathsf{rank}(\mathcal{P}_{R,G,t}, O, t')) \in [l,u] \\ 0 & \textbf{otherwise} \end{cases}$$

$$\mathsf{rank}(R, O, t) = |\{\ t'.O \mid R(t') > 0 \wedge t' <_O t\ \}|$$

$$\omega^{[l,u]}_{f(A)\to X;\, G;\, O}(R)(t) = \pi_{\mathrm{Sch}(R),X}(\mathcal{R}(R))$$

$$\mathcal{R}(R)(t) = \begin{cases} 1 & \textbf{if } t = t' \circ f(\pi_A(\mathcal{W}_{R,t',i})) \circ i \\ & \qquad \wedge i \in [0, R(t') - 1] \\ 0 & \textbf{otherwise} \end{cases}$$

$$\mathcal{P}_{R,t}(t') = \begin{cases} R(t') & \textbf{if } t'.G = t.G \\ 0 & \textbf{otherwise} \end{cases}$$

$$\mathcal{W}_{R,t,i}(t') = |\, cover(\mathcal{P}_{R,t}, t') \cap bounds(\mathcal{P}_{R,t}, t, i)\,|$$

$$\mathsf{pos}(R, t, i) = i + \sum_{t' <^{total}_O t} R(t')$$

$$cover(R, t) = [\mathsf{pos}(R, t, 0), \mathsf{pos}(R, t, R(t) - 1)]$$

$$bounds(R, t, i) = [\mathsf{pos}(R, t, i) + l, \mathsf{pos}(R, t, i) + u]$$

Figure 5.3. Windowed Aggregation

5.6.2 Sort Operator. We now define a sort operator $\text{SORT}_{O\to\tau}(R)$ which extends each row of R with an attribute τ that stores the position of this row in R according to $<_O^{total}$. This operator is just "syntactic sugar" as it can be expressed using windowed aggregation.

Definition 35 (Sort Operator)**.** *Consider a relation R with schema $(A_1, \ldots, A_n)$, list of attributes $O = (B_1, \ldots, B_m)$ where each B_i is in* $\text{Sch}(R)$*. The* sort operator $\text{SORT}_{O\to\tau}(R)$ *returns a relation with schema $(A_1, \ldots, A_n, \tau)$ as defined below.*

$$\text{SORT}_{O\to\tau}(R) = \pi_{\text{Sch}(R),\tau-1\to\tau}(\omega^{[-\infty,0]}_{count(1)\to\tau;\,\emptyset;\,O}(R))$$

Top-k queries can be expressed using the sort operator followed by a selection. For instance, the SQL query shown below can be written as $\pi_{A,B}(\sigma_{r\leq 3}(\text{SORT}_{A\to r}(R)))$.

```
SELECT A,B FROM R ORDER BY A LIMIT 3;
```

5.7 AU-DB Sorting and Top-k Semantics

We now develop a bound-preserving semantics for sorting and top-k queries over AU-DBs. Recall that each tuple in an AU-DB is annotated with a triple of multiplicities and that each (range-annotated) value is likewise a triple. Elements of a range-annotated value $\mathbf{c} = [c_1/c_2/c_3]$ or multiplicity triple (n_1, n_2, n_3) are accessed as: $\mathbf{c}^{\downarrow} = c_1$, $\mathbf{c}^{sg} = c_2$, and $\mathbf{c}^{\uparrow} = c_3$. Both the uncertainty of a tuple's multiplicity and the uncertainty of the values of order-by attributes creates uncertainty in a tuple's position in the sort order. The earlier, because it determines how many duplicates of a tuple appear in the sort order which affects the position of tuples which may be larger wrt. to the sort order and the latter because it affects which tuples are smaller than a tuple wrt. the sort order. As mentioned before, a top-k query is a selection over the result of a sort operator which checks that the sort position of a tuple is less than or equal to k.

Comparison of Uncertain Values: Before introducing sorting over AU-DBs, we first discuss the evaluation of $<_O$ over tuples with uncertain values (recall that $<_O^{total}$ is defined in terms of $<_O$). A Boolean expression over range-annotated values evaluates to a bounding triple (using the order $\bot < \top$ where $\bot$ denotes false and $\top$ denotes true). Recall that the result of an evaluation of an expression e is denoted as $[\![e]\!]$. For instance, $[\![[1/1/3] < [2/2/2]]\!]_{=}[\bot/\top/\top]$, because the expression may evaluate to false (e.g., if the first value is 3 and the second values is 2), evaluates to true in the selected-guess world, and may evaluate to true (if the 1^{st} value is 1 and the 2^{nd} value is 2). The extension of $<$ to comparison of tuples on attributes O using $<_O$ is shown below. For example, consider tuples $\mathbf{t_1} = ([1/1/3], [a/a/a])$ and $\mathbf{t_2} = ([2/2/2], [b/b/b])$ over schema $R(A, B)$. We have $\mathbf{t_1} <_{A,B} \mathbf{t_2} = [\bot/\top/\top]$, because $\mathbf{t_1}$ could be ordered before $\mathbf{t_2}$ (if $\mathbf{t_1}.A$ is 1), is ordered before $\mathbf{t_2}$ in the selected-guess world ($1 < 2$), and may be ordered before $\mathbf{t_2}$ (if A is 3).

$$
\begin{aligned}
[\![\mathbf{t} <_O \mathbf{t}']\!]^{\downarrow} = \exists i \in \{1, \ldots, n\} : \forall j \in \{1, \ldots, i-1\} : \\
[\![\mathbf{t}.A_j = \mathbf{t}'.A_j]\!]^{\downarrow} \wedge [\![\mathbf{t}.A_i < \mathbf{t}'.A_i]\!]^{\downarrow} \\
[\![\mathbf{t} <_O \mathbf{t}']\!]^{sg} = \exists i \in \{1, \ldots, n\} : \forall j \in \{1, \ldots, i-1\} : \\
[\![\mathbf{t}.A_j = \mathbf{t}'.A_j]\!]^{sg} \wedge [\![\mathbf{t}.A_i < \mathbf{t}'.A_i]\!]^{sg} \\
[\![\mathbf{t} <_O \mathbf{t}']\!]^{\uparrow} = \exists i \in \{1, \ldots, n\} : \forall j \in \{1, \ldots, i-1\} : \\
[\![\mathbf{t}.A_j = \mathbf{t}'.A_j]\!]^{\uparrow} \wedge [\![\mathbf{t}.A_i < \mathbf{t}'.A_i]\!]^{\uparrow}
\end{aligned}
$$

To simplify notation, we will use $\mathbf{t} <_O \mathbf{t}'$ instead of $[\![\mathbf{t} <_O \mathbf{t}']\!]$.

Tuple Rank and Position: To define windowed aggregation and sorting over AU-DBs, we generalize pos using the uncertain version of $<_O$. The lowest possible position of the first duplicate of a tuple $\mathbf{t}$ in an AU-DB relation $\mathbf{R}$ is the total multiplicity of tuples $\mathbf{t}'$ that certainly exist ($\mathbf{R}(\mathbf{t}')^{\downarrow} > 0$) and are certainly smaller than $\mathbf{t}$ (i.e., $[\![\mathbf{t}' <_O \mathbf{t}]\!]^{\downarrow} = \top$). The selected-guess position of a tuple is the position of the tuple in

the selected-guess world, and the greatest possible position of $\mathbf{t}$ is the total multiplicity of tuples that possibly exist ($\mathbf{R}(\mathbf{t}')^{\uparrow} > 0$) and possibly precede $\mathbf{t}$ (i.e., $[\![\mathbf{t}' <_O \mathbf{t}]\!]^{\uparrow} = \top$). The sort position of the i^{th} duplicate (with the first duplicate being 0) is computed by adding i to the position bounds of the first duplicate.

$$\mathsf{pos}(\mathbf{R}, O, \mathbf{t}, i)^{\downarrow} = i + \sum_{(\mathbf{t}' <_O \mathbf{t})^{\downarrow}} \mathbf{R}(\mathbf{t}')^{\downarrow} \tag{5.25}$$

$$\mathsf{pos}(\mathbf{R}, O, \mathbf{t}, i)^{sg} = i + \sum_{(\mathbf{t}' <_O \mathbf{t})^{sg}} \mathbf{R}(\mathbf{t}')^{sg} \tag{5.26}$$

$$\mathsf{pos}(\mathbf{R}, O, \mathbf{t}, i)^{\uparrow} = i + \sum_{(\mathbf{t}' <_O \mathbf{t})^{\uparrow}} \mathbf{R}(\mathbf{t}')^{\uparrow} \tag{5.27}$$

5.7.1 AU-DB Sorting Semantics. To define AU-DB sorting, we split the possible duplicates of a tuple and extend the resulting tuples with a range-annotated value denoting the tuple's (possible) positions in the sort order. The certain multiplicity of the i^{th} duplicate of a tuple $\mathbf{t}$ in the result is either 1 for duplicates that are guaranteed to exist ($i < \mathbf{R}(\mathbf{t})^{\downarrow}$) and 0 otherwise. The selected-guess multiplicity is 1 for duplicates that do not certainly exist (in some possible world there may be less than i duplicates of the tuple), but are in the selected-guess world (the selected-guess world has i or more duplicates of the tuple). Finally, the possible multiplicity is always 1.

Definition 36 (AU-DB Sorting Operator)**.** *Let* $\mathbf{R}$ *be an AU-DB relation and* $O \subseteq \mathrm{Sch}(\mathbf{R})$. *The result of applying the sort operator* $\mathrm{SORT}_{O \to \tau}$ *to* $\mathbf{R}$ *is defined in Fig. 5.4*

Every tuple in the result of sorting is constructed by extending an input tuple $\mathbf{t}'$ with the range of positions $\mathsf{pos}(\mathbf{R}, O, \mathbf{t}', i)$ it may occupy wrt. the sort order. The definition decomposes $\mathbf{t}$ into a base tuple $\mathbf{t}'$, and a position triple for each duplicate of $\mathbf{t}$ in $\mathbf{R}$. We annotate all certain duplicates as certain $(1, 1, 1)$, remaining selected-guess (but uncertain) duplicates as uncertain $(0, 1, 1)$ and non-selected guess duplicates as possible $(0, 0, 1)$.

Example 22 (AU-DB Sorting)**.** *Consider the AU-DB relation* $\mathbf{R}$ *shown on the left below with certain, selected guess and possible multiplicities from* $\mathbb{N}^3$ *assigned to each*

$$\text{SORT}_{O\to\tau}(\mathbf{R})(\mathbf{t}) = \begin{cases} (1,1,1) & \textbf{if } \mathbf{t} = \mathbf{t}' \circ \mathsf{pos}(\mathbf{R}, O, \mathbf{t}', i) \wedge i \in \left[0, \mathbf{R}(\mathbf{t}')^{\downarrow}\right) \\ (0,1,1) & \textbf{if } \mathbf{t} = \mathbf{t}' \circ \mathsf{pos}(\mathbf{R}, O, \mathbf{t}', i) \wedge i \in \left[\mathbf{R}(\mathbf{t}')^{\downarrow}, \mathbf{R}(\mathbf{t}')^{sg}\right) \\ (0,0,1) & \textbf{if } \mathbf{t} = \mathbf{t}' \circ \mathsf{pos}(\mathbf{R}, O, \mathbf{t}', i) \wedge i \in \left[\mathbf{R}(\mathbf{t}')^{sg}, \mathbf{R}(\mathbf{t}')^{\uparrow}\right) \\ (0,0,0) & \textbf{otherwise} \end{cases}$$

Figure 5.4. Range-annotated sort operator semantics.

tuple. For values or multiplicities that are certain, we write only the certain value instead of the triple. The result of sorting the relation on attributes A, B using AU-DB sorting semantics and storing the sort positions in column `pos` *($\text{SORT}_{A,B\to pos}(\mathbf{R})$) is shown below on the right. Observe how the 1^{th} input tuple $\mathbf{t_1} = (1, [1/1/3])$ was split into two result tuples occupying adjacent sort positions. The 3^{rd} input tuple $\mathbf{t_3} = ([1/1/2], 2)$ could be the 1^{th} in sort order (if it's A value is 1 and the B values of the duplicates of $\mathbf{t_1}$ are equal to 3) or be at the 3^{rd} position if two duplicates of $\mathbf{t_1}$ exist and either A is 2 or the B values of $\mathbf{t_1}$ are all < 3.*

Table 5.5. ex. 22

A	**B**	$\mathbb{N}^3$
1	[1/1/3]	(1,1,2)
[2/3/3]	15	(0,1,1)
[1/1/2]	2	(1,1,1)

A	**B**	**pos**	$\mathbb{N}^3$
1	[1/1/3]	[0/0/1]	(1,1,1)
1	[1/1/3]	[1/1/2]	(0,0,1)
[1/1/2]	2	[0/1/2]	(1,1,1)
[2/3/3]	15	[2/2/3]	(0,1,1)

5.7.2 Bound Preservation. We now prove that our semantics for the sorting operator on AU-DB relations is bound preserving, i.e., given an AU-DB $\mathbf{R}$ that bounds

an incomplete bag database $\mathcal{R}$, the result of a sort operator $\text{SORT}_{O \to \tau}$ applied to $\mathbf{R}$ bounds the result of $\text{SORT}_{O \to \tau}$ evaluated over $\mathcal{R}$.

Theorem 15 (Bound Preservation of Sorting)**.** *Given an AU-DB relation* $\mathbf{R}$ *and incomplete bag relation* $\mathcal{R}$ *such that* $\mathcal{R} \sqsubset \mathbf{R}$*, and* $O \subseteq \text{Sch}(\mathcal{R})$*. We have:*

$$\text{SORT}_{O \to \tau}(\mathcal{R}) \sqsubset \text{SORT}_{O \to \tau}(\mathbf{R})$$

Proof: Since $\mathcal{R} \sqsubset \mathbf{R}$, for every possible world $R \in \mathcal{R}$, there has to exist a tuple matching $\mathcal{TM}_{\mathbf{R}}$ based on which this property holds. We will show that based on $\mathcal{TM}_{\mathbf{R}}$ we can generate a tuple matching for $\mathbf{R}_{res} = \text{SORT}_{O \to \tau}(\mathbf{R})$ and $\text{SORT}_{O \to \tau}(\mathcal{R})$. The existence of such a tuple matching for every $\text{SORT}_{O \to \tau}(R) \in \text{SORT}_{O \to \tau}(\mathcal{R})$ implies that $\text{SORT}_{O \to \tau}(\mathcal{R}) \sqsubset \text{SORT}_{O \to \tau}(\mathbf{R})$. WLOG consider tuple $\mathbf{t} \in \mathbf{R}$ and output tuples $\mathbf{t}_{res} = \mathbf{t} \circ \mathsf{pos}(\mathbf{R}, O, \mathbf{t}, i)$.

We first show that split the tuple $\mathbf{t}$ preserves tuple multiplicities. Applying the definitions from Fig. 5.4,

$$\mathbf{R}(\mathbf{t})^{\downarrow} = \sum_i \mathbf{R}_{res}(\mathbf{t} \circ \mathsf{pos}(\mathbf{R}, O, \mathbf{t}, i))^{\downarrow}$$

$$\mathbf{R}(\mathbf{t})^{sg} = \sum_i \mathbf{R}_{res}(\mathbf{t} \circ \mathsf{pos}(\mathbf{R}, O, \mathbf{t}, i))^{sg}$$

$$\mathbf{R}(\mathbf{t})^{\uparrow} = \sum_i \mathbf{R}_{res}(\mathbf{t} \circ \mathsf{pos}(\mathbf{R}, O, \mathbf{t}, i))^{\uparrow}$$

Thus, the split preserves upper and lower multiplicity bounds.

Then we propose a tuple matching $\mathcal{TM}_{res}$ that maps each split tuple to a specific deterministic tuple. Let $S_{\mathbf{t}} = \{t_1, \ldots, t_n\} \in R$ be the only tuples such that $\mathcal{TM}_{\mathbf{R}}(\mathbf{t}, t_i) > 0$ and let us assume that $t_i \leq_O t_j$ if $i < j$ which we can ensure by sorting these tuples based on O. Recall that both AU-DB sorting and deterministic sorting splits each tuple $\mathbf{t}$ (t) into individual tuples $\mathbf{t_i}$ (t_{ik}).

Let us denote n_i the total multiplicity of tuples orders before t_i, i.e., $n_i = \sum_{j<i} S_{\mathbf{t}}(t_j)$.

Note that n_i is the relative sort position of the first duplicate of t_j wrt. the first duplicate of $t_1(t_{11})$. Then in the result tuple matching we will define

$$\mathcal{TM}_{res}(\mathbf{t} \circ \mathsf{pos}(\mathbf{R}, O, \mathbf{t}, n_i + k), t_i \circ \mathsf{pos}(R, O, t_i, k)) = 1$$

for all $k < R(t_i)$.

Because of the definitaiton of ranged comparison that utilizes range-based scalar expression semantics which is bound preserving we know that if $t \sqsubseteq \mathbf{t}$ and $\mathbf{t}' \sqsubseteq \mathbf{t}'$ then $(t' <_O t) \leftarrow (\mathbf{t}' <_O \mathbf{t})^{\downarrow}$. First we use the definition of tuple matching. Because only part of $R(t')$ may be matched against $\mathbf{t}$ we have $\mathcal{TM}_{\mathbf{R}}(\mathbf{t}', t') \leq R(t')$

$$\begin{aligned}
\mathsf{pos}(\mathbf{R}, O, \mathbf{t}, 0)^{\downarrow} &= \sum_{(\mathbf{t}' <_O \mathbf{t})^{\downarrow}} \mathbf{R}(\mathbf{t}')^{\downarrow} \\
&\leq \sum_{(\mathbf{t}' <_O \mathbf{t})^{\downarrow}} \mathcal{TM}_{\mathbf{R}}(\mathbf{t}', t') \\
&\leq \sum_{t' : t' <_O t_1} R(t') = \mathsf{pos}(R, O, t_1, 0)
\end{aligned}$$

this is only for first tuple matched against $\mathbf{t}$. For t_i we know that

$$\mathsf{pos}(R, O, t_i, k) = n_i + k + \mathsf{pos}(R, O, t_1, 0) + \sum_{t'' \geq_O t_1 \wedge t'' <_O t_i \wedge t'' \notin S_{\mathbf{t}}} R(t'')$$

The position of k-th duplicate of t_i is constructed by adding number of matched preceding tuples to the position of the first matched tuple t_i, and we also need to add all tuples that are not matched to $\mathbf{t}$ but is in preceding of t_i.

For $\mathbf{t}$ we know that

$$\begin{aligned}
\mathsf{pos}(\mathbf{R}, O, \mathbf{t}, n_i + k)^{\downarrow} &= n_i + k + \mathsf{pos}(\mathbf{R}, O, \mathbf{t}, 0)^{\downarrow} \\
&\leq n_i + k + \mathsf{pos}(R, O, t_1, 0) \\
&\leq n_i + k + \mathsf{pos}(R, O, t_1, 0) + \sum_{t'' \geq_O t_1 \wedge t'' <_O t_i \wedge t'' \notin S_{\mathbf{t}}} R(t'') \\
&= \mathsf{pos}(R, O, t_i, k)
\end{aligned}$$

Thus, $\mathsf{pos}(\mathbf{R}, O, \mathbf{t}, n_i + k)$ lower bounds $\mathsf{pos}(R, O, t_i, k)$.

In an analog way, we have

$$\begin{aligned}
\mathsf{pos}(\mathbf{R}, O, \mathbf{t}, 0)^{\uparrow} &= \sum_{(\mathbf{t}' <_O \mathbf{t})^{\uparrow}} \mathbf{R}(\mathbf{t}')^{\uparrow} \\
&\geq \sum_{(\mathbf{t}' <_O \mathbf{t})^{\uparrow}} \mathcal{TM}_{\mathbf{R}}(\mathbf{t}', t') \\
&\geq \sum_{t' : t' <_O t_n} R(t') = \mathsf{pos}(R, O, t_n, 0) + \sum_{t'' \geq_O t_1 \wedge t'' <_O t_n \wedge t'' \notin S_{\mathbf{t}}} R(t'')
\end{aligned}$$

Following with

$$\begin{aligned}
\mathsf{pos}(\mathbf{R}, O, \mathbf{t}, n_i + k)^{\uparrow} &= n_i + k + \mathsf{pos}(\mathbf{R}, O, \mathbf{t}, 0)^{\uparrow} \\
&\geq n_i + k + \mathsf{pos}(R, O, t_n, 0) + \sum_{t'' \geq_O t_1 \wedge t'' <_O t_n \wedge t'' \notin S_{\mathbf{t}}} R(t'') \\
&\geq n_i + k + \mathsf{pos}(R, O, t_1, 0) + \sum_{t'' \geq_O t_1 \wedge t'' <_O t_i \wedge t'' \notin S_{\mathbf{t}}} R(t'') \\
&= \mathsf{pos}(R, O, t_i, k)
\end{aligned}$$

Thus, $\mathsf{pos}(R, O, t_i, k) \sqsubseteq \mathsf{pos}(\mathbf{R}, O, \mathbf{t}, n_i + k)$. Also since $t_i \sqsubseteq \mathbf{t}$, $\mathcal{TM}_{res}$ is valid.

5.8 AU-DB Windowed Aggregation

We now introduce a bound preserving semantics for windowed aggregation over AU-DBs. We have to account for three types of uncertainty: (i) uncertain partition membership if a tuple may not exist ($\mathbf{R}(\mathbf{t})^{\downarrow} = 0$) or has uncertain partition attribute values; (ii) uncertain window membership if a tuple's partition membership, position, or multiplicity are uncertain; and (iii) uncertain aggregation results from either preceding type of uncertainty, or if we are aggregating over uncertain values. We compute the windowed aggregation result for each input tuple in multiple steps: (i) we first use AU-DB sorting to split each input tuple into tuples whose multiplicities are at most one. This is necessary, because the aggregation function result may differ among the duplicates of a tuple (as is already the case for deterministic windowed

aggregation); (ii) we then compute for each tuple $\mathbf{t}$ an AU-DB relation $\mathcal{P}_{\mathbf{t}}(\mathbf{R})$ storing the tuples that certainly and possibly belong to the partition for that tuple; (iii) we then compute an AU-DB relation $\mathcal{W}_{\mathbf{R},\mathbf{t}}$ encoding which tuples certainly and possibly belong to the tuple's window; (iv) since row-based windows contain a fixed number of tuples, we then determine from the tuples that possibly belong to the window, the subset that together with the tuples that certainly belong to the window (these tuples will be in the window in every possible world) minimizes / maximizes the aggregation function result. This then enables us to bound the aggregation result for each input tuple from below and above. For instance, for a row-based window $[-2, 0]$, we know that the window for a tuple $\mathbf{t}$ will never contain more than 3 tuples. If we know that two tuples certainly belong to the window, then at most one of the possible tuples can be part of the window.

5.8.1 Windowed Aggregation Semantics. As before, we omit windowed aggregation parameters (G,O,l,u,f,A) from the arguments of intermediate constructs and assume they are passed along where needed.

Partitions We start by defining AU-DB relation $\mathcal{P}_{\mathbf{t}}(\mathbf{R})$ which encodes the multiplicity with which a tuple $\mathbf{t}'$ belongs to the partition for $\mathbf{t}$ based on partition-by attributes G. This is achieved using selection, comparing a tuple's values in G with the values of $\mathbf{t}.G$ on equality. AU-DB selection sets the certain, selected-guess, possible multiplicity of a tuple to 0 if the tuple possibly, in the selected-guess world, or certainly does not fulfill the selection condition.

$$\mathcal{P}_{\mathbf{t}}(\mathbf{R}) = [\![\sigma_{G=\mathbf{t}.G}(\mathbf{R})]\!]$$

Certain and Possible Windows: We need to be able to reason about which tuples (and with which multiplicity) belong certainly to the window for a tuple and which tuples (with which multiplicity) could possibly belong to a window. For a tuple $\mathbf{t}$, we model the window's tuples as an AU-DB relation $\mathcal{W}_{\mathbf{R},\mathbf{t}}$ where a tuple's lower bound

multiplicity encodes the number of duplicates of the tuple that are certainty in the window, the selected-guess multiplicity encodes the multiplicity of the tuple in the selected-guess world, and the upper bound encodes the largest possible multiplicity with which the tuple may occur in the window minus the certain multiplicity. In the remainder of this section we omit the definition of the select-guess, because it can be computed using the deterministic semantics for windowed aggregation. We formally define $\mathcal{W}_{\mathbf{R},\mathbf{t}}$ in Fig. 5.6. Recall that in the first step we used sort to split the duplicates of each tuple into tuples with multiplicity upper bounds of 1. Thus, the windows we are constructing here are for tuples instead of for individual duplicates of a tuple. A tuple $\mathbf{t}'$ is guaranteed to belong to the window for of a tuple $\mathbf{t}$ with a multiplicity of $n = \mathbf{R}(\mathbf{t}')^{\downarrow}$ (the number of duplicates of the tuple that certainly exist) if the tuple certainly belongs to the partition for $\mathbf{t}$ and all possible positions that these n duplicates of the tuple occupy in the sort order are guaranteed to be contained in the smallest possible interval of sort positions contained in the bounds of the window for $\mathbf{t}$. Tuple $\mathbf{t}'$ possibly belongs to the window of $\mathbf{t}$ if any of its possible positions falls within the interval of all possible positions of $\mathbf{t}$. As an example consider Fig. 5.5 which shows the sort positions that certainly (red) and possibly (green) belong to tuple $\mathbf{t}$'s window (window bounds [-1,4]). For any window $[l, u]$, sort positions certainly covered by the window start from latest possible starting position for $\mathbf{t}$'s window which is $\mathbf{t}.\tau^{\uparrow} + l$ ($6 + (-1) = 5$ in our example) and end at the earliest possible upper bound for the window which is $\mathbf{t}.\tau^{\uparrow} + u$ ($4 + 4 = 8$ in our example). Furthermore, Fig. 5.5 shows the membership of three tuples in the window. Tuple $\mathbf{t_1}$ does certainly not belong to the window, because none of its possible sort positions are in the window's set of possible sort positions, $\mathbf{t_2}$ does certainly belong to the window, because all of its possible sort positions are in the set of positions certainly in the window. Finally, $\mathbf{t_3}$ possibly belongs to the window, because some of its sort positions are in the set of positions possibly covered by the window.

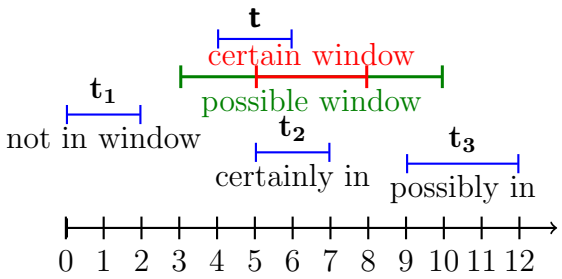


Figure 5.5. Possible and certain window membership of tuples in window of $\mathbf{t}$ based on their possible sort positions for window spec [-1,4].

$$\mathcal{W}_{\mathbf{R},\mathbf{t}}(\mathbf{t}')^{\downarrow} = \begin{cases} \mathcal{P}_{\mathbf{t}}(\mathbf{R})(\mathbf{t}')^{\downarrow} & \textbf{if } [\mathsf{pos}(\mathcal{P}_{\mathbf{t}}(\mathbf{R}), O, \mathbf{t}', 0)^{\downarrow}, \mathsf{pos}(\mathcal{P}_{\mathbf{t}}(\mathbf{R}), O, \mathbf{t}', 0)^{\uparrow}] \subseteq \\ & \quad [\mathsf{pos}(\mathcal{P}_{\mathbf{t}}(\mathbf{R}), O, \mathbf{t}, 0)^{\uparrow} + l, \mathsf{pos}(\mathcal{P}_{\mathbf{t}}(\mathbf{R}), O, \mathbf{t}, 0)^{\downarrow} + u] \\ 0 & \textbf{otherwise} \end{cases}$$

$$\mathcal{W}_{\mathbf{R},\mathbf{t}}(\mathbf{t}')^{\uparrow} = \begin{cases} \mathcal{P}_{\mathbf{t}}(\mathbf{R})(\mathbf{t}')^{\uparrow} - \mathcal{W}_{\mathbf{R},\mathbf{t},i}(\mathbf{t}')^{\downarrow} & \textbf{if } ([\mathsf{pos}(\mathcal{P}_{\mathbf{t}}(\mathbf{R}), O, \mathbf{t}', 0)^{\downarrow}, \mathsf{pos}(\mathcal{P}_{\mathbf{t}}(\mathbf{R}), O, \mathbf{t}', 0)^{\uparrow}] \\ & \quad \cap [\mathsf{pos}(\mathcal{P}_{\mathbf{t}}(\mathbf{R}), O, \mathbf{t}, 0)^{\downarrow} + l, \mathsf{pos}(\mathcal{P}_{\mathbf{t}}(\mathbf{R}), O, \mathbf{t}, 0)^{\uparrow} + u]) \neq \emptyset \\ 0 & \textbf{otherwise} \end{cases}$$

$$\mathcal{W}_{\mathbf{R},\mathbf{t}}(\mathbf{t}')^{sg} = \begin{cases} \mathcal{P}_{\mathbf{t}}(\mathbf{R})(\mathbf{t}')^{sg} & \textbf{if } \mathsf{pos}(\mathcal{P}_{\mathbf{t}}(\mathbf{R}), O, \mathbf{t}', 0)^{sg} \subseteq \\ & \quad [\mathsf{pos}(\mathcal{P}_{\mathbf{t}}(\mathbf{R}), O, \mathbf{t}, 0)^{sg} + l, \mathsf{pos}(\mathcal{P}_{\mathbf{t}}(\mathbf{R}), O, \mathbf{t}, 0)^{sg} + u] \\ 0 & \textbf{otherwise} \end{cases}$$

Figure 5.6. Certain and possible windows for row-based windowed aggregation over AU-DBs

Combining and Filtering Certain and Possible Windows: As mentioned above, row-based windows contain a fixed maximal number of tuples based on their bounds. We use $\mathsf{size}([l, u])$ to denote the size of a window with bounds $[l, u]$, i.e., $\mathsf{size}([l, u]) = (u - l) + 1$. This limit on the number of tuples in a window should be taken into account when computing bounds on the result of an aggregation function. For that, we combine the tuples certainly in the window (say there are m such tuples)

with a selected bag of up to $\mathsf{size}([l, u]) - m$ rows possibly in the window that minimizes (for the lower aggregation result bound) or maximizes (for the upper aggregation result bound) the aggregation function result for an input tuple. Let us use $\mathsf{possn}(\mathbf{R}, \mathbf{t})$ to denote $\mathsf{size}([l, u]) - m$:

$$\mathsf{possn}(\mathbf{R}, \mathbf{t}) = \mathsf{size}([l, u]) - \sum_{\mathbf{t}'} \mathcal{W}_{\mathbf{R},\mathbf{t}}(\mathbf{t}')^{\downarrow}$$

Which bag of up to $\mathsf{possn}(\mathbf{R}, \mathbf{t})$ tuples minimizes / maximizes the aggregation result depends on what aggregation function is applied. For **sum**, the up to $\mathsf{possn}(\mathbf{R}, \mathbf{t})$ rows with the smallest negative values are included in the lower bound and the up to $\mathsf{possn}(\mathbf{R}, \mathbf{t})$ rows with the greatest positive values for the upper bound. For **count** no additional row are included for the lower bound and up to $\mathsf{possn}(\mathbf{R})$ rows for the upper bound.

