Provenance for Aggregate Queries *

Yael Amsterdamer
Tel Aviv University
and University of Pennsylvania
yaelamst@post.tau.ac.il

Daniel Deutch
Ben Gurion University
and University of Pennsylvania
deutchd@cs.bgu.ac.il

Val Tannen
University of Pennsylvania
val@cis.upenn.edu

ABSTRACT

We study in this paper provenance information for queries with *aggregation*. Provenance information was studied in the context of various query languages that do not allow for aggregation, and recent work has suggested to capture provenance by annotating the different database tuples with elements of a *commutative semiring* and propagating the annotations through query evaluation. We show that aggregate queries pose novel challenges rendering this approach inapplicable. Consequently, we propose a new approach, where we annotate with provenance information not just tuples but also the *individual values* within tuples, using provenance to describe the values computation. We realize this approach in a concrete construction, first for "simple" queries where the aggregation operator is the last one applied, and then for arbitrary (positive) relational algebra queries with aggregation; the latter queries are shown to be more challenging in this context. Finally, we use aggregation to encode queries with *difference*, and study the semantics obtained for such queries on provenance annotated databases.

Categories and Subject Descriptors

H.2.1 [**Database Management**]: [Data Models]

General Terms

Algorithms, Theory

1. INTRODUCTION

The annotation of the results of database transformations with provenance information has quite a few applications [17, 4, 36, 9, 10, 38, 35, 23, 25, 39, 27, 2]. Recent work [22, 15, 19] has proposed a framework of *semiring annotations* that allows us to state formally what is expected of such provenance information. These papers have developed the framework for the positive fragment of the relational algebra (as well as for Datalog, the positive Nested Relational Calculus, and some query languages on trees/XML). The main goal of this paper is to extend the framework to *aggregate operations*.

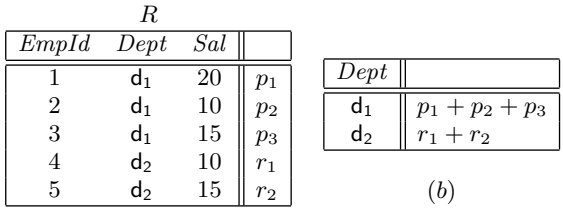

Figure 1: Projection on annotated relations

In the perspective promoted by these papers, *provenance* is a general form of annotation information that can be specialized for different purposes, such as multiplicity, trust, cost, security, or identification of "possible worlds" which in turn applies to incomplete databases, deletion propagation, and probabilistic databases. In fact, the introduction of the framework in [22] was motivated by the need to track trust and deletion propagation in the Orchestra system [21]. What makes such a diversity of applications possible is that each is captured by a different semiring, while provenance is represented by elements of a semiring of *polynomials*. One then relies on the property that any semiring-annotation semantics *factors* through the provenance polynomials semantics. This means that storing provenance polynomials allows for many other practical applications. For example, to capture access control, where the access to different tuples require different security credentials, we can simply evaluate the polynomials in the *security semiring*, and propagate the security annotations through query evaluation (see Section 2.1), assigning security levels to query results.

Let us briefly illustrate *deletion propagation* as an application of provenance. Consider a simple example of an employee/department/salary relation R shown in Figure 1(a).

The variables p_1, p_2, p_3, r_1, r_2 can be thought of as tuple identifiers and in the framework of *provenance polynomials* [22] they are the "provenance tokens" or "indeterminates" out of which provenance is built. We denote by $\mathbb{N}[X]$ the set of provenance polynomials (here $X = \{p_1, p_2, p_3, r_1, r_2\}$). R can be seen as an $\mathbb{N}[X]$-annotated relation; as defined in [22] the evaluation of query, for example $\Pi_{Dept}R$, produces another $\mathbb{N}[X]$-annotated relation, in this example the one shown in Figure 1(b). Intuitively, in this simple example,

*This work has been partially funded by the NSF grant IIS-0629846, the NSF grant IIS-0803524, and by the ERC grant Webdam under agreement 226513.

PODS'11, June 13–15, 2011, Athens, Greece.

Dept	*SalMass*	
d_1	45	$p_1p_2p_3$
d_1	30	$p_1p_2\widehat{p_3}$
d_1	35	$p_1\widehat{p_2}p_3$
d_1	25	$\widehat{p_1}p_2p_3$
d_1	20	$p_1\widehat{p_2}\widehat{p_3}$
d_1	10	$\widehat{p_1}p_2\widehat{p_3}$
d_1	15	$\widehat{p_1}\widehat{p_2}p_3$
...	...	...

(a)

Dept	*SalMass*	
d_1	30	p_1p_2
d_1	20	$p_1\widehat{p_2}$
d_1	10	$\widehat{p_1}p_2$
...	...	...

(b)

Figure 2: A naive approach to aggregation

the summation in the annotation of every result tuple is over the identifiers of its alternative origins [1].

Now, the result of propagating the deletions of tuples with *EmpId* 3 and 5 in R is obtained by simply setting $p_3 = r_2 = 0$ in the answer. We get the same two tuples in the query answer but their provenances change to $p_1 + p_2$ and r_1, respectively. If the tuple with *EmpId* 4 is also deleted from R then we also set $r_1 = 0$, and the second tuple in the answer is deleted because its provenance has now become 0. This algebraic treatment of deletions is related to the counting algorithm for view maintenance [24], but is more general as it incrementally maintains not just the data but also the provenance.

An intuitive way of understanding what happens is that provenance-aware evaluation of queries conveniently "commutes" with deletions. In fact, in [22, 15] this intuition is captured formally by theorems that state that query evaluation commutes with semiring homomorphisms. The factorization through provenance relies on this and on the fact that the polynomial provenance semiring is "freely generated". All applications of provenance polynomials we have listed, for trust, security, etc., are based on these theorems.

Thus, commutation with homomorphisms is an essential criterion for our proposed framework extension to aggregate operations. However, in Section 3.1 we prove that the framework of semiring-annotated relations introduced in [22] cannot be extended to handle aggregation while both satisfying commutation with homomorphisms and working as usual on set or bag relations.

If the semiring operations are not enough then perhaps we can add others? This is a natural idea so we illustrate it on the same R in Figure 1(a) and we use again the necessity to support deletion propagation to guide the approach. Consider the query that groups by *Dept* and sums *Sal*. The result of the summation depends on which tuples participate in it. To provide enough information to obtain all the possible summation results for all possible sets of deletions, we could use the representation in Figure 2(a) where we add to the semiring operations an unary operation $\widehat{\ }$ with the property that $\widehat{p} = 1$ whenever $p = 0$. This will indeed satisfy the deletion criterion. For example when the tuple with *Id* 3 is deleted we get the relation in Figure 2(b). In fact, there exist semirings with the additional structure needed to define $\widehat{\ }$. For example in the semiring of polynomials with *integer* coefficients, $\mathbb{Z}[X]$, we can take $\widehat{p} = 1 - p$ while in the semiring of boolean expressions with variables from X, BoolExp(X), we can take $\widehat{p} = \neg p$. The latter is essentially the approach taken in [32]. However, whether we use $\mathbb{Z}[X]$ or BoolExp(X), we still have, in the worst case, *exponentially many different results* to account for, at least in the case of summation (a lower bound recognized also in [32]). It follows that summation in particular (and therefore any uniform approach to aggregation) cannot be represented with a feasible amount of annotation as long as the annotation stays at the tuple level.

Instead, we will present a provenance representation for aggregation results that leads only to a *poly-size increase* with respect to the size of the input database, one that we believe is tractably implementable using methods similar to the ones used in Orchestra [21]. We achieve this via a more radical approach: we annotate with provenance information not just the tuples of the answer but also *the manner in which the values in those tuples are computed*.

We can gain intuition towards our representation from the particular case of bags, which are in fact $\mathbb{N}$-relations, i.e., relations whose tuples are annotated with elements of the semiring $(\mathbb{N}, +, \cdot, 0, 1)$. Assume that R in Figure 1(a) is such a relation, i.e., $p_1, \ldots, r_2 \in \mathbb{N}$ are tuple multiplicities. Then, after sum-aggregation the value of the attribute *SalMass* in the tuple with *Dept* d_1 is computed by $p_1 \times 20 + p_2 \times 10 + p_3 \times 15$. Now, if the multiplicities are, for example $p_1 = 2, p_2 = 3, p_3 = 1$ then the aggregate value is 85. But what if R is a relation annotated with provenance polynomials rather than multiplicities? Then, the aggregate value does not correspond to any number.

We will make $p_1 \times 20$ into an *abstract* construction, $p_1 \otimes 20$ and the aggregate value will be the formal expression $p_1 \otimes 20 + p_2 \otimes 10 + p_3 \otimes 15$.

Intuitively, we are embedding the domain of sum-aggregates, i.e., the reals $\mathbb{R}$, into a larger domain of formal expressions that capture how the sum-aggregates are computed from values annotated with provenance. We do the same for other kinds of aggregation, for instance min-aggregation gives $p_1 \otimes 20 \min p_2 \otimes 10 \min p_3 \otimes 15$. We call these *annotated aggregate* expressions.

In this paper we consider only aggregations defined by commutative-associative operations [2]. Specifically, our framework can accommodate aggregation based on any *commutative monoid*. For example the commutative monoid for summation is $(\mathbb{R}, +, 0)$ while the one for min is $(\mathbb{R}^\infty, \min, \infty)$[3].

To combine an aggregation monoid M with an annotation semiring K, in a way capturing aggregates over K-relations, we propose the use of the algebraic structure of *K-semimodules* (see Section 2.2). Semimodules are a way of generalizing (a lot!) the operations considered in linear algebra. Its "vectors" form only a commutative monoid (rather than an abelian group) and its "scalars" are the elements of K which is only a commutative semiring (rather than a field).

In general, a commutative monoid M does not have an obvious structure of K-semimodule. To make it such we may need to add new elements corresponding to the scalar multiplication of elements of M with elements from K, thus ending up with the formal expressions that represent aggregate computations, motivated above, as elements of a *tensor product* construction $K \otimes M$. We show that the use of tensor product expressions as a formal representation of aggrega-

[1] We explain how the annotations of query results are computed in Section 2.1

[2] As shown in [33], for list collections it also makes sense to consider non-commutative aggregations.

[3] K-annotated relations with *union* (see Section 2.1) also form such a structure.

tion result is effective in managing the provenance of "simple aggregate" queries, namely queries where the aggregation operators are the last ones applied.

We show that certain semirings are "compatible" with certain monoids, in the sense that the results of computation done in $K \otimes M$ may be mapped to M, faithfully representing the aggregation results. Interestingly, compatibility is aligned with common wisdom: it is known that some (idempotent) aggregation functions such as MIN and MAX work well for set relations, while SUM and PROD require the treatment of relations as bags. We show that non-idempotent monoids are compatible only with "bag" semirings, from which there exists an homomorphism to $\mathbb{N}$.

In general, aggregation results may be used by the query as the input to further operators, such as value-based joins or selections. Here the formal representation of values leads to a difficulty: the truth values of comparison operators on such formal expressions is undetermined! Consequently, we extend our framework and construct semirings in which formal comparison expressions between elements of the corresponding semimodule are elements of the semiring. This means that an expression like $[p_1 \otimes 20 = p_2 \otimes 10 + p_3 \otimes 15]$ may be used as part of the provenance of a tuples in the join result. This expression is simply treated as a new provenance token (with constraints), until p_1, p_2, p_3 are assigned e.g. values from $\mathbb{B}$ or $\mathbb{N}$, in which case we can interpret both sides of the equality as elements of the monoid and determine the truth value of the equality (see Section 4). We show in Section 4 that this construction allows us to manage provenance information for arbitrary queries with aggregation, while *keeping the representation size polynomial in the size of the input database.*

We note in this context that provenance expressions are inherently large and it may be difficult for humans to read and understand them. In this paper we will present simple examples that are human-readable, to ease the understanding of the introduced concepts. However, in general, the provenance expressions that we will introduce are mainly intended for automated processing. Presenting it to human users is the role of a provenance query language that takes these expressions as input, such as in [28].

The main result of this paper is providing, for the first time, a semantics for aggregation (including group by) on semiring-annotated relations that:

- Coincides with the usual semantics on set/bag relations for min/max/sum/prod.
- Commutes with homomorphisms of semirings
- Is representable with only poly-size overhead with respect to the database size.

A second result of this paper is a new semantics for *difference* on relations annotated with elements from *any* commutative semiring. This is done via an encoding of relational difference using nested aggregation. The fact that such an encoding can be done is known (see e.g. [29, 8]), but combined with our provenance framework, the encoding gives a semantics for "provenance-aware" difference. Our new semantics for $R - S$ is a hybrid of bag-style and set-style semantics, in the sense that tuples of R that appear in S do not appear in $R - S$ (i.e. a boolean negative condition is used), while those that do not appear in S appear in $R - S$ with the same annotation (multiplicity, if $K = \mathbb{N}$ is used) that they had in R. This makes the semantics different from the bag-monus semantics and its generalization to "monus-semirings" in [17] as well as from the "negative multiplicities" semantics in [20] (more discussion in Section 6). We examine the equational laws entailed by this new semantics, in contrast to those of previously proposed semantics for difference. In our opinion, this semantics is probably not the last word on difference of annotated relations, but we hope that it will help inform and calibrate future work on the topic.

Paper Organization. The rest of the paper is organized as follows. Section 2 describes and exemplifies the main mathematical ingredients used throughout the paper. Section 3 describes our proposed framework for "simple" aggregation queries, and this framework is extended in Section 4 to nested aggregation queries. We consider difference queries in Section 5. Related Work is discussed in Section 6, and we conclude in Section 7.

2. PRELIMINARIES

We provide in this section the algebraic foundations that will be used throughout the paper. We start by recalling the notion of semiring and its use in [22] to capture provenance for the SPJU algebra queries. We then consider aggregates, and show the new algebraic construction that is required to accurately support it.

2.1 Semirings and SPJU

We briefly review the basic framework introduced in [22]. A *commutative monoid* is an algebraic structure $(M, +_M, 0_M)$ where $+_M$ is an associative and commutative binary operation and 0_M is an identity for $+_M$. A monoid homomorphism is a mapping $h : M \to M'$ where M, M' are monoids, and $h(0_M) = 0_{M'}$,$h(a + b) = h(a) + h(b)$. We will consider database operations on relations whose tuples are annotated with elements from *commutative semirings*. These are structures $(K, +_K, \cdot_K, 0_K, 1_K)$ where $(K, +_K, 0_K)$ and $(K, \cdot_K, 1_K)$ are commutative monoids, $\cdot_K$ is distributive over $+_K$, and $a \cdot_K 0_K = 0 \cdot_K a = 0_K$. A semiring homomorphism is a mapping $h : K \to K'$ where K, K' are semirings, and $h(0_K) = 0_{K'}, h(1_K) = 1_{K'}$,$h(a + b) = h(a) + h(b)$, $h(a \cdot b) = h(a) \cdot h(b)$. Examples of commutative semirings are any commutative ring (of course) but also any distributive lattice, hence any boolean algebra. Examples of particular interest to us include the boolean semiring $(\mathbb{B}, \vee, \wedge, \bot, \top)$ (for usual set semantics), the natural numbers semiring $(\mathbb{N}, +, \cdot, 0, 1)$ (its elements are multiplicities, i.e., annotations that give bag semantics), and the *security* semiring $(\mathbb{S}, \min, \max, 0_\mathbb{S}, 1_\mathbb{S})$ where $\mathbb{S}$ is the ordered set, $1_\mathbb{S} < \mathsf{C} < \mathsf{S} < \mathsf{T} < 0_\mathbb{S}$ whose elements have the following meaning when used as annotations: $1_\mathbb{S}$: public ("always available"), C : confidential, S : secret, T : top secret, and $0_\mathbb{S}$ means "never available".

Certain semirings play an essential role in capturing provenance information. Given a set X of *provenance tokens* which correspond to "atomic" provenance information, e.g., tuple identifiers, the semiring of *polynomials* $(\mathbb{N}[X], +, \cdot, 0, 1)$ was shown in [22] to adequately, and most generally, capture provenance for positive relational queries. The *provenance interpretation* of the semiring structure is the following. The $+$ operation on annotations corresponds to *alternative use* of data, the $\cdot$ operation to *joint use* of data, 1 annotates data that is always and unrestrictedly available, and 0 annotates

absent data. The definition of the K-relational algebra (see bellow for union, projection and join) fits indeed this interpretation. Algebraically, $\mathbb{N}[X]$ is the commutative semiring freely generated by X, i.e., for any other commutative semiring K, any valuation of the provenance tokens $X \to K$ extends uniquely to a semiring homomorphism $\mathbb{N}[X] \to K$ (an evaluation in K of the polynomials). We say that any semiring annotation semantics *factors* through the provenance polynomials semantics, which means that for practical purposes storing provenance information suffices for many other applications too. Other semirings can also be used to capture certain forms of provenance, albeit less generally than $\mathbb{N}[X]$ [22, 19]. For example, boolean expressions capture enough provenance to serve in the intensional semantics of queries on incomplete [26] and probabilistic data [16, 40].