For each tuple $\mathbf{t}$, we define AU-DB relation $\mathcal{RW}_{\mathbf{R},\mathbf{t}}$ where each tuple's lower/upper bound multiplicities encode the multiplicity of this tuple contributing to the lower and upper bound aggregation result, respectively. We only show the definition for **sum**, the definitions for other aggregation functions are similar. In the definition, we make use $\mathbf{R}^{\downarrow}$ and $\mathbf{R}^{\uparrow}$:

$$\mathbf{R}^{\downarrow}(\mathbf{t}) = \mathbf{R}(\mathbf{t})^{\downarrow} \qquad\qquad \mathbf{R}^{\uparrow}(\mathbf{t}) = \mathbf{R}(\mathbf{t})^{\uparrow}$$

Note that $\mathbf{R}^{\downarrow}$ and $\mathbf{R}^{\uparrow}$ are bags ($\mathbb{N}$-relations) over range-annotated tuples. Furthermore, we define $\mathsf{min\text{-}k}(\mathbf{R}, \mathbf{t}, A)$ (and $\mathsf{max\text{-}k}(\mathbf{R}, \mathbf{t}, A)$) that are computed by restricting $\mathcal{W}_{\mathbf{R},\mathbf{t}}$ to the tuples with the smallest negative values (largest positive values) as lower (upper) bounds on attribute A that could contribute to the aggregation, keeping tuples with a total multiplicity of up to $\mathsf{possn}(\mathbf{R}, \mathbf{t})$. Note that the deterministic conditions / expressions in the definition of $\mathsf{min\text{-}k}(\mathbf{R}, \mathbf{t}, A)$ (and $\mathsf{max\text{-}k}(\mathbf{R}, \mathbf{t}, A)$) are well-defined, because single values are extracted from all range-annotated values. For

max (resp., **min**) and similar idempotent aggregates, it suffices to know the greatest (resp., least) value possibly in the window.

$$\mathcal{RW}_{\mathbf{R},\mathbf{t}}(\mathbf{t}')^{\downarrow} = \mathcal{W}_{\mathbf{R},\mathbf{t}}(\mathbf{t}')^{\downarrow} + \mathsf{min\text{-}k}(\mathbf{R},\mathbf{t},A)(\mathbf{t}')$$

$$\mathcal{RW}_{\mathbf{R},\mathbf{t}}(\mathbf{t}')^{\uparrow} = \mathcal{W}_{\mathbf{R},\mathbf{t}}(\mathbf{t}')^{\downarrow} + \mathsf{max\text{-}k}(\mathbf{R},\mathbf{t},A)(\mathbf{t}')$$

$$\mathcal{RW}_{\mathbf{R},\mathbf{t}}(\mathbf{t}')^{sg} = \mathcal{W}_{\mathbf{R},\mathbf{t}}(\mathbf{t}')^{sg}$$

$$\mathsf{min\text{-}k}(\mathbf{R},\mathbf{t},A) = \sigma_{\tau<\mathsf{possn}(\mathbf{R},\mathbf{t})}(\mathrm{SORT}_{A^{\downarrow}\to\tau}(\sigma_{A^{\downarrow}<0}(\mathcal{W}_{\mathbf{R},\mathbf{t}}{}^{\uparrow})))$$

$$\mathsf{max\text{-}k}(\mathbf{R},\mathbf{t},A) = \sigma_{\tau<\mathsf{possn}(\mathbf{R},\mathbf{t})}(\mathrm{SORT}_{-A^{\uparrow}\to\tau}(\sigma_{A^{\uparrow}>0}(\mathcal{W}_{\mathbf{R},\mathbf{t}}{}^{\uparrow})))$$

Windowed Aggregation: Using the filtered combined windows we are ready to define row-based windowed aggregation over AU-DBs. To compute aggregation results, we utilize the operation $\circledast_f$ defined in [40] for aggregation function f that combines the range-annotated aggregation attribute value of a tuple with the tuple's multiplicity bounds. For instance, for **sum**, $\circledast_{\mathbf{sum}}$ is multiplication, e.g., if a tuple with A value [10/20/30] has multiplicity $(1,2,3)$ it contributes [10/40/90] to the sum. Here, $\bigoplus$ denotes the application of the aggregation function over a set of elements (e.g., $\sum$ for **sum**). Note that, as explained above, the purpose of $\mathsf{expand}(\mathbf{R})$ is to split a tuple with n possible duplicates into n tuples with a multiplicity of 1. Furthermore, note that the bounds on the aggregation result may be the same for the i^{th} and j^{th} duplicate of a tuple. To deal with that we apply a final projection to merge such duplicate result tuples.

Definition 37 (Row-based Windowed Aggregation)**.** *Let* $\mathbf{R}$ *be an AU-DB relation.*

We define window operator $\omega^{[l,u]}_{f(A)\to X;\, G;\, O}$ *as:*

$$\omega^{[l,u]}_{f(A)\to X;\, G;\, O}(\mathbf{R})(\mathbf{t}) = \pi_{\mathrm{Sch}(\mathbf{R}),X}(\mathcal{R}(\mathbf{R}))$$

$$\mathcal{R}(\mathbf{R})(\mathbf{t} \circ \mathit{aggres}(\mathbf{t})) = \mathit{expand}(\mathbf{R})(\mathbf{t})$$

$$\mathit{aggres}(\mathbf{t}) = \bigoplus_{\mathbf{t}'} \mathbf{t}'.A \circledast_f \mathcal{RW}_{\mathit{expand}(\mathbf{R}),\mathbf{t}}(\mathbf{t}')$$

$$\mathit{expand}(\mathbf{R}) = \pi_{\mathrm{Sch}(\mathbf{R}),\tau_{id}}(\mathrm{SORT}_{\mathrm{Sch}(\mathbf{R})\to\tau_{id}}(\mathbf{R}))$$

Example 23 (AU-DB Windowed Aggregation)**.** *Consider the AU-DB relation* $\mathbf{R}$ *shown below and query* $\omega^{[-1,0]}_{sum(C)\to SumA;\, A;\, B}(\mathbf{R})$*, i.e., windowed aggregation partitioning by* A*, ordering on* B*, and computing* $sum(C)$ *over windows including* 1 *preceding and the current row. For convenience we show an identifier for each tuple on the left. As mentioned above, we first expand each tuple with a possible multiplicity larger then one using sorting. Consider tuple* $\mathbf{t_3}$*. Both* $\mathbf{t_1}$ *and* $\mathbf{t_2}$ *may belong to the same partition as* $\mathbf{t_3}$ *as their* A *value ranges overlap. There is no tuple that certainly belongs to the same partition as* $\mathbf{t_3}$*. Thus, only tuple* $\mathbf{t_3}$ *itself will certainly belong to the window. To compute the bounds on the aggregation result we first determine which tuples (in the expansion created through sorting) may belong to the window for* $\mathbf{t_3}$*. These are the two tuples corresponding to the duplicates of* $\mathbf{t_1}$*, because these tuples may belong to the partition for* $\mathbf{t_3}$ *and their possible sort positions (*$[0/0/1]$ *and* $[1/1/2]$*) overlap with the sort positions possibly covered by the window for* $\mathbf{t_3}$ *(*$[0/1/2]$*). Since the size of the window is 2 tuples, the bounds on the sum are computed using the lower / upper bound on the* C *value of* $\mathbf{t_3}$ *(*$[2/4/5]$*) and no additional tuple from the possible window (because the* C *value of* $\mathbf{t_1}$ *is positive) for the lower bound and the largest possible* C *value of one copy (we can only fit one additional tuple into the window) of* $\mathbf{t_1}$ *(*7*) for the upper bound. Thus, we get the aggregation result* $[2/11/12]$ *as shown below.*

5.8.2 Bound Preservation. We now prove this semantics for group-based and row-based windowed aggregation over AU-DBs to be bound preserving.

Table 5.6. Ex. 23

	A	B	C	$\mathbb{N}^3$
$\mathbf{t_1}$	1	[1/1/3]	7	(1,1,2)
$\mathbf{t_2}$	[2/3/3]	15	4	(0,1,1)
$\mathbf{t_3}$	[1/1/2]	2	[2/4/5]	1

	A	B	C	SumC	$\mathbb{N}^3$
$\mathbf{r_1}$	1	[1/1/3]	7	[7/7/14]	1
$\mathbf{r_2}$	1	[1/1/3]	7	[7/7/14]	(0,0,1)
$\mathbf{r_3}$	[1/1/2]	2	[2/4/5]	[2/11/12]	1
$\mathbf{r_4}$	[2/3/3]	15	4	[4/4/9]	(0,1,1)

Theorem 16 (Bound Preservation for Windowed Aggregation)**.** *Given an AU-DB relation* $\mathbf{R}$ *and incomplete bag relation* $\mathcal{R}$ *such that* $\mathcal{R} \sqsubset \mathbf{R}$*, and* $O \subseteq \mathrm{Sch}(\mathcal{R})$*. For any row-based windowed aggregation* $\omega^{[l,u]}_{f(A)\to X;\, G;\, O}$*, we have:*

$$\omega^{[l,u]}_{f(A)\to X;\, G;\, O}(\mathcal{R}) \sqsubset \omega^{[l,u]}_{f(A)\to X;\, G;\, O}(\mathbf{R})$$

Proof: For simplicity, we denote $\mathcal{R}_o$ and $\mathbf{R}_o$ as the input incomplete bag relation and input AU-DB relation s.t. $\mathcal{R}_o \sqsubset \mathbf{R}_o$. We use $\mathcal{R}$ and $\mathbf{R}$ as the output of $\mathsf{expand}()$ s.t. $\mathcal{R} \coloneqq \mathsf{expand}(\mathit{prel}_o)$ and $\mathbf{R} \coloneqq \mathsf{expand}(\mathbf{R}_o)$. By Thm. 10 with selection bounding proven and Thm. 15 with sort bounding proven, we have $\mathcal{R} \sqsubset \mathbf{R}$. WLOG, since $\mathsf{expand}(\mathbf{R})$ return relation with multiplicity of 1s, we use $\mathbf{t} \in \mathbf{R}$ and $t \in R \in \mathcal{R}$ s.t. $\mathcal{TM}_R(\mathbf{t}, t) = 1$. By Thm. 10, we have $\mathcal{P}_{\mathbf{t}}(\mathbf{R}) \sqsubset \mathcal{P}_t(\mathbf{R})$, so there exist valid tuple matching $\mathcal{TM}_{\mathcal{P}}$ bounds the output of $\mathcal{P}_t(\mathbf{R})$ and $\mathcal{P}_{\mathbf{t}}(\mathbf{R})$.

For $\mathcal{W}_{\mathbf{R},\mathbf{t}}$ computing uncertain window for $\mathbf{t}$ and $\mathcal{W}_{R,t}$ computing deterministic window for t, we first recast the definition of the AU-DB window for $\mathbf{R}$ and the window

for t as a selection query. Define conditions,

$$\theta \coloneqq \mathsf{pos}(\mathcal{P}_t(R), O, t', 0) \subseteq [\mathsf{pos}(\mathcal{P}_t(R), O, t, 0) + l, \mathsf{pos}(\mathcal{P}_t(R), O, t, 0) + u]$$

$$\boldsymbol{\theta}^{\uparrow} \coloneqq ([\mathsf{pos}(\mathcal{P}_{\mathbf{t}}(\mathbf{R}), O, \mathbf{t}', 0)^{\downarrow}, \mathsf{pos}(\mathcal{P}_{\mathbf{t}}(\mathbf{R}), O, \mathbf{t}', 0)^{\uparrow}] \cap [\mathsf{pos}(\mathcal{P}_{\mathbf{t}}(\mathbf{R}), O, \mathbf{t}, 0)^{\downarrow} + l, \mathsf{pos}(\mathcal{P}_{\mathbf{t}}(\mathbf{R}), O, \mathbf{t}, 0)^{\uparrow} + u]) \neq \emptyset$$

$$\boldsymbol{\theta}^{\downarrow} \coloneqq [\mathsf{pos}(\mathcal{P}_{\mathbf{t}}(\mathbf{R}), O, \mathbf{t}', 0)^{\downarrow}, \mathsf{pos}(\mathcal{P}_{\mathbf{t}}(\mathbf{R}), O, \mathbf{t}', 0)^{\uparrow}] \subseteq [\mathsf{pos}(\mathcal{P}_{\mathbf{t}}(\mathbf{R}), O, \mathbf{t}, 0)^{\uparrow} + l, \mathsf{pos}(\mathcal{P}_{\mathbf{t}}(\mathbf{R}), O, \mathbf{t}, 0)^{\downarrow} + u]$$

By rule of uncertain scalar evaluation, $\theta \sqsubseteq [\boldsymbol{\theta}^{\downarrow}, \boldsymbol{\theta}^{\uparrow}]$, so deterministic window candidate condition θ bounded by $\boldsymbol{\theta}^{\downarrow}$ and $\boldsymbol{\theta}^{\uparrow}$. By Thm. 8, we have

$$\begin{aligned}\mathcal{W}_{\mathbf{R},\mathbf{t}}(\mathbf{t}')^{\downarrow} =& \mathcal{P}_{\mathbf{t}}(\mathbf{R})(\mathbf{t}')^{\downarrow} \cdot_{\mathcal{K}} \boldsymbol{\theta}^{\downarrow} \\ =& \sigma_{\boldsymbol{\theta}}(\mathcal{P}_{\mathbf{t}}(\mathbf{R}))(\mathbf{t}')^{\downarrow}\end{aligned}$$

So $\mathcal{W}_{\mathbf{R},\mathbf{t}}{}^{\downarrow}$ lower bounds $\mathcal{W}_{R,t}$ and there exists valid tuple matching $\mathcal{TM}_{\mathcal{W}}$.

analog to lower bound,

$$\begin{aligned}\mathcal{W}_{\mathbf{R},\mathbf{t}}(\mathbf{t}')^{\uparrow} + \mathcal{W}_{\mathbf{R},\mathbf{t}}(\mathbf{t}')^{\uparrow} =& \mathcal{P}_{\mathbf{t}}(\mathbf{R})(\mathbf{t}')^{\uparrow} \cdot_{\mathcal{K}} \boldsymbol{\theta}^{\downarrow} \\ =& \sigma_{\boldsymbol{\theta}}(\mathcal{P}_{\mathbf{t}}(\mathbf{R}))(\mathbf{t}')^{\downarrow}\end{aligned}$$

So $\mathcal{W}_{\mathbf{R},\mathbf{t}}(\mathbf{t}')^{\uparrow} + \mathcal{W}_{\mathbf{R},\mathbf{t}}(\mathbf{t}')^{\uparrow}$ upper bounds $\mathcal{W}_{R,t}$. Put differently, $\mathcal{W}_{\mathbf{R},\mathbf{t}}{}^{\uparrow}$ upper bounds $\mathcal{W}_{R,t} - \mathcal{W}_{\mathbf{R},\mathbf{t}}{}^{\downarrow}$

Define possible window matched tuples $\mathcal{PW}_{R,t}(t') = \mathcal{W}_{R,t}(t') - \sum_{t'} \mathcal{TM}_{\mathcal{W}}(\mathbf{t}', t')$, by definition of tuple matching, $\forall_{\mathbf{t}'} : \sum_{t'} \mathcal{TM}_{\mathcal{W}}(\mathbf{t}', t') \geq \mathcal{W}_{\mathbf{R},\mathbf{t}}{}^{\downarrow}(\mathbf{t}')$ we get $\mathcal{W}_{R,t} - \mathcal{W}_{\mathbf{R},\mathbf{t}}{}^{\downarrow}$ upper bounds $\mathcal{PW}_{R,t}$, so $\mathcal{W}_{\mathbf{R},\mathbf{t}}{}^{\uparrow}$ upper bounds $\mathcal{PW}_{R,t}$

For t, defined by Sec. 5.6.1, we know that

$$\mathsf{aggres}(t) = \bigoplus_{t'} t'.A \otimes_f \mathcal{W}_{R,t}(t')$$

For $\mathbf{t}$, substituting based on Def. 37 we get

$$\mathsf{aggres}(\mathbf{t})^{\downarrow} = \left(\bigoplus_{\mathbf{t}'} \mathbf{t}'.A \circledast_f \mathcal{RW}_{\mathbf{R},\mathbf{t}}(\mathbf{t}') \right)^{\downarrow}$$

Since $\bigoplus$ is point-wise for **min**, **max** and **sum**,

$$= \bigoplus_{\mathbf{t}'} (\mathbf{t}'.A \circledast_f \mathcal{RW}_{\mathbf{R},\mathbf{t}}(\mathbf{t}'))^{\downarrow}$$

$$= \bigoplus_{\mathbf{t}'} min(\mathbf{t}'.A^{\downarrow} \otimes_f \mathcal{RW}_{\mathbf{R},\mathbf{t}}(\mathbf{t}')^{\downarrow}, \mathbf{t}'.A^{\uparrow} \otimes_f \mathcal{RW}_{\mathbf{R},\mathbf{t}}(\mathbf{t}')^{\downarrow},$$
$$\mathbf{t}'.A^{\downarrow} \otimes_f \mathcal{RW}_{\mathbf{R},\mathbf{t}}(\mathbf{t}')^{\uparrow}, \mathbf{t}'.A^{\uparrow} \otimes_f \mathcal{RW}_{\mathbf{R},\mathbf{t}}(\mathbf{t}')^{\uparrow})$$

Since $\mathcal{RW}_{\mathbf{R},\mathbf{t}}{}^{\downarrow}$ is constructed to minimize $\mathbf{t}'.A$,

$$= \bigoplus_{\mathbf{t}'} \mathbf{t}'.A^{\downarrow} \otimes_f \mathcal{RW}_{\mathbf{R},\mathbf{t}}(\mathbf{t}')^{\downarrow}$$

By definition of $\mathcal{RW}_{\mathbf{R},\mathbf{t}}(\mathbf{t}')^{\downarrow}$,

$$= \bigoplus_{\mathbf{t}'} \mathbf{t}'.A^{\downarrow} \otimes_f (\mathcal{W}_{\mathbf{R},\mathbf{t}}(\mathbf{t}')^{\downarrow} + \mathsf{min\text{-}k}(\mathbf{R},\mathbf{t},A)(\mathbf{t}'))$$

By semimodule law Equation (5.10),

$$= \bigoplus_{\mathbf{t}'} \mathbf{t}'.A^{\downarrow} \otimes_f \mathcal{W}_{\mathbf{R},\mathbf{t}}(\mathbf{t}')^{\downarrow} +_f \bigoplus_{\mathbf{t}'} \mathbf{t}'.A^{\downarrow} \otimes_f \mathsf{min\text{-}k}(\mathbf{R},\mathbf{t},A)(\mathbf{t}')$$

Since $t' \sqsubseteq \mathbf{t}'$ and $\mathcal{W}_{\mathbf{R},\mathbf{t}}{}^{\downarrow}$ lower bounds $\mathcal{W}_{R,t}$,

$$\leq \bigoplus_{t'} t'.A \otimes_f \sum_{t'} \mathcal{TM}_{\mathcal{W}}(\mathbf{t}',t) +_f \bigoplus_{\mathbf{t}'} \mathbf{t}'.A^{\downarrow} \otimes_f \mathsf{min\text{-}k}(\mathbf{R},\mathbf{t},A)(\mathbf{t}')$$

By using $k = \mathsf{possn}(\mathbf{R},\mathbf{t})$, we denote the deterministic top-k operation as $\mathsf{min\text{-}k}(\mathbf{R},\mathbf{t},A)(\mathbf{t}') = mink_{A^{\downarrow}}(\mathcal{W}_{\mathbf{R},\mathbf{t}}{}^{\uparrow})(\mathbf{t}')$ and we get

$$= \bigoplus_{t'} t'.A \otimes_f \sum_{t'} \mathcal{TM}_{\mathcal{W}}(\mathbf{t}',t) +_f \bigoplus_{\mathbf{t}'} \mathbf{t}'.A^{\downarrow} \otimes_f mink_{A^{\downarrow}}(\mathcal{W}_{\mathbf{R},\mathbf{t}}{}^{\uparrow})(\mathbf{t}')$$

Since $t' \sqsubseteq \mathbf{t}$ and $\mathcal{W}_{\mathbf{R},\mathbf{t}}{}^{\uparrow}$ upper bounds $\mathcal{PW}_{R,t}$,

$$\leq \bigoplus_{t'} t'.A \otimes_f \sum_{t'} \mathcal{TM}_{\mathcal{W}}(\mathbf{t}',t) +_f \bigoplus_{t'} t'.A \otimes_f mink_A(\mathcal{PW}_{R,t})(t)$$

trivially,

$$\leq \bigoplus_{t'} t'.A \otimes_f \sum_{t'} \mathcal{TM}_{\mathcal{W}}(\mathbf{t}', t) +_f \bigoplus_{t'} t'.A \otimes_f \mathcal{PW}_{R,t}(t')$$

$$= \bigoplus_{t'} t'.A \otimes_f \left(\sum_{t'} \mathcal{TM}_{\mathcal{W}}(\mathbf{t}', t) + \mathcal{W}_{R,t}(t') - \sum_{t'} \mathcal{TM}_{\mathcal{W}}(\mathbf{t}', t) \right)$$

$$= \bigoplus_{t'} t'.A \otimes_f \mathcal{W}_{R,t}(t') = \mathsf{aggres}(t)$$

So $\mathsf{aggres}(\mathbf{t})$ lower bounds $\mathsf{aggres}(t)$.

$\mathsf{aggres}(\mathbf{t})$ upper bounds $\mathsf{aggres}(t)$ can be proven in analog way.

Given that $t \sqsubseteq \mathbf{t}$ and $\mathsf{aggres}(t) \sqsubseteq \mathsf{aggres}(\mathbf{t})$, we get

$$\mathcal{TM}_{res}(\mathbf{t} \circ \mathsf{aggres}(\mathbf{t}), t \circ \mathsf{aggres}(t)) = \mathcal{TM}_{\mathbf{R}}(\mathbf{t}, t)$$

As we made no assumptions about $\mathbf{t}$ and t apart from $\mathcal{TM}_{\mathbf{R}}(\mathbf{t}, t) = 1$, this implies that $\mathcal{TM}_{res}$ is a tuple matching and, thus, this concludes the proof.

5.9 Implementation

In this section we discuss about the implementation of our AU-DB model as a middleware running on top of conventional database systems. For that we define an encoding of $\mathbb{N}_{AU}$-relations as classical bag semantics relations implemented as a function Enc which maps a $\mathbb{N}_{AU}$-database to a bag semantics database. We use Dec to denote the inverse of Enc. Using the encoding we apply query rewriting to propagate annotations and implement $\mathbb{N}_{AU}$-relational query semantics over the encoding. Our frontend rewriting engine receives a query Q over an $\mathbb{N}_{AU}$-annotated database $\mathbf{D}$ and rewrites this into a query $\text{REWR}(Q)$ that evaluated over $\text{Enc}(\mathbf{D})$ returns the encoding of $Q(\mathbf{D})$. That is, we will show that:

$$Q(\mathbf{D}) = \text{Dec}(Q_{merge}\text{Enc}(\mathbf{D}))$$

5.9.1 Relational encoding of AU-DBs. We now define Enc for a single $\mathbb{N}_{AU}$-relation $\mathbf{R}$. $\text{Enc}(\mathbf{D})$ is then defined as the database generated by applying Enc to

each relation $\mathbf{R} \in \mathbf{D}$. We use $\bar{A}$ to denote a set of attributes. We use $\text{Sch}(R)$ to denote the schema of input relation $\leftrightarrow$. The schema of $\text{Enc}(\mathbf{R})$ for an $\mathbb{N}_{AU}$-relation $\mathbf{R}$ with schema $\text{Sch}(\mathbf{R}) = (a_1, \ldots, a_n)$ is

$$\text{Sch}(\text{Enc}(\mathbf{R})) = (\bar{A}, \bar{A}^{\uparrow}, \bar{A}^{\downarrow}, row^{\downarrow}, row, row^{\uparrow})$$

$$\text{where } \bar{A} = \{{A_1}^{sg}, \ldots, {A_n}^{sg}\},$$
$$\bar{A}^{\uparrow} = \{{A_1}^{\uparrow}, \ldots, {A_n}^{\uparrow}\}, \bar{A}^{\downarrow} = \{{A_1}^{\downarrow}, \ldots, {A_n}^{\downarrow}\}$$

Example 24. *The schema of* $\text{Enc}(\mathbf{R})$ *for AU-DB relation* $\mathbf{R}(A, B)$ *is* $(A, B, A^{\downarrow}, B^{\downarrow}, A^{\uparrow}, B^{\uparrow}, row^{\downarrow}, row, row^{\uparrow})$.

For each tuple $\mathbf{t}$ with $\mathbf{R}(\mathbf{t}) \neq \mathbb{0}_{\mathbb{N}_{AU}}$, there will be one tuple $t = \text{enc}(\mathbf{t}, \mathbf{R}(\mathbf{t}))$ in $\text{Enc}(\mathbf{R})$ where $\text{enc}()$ is a function that maps tuples from $\mathbf{R}$ and their annotations to the corresponding tuple from $\text{Enc}(\mathbf{R})$. Attributes $row^{\downarrow}$, row, and $row^{\uparrow}$ are used to store $\mathbf{R}(\mathbf{t})$:

$$\text{enc}(\mathbf{t}, k).row^{\downarrow} = k^{\downarrow}$$
$$\text{enc}(\mathbf{t}, k).row = k^{sg}$$
$$\text{enc}(\mathbf{t}, k).row^{\uparrow} = k^{\uparrow}$$

For each attribute A_i, the three attributes ${A_i}^{\downarrow}$, ${A_i}^{sg}$, and ${A_i}^{\uparrow}$ are used to store the range-annotated value $\mathbf{t}.A_i$:

$$\text{enc}(\mathbf{t}, k).{A_i}^{\downarrow} = \mathbf{t}.{A_i}^{\downarrow}$$
$$\text{enc}(\mathbf{t}, k).{A_i}^{sg} = \mathbf{t}.{A_i}^{sg}$$
$$\text{enc}(\mathbf{t}, k).{A_i}^{\uparrow} = \mathbf{t}.{A_i}^{\uparrow}$$

In addition we define a function $\text{dec}()$ which takes a tuple in the encoding and returns the corresponding range-annotated tuple $\mathbf{t}$. Given a tuple t with schema

$\textsc{Sch}(\textsc{Enc}(\mathbf{R})) = (\bar{A}, \bar{A}^{\uparrow}, \bar{A}^{\downarrow}, row^{\downarrow}, row, row^{\uparrow})$ where $\mathbf{R}$ is a $\mathbb{N}_{AU}$-relation $\mathbf{R}$, $\textsc{dec}()$ returns a tuple with schema $\textsc{Sch}(\mathbf{R}) = (\bar{A})$ such that for all $A_i \in \text{Sch}(\mathbf{R})$:

$$\textsc{dec}(t).A_i = [t.A_i{}^{\downarrow} / t.A_i{}^{sg} / t.A_i{}^{\uparrow}]$$

Furthermore, we define a function $\textsc{rowdec}(t)$ which extracts the row annotation encoded by a tuple t in the encoding:

$$\textsc{rowdec}(t) = (t.row^{\downarrow}, t.row^{sg}, t.row^{\uparrow})$$

Having defined the schema and tuple-level translation, we define $\textsc{Enc}$ and its inverse $\textsc{Dec}$ below.

Definition 38 (Relational Encoding)**.** *Let* $\mathbf{R}$ *be a* $\mathbb{N}_{AU}$*-relation with schema* $(A_1, \ldots, A_n)$ *and let* $R = \textsc{Enc}(\mathbf{R})$*. Furthermore, let* $\mathbf{t}$ *be a tuple with schema* $\text{Sch}(\mathbf{R})$ *and* t *be a tuple with schema* $\text{Sch}(R)$*.*

$$\textsc{Enc}(\mathbf{R})(t) \coloneqq \begin{cases} 1 & \text{if } \exists \mathbf{t} : t = \textsc{enc}(\mathbf{t}) \wedge \mathbf{R}(\mathbf{t}) > \mathbb{0}_{\mathbb{N}_{AU}} \\ 0 & \text{otherwise} \end{cases}$$

$$\textsc{Dec}(R)(\mathbf{t}) \coloneqq \sum_{t : \textsc{dec}(t) = \mathbf{t}} \textsc{rowdec}(t) \cdot (R(t), R(t), R(t))$$

5.9.2 Rewriting. We now define the rewriting $\textsc{rewr}(\cdot)$. We assume that for each input relation $\mathbf{R}$ of a query, $\textsc{Enc}(\mathbf{R})$ has been materialized as a relation $R_{\textsc{Enc}}$. We will discuss how to create AU-DBs in Sec. 5.11. These techniques enable AU-DBs to be generated as part of the rewritten query in addition to supporting reading from a materialized input table. We call two tuples in the relational encoding value equivalent if they are equal after projecting away the row annotation attributes ($row^{\downarrow}$, row^{sg}, and $row^{\uparrow}$). Note that $\textsc{Enc}$ does never produce an output where two tuples are

value-equivalent. To ensure that intermediate results are valid encodings, we have to sum up the row annotations of value-equivalent tuples which requires aggregation for operators like projection and union. Observe that we only need to ensure that the final result of a rewritten query is a valid encoding. Thus, we can allow for valid-equivalent tuples as long as we ensure that they are merged in the final result. In the following let Q_1 be a query with result schema $\bar{A} = (A_1, \ldots, A_n)$ and Q_2 be a query with schema $\bar{B} = (B_1, \ldots, B_m)$. We use $e \to A$ in generalized projections to denote that the projection onto expression e renaming the result to A, e.g., $\pi_{A+B \to C, D \to E}$ has schema (C, E). We will use $\text{ENC}(\mathbf{t})$ to refer to the deterministic tuple in $\text{ENC}(\mathbf{R})$ that encodes the AU-DB tuple $\mathbf{t}$ and its annotation $\mathbf{R}(\mathbf{t})$.