To define annotated relations we use here the named perspective of the relational model [1]. Fix a countably infinite *domain* $\mathbb{D}$ of values (constants). For any finite set U of attributes a tuple is a function $t : U \to \mathbb{D}$ and we denote the set of all such possible tuples by $\mathbb{D}^U$. Given a commutative semiring K, a *K-relation* (with schema U) is a function $R : \mathbb{D}^U \to K$ whose *support*, $\text{supp}(R) = \{t \mid R(t) \neq 0_K\}$ is *finite*. For a fixed set of attributes U we denote by K-Rel (when U is clear from the context) the set of K-relations with schema U. We also define a *K-set* to be a function $S : \mathbb{D} \to K$ again of finite support. We then define:

Union If $R_i : \mathbb{D}^U \to K,\ i = 1, 2$ then $R_1 \cup_K R_2 : \mathbb{D}^U \to K$ is defined by $(R_1 \cup_K R_2)(t) = R_1(t) +_K R_2(t)$. The definition of union of K-sets follows similarly.

We also define the *empty* K-relation (K-set) by $\emptyset_K(t) = 0_K$. It is easy to see that $(K\text{-Rel}, \cup_K, \emptyset_K)$ is a commutative monoid [4].Similarly, we get the commutative monoid of K-sets $(K\text{-Set}, \cup_K, \emptyset_K)$.

Given a named relational schema, K-databases are defined from K-relations just as relational databases are defined from usual relations, and in fact the usual (set semantics) databases correspond to the particular case $K = \mathbb{B}$. The (positive) *K-relational algebra* defined in [22] corresponds to a semantics on K-databases for the usual operations of the relational algebra. We have already defined the semantics of union above and we give here just two other cases referring the reader to [22] for the rest (for a tuple t and an attributes set U', $t|_{U'}$ is the restriction of t to U'):

Projection If $R : \mathbb{D}^U \to K$ and $U' \subseteq U$ then $\Pi_{U'} R : \mathbb{D}^{U'} \to K$ is defined by $(\Pi_{U'} R)(t) = \sum_K R(t')$ where the $+_K$ sum is over all $t' \in \text{supp}(R)$ such that $t'|_{U'} = t$.

Natural Join If $R_i : \mathbb{D}^{U_i} \to K,\ i = 1, 2$ then $R_1 \bowtie R2 : \mathbb{D}^{U_1 \cup U_2} \to K$ is defined by $(R_1 \bowtie R_2)(t) = R_1(t_1) \cdot_K R_2(t_2)$ where $t_i = t|_{U_i},\ i = 1, 2$.

2.2 Semimodules and aggregates

We will consider aggregates defined by commutative monoids. Some examples are SUM $= (\mathbb{R}, +, 0)$ for summation [5], MIN $= (\mathbb{R}^{\pm\infty}, \min, +\infty)$ for min, MAX $= (\mathbb{R}^{\pm\infty}, \max, -\infty)$ for max, and PROD $= (\mathbb{R}, \times, 1)$ for product.

In dealing with aggregates we have to extend the operation of a commutative monoid to operations on relations annotated with elements of semirings. This interaction will be captured by *semimodules*.

Definition 2.1. *Given a commutative semiring K, a structure $(W, +_W, 0_W, *_W)$ is a K-semimodule if $(W, +_W, 0_W)$ is a commutative monoid and $*_W$ is a binary operation $K \times W \to W$ such that (for all $k, k_1, k_2 \in K$ and $w, w_1, w_2 \in W$):*

$$k *_W (w_1 +_W w_2) = k *_W w_1 +_W k *_W w_2 \quad (1)$$
$$k *_W 0_W = 0_W \quad (2)$$
$$(k_1 +_K k_2) *_W w = k_1 *_W w +_W k_2 *_W w \quad (3)$$
$$0_K *_W w = 0_W \quad (4)$$
$$(k_1 \cdot_K k_2) *_W w = k_1 *_W (k_2 *_W w) \quad (5)$$
$$1_K *_W w = w \quad (6)$$

In any (commutative) monoid $(M, +_M, 0_M)$ define for any $n \in \mathbb{N}$ and $x \in M$

$$nx = x +_M \cdots +_M x \quad (n \text{ times})$$

in particular $0x = 0_M$. Thus M has a canonical structure of $\mathbb{N}$-semimodule. Moreover, it is easy to check that a commutative monoid M is a $\mathbb{B}$-semimodule if and only if its operation is *idempotent*: $x +_M x = x$. The K-relations themselves form a K-semimodule $(K\text{-Rel}, \cup_K, \emptyset_K, *_K)$ where $(k *_K R)(t) = k \cdot_K R(t)$ [6].

We now show, for any K-semimodule W, how to define *W-aggregation* of a K-set of elements from W. We assume that $W \subseteq \mathbb{D}$ and that we have just one attribute, whose values are all from W. Consider the K-set S such that $\text{supp}(S) = \{w_1, \ldots, w_n\}$ and $S(w_i) = k_i \in K, i = 1, \ldots, n$ (i.e., each w_i is annotated with k_i). Then, the result of W-aggregating S is defined as

$$\text{SetAgg}_W(S) = k_1 *_W w_1 +_W \cdots +_W k_n *_W w_n \quad \in W$$

For the empty K-set we define $\text{SetAgg}_W(\emptyset_K) = 0_W$. Clearly, SetAgg_W is a *semimodule homomorphism* [7]. Since all commutative monoids are $\mathbb{N}$-semimodules this gives the usual sum, prod, min, and max aggregations on bags. Since MIN and MAX are $\mathbb{B}$-semimodules this gives the usual min and max aggregation on sets [8].

Note that SetAgg is an operation on sets, not an operation on relations. In the sequel we show how to extend it to one.

2.3 A tensor product construction

More generally, we want to investigate M-aggregation on K-relations where M is a commutative monoid and K is some commutative semiring. Since M may not have enough elements to represent K-annotated aggregations we construct a K-semimodule in which M can be embedded, by transferring to semimodules the basic idea behind a standard algebraic construction, as follows.

Let K be any commutative semiring and M be any commutative monoid. We start with $K \times M$, denote its elements $k \otimes m$ instead of $\langle k, m \rangle$ and call them "simple tensors". Next we consider (finite) bags of such simple tensors, which, with

[4]In fact, it also has a semiring structure.

[5]COUNT is particular case of summation and AVG is obtained from summation and COUNT.

[6]In fact, it is the K semimodule freely generated by $\mathbb{D}^U$.

[7]In fact, it is the free homomorphism determined by the identity function on W.

[8]The fact that the right algebraic structure to use for aggregates is that of *semimodules* can be justified in the same way in which using semirings was justified in [22]: by showing how the laws of semimodules follow from desired equivalences between aggregation queries.

bag union and the empty bag, form a commutative monoid. It will be convenient to denote bag union by $+_{K\otimes M}$, the empty bag by $0_{K\otimes M}$ and to abuse notation denoting singleton bags by the unique element they contain. Then, every non-empty bag of simple tensors can be written (repeating summands by multiplicity) $k_1\otimes m_1 +_{K\otimes M} \cdots +_{K\otimes M} k_n\otimes m_n$. Now we define

$$k *_{K\otimes M} \sum k_i\otimes m_i = \sum (k \cdot_K k_i)\otimes m_i$$

Let $\sim$ be the smallest congruence w.r.t. $+_{K\otimes M}$ and $*_{K\otimes M}$ that satisfies (for all k, k', m, m'):

$$\begin{aligned}(k +_K k')\otimes m &\sim k\otimes m +_{K\otimes M} k'\otimes m\\ 0_K\otimes m &\sim 0_{K\otimes M}\\ k\otimes(m +_M m') &\sim k\otimes m +_{K\otimes M} k\otimes m'\\ k\otimes 0_M &\sim 0_{K\otimes M}\end{aligned}$$

We denote by $K \otimes M$ the set of *tensors* i.e., equivalence classes of bags of simple tensors modulo $\sim$. We can show that $K \otimes M$ forms a K-semimodule.

Lifting homomorphisms. Given a homomorphism of semirings $h : K \to K'$, and some commutative monoid M, we can "lift" h to a homomorphism of monoids in a natural way. The lifted homomorphism is denoted $h^M : K \otimes M \to K' \otimes M$ and defined by:

$$h^M(\sum k_i\otimes m_i) = \sum h(k_i)\otimes m_i$$

3. SIMPLE AGGREGATION QUERIES

In this section we begin our study of the "provenance-aware" evaluation of aggregate queries, focusing on "simple" such queries, that is, queries in which aggregations are done last; for example, un-nested SELECT FROM WHERE GROUP BY queries. This avoids the need to compare values which are the result of annotated aggregations and simplifies the treatment. The restriction is relaxed in the more general framework presented in Section 4.

The section is organized as follows. We list the desired features of a provenance-aware semantics for aggregation, and first try to design a semantics with these features, *without* using the tensor product construction, i.e. by simply working with K-relations as done in [22]. We show that this is impossible. Consequently, we turn to semantics that are based on combining aggregation with values via the tensor product construction. We propose such semantics that do satisfy the desired features, first for relational algebra with an additional AGG operator on relations (that allows aggregation of all values in a chosen attributes, but no grouping); and then for GROUP BY queries.