Merge Annotations: After rewriting a Q using the rewriting scheme $\text{REWR}(\cdot)$ shown below, we merge the annotation of value-equivalent tuples to generate the final encoding. Given $\text{REWR}(Q)$, we return a rewritten query Q_{merge} to realize this:

$$Q_{merge} \coloneqq \gamma_{\bar{A}, \bar{A}^{\downarrow}, \bar{A}^{\uparrow}, e_c, e_{sg}, e_p}(\text{REWR}(Q))$$

$$e_c \coloneqq \mathbf{sum}(row^{\downarrow}) \to row^{\downarrow}$$

$$e_{sg} \coloneqq \mathbf{sum}(row^{sg}) \to row^{sg}$$

$$e_p \coloneqq \mathbf{sum}(row^{\uparrow}) \to row^{\uparrow}$$

Table Access: Each access to a table $\mathbf{R}$ is rewritten into an access to $\text{ENC}(\mathbf{R})$ which is materialized as R_{ENC}.

$$\text{REWR}(\mathbf{R}) \coloneqq R_{\text{ENC}}$$

Selection: For a selection we only filter out tuples $\text{ENC}(\mathbf{t})$ which are guaranteed to not fulfill the selection condition θ, i.e., where $\theta(\mathbf{t})^{\uparrow} = \bot$. Given an expression e, we use $e^{\uparrow}$ ($e^{\downarrow}$, and e^{sg}) to denote an expression that if applied to $\text{ENC}(\mathbf{t})$ for range-annotated tuple $\mathbf{t}$ returns $[\![e]\!]_{\tilde{\varphi}_{\mathbf{t}}}{}^{\uparrow}$ ($[\![e]\!]_{\tilde{\varphi}_{\mathbf{t}}}{}^{\downarrow}$, $[\![e]\!]_{\tilde{\varphi}_{\mathbf{t}}}{}^{sg}$), i.e., the upper (lower, SG) result of evaluating e over the $\mathbf{t}$ using range-annotated expression semantics (Def. 15). Recall

that $\tilde{\varphi}_{\mathbf{t}}$ denotes the range-annotated valuation that assigns tuple $\mathbf{t}$'s attribute values to the variables of expression e. We will use $\tilde{\varphi}_{\text{ENC}(\mathbf{t})}$ to denote the valuation that contains three variables $A^{\downarrow}, A^{sg}, A^{\uparrow}$ for each variable A in $\tilde{\varphi}$ and assigns these variables to values from $\text{ENC}(\mathbf{t})$ as follows:

$$\tilde{\varphi}_{\text{ENC}(\mathbf{t})}(A^{\downarrow}) = \text{ENC}(\mathbf{t}).A^{\downarrow} = \tilde{\varphi}_{\mathbf{t}}(A)^{\downarrow}$$
$$\tilde{\varphi}_{\text{ENC}(\mathbf{t})}(A^{sg}) = \text{ENC}(\mathbf{t}).A^{sg} = \tilde{\varphi}_{\mathbf{t}}(A)^{sg}$$
$$\tilde{\varphi}_{\text{ENC}(\mathbf{t})}(A^{\uparrow}) = \text{ENC}(\mathbf{t}).A^{\uparrow} = \tilde{\varphi}_{\mathbf{t}}(A)^{\uparrow}$$

Note that the expression semantics of Def. 15 only uses deterministic expression evaluation and it is always possible to generate deterministic expressions $e^{\downarrow}$, e^{sg}, and $e^{\uparrow}$. For instance, for a condition $e \coloneqq A \leq B$, we would generate $e^{\uparrow} \coloneqq A^{\uparrow} \leq B^{\downarrow}$. For instance, consider a tuple $\mathbf{t} = ([1/1/1], [0/1/2])$ with $\mathbf{R}(\mathbf{t}) = (1, 1, 1)$. This tuple would be encoded in $\text{ENC}(\mathbf{R})$ as $(A^{sg} : 1, B^{sg} : 1, A^{\downarrow} : 1, B^{\downarrow} : 0, A^{\uparrow} : 1, B^{\uparrow} : 1, row^{\downarrow} : 1, row^{sg} : 1, row^{\uparrow} : 1)$. We get $[\![e^{\uparrow}]\!]_{\tilde{\varphi}_{\text{ENC}(\mathbf{t})}} = 1 \leq 0 = \bot$. In the result of selection, the annotation of tuples is determined based on their annotation in the input and whether they certainly or in the SG world fulfill the selection conditions (see Def. 26).

$$\text{REWR}(\sigma_{\theta}(Q_1)) \coloneqq \pi_{\bar{A}, \bar{A}^{\uparrow}, \bar{A}^{\downarrow}, e_c, e_{sg}, row^{\uparrow}}(\sigma_{\theta^{\uparrow}}(\text{REWR}(Q_1)))$$
$$e_c \coloneqq (\textbf{if } \theta^{\downarrow} \textbf{ then } 1 \textbf{ else } 0) * row^{\downarrow} \to row^{\downarrow}$$
$$e_{sg} \coloneqq (\textbf{if } \theta^{sg} \textbf{ then } 1 \textbf{ else } 0) * row^{sg} \to row^{sg}$$

Projection: For a generalized projection $\pi_U(\mathbf{R})$ with $U \coloneqq e_1 \to A_1, \ldots, e_k \to A_k$, we rewrite each projection expression e_i into three expressions ${e_i}^{\downarrow}$, ${e_i}^{sg}$, and ${e_i}^{\uparrow}$ as explained for selection above. Then let $U^{\downarrow} = {e_1}^{\downarrow} \to {A_1}^{\downarrow}, \ldots, {e_k}^{\downarrow} \to {A_k}^{\downarrow}$ and let U^{sg} and $U^{\uparrow}$ be defined analog.

$$\text{REWR}(\pi_U(Q_1)) = \pi_{U^{sg}, U^{\uparrow}, U^{\downarrow}, row^{\downarrow}, row^{sg}, row^{\uparrow}}(\text{REWR}(Q_1))$$

Cross Product: Recall that $\cdot_{\mathbb{N}_{AU}}$ is defined as pointwise multiplication. Thus, for

crossproduct we have to multiply the bounds of row annotations of input tuples.

$$\begin{aligned}
\text{REWR}(Q_1 \times Q_2) &\coloneqq \pi_{\bar{A},\bar{B},\bar{A}^{\downarrow},\bar{B}^{\downarrow}\bar{A}^{\uparrow},\bar{B}^{\uparrow},e_c,e_{sg},e_p}(Q_{prod}) \\
Q_{prod} &\coloneqq \text{REWR}(Q_1) \times \text{REWR}(Q_2) \\
e_c &\coloneqq Q_1.row^{\downarrow} \cdot Q_2.row^{\downarrow} \rightarrow row^{\downarrow} \\
e_{sg} &\coloneqq Q_1.row^{sg} \cdot Q_2.row^{sg} \rightarrow row^{sg} \\
e_p &\coloneqq Q_1.row^{\uparrow} \cdot Q_2.row^{\uparrow} \rightarrow row^{\uparrow}
\end{aligned}$$

Union: A union is rewritten as the union of its rewritten inputs.

$$\text{REWR}(Q_1 \cup Q_2) \coloneqq \text{REWR}(Q_1) \cup \text{REWR}(Q_2)$$

Set Difference: For set difference we need to develop a rewrite that implements the combiner operator Ψ which merges all tuples with the same values in the SGW. This rewrite is shown below. To calculate the lower bound of the range-bounded annotation for a tuple in the result of a set difference operator we have to determine for each tuple $\mathbf{t}$ from the left input the set of all tuples $\mathbf{t}'$ from the right input whose values overlap with $\mathbf{t}$, i.e., where $\mathbf{t} \simeq \mathbf{t}'$. These are tuples that may be equal to $\mathbf{t}$ in some possible world. To calculate the lower bound we have to assume that all these tuples are equal to $\mathbf{t}$ and appear with the maximum possible multiplicity. That is, we have to subtract from the lower annotation bound of $\mathbf{t}$ the sum of the upper annotation bounds of these tuples. For that we join the inputs on a condition θ_{join} shown below that checks whether $\mathbf{t} \simeq \mathbf{t}'$ holds by checking that $[\mathbf{t}.A^{\downarrow}, \mathbf{t}.A^{\uparrow}]$ overlaps with $[\mathbf{t}'.B^{\downarrow}, \mathbf{t}'.B^{\uparrow}]$ for each attribute A of the left input and the corresponding attribute B of the right input. The AU-DB-annotation of a tuple is then computed by grouping on the SG values of the LHS, summing up the upper bounds of tuples from the RHS

and then subtract them from the lower bound of the LHS tuple's annotation. Below $Q_{SumRight}$ implements this step. To calculate the upper bound of a tuple's annotation we only subtract the lower bound annotations of tuples from the RHS if the tuples are guaranteed to be equal to the LHS in all possible worlds. That is the case if both the LHS and RHS tuple's attribute values are all certain (the lower bound is equal to the upper bound) and the tuples are equal. This is checking using condition θ_c shown below. Using a conditional expression e_{pv} we only sum up the lower bounds of the annotation of RHS tuples fulfilling θ_c. Finally, to calculate the multiplicity of a tuple in the SGW, we sum of the SG-annotations of RHS which are equal to the LHS tuple wrt. the tuples' SG-values. Finally, since this can result in negative multiplicities.

$$
\begin{aligned}
\text{REWR}(Q_1 - Q_2) &:= \sigma_{row^{\uparrow}>0}(\pi_{\bar{A},\bar{A}^{\downarrow},\bar{A}^{\uparrow},e_c,e_{sg},e_p}(Q_{sumright})) \\
e_c &:= \mathbf{max}(row^{\downarrow} - rrow^{\downarrow}, 0) \rightarrow row^{\downarrow} \\
e_{sg} &:= \mathbf{max}(row^{sg} - rrow^{sg}, 0) \rightarrow row^{sg} \\
e_p &:= \mathbf{max}(row^{\uparrow} - rrow^{\uparrow}, 0) \rightarrow row^{\uparrow} \\
Q_{sumright} &:= \gamma_{\bar{A},\bar{A}^{\uparrow},\bar{A}^{\downarrow},row^{\downarrow},row^{sg},row^{\uparrow},e_{sc},e_{ssg},e_{sp}}(Q_{preagg}) \\
e_{sc} &:= \mathbf{sum}(rrow^{\downarrow}) \rightarrow rrow^{\downarrow} \\
e_{ssg} &:= \mathbf{sum}(rrow^{sg}) \rightarrow rrow^{sg} \\
e_{sp} &:= \mathbf{sum}(rrow^{\uparrow}) \rightarrow rrow^{\uparrow} \\
Q_{preagg} &:= \pi_{\bar{A},\bar{A}^{\uparrow},\bar{A}^{\downarrow},Q_1.row^{\downarrow},Q_1.row^{sg},Q_1.row^{\uparrow},e_{cv},e_{sg},e_{pv}}(Q_{join}) \\
e_{cv} &:= Q_2.row^{\uparrow} \rightarrow rrow^{\downarrow} \\
e_{sgv} &:= \mathbf{if}\ \theta_{sg}\ \mathbf{then}\ Q_2.row^{sg}\ \mathbf{else}\ 0 \rightarrow rrow^{sg} \\
\theta_{sg} &:= \bigwedge_{A \in \bar{A}, B \in \bar{B}} A^{sg} = B^{sg} \\
e_{pv} &:= \mathbf{if}\ \theta_c\ \mathbf{then}\ Q_2.row^{\downarrow}\ \mathbf{else}\ 0 \rightarrow rrow^{\uparrow} \\
\theta_c &:= \bigwedge_{A \in \bar{A}, B \in \bar{B}} A^{\downarrow} = A^{\uparrow} \wedge A^{\uparrow} = B^{\downarrow} \wedge B^{\downarrow} = B^{\uparrow} \\
Q_{join} &:= \text{REWR}(\Psi(Q_1)) \ \sqcup_{\ \theta_{join}} \text{REWR}(Q_2) \\
\theta_{join} &:= \bigwedge_{i \in \{1,\ldots,n\}} {A_i}^{\uparrow} \geq {B_i}^{\downarrow} \wedge {B_i}^{\uparrow} \geq {A_i}^{\downarrow}
\end{aligned}
$$

$\Psi(Q)$**:** The SG combiner merges all tuples with the same values in the SGW by summing up their annotations and by merging their range-annotated values. We can

implement this in relational algebra using aggregation.

$$
\begin{aligned}
\text{REWR}(\Psi(Q)) &\coloneqq \gamma_{\bar{A},U_c,U_p,e_c,e_{sg},e_p}(\text{REWR}(Q)) \\
U_c &\coloneqq \min({A_1}^{\downarrow}) \to {A_1}^{\downarrow}, \ldots, \min({A_n}^{\downarrow}) \to {A_n}^{\downarrow} \\
U_p &\coloneqq \max({A_1}^{\uparrow}) \to {A_1}^{\uparrow}, \ldots, \max({A_n}^{\uparrow}) \to {A_n}^{\uparrow} \\
e_c &\coloneqq sum(row^{\downarrow}) \to row^{\downarrow} \\
e_{sg} &\coloneqq sum(row^{sg}) \to row^{sg} \\
e_p &\coloneqq sum(row^{\uparrow}) \to row^{\uparrow}
\end{aligned}
$$

Aggregation: Our rewrite for aggregation support **max**, **min**, **sum**, and **count** directly. For **avg**, we calculate **sum** and **count** and then calculate $\textbf{avg}(A) = \textbf{if count}(*) = 0 \textbf{ then } 0.0 \textbf{ else } \frac{\textbf{sum}(A)}{\textbf{count}(*)}$ using projection (not shown here). In the rewrite, we first determine output groups and ranged-bounded values for the group-by attributes of each of this output. This is achieved by grouping the input tuples based on the group-by SG values and calculating the minimum/maximum bounds of group-by values (query $Q_{gbounds}$). Each such output is then joined with the aggregation's input to match all inputs with an output that could contribute to the groups represented by this output. For that we have to check output's group-by bounds overlap with the input's group-by bound (query Q_{join}). Afterwards, we determine the bounds on the number of groups represented by each output and prepare expressions that calculate bounds for aggregation function results. These expression (*lba*, *sba*, and *uba*) are specific to the aggregation function f and are explained below. Finally, we use aggregation to calculate aggregation function result bounds and row annotations. Recall that $\bar{A}$ denotes the attributes from relation **R**.

$$\text{REWR}(\gamma_{G,f(A)}(R)) \coloneqq \gamma_{G^{sg},G^{\uparrow},G^{\downarrow},e_{aggbounds}}(Q_{proj})$$

$$
\begin{aligned}
e_{aggbounds} \coloneqq & f(A^{sg}), f(A^{\uparrow}), f(A^{\downarrow}) \\
& \mathbf{max}(row^{\downarrow}) \to row^{\downarrow}, \\
& \mathbf{max}(row^{sg}) \to row^{sg}, \\
& \mathbf{sum}(row^{\uparrow}) \to row^{\uparrow}
\end{aligned}
$$

$$
\begin{aligned}
Q_{proj} &\coloneqq \pi_{G^{sg},G^{\uparrow},G^{\downarrow},lba,sga,uba,e_c,e_{sg},e_p}(Q_{join}) \\
e_c &\coloneqq \mathbf{if}\ \theta_c\ \mathbf{then}\ 1\ \mathbf{else}\ 0 \to row^{\downarrow} \\
\theta_c &\coloneqq \bigwedge_{A_i \in G} {A_i}^{\uparrow} = {B_i}^{\uparrow} \wedge = {A_i}^{\downarrow} = {B_i}^{\downarrow} \wedge {A_i}^{\downarrow} = {A_i}^{\uparrow} \\
e_{sg} &\coloneqq \mathbf{if}\ \theta_{sg}\ \mathbf{then}\ 1\ \mathbf{else}\ 0 \to row^{sg} \\
\theta_{sg} &\coloneqq \bigwedge_{A_i \in G} {A_i}^{sg} = {B_i}^{sg} \\
e_p &\coloneqq \mathbf{if}\ \theta_{sg}\ \mathbf{then}\ row^{\uparrow}\ \mathbf{else}\ 0 \to row^{\uparrow}
\end{aligned}
$$

$$
\begin{aligned}
Q_{join} &\coloneqq Q_{gbounds} \ \sqcup_{\ \theta_{join}} \rho_{e_{rename}}(\text{REWR}(R)) \\
\theta_{join} &\coloneqq \bigwedge_{A_i \in G} {A_i}^{\uparrow} \geq {B_i}^{\downarrow} \wedge {B_i}^{\uparrow} \geq {A_i}^{\downarrow} \\
e_{rename} &\coloneqq {A_1}^{\downarrow} \to {B_1}^{\downarrow}, \ldots, {A_n}^{\downarrow} \to {B_n}^{\downarrow}, \ldots, {A_n}^{\uparrow} \to {B_n}^{\uparrow}
\end{aligned}
$$

$$
\begin{aligned}
Q_{gbounds} &\coloneqq \gamma_{G^{sg},e_{gbounds}}(\text{REWR}(R)) \\
e_{gbounds} &\coloneqq e^{A_1}_{bound}, \ldots, e^{A_k}_{bound} \qquad \text{(for } G = (A_1, \ldots, A_k)\text{)} \\
e^{A}_{bound} &\coloneqq \mathbf{min}(A^{\downarrow}) \to A^{\downarrow}, \mathbf{max}(A^{\uparrow}) \to A^{\uparrow}
\end{aligned}
$$

The expressions that calculate bounds for aggregation function results (*lba*, *sga*, *uba*) are shown below. These expressions make use of expressions lba_f, sga_f, and uba_f that are specific to the aggregation function f. We use expression e_{gc} shown below to determine whether a tuple certainly belongs to a particular group, i.e., its group-by values are certain and its lower bound multiplicity is larger than zero. If a

tuple's group membership is uncertain, then we need account for the case where the tuple does not contribute to the aggregation function result. For that we calculate the minimum/maximum of $\mathbb{0}_M$ and lba_f/uba_f.

$$
\begin{aligned}
lba &\coloneqq \textbf{if } e_{gc} \textbf{ then } lba_f \textbf{ else } \min(\mathbb{0}_M, lba_f) \rightarrow A^{\downarrow} \\
sga &\coloneqq sga_f \rightarrow A^{sg} \\
uba &\coloneqq \textbf{if } e_{gc} \textbf{ then } uba_f \textbf{ else } \max(\mathbb{0}_M, lba_f) \rightarrow A^{\uparrow} \\
e_{gc} &\coloneqq \theta_c \wedge row^{\downarrow} > 0
\end{aligned}
$$

For **sum**, we need to treat positive and negative numbers differently by multiplying them either with $row^{\downarrow}$ or $\text{UBAGG}(row)$ to return the smallest/greatest possible aggregation function result.

$$
\begin{aligned}
lba_{\textbf{sum}} &\coloneqq \textbf{if } A^{\downarrow} < 0 \textbf{ then } A^{\downarrow} \cdot row^{\uparrow} \textbf{ else } A^{\downarrow} \cdot row^{\downarrow} \\
sga_{\textbf{sum}} &\coloneqq \textbf{if } \bigwedge_{A_i \in G} {A_i}^{sg} = {B_i}^{sg} \textbf{ then } A^{sg} \cdot row^{sg} \textbf{ else } 0 \\
uba_{\textbf{sum}} &\coloneqq \textbf{if } A^{\uparrow} < 0 \textbf{ then } A^{\uparrow} \cdot row^{\downarrow} \textbf{ else } A^{\uparrow} \cdot row^{\uparrow}
\end{aligned}
$$

The neutral element of MIN is ∞ which is larger than any other value. Recall that $m *_{\mathsf{MIN}} k$ is the identify on m except when $k = 0$ where it returns ∞. Thus, the lowest possible value can be achieved if $k \neq 0$. Since $row^{\downarrow} \leq row^{\uparrow}$, $\textbf{if } row^{\uparrow} > 0 \textbf{ then } A^{\downarrow} \textbf{ else } \infty$ is a valid lower bound. Analog, $\textbf{if } row^{\downarrow} > 0 \textbf{ then } A^{\uparrow} \textbf{ else } \infty$ is an upper bound.

$$
\begin{aligned}
lba_{\textbf{min}} &\coloneqq \textbf{if } row^{\uparrow} > 0 \textbf{ then } A^{\downarrow} \textbf{ else } \infty \\
sga_{\textbf{min}} &\coloneqq \textbf{if } \bigwedge_{A_i \in G} {A_i}^{sg} = {B_i}^{sg} \textbf{ then } A \cdot row^{sg} \textbf{ else } \infty \\
uba_{\textbf{min}} &\coloneqq \textbf{if } row^{\downarrow} > 0 \textbf{ then } A^{\uparrow} \textbf{ else } \infty
\end{aligned}
$$

The neutral element of MAX is $-\infty$ which is smaller than any other value. $m *_{\mathsf{MIN}} k$ is the identify on m except when $k = 0$ where it returns ∞. Thus, the lowest

possible value can be achieved if $k = 0$. Thus, $\textbf{if } \sqcup = 0 \textbf{ then } -\infty \textbf{ else } A^{\downarrow}$ is a valid lower bound. Analog, $\textbf{if } row^{\downarrow} > 0 \textbf{ then } A^{\uparrow} \textbf{ else } \infty$ is an upper bound.

$$lba_{\textbf{max}} \coloneqq \textbf{if } row^{\downarrow} > 0 \textbf{ then } A^{\downarrow} \textbf{ else } -\infty$$

$$sga_{\textbf{max}} \coloneqq \textbf{if } \bigwedge_{A_i \in G} {A_i}^{sg} = {B_i}^{sg} \textbf{ then } A^{sg} \cdot row^{sg} \textbf{ else } -\infty$$

$$uba_{\textbf{max}} \coloneqq \textbf{if } row^{\uparrow} > 0 \textbf{ then } A^{\uparrow} \textbf{ else } -\infty$$

Sorting: The query rewrite rule for our ordering operator creates two copies of the input relation: Q_{lower} containing the lower bounds of the sort attributes, and Q_{upper} with the upper bounds. These tuples represent endpoints of value ranges for O, and so we refer to the tuples of the former as start tuples, and the latter as end tuples. The lower bound on the tuple's position wrt. $<_O^{total}$ is the number of tuples that certainly precede it: For a given start tuple and $\mathbf{t}.\# = row$, this is the total certain multiplicity ($\mathbf{t}.\#^{\downarrow}$) of end tuples that appear before it. The upper bound is computed similarly from the total possible multiplicity ($\mathbf{t}.\#^{\uparrow}$) of start tuples that precede an end tuple. Q_{bounds} computes these values for each start and end tuple using windowed aggregation to find all end tuple / start tuples that precede a point and then sum up their certain / possible multiplicity. Note that we use windowed aggregation with more than one aggregation function here which can be expressed as two windowed aggregation operators in our formalism. The resulting start tuples store lower bounds (resp., upper bounds for end tuples). The final rewrite is obtained by merging the start and end tuples back together using a group-by aggregate. The selected-guess position, not shown above, is computed analogously using a second window specification as part of Q_{bounds}.

$$
\begin{aligned}
\text{REWR}(\text{SORT}_{O\to\tau}(Q)) &\coloneqq \gamma_{\text{Sch}(\text{REWR}(Q)),e_{pos}}(Q_{bounds}) \\
e_{pos} &\coloneqq \mathbf{sum}(pos^{\downarrow}) \to \tau^{\downarrow}, \mathbf{sum}(pos^{sg}) \to \tau^{sg}, \mathbf{sum}(pos^{\uparrow}) \to \tau^{\uparrow} \\
Q_{bounds} &\coloneqq \omega^{[-\infty,0]}_{e^{\downarrow}\to pos^{\downarrow}, e^{sg}\to pos^{sg}, e^{\uparrow}\to pos^{\uparrow};\ \emptyset;\ pt}(Q_{lower} \cup Q_{upper} \cup Q_{sg}) \\
e^{\downarrow} &\coloneqq \textbf{if } \mathsf{isend} = 0 \textbf{ then } \mathbf{sum}(\mathsf{isend} \cdot \mathbf{t}.\#^{\downarrow}) \textbf{ else } 0 \\
e^{sg} &\coloneqq \textbf{if } \mathsf{isend} = -1 \textbf{ then } \mathbf{sum}(\mathbf{t}.\#^{sg}) \textbf{ else } 0 \\
e^{\uparrow} &\coloneqq \textbf{if } \mathsf{isend} = 1 \textbf{ then } \mathbf{sum}((1 - \mathsf{isend}) \cdot \mathbf{t}.\#^{\uparrow}) \textbf{ else } 0 \\
Q_{lower} &\coloneqq \pi_{\text{Sch}(\text{REWR}(Q)),O^{\downarrow}\to pt, 0\to \mathsf{isend}}(\text{REWR}(Q)) \\
Q_{sg} &\coloneqq \pi_{\text{Sch}(\text{REWR}(Q)),O^{sg}\to pt, -1\to \mathsf{isend}}(\text{REWR}(Q)) \\
Q_{upper} &\coloneqq \pi_{\text{Sch}(\text{REWR}(Q)),O^{\uparrow}\to pt, 1\to \mathsf{isend}}(\text{REWR}(Q))
\end{aligned}
$$

Windowed Aggregation: The ranged windowed aggregation query rewriting rule is shown below. The method uses a range overlap self-join on the partition-by attributes to link partition-definition tuple from Q_1 with potential members of the group from Q_2; The result is denoted Q_{join}. The relation Q_{rank} is defined in a manner analogous to the sorting rewrite, assigning each tuple to a position within its partition. Q_{window} builds the window, first filtering out all tuples that certainly do not belong to the window, and then Q_{cert} labels tuples with whether they definitely belong to the window. Next Q_{out} computes the windowed aggregate, and then the final rewritten query includes a join with a computation of the selected-guess result.

$$
\begin{aligned}
\text{REWR}(\omega^{[l,u]}_{\mathbf{sum}(A)\to X;\ G;\ O}(Q)) \coloneqq \pi_{\text{Sch}(Q_{aggbnds}),X}(\\
Q_{aggbnds} \bowtie_{id_1 = id} \omega^{[l,u]}_{\mathbf{sum}(A)\to X;\ G;\ O}(Q))
\end{aligned}
$$

$$Q_{aggbnds} \coloneqq \gamma_{\mathrm{Sch}(\textsc{rewr}(Q));\ \mathbf{sum}(e_{lb})\to X^{\downarrow},\mathbf{sum}(e_{ub})\to X^{\uparrow}}(Q_{inwindow})$$

$$e_{lb} \coloneqq \mathbf{if}\ pos^{\downarrow} < l + u \wedge (iscert \vee A^{\downarrow} < 0)$$

$$\mathbf{then}\ (\mathbf{if}\ A^{\downarrow} < 0\ \mathbf{then}\ A^{\downarrow}\cdot\mathbf{t}.\#_2{}^{\uparrow}\ \mathbf{else}\ A^{\downarrow}\cdot\mathbf{t}.\#_2{}^{\downarrow})\ \mathbf{else}\ 0$$

$$e_{ub} \coloneqq \mathbf{if}\ pos^{\uparrow} < l + u \wedge (iscert \vee A^{\uparrow} > 0)$$

$$\mathbf{then}\ (\mathbf{if}\ A^{\uparrow} > 0\ \mathbf{then}\ A^{\uparrow}\cdot\mathbf{t}.\#_2{}^{\uparrow}\ \mathbf{else}\ A^{\uparrow}\cdot\mathbf{t}.\#_2{}^{\downarrow})\ \mathbf{else}\ 0$$

$$Q_{inwindow} \coloneqq \sigma_{pos^{\downarrow}<l+u\ \vee\ pos^{\uparrow}<l+u}(Q_{aggupper})$$

$$Q_{aggupper} \coloneqq \omega^{[-\infty,0]}_{count(*)\to pos^{\uparrow};\ id_1;\ iscert,A^{\uparrow}}(Q_{agglower})$$

$$Q_{agglower} \coloneqq \omega^{[-\infty,0]}_{count(*)\to pos^{\downarrow};\ id_1;\ iscert,A^{\downarrow}}(Q_{markcert})$$

$$Q_{markcert} \coloneqq \pi_{\mathrm{Sch}(Q_{winposs}),e_{iscert}\to iscert}(Q_{winposs})$$

$$e_{iscert} \coloneqq e_{iscert_p} \wedge e_{iscert_w}$$

$$e_{iscert_p} \coloneqq Q_1.G^{\downarrow} = Q_1.G^{\uparrow} = Q_2.G^{\downarrow} = Q_2.G^{\uparrow}$$

$$e_{iscert_w} \coloneqq \tau^{\downarrow} >= selfpos^{\uparrow} - l \wedge \tau^{\uparrow} <= selfpos^{\downarrow} + u$$

$$Q_{winposs} \coloneqq \sigma_{e_{isposs_w}}(Q_{withselfpos})$$

$$e_{isposs_w} \coloneqq selfpos^{\downarrow} - l \leq \tau^{\uparrow} \wedge selfpos^{\uparrow} + u \geq \tau^{\downarrow}$$

$$Q_{withselfpos} \coloneqq Q_{pos} \bowtie_{id_1=id} Q_{selfpos}$$

$$Q_{selfpos} \coloneqq \pi_{id_1\to id,\tau^{\downarrow}\to selfpos^{\downarrow},\tau^{\uparrow}\to selfpos^{\uparrow}}(\sigma_{id_1=id_2}(Q_{pos}))$$

$$
\begin{aligned}
Q_{pos} &:= \gamma_{\text{Sch}(Q_{bnds}),e_{pos}}(Q_{bnds}) \\
e_{pos} &:= \mathbf{sum}(pos^{\downarrow}) \to \tau^{\downarrow}, \mathbf{sum}(pos^{\uparrow}) \to \tau^{\uparrow} \\
Q_{bnds} &:= \omega_{e_c \to pos^{\downarrow}, e_p \to pos^{\uparrow};\, id_1;\, pt}(Q_{endpoints}) \\
e_c &:= \mathbf{if}\ \mathsf{isend} = 1\ \mathbf{then}\ \mathbf{sum}(\mathsf{isend} \cdot \mathbf{t}.\#_2{}^{\downarrow})\ \mathbf{else}\ 0 \\
e_p &:= \mathbf{if}\ \mathsf{isend} = 1\ \mathbf{then}\ \mathbf{sum}((1 - \mathsf{isend}) \cdot \mathbf{t}.\#_2{}^{\uparrow})\ \mathbf{else}\ 0 \\
Q_{endpoints} &:= Q_{pos^{\downarrow}} \cup Q_{pos^{\uparrow}} \\
Q_{pos^{\downarrow}} &:= \pi_{\text{Sch}(Q_{part}),O^{\downarrow} \to pt, 0 \to \mathsf{isend}}(Q_{part}) \\
Q_{pos^{\uparrow}} &:= \pi_{\text{Sch}(Q_{part}),O^{\uparrow} \to pt, 1 \to \mathsf{isend}}(Q_{part}) \\
Q_{part} &:= \rho_{\{B_1 \leftarrow B \mid B \in \text{Sch}(\text{REWR}(Q))\}}(\text{REWR}(Q)) \\
&\qquad \bowtie_{\theta_{join}} \rho_{\{B_2 \leftarrow B \mid B \in \text{Sch}(\text{REWR}(Q))\}}(\text{REWR}(Q)) \\
\theta_{join} &:= Q_1.G^{\downarrow} \leq Q_2.G^{\uparrow} \wedge Q_1.G^{\uparrow} \geq Q_2.G^{\downarrow}
\end{aligned}
$$

5.9.3 Correctness. To demonstrate that our encoding and rewrites correctly implement AU-DB query semantics, we have to show that (i) the encoding is invertible, i.e., that there exists a mapping Dec such that $\text{Dec}(\text{Enc}(\mathbf{D})) = \mathbf{D}$, and (ii) that the rewrite correctly simulates AU-DB query semantics, i.e., $\text{rewr}(Q)(\text{Enc}(\mathbf{D})) = \text{Enc}(Q(\mathbf{D}))$.

Theorem 17 (Rewrite Correctness)**.** *Let* $\mathbf{D}$ *be a* $\mathbb{N}_{AU}$*-database,* Q *be a* $\mathcal{RA}^{agg}$ *query, then*

$$\text{Dec}(\text{Enc}(\mathbf{D})) = \mathbf{D} \qquad (\text{Enc } \textbf{is invertible})$$

$$Q_{merge}(\text{Enc}(\mathbf{D})) = \text{Enc}(Q(\mathbf{D})) \qquad (\text{rewr}(\cdot) \textbf{ is correct})$$

Proof:

Enc is invertible: Observe that by construction there exists a 1-to-1 mapping between the tuples in $\mathbf{R}$ and $\text{Enc}(\mathbf{R})$. A tuple $\mathbf{t}$ and its annotation $\mathbf{R}(\mathbf{t})$ can be trivially recon-

structed from the corresponding tuple t in $\text{ENC}(\mathbf{R})$ by setting $\mathbf{t}.A = [t.A^{\downarrow}/t.A^{sg}/t.A^{\uparrow}]$ for each attribute A of $\mathbf{R}$ and then setting $\mathbf{R}(\mathbf{t}) = (t.row^{\downarrow}, t.row^{sg}, t.row^{\uparrow})$.

REWR$(\cdot)$ is correct: We prove the claim by induction over the structure of a query. Since we have proven that $\text{DEC}(\text{ENC}(\mathbf{D})) = \mathbf{D}$, we prove the correctness of REWR$(\cdot)$ by showing that $\text{DEC}(Q_{merge}(\text{ENC}(\mathbf{D}))) = Q(\mathbf{D})$. We will we make use of this fact in the following.

Base case: Table access $Q := R$:: REWR(R) is the identify on $\text{ENC}(\mathbf{R})$. Thus, REWR(R) does not contain value-equivalent tuples and Q_{merge} is the identity on REWR(R) and the claim holds.

Induction: Assume that the claim holds for queries Q_1 and Q_2 modulo merging of value-equivalent tuples. We have to show that the claim holds modulo merging of value-equivalent tuples for each algebra operator applied to Q_1 (or Q_1 and Q_2 for binary operators). From this follows then that the claim holds for Q_{merge} which merges such tuples. Consider an input database $\mathbf{D}$. In the following let $\mathbf{R}_1 = Q_1(\mathbf{D})$ and $\mathbf{R}_2 = Q_2(\mathbf{D})$. Similarly, let $R_1 = \text{REWR}(Q_1)(\text{ENC}(\mathbf{D}))$ and $R_2 = \text{REWR}(Q_2)(\text{ENC}(\mathbf{D}))$.

Projection $Q := \pi_U(Q_1)$: Recall that $\pi_U(Q_1)$ is rewritten into

$$\pi_{U,U^{\downarrow},U^{\uparrow},row^{\downarrow},row^{sg},row^{\uparrow}}(\text{REWR}(Q_1))$$

. Let $U' = (U, U^{\downarrow}, U^{\uparrow})$ and $U_{all} = (U, U^{\downarrow}, U^{\uparrow}, row^{\downarrow}, row^{sg}, row^{\uparrow})$. The annotation of a tuple $\mathbf{t}$ in the result of Q is the sum of annotations of all input tuples $\mathbf{u}$ projected onto $\mathbf{t}$:

$$\pi_U(Q_1)(\mathbf{t}) = \sum_{\mathbf{u}:\mathbf{u}.U=\mathbf{t}} Q_1(\mathbf{u})$$

Let $\{\mathbf{u}_1, \ldots, \mathbf{u}_m\}$ be the sets of tuples for which $\mathbf{u_i}.U = \mathbf{t}$. Since, $\mathbf{u_i}.U = \mathbf{u_j}.A$, for any $i, j \in \{1, \ldots, m\}$ it follows that $\text{ENC}(\mathbf{u_i}, \mathbf{R}_1(\mathbf{u_i})).U' = \text{ENC}(\mathbf{u_j}, \mathbf{R}_1(\mathbf{u_j})).U'$ and in turn

that $\textsc{dec}(\textsc{enc}(\mathbf{u_i}).U') = \mathbf{t}$. Note that, $\textsc{enc}(\mathbf{u_i}, \mathbf{R}_1(\mathbf{u_i})).U_{all} \neq \textsc{enc}(\mathbf{u_j}, \mathbf{R}_1(\mathbf{u_j})).U_{all}$ if $\mathbf{R}_1(\mathbf{u_i}) \neq \mathbf{R}_1(\mathbf{u_j})$. Based on the induction hypothesis, each tuple $\mathbf{u_i}$ is encoded in R_1 as one or more value-equivalent tuples. Let $t_1, \ldots, t_l$ be the distinct tuples in the set $\{t \mid i \in \exists i \in \{1, \ldots, m\} : \mathbf{R}_1(\mathbf{u_i}).U_{all}\}$. We use $\mathbf{u_{i_1}}, \ldots, \mathbf{u_{m_i}}$ to denote the tuples for which $\mathbf{R}_1(\mathbf{u_{i_j}}).U_{all} = t_i$. Furthermore, for any such tuple t_i, let $u_{i_{j_1}}$, $\ldots$, $u_{i_{o_j}}$ be the value-equivalent tuples that are projected onto t_i. Then,

$$
\begin{aligned}
&\textsc{Dec}(\textsc{rewr}(Q)(\mathbf{D}))(\mathbf{t}) \\
=& \sum_{i \in \{1,\ldots,l\}} \textsc{rowdec}(t_i) \cdot \textsc{rewr}(Q)(\mathbf{D})(t_i)
\end{aligned}
$$

The definition of $\textsc{Enc}$ ensures that every tuple $\textsc{enc}(\mathbf{u_{i_j}})$ is annotated with 1. Given the definition of projection, t_i is annotated with the sum of annotations of all tuples $\textsc{enc}(\mathbf{u_i})$. That is, t_i is annotated with $\sum_{j=1}^{m_j} 1 = m_j$.

$$
\begin{aligned}
&= \sum_{i \in \{1,\ldots,l\}} \textsc{rowdec}(t_i) \cdot u_{m_i} \\
&= \sum_{i \in \{1,\ldots,l\}} \sum_{j \in \{1,\ldots,m_i\}} \textsc{rowdec}(t_i)
\end{aligned}
$$

$\textsc{rewr}(\pi_U(Q_1))$ does retain the row annotation attributes of input tuples unmodified. Thus, we have:

$$
= \sum_{i \in \{1,\ldots,l\}} \sum_{j \in \{1,\ldots,m_i\}} \textsc{rowdec}(\textsc{enc}(\mathbf{u_i}))
$$

Since we assume that claim holds for Q_1, we know that

$$
\begin{aligned}
&= \sum_{i = \{1,\ldots,m\}} \textsc{rowdec}(\textsc{enc}(\mathbf{u_i})) \\
&= \sum_{i = \{1,\ldots,m\}} \mathbf{R}_1(\textsc{enc}(\mathbf{u_i})) = Q(\mathbf{D})(\mathbf{t})
\end{aligned}
$$

We have proven that for any tuple $\mathbf{t}$, we have

$$
\textsc{Dec}(\textsc{rewr}(Q)(\mathbf{D}))(\mathbf{t}) = Q(\mathbf{D})(\mathbf{t})
$$

which implies that $\textsc{Dec}(\textsc{rewr}(Q)(\mathbf{D})) = Q(\mathbf{D})$.

Selection $Q \coloneqq \sigma_\theta(Q_1)$: Consider a tuple $\mathbf{t}$ such that $Q(\mathbf{D})(\mathbf{t})^\uparrow > 0$, i.e., tuples that may exists in the result of the selection. Selection over $\mathbb{N}_{AU}$-relations calculates the annotation of a result tuple $\mathbf{t}$ by multiplying its annotation in the input with the bounds of the result of the selection condition evaluated over $\mathbf{t}$ mapped from a range annotated Boolean value to a $\mathbb{N}_{AU}$-value where $\top$ is mapped to 1 and $\bot$ to 0. $\textsc{rewr}(Q)$ uses a triple of deterministic expressions to compute the elements of $\theta(\mathbf{t})$ individually over $t = \textsc{enc}(\mathbf{t}, \mathbf{R}_1(\mathbf{t}))$. Thus, $\textsc{rowdec}(t) = Q(\mathbf{D})(\mathbf{t})$ and the claim holds.

Union $Q \coloneqq Q_1 \cup Q_2$: Consider a tuple $\mathbf{t}$ such that either $\mathbf{R}_1(\mathbf{t}) > 0$ or $\mathbf{R}_2(\mathbf{t}) > 0$. Let T be the set of tuples that are value-equivalent to $\mathbf{t}$. Based on the induction hypothesis we know that $\mathbf{R}_1(\mathbf{t}) = \sum_{t \in T} \textsc{rowdec}(t) \cdot (R_1(t), R_1(t), R_1(t))$ and $\mathbf{R}_2(\mathbf{t}) = \sum_{t \in T} \textsc{rowdec}(t) \cdot (R_2(t), R_2(t), R_2(t))$. Recall that $\textsc{rewr}(Q_1 \cup Q_2) = \textsc{rewr}(Q_1) \cup \textsc{rewr}(Q_2)$. Based on the definition of union (semiring addition), we know that for any $t \in T$, it holds that $(R_1 \cup R_2)(t) = R_1(t) + R_2(t)$. Thus,

$$
\begin{aligned}
\textsc{Dec}(R)(\mathbf{t}) = & \sum_{t:\textsc{Dec}(t)=\mathbf{t}} (\textsc{rowdec}(t) \cdot (R_1(t), R_1(t), R_1(t))) \\
& \quad + (\textsc{rowdec}(t) \cdot (R_2(t), R_2(t), R_2(t)))) \\
= & \sum_{t:\textsc{Dec}(t)=\mathbf{t}} \textsc{rowdec}(t) \cdot (R_1(t), R_1(t), R_1(t)) \\
& + \sum_{t:\textsc{Dec}(t)=\mathbf{t}} \textsc{rowdec}(t) \cdot (R_2(t), R_2(t), R_2(t)) \\
= & \mathbf{R}_1(\mathbf{t}) + \mathbf{R}_2(\mathbf{t}) \\
= & Q(\mathbf{D})(\mathbf{t})
\end{aligned}
$$

Cross Product $Q \coloneqq Q_1 \times Q_2$: Consider a tuple $\mathbf{t}$ that is the result of joining tuples

$\mathbf{t}_1$ from the result of Q_1 and $\mathbf{t}_2$ from the result of Q_2. The annotation of a result tuple $\mathbf{t}$ of Q in $\mathbb{N}_{AU}$ relations are computed by multiplying the annotations of the input tuples $\mathbf{t}_1$ and $\mathbf{t}_2$ which are joined to form $\mathbf{t}$. Based on the induction hypothesis we know that $\mathbf{R}_1(\mathbf{t}) = \sum_{t \in T} \text{DEC}(t) \cdot (R_1(t), R_1(t), R_1(t))$ and $\mathbf{R}_2(\mathbf{t}) = \sum_{t \in T} \text{DEC}(t) \cdot (R_2(t), R_2(t), R_2(t))$. Recall that $\text{REWR}(Q_1 \times Q_2) = \text{REWR}(Q_1) \times \text{REWR}(Q_2)$. Using the fact that multiplication distributes over addition and that semiring operations are commutative and associativity we get:

$$\begin{aligned}\textsc{Dec}(R)(\mathbf{t}) = \sum_{t_1,t_2:\text{DEC}(t_1)=\mathbf{t}_1 \wedge \text{DEC}(t_2)=\mathbf{t}_2} &((\text{ROWDEC}(t_1) \cdot (R_1(t_1), R_1(t_1), R_1(t_1))) \\ &\cdot (\text{ROWDEC}(t_2) \cdot (R_2(t_2), R_2(t_2), R_2(t_2))))\end{aligned}$$

In the following let n_i denote $\text{ROWDEC}(t_i) \cdot (R_i(t_i), R_i(t_i), R_i(t_i))$ for $i \in \{1, 2\}$.

$$\begin{aligned}&= \sum_{t_1,t_2:\text{DEC}(t_1)=\mathbf{t}_1 \wedge \text{DEC}(t_2)=\mathbf{t}_2} (n_1 \cdot n_2) \\ &= \left(\sum_{t_1:\text{DEC}(t_1)=\mathbf{t}_1} n_1\right) \cdot \left(\sum_{t_2:\text{DEC}(t_2)=\mathbf{t}_2} n_2\right) \\ &= \mathbf{R}_1(\mathbf{t}_1) + \mathbf{R}_2(\mathbf{t}_2) \\ &= Q(\mathbf{D})(\mathbf{t})\end{aligned}$$

SG Combiner $Q \coloneqq \Psi(Q_1)$: Recall that Ψ merges the attribute bounds of tuples that agree on their SG values and sums their annotations. $\text{REWR}(\Psi(Q_1))$ groups input tuples on their SG values. Each group contains all tuples that agree with each other on SG attribute values. Then the minimum (maximum) over attributes storing attribute bounds is computed to calculate the value of attributes storing lower (upper) bounds for attributes. The values of attributes storing tuple annotations are computed by summing up the values of these attributes for each group.

Except $Q \coloneqq Q_1 - Q_2$: Recall the definition of set difference over $\mathcal{K}_{AU}$-relations:

$$(\mathbf{R}_1 - \mathbf{R}_2)(\mathbf{t})^{\downarrow} \coloneqq \Psi(\mathbf{R}_1)(\mathbf{t})^{\downarrow} -_{\mathcal{K}} \sum_{\mathbf{t} \simeq \mathbf{t}'} \mathbf{R}_2(\mathbf{t}')^{\uparrow}$$

$$(\mathbf{R}_1 - \mathbf{R}_2)(\mathbf{t})^{sg} \coloneqq \Psi(\mathbf{R}_1)(\mathbf{t})^{sg} -_{\mathcal{K}} \sum_{\mathbf{t}^{sg} = \mathbf{t}'^{sg}} \mathbf{R}_2(\mathbf{t}')^{sg}$$

$$(\mathbf{R}_1 - \mathbf{R}_2)(\mathbf{t})^{\uparrow} \coloneqq \Psi(\mathbf{R}_1)(\mathbf{t})^{\uparrow} -_{\mathcal{K}} \sum_{\mathbf{t} \equiv \mathbf{t}'} \mathbf{R}_2(\mathbf{t}')^{\downarrow}$$

Note that $\mathbf{t} \simeq \mathbf{t}'$ if the bounds of all attributes of $\mathbf{t}$ and $\mathbf{t}'$ overlap, i.e., the tuples may represent the same tuple in some world. Furthermore, $\mathbf{t} \equiv \mathbf{t}'$ if these tuples are equal and are certain ($\mathbf{t}.A^{\downarrow} = \mathbf{t}.A^{\uparrow}$ for all attributes A). Obviously, $\mathbf{t} \equiv \mathbf{t}' \Rightarrow \mathbf{t} \simeq \mathbf{t}'$. Since the rewrite for Ψ was proven to be correct above, we only need to prove that tuple annotations are calculated according to the definition repeated above. The first step of $\text{REWR}(Q)$ joins the results of $\text{REWR}(\Psi(Q_1))$ with $\text{REWR}(Q_2)$ based on overlap of their attribute bounds. Note that in the result of $\text{REWR}(\Psi(Q_1))$ each tuple $\mathbf{t}$ from $\Psi(Q_1)$ is encoded as a single tuple t. However, in R_2 (the result of $\text{REWR}(Q_2)$), each tuple $\mathbf{t}_2$ from $\mathbf{R}_2$ may be encoded as multiple value equivalent tuples. The annotations of these tuples multiplied with the their row annotation attributes sum up to the annotation $\mathbf{R}_2(\mathbf{t}_2)$:

$$\mathbf{R}_2(\mathbf{t}) = \sum_{t_2 : \text{DEC}(t_2) = \mathbf{t}_2} \text{ROWDEC}(t_2) \cdot (R_2(t_2), R_2(t_2), R_2(t_2))$$

For each such tuple $\mathbf{t}_i$ let $t_{i,1}, \ldots, t_{i,n_i}$ be the set of these value-equivalent tuples. Thus, the join will pair t with all such tuples for each tuple $\mathbf{t}_2$ for which $\mathbf{t} \simeq \mathbf{t}'$. After the join, the row annotation attributes of each RHS tuple t' paired with a tuple t is modified as follows: the lower bound is replaced with the upper bound (expression e_{cv}); the SG row annotation is retained unless t and t' do not agree on their SG attribute values (in this case the lower bound is set to 0); and the upper bound is set to the lower bound if t and t' are equal on all attributes and are certain (otherwise the upper bound is set to 0). Afterwards, tuples are grouped

based on their LHS values and the RHS row annotations are summed up. As shown above in the proof for cross product, the fact the one tuple $\mathbf{t}$ is encoded as multiple value-equivalent tuples is unproblematic for joins. The net effect is that for each LHS tuple t corresponding to a tuple $\mathbf{t}$ there exists one tuple t_{ext} in the result of the aggregation that (i) agrees with t on all attribute values and for which (ii) attribute $rrow^{\downarrow}$ stores $\sum_{\mathbf{t}':\mathbf{t}\simeq\mathbf{t}'} \mathbf{R}_2(\mathbf{t}')^{\uparrow}$, attribute $rrow^{sg}$ stores $\sum_{\mathbf{t}':\mathbf{t}^{sg}=\mathbf{t}'^{sg}} \mathbf{R}_2(\mathbf{t}')^{sg}$, and finally attribute $rrow^{\uparrow}$ stores $\sum_{\mathbf{t}\equiv\mathbf{t}'} \mathbf{R}_2(\mathbf{t}')^{\downarrow}$. In a last, step projection is used to calculate the annotation of t in the result corresponding to and selection is applied to remove tuples which certainly do not exist (their upper bound row annotation is 0).

Aggregation $Q \coloneqq \gamma_{G,f(A)}(Q_1)$: To demonstrate that the rewrite rule for aggregation is correct, we need to show that (i) tuples are grouped according to the default grouping strategy (Def. 30; (ii) that group-by bounds for each output tuple are calculated as defined in Def. 31; (iii) that aggregation function result bounds are computed following Def. 32); and (iv) that the multiplicity bounds for each output are correct (Def. 33 and (Def. 34).

Consider $Q_{gbounds}$ that is part of the rewrite for aggregation. This query groups the input on their SG group-by values and then calculates the group-by bounds for each group. Each group created in this way corresponds to one output group produced by the default grouping strategy (recall that this strategy defines one output group SG group-by value that exists in the input. Thus, (i) holds.

Note that according to Def. 30, the group-by attribute bounds for an output are determined as the minimum (maximum) value of the lower (upper) bound on a group-by attribute across all tuples that have the same SG group-by values as the output. This set of tuples corresponds exactly to one group in $Q_{gbounds}$. Since, Q_{bounds} computes group-by attribute bounds as the minimum (maximum) of the bounds of the input tuples belonging to a group, (ii) also holds.

The next step in the rewritten query is Q_{join} which pairs every tuple from $Q_{gbounds}$ with all input tuples that overlap in their group-by bounds with the output tuple's group-by bounds. Note that this is precisely the set $\eth(g)$ for output group g as iterated over in Def. 32 to calculate aggregation function result bounds. Recall that to calculate the lower bound of for an aggregation function result $\mathbf{t}_i.f(A)^{\downarrow}$ by summing up (in the aggregation monoid M) for each tuple in $\eth(g)$ either the minimum of $\mathbb{0}_M$ and $(\mathbf{R}(\mathbf{t}) *_{\mathbb{N}_{AU},M} \mathbf{t}.A)^{\downarrow}$ if $\text{UG}(G, \mathbf{R}, \mathbf{t})$ which is the case when either some group-by attributes of $\mathbf{t}$ are uncertain or if the tuple may not exit ($\mathbf{R}(\mathbf{t})^{\downarrow} = 0$). Otherwise, $(\mathbf{R}(\mathbf{t}) *_{\mathbb{N}_{AU},M} \mathbf{t}.A)^{\downarrow}$ is used instead. Expressions lba, sga, and uba used in the rewritten query implement this logic. Furthermore, it is trivial to see that expressions lba_f, sga_f, and uba_f implement the semantics of $*_{\mathbb{N}_{AU},M}$. Thus, the computation of the lower bound in the rewritten query correctly reflects the computation in $\mathbb{N}_{AU}$. Note that each input tuple $\mathbf{t}$ of the aggregation may be encoded as multiple value-equivalent tuples in the encoding. However, this is not a problem, according to 11 if $\vec{k} = \vec{k_1} +_{\mathbb{N}_{AU}} \vec{k_2}$, then $\vec{k} \circledast_M \vec{m} = \vec{k_1} \circledast_M \vec{m} +_{M_I} \vec{k_2} \circledast_M \vec{m}$. Thus, (iii) holds for lower bounds. The prove for upper bounds is symmetric. Finally, for the multiplicity bounds of tuples all input tuples that agree with an output on their SG group-by values are considered by the default grouping strategy. For the upper bound the bound bound on the multiplicity of these inputs is summed up. For the lower bound, the result of summing up the lower bound multiplicities for all tuples with certain group-by values (only these tuples are guaranteed to below to a group) is passed to $\delta_{\mathbb{N}}$ which returns 1 if the sum is non-zero and 0 otherwise. The SG and upper bound multiplicities computed in the same way except that all tuples are considered (and $\delta_{\mathbb{N}}$ is not used for the upper bound). In the rewritten query this is achieved by conditionally replacing the multiplicity bounds of tuples that are not part of the sum to 0 using a condition θ_c for the lower bound and $\theta_S G$ for the upper bound and SG multiplicities. $\delta_{\mathbb{N}}$ is implemented by replacing non-zero multiplicities with

1 and using aggregation function **max** instead of **sum** to combine the multiplicities of input tuples. Since the same multiplicity bounds are calculated by the rewritten query as for the query under $\mathbb{N}_{AU}$ semantics, (iv) holds. From (i) to (iv) follows that the aggregation rewrite is correct.

5.9.4 Optimizations for Joins. One potential performance bottleneck of query evaluation over AU-DBs is that joins may degenerate into cross products if the bounds of join attribute values are loose. As shown in the example below, in the worst case, each tuple from the LHS input may join with every tuple from the RHS leading to a join result whose size is quadratic in the input size. Even if most join attribute values are certain, the DBMS is likely to chose a nested loop join since we join on inequalities to test for overlap of join attribute bounds leading to $\mathcal{O}(n^2)$ runtime for the join.

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

(a) AU-DB relation **R**

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

(b) AU-DB relation **S**

A	**C**	$\underline{\mathbb{N}^3}$
[1/2/2]	[1/2/2]	(1,2,4)

(c) SGW result of $R \sqcup_{A=C} S$

A	**C**	$\underline{\mathbb{N}^3}$
[1/1/2]	[1/3/3]	(0,0,3)
[1/1/2]	[1/2/2]	(0,0,6)
[1/2/2]	[1/3/3]	(0,0,2)
[1/2/2]	[1/2/2]	(1,2,4)

(d) AU-DB result of $R \sqcup_{A=C} S$

Figure 5.7. Join rewriting without optimization

A	$\underline{\mathbb{N}^3}$
[1/1/1]	(0,2,2)
[2/2/2]	(0,1,1)

(a) $\textsc{split}^{sg}(R)$

C	$\underline{\mathbb{N}^3}$
[3/3/3]	(0,1,1)
[2/2/2]	(0,2,2)

(b) $\textsc{split}^{sg}(S)$

A	$\underline{\mathbb{N}^3}$
[1/1/2]	(0,0,3)
[1/2/2]	(0,0,2)

(c) $\textsc{split}^{\uparrow}(R)$

C	$\underline{\mathbb{N}^3}$
[1/3/3]	(0,0,1)
[1/2/2]	(0,0,2)

(d) $\textsc{split}^{\uparrow}(S)$

A	$\underline{\mathbb{N}^3}$
[1/1/2]	(0,0,5)

(e) $\textsc{Cpr}_{A,1}(\textsc{split}^{\uparrow}(R))$

C	$\underline{\mathbb{N}^3}$
[1/2/3]	(0,0,3)

(f) $\textsc{Cpr}_{C,1}(\textsc{split}^{\uparrow}(S))$

A	**C**	$\underline{\mathbb{N}^3}$
[2/2/2]	[2/2/2]	(0,2,2)
[1/1/2]	[1/2/3]	(0,0,15)

(g) $\textsc{opt}(R \sqcup_{A=C} S)$

Figure 5.8. Join rewriting with optimization

Example 25. *Figure 5.7 shows an example of joining two AU-DB relations* $R \bowtie_{A=C} S$*. The SGW result consists of a single tuple. However, the AU-DB result is a cross product of the two input tables since join attribute bounds of all tuples from* $\mathbf{R}$ *overlap with the join attribute bounds of all tuples from* $\mathbf{S}$*.*

In order to reduce the running time of joins, we introduce an optimized version of the rewrite rule for join. This optimization trades accuracy for performance by compressing the overestimation of possible answers encoded by the input relations of the join. We introduce an operator called *split* that splits the input relation $\mathbf{R}$ into two parts: $\text{SPLIT}^{sg}(R)$ encodes the SGW removing all attribute-level uncertainty and $\text{SPLIT}^{\uparrow}(\mathbf{R})$ encodes the over estimation of possible worlds encoded by $\mathbf{R}$. We will show that for any incomplete $\mathbb{N}$-relation $\mathcal{R}$ that is bounded by a $\mathbb{N}_{AU}$-relation $\mathbf{R}$, $\mathcal{R}$ is also bound by $\text{SPLIT}^{sg}(\mathbf{R}) \cup \text{SPLIT}^{\uparrow}(\mathbf{R})$. We define these two operations below. For each tuple, $\mathbf{t}$, $\text{SPLIT}^{sg}(\mathbf{R})$ contains a corresponding tuple $\mathbf{t}'$ from which all attribute uncertainty has been removed by setting $\mathbf{t}'.A^{\downarrow} = \mathbf{t}'.A^{sg} = \mathbf{t}'.A^{\uparrow} = \mathbf{t}.A^{sg}$. The annotation of such a tuple $\mathbf{t}'$ is determined as follows.: $\text{SPLIT}^{sg}(\mathbf{R})(\mathbf{t}')^{sg} = \mathbf{R}(\mathbf{t})^{sg}$, $\text{SPLIT}^{sg}(R)(\mathbf{t})^{\uparrow} = \mathbf{R}^{sg}(\mathbf{t})$, i.e., the overestimation of possible annotations is removed and $\text{SPLIT}^{sg}(R)(\mathbf{t})^{\downarrow} = \mathbf{R}(\mathbf{t})^{\downarrow}$ if the tuple's attribute values are all uncertain and to 0 otherwise. $\text{SPLIT}^{\uparrow}(\mathbf{R})$ retains the tuples from $\mathbf{R}$, keep the $\mathbf{R}(\mathbf{t})^{\uparrow}$ as the upper bound of a tuple's annotation, and sets $\text{SPLIT}^{\uparrow}(\mathbf{R})(\mathbf{t})^{\downarrow} = \text{SPLIT}^{\uparrow}(\mathbf{R})(\mathbf{t})^{sg} = 0$. Consider a AU-DB relation $\mathbf{R}$ with schema $\bar{A} = (A_1, \ldots, A_n)$. Let $\text{CERT}(\mathbf{t})$ denote the tuple derived from $\mathbf{t}$ by making all attribute values certain, i.e., replacing each attribute value $[c_1/c_2/c_3]$ with $[c_2/c_2/c_2]$.

$$\text{SPLIT}^{sg}(\mathbf{R})(\mathbf{t})^{\downarrow} \coloneqq \sum_{\mathbf{t}':\text{CERT}(\mathbf{t}')=\mathbf{t}} \begin{cases} \mathbf{R}(\mathbf{t}')^{\downarrow} & \text{if } \bigwedge_{i\in\{1,\ldots,n\}} \mathbf{t}'.A_i^{\downarrow} = \mathbf{t}'.A_i^{\uparrow} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{SPLIT}^{sg}(\mathbf{R}')(\mathbf{t})^{sg} = \text{SPLIT}^{sg}(\mathbf{R})(\mathbf{t})^{\uparrow} \coloneqq \sum_{\mathbf{t}':\text{CERT}(\mathbf{t}')=\mathbf{t}} \mathbf{R}(\mathbf{t})^{sg}$$

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

We rewrite the two split operators as shown below.

$$\begin{aligned}
\text{REWR}(\text{SPLIT}^{sg}(R)) &= \pi_{e_A, e_c \to row^{\downarrow}, row^{sg}, row^{sg} \to row^{\uparrow})}(\sigma_{row^{sg} > 0}(\mathbf{R})) \\
e_A &= {A_1}^{sg} \to {A_1}^{\downarrow}, {A_1}^{sg} \to {A_1}^{sg}, {A_1}^{sg} \to {A_1}^{\uparrow}, \ldots, \\
e_c &= \textbf{if} \bigwedge_{i \in \{1,\ldots,n\}} {A_i}^{\downarrow} = {A_i}^{\uparrow} \textbf{ then } row^{\downarrow} \textbf{ else } 0 \\
\text{REWR}(\text{SPLIT}^{\uparrow}(R)) &= \pi_{A^{\downarrow}, A, A^{\uparrow}, 0 \to row^{\downarrow}, 0 \to row^{sg}, row^{\uparrow}}(\mathbf{R})
\end{aligned}$$

Based on the following lemma, we can split any AU-DB relation $\mathbf{R}$ without loosing its bounding properties while preserving the SGW encoded by $\mathbf{R}$.

Lemma 13 (Split preserves bounds)**.** *Let* $\mathbf{R}$ *be a* $\mathbb{N}_{AU}$*-relation that bounds an incomplete* $\mathbb{N}$*-relation* $\mathcal{R}$*, then* $\text{SPLIT}^{sg}(\mathbf{R}) \cup \text{SPLIT}^{\uparrow}(\mathbf{R})$ *also bounds* $\mathcal{R}$*. Furthermore,* $\mathbf{R}^{sg} = \text{SPLIT}^{sg}(\mathbf{R}) \cup \text{SPLIT}^{\uparrow}(\mathbf{R})^{sg}$*.*

Proof: We first show that $\mathbf{R}^{sg} = \text{SPLIT}^{sg}(\mathbf{R}) \cup \text{SPLIT}^{\uparrow}(\mathbf{R})^{sg}$. Since, the SG annotations of $\text{SPLIT}^{\uparrow}(\mathbf{R})$ are zero and $\text{SPLIT}^{sg}(\mathbf{R})^{sg} = \mathbf{R}^{sg}$ by definition, the claim holds. Let $\mathbf{R}_{split} = \text{SPLIT}^{sg}(\mathbf{R}) \cup \text{SPLIT}^{\uparrow}(\mathbf{R})$. To demonstrate that the split operator preserves bounds, we have to show that for any possible world $\leftrightarrow \in \mathcal{R}$ we can extend a tuple matching $\mathcal{TM}$ based on which $\mathbf{R}$ bounds $\leftrightarrow$ to a tuple matching $\mathcal{TM}_{split}$ based on which $\mathbf{R}_{split}$ bounds $\leftrightarrow$. Consider a tuple $\mathbf{t}$ such that $\mathbf{R}(\mathbf{t}) \neq 0$. We distinguish two cases. If $\bigwedge_{i \in \{1,\ldots,n\}} \mathbf{t}.{A_i}^{\downarrow} = \mathbf{t}.{A_i}^{\uparrow}$, i.e., the tuple's attribute values are certain then

$$\mathbf{R}_{split}(\mathbf{t}) = \text{SPLIT}^{sg}(\mathbf{R})(\mathbf{t}) + \text{SPLIT}^{\uparrow}(\mathbf{R})(\mathbf{t}) = \\ (\mathbf{R}(\mathbf{t})^{\downarrow}, \sum_{\mathbf{t}' : \text{CERT}(\mathbf{t}') = \mathbf{t}} \mathbf{R}(\mathbf{t}')^{sg}, \sum_{\mathbf{t}' : \text{CERT}(\mathbf{t}') = \mathbf{t}} \mathbf{R}(\mathbf{t}')^{sg} + \mathbf{R}(\mathbf{t})^{\uparrow})$$

. Thus, for every tuple t we can set $\mathcal{TM}_{split}(\mathbf{t}, t) = \mathcal{TM}(\mathbf{t}, t)$ and wrt. $\mathbf{t}$ the tuple matching $\mathcal{TM}_{split}$ is fulfilling the requirements for being a tuple matching based on

which $\mathbf{R}_{spli}$ bounds $\leftrightarrow$ since its annotation include the bounds of $\mathbf{R}(\mathbf{t})$. Otherwise, at least one value of $\mathbf{t}$ is uncertain and $\mathbf{R}_{split}(\mathbf{t}) = \text{SPLIT}^{\uparrow}(\mathbf{R})(\mathbf{R}) = (0, 0, \mathbf{R}(\mathbf{t})^{\uparrow})$. Thus, we can safely set $\mathcal{TM}_{split}(\mathbf{t}, t) = \mathcal{TM}(\mathbf{t}, t)$ and $\mathcal{TM}_{split}$ is a tuple matching based on which $\mathbf{R}_{split}$ bounds $\leftrightarrow$.

To improve the performance of joins, we split both input relations of the join. We then employ another new operator $\text{CPR}_{A,n}$ that compresses the output of $\text{SPLIT}^{\uparrow}$ into a relation with n tuples by grouping the input tuples into n buckets based on their A values. For that we split the range of A values appearing in the input into n buckets containing roughly the same number of values each. All tuples from a bucket are aggregated into a single result tuple by merging their attribute bounds and summing up their annotations. Let $\mathbf{t}_b$ denote the tuple constructed for bucket b in this fashion. Let $B = \{b_1, \ldots, b_n\}$ be the set of buckets for $\text{CPR}_{A,n}(\mathbf{R})$.

$$\text{CPR}_{A,n}(\mathbf{R})(\mathbf{t}) = \begin{cases} (0, 0, \sum_{\mathbf{t}' \in b} \mathbf{R}(\mathbf{t}')^{\uparrow}) & \text{if } \exists b \in B : \mathbf{t} = \mathbf{t}_b \\ 0 & \text{otherwise} \end{cases}$$

We rewrite the CPR operator as shown below. Assume that the bucket b_i covers the interval $[l_i, u_i]$ from the domain of A.

$$\begin{aligned} \text{REWR}(\text{CPR}_{A,n}(\mathbf{R})) &= \pi_{\bar{A}^{\downarrow}, \bar{A^{sg}}, \bar{A}^{\uparrow}, 0 \to row^{\downarrow}, 0 \to row^{sg}, row^{\uparrow}}(Q_{merge}) \\ Q_{merge} &\coloneqq \gamma_{B, e_{merge}, \mathbf{sum}(row^{\uparrow}) \to row^{\uparrow}}(\mathbf{R}) \\ e_{merge} &= min(A_1{}^{\downarrow}) \to A_1{}^{\downarrow}, min(A_1{}^{sg}) \to A_1{}^{sg}, max(A_1{}^{\uparrow}) \to A_1{}^{\uparrow}, \ldots \\ e_{bucketize} &= \mathbf{if}\ A^{\uparrow} \geq l_1 \wedge A^{\downarrow} \leq u_1\ \mathbf{then}\ 1\ \mathbf{else}\ (\mathbf{if} \ldots \end{aligned}$$

Note that CPR does not preserve the SGW encoded by its input. This is not problematic, because we only apply CPR to the output of $\text{SPLIT}^{\uparrow}$ for which the SG annotation of each tuple is zero anyways.

Lemma 14 (CPR preserves bounds)**.** *Let* $\mathbf{R}$ *be a* $\mathbb{N}_{AU}$*-relation that bounds an incomplete* $\mathbb{N}$*-relation* $\mathcal{R}$*,* $n \in \mathbb{N}$*, and* $A \in \text{Sch}(\mathbf{R})$*, then* $\text{CPR}_{A,n}(\mathbf{R})$ *also bounds* $\mathcal{R}$*.*

Proof: Let $\leftrightarrow$ be one possible world of $\mathcal{R}$. Let $\mathcal{TM}$ be a tuple matching based on which $\mathbf{R}$ bounds $\leftrightarrow$. Furthermore, let $\mathbf{t}_r$ denote the result tuple produced for a bucket b. We construct a tuple matching $\mathcal{TM}_{\text{CPR}}$ which bounds $\mathcal{R}$ by setting for all buckets b and tuple t from $\leftrightarrow$:

$$\mathcal{TM}_{\text{CPR}}(\mathbf{t}_b, t) = \sum_{\mathbf{t} \in b} \mathcal{TM}(\mathbf{t}, t)$$

Since by definition $\text{CPR}_{A,n}(\mathbf{R})(\mathbf{t}_b) = \sum_{\mathbf{t} \in b} \mathbf{R}(\mathbf{t})$ and since $t \sqsubseteq \mathbf{t} \Rightarrow t \sqsubseteq \mathbf{t}_b$ for all $\mathbf{t} \in b$ (because the attribute bounds of $\mathbf{t}_b$ are defined to cover the attribute bounds of all tuples from bucket b), we have that $\mathcal{TM}_{\text{CPR}}$ is a tuple matching based on which $\text{CPR}_{A,n}(\mathbf{R})$ bounds $\leftrightarrow$.