3.1 Semantic desiderata and first attempts

We next explain the desired features of a provenance-aware semantics for aggregation. To illustrate the difficulties and the need for a more complex construction, we will first attempt to define a semantics on K-relations, without using the tensor product construction of Section 2.3.

We consider a commutative semiring K (e.g., $\mathbb{B}, \mathbb{N}, \mathbb{N}[X], \mathbb{S}$, etc.) for tuple annotations and a commutative monoid M (e.g., SUM $= (\mathbb{R}, +, 0)$, PROD $= (\mathbb{R}, \times, 1)$, MAX $= (\mathbb{R}^{-\infty}, \max, -\infty)$, MIN $= (\mathbb{R}^{\infty}, \min, \infty)$ etc.) for aggregation. We will assume that the elements of M already belong to the database domain, $M \subseteq \mathbb{D}$.

We have recalled the semantics of SPJU queries in Section 2.1. Now we wish to add an M-aggregation operation AGG on relations. We then denote by SPJU-A the restricted class of queries consisting of any SPJU-expression followed possibly by just one application of AGG. This corresponds to SELECT AGG(*) FROM WHERE queries (no grouping).

For the moment, we do not give a concrete semantics to $\mathrm{AGG}_M(R)$, allowing any possible semantics where the result of $\mathrm{AGG}_M(R)$ is a K-relation. We note that $\mathrm{AGG}_M(R)$ should be defined iff R is a K-relation with one attribute whose values are in M.

What properties do we expect of a reasonable semantics for SPJU-A (including, of course, a semantics for $\mathrm{AGG}_M(R)$)? A basic sanity check is

Set/Bag Compatibility The semantics coincides with the usual one when $K = \mathbb{B}$ (sets) and $M =$ MAX or MIN, and when $K = \mathbb{N}$ (bags) and $M =$ SUM or PROD.

Note that we associate min and max with sets and sum and product with bags. Min and max work fine with bags too, but we get the same result if we convert a bag to a set (eliminate duplicates) and then apply them. Sum and product (in the context of other operations such as projection) require us to use bags semantics in order to work properly. This is well-known, but our general approach sheds further light on the issue by discussing such "compatibility" for arbitrary semirings and monoids in Section 3.4.

As discussed in the introduction, a fundamental desideratum with many applications is *commutation with homomorphisms*. Note that a semiring homomorphism $h : K \to K'$ naturally extends to a mapping $h_{Rel} : K\text{-Rel} \to K'\text{-Rel}$ via $h_{Rel}(R) = h \circ R$ (i.e. the homomorphism is applied on the annotation of every tuple), which then further extends to K-databases. With this, the second desired property is

Commutation with Homomorphisms Given any two commutative semirings K, K' and any homomorphism $h : K \to K'$, for any query Q, its semantics on K-databases and on K'-databases satisfy $h_{Rel}(Q(D)) = Q(h_{Rel}(D))$ for any K-database D.

It turns out that this property determines quite precisely the way in which tuple annotations are defined. We say that the semantics of an operation Ω on K-databases is *algebraically uniform* with respect to the class of commutative semirings if the annotations of the output $\Omega(D)$ are defined by the same (for all K) $\{+_K, \cdot_K, 0_K, 1_K\}$-expressions, where the elements in the expressions are the annotations of the input D. One can see that the definition of the SPJU-algebra is indeed algebraically uniform and was shown in [22, 15] to commute with homomorphisms. The connection between the two properties is general:

PROPOSITION 3.1. *A semantics commutes with homomorphisms iff it is algebraically uniform.*

After stating two of the desired properties, namely set/bag compatibility and commutation with homomorphisms we can already show that it is not possible to give a satisfactory semantics to the SPJU-A algebra within the framework used in [22] for the SPJU-algebra.

PROPOSITION 3.2. *There is no K-relation semantics for MAX-(or MIN-)aggregation that is both set-compatible and*

commutes with homomorphisms. Similarly, there is no K-relation semantics for SUM-aggregation that is both bag-compatible and commutes with homomorphisms.

Alternatively, one may consider going beyond semirings, to algebraic structures with additional operations. We have briefly explored the use of "negative" information in the introduction. As we show there, one could use the ring structure on $\mathbb{Z}[X]$ (the additional subtraction operation) or the boolean algebra structure on $\text{BoolExp}(X)$ (the additional complement operation) but the use of negative operation does not avoid the need to enumerate in separate tuples of the answer all the possible aggregation results given by subsets of the input. In the case of summation, at least, there are exponentially many such tuples. We reject such an approach and we state as an additional desideratum:

Poly-Size Overhead For any query Q and database D, the size of $Q(D)$, including annotations, should be only polynomial in the size of the databse D.

We shall next show a semantics to the SPJU-A -algebra that satisfies all three properties we have listed.

3.2 Annotations $\otimes$ values and SPJU-A

Let us fix a commutative monoid M (for aggregation) and a commutative semiring K (for annotation). The *inputs* of our queries are, as before, K-databases whose domain $\mathbb{D}$ includes the values M over which we aggregate. However, the outputs are more complicated. The basic idea for the semantics of aggregation was already shown in Section 2.2 where it is assumed that the domain of aggregation has a K-semimodule structure. As we have shown in Section 2.3, we can give a tensor product construction that embeds M in the K-semimodule $K \otimes M$ (note that this embedding is not always *faithful*, as discussed in Section 3.4).

For the output relations of our algebra queries, we thus need results of aggregation (i.e., the elements of $K \otimes M$) to also be part of the domain out of which the tuples are constructed. Thus, for the output domain we will assume that $K \otimes M \subseteq \mathbb{D}$, i.e. the result "combines annotations with values". The elements of M (e.g., real numbers for sum or max aggregation) are still present, but only via the embedding $\iota : M \to K \otimes M$ defined by $\iota(m) = 1_K \otimes m$.

Having annotations from K appear in the values will change the way in which we apply homomorphisms to query results, so to emphasize the change we will call (M, K)*-relations* the K-annotated relations over data domain $\mathbb{D}$ that includes $K \otimes M$. To summarize, the semantics of the SPJU-A -algebra will map databases of K-relations (with $M \subseteq \mathbb{D}$) to (M, K)-relations (with $K \otimes M \subseteq \mathbb{D}$).

As we define the semantics of the SPJU-A -algebra, we first note that for selection, projection, join and union the definition is the same as for the SPJU-algebra on K-databases. The last step of the query is aggregation, denoted $\text{AGG}_M(R)$, and is well-defined iff R is a K-relation with one attribute whose values are in the M subset of $\mathbb{D}$. To apply the definition that uses the semimodule structure (shown in Section 2.2), we convert R to an (M, K)-relation $\iota(R)$ by replacing each $m \in M$ with $\iota(m) = 1_K \otimes m \in K \otimes M$. Then, if $\text{supp}(R) = \{m_1, \ldots, m_n\}$ and $R(m_i) = k_i \in K, i = 1, \ldots, n$ (i.e., each m_i is annotated with k_i) we define $\text{AGG}_M(R)$ as a one-attribute relation, with one tuple, annotated with 1_K whose content is $\text{SetAgg}_{K\otimes M}(\iota(R))$; the latter is equal to

$$k_1 *_{K\otimes M} \iota(m_1) +_{K\otimes M} \cdots +_{K\otimes M} k_n *_{K\otimes M} \iota(m_n) = k_1 \otimes m_1 +_{K\otimes M} \cdots +_{K\otimes M} k_n \otimes m_n$$

We define the annotation of the only tuple in the output of AGG_M to be 1_K, since this tuple is always available. However, the *content* of this tuple does depend on R. For example, even when R is empty the output is not empty: by the semimodule laws, its content is $0_{K\otimes M} = \iota(0_M)$.

Commutation with Homomorphisms. We have explained in Section 2.3 how to lift a homomorphism $h : K \to K'$ to a homomorphism $h^M : K \otimes M \to K' \otimes M$. This allows us to lift h to a homomorphism h_{Rel} on (M, K)*-relations*, as follows. Let R be such a relation and recall that some values in R are elements of $K \otimes M$, and the annotations of these tuples are elements of K. Then $h_{Rel}(R)$ denotes the relation obtained from R by replacing every $k \in K$ with $h(k)$, and additionally replacing every $k \otimes M \in K \otimes M$ with $h^M(k \otimes m)$. All other values in R stay intact. Applying h_{Rel} on a (M, K)-database D amounts to applying h_{Rel} on each (M, K)-relation in D.

We can now state the main result for our SPJU-A -algebra:

THEOREM 3.3. *Let K, K' be semirings, $h : K \to K'$, Q an SPJU-A query and let M be a commutative monoid. For every (M, K)-database D, $Q(h_{Rel}(D)) = h_{Rel}(Q(D))$ if and only if h is a semiring homomorphism.*

The proof is by induction on the query structure, and is straightforward given that for the constructs of SPJU queries homomorphism commutation was shown in [22], while commutation for the new AGG_M construct follows directly from the definition.