Our optimized rewrite for join first splits both inputs of the join. We join $\text{SPLIT}^{\uparrow}$ and SPLIT^{sg} separately and then union the result. For a join with condition θ, let θ^{sg} denote the result of replacing references to an attribute A with A^{sg}. Note that for SPLIT^{sg} since all attribute values are certain, we can apply θ^{sg} instead of having to apply range-anntotated expression evaluation. Thus, the join over SPLIT^{sg} will result only in minimal overhead compared to a regular join. Since $\text{SPLIT}^{\uparrow}(\mathbf{R}) \sqcup \text{SPLIT}^{\uparrow}(\mathbf{S})$ may potentially produce a large number of results, we apply CPR to the inputs to bound the size of the join result. Thus, we control the worst case join result size by setting the parameter n of CPR. In principle we can compress on any attribute of the input relations. However, it is typically better to choose attributes that are referenced in θ, e.g., for a condition $A = B$ if we compress on A respective B using n buckets with the same bucket boundaries for A and B, then the join result will contain at most n results since each tuple from $\text{CPR}_{A,n}(\mathbf{R})$ will join at most with one tuple from $\text{CPR}_{B,n}(\mathbf{S})$. Let A (B) denote an attribute from Q_1 (Q_2) that appears in θ, preferably in an equality comparison. The optimized rewrite for join $\text{OPT}(\cdot)$ using

A and B is defined below.

$$\text{OPT}(Q_1 \sqcup_\theta Q_2) \coloneqq Q_{sg} \cup Q_{pos}$$

$$\begin{aligned}
Q_{sg} &\coloneqq \pi_{\bar{A},\bar{B},\bar{A}^\downarrow,\bar{B}^\downarrow\bar{A}^\uparrow,\bar{B}^\uparrow,e_c,e_{sg},e_p}(\\
&\quad \text{REWR}(\text{SPLIT}^{sg}(Q_1)) \sqcup_{\theta^{sg}} \text{REWR}(\text{SPLIT}^{sg}(Q_2)))
\end{aligned}$$

$$\begin{aligned}
Q_{pos} &\coloneqq \text{REWR}(\text{CPR}_{A,n}(\text{SPLIT}^{\uparrow}(Q_1)) \sqcup_\theta \text{CPR}_{B,n}(\text{SPLIT}^{\uparrow}(Q_2))) \\
e_c &\coloneqq Q_1.row^\downarrow \cdot Q_2.row^\downarrow \to row^\downarrow \\
e_{sg} &\coloneqq Q_1.row^{sg} \cdot Q_2.row^{sg} \to row^{sg} \\
e_p &\coloneqq Q_1.row^\uparrow \cdot Q_2.row^\uparrow \to row^\uparrow
\end{aligned}$$

Lemma 15 (The optimized rewrite preserves bounds)**.** *The optimized join rewrite is correct, i.e., for any $\mathbb{N}_{AU}$-database and query $Q \coloneqq Q_1 \sqcup_\theta Q_2$, let $Q_{optMerge}$ be query Q_{merge}, but using* $\text{OPT}(\cdot)$ *instead of* $\text{REWR}(\cdot)$ *for joins. Then,*

$$\text{DEC}(Q_{merge}(\text{ENC}(\mathbf{D}))) \preceq_I \text{DEC}(Q_{optMerge}(\text{ENC}(\mathbf{D})))$$

Proof: Based on Lem. 13 and Lem. 14 we know that the split and compression operators preserve bounds. As mentioned above, all tuples are attribute-level certain in the result of SPLIT^{sg}. Thus,

$$\text{REWR}(\text{SPLIT}^{sg}(R)) \sqcup_{\theta^{sg}} \text{REWR}(\text{SPLIT}^{sg}(S)))$$

and

$$\text{REWR}(\text{SPLIT}^{sg}(R) \sqcup_\theta \text{SPLIT}^{sg}(S))$$

are equivalent. Since the rewrite for union is returning of the rewritten inputs we know that $\text{OPT}(Q_1 \sqcup_\theta Q_2)$ is equivalent to $\text{REWR}(Q_{sg'} \cup Q_{pos'})$ where $Q_{sg'}$ and $Q_{pos'}$ are defined as shown below.

$$\text{REWR}(Q_{sg'} \cup Q_{pos'})$$

$$Q_{sg'} \coloneqq \text{SPLIT}^{sg}(Q_1) \ \sqcup_{\ \theta^{sg}} \text{SPLIT}^{sg}(Q_2)$$

$$Q_{pos'} \coloneqq \text{CPR}_{A,n}(\text{SPLIT}^{\uparrow}(Q_1)) \ \sqcup_{\ \theta} \text{CPR}_{B,n}(\text{SPLIT}^{\uparrow}(Q_2))$$

Using the fact that join distributes over union, we can rewrite $Q_{sg'} \cup Q_{pos'}$ into:

$$(\text{SPLIT}^{sg}(Q_1) \cup \text{SPLIT}^{\uparrow}(Q_1)) \ \sqcup_{\ \theta} (\text{SPLIT}^{sg}(Q_2) \cup \text{SPLIT}^{\uparrow}(Q_2))$$

Since replacing Q with $\text{SPLIT}^{sg}(Q) \cup \text{SPLIT}^{\uparrow}(Q)$ preserves bounds, it follows that $\text{OPT}(\cdot)$ is correct.

Example 26. *Fig. 5.8 shows the result of joining the two tables from Fig. 5.7 using the optimized rewrite. By sacrificing accuracy, the number of result tuples can be reduced by limiting the number of output tuples.*

5.9.5 Optimization for Aggregation. Similar to the optimization for joins, the self-join used in the rewrite for aggregation to determine which tuples could possibly belong to a group may also degenerate into a cross product if the attribute-level bounds are loose, resulting in a potential performance bottleneck. We now introduce an optimized version of the rewrite rule for aggregation. This optimization improves over the naive aggregation rewrite in two aspects: (i) we piggy-back the computation of SG aggregation function results on the computation of output groups ($Q_{gbounds}$) and (ii) we use the compression operator used in the optimize join rewrite to compress the RHS of subquery Q_{join} from the naive aggregation rewrite.

$$
\begin{aligned}
\text{OPT}(\gamma_{G,f(A)}(R)) &\coloneqq \gamma_{'f(A^{sg})',G^{sg},G^{\uparrow},G^{\downarrow},e_{aggbounds}}(Q_{proj}) \\
e_{aggbounds} &\coloneqq f(A^{\uparrow}), f(A^{\downarrow}), \\
&\quad\ \mathbf{max}(row^{\downarrow}) \to row^{\downarrow}, \\
&\quad\ \mathbf{max}(row^{sg}) \to row^{sg}, \\
&\quad\ \mathbf{sum}(row2^{\uparrow}) \to row^{\uparrow}
\end{aligned}
$$

$$
Q_{proj} \coloneqq \pi_{'f(A^{sg})',G^{sg},G^{\uparrow},G^{\downarrow},lba,uba,row^{\downarrow},row^{sg},row2^{\uparrow}}(Q_{join})
$$

$$
\begin{aligned}
Q_{join} &\coloneqq Q_{gbounds} \ \sqcup_{\ \theta_{join}} \text{CPR}_{A,n}(\rho_{e_{rename}}(\text{REWR}(R))) \\
\theta_{join} &\coloneqq \bigwedge_{A_i \in G} A_i^{\uparrow} \geq B_i^{\downarrow} \wedge B_i^{\uparrow} \geq A_i^{\downarrow} \\
e_{rename} &\coloneqq A_1 \to B_1, \ldots, A_n \to B_n, row \to row2
\end{aligned}
$$

$$
\begin{aligned}
Q_{gbounds} &\coloneqq \gamma_{G^{sg},e_{gbounds},\mathbf{max}(e_c)\to row^{\downarrow},\mathbf{max}(row^{sg})\to row^{sg}}(\text{REWR}(R)) \\
e_c &\coloneqq \mathbf{if}\ \theta_c \wedge row^{\downarrow} > 0\ \mathbf{then}\ 1\ \mathbf{else}\ 0 \\
\theta_c &\coloneqq \bigwedge_{A_i \in G} A_i^{\uparrow} = B_i^{\uparrow} \wedge = A_i^{\downarrow} = B_i^{\downarrow} \wedge A_i^{\downarrow} = A_i^{\uparrow} \\
e_{gbounds} &\coloneqq e_{bound}^{A_1}, \ldots, e_{bound}^{A_k}, sga \qquad\qquad (\text{for } G = (A_1, \ldots, A_k)) \\
e_{bound}^{A} &\coloneqq \mathbf{max}(A^{\uparrow}) \to A^{\uparrow}, \mathbf{min}(A^{\downarrow}) \to A^{\downarrow}
\end{aligned}
$$

As mentioned above, the main goal of the optimization is to decrease the number of input tuples for the join operation. Note how in the optimized rewrite shown above, a compression operator is applied on the RHS input of the join which limits the number of tuples that need to be matched to the output groups produced by the LHS. Because we still need to produce the correct SG aggregation results for each output group, we now need to perform this when calculating output groups in the LHS since compression may group multiple SG groups into one tuple in the

compressed RHS which makes it impossible to compute the precise SG aggregation function results from the RHS. There are also some minor additional modifications like pre-computing the certain/selected guess tuple multiplicities.

Lemma 16 (The optimized rewrite preserves bounds)**.** *The optimized aggregation rewrite is correct, i.e., for any* $\mathbb{N}_{AU}$*-database and query* $Q \coloneqq \gamma_{G,f(A)}(Q_1)$*, let* $Q_{optMerge}$ *be query* Q_{merge}*, but using* OPT$(\cdot)$ *instead of* REWR$(\cdot)$ *for aggregation. Then*

$$\textsc{Dec}(Q_{merge}(\textsc{Enc}(\mathbf{D}))) \preceq_I \textsc{Dec}(Q_{optMerge}(\textsc{Enc}(\mathbf{D})))$$

Proof: We already proved that the naive aggregation rewrite preserves bounds (Thm. 13). The optimized rewrite computes BestGuess() results in the LHS of the join. However, this does not affect the computed values. We also apply the compression operator to compress the RHS. As shown in Lem. 14, this operator preserves bounds. Consider two tuples t_{left} and t_{right} from the LHS and RHS of the naive rewrite that are joined by Q_{join}. In the optimized rewrite, t_{right} may have been merged with multiple other tuples into a compressed tuple $t_{compressed}$. By definition of Cpr, the attribute bounds of $t_{compress}$ include the attribute bounds of t_{right}. Since the join condition of Q_{join} is the same in the naive and optimized versions of the aggregation rewrite, we know that t_{left} joins with $t_{compress}$. Using the fact Cpr is bound preserving it follows that the aggregation function result bounds computed by the optimized aggregation rewrite have to include the bounds produced by the naive rewrite. The group-by bounds are computed in the same way in both cases and, thus, both rewrite compute the same group-by bounds for each result tuple. Thus, it follows that the optimized aggregation rewriting is correct.

5.10 Native Algorithms

We now introduce optimized algorithms for ranking and windowed aggregation over AU-DBs that are more efficient than their SQL counterparts presented in Sec. 5.9.

Through a *connected heap* data structure, these algorithms leverage the fact that lower and upper position bounds are typically close approximations of one another to avoid performing multiple passes over the data. We assume a physical encoding of an AU-DB relation $\mathbf{R}$ as a classical relation [40] where each range-annotated value of an attribute A is stored as three attributes $A^{\downarrow}$, A^{sg}, and $A^{\uparrow}$. In this encoding, attributes $\mathbf{t}.\#^{\downarrow}$, $\mathbf{t}.\#^{sg}$, and $\mathbf{t}.\#^{\uparrow}$ store the tuple's multiplicity bounds.

5.10.1 Non-deterministic Sort, Top-k. Algorithm 1 sorts an input AU-DB $\mathbf{R}$. The algorithm assigns to each tuple its position τ given as lower and upper bounds: $\mathtt{t}.\tau^{\downarrow}, \mathtt{t}.\tau^{\uparrow}$, respectively. [10] Given a parameter $\mathtt{k}$, the algorithm can also be used to find the top-$\mathtt{k}$ elements; otherwise we set $\mathtt{k} = |\mathbf{R}|$ (the maximal possible size of the input relation). Algorithm 1 (Fig. 5.9) takes as input the relational encoding of an AU-DB relation $\mathbf{R}$ sorted on $O^{\downarrow}$, the lower-bound of the sort order attributes. Recall from Equation (5.25) that to determine a lower bound on the sort position of a tuple $\mathbf{t}$ we have to sum up the smallest multiplicity of tuples $\mathbf{s}$ that are certainly sorted before $\mathbf{t}$, i.e., where $\mathbf{s}.O^{\uparrow} <_O^{total} \mathbf{t}.O^{\downarrow}$. Since $\mathbf{s}.O^{\downarrow} <_O^{total} \mathbf{s}.O^{\uparrow}$ holds for any tuple, we know that these tuples are visited by Algorithm 1 before $\mathbf{t}$. We store tuples in a min-heap `todo` sorted on $O^{\uparrow}$ and maintain a variable $\mathtt{rank}^{\downarrow}$ to store the current lower bound. For every incoming tuple $\mathbf{t}$, we first determine all tuples $\mathbf{s}$ from `todo` certainly preceding $\mathbf{t}$ ($\mathbf{s}.O^{\uparrow} < \mathbf{t}.O^{\downarrow}$) and update $\mathtt{rank}^{\downarrow}$ with their multiplicity. Since $\mathbf{t}$ is the first tuple certainly ranked after any such tuple $\mathbf{s}$ and all tuples following $\mathbf{t}$ will also certainly ranked after $\mathbf{s}$, we can now determine the upper bound on $\mathbf{s}$'s position. Based on Equation (5.27) this is the sum of the maximal multiplicity of all tuples that may precede $\mathbf{s}$. These are all tuples $\mathbf{u}$ such that $\mathbf{s}.O^{\uparrow} \geq \mathbf{u}.O^{\downarrow}$, i.e., all tuples we have processed so far. We store the sum of the maximal multiplicity of these tuples in a variable $\mathtt{rank}^{\uparrow}$ which is updated for every incoming tuple. We use a function

[10] The selected guess position τ^{sg} is trivially obtained using an additional heap, and omitted here for clarity.

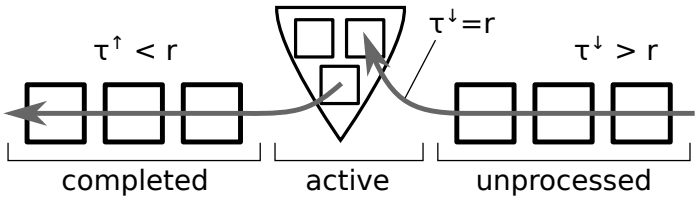


Figure 5.9. The lifecycle of tuples in Algorithm 1

`emit` to compute $\mathbf{s}$'s upper bound sort position, adapt $\mathbf{s}.\#^{\downarrow}$ (for a top-k query, $\mathbf{s}$ may not exist in the result if its position may be larger than k), add $\mathbf{s}$ to the result, and adapt $\texttt{rank}^{\downarrow}$ (all tuples processed in the following are certainly ranked higher than $\mathbf{s}$). Function `split` splits a tuple with $\mathbf{t}.\# > 1$ into multiple tuples as required by Def. 36. If we are only interested in the top-k results, then we can stop processing the input once $\texttt{rank}^{\downarrow}$ is larger than k, because all following tuples will be certainly not in the top-k. Once all inputs have been processed, the heap may still contain tuples whose relative sort position wrt. to each other is uncertain. We flush these tuples at the end.

Algorithm 2 defines function $split(\mathbf{t})$ which split multiplicities of range tuple $\mathbf{t}$ into multiplicity of ones using the semantics in Fig. 5.4.

Complexity Analysis: Let $n = |\mathbf{R}|$. The algorithm requires $O(n \cdot \log n)$ to sort the input. It then processes the data in one-pass. For each tuple, we compare $\mathbf{t}.O^{\uparrow}$ in $O(1)$ with the root of the heap and insert the tuple into the heap in $O(\log |heap|)$. Tuples are removed from the heap just once in $O(\log |heap|)$. In the worst-case, if the sort positions of all tuples may be less than k, then the heap will contain all n tuples at the end before flushing. Thus, $|heap|$ is bound n and we get $O(n \cdot \log n)$ as the worst-case runtime complexity for our algorithm requiring $O(n)$ memory. However, in practice, heap sizes are typically much smaller.

5.10.2 Connected Heaps. In our algorithm for windowed aggregation that we will present in Sec. 5.10.3, we need to maintain the tuples possibly in a window ordered increasingly on $\tau^{\uparrow}$ (for fast eviction), sorted on $A^{\downarrow}$ to compute $\mathsf{min\text{-}k}(\mathbf{R}, \mathbf{t}, A)$, and

Input: $\mathbf{R}$ (sorted on $O^{\downarrow}$), $\mathtt{k} \in \mathbb{N}$ (or $\mathtt{k} = |\mathbf{R}^{\uparrow}|$)

```
1  todo ← minheap(O↑) ; rank↓ ← 0 ; rank↑ ← 0 ; res ← ∅
2  for t ∈ R do
3      while todo.peek().O↑ < t.O↓ do                    // emit tuples
4          emit(todo.pop())
5          if rank↓ > k then                  // tuples certainly out of top-k?
6              return res
7      t.τ↓ ← rank↓                                  // set position lower bound
8      todo.insert(t)                                  // insert into todo heap
9      rank↑ += t.#↑                                // update position upper bound
10 while not todo.isEmpty() do                        // flush remaining tuples
11     emit(todo.pop())
12 return res

13 def emit(s)
14     s.τ↑ ← min(k, rank↑)                 // position upper bound capped at k
15     if rank↑ > k then                    // s may not be in result if s.τ↑ > k
16         s.#↓ ← 0
17     res ← res ∪ split({s})
18     rank↓ += s.#↓                                // update position lower bound
```

Algorithm 1: Non-deterministic sort (top-k) on attributes O

sorted decreasingly on $A^{\uparrow}$ to compute $\mathsf{max\text{-}k}(\mathbf{R}, \mathbf{t}, A)$. We could use separate heaps to access the smallest element(s) wrt. to any of these orders efficiently. However, if a tuple needs to be deleted, the tuple will likely not be the root element in all heaps which means we have to remove non-root elements from some heaps which is inefficient (linear in the heap size). Of course it would be possible to utilize other data structures that maintain order such as balanced binary trees. However, such data structures do not achieve the $O(1)$ lookup performance for the smallest element that heaps provide. Furthermore, trees are typically are not as efficient in practice

```
1 Function split(t):
2     for i ∈ [1, t.t.#↑] do
3         t_i = copy(t) ;
4         t_i.t.#^sg = t.t.#^sg < i : 1?0 ;
5         t_i.t.#↑ = 1 ;
6         t_i.t.#↓ = t.t.#↓ < i : 1?0 ;
7         t_i.τ↑+ = i ;
8         t_i.τ↓+ = i ;
9         return t_i ;
```

Algorithm 2: Split bag tuple

as heaps which can be implemented as arrays. Instead, we introduce a simple, yet effective, data structure we refer to as a *connected heap*.

A *connected heap* is comprised of H heaps which store pointers to a shared set of records. Each heap has its own sort order. A record stored in a connected heap consists of a tuple (the payload) and H backwards pointers that point to the nodes of the individual heaps storing this tuple. These backward pointers enable efficient deletion ($O(H \cdot \log n)$) of a tuple from all heaps when it is popped as the root of one of the component heaps. When a tuple is inserted into a connected heap, it is inserted into each component heap in $O(\log n)$ in the usual way with the exception that the backwards pointers are populated.

Preliminary experiments: To measure the impact of the backpointers in connected heaps on performance, we did a preliminary experimental comparison with using a set of independent heaps. Without the backlinks, removing an non-root element from a heap is linear in the size of the heap in the worst-case, because it may require a search over the whole heap to find the position of such an element. Afterwards, the element can be deleted and the heap property can be restored in $O(\log n)$. Using the backlinks, finding the positions of an element in other heaps is $O(1)$ and so popping the root

element of one heap and removing it from all other heaps is in $O((\log n) \cdot m)$ where n is the size of the largest heap and m is the number of heaps. The table below shows the execution times in milliseconds using connected heaps (back pointers) versus classical unconnected heaps (linear search). This experiment was run on a database with 50k tuples and 1%-5% uncertainty (amount of tuples that are uncertain) varying the size of the ranges for the attribute we are aggregating over. The main factor distinguishing linear search performance from back pointers is the heap size which for our windowed aggregation algorithm is affected by attribute range size, percentage of tuples which have uncertain order-by and data size. Even though in this experiment the amount of uncertain data and database size are quite low, we already see 25% up to a factor of ~ 10 improvement. For larger databases or larger percentage of uncertain data, the sizes of heaps will increase and, thus, we will see even more significant performance improvements.

Table 5.7. Back pointer performance

Uncert	**Range**	**Connected heaps (Back pointers)(ms)**	**Unconnected heaps (Linear search)(ms)**
1%	2000	1979.272	3479.042
1%	15000	2045.162	6676.732
1%	30000	2103.974	9646.330
5%	2000	1976.651	4078.487
5%	15000	2149.990	15186.657
5%	30000	2191.823	22866.713

Deletion from a connected heap: When a node is popped from one of the component heaps the nodes of the other heaps storing the tuple are identified in $O(H)$ using the backwards pointers. Like in standard deletion of nodes from a heap, a deleted node is replaced with the right-most node at the leaf level. Standard sift-down and

sift-up are then used to restore the heap property in $O(\log n)$. Recall that the heap property for a min-heap requires that for each node in the heap its value is larger than the value of its parent. Insertion of a new node v into a heap places the new element at the next free position at the leaf level. This may violate the heap property. The heap property can be restored in $O(\log n)$ using sift-up (repeatedly replacing a node with its parent). To delete the root of a heap, we replace the root with the right-most child. This may violate the heap property if the new root is larger than one of its children. The heap property can be restored in $O(\log n)$ steps using sift-down, i.e., replacing a node that is larger than a child with the smaller of its children. For a connected heap, deletion may cause a node to be deleted that is currently not the root of the heap. Like in standard heaps, we replace the node v to be deleted with the right-most node v_l from the leaf level. This may violate the heap property (every child is larger than its parent) in two possible ways: either v_l is smaller than the parent of v or v_l is larger than one of the children of v. Note that it is not possible for both cases to occur at the same time, because the heap was valid before and, thus, if v_l is larger than a child of v, then it has to be larger than the parent of v. If v_l is smaller than the parent of v, then it has to be smaller than all other nodes in the subtree rooted at v. We can restore the heap property by sifting up v_l. Now consider the case where v_l is larger than one of the children of v and let v_c denote that child (or the smaller child if v_l is larger than both children). Note that the subtree rooted at v was a valid heap. Thus, replacing v with v_l is replacing the root element of this subheap and the heap property for the subheap can be restored using sift-down. Since v_l is larger than the parent of v this restores the heap property for the whole heap.

Example 27 (Connected heap)**.** *Consider the connected heap shown below on the left storing tuples* $t_1 = (1, 3)$*,* $t_2 = (2, 6)$*,* $t_3 = (3, 2)$*, and* $t_4 = (4, 1)$*. Heap* h_1 *(*h_2*) is sorted on the first (second) attribute. Calling* `pop()` *on* h_1 *removes* t_1 *from* h_1*. Using*

the backwards pointer from t_1 to the corresponding node in h_2 (shown in red), we also remove t_1 from h_2. The node pointing to t_1 from h_2 is replaced with the right most leaf node of h_2 (pointing to t_2). In this case the heap property is not violated and, thus, no sift-down / up is required.

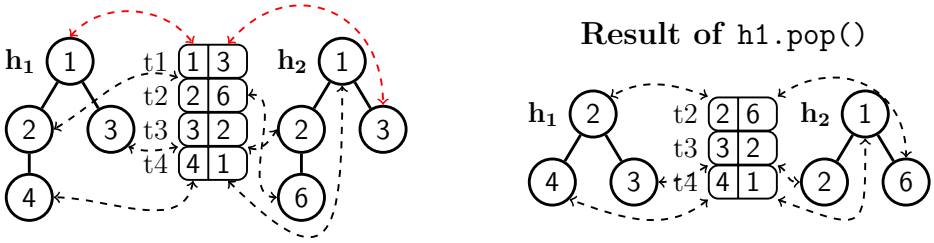


Figure 5.10. ex. 27

5.10.3 Ranged Windowed Aggregation. Without loss of generality, we focus on window specifications with only a `ROWS PRECEDING` clause; a `FOLLOWING` clause can be simulated by offsetting the window, i.e., a window bound of $[-N, 0]$. Algorithm 3 uses a function `compBounds` to compute the bounds on the aggregation function result based on the certain and possible content of a window. We discuss the code for these functions below for aggregation functions **min**, **max**, and **sum** (**count** uses the same algorithm as **sum** using [1/1/1] instead of the values of an attribute A). Algorithm 3 follows a sweeping pattern similar to Algorithm 1 to compute the windowed aggregate in a single pass over the data which has been preprocessed by applying $\text{SORT}_{O \to \tau}(\mathbf{R})$ and then has been sorted on $\tau^{\downarrow}$. The algorithm uses a minheap `openw` which is sorted on $\tau^{\uparrow}$ to store tuples for which have not seen yet all tuples that could belong to their window. Additionally, the algorithm maintains the following data structures: `cert` is a map from a sort position i to a tree storing tuples $\mathbf{t}$ that certainly exist and for which $\mathbf{t}.\tau^{\downarrow} = i$ sorted on $\tau^{\uparrow}$. This data structure is used to determine which tuples certainly belong to the window of a tuple; $(\texttt{poss}, \texttt{pagg}^{\downarrow}, \texttt{pagg}^{\uparrow})$ is a connected minheap where `poss`, $\texttt{pagg}^{\downarrow}$, and $\texttt{pagg}^{\uparrow}$ are sorted on $\tau^{\uparrow}$, $A^{\downarrow}$, $-A^{\uparrow}$, respectively. This connected heap

stores tuples possibly in a window. The different sort orders are needed to compute bounds on the aggregation function result for a window efficiently (we will expand on this later). Finally, we maintain a watermark $\texttt{c-rank}^{\downarrow}$ for the lower bound position of the certain part of windows.

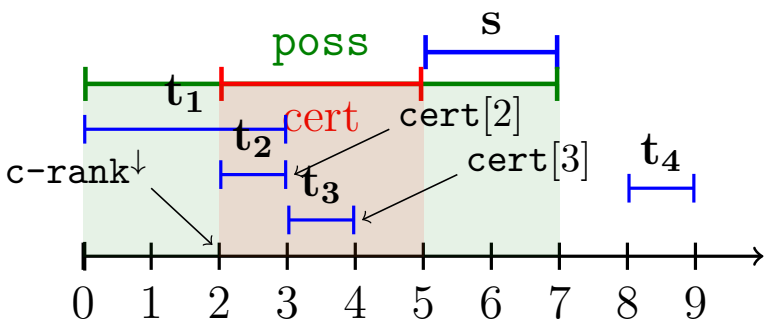


Figure 5.11. Example state for Algorithm 3, N=5, $\texttt{c-rank}^{\downarrow}$=2.

Input: f, X,O, N, A, $\text{SORT}_{O \to \tau}(\mathbf{R})$ **sorted on** $\tau^{\downarrow}$

```
1  openw ← minheap(τ↑)                                   // tuples with open windows
2  cert ← Map(int, Tree(τ↑))                             // certain window members by pos.
3  (poss, pagg↓, pagg↑) ← connected-minheap(τ↑, A↓, A↑)
4  c-rank↓ ← 0                                           // watermark for certain window
5  res ← ∅
6  for t ∈ R do
7      openw .insert(t)
8      if t.#↓ > 0 then                                  // insert into potential certain window
9          cert[t.τ↓].insert(t)
10     while openw .peek().τ↑ < t.τ↓ do                  // close windows
11         s ← openw .pop()
12         while c-rank↓ < s.τ↑ − N do                   // evict certain win.
13             cert[c-rank↓] = null
14             c-rank↓ + +
15         s.X ← compBounds (f, s, cert , poss )         // compute agg.
16         while poss .peek.τ↑ < s.τ↓ − N do             // evict poss. win.
17             poss .pop()
18         res ← res ∪ {s}
19     poss .insert(t)                                   // insert into poss. win.
```

Algorithm 3: Aggregate $f(A) \to X$, sort on O, N precedings

```
1  def compBounds(f, t, cert, poss)                    // compute bounds on sum(A)
2      if f = sum then
3          return computeSumBounds(t,cert, poss)
4      if f = min then
5          return computeMinBounds(t,cert, poss)
6      if f = max then
7          return computeMaxBounds(t,cert, poss)
8      if f = count then
9          return computeCountBounds(t,cert, poss)
10     if f = avg then
11         return computeMinBounds(t,cert, poss)
```

Algorithm 4: Computing bounds for $f(A) \rightarrow X$ for tuple $\mathbf{t}$

```
1  def ComputeSumBounds(t, cert , poss )          // compute bounds on sum(A)
2      n ← N − 1; X↓ ← t.A↑; X↑ ← t.A↑                          // pos.  and bounds
3      for x ∈ [t.τ↑ − N, t.τ↓] do
4          for s ∈ cert [x] do
5              if s.τ↑ ≤ t.τ↓ then                  // belongs to cert.  window of s
6                  X↑+ = s.A↑; X↓+ = s.A↓
7                  n − −
8              else break ;
9      lb_poss ← copy(pagg↓); ub_poss ← copy(pagg↑)
10     n_lb ← n; n_ub ← n              // max.  num.  of tuples possibly in win.
11     while n_lb > 0 ∧ ¬lb_poss.isEmpty() do                          // compute X↓
12         s ← lb_poss.pop()
13         if s.A↓ < 0 then                  // only values < 0 contribute to X↓
14             X↓+ = s.A↓
15             n_lb − −
16         else break ;
17     while n_ub > 0 ∧ ¬ub_poss.isEmpty() do                          // compute X↑
18         s ← ub_poss.pop()
19         if s.A↑ > 0 then                  // only values > 0 contribute to X↑
20             X↑+ = s.A↑
21             n_ub − −
22         else break ;
23     return [X↓, X↑]
```

Algorithm 5: Computing bounds for $\mathbf{sum}(A) \rightarrow X$ for tuple $\mathbf{t}$

```
1 def computeMinBounds(t, cert , poss )          // compute bounds on min(A)
2     n ← N − 1; X↓ ← t.A↑; X↑ ← t.A↑                          // pos.  and bounds
3     for x ∈ [t.τ↑ − N, t.τ↓] do
4         if cert [x] then                          // min of certain lower-bound
5             X↑ ← x
6             break
7     X↓ = pagg↓.peek().A↓                        // min of possible lower-bound
8     return [X↓, X↑]
```

Algorithm 6: Computing bounds for $\mathbf{min}(A) \rightarrow X$ for tuple $\mathbf{t}$

Algorithm 3 first inserts each incoming tuple into `openw` (Line 7). If the tuple certainly exists, it is inserted into the tree of certain tuples whose lower bound position is $\mathbf{t}.\tau^{\downarrow}$. Note that each of these trees is sorted on $\tau^{\uparrow}$ which will be relevant later. Next the algorithm determines for which tuples from `openw`, their windows have been fully observed. These are all tuples $\mathbf{s}$ which are certainly ordered before the tuple $\mathbf{t}$ we are processing in this iteration ($\mathbf{s}.\tau^{\uparrow} < \mathbf{t}.\tau^{\downarrow}$). To see why this is the case, first observe that (i) we are processing input tuples in increasing order of $\tau^{\downarrow}$ and (ii) tuples are "finalized" by computing the aggregation bounds in monotonically increasing order of $\tau^{\uparrow}$. Given that we are using a window bound $[-N, 0]$, all tuples $\mathbf{s}$ that could possibly belong to the window of a tuple $\mathbf{t}$ have to have $\mathbf{s}.\tau^{\downarrow} \leq \mathbf{t}.\tau^{\uparrow}$. Based on these observations, once we processed a tuple $\mathbf{t}$ with $\mathbf{t}.\tau^{\downarrow} > \mathbf{s}.\tau^{\uparrow}$ for a tuple $\mathbf{s}$ in `openw`, we know that no tuples that we will process in the future can belong to the window for $\mathbf{s}$. In Line 11 we iteratively pop such tuples from `openw`. For each such tuple $\mathbf{s}$ we evict tuples from `cert` and update the high watermark $\texttt{c-rank}^{\downarrow}$ (Line 12). Recall that for a tuple $\mathbf{u}$ to certainly belong to the window for $\mathbf{s}$ we have to have $\mathbf{s}.\tau^{\uparrow} - N \geq \mathbf{t}.\tau^{\downarrow}$. Thus, we update $\texttt{c-rank}^{\downarrow}$ to $\mathbf{s}.\tau^{\uparrow} - N$ and evict from `cert` all trees storing tuples for sort positions smaller than $\mathbf{s}.\tau^{\uparrow} - N$. Afterwards, we compute the bounds on the aggregation result for $\mathbf{s}$ using `cert` and `poss` (we will describe this step in more detail in the following). Finally, we evict tuples from `poss` (and, thus, also $\texttt{pagg}^{\downarrow}$ and $\texttt{pagg}^{\uparrow}$) which cannot belong to any windows we will close in the future. These are tuples which are certainly ordered before the lowest possible position in the window of $\mathbf{s}$, i.e., tuples $\mathbf{u}$ with $\mathbf{u}.\tau^{\uparrow} < \mathbf{s}.s^{\downarrow} - N$ (see Fig. 5.5). Evicting tuples from `poss` based on the tuple for which we are currently computing the aggregation result bounds is safe because we are emitting tuples in increasing order of $\tau^{\uparrow}$, i.e., for all tuples $\mathbf{u}$ emitted after $\mathbf{s}$ we have $\mathbf{u}.\tau^{\uparrow} > \mathbf{s}.\tau^{\uparrow}$. Fig. 5.11 shows an example state for the algorithm when tuple $\mathbf{s}$ is about to be emitted. Tuples fully included in the red region ($\mathbf{t_2}$ and $\mathbf{t_3}$) are currently in $\texttt{cert}[i]$ for sort positions certainly in the window

for $\mathbf{s}$. Tuples with sort position ranges overlapping with the green region are in the possible window (these tuples are stored in `poss`). Tuples like $\mathbf{t_4}$ with upper-bound position higher than $\mathbf{s}$ will be popped and processed after $\mathbf{s}$. Once all input tuples have been processed, we have to close the windows for all tuples remaining in `openw`. This process is the same as emitting tuples before we have processed all inputs and, thus, is omitted from Algorithm 3.

Algorithm 3 uses function `compBounds` to compute the bounds on the aggregation function result for a tuple $\mathbf{t}$ using `cert`, $\texttt{pagg}^{\downarrow}$ and $\texttt{pagg}^{\uparrow}$ following the definition from Sec. 5.8.1. First, we fetch all tuples that are certainly in the window from `cert` based on the sort positions that certainly belong to the window for $\mathbf{t}$ ($[\mathbf{t}.\tau^{\uparrow}$ - N, $\mathbf{t}.\tau^{\downarrow}]$) and aggregate their A bounds. Afterwards, we use $\texttt{pagg}^{\downarrow}$ and $\texttt{pagg}^{\uparrow}$ to efficiently fetch up $\mathsf{possn}(\mathbf{R}, \mathbf{t})$ tuples possibly in the window for $\mathbf{t}$ to calculate the final bounds based on $\mathsf{max\text{-}k}$ and $\mathsf{min\text{-}k}$.

Complexity Analysis: Algorithm 3 first sorts the input in $O(n \log n)$ time using Algorithm 1 followed by a deterministic sort on $\tau^{\downarrow}$. Each tuple is inserted into `openw`, `poss`, and `cert` at most once and poped from `openw` exactly once. The size of the heaps the algorithm maintains is certainly less than n at all times. To compute the aggregation function bounds, we have to look at the certain tuples in $\texttt{cert}[i]$ for at most $size([N, 0]) = N+1$ sort positions i and at most $N+1$ tuples from `poss` that can be accessed using the connected heaps in $O(N \cdot \log n)$. Thus, the overall worst-case runtime of the algorithm is $O(N \cdot n \cdot \log n)$.

5.11 Creating AU-DBs

In this section we discuss how to translate data represented in incomplete and probabilistic data models into AU-DBs such that the generated AU-DB bounds the input uncertain database. Furthermore, we enable the user to specify bounds for an

expression. This enables integration of AU-DBs with Lenses [6, 51] to generate AU-DBs that encode the uncertainty introduced by data cleaning and curation heuristics.

5.11.1 tuple-independent database. A tuple-independent database (TI-DB) $\mathcal{D}$ is a database where each tuple t is marked as optional or not. The incomplete database represented by a TI-DB $\mathcal{D}$ is the set of instances that include all non-optional tuples and some subset of the optional tuples. That is, the existence of a tuple t is independent of the existence of any other tuple t'. In the probabilistic version of TI-DBs each tuple is associated with its marginal probability. The probability of a possible world is then the product of the probability of all tuples included in the world multiplied by the product of $1 - P(t)$ for all tuples from $\mathcal{D}$ that are not part of the possible world. We define a translation function $\mathsf{trans}^{AUDB}_{\text{TI-DB}}$ for TI-DBs that returns a AU-DB relation with certain attribute values. We extend this function to databases in the obvious way. A tuple's lower multiplicity bound is 1 if the tuples is not certain (marked as optional or its probability is less than 1). That is, like for UA-DBs, $\mathsf{trans}^{AUDB}_{\text{TI-DB}}$ encodes exactly the certain answers of the TI-DB and the SG world is selected as the world that contains all tuples whose marginal probability is larger than or equal to 0.5. Note that this is indeed one of possible worlds that have the highest probability among all worlds encoded by the TI-DB. Let $\mathcal{T}_{TI}(t)$ denote the range-annotated tuple which encodes the certain tuple t and $\mathcal{D}$ be a probabilistic TI-relation. We define:

$$\forall A \in \text{Sch}(\mathcal{R}) : \mathcal{T}_{TI}(t).A^{\downarrow} = \mathcal{T}_{TI}(t).A^{\uparrow} = \mathcal{T}_{TI}(t).A^{sg} = t.A$$

Using this definition, we define $\mathsf{trans}^{AUDB}_{\text{TI-DB}}$ for probabilistic TI-DBs as shown below.

$$\mathsf{trans}^{AUDB}_{\text{TI-DB}}(\mathcal{R})(\mathcal{T}_{TI}(t))^{\downarrow} := \begin{cases} 1 & \text{if } P(t) = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$\mathsf{trans}^{AUDB}_{\text{TI-DB}}(\mathcal{R})(\mathcal{T}_{TI}(t))^{sg} := \begin{cases} 1 & \text{if } P(t) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

$$\mathsf{trans}^{AUDB}_{\text{TI-DB}}(\mathcal{R})(\mathcal{T}_{TI}(t))^{\uparrow} := \begin{cases} 1 & \text{if } P(t) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Theorem 18 ($\mathsf{trans}^{AUDB}_{\text{TI-DB}}$ is bound preserving)**.** *Given a probabilistic TI-DB $\mathcal{D}$, $\mathsf{trans}^{AUDB}_{\text{TI-DB}}(\mathcal{D})$ is a bound for $\mathcal{D}$.*

Proof: By definition of the translation function $\mathsf{trans}^{AUDB}_{\text{TI-DB}}$, all attribute-values are certain. Thus, with exception of the upper bound on a tuple's multiplicity (1 is a trivial upper bound on the multiplicity of any tuple in an TI-DB), the claim follows from [16][Theorem 2] which proved the lower bounding property of this translation. Thus, the result of $\mathsf{trans}^{AUDB}_{\text{TI-DB}}$ trivially bounds $\mathcal{R}$. We omit the proof for incomplete TI-DBs since it is analog.

5.11.2 x-DBs. An x-DB [17], records for each tuple a number of alternatives. Such alternatives are encoded as so-called x-tuples. An x-tuple τ is simply a set of tuples $\{t_1, \ldots, t_n\}$ with a label indicating whether the x-tuple is optional. We use $|\tau|$ to denote the number of alternatives of x-tuple τ. x-relations are sets of such x-tuples and x-databases are sets of x-relations. A possible world of an x-relation $\leftrightarrow$ is a deterministic relation that is generated by selecting at most one alternative $t \in \tau$ for every x-tuple τ from $\leftrightarrow$ if τ is optional, or exactly one if it is not optional. That is, each x-tuple is assumed to be independent of the others, and its alternatives are

assumed to be disjoint (hence the name block-independent incomplete database). The probabilistic version of x-DBs (also called a *block-independent database* [17]) assigns each alternative a probability such that $P(\tau) = \sum_{t \in \tau} P(t) \leq 1$. Thus, a probabilistic x-tuple is optional if $P(\tau) < 1$. For an x-tuple $\tau = \{t_1, \ldots, t_n\}$, we use $\text{PICKMAX}(\tau)$ to denote the alternative of τ with the highest probability among all alternatives, picking the first alternative if there are multiple such alternatives. For instance, for $\tau = \{t_1, t_2\}$ with $P(t_1) = 0.5$ and $P(t_2) = 0.5$ we get $\text{PICKMAX}(\tau) = t_1$.

We now define a translation function $\mathsf{trans}_{\text{X-DB}}^{AUDB}$ that maps x-dbs into AU-DBs. In the result of this translation each x-tuple from the input is encoded as a single range-annotated tuple whose attribute bounds include the attribute values of all alternatives of the x-tuple. For the SG values of a tuple we choose the alternative with the highest probability. We use $\mathcal{T}_X(\tau)$ to denote a range-annotated tuple that we construct as shown below such that it bounds all alternatives for x-tuple τ. For any $A \in \text{Sch}(\tau)$ we define:

$$\mathcal{T}_X(\tau).A^{\downarrow} = \min_{t \in \tau} t.A$$

$$\mathcal{T}_X(\tau).A^{sg} = \text{PICKMAX}(\tau).A$$

$$\mathcal{T}_X(\tau).A^{\uparrow} = \max_{t \in \tau} t.A$$

X-tuples in an x-relation are certain if $P(\tau)$, i.e., if some alternative of τ exists in every possible world. For the incomplete version of x-dbs, a tuple is certain if it is not optional. We set the lower multiplicity bound for such tuples to 1. The lower multiplicity bound for all other tuples is set to 0. Any alternative of an x-tuple is possible. This alternative $\text{PICKMAX}(\tau)$ is part of the SGW if $(1 - P(\tau)) \leq P(\text{PICKMAX}(\tau))$, i.e., it is more likely that $\text{PICKMAX}(\tau)$ exists than that no alternative of the x-tuple exists. The multiplicity upper bound for any $\mathcal{T}_X(t)$ is 1 and its lower bound is 1 iff the

x-tuple is not optional (one of the alternatives of the x-tuple exists in every world). As mentionIn probabilistic x-DBs we check that $P(\tau) = 1$. We the show version of $\mathsf{trans}^{AUDB}_{\text{x-DB}}$ for probabilistic x-DBs below.

$$\mathsf{trans}^{AUDB}_{\text{x-DB}}(\mathcal{R})(\tau)^{\downarrow} = \begin{cases} 1 & \text{if } P(\tau) = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$\mathsf{trans}^{AUDB}_{\text{x-DB}}(\mathcal{R})(\tau)^{sg} = \begin{cases} 1 & \text{if } (1 - P(\tau)) \leq P(\textsc{pickMax}(\tau)) \\ 0 & \text{otherwise} \end{cases}$$

$$\mathsf{trans}^{AUDB}_{\text{x-DB}}(\mathcal{R})(\tau)^{\uparrow} = \begin{cases} 1 & \text{if } P(\tau) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Theorem 19 ($\mathsf{trans}^{AUDB}_{\text{x-DB}}$ preserves bounds)**.** *Given an x-table $\mathcal{D}$, $\mathsf{trans}^{AUDB}_{\text{x-DB}}(\mathcal{D})$ bounds $\mathcal{D}$.*

Proof: Trivially, $\mathcal{T}_X(\tau)$ bounds all alternatives of τ by construction. Each possible world contains at most one alternative per x-tuple (exactly one if the x-tuple is certain). Thus, $\mathcal{T}_X(\tau)^{\uparrow} = 1$ is an upper bound on the multiplicity of any x-tuple's alternative in every world. The lower bound multiplicity $\mathcal{T}_X(\tau)^{\downarrow}$ is 1 if $P(\tau) = 1$ and 0 otherwise. Thus, $\mathcal{T}_X(\tau)^{\downarrow}$ a lower bound the multiplicity of an alternative of the x-tuple in every world. Since x-tuples are assumed to be independent of each other and each possible world contains at most one alternative of an x-tuple, the probability of a possible world D of an x-table $\mathcal{R}$ is calculated as

$$\left(\prod_{t \in D} P(t)\right) \cdot \left(\prod_{\tau : (\nexists t \in \tau : t \in D)} 1 - P(\tau)\right)$$

Thus, the SG world is indeed the world with the highest probability of the x-DB, because it contains the highest probability alternative for each x-tuple (or no alternative if this is the highest probability option).

5.11.3 C-tables. In contrast to the proof of Thm. 9, we now consider *C-tables* [12] where variables from the set Σ of variable symbols can also be used as attributes values, i.e., tuples over $\mathbb{D} \cup \Sigma$. Recall from the proof of Thm. 9 that an C-table [12] $\mathcal{R} = (\leftrightarrow, \phi, \Phi)$ is a relation $\leftrightarrow$ paired with (i) a global condition Φ which is also a logical condition over Σ and (ii) a function ϕ that assigns to each tuple $t \in \leftrightarrow$ a logical condition over Σ. Given a valuation μ that assigns to each variable from Σ a value, the global condition and all local conditions evaluate to either $\top$ or $\bot$. The incomplete database represented by a generalized C-table $\mathcal{R}$ is the set of all relations $\leftrightarrow$ such that there exists a valuation μ for which $\mu(\Phi)$ is true and $\leftrightarrow = \{\mu(t) \mid \mu(\phi(t))\}$, i.e., $\leftrightarrow$ contains all tuples for which the local condition evaluates to true where each variable in the tuple is replaced based on μ.

Since determining whether a tuple in a c-table is certain is coNP-complete[11], we settle for a transformation that creates lower and upper bounds for both attribute-values and multiplicities that are not tight. Using a constraint solver, we can determine (i) whether the local condition $\phi(t)$ of a tuple is a tautology (the tuple exists in every world) and (ii) lower and upper bounds on the value of a tuple's attribute. For instance, for the lower bound we have to solve the following optimization problem:

[11]Determining the certain answers to a query over a Codd-table is coNP-complete [11, 94]. Since, the result of any first order query over a Codd-table can be encoded as a C-table and evaluating a query in this fashion is in `PTIME`, it follows that determining whether a tuple is certain in a C-table cannot be in `PTIME`, because otherwise we could use this to compute the certain answers to a query in `PTIME` by evaluating it in over a C-table encoding the input Codd-table and then calculating the certain tuples of the resulting C-table.

Minimize:

$$\mu(t.A)$$

Subject to:

$$\phi(\mu(t))$$
$$\forall x \in \Sigma : \mu(x) \in \mathbb{D}$$

We use $\min(t.A)$ and $\max(t.A)$ to denote the results of these optimization problems. While solving constraints is not in PTIME, this can be acceptable if the size of formulas used in local conditions is relatively small. If that is not the case, we can trade tightness of bounds for performance and fall back to a simpler method, e.g., in worst case the minimum and maximum values of $\mathbb{D}$ are safe bounds and the bounds for a constant value is the constant itself. To select a SGW, we have to find a valuation μ such that the global condition holds. In general, this may be computationally hard. For C-tables without global conditions, any valuation would do. Let μ_{SG} denote the valuation we select. To define our transformation, we again first define a function $\mathcal{T}_C$ that maps tuples and their local conditions to range-annotated tuples. For all $A \in \text{Sch}(t)$, this function is defined as:

$$\mathcal{T}(t).A^{\downarrow} \coloneqq \min(t.A)$$
$$\mathcal{T}(t).A^{sg} \coloneqq \mu_{SG}(t.A)$$
$$\mathcal{T}(t).A^{\uparrow} \coloneqq \max(t.A)$$

Now to determine the multiplicity bounds of tuples we have to reason about whether a local condition is a tautology and whether it is satisfiable. This can again be checked using constraint solvers. We can fall back to a PTIME method that can only detect certain types of tautologies, e.g., if the local condition is a conjunction

of inequalities. Note that even the first version does not guarantee tight bounds, because it (i) ignores global conditions and (ii) does not take into account that the set of tuples encoded by two tuples from the input C-table may overlap (e.g., a tuple may be certain even though no local condition is a tautology). We define predicate $\text{isTAUTOLOGY}(\psi)$ that evaluates to true if ψ is a tautology (potentially using the optimization as described above). Similarly, $\text{isSATISFIABLE}(\psi)$ is true if ψ is satisfiable. Using these predicates we define the multiplicity bounds generated by $\mathsf{trans}^{AUDB}_{\text{C-TABLE}}$ as shown below.

$$\mathsf{trans}^{AUDB}_{\text{C-TABLE}}(\mathcal{R})(\mathcal{T}_C(t,\phi(t)))^{\downarrow} \coloneqq \begin{cases} 1 & \text{if isTAUTOLOGY}(\psi) \\ 0 & \text{otherwise} \end{cases}$$

$$\mathsf{trans}^{AUDB}_{\text{C-TABLE}}(\mathcal{R})(\mathcal{T}_C(t,\phi(t)))^{sg} \coloneqq \begin{cases} 1 & \text{if } \mu_{SG}(\phi(t)) \\ 0 & \text{otherwise} \end{cases}$$

$$\mathsf{trans}^{AUDB}_{\text{C-TABLE}}(\mathcal{R})(\mathcal{T}_C(t,\phi(t)))^{\uparrow} \coloneqq \begin{cases} 1 & \text{if isSATISFIABLE}(\psi) \\ 0 & \text{otherwise} \end{cases}$$

Theorem 20 ($\mathsf{trans}^{AUDB}_{\text{C-TABLE}}$ is bound preserving)**.** *Given an incomplete database $\mathcal{D}$ encoded as C-tables, $\mathsf{trans}^{AUDB}_{\text{C-TABLE}}(\mathcal{D})$ bounds $\mathcal{D}$.*

Proof: Based on the definition of C-tables, any tuple whose local condition is a tautology exists in every possible world. Furthermore, if the local condition of a tuple is satisfiable then it may exist in some world (only if the global condition for the valuation that satisfies the local condition evaluates to true). Finally, by construction the tuples of the AU-DB created by $\mathsf{trans}^{AUDB}_{\text{C-TABLE}}$ for a C-table $\mathcal{R}$ bound the possible values of the tuples of $\mathcal{R}$ across all possible worlds. By construction μ_{SG} is a possible world of $\mathcal{R}$. Thus, $\mathsf{trans}^{AUDB}_{\text{C-TABLE}}(\mathcal{R})$ bounds $\mathcal{R}$.

The probabilistic version of C-tables [55] associates each variable with a probability distribution over its possible values. Variables are considered independent of each other. The probability of a variable assignment μ is then the product of all probabilities $P(x = \mu(x))$ for each $x \in \Sigma$. The probability of a possible world is the sum of the probabilities of all valuations that produce this world. Our translation scheme can be adapted to the probabilistic version of C-tables by picking the SG by selecting the highest probability assignment for each variable x to approximate the world with the highest probability. For probabilistic C-tables with global conditions, finding a possible world may again be computationally hard.

5.11.4 Lenses. Lenses as presented in [6, 41, 51] are a principled method for exposing the uncertainty in the result of data cleaning, curation and integration methods as incomplete data. For instance, when repairing primary key violations by picking one tuple t for each set of tuples with the same key, the choice of t is typically made based on some heuristic. However, all other possible picks cannot be ruled out in general. A lens applies such a cleaning heuristic to select on possible repair as a SGW and then encodes the space of possible repairs for a method as an incomplete database. Technically, this is achieved using the so-called variable-generating algebra which allows queries to introduce uncertainty in the form of random variables. The result are Virtual C-tables which generalize C-tables by allowing symbolic expressions as attribute values. In our primary key repair example, for a key k and the set of tuples $T = \{t \mid t.K = k\}$ which have this key value (K are the key attributes), random variables are introduced for each attribute A such that $|\{t.A \mid t \in T\}| > 1$, i.e., the attribute's value depends on the choice of repair for k. The possible value for such a variable $x_{k,A}$ are then all values that appear in T: $x_{k,A} \in \{t.A \mid t \in T\}$. This operation can be implemented in the variable-generating algebra as a query that uses aggregation grouping on the key attributes K, to check for each non-key attribute and key value k whether there is more than one tuple with this key value. If this is

the case, then the value of each attribute is replaced with a variable if there is more than one value for this attribute in a group.

Example 28 (Cleaning with Lenses)**.** *Consider repairing the key of a relation $R(A, B)$ with key A. In systems like Mimir [6] and Vizier [41, 51] that support lenses, variables are introduced by queries while their distribution is specified separately. Here we assume that the construct $Var(name)$ creates a new random variables with name name. Using this construct, we can generate a Virtual C-table that encodes all possible repairs using the query shown below. We first count the number of distinct values of attribute B for each key (subquery* `keys`*). Then in the outer query we replace each B value with a variable if there is more than one possible value. Note that a full solution also requires us to separately specify the possible values for these variables. However, we omit this here.*

```
SELECT A,
       CASE WHEN numB > 1
            THEN Var(tid || 'B')
       ELSE theB
       END AS B
  FROM (SELECT A,
               count(DISTINCT B) AS numB,
               min(B) AS theB
          FROM R
         GROUP BY A) keys
```

To support AU-DBs as an approximation of Virtual C-tables created by Lenses (and thus to support tracking of a wide range of cleaning and curation operations), we can either develop a transformation for Virtual C-tables or define a construct similar to the one for Virtual C-tables used in the example above to allow uncertainty to

be introduced as part of a query. We opted for the second option, since it would allow new Lenses to be implemented directly in AU-DBs without the need for an excursion to Virtual C-tables for which evaluation of certain operators (e.g., joins) is expensive. Towards that goal we introduce a construct $\textsc{MakeUncertain}(e^{\downarrow}, e^{sg}, e^{\uparrow})$ which takes three expressions $e^{\downarrow}$, e^{sg}, $e^{\uparrow}$ that calculate a SG value, and an lower an upper bound for a value. We require that this construct can only be applied to AU-DBs, assumed to have been created using one of our transformations explained above. For deterministic inputs we produce a dummy transformation $\mathsf{trans}^{AUDB}_{\textsc{certain}}$ that assumes that all tuples and attribute values are certain.

Example 29. *Reconsider the key repair task from Ex. 28. We can create a AU-DB bounding the space of repairs using the query shown below. Here we select the minimum B value for each group as the SG value for attribute B.*

```
SELECT A,
       CASE WHEN numB > 1
            THEN
       ELSE MakeUncertain(minB,minB,maxB)
       END AS B
  FROM (SELECT A,
               count(DISTINCT B) AS numB,
               min(B) AS minB,
               max(B) AS maxB
          FROM trans^AUDB_CERTAIN(R)
         GROUP BY A) keys
```

CHAPTER 6

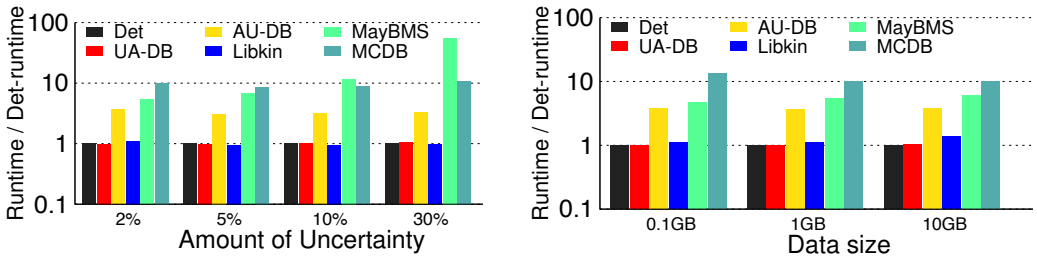
EXPERIMENTS

We compare UA-DBs and AU-DBs (`AU-DB`) implemented on Postgres against (1) `Det`: Deterministic SGQP; (2) `Libkin`: An under-approximation of certain answers [35, 47]; (3) `UA-DB`: An under-approximation of certain answers combined with SGQP [16]; (4) `MayBMS`: `MayBMS` used to compute all possible answers[12]; (5) `MCDB`: Database sampling (10 samples) in the spirit of MCDB [59] to over-approximate certain answers; (6) `Trio`: A probabilistic DB with bounds for aggregation [17]; and (7) `Symb`: An SMT solver (Z3) calculating aggregation result bounds based on the symbolic representation from [23]. All experiments are run on a 2×6 core AMD Opteron 4238 CPUs, 128GB RAM, 4×1TB 7.2K HDs (RAID 5). Against these mentioned approaches, we use synthetic data set `PDBench` for general performance testing. We use parameterized synthetic dataset for micro benchmarks by adjusting factors that affecting performance like uncertainty percentage and uncertainty range. We performed both time and quality measurements for real queries over real world datasets that get uncertainty from resolving data quality issues. We report the average of 10 runs.

6.1 `PDBench`

For a general performance comparison over both of our approaches, we use `PDBench` [96], a modified TPC-H data generator [97] 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

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



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

Figure 6.1. PDbench 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 [47]. 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 UA-DBs and AU-DBs compare to other incomplete database approaches, we use the queries of `PDBench` (simple SPJ queries). With a scale factor 1 (SF1) database (∼1GB per world), we evaluate scalability relative to the 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. 6.1a 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. 6.1b, UA-DBs and AU-DBs scale linearly in the SF for such queries where UA-DBs have nearly no overhead comparing with `Det`.

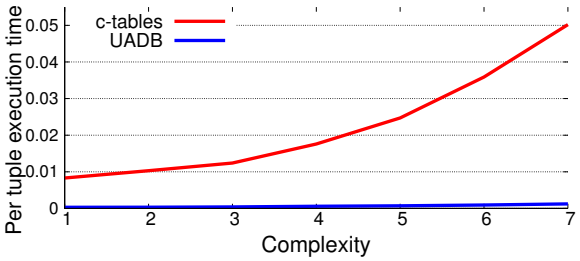


Figure 6.2. Certain answers over C-tables

6.2 UA-DB experiments

Certain Answers over C-tables: As an example of a more complex incomplete data model, we evaluate the performance of UA-DBs against computing certain answers over C-tables. We create a synthetic table with 8 attributes. For each tuple we randomly chose half of its attributes to be variables and the other half to be floating point constants. We construct random queries by assembling a number of randomly chosen self-joins, projections, or selections. We measure query execution time using UA-DBs. The exact certain answers for a query over the C-tables are computed by instrumenting the query to calculate a local condition for every result tuple and running the Z3 constraint solver (`https://github.com/Z3Prover/z3`) over the resulting boolean expression. An answer is certain iff its local condition is a tautology. Each local condition's complexity of depends on how tuples are combined by the query. Joins combine tuple conditions by conjunction, while projections and unions combine matching result tuples by disjunction. Selection extends the local condition on rows where the selection predicate accesses a variable-valued attribute. Each selection operator further increases complexity for each conjunction, disjunction or arithmetic operation. Sec. 6.2 shows the average runtime per result tuple for both C-tables and UA-DBs averaged over all randomly generated queries. The x-axis is the number of operators (i.e., selection, projection or join) in the source query. Overhead for C-tables increases super-linearly in query complexity from about 27× to over 40×.

6.2.1 UA-DB real world datasets. We use multiple real world datasets. from

Table 6.1. Real World Datasets

Dataset	Rows	Cols	U_{Attr}	U_{Row}	URL
Building Violations	1.3M	35	0.82%	12.8%	https://data.cityofchicago.org/Buildings/Building-Violations/22u3-xenr
Shootings in Buffalo	2.9K	21	0.24%	2.1%	http://projects.buffalonews.com/charts/shootings/index.html
Business Licenses	63K	25	1.39%	14.0%	https://data.cityofchicago.org/Community-Economic-Development/Business-Licenses-Current-Active/uupf-x98q
Chicago Crime	6.6M	17	0.21%	0.9%	https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-present/ijzp-q8t2
Contracts	94K	13	1.50%	19.2%	https://data.cityofchicago.org/Administration-Finance/Contracts/rsxa-ify5
Food Inspections	169K	16	0.34%	4.6%	https://data.cityofchicago.org/Health-Human-Services/Food-Inspections/4ijn-s7e5
Graffiti Removal	985K	15	0.09%	0.8%	https://data.cityofchicago.org/Service-Requests/311-Service-Requests-Graffiti-Removal/hec5-y4x5
Building Permits	198K	19	0.42%	5.3%	https://https://www.kaggle.com/aparnashastry/building-permit-applications-data/data
Public Library Survy	9.2K	99	1.19%	14.2%	https://www.imls.gov/research-evaluation/data-collection/public-libraries-survey/explore-pls-data/pls-data

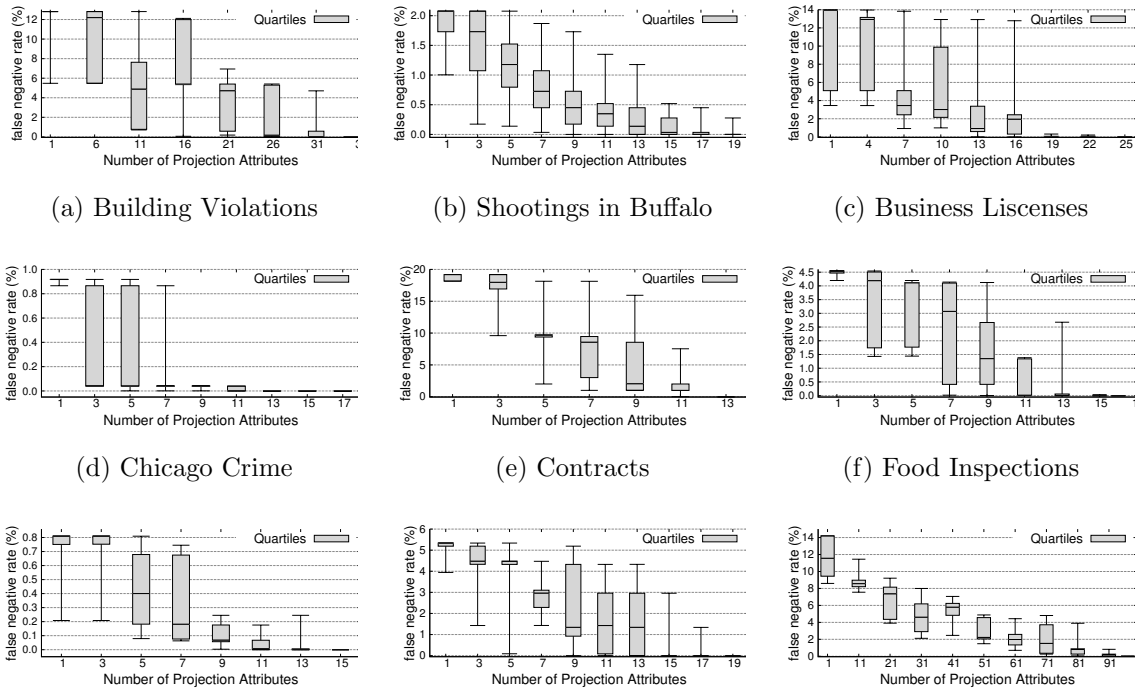
a wide variety of domains to evaluate how our approach performs for real world data. We use SparkML to impute missing values in the datasets, treating alternative imputations as a source of uncertainty. The resulting dataset, represented as an x-DB, was converted to a UA-DB using $\mathsf{trans}^{UADB}_{\text{x-DB}}$ (Sec. 4.5), which marks all tuples with at least one uncertain attribute as uncertain. Table 6.1 shows basic statistics for the cleaned datasets and URLs for the original datasets: the #rows, #attributes, the percentage of attribute values that are uncertain (U_{attr}), and the percentage of rows marked as uncertain by our exact approximation (U_{row}).

Table 6.2. Real Query Results

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

Incompleteness: To measure the false negative rate (fraction of answers that are misclassified as uncertain) of our approach, we use queries that project on a randomly chosen set of attributes. The rationale for this is that projecting an uncertain tuple onto a subset of its attributes that are certain causes the tuple to produce a certain answer. This is the primary situation in which UA-DBs mis-classify results, so this experiment represents a worst case scenario for UA-DBs. We evaluate queries which project on a randomly chosen set of attributes and measure the false negative rate (*FNR*). Fig. 6.3a to Fig. 6.3i show the distribution of the FNR (min, 25-percentile, median, 75-percentile, max) for queries with a fixed number of projection attributes. As expected, the FNR decreases as the number of projection attributes grows, but is low in general (less than 20% in the worst cases).For most datasets, the median FNR is below 5% when at least half of the attributes are involved in the projection. Note that selection and join do not produce any "new" false negative results (see proof of Theorem 6 in Appendix 4.8). This shows that for real world datasets with correlated errors, the FNR is typically low.

6.2.2 UA-DB Real Queries. We next evaluate the effectiveness of our approach on five queries over the real world datasets (the SQL code and descriptions of these queries are shown below). Most of our real world datasets are from open data portals that associate analyses (e.g., visualizations) with datasets. Test queries are reverse engineered from these analyses. We measure the performance overhead and false negative rate of UA-DBs. Performance overhead is measured as the slowdown relative to deterministic query processing. As Table 6.2 shows, our approach introduces a slight (less than 4%) overhead for these queries. The worst case (4%) is Q5, which involves



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

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

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

Figure 6.3. Measuring incompleteness as the fraction of certain answers that were misclassified as uncertain

a join operator. All other queries, which contain only selections and projections have under 3% overhead. In each case, we saw a 1% false negative rate or lower. Notably, Q3 returns no misclassified results due to its small result size.