EXAMPLE 3.4. *Consider the following $\mathbb{N}[X]$-relation R:*

Sal	
20	r_1
10	r_2
30	r_3

Let M be some commutative monoid, then $AGG_M(R)$ consist of a single tuple with value $r_1 \otimes 20 +_{K\otimes M} r_2 \otimes 10 +_{K\otimes M} r_3 \otimes 30$. The intuition is that this value captures multiple possible aggregation values, each of which may be obtained by mapping the r_i annotations to $\mathbb{N}$, standing for the multiplicity of the corresponding tuple. The commutation with homomorphism allows us to first evaluate the query and only then map the r_i's, changing directly the expression in the query result. For example, if $M = SUM$ and we map r_1 to 1,r_2 to 0,r_3 to 2, we obtain $1 \otimes 20 +_{K\otimes M} 2 \otimes 30 = 1 \otimes 20 +_{K\otimes M} 1 \otimes 30 +_{K\otimes M} 1 \otimes 30 = 1 \otimes 80$ (which corresponds to the M element 80). As another example, the commutation with homomorphisms allows us to propagate the deletion of the first tuple in R, by simply setting in the aggregation result $r_1 = 0$ (keeping the other annotations intact) and obtaining $2 \otimes 30 = (1+1) \otimes 30 = 1 \otimes 30 + 1 \otimes 30 = 1 \otimes (30+30) = 1 \otimes 60$.

We further demonstrate an application for *security*.

EXAMPLE 3.5. *Consider the following relation R, annotated by elements from the security semiring* $\mathbb{S}$.

Sal	
20	S
10	$1_{\mathbb{S}}$
30	S

Recall (from Section 2.1) the order relation $1_{\mathbb{S}} < \mathsf{C} < \mathsf{S} < \mathsf{T} < 0_{\mathbb{S}}$*; a user with credentials* cred *can only view tuples annotated with security level equal or less than* cred. *Now let* $M = MAX$ *and we obtain:* $AGG_{MAX}(R) = \mathsf{S}\otimes 20 +_{K\otimes M} 1_{\mathbb{S}}\otimes 10 +_{K\otimes M} \mathsf{S}\otimes 30 = \mathsf{S}\otimes(20 +_{MAX} 30) + 1_{\mathbb{S}}\otimes 10$ *and we get* $\mathsf{S}\otimes 30 + 1_{\mathbb{S}}\otimes 10$.

Assume now that we wish to compute the query results as viewed by a user with security credentials cred. *A naive computation would delete from R all tuples that require higher credentials, and re-evaluate the query (which in general may be complex). But observe that the deletion of tuples is equivalent to applying to R a homomorphism that maps every annotation* $t >$ cred *to* 0*, and* $t \leq$ cred *to* 1*. Using homomorphism commutation we can do better by applying this homomorphism only on the result representation (namely* $\mathsf{S}\otimes 30 + 1_{\mathbb{S}}\otimes 10$*). For example, for a user with credentials* C*, we map* S *to* 0 *and* $1_{\mathbb{S}}$ *to* 1*, and obtain* $0\otimes 30+1\otimes 10 = 1\otimes 10$*; similarly for a user with credentials* S *we get* $1\otimes 30+1\otimes 10 = 1\otimes(30 +_{MAX} 10) = 1 \otimes 30$.

We can show that the defined semantics fulfills the poly-size overhead property.

3.3 Group By

So far we have considered aggregation in a limited context, where the input relation contains a single attribute. In common cases, however, aggregation is used on arbitrary relations and in conjunction with grouping, so we next extend the algebra to handle such an operation. The idea behind the construction is quite simple: we separately group the tuples according to the values of their "group-by" attributes, and the aggregated values for each such group are computed similarly to the computation for the *AGG* operator. When considering the *annotation* of the aggregated tuple, we encounter a technical difficulty: we want this annotation to be equal 1_K if the input relation includes at least one tuple in the corresponding group, and 0_K otherwise (for intuition, consider the case of bag relations, in which the aggregated result can have at most multiplicity 1); we consequently enrich our structure to include an additional construct δ that will capture that, as follows:

DEFINITION 3.6. *A (commutative)* δ*-semiring is an algebraic structure* $(K, +_K, \cdot_K, 0_K, 1_K, \delta_K)$ *where* $(K, +_K, \cdot_K, 0_K, 1_K)$ *is a commutative semiring and* $\delta_K : K \to K$ *is a unary operation satisfying the "*δ*-laws"* $\delta_K(0_K) = 0_K$ *and* $\delta_K(n1_K) = 1_K$ *for all* $n \geq 1$*. If K and* K' *are* δ*-semirings then a* homomorphism *between them is a semiring homomorphism* $h : K \to K'$*, for which we have in addition* $h(\delta_K(k)) = \delta_{K'}(h(k))$.

The δ-laws completely determine $\delta_{\mathbb{B}}$ and $\delta_{\mathbb{N}}$. But they leave a lot of freedom for the definition of δ_K in other semirings; in particular for the security semiring, a reasonable choice for $\delta_{\mathbb{S}}$ is the identity function.

As with any equational axiomatization, we can construct the commutative δ-semiring *freely generated* by a set X, denoted $\mathbb{N}[X, \delta]$, by taking the quotient of the set of $\{+, \cdot, 0, 1, \delta\}$-algebraic expressions by the congruence generated by the equational laws of commutative semirings and the δ-laws. For example, if e and e' are elements of $\mathbb{N}[X, \delta]$ (i.e., congruence classes of expressions given by some representatives) then $e +_{\mathbb{N}[X,\delta]} e'$ is the congruence class of the expression $e + e'$. The elements of $\mathbb{N}[X, \delta]$ are not standard polynomials but certain subexpressions can be put in polynomial form, for example $\delta(2 + 3xy^2)$ or $3 + 2\delta(x^2 + 2y)z^2$.

We are now ready to define the group by (denoted *GB*) operation; subsequently we exemplify its use, including in particular the role of δ:

DEFINITION 3.7. *Let R be a K-relation on set of attributes U, let* $U' \subseteq U$ *be a subset of attributes that will be grouped and* $U'' \in U$ *be the subset of attributes with values in M (to be aggregated). We assume that* $U' \cap U'' = \emptyset$*. For a tuple t, we define* $T = \{t' \in supp(R) \mid \forall u \in U' \;\; t'(u) = t(u)\}$.

We then define the aggregation result $R' = GB_{U',U''}(R)$ *as follows:*

$$R'(t) = \begin{cases} \delta_K(\Sigma_{t' \in T} R(t')) & T \neq \phi, \text{ and} \\ & \forall u \in U'' \;\; t(u) = \Sigma_{t' \in T} R(t') \otimes t'(u) \\ 0 & \text{Otherwise.} \end{cases}$$

EXAMPLE 3.8. *Consider the relation R:*

Dept	*Sal*	
d_1	20	r_1
d_1	10	r_2
d_2	10	r_3

and a query $GB_{\{Dept\},Sal}R$*, where the monoid used is SUM. The result (denoted R') is:*

Dept	*Sal*	
d_1	$r_1\otimes 20 +_{K\otimes SUM} r_2\otimes 10$	$\delta_K(r_1 +_K r_2)$
d_2	$r_3\otimes 10$	$\delta_K(r_3)$

Each aggregated value (for each department) is computed very similarly to the computation in Example 3.4. Consider the provenance annotation of the first tuple: intuitively, we expect it to be 1_K *if at least one of the first two tuples of R exists, i.e. if at least one out of* r_1 *or* r_2 *is non-zero. Indeed the expression is* $\delta(r_1 +_K r_2)$ *and if we map* r_1, r_2 *to e.g.* 2 *and* 1 *respectively, we obtain* $\delta_{\mathbb{N}}(3) = 1$.

We use SPJU-AGB as the name for relational algebra with the two new operators AGG and GB. We note that the poly-size overhead property (w.r.t. data complexity) is still fulfilled for queries in SPJU-AGB ; commutation with homomorphism also extends to SPJU-AGB .

Recall that an additional desideratum from the semantics was bag / set compatibility. Recall that sets and bags are modeled by $K = \mathbb{N}$ and $K = \mathbb{B}$ respectively. We next study compatibility in a more general way, for arbitrary K and M.

3.4 Annotation-aggregation compatibility

The first desideratum we listed was an obvious sanity check: whatever semantics we define, when specialized to the familiar aggregates of max, min and summation, it should produce familiar results. Since we had to take an excursion through the tensor product $K \otimes M$, this familiarity is not immediate. However, the following proposition holds (its correctness will follow from theorems 3.12 and 3.13).