Probabilistic databases: We next compare the performance and accuracy of UA-DBs against MayBMS. For this experiment, we use a BI-DB (an x-DB with probabilities), varying the number of alternatives for each block and use three queries Q_{P1}, Q_{P2} and Q_{P3} of varying complexity described in [93]. For MayBMS, we treat tuples with probability $p \geq 1$ as certain[13]. Table 6.3 shows both runtime and error rate for both systems, with 2, 5, 10, or 20 alternatives. For MayBMS we show the result for exact probability computation and for approximation using the scheme from [57] with an error bound of 0.3 (shown in parentheses). Note that query processing in a UA-DB is independent of the number of possible worlds. Only a single alternative

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

Table 6.3. Probabilistic database

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

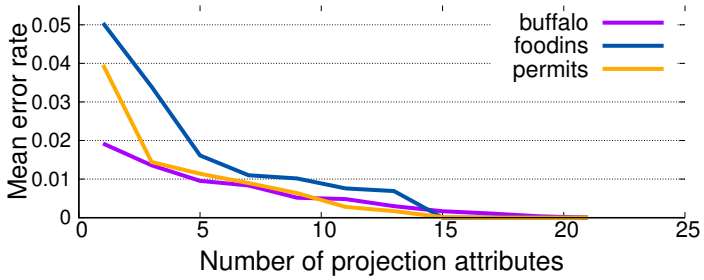


Figure 6.4. Bag semantics - mislabelings

is used for each block. We observe that MayBMS's results include both false positives and false negatives. Because results are computed by summing floating point numbers, even MayBMS' exact probability computations exhibits a small amount of rounding error that is more noticeable for larger number of alternatives (e.g., MB-20). Although approximating probabilities can improve performance especially for complex queries, MayBMS is still orders of magnitude slower than UA-DBs. Q_{P3} includes a self-join which further slows MayBMS down due to the increase in possible worlds and expression complexity.

Beyond Set Semantics: In this experiment we evaluate the FNR of our approach using bag semantics (semiring $\mathbb{N}$) and the access control semiring $\mathbb{A}$ [34]. For the bag semantics experiment we evaluate projections under bag semantics over some of the real world datasets from Table 6.1. The results for this experiment are shown in Fig. 6.4. Observe that the FNR is similar to the set semantics case. The access control semiring annotates each tuple with an access control level (one of **0** - "nobody

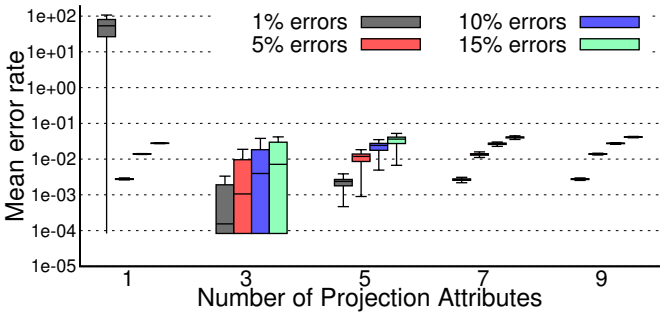


Figure 6.5. Access control semiring - mislabelings

can access the data', **T** is "top secret", **S** is "secret", **C** is "confidential", and **P** is "public") to determine what clearance-level is necessary to view the tuple. Addition (multiplication) is max (min) according to the following order over the elements $\mathbf{0} < \mathbf{T} < \mathbf{S} < \mathbf{C} < \mathbf{P}$. For this experiment, we emulate a scenario where private information in a dataset is heuristically detected and secured with an $\mathbb{A}$ annotation. Using 5 real world datasets from Table 6.1, we randomly assigned access control labels to each tuple, and then created multiple instances with 1%, 2%, 5%, and 10% of misclassified tuples. We evaluated random projection queries over these datasets and measured the amount of misclassified query results weighted by the distance between the certain annotation and the approximations, e.g., the distance of **C** and **T** is $\frac{2}{5} = 0.4$. In Fig. 6.5, we vary the number of projection attributes and show the distribution of the amount of misclassified query results over 9 randomly selected projection queries for 5 datasets. The FNR increases when the input error rate is increased, but is quite low in most cases.

6.2.3 Query Descriptions. Q1: This query is expressed over the Chicago crime dataset. The query returns all crime ids and case numbers for all thefts, domestic batteries, and criminal damages. Here, attribute `IUCR` (Illinois Uniform Crime Reporting code) is a system for specifying crime types.

```
SELECT id, case_number,
       CASE IUCR
```

```
          WHEN 0820 then 'Theft'
          WHEN 0486 then 'Domestic Battery'
          WHEN 1320 then 'Criminal Damage'
        END AS crime_type
FROM Q
WHERE IUCR=0820 OR IUCR=0486 OR IUCR=1320
```

Q2: Find all crime ids, case numbers, longitudes and latitudes of crimes within a retangular area containing Chicago WaterTower.

```
SELECT id, case_number, Longitude, Latitude
FROM crime
WHERE Longitude BETWEEN -87.674 AND -87.619
      AND Latitude BETWEEN 41.892 AND 41.903
```

Q3: This is a query over the graffiti dataset. Q3 returns all street addresses and zip codes for graffiti removal requests that are currently open.

```
SELECT Street_Address, ZIP_Code, status
FROM graffiti
WHERE status='Open'
```

Q4: Find all dates, addresses and zip codes of food inspections of restaurants that passed, but were identified as "high risk".

```
SELECT Inspection_Date, address, zip
FROM foodinspections
WHERE results = 'Pass w/ Conditions'
      AND risk = 'Risk 1 (High)'
```

Q5: For each crime id, case numbers, and IUCRs of crimes, find all status, service request numbers and community areas from graffiti removal requests where both take place in district 8 and the graffiti removal request's location is within 100 coordinate units of the crime's location.

```
SELECT c.ID,
       c.Case_Number,
       c.IUCR,
       g.status,
       g.Service_Request_Number,
       g.Community_Area
FROM
    (SELECT * FROM graffiti
     WHERE police_district = 8) g,
    (SELECT * FROM crime
     WHERE district = '008') c
WHERE c.X_Coordinate < g.X_Coordinate + 100
  AND c.X_Coordinate > g.X_Coordinate - 100
  AND c.Y_Coordinate < g.Y Coordinate + 100
  AND c.Y_Coordinate > g.Y_Coordinate - 100
```

MayBMS-Q_{P1}: Find probability for a randomly chosen tuple.

```
SELECT conf()
FROM buffalo
WHERE index=1;
```

MayBMS-Q_{P2}: Find probability for shooting in each district for a random range of

incidence.

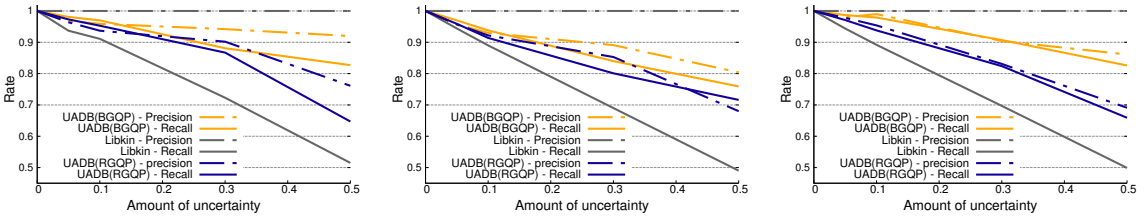
```
SELECT *
FROM
   (SELECT "District_shooting",
         index,
         conf()
    FROM bp20
    GROUP BY "District_shooting",index) x
WHERE index<2000
  AND index>650
  AND "District_shooting"='BD';
```

MayBMS-Q_{P3}: Find probabilities for all incidences that happened in the same district with same type of shooting for a random incident.

```
SELECT xind, yind,p
FROM
   (SELECT y.index AS yind,
         x.index AS xind,
         x."District_shooting" AS xds,
         y."District_shooting" AS yds,
         x."Type_shooting" AS xts,
         y."Type_shooting" AS yts,
         conf() AS p
    FROM bp20 y, bp20 x
    GROUP BY y.index,
           x.index,
           y."District_shooting",
```

```
            x."District_shooting",
            x."Type_shooting",
            y."Type_shooting") z
WHERE xds=yds
  AND xts=yts
  AND xind=692;
```

6.2.4 Utility of Query Answers. We claimed that SGQP and, thus also UA-DBs, have better utility than certain answers, as additional, useful possible answers are included in the result. The next experiment supports this claim quantitatively by contrasting the under-approximation of Libkin with UA-DBs evaluating two methods for extracting a best-guess world. To start, we create an incomplete database for which we have the ground truth (i.e., a "correct" possible world). This world (denoted as D_{ground}) is created by processing a source dataset to remove all rows with nulls. We next use D_{ground} to create an incomplete database $\mathcal{D}$ by replacing a random set of attribute values with nulls, varying the fraction of attributes replaced from 0% (deterministic input), to 50%. Then, we derive a best guess world D_{clean} from $\mathcal{D}$ by either using a standard missing value imputation algorithm (we refer to this method as **BGQP**) or randomly pick a replacement value (random-guess query processing or **RGQP**). We evaluate queries over $\mathcal{D}$ and D_{clean} using Libkin and UA-DBs respectively, and compare the result with the ground truth D_{ground}. Fig. 6.6 shows both precision (fraction of results in D_{ground}) and recall (fraction of D_{ground} in the results) as we vary the level of uncertainty. Libkin's method always under-approximates, guaranteeing 100% precision. However, recall is much lower than for UA-DBs and drops rapidly when the amount of uncertainty is increased. In contrast, the precision and recall achieved by UA-DBs remains between 80-90% for BGQP, even when half of all attribute values are uncertain. This supports our conjecture that certain answers are less similar to actual answers than answers obtained over a selected-guess



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

Figure 6.6. Utility - varying the amount of uncertainty

world. Compared with BGQP, RGQP is less accurate and less complete, but still has a higher recall than Libkin.

6.3 AU-DB experiments

TPC-H queries: We now evaluate actual TPC-H queries on `PDBench` data for AU-DBs. These queries contain aggregation with uncertain group-by attributes (only supported by `AU-DB` and `MCDB`). Results are shown in Table 6.4. 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. 6.7 shows the runtime of our technique compared to `Trio` which is significantly slower, and `Symb` (only competitive for low #agg-ops values).

6.3.1 AU-DB Micro-benchmarks. We use a synthetic table with 100 attributes with uniform random values to evaluate the performance and accuracy of our approach.

Table 6.4. TPC-H query performance (runtime in sec)

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

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: possible groups are compressed before being joined with the output groups. This improves performance when there are fewer result groups. As Fig. 6.8a 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. 6.8b

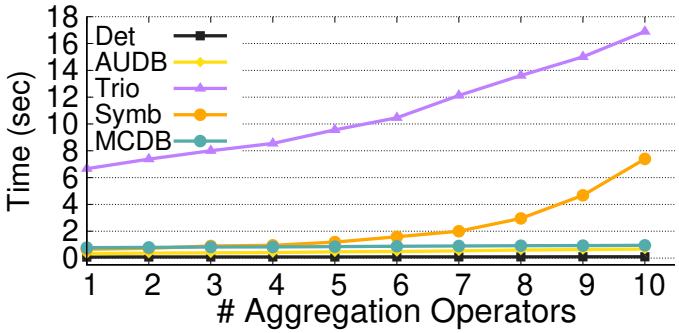
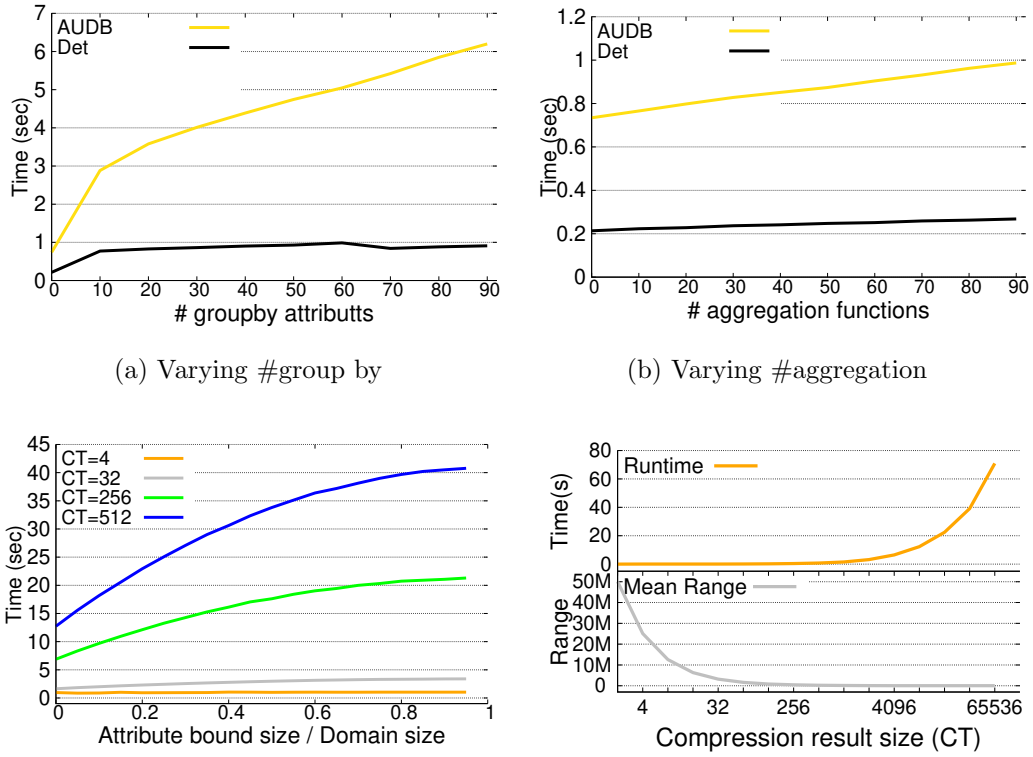


Figure 6.7. Simple aggregation over TPC-H data

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 tradeoffs between tightness and compression for aggregation using **sum** aggregation with group-by. Fig. 6.8d 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. 6.8c), 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. Fig. 6.10a and 6.10b show the percentage of over-grouping



(a) Varying #group by

(b) Varying #aggregation

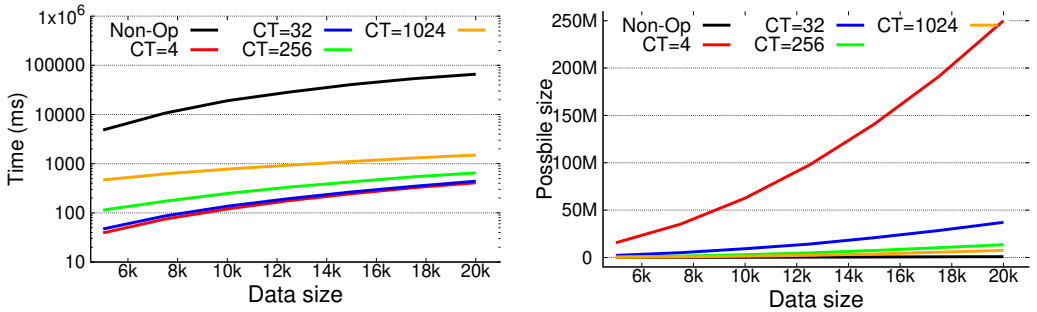
(c) Varying attribute range

(d) Varying compression rate

Figure 6.8. Aggregation Microbenchmarks - Performance and Accuracy

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.

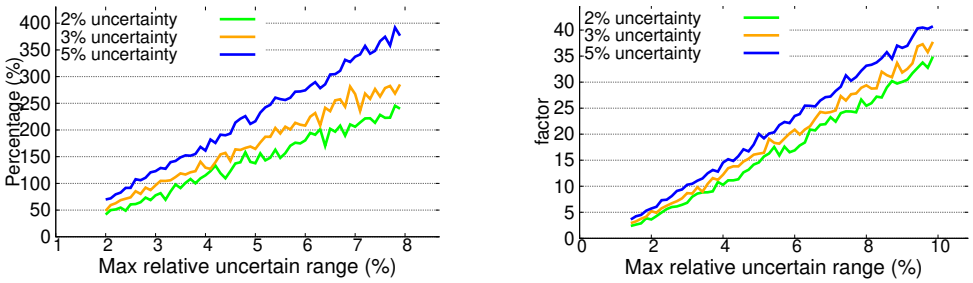
Join Optimizations: Next, we evaluate the impact of our join optimization. Fig. 6.9a



(a) Runtime

(b) Attribute-level Accuracy

Figure 6.9. Join Optimizations - Performance and Accuracy



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

Figure 6.10. Aggregation - varying attribute range

Table 6.5. Join query performance (runtime in sec)

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

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 $\sim$ 1 and $\sim$ 2 orders of magnitude faster depending on the compression rate (i.e., CT). As a simple accuracy measure, Fig. 6.9a 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 Table 6.5, 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.

6.3.2 AU-DB Real World Data. For this experiment, we repaired key violations for real world datasets (references shown in Table 6.6). 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. Table 6.6 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 Table 6.6, see [43] for additional details). Table 6.6 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 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. Table 6.6 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 Table 6.6, 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.

The real world queries are listed below with brief explanation of what the queries are doing:

Qn1:

```
SELECT title, release_year, director
FROM netflix
WHERE release_year < '2017';
```

Select all shows with year earlier than 2017.

Qn2:

```
SELECT director, MAX(release_year)
FROM netflix
GROUP BY director;
```

What is the year of the most current show for each director.

Qc1:

```
SELECT date, block, District
FROM crimes
WHERE Primary_Type='HOMICIDE' AND Arrest='False';
```

What is the date, block and district of all HOMICIDE crimes that are not arrested.

Qc2:

```
SELECT year, count(*)
FROM crimes
```

```
GROUP BY year;
```

Count the number of crimes for each year.

```
Qh1:
SELECT Facility_Name, Measure_Name, score
FROM healthcare_c
WHERE state != 'TX'
AND state !='CA'
AND measure_id='HAI_1_SIR';
```

What is the facility name, measure name and score for all records that measuring HAI_1_SIR except state TX and CA?

```
Qh2:
SELECT sum(score)
FROM healthcare_c
GROUP BY Facility_Name;
```

What is the total score for each facility?

Detailed configurations for each microbench test is listed in figure 6.7. A brief data description is listed in Table 6.8.

6.4 Sorting and windowed aggregation experiments

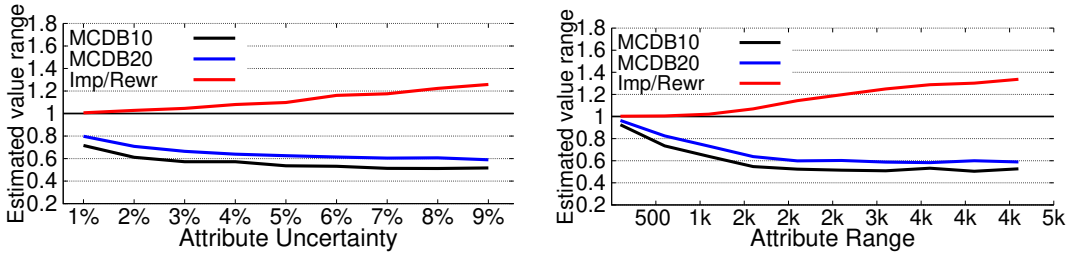
Compared Algorithms: We compare against several baselines: `MCDB10` and `MCDB20` are MCDB with 10 and 20 sampled worlds, respectively. For MCDB, we treat the highest and lowest possible value across all samples as the upper and lower bounds and compare against the tight bounds produced by the other algorithms (since computing optimal bounds is often intractable). Given a exact bound $[c, d]$, we define the

recall of a bound $[a, b]$ as $\frac{min(b,d)-max(a,c)}{d-c}$ and the accuracy of $[a, b]$ as $\frac{max(b,d)-min(a,c)}{min(b,d)-max(a,c)}$. The recall/accuracy for a relation is then the average recall/accuracy of all tuples. `PT-k` [38] only supports sorting and returns all answers with a probability larger than a user-provided threshold of being among the top-k answers. By setting the threshold to 1 (0) we can use this approach to compute all certain (possible) answers. `Symb` represents aggregation results, rank of tuples, and window membership as symbolic expressions which compactly encode the incomplete database produced by possible world semantics using the model from [101] for representing aggregation results and a representation similar to [84] to encode uncertainty in the rank of tuples. We use an SMT solver (Z3 [102]) to compute tight bounds on the possible ranks / aggregation results for tuples. `Rewr` is a rewrite-based approach we implemented uses self-unions for sorting queries and self-joins for windowed aggregation queries. `Imp` is the native implementation of our algorithms in Postgres.

6.4.1 Microbenchmarks on Synthetic Data. To evaluate how specific characteristics of the data affect our system's performance and accuracy, we generated synthetic data consisting of a single table with 2 attributes for sorting and 3 attributes for windowed aggregation. Attribute values are uniform randomly distributed. Except where noted, we default to 50k rows and 5% uncertainty with a maximum 1k attribute range on uncertain values.

6.4.1.1 Sorting and Top-k Queries. Scaling Data Size: Fig. 6.13 shows the runtime of sorting, varying the dataset size. Since `Symb` and `PT-k` perform significantly worse, we only include these methods for smaller datasets (Fig. 6.13a). MCDB and our techniques significantly outperform `Symb` and `PT-k` ($\sim$2+ OOM). `Rewr` is roughly on par with `MCDB20` while `Imp` outperforms `MCDB10`. Given their poor performance and their lack of support for windowed aggregation, we exclude `Symb` and `PT-k` from the remaining microbenchmarks.

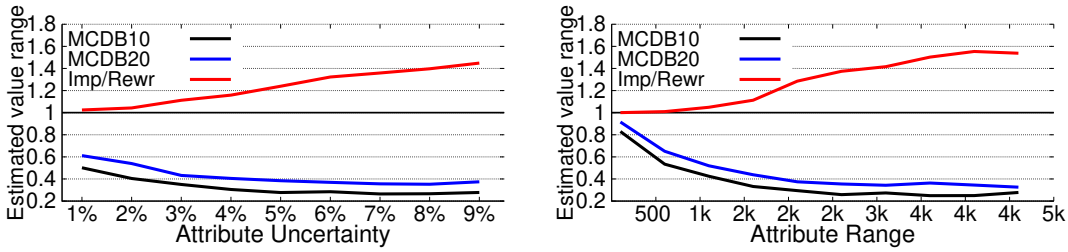
Varying k, Ranges, and Rate: Table 6.9 shows runtime of top-k (k is specified) and sorting queries (k is not specified) when varying (i) the number of tuples returned (k), (ii) the size of the ranges of uncertain order-by attributes (*range*), and (iii) the fraction of tuples with uncertain order-by attributes. `Imp` is the fastest method, with an overhead of deterministic query processing between 3.5 (top-k) and 10 (full sorting). `Rewr` has higher overhead over `Det` than `MCDB`. Notably, the performance of `MCDB` and `Rewr` is independent of all three varied parameters. Uncertainty and *range* have small impact on the performance of `Imp` while computing top-k results is significantly faster than full sorting when k is small.



(a) Varying uncertainty (b) Varying range

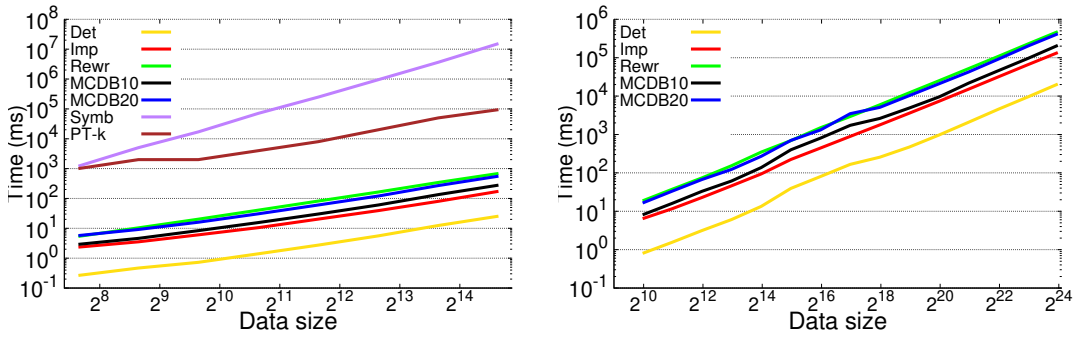
Figure 6.11. Sorting microbenchmarks - approximation quality

Accuracy: Fig. 6.11 shows the error of the bounds generated by `Imp` (`Rewr` produces identical outputs) and `MCDB`. Recall that `Imp` is guaranteed to over-approximate the correct bounds, while `MCDB` is guaranteed to under-approximate the bounds, because it does not compute all possible results. We measure the size of the bounds relative to the size of the correct bound (as computed by `Symb` and `PT-k`), and then take the average over all normalized bound sizes. In all cases our approach produces bounds that are closer to the exact bounds than `MCDB` (∼30% over-approximation versus ∼70% under-approximation in the worst case). We further note that an over-approximation of possible answers is often preferable to an under-approximation because no possible results will be missed.



(a) Varying uncertainty (b) Varying range

Figure 6.12. Window microbenchmarks - approximation quality

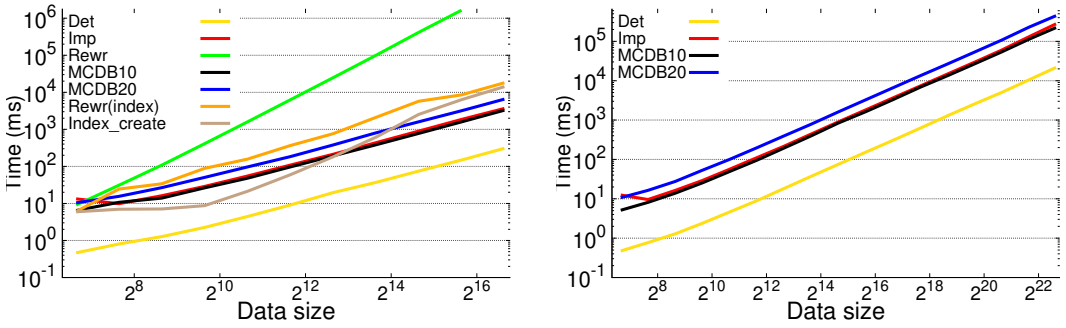


(a) Smaller datasets (b) Smaller datasets

Figure 6.13. Sorting performance varying dataset size

6.4.1.2 Windowed Aggregation. Scaling Data Size: Fig. 6.14 shows the runtime of windowed aggregation when varying dataset size. We compare two variants of our rewrite-based approach which uses a range overlap join to determine which tuples could possibly belong to a window. `Rewr(Index)` uses a range index supported by Postgres. We show index creation time and query time separately. We exclude `Symb`, because for more than 1k tuples, Z3 exceeds the maximal allowable call stack depth and crashes. The performance of `Imp` is roughly on par with `MCDB10`. `Rewr(Index)` is almost as fast as `MCDB20`, but is 5 × slower than `Imp`.

Varying window spec, Ranges, and Rate: Table 6.10 shows the runtime of windowed aggregation varying the value ranges of uncertain attribute (on all columns), percentage of uncertain tuples, and window size. For `Imp` (Table 6.10a) we use a



(a) Smaller datasets (b) Larger datasets

Figure 6.14. Windowed aggregation performance varying dataset size

query without partition-by. We also compare runtime of our rewriting based approach (Table 6.10b) using both partition-by and order-by on 8k rows. `Imp` exhibits similar runtime to `MCDB10` and outperforms `MCDB20`. Doubling the window size have only a slight impact (about 10%) on our implementation performance. `Rewr` is slower than `MCDB` by several magnitudes due to the range-overlap join. Our techniques are not significantly affected by the range and uncertainty rate.

6.4.2 Sorting and Top-k on Real World Datasets. We evaluate our approach on real datasets (Iceberg [103], Chicago crime data [99], and Medicare provider data [100]) using realistic sorting and windowed aggregation queries. To prepare the datasets, we perform data cleaning methods (entity resolution and missing value imputation) that output a AU-DB encoding of the space of possible repairs. Table 6.11 shows the performance of real queries on these datasets reporting basic statistics (uncertainty and #rows).

We use the following queries.

iceberg:

Find top 3 sizes of ice-bergs mostly observed.

```
SELECT size,count(*) AS ct FROM iceberg
GROUP BY size
```

```
ORDER BY ct DESC LIMIT 3;
```

Window: For each day, find rolling sum of number of icebergs observed on that day and following 3 days.

```
SELECT date, sum(number) OVER (ORDER BY date
   BETWEEN CURRENT ROW AND 3 FOLLOWING) AS r_sum
FROM iceberg;
```

Crimes:

Rank: Find top three days with most incidents of crimes.

```
SELECT date, count(*) AS ct
   FROM crimes GROUP BY date
   ORDER BY ct DESC LIMIT 3;
```

Window: For each crime in 2016, find the earliest year among the crime itself and nearest crime at north and south of it.

```
SELECT rid, min(year) OVER
   (ORDER BY latitude BETWEEN
      1 PRECEDING AND 1 FOLLOWING) AS min_year
FROM crimes WHERE year='2016';
```

healthcare:

Rank: Find top 5 facility with highest score on MRSA Bacteremia.

```
SELECT facility_id,facility_name,score FROM healthcare
WHERE measure_name = 'MRSA␣Bacteremia'
ORDER BY score LIMIT 5;
```

Window: get in-line rank of facility on MRSA Bacteremia scores.

```
SELECT facility_id,facility_name,count(*) OVER
(ORDER BY score DESC) AS rank
FROM healthcare
WHERE measure_name = 'MRSA␣Bacteremia';
```

For sorting and top-k queries that contain aggregation which is common in real use-cases, we only measure the performance of the sorting/top-k part over pre-aggregated data (see [40] for an evaluation of the performance of aggregation over AU-DBs). In general, our approach (`Imp`) is faster than `MCDB20`. `Symb` and `PT-k` are significantly more expensive. Table 6.12 shows the approximation quality for our approach and `MCDB`. Our approach has precision close to 100% except for sorting on the Iceberg dataset which has a larger fraction of uncertain tuples and wider ranges of uncertain attribute values due to the pre-aggregation. `MCDB` has lower recall on Iceberg and Healthcare sorting queries since these two datasets have more uncertain tuples (10 times more than the Crimes dataset). Table 6.13 shows the approximation quality of our approach and `MCDB` for windowed aggregation queries. We measured both the approximation quality of grouping of tuples to windows and for the aggregation result values. For Crimes and Iceberg, the aggregation accuracy is affected by the partition-by/order-by attribute accuracy and the uncertainty of the aggregation attribute itself. The healthcare query computes a count, i.e., there is no uncertainty in the aggregation attribute and approximation quality is similar to the one for sorting. Overall, we provide good approximation quality at a significantly lower cost than the two exact competitors.