PROPOSITION 3.9. *In the following constructions:* $\mathbb{B} \otimes MAX$, $\mathbb{B} \otimes MIN$, *and* $\mathbb{N} \otimes SUM$, $\iota : M \to K \otimes M$ *where* $\iota(m) = 1_K \otimes m$ *is a monoid* isomorphism.

and this means our semantics satisfies the set/bag compatibility property because in these cases computing in $K \otimes M$ exactly mirrors computing in M.

But of course, we are also interested in working with other semirings, in particular the provenance semiring, for which $\mathbb{N}[X] \otimes M$ and M are in general not isomorphic (in particular, ι is not surjective and thus not an isomorphism). In fact, the whole point of working in $\mathbb{N}[X] \otimes$ MAX, for example, is to add annotated aggregate computations to the domain of values. Most of these do not correspond to actual real numbers as e.g. ι(MAX) is a strict subset of $\mathbb{N}[X] \otimes$ MAX (and similarly ι(SUM) is a strict subset of $\mathbb{N}[X] \otimes$SUM etc.). However, when provenance tokens are valuated to obtain set (or bag) instances, we can go back into ι(MAX) (or ι(SUM) etc.), and then we should obtain familiar results by "stripping off" the ι. It turns out that this works correctly with $\mathbb{N}[X]$ but not necessarily with arbitrary commutative semirings K. this is because ι in general may be *unfaithful* (not injective). Indeed $\iota : \text{SUM} \to \mathbb{B} \otimes \text{SUM}$ is not injective:

$$\iota(4) = \iota(2+2) = \iota(2) +_{K \otimes M} \iota(2) = \top \otimes 2 +_{K \otimes M} \top \otimes 2 =$$

$$= (\top \vee \top) \otimes 2 = \top \otimes 2 = \iota(2)$$

This is not surprising, as it is related to the well-known difficulty of making summation work properly with set semantics. In general, we thus define compatibility as follows:

DEFINITION 3.10. *We say that a commutative semiring* K *and a commutative monoid* M *are* compatible *if* ι *is injective.*

The point of the definition is that when there is compatibility, we can work in $K \otimes M$ and whenever the results belong to $\iota(M)$, we can *safely* read them as familiar answers from M. We give three theorems that capture some general conditions for compatibility.

First, we note that if we work with a semiring in which $+_K$ is idempotent, such as $\mathbb{B}$ or $\mathbb{S}$, a compatible monoid must also be idempotent (e.g. MIN or MAX but not SUM):

PROPOSITION 3.11. *Let* K *be some commutative semiring such that* $+_K$ *is idempotent, and let* M *be some commutative monoid. If* M *is compatible with* K*, then* $+_M$ *is idempotent.*

PROOF. $\iota(m) = 1_K \otimes m = (1_K +_K 1_K) \otimes m = 1_K \otimes m +_{K \otimes M} 1_K \otimes m = 1_K \otimes (m +_M m) = \iota(m +_M m)$ □

Nicely enough, idempotent aggregations are compatible with *every* annotation semiring K that is *positive with respect to* $+_K$. K is said to be positive with respect to $+_K$ if $k +_K k' = 0_K \Rightarrow k = k' = 0_K$. For instance, $\mathbb{B}$, $\mathbb{N}$, $\mathbb{S}$ and $\mathbb{N}[X]$ are such semirings (but not $(\mathbb{Z}, +, \cdot, 0, 1)$). The following theorem holds:

THEOREM 3.12. *If* M *is a commutative monoid such that* $+_M$ *is idempotent, then* M *is compatible with any commutative semiring* K *which is positive with respect to* $+_K$.

PROOF SKETCH. We define $h : K \otimes M \to M$ as $h(\sum_{i \in I} k_i \otimes m_i) = \sum_{j \in J} m_j$ where $J = \{j \in I | k_j \neq 0\}$. We can show that h is well-defined; since $\forall m \in M \;\; h \circ \iota(m) = m$, ι is injective and thus K and M are compatible. □

For general (and in particular non-idempotent) monoids (e.g. SUM) we identify a sufficient condition on K (which in particular holds for $\mathbb{N}[X]$), that allows for compatibility:

THEOREM 3.13. *Let* K *be a commutative semiring. If there exists a semiring homomorphism from* K *to* $\mathbb{N}$ *then* K *is compatible with all commutative monoids.*

PROOF SKETCH. Let h' be a homomorphism from K to $\mathbb{N}$, and M be an arbitrary commutative monoid. We define a mapping $h : K \otimes M \to M$ by $h(\Sigma k_i \otimes m_i) = \Sigma h'(k_i) m_i$. We can show that h is well-defined and that $h \circ \iota$ is the identity function hence ι is injective. □

COROLLARY 3.14. *The semiring of provenance polynomials* $\mathbb{N}[X]$ *is compatible with all commutative monoids.*

Now consider the security semiring $\mathbb{S}$. It is idempotent, and therefore not compatible with non-idempotent monoids such as SUM. Still, we want to be able to use $\mathbb{S}$ and other non-idempotent semirings, while allowing the evaluation of aggregation queries with non-idempotent aggregates. This would work if we could construct annotations that would allow us to use Theorem 3.13, in other words, if we could *combine* annotations from $\mathbb{S}$, with multiplicity annotations (i.e. annotations from $\mathbb{N}$). We explain next the construction of such a semiring $\mathbb{SN}$ (for security-bag), and its compatibility with any commutative monoid M will follow from the existence of a homomorphism $h : \mathbb{SN} \to \mathbb{N}$.

Constructing a compatible semiring. We start with the semiring of polynomials $\mathbb{N}[\mathbb{S}]$, i.e. polynomials where instead of indeterminates(variables) we have members of $\mathbb{S}$, and the coefficients are natural numbers. Already $\mathbb{N}[\mathbb{S}]$ is compatible with any commutative monoid M, as there exists a homomorphism $h : \mathbb{N}[\mathbb{S}] \to \mathbb{N}$; but if we work with $\mathbb{N}[\mathbb{S}]$ we lose the ability to use the identities that hold in $\mathbb{S}$ and to thus reduce the size of annotations in query results. We can do better by taking the quotient of $\mathbb{N}[\mathbb{S}]$ by the smallest congruence containing the following identities:

- $\forall s_1, s_2 \in \mathbb{S} \;\; s_1 \geq s_2 \Longrightarrow s_1 \cdot_{\mathbb{N}[\mathbb{S}]} s_2 = s_1$.
- $\forall c \in \mathbb{N}, s \in \mathbb{S} \;\; 0 \cdot_{\mathbb{N}[\mathbb{S}]} s = c \cdot_{\mathbb{N}[\mathbb{S}]} 0_{\mathbb{S}} = 0$.
- $\forall c \in \mathbb{N} \;\; c \cdot_{\mathbb{N}[\mathbb{S}]} 1_{\mathbb{S}} = c$.

We will denote the resulting quotient semiring by $\mathbb{SN}$. It is easy to check that the faithfulness of the embeddings of $\mathbb{N}$ and $\mathbb{S}$ in $\mathbb{N}[\mathbb{S}]$ is preserved by taking the quotient. Most importantly, $\mathbb{SN}$ is still homomorphic to $\mathbb{N}$. Thus,

COROLLARY 3.15. $\mathbb{SN}$ *is compatible with any commutative monoid* M.

EXAMPLE 3.16. *Consider the SUM monoid. Let* R, S *be the following* $\mathbb{SN}$*-relations:*

A	
30	S
10	S

R

A	
30	T

S

Consider the query: $AGG(R \cup \Pi_{S.A}(S \bowtie R))$. *Ignoring the annotations, the expected result (under bag semantics) is 70. Working within the (compatible) semantics defined by*

$\mathbb{SN} \otimes SUM$, *the query result contains an aggregated value of* $(\mathsf{T} \cdot_{\mathbb{SN}} \mathsf{S} +_{\mathbb{SN}} \mathsf{S}) \otimes 30 + \mathsf{S} \otimes 10$. *We can further simplify this to* $\mathsf{T} \otimes 30 + \mathsf{S} \otimes 30 + \mathsf{S} \otimes 10 = \mathsf{T} \otimes 30 + \mathsf{S} \otimes 40$. *This means that e.g. for a user with credentials* T *the query result is* $1_{\mathbb{SN}} \otimes 70$, *and we can use the inverse of* ι *to map it to* $\mathbb{N}$ *and obtain* 70. *Similarly, for a user with credentials* S, *the query result is mapped to* 40. *These are indeed the expected results.*

Note that if we would have used in the above example $\mathbb{S}$ instead of $\mathbb{SN}$ we would have $(\mathsf{T} +_{\mathbb{S}} \mathsf{S}) = \mathsf{S}$ so $(\mathsf{T} +_{\mathbb{S}} \mathsf{S}) \otimes 30$ would be the same as $\mathsf{S} \otimes 30$. For a user with credentials T we could either use this, leading to the result of $1_{\mathbb{S}} \otimes 40$, or use the same computation done in the example, to obtain $1_{\mathbb{S}} \otimes 70$. Indeed, in $\mathbb{S} \otimes SUM$, we have $1_{\mathbb{S}} \otimes 40 = 1_{\mathbb{S}} \otimes 70$. This is the same phenomenon demonstrated in the beginning of this subsection for $\mathbb{B}$, where ι is not injective, preventing us from stripping it away.

Note also that if we would have used $\mathbb{N}[\mathbb{S}]$ instead of $\mathbb{SN}$ then we could not have done the illustrated simplifications.

4. NESTED AGGREGATION QUERIES

So far we have studied only queries where the aggregation operator is the last one performed. In this section we extend the discussion to queries that involve comparisons on aggregate values.

Note. For simplicity, all results and examples are presented for queries in which the comparison operator is equality (=). However the results can easily be extended to arbitrary comparison predicates, that can be decided for elements of M.

4.1 Difficulties

We start by exemplifying where the algebra proposed for restricted aggregation queries, fails here:

EXAMPLE 4.1. *Reconsider the relation (denoted* R'*) which is the result of aggregation query, depicted in Example 3.8. Further consider a query* Q_{select} *that selects from* R' *all tuples for which the aggregated salary equals* 20. *The crux is that deciding the truth value of the selection condition involves interpreting the comparison operator on symbolic representation of values in* R'*; so far, we have no way of interpreting the obtained comparison expression, for instance* $r_1 \otimes 20 + r_2 \otimes 10$ *"equals"* 20, *and thus we cannot decide the existence of tuples in the selection result.*

Note that in this example, the comparison truth value (and consequently the tuples in the query result) depends in a *non-monotonic way* on existence of tuples in the input relation R: if we map r_1 to 1 and r_2 to 0 then the tuple with dept. d_1 appears in the query result, but not if we map both to 1. We can show that the challenge that this non-monotonicity poses is fundamental, and encountered by any algebra on (M, K)-relations. A more intricate construction is thus required for nested aggregation queries.

4.2 An Extended Structure

We start with an example of our treatment of nested aggregation queries, then give the formal construction.

EXAMPLE 4.2. *Reconsider example 4.1, and recall that the challenge in query evaluation lies in comparing elements of* $K \otimes M$ *with elements of* M *(or* $K \otimes M$, *e.g. in case of joins). Our solution is to introduce to the semiring* K *new elements, of the form* $[x = y]$ *where* $x, y \in K \otimes M$ *(if we need to compare with* $m \in M$, *we use* $\iota(m)$ *instead). The result of evaluating the query in example 4.1 (using* $M = SUM$*) will then be captured by:*

Dept	*Sal*	
d_1	$r_1 \otimes 20$	$\delta(r_1 +_K r_2)_K$
	$+_{K \otimes M} r_2 \otimes 10$	$[r_1 \otimes 20 +_{K \otimes M} r_2 \otimes 10 = 1_K \otimes 20]$
d_2	$r_3 \otimes 10$	$\delta(r_3) \cdot_K [r_3 \otimes 10 = 1_K \otimes 20]$

Intuitively, since we do not know which tuples will satisfy the selection criterion, we keep both tuples and multiply the provenance annotation of each of them by a symbolic equality expression. These equality expressions are kept as symbols until we can embed the values in $M = SUM$ *and decide the equality (e.g. if* $K = \mathbb{N}$*), in which case we "replace" it by* 1_K *if it holds or* 0_K *otherwise. For example, given a homomorphism* $h : \mathbb{N}[X] \to \mathbb{N}$, $h(r_1) = h(r_2) = 1$, *then* $h^M(r_1 \otimes 20 +_{K \otimes M} r_2 \otimes 10) = h(r_1) \otimes 20 +_{K \otimes M} h(r_2) \otimes 10 = 1 \otimes 30 \neq 1 \otimes 20$, *thus the equality expression is replaced with (i.e. mapped by the homomorphism to)* 0_K.

We next define the construction formally; the idea underlying the construction is to define a semiring whose elements are polynomials, in which equation elements are additional indeterminates. To achieve that, we introduce for any semiring K and any commutative monoid M, the "domain" equation $\widehat{K} = \mathbb{N}[K \cup \{[c_1 = c_2] \mid c_1, c_2 \in \widehat{K} \otimes M\}]$. The right-hand-side is a monotone, in fact continuous w.r.t. the usual set inclusion operator, hence this equation has a set-theoretic least solution (no need for order-theoretic domain theory). The solution also has an obvious commutative semiring structure. The solution semiring is denoted $\widehat{K} = (X, +_{\widehat{K}}, \cdot_{\widehat{K}}, 0_{\widehat{K}}, 1_{\widehat{K}})$, and we continue by taking the quotient on $\widehat{K}$ defined by the following axioms.

For all $k_1, k_2 \in K, c_1, c_2, c_3, c_4 \in \widehat{K} \otimes M$:

$$\begin{aligned} 0_{\widehat{K}} &\sim 0_K \\ 1_{\widehat{K}} &\sim 1_K \\ k_1 +_{\widehat{K}} k_2 &\sim k_1 +_K k_2 \\ k_1 \cdot_{\widehat{K}} k_2 &\sim k_1 +_K k_2 \\ [c_1 = c_3] &\sim [c_2 = c_4] \,(\text{if } c_1 =_{\widehat{K} \otimes M} c_2,\ c_3 =_{\widehat{K} \otimes M} c_4) \end{aligned}$$

and if K and M are such that ι defined by $\iota(m) = 1_K \otimes m$ is an isomorphism (and let h be its inverse), we further take the quotient defined by: for all $a, b \in K \otimes M$,

$$\begin{aligned} (*)\,[a = b] &\sim 1_K \,(\text{if } h(a) =_M h(b)) \\ [a = b] &\sim 0_K \,(\text{if } h(a) \neq_M h(b)) \end{aligned}$$

We use K^M to denote the semiring obtained by applying the above construction on a semiring K and a commutative monoid M. A key property is that, when we are able to interpret the equalities in M, K^M collapses to K. Formally,

PROPOSITION 4.3. *If* K *and* M *are such that* $K \otimes M$ *and* M *are isomorphic via* ι *then* $K^M = K$.

The proof is by induction on the structure of elements in K^M, showing that at each step we can "solve" an equality sub-expression, and replace it with 0_K or 1_K.

Lifting homomorphisms. To conclude the description of the construction we explain how to lift a semiring homomorphism from $h : K \to K'$ to $h^M : K^M \to K'^M$, for any commutative monoid M and semirings K, K'. h^M is defined recursively on the structure of $a \in K^M$: if $a \in K$ we define $h^M(a) = h(a)$, otherwise $a = [b \otimes m_1 = c \otimes m_2]$ for some $b, c \in K^M$ and $m_1, m_2 \in M$ and we define $h^M(a) = \left[h^M(b) \otimes m_1 = h^M(c) \otimes m_2\right]$. Note that the application of a homomorphism h^M maps equality expressions to equality expressions (in which elements in K' appear instead of elements of K appeared before). If K' and M are such that their corresponding $\iota : M \to K \otimes M$ defined by $\iota(m) = 1_K \otimes M$ is injective, then we may "resolve the equalities", otherwise the (new) equality expression remains.

4.3 The Extended Semantics

The extended semiring construction allows us to design a semantics for general aggregation queries.

In the sequel we assume, to simplify the definition, that the query aggregates and compares only values of $K^M \otimes M$ (a value $m \in M$ is first replaced by $\iota(m) = 1_K \otimes m$). In what follows, let $R(R_1, R_2)$ be (M, K^M)-relations on an attributes set U. Recall that for a tuple t, $t(u)$ (where $u \in U$) is the value of attribute u in t; also for $U' \subseteq U$, recall that we use $t\mid_{U'}$ for the restriction of t to the attributes in U'. Last, we use $(K^M \otimes M)^U$ to denote the set of all tuples on U, with values from $K^M \otimes M$. The semantics follows:

1. *empty relation:* $\forall t \quad \phi(t) = 0_K$.
2. *union:* $(R_1 \cup R_2)(t) =$

$$\begin{cases} \sum_{t' \in supp(R_1)} R_1(t') \cdot_K \prod_{u \in U}[t'(u) = t(u)] & \text{if } t \in supp(R_1) \\ + \sum_{t' \in supp(R_2)} R_2(t') \cdot_K \prod_{u \in U}[t'(u) = t(u)] & \cup supp(R_2) \\ 0_K & \text{Otherwise.} \end{cases}$$

3. *projection:* Let $U' \subseteq U$, and let $\mathrm{T} = \{t\mid_{U'} \mid t \in supp(R)\}$. Then $\Pi_{U'}(t) =$

$$\begin{cases} \sum_{t' \in Supp(R)} R(t') \cdot_K \prod_{u \in U'}[t(u) = t'(u)] & \text{if } t \in \mathrm{T} \\ 0_K & \text{Otherwise.} \end{cases}$$

4. *selection:* If P is an equality predicate involving the equation of some attribute $u \in U$ and a value $m \in M$ then $(\sigma_P(R))(t) = R(t) \cdot_K [t(u) = \iota(m)]$.
5. *value based join:* We assume for simplicity that R_1 and R_2 have disjoint sets of attributes, U_1 and U_2 resp., and that the join is based on comparing a single attribute of each relation. Let $u_1' \in U_1$ and $u_2' \in U_2$ be the attributes to join on. For every $t \in (K^M \otimes M)^{U_1 \cup U_2}$: $(R_1 \bowtie_{R_1.u_1 = R_2.u_2} R_2)(t) =$ $R_1(t\mid_{U_1}) \cdot_K R_2(t\mid_{U_2}) \cdot_K [t(u_1) = t(u_2)]$.
6. *Aggregation:* For relations with a single attribute u $AGG_M(R)(t) =$

$$\begin{cases} 1 & t(u) = \sum_{t' \in supp(R)} R(t') *_{K \otimes M} t'(u) \\ 0 & \text{otherwise} \end{cases}$$

7. *Group By:* Let $U' \subseteq U$ be a subset of attributes that will be grouped and $u \in U \backslash U'$ be the aggregated attribute. Then for every $t \in (K^M \otimes M)^{U' \cup \{u\}}$:

$GB_{U',u}R(t) =$

$$\begin{cases} \delta((\Pi_{U'}R)(t\mid_{U'})) & t(u) = \sum_{t' \in supp(R)} (R(t') \cdot_K \\ & \prod_{u \in U'}[t'(u) = t(u)]) *_{K \otimes M} t'(u) \\ 0 & \text{otherwise} \end{cases}$$

We can show that the algebra satisfies set/bag compatibility, poly-size overhead, and homomorphism coomutation.

Example 4.4. *Reconsider the relation in Example 4.2, and let us perform another sum aggregation on Sal. The value in the result now contains equation expressions:*

$$\begin{aligned} &\delta(r_1 +_K r_2) \cdot_K \left[r_1 \otimes 20 +_{K \otimes M} r_2 \otimes 10 = 1_K \otimes 20\right] \\ &*_{K \otimes M} \left(r_1 \otimes 20 +_{K \otimes M} r_2 \otimes 10\right) \\ &+_{K \otimes M} \delta(r_3) \cdot_K [r_3 \otimes 10 = 1_K \otimes 20] *_{K \otimes M} r_3 \otimes 10 \end{aligned}$$

Given a homomorphism $h : \mathbb{N}[X] \to \mathbb{N}$ we can "solve" the equations, e.g. if $h(r_1) = 1$, $h(r_2) = 0$ and $h(r_3) = 2$, we obtain an aggregated value of $1 \otimes 40$. Note that the aggregation value is not monotone in r_1, r_2, r_3: map r_2 to 1 (and keep r_1,r_3 as before), to obtain $1 \otimes 20$.

5. DIFFERENCE

We next show that via our semantics for aggregation, we can obtain for the first time a semantics for arbitrary queries with difference on K-relations. We describe the obtained semantics and study some of its properties.

5.1 Semantics for Difference

We first note that difference queries may be encoded as queries with aggregation, using the monoid $\widehat{\mathbb{B}} = (\{\bot, \top\}, \vee, \bot)$ (the following encoding was inspired by [29, 8]):

$$\begin{aligned} R - S = \Pi_{a_1 \ldots a_n}\{&\left(GB_{\{a_1,\ldots a_n\},b}(R \times \bot_b \cup S \times \top_b)\right) \\ &\bowtie_{a_1,\ldots a_n} (R \times \bot_b)\}. \end{aligned}$$

$\bot_b$ and $\top_b$ are relations on a single attribute b, containing a single tuple ($\bot$) and ($\top$) respectively, with provenance 1_K. Using the semantics of Section 4, we obtain a semantics for the difference operation.

Interestingly, we next show that the obtained semantics can be captured by a simple and intuitive expression. First, we note that since $\widehat{\mathbb{B}}$ is idempotent, every semiring K positive with respect to $+_K$ is compatible with $\widehat{\mathbb{B}}$ (see Theorem 3.12). The following proposition then holds for every K, K' and every two $(\widehat{\mathbb{B}}, K)$-relations R, S:

Proposition 5.1. *For every tuple t, semirings K, K' such that $K'^{\widehat{\mathbb{B}}} \otimes \widehat{\mathbb{B}}$ is isomorphic to $\widehat{\mathbb{B}}$ via $\iota(m) = 1_K \otimes m$, if $h : K \to K'$ is a semiring homomorphism then:* $h^{\widehat{\mathbb{B}}}([(R - S)(t)]) = h^{\widehat{\mathbb{B}}}([S(t) \otimes \top = 0] \cdot_K R(t))$.

The obtained provenance expression is thus "equivalent" (in the precise sense of Proposition 5.1) to $[S(t) \otimes \top = 0] \cdot R(t)$. The following lemma helps us to understand the meaning of the obtained equality expression:

Lemma 5.2. *For every semiring K positive w.r.t. $+_K$ and $h : K \to \mathbb{B}$, $h^M([S(t) \otimes \top = 0]) = \top$ iff $h(S(t)) = \bot$.*

Intuitive Interpretation. It follows from our definition that a tuple t appears in $R - S$ if it appears in R, but does not appear in S. When the tuple appears in the result of $R - S$, it carries its original annotation from R. I.e. the existence of t in S is used as a boolean condition. This means that in particular, for $\mathbb{B}$, or semantics is equivalent to set semantics. But for $\mathbb{N}$ it is a hybrid of set and bag semantics: tuples that appear in R but not in S, appear in $R-S$ with their original multiplicity that they had in R.

EXAMPLE 5.3. *Let R, S be the following relations, where R contains employees and their departments and S containing departments that are designated to be closed:*

ID	*Dep*	
1	d_1	t_1
2	d_1	t_2
2	d_2	t_3

R

Dep	
d_1	t_4

S

To obtain a relation with all departments that remains active, we can use the query $(\Pi_{Dep}R) - S$, resulting in:

Dep	
d_1	$[t_4 \otimes \top = 0]\cdot(t_1 + t_2)$
d_2	$[0 = 0]\cdot t_3 \quad (= t_3)$

Now consider some homomorphism $h : \mathbb{N}[X] \to \mathbb{N}$ (multiplicity e.g. stands for number of employees in the department). Note that if $h(t_4) > 0$ then the department d_1 is closed and indeed d_1 is omitted from the support of the difference query result, otherwise it retains each original annotation that it had in R. Assume now that we decide to revoke the decision of closing the department d_1. This corresponds to mapping t_4 to 0; we can easily propagate this deletion to the query results; the equality appearing in the annotation of the first tuple is now $[0 = 0] = 1_K$ and we obtain as expected:

Dep	
d_1	$t_1 + t_2$
d_2	t_3

In particular, we obtain a semantics for the entire Relational Algebra, including difference. It is interesting to study the specialization of the obtained semantics for particular semirings: $\mathbb{B}, \mathbb{N}, \mathbb{Z}$, and to compare it to previously studied semantics for difference.

5.2 Comparison with other semantics

For a semiring K and a commutative monoid M we say that two queries Q, Q' are equivalent if for every input (M, K)-database D, the results (including annotations) $Q(D)$ and $Q'(D)$ are congruent (namely the corresponding values and annotations are congruent) according to the axioms of $K^M \otimes M$ and K^M. In the sequel we fix $M = \widehat{B}$ and consider different instances of K, exemplifying different equivalence axioms for queries with difference while comparing them with previously suggested semantics. We use $Q \equiv_K Q'$ to denote the equivalence of Q, Q' with respect to K and $\widehat{B}$.

$\mathbb{B}$-relations. For $K = \mathbb{B}$, our semantics is the same as set-semantics, thus the following proposition holds:

PROPOSITION 5.4. *For $Q, Q' \in RA$ it holds that $Q \equiv_{\mathbb{B}} Q'$ if and only if $Q \equiv Q'$ under set semantics.*

$\mathbb{N}$-relations. For $K = \mathbb{N}$, we compare our semantics to bag equivalence and observe that they are different (for queries with difference, even without aggregation). Intuitively, this is because in our semantics, the righthand side of the difference is treated as a boolean condition, rather than having the effect of decreasing the multiplicity. Formally,

PROPOSITION 5.5. *$Q \equiv_{\mathbb{N}} Q'$ does not imply that $Q \equiv Q'$ under bag semantics, and vice versa.*

PROOF. Observe that $A-(B\cup B) \equiv_{\mathbb{N}} A-B$; but this does not hold for bag semantics. In contrast, under bag semantics $(A \cup B) - B \equiv A$, but not for our semantics. □

$\mathbb{Z}$-relations. Finally, in [20] the authors have presented $\mathbb{Z}$ semantics for difference. Intuitively, their semantics entails that an element appears in $R - S$ with