Table 6.6. Real world data - performance and accuracy

Datasets & Queries			Time (sec)	cert. tup.	attr. bounds min	attr. bounds max	pos.tup. by id	pos.tup. by val
Netflix [98] (1.9%, 2.1)		AU-DB	**0.011**	100%	1	1	100%	100%
	$Q_{n,1}$	Trio	0.900	100%	1	1	100%	100%
	SPJ	MCDB	0.049	N.A.	1	1	99.6%	98.5%
		UA-DB	0.006	100%	N.A.	N.A.	99.1%	97.3%
		AU-DB	**0.082**	100%	1	4	100%	100%
	$Q_{n,2}$	Trio	1.700	100%	1	1	98.8%	98.0%
	GB	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 [99] (0.1%, 3.2)		AU-DB	**1.58**	100%	1	1	100%	100%
	$Q_{c,1}$	Trio	59.0	100%	1	1	100%	100%
	SPJ	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%
		AU-DB	**2.09**	100%	1	1.01	100%	100%
	$Q_{c,2}$	Trio	103.1	100%	1	1	100%	100%
	GB	MCDB	5.24	N.A.	0.99	0	100%	$\sim$ 0%
		UA-DB	0.47	0%	N.A.	N.A.	100%	$\sim$ 0%
Healthcare [100] (1.0%, 2.7)		AU-DB	**0.179**	99.5%	1	1	100%	100%
	$Q_{h,1}$	Trio	20.6	100%	1	1	100%	100%
	SPJ	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%
		AU-DB	**0.859**	100%	1	45	100%	100%
	$Q_{h,2}$	Trio	29.2	100%	1	1	100%	100%
	GB	MCDB	2.31	N.A.	0.78	1	100%	$\sim$ 0%
		UA-DB	0.235	0%	N.A.	N.A.	100%	$\sim$ 0%

Table 6.7. Microbenchmark Configurations

(a) Configurations

Test	**#Rows**	**Domain**	**Range**	**Uncert.%**	**CT**
Groupby	35k	[1,100]	5	5%	2^5
Aggregation	35k	[1,100]	5	5%	2^5
Range	35k	[1,100k]	5k-100k	10%	$2^{2,5,8,9}$
Compression	10k	[1,10k]	20	2%	$2^{1\sim16}$
Join	5k-20k	[1,1k]	15	3%	non,$2^{2,5,8,10}$
Multi-join	4k	[1,4k]	300	3%,10%	non,$2^{2,4,6,8}$

(b) Queries

Test	**Query**
Groupby	`SELECT SUM(a0) FROM t GROUP BY [...]`
Aggregation	`SELECT [...] FROM t GROUP BY a0`
Range	`SELECT a0,SUM(a1) FROM t GROUP BY a0`
Compression	`SELECT a0,SUM(a1) FROM t GROUP BY a0`
Join	`SELECT * FROM t1 JOIN t2 ON t1.a0 = t2.a0`
Multi-join	`... (t1 JOIN t2 ON t1.a1 = t2.a0) JOIN t3 on t2.a1=t3.a0 ...`

Table 6.8. Data Descriptions

Dataset	#columns	#rows	**source**
Netflix	12	$> 6K$	`https://www.kaggle.com/shivamb/netflix-shows`
Crimes	22	$> 1.4M$	`https://www.kaggle.com/currie32/crimes-in-chicago`
Healthcare	15	$> 171K$	`https://data.medicare.gov/data/hospital-compare`

Table 6.9. Sorting and Top-K Microbenchmarks - Performance

Configurations	Det	Imp	Rewr	MCDB10	MCDB20
r=1k,u=5%	31.5ms	233.1ms	786.7ms	310.1ms	639.3ms
r=10k,u=5%	30.9ms	286.1ms	792.6ms	314.3ms	621.2ms
r=1k,u=20%	31.8ms	266.3ms	794.9ms	325.8ms	651.2ms
r=1k,u=5%,k=2	13.4ms	48.3ms	750.4ms	149.1ms	295.2ms
r=1k,u=5%,k=10	13.4ms	48.2ms	751.1ms	150.4ms	296.1ms

Range(r),Uncertainty(u),k or full sorting

Table 6.10. Windowed aggregation microbenchmarks - Performance

(a) Order-by, Window size (w), Range (r), Uncertainty (u)

Configurations		Det	Imp	MCDB10	MCDB20
Order-by + Window size	w=3,r=1k,u=5%	85.3ms	895.3ms	948.6ms	1850.4ms
	w=3,r=10k,u=5%	87.1ms	899.7ms	931.3ms	1877.5ms
	w=3,r=1k,u=20%	88.7ms	903.2ms	944.7ms	1869.7ms
	w=6,r=1k,u=5%	86.2ms	1008.3ms	953.1ms	1885.1ms

(b) Order-by + partition-by, Window size (w), Range (r), Uncertainty (u)

Configurations		Det	Rewr	MCDB10	MCDB20
Order-by + Partiton-by + Window size	w=3,r=1k,u=5%	105.1ms	73.5s	1209.4ms	2127.1ms
	w=3,r=10k,u=5%	101.7ms	75.2s	1231.3ms	2142.9ms
	w=3,r=1k,u=20%	104.2ms	81.1s	1201.1ms	2102.3ms

Table 6.11. Real world data - performance

Datasets & Queries		`Imp` **(time)**	`Det` **(time)**	`MCDB20` **(time)**	`Rewr` **(time)**	`Symb` **(time)**	`PT-k` **(time)**
Iceberg [103] (1.1%, 167K)	Rank	0.816ms	0.123ms	2.337ms	1.269ms	278ms	1s
	Window	2.964ms	0.363ms	7.582ms	1.046ms	589ms	N.A.
Crimes [99] (0.1%, 1.45M)	Rank	1043.505ms	94.306ms	2001.12ms	14787.723ms	>10min	>10min
	Window	3.050ms	0.416ms	8.337ms	2.226ms	>10min	N.A.
Healthcare [100] (1.0%, 171K)	Rank	287.515ms	72.289ms	1451.232ms	4226.260ms	15s	8s
	Window	130.496ms	15.212ms	323.911ms	13713.218ms	>10min	N.A.

Table 6.12. Real world data - sort position accuracy and recall

Datasets & Measures		`Imp/Rewr`	`MCDB20`	`PT-k/Symb`
Iceberg [103]	bound accuracy	0.891	1	1
	bound recall	1	0.765	1
Crimes [99]	bound accuracy	0.996	1	1
	bound recall	1	0.919	1
Healthcare [100]	bound accuracy	0.990	1	1
	bound recall	1	0.767	1

Table 6.13. Real world data - windowed aggregation accuracy and recall

Datasets & Methods		Grouping/Order accuracy	Grouping/Order recall	Aggregation accuracy	Aggregation recall
Iceberg [103]	Imp/Rewr	0.977	1	0.925	1
	MCDB20	1	0.745	1	0.604
	Symb	1	1	1	1
Crimes [99]	Imp/Rewr	0.995	1	0.989	1
	MCDB20	1	0.916	1	0.825
	Symb	1	1	1	1
Healthcare [100]	Imp/Rewr	0.998	1	0.998	1
	MCDB20	1	0.967	1	0.967
	Symb	1	1	1	1

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

We proposed two models for dealing with uncertain data by using representations that compactly represent two or more possibilities to a tuple using a single tuple with proven correctness guarantees. UA-DBs are a novel and efficient way to represent uncertainty as bounds on certain answers. Being based on $\mathcal{K}$-relations, our approach applies to the incomplete version of any data model that can be encoded as $\mathcal{K}$-relations including set and bag semantics. Then on top of UA-DBs, we present attribute-annotated uncertain databases (AU-DBs) that support full relational algebra including aggregations, sortings(top-K) and windowed aggregations. Our approach stands out in that it is (i) more general in terms of supported queries than past works, (ii) has guaranteed `PTIME` data complexity, and (iii) compactly encodes over-approximations of incomplete databases. We demonstrated the efficiency of our approach over a wide range of datasets and queries, including TPC-H queries.

7.2 Future works

For future works, in order to extend our work beyond SQL query processing, one possible direction is to expand our uncertainty models to encompass a wider range of data-related operations, such as imperative programming (especially looping and recursion) which are wild used to model machine learning tasks. This would involve enabling processing of uncertain input values, and possibly even uncertain code. While keeping the ability to produce a deterministic execution result given a selected guess world, we can produce extra sensitivity information indicating the potential quality of the output to build an uncertain sensitivity aware machine learning model. An alternative approach involves expanding AU-DB to incorporate diverse forms of

uncertainty representations, such as continuous or possibility distributions, and devising semantics to propagate this information across queries, while keeping query processing overhead reasonably low.

APPENDIX A

APPENDIX

A.1 UA-DB Proofs

Proof: [Proof of Lemma 1] Proven by substitution of definitions.

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

$$\begin{aligned}
\text{PW}_i(\vec{k_1} +_{\mathcal{K}_W} \vec{k_2}) &= (\vec{k_1} +_{\mathcal{K}_W} \vec{k_2})[i] = \vec{k_1}[i] +_{\mathcal{K}} \vec{k_2}[i] \\
&= \text{PW}_i(\vec{k_1}) +_{\mathcal{K}} \text{PW}_i(\vec{k_2}) \\
\text{PW}_i(\vec{k_1} \cdot_{\mathcal{K}_W} \vec{k_2}) &= (\vec{k_1} \cdot_{\mathcal{K}_W} \vec{k_2})[i] = \vec{k_1}[i] \cdot_{\mathcal{K}} \vec{k_2}[i] \\
&= \text{PW}_i(\vec{k_1}) \cdot_{\mathcal{K}} \text{PW}_i(\vec{k_2})
\end{aligned}$$

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

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

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

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

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

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

$$
\begin{aligned}
&(k_1 \cdot_{\mathcal{K}} k_2) \\
\preceq_{\mathcal{K}} &(k_1 \cdot_{\mathcal{K}} k_2) +_{\mathcal{K}} (k_1 \cdot_{\mathcal{K}} {k_2}') +_{\mathcal{K}} ({k_1}' \cdot_{\mathcal{K}} k_2) +_{\mathcal{K}} ({k_1}' \cdot_{\mathcal{K}} {k_2}') \\
= &(k_1 +_{\mathcal{K}} {k_1}') \cdot_{\mathcal{K}} (k_2 +_{\mathcal{K}} {k_2}') = k_3 \cdot_{\mathcal{K}} k_4
\end{aligned}
$$

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

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

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

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

$$
\textsc{cert}_{\mathcal{K}}(\vec{k_1}) +_{\mathcal{K}} \textsc{cert}_{\mathcal{K}}(\vec{k_2}) \preceq_{\mathcal{K}} \textsc{cert}_{\mathcal{K}}(\vec{k_1} +_{\mathcal{K}_W} \vec{k_2})
$$

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

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

Proof: [Proof of Lemma 4] Consider an $\mathcal{RA}^+$ query Q and $\mathcal{D}$ a $\mathcal{K}_W$-database. To prove preservation of certain lower bound, we have to show that the result of $Q(\mathcal{D}^{\downarrow})$ is an under approximation for $Q(\mathcal{D})$, i.e., that for any tuple t we have $Q(\mathcal{D}^{\downarrow})(t) \preceq_{\mathcal{K}} \textsc{cert}_{\mathcal{K}}(Q(\mathcal{D}), t)$. Recall that $\mathcal{RA}^+$ queries over $\mathcal{K}_W$-relations and queries over $\mathcal{K}$-approximations are defined using the semiring addition and multiplication operations. Hence, the claim

$$Q(\mathcal{D}^{\downarrow})(t) \preceq_{\mathcal{K}} \textsc{cert}_{\mathcal{K}}(Q(\mathcal{D}), t)$$

follows immediately from the superadditivity and supermultiplicativity of $\textsc{cert}_{\mathcal{K}}$ (Lemma 3) and the fact that $\mathcal{D}^{\downarrow}$ is a exact approximation.

Proof: [Proof of Theorem 5] Since $\mathcal{D}^{\downarrow}$ is an under approxmation, for any tuple t we have $\mathcal{D}^{\downarrow}(t) \preceq_{\mathcal{K}} \textsc{cert}_{\mathcal{K}}(\mathcal{D}, t)$. We have to prove that for any t we have $Q(\mathcal{D}^{\downarrow})(t) \preceq_{\mathcal{K}} \textsc{cert}_{\mathcal{K}}(Q(\mathcal{D}), t)$. For that we show that for any $k_1, k_2 \in \mathcal{K}$ and $\vec{k_3}, \vec{k_4} \in K^W$ such that $k_1 \preceq_{\mathcal{K}} \textsc{cert}_{\mathcal{K}}(\vec{k_3})$ and $k_2 \preceq_{\mathcal{K}} \textsc{cert}_{\mathcal{K}}(\vec{k_4})$, we have $(k_1 +_{\mathcal{K}} k_2) \preceq_{\mathcal{K}} \textsc{cert}_{\mathcal{K}}(\vec{k_3} +_{\mathcal{K}_W} \vec{k_4})$ and $k_1 \cdot_{\mathcal{K}} k_2 \preceq_{\mathcal{K}} \textsc{cert}_{\mathcal{K}}(\vec{k_3} \cdot_{\mathcal{K}_W} \vec{k_4})$.

$$
\begin{aligned}
k_1 +_{\mathcal{K}} k_2 &\preceq_{\mathcal{K}} \textsc{cert}_{\mathcal{K}}(\vec{k_3}) +_{\mathcal{K}} \textsc{cert}_{\mathcal{K}}(\vec{k_4}) && \text{(by Lemma 2)} \\
&\preceq_{\mathcal{K}} \textsc{cert}_{\mathcal{K}}(\vec{k_3} +_{\mathcal{K}_W} \vec{k_4}) && \text{(by Lemma 3)} \\
k_1 \cdot_{\mathcal{K}} k_2 &\preceq_{\mathcal{K}} \textsc{cert}_{\mathcal{K}}(\vec{k_3}) \cdot_{\mathcal{K}} \textsc{cert}_{\mathcal{K}}(\vec{k_4}) && \text{(by Lemma 2)} \\
&\preceq_{\mathcal{K}} \textsc{cert}_{\mathcal{K}}(\vec{k_3} \cdot_{\mathcal{K}_W} \vec{k_4}) && \text{(by Lemma 3)}
\end{aligned}
$$

Since by assumption the input is an under approximation, we have $\mathcal{D}^{\downarrow}(t) \preceq_{\mathcal{K}} \textsc{cert}_{\mathcal{K}}(\mathcal{D}, t)$ for any tuple t. Thus, based on the property we have just proven and the

fact the $\mathcal{K}$-relational query semantics is defined based on the operations of semirings only, this implies that for any tuple t: $Q(\mathcal{D}^{\downarrow})(t) \preceq_{\mathcal{K}} \text{CERT}_{\mathcal{K}}(Q(\mathcal{D}), t)$. Thus, $Q(\mathcal{D}^{\downarrow})$ is an under approximation for $Q(\mathcal{D})$.

Proof: [Proof of Theorem 20] Let $\mathcal{D}^{\downarrow} = \mathsf{trans}^{UADB}_{\text{C-TABLE}}(\mathcal{D})$. A tuple t is labeled as certain iff $\phi_{\mathcal{D}}(t)$ is in CNF and $\models \phi_{\mathcal{D}}(t)$, which means the expression $\phi_{\mathcal{D}}$ is a tautology. By definition of C-tables, a tuple t exists in a possible world if $\phi_{\mathcal{D}}(t)$ evaluates to true in that possible world. Thus, t must exist in all possible worlds if $\phi_{\mathcal{D}}(t)$ is a tautology and $\mathcal{D}^{\downarrow}$ is an under approximation of certain.

Proof: [Proof of Theorem 19] Trivially holds, since a tuple is certain iff it is not optional and has only one alternative. Even though multiple x-tuples may share an alternative, the independence of x-tuples guarantees that this does not lead to additional certain tuples.

BIBLIOGRAPHY

[1] S. R. Jeffery, G. Alonso, M. J. Franklin, W. Hong, and J. Widom, "Declarative support for sensor data cleaning," in *PERVASIVE*, pp. 83–100, 2006.

[2] S. Sarawagi *et al.*, "Information extraction," *Foundations and Trends® in Databases*, vol. 1, no. 3, pp. 261–377, 2008.

[3] D. Olteanu, L. Papageorgiou, and S. J. van Schaik, "Pigora: An integration system for probabilistic data," in *ICDE*, pp. 1324–1327, 2013.

[4] P. Agrawal, A. D. Sarma, J. Ullman, and J. Widom, "Foundations of uncertain-data integration," *PVLDB*, vol. 3, no. 1-2, pp. 1080–1090, 2010.

[5] A. Halevy, A. Rajaraman, and J. Ordille, "Data integration: the teenage years," in *VLDB*, pp. 9–16, 2006.

[6] Y. Yang, N. Meneghetti, R. Fehling, Z. H. Liu, and O. Kennedy, "Lenses: An on-demand approach to etl," *PVLDB*, vol. 8, no. 12, pp. 1578–1589, 2015.

[7] W. Fan, "Dependencies revisited for improving data quality," in *PODS*, pp. 159–170, 2008.

[8] G. Beskales, I. F. Ilyas, L. Golab, and A. Galiullin, "Sampling from repairs of conditional functional dependency violations," *VLDBJ*, vol. 23, no. 1, pp. 103–128, 2014.

[9] M. Console, P. Guagliardo, L. Libkin, and E. Toussaint, "Coping with incomplete data: Recent advances," in *PODS*, pp. 33–47, ACM, 2020.

[10] D. Suciu, D. Olteanu, C. Ré, and C. Koch, "Probabilistic databases," *Synthesis Lectures on Data Management*, vol. 3, no. 2, pp. 1–180, 2011.

[11] S. Abiteboul, P. C. Kanellakis, and G. Grahne, "On the representation and querying of sets of possible worlds," *Theor. Comput. Sci.*, vol. 78, no. 1, pp. 158–187, 1991.

[12] T. Imielinski and W. L. Jr., "Incomplete information in relational databases," *J. ACM*, vol. 31, no. 4, pp. 761–791, 1984.

[13] F. Geerts, F. Pijcke, and J. Wijsen, "First-order under-approximations of consistent query answers," *International Journal of Approximate Reasoning*, vol. 83, pp. 337–355, 2017.

[14] R. Fink, J. Huang, and D. Olteanu, "Anytime approximation in probabilistic databases," *VLDBJ*, vol. 22, no. 6, pp. 823–848, 2013.

[15] P. Koutris and J. Wijsen, "Consistent query answering for primary keys and conjunctive queries with negated atoms," in *PODS*, 2018.

[16] S. Feng, A. Huber, B. Glavic, and O. Kennedy, "Uncertainty annotated databases - a lightweight approach for approximating certain answers," in *SIGMOD*, 2019.

[17] P. Agrawal, O. Benjelloun, A. D. Sarma, C. Hayworth, S. U. Nabar, T. Sugihara, and J. Widom, "Trio: A system for data, uncertainty, and lineage," in *VLDB*, 2006.

[18] M. Arenas, L. E. Bertossi, J. Chomicki, X. He, V. Raghavan, and J. P. Spinrad, "Scalar aggregation in inconsistent databases," *Theor. Comput. Sci.*, vol. 296, no. 3, pp. 405–434, 2003.

[19] F. N. Afrati and P. G. Kolaitis, "Answering aggregate queries in data exchange," in *PODS*, pp. 129–138, 2008.

[20] R. Murthy, R. Ikeda, and J. Widom, "Making aggregation work in uncertain and probabilistic databases," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 8, pp. 1261–1273, 2011.

[21] O. Kennedy and C. Koch, "Pip: A database system for great and small expectations," in *ICDE*, pp. 157–168, 2010.

[22] R. Fink, L. Han, and D. Olteanu, "Aggregation in probabilistic databases via knowledge compilation," *PVLDB*, vol. 5, no. 5, pp. 490–501, 2012.

[23] Y. Amsterdamer, D. Deutch, and V. Tannen, "Provenance for aggregate queries," in *PODS*, pp. 153–164, 2011.

[24] M. A. Soliman, I. F. Ilyas, and K. Chen-Chuan Chang, "Top-k query processing in uncertain databases," in *2007 IEEE 23rd International Conference on Data Engineering*, pp. 896–905, 2007.

[25] G. Cormode, F. Li, and K. Yi, "Semantics of ranking queries for probabilistic data and expected ranks," in *2009 IEEE 25th International Conference on Data Engineering*, pp. 305–316, 2009.

[26] C. Re, N. Dalvi, and D. Suciu, "Efficient top-k query evaluation on probabilistic data," in *2007 IEEE 23rd International Conference on Data Engineering*, pp. 886–895, 2007.

[27] J. Li, B. Saha, and A. Deshpande, "A unified approach to ranking in probabilistic databases," *Proc. VLDB Endow.*, vol. 2, p. 502–513, aug 2009.

[28] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang, "Probabilistic top-k and ranking-aggregate queries," *TODS*, vol. 33, no. 3, pp. 13:1–13:54, 2008.

[29] P. G. Kolaitis, E. Pema, and W.-C. Tan, "Efficient querying of inconsistent databases with binary integer programming," *PVLDB*, vol. 6, no. 6, pp. 397–408, 2013.

[30] P. G. Kolaitis and E. Pema, "A dichotomy in the complexity of consistent query answering for queries with two atoms," *Inf. Process. Lett.*, vol. 112, no. 3, pp. 77–85, 2012.

[31] L. E. Bertossi, *Database Repairing and Consistent Query Answering*. Synthesis Lectures on Data Management, Morgan & Claypool Publishers, 2011.

[32] A. D. Fuxman and R. J. Miller, "First-order query rewriting for inconsistent databases," in *ICDT*, 2005.

[33] M. Arenas, L. E. Bertossi, and J. Chomicki, "Consistent query answers in inconsistent databases," in *PODS*, 1999.

[34] T. J. Green, G. Karvounarakis, and V. Tannen, "Provenance semirings," in *PODS*, 2007.

[35] L. Libkin, "Sql's three-valued logic and certain answers," *TODS*, vol. 41, no. 1, pp. 1:1–1:28, 2016.

[36] R. Reiter, "A sound and sometimes complete query evaluation algorithm for relational databases with null values," *J. ACM*, vol. 33, no. 2, pp. 349–370, 1986.

[37] D. Piatov and S. Helmer, "Sweeping-based temporal aggregation," in *Advances in Spatial and Temporal Databases* (M. Gertz, M. Renz, X. Zhou, E. Hoel, W.-S. Ku, A. Voisard, C. Zhang, H. Chen, L. Tang, Y. Huang, C.-T. Lu, and S. Ravada, eds.), (Cham), pp. 125–144, Springer International Publishing, 2017.

[38] M. Hua, J. Pei, W. Zhang, and X. Lin, "Ranking queries on uncertain data: A probabilistic threshold approach," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, (New York, NY, USA), p. 673–686, Association for Computing Machinery, 2008.

[39] X. Zhang and J. Chomicki, "On the semantics and evaluation of top-k queries in probabilistic databases," in *2008 IEEE 24th International Conference on Data Engineering Workshop*, pp. 556–563, 2008.

[40] S. Feng, A. Huber, B. Glavic, and O. Kennedy, "Efficient uncertainty tracking for complex queries with attribute-level bounds," in *Proceedings of the 46th International Conference on Management of Data*, p. 528 – 540, 2021.

[41] M. Brachmann, W. Spoth, O. Kennedy, B. Glavic, H. Müller, S. Castel, C. Bautista, and J. Freire, "Your notebook is not crumby enough, replace it," in *CIDR*, 2020.

[42] P. Kumari, S. Achmiz, and O. Kennedy, "Communicating data quality in on-demand curation," in *QDB*, 2016.

[43] S. Feng, A. Huber, B. Glavic, and O. Kennedy, "Efficient uncertainty tracking for complex queries with attribute-level bounds (extended version)," 2021.

[44] R. Albright, A. J. Demers, J. Gehrke, N. Gupta, H. Lee, R. Keilty, G. Sadowski, B. Sowell, and W. M. White, "SGL: a scalable language for data-driven games," in *SIGMOD Conference*, pp. 1217–1222, ACM, 2008.

[45] A. J. Ratner, S. H. Bach, H. R. Ehrenberg, and C. Ré, "Snorkel: Fast training set generation for information extraction," in *SIGMOD*, pp. 1683–1686, 2017.

[46] P. Guagliardo and L. Libkin, "Correctness of sql queries on databases with nulls," *SIGMOD Record*, vol. 46, no. 3, pp. 5–16, 2017.

[47] P. Guagliardo and L. Libkin, "Making sql queries correct on incomplete databases: A feasibility study," in *PODS*, 2016.

[48] B. Sundarmurthy, P. Koutris, W. Lang, J. F. Naughton, and V. Tannen, "m-tables: Representing missing data," in *ICDT*, 2017.

[49] W. Lang, R. V. Nehme, E. Robinson, and J. F. Naughton, "Partial results in database systems," in *SIGMOD*, pp. 1275–1286, 2014.

[50] X. Liang, Z. Shang, S. Krishnan, A. J. Elmore, and M. J. Franklin, "Fast and reliable missing data contingency analysis with predicate-constraints," in *SIGMOD*, pp. 285–295, 2020.

[51] M. Brachmann, C. Bautista, S. Castelo, S. Feng, J. Freire, B. Glavic, O. Kennedy, H. Müller, R. Rampin, W. Spoth, and Y. Yang, "Data debugging and exploration with vizier," in *SIGMOD*, 2019.

[52] S. Feng, B. Glavic, and O. Kennedy, "Efficient approximation of certain and possible answers for ranking and window queries over uncertain data (extended version)," 2023.

[53] E. F. Codd, "Extending the database relational model to capture more meaning," *TODS*, vol. 4, no. 4, pp. 397–434, 1979.

[54] S. Abiteboul and G. Grahne, "Update semantics for incomplete databases," in *VLDB*, pp. 1–12, 1985.

[55] T. J. Green and V. Tannen, "Models for incomplete and probabilistic information," *IEEE Data Eng. Bull.*, vol. 29, no. 1, pp. 17–24, 2006.

[56] N. Dalvi and D. Suciu, "The dichotomy of conjunctive queries on probabilistic structures," in *PODS*, pp. 293–302, 2007.

[57] D. Olteanu, J. Huang, and C. Koch, "Approximate confidence computation in probabilistic databases," in *ICDE*, 2010.

[58] R. Fink, A. Hogue, D. Olteanu, and S. Rath, "Sprout^2: a squared query engine for uncertain web data," in *SIGMOD*, pp. 1299–1302, 2011.

[59] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. Jermaine, and P. J. Haas, "Mcdb: a monte carlo approach to managing uncertain data," in *SIGMOD*, 2008.

[60] A. Nandi, Y. Yang, O. Kennedy, B. Glavic, R. Fehling, Z. H. Liu, and D. Gawlick, "Mimir: Bringing ctables into practice," *CoRR*, vol. abs/1601.00073, 2016.

[61] J. Boulos, N. N. Dalvi, B. Mandhani, S. Mathur, C. Ré, and D. Suciu, "Mystiq: a system for finding more answers by using probabilities," in *SIGMOD*, pp. 891–893, 2005.

[62] W. Gatterbauer and D. Suciu, "Dissociation and propagation for approximate lifted inference with standard relational database management systems," *VLDBJ*, vol. 26, no. 1, pp. 5–30, 2017.

[63] A. D. Sarma, M. Theobald, and J. Widom, "Exploiting lineage for confidence computation in uncertain and probabilistic databases," in *ICDE*, 2008.

[64] E. V. Kostylev and P. Buneman, "Combining dependent annotations for relational algebra," in *ICDT*, pp. 196–207, 2012.

[65] A. Fuxman, E. Fazli, and R. Miller, "Conquer: Efficient management of inconsistent databases," in *SIGMOD*, pp. 155–166, 2005.

[66] S. Abiteboul, T.-H. H. Chan, E. Kharlamov, W. Nutt, and P. Senellart, "Aggregate queries for discrete and continuous probabilistic xml," in *ICDT*, pp. 50–61, 2010.

[67] J. Lechtenbörger, H. Shu, and G. Vossen, "Aggregate queries over conditional tables," *J. Intell. Inf. Syst.*, vol. 19, no. 3, pp. 343–362, 2002.

[68] C. Ré and D. Suciu, "The trichotomy of having queries on a probabilistic database," *VLDBJ*, vol. 18, no. 5, pp. 1091–1116, 2009.

[69] A. L. P. Chen, J.-S. Chiu, and F. S.-C. Tseng, "Evaluating aggregate operations over imprecise data," *IEEE Trans. Knowl. Data Eng.*, vol. 8, no. 2, pp. 273–284, 1996.

[70] T. S. Jayram, S. Kale, and E. Vee, "Efficient aggregation algorithms for probabilistic data," in *SODA*, pp. 346–355, 2007.

[71] D. Burdick, P. M. Deshpande, T. S. Jayram, R. Ramakrishnan, and S. Vaithyanathan, "Olap over uncertain and imprecise data," *VLDBJ*, vol. 16, no. 1, pp. 123–144, 2007.

[72] D. Calvanese, E. Kharlamov, W. Nutt, and C. Thorne, "Aggregate queries over ontologies," in *International Workshop on Ontologies and Information Systems for the Semantic Web (ONISW)*, pp. 97–104, 2008.

[73] E. V. Kostylev and J. L. Reutter, "Answering counting aggregate queries over ontologies of the dl-lite family," in *AAAI*, 2013.

[74] M. Yang, H. Wang, H. Chen, and W.-S. Ku, "Querying uncertain data with aggregate constraints," in *SIGMOD*, pp. 817–828, 2011.

[75] Y. Sismanis, L. Wang, A. Fuxman, P. J. Haas, and B. Reinwald, "Resolution-aware query answering for business intelligence," in *ICDE*, pp. 976–987, 2009.

[76] W. Lipski, "On semantic issues connected with incomplete information databases," *TODS*, vol. 4, no. 3, pp. 262–296, 1979.

[77] M. Console, P. Guagliardo, and L. Libkin, "Fragments of bag relational algebra: Expressiveness and certain answers," in *ICDT*, pp. 8:1–8:16, 2019.

[78] A. Calì, D. Lembo, and R. Rosati, "On the decidability and complexity of query answering over inconsistent and incomplete databases," in *PODS*, 2003.

[79] J. Wijsen, "Certain conjunctive query answering in first-order logic," *TODS*, vol. 37, no. 2, pp. 9:1–9:35, 2012.

[80] J. Wijsen, "On the first-order expressibility of computing certain answers to conjunctive queries over uncertain databases," in *PODS*, 2010.

[81] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang, "Probabilistic top-k and ranking-aggregate queries," *ACM Trans. Database Syst.*, vol. 33, Sept. 2008.

[82] A. Amarilli, M. L. Ba, D. Deutch, and P. Senellart, "Possible and certain answers for queries over order-incomplete data," in *Proc. TIME*, (Mons, Belgium), pp. 4:1–4:19, oct 2017.

[83] A. Amarilli, M. L. Ba, D. Deutch, and P. Senellart, "Computing possible and certain answers over order-incomplete data," *Theor. Comput. Sci.*, vol. 797, pp. 42–76, 2019.

[84] A. Amarilli, M. L. Ba, D. Deutch, and P. Senellart, "Provenance for nondeterministic order-aware queries," *Prepr int: http://a3nm. net/publications/amarilli2014provenance. pdf*, 2014.

[85] P. Tuma, *Implementing Historical Aggregates in Tempis*. Wayne State University, 1993.

[86] B. Moon, I. Lopez, and V. Immanuel, "Scalable algorithms for large temporal aggregation," in *Proceedings of 16th International Conference on Data Engineering (Cat. No.00CB37073)*, pp. 145–154, 2000.

[87] N. Kline and R. Snodgrass, "Computing temporal aggregates," in *Proceedings of the Eleventh International Conference on Data Engineering*, pp. 222–231, 1995.

[88] J. Yang and J. Widom, "Incremental computation and maintenance of temporal aggregates," in *Proceedings 17th International Conference on Data Engineering*, pp. 51–60, 2001.

[89] D. Zhang, A. Markowetz, V. Tsotras, D. Gunopulos, and B. Seeger, "Efficient computation of temporal aggregates with range predicates," in *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '01, (New York, NY, USA), p. 237–245, Association for Computing Machinery, 2001.

[90] L. Antova, C. Koch, and D. Olteanu, "Maybms: Managing incomplete information with probabilistic world-set decompositions," in *ICDE*, pp. 1479–1480, 2007.

[91] F. Geerts and A. Poggi, "On database query languages for k-relations," *J. Applied Logic*, vol. 8, no. 2, pp. 173–185, 2010.

[92] S. Burris and H. Sankappanavar, *A Course in Universal Algebra*. Springer, 2012.

[93] S. Feng, A. Huber, B. Glavic, and O. Kennedy, "Uncertainty annotated databases - a lightweight approach for approximating certain answers (extended version)," Tech. Rep. CoRR, `https://arxiv.org/pdf/1904.00234`, 2019.

[94] M. Y. Vardi, "Querying logical databases," *J. Comput. Syst. Sci.*, vol. 33, no. 2, pp. 142–160, 1986.

[95] J. Li, B. Saha, and A. Deshpande, "A unified approach to ranking in probabilistic databases," *VLDBJ*, vol. 20, no. 2, pp. 249–275, 2011.

[96] L. Antova, T. Jansen, C. Koch, and D. Olteanu, "Fast and simple relational processing of uncertain data," in *ICDE*, 2008.

[97] T. P. P. Council, "TPC-H specification." http://www.tpc.org/tpch/.

[98] "Netflix dataset," `https://www.kaggle.com/shivamb/netflix-shows`.

[99] "Chicago crimes dataset," https://www.kaggle.com/currie32/crimes-in-chicago.

[100] "Medicare hospital dataset," https://data.medicare.gov/data/hospital-compare.

[101] Y. Amsterdamer, D. Deutch, and V. Tannen, "Provenance for aggregate queries," in *PODS*, pp. 153–164, 2011.

[102] L. M. de Moura and N. Bjørner, "Z3: an efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings* (C. R. Ramakrishnan and J. Rehof, eds.), vol. 4963 of *Lecture Notes in Computer Science*, pp. 337–340, Springer, 2008.

[103] "Iceberg dataset," https://nsidc.org/data/g00807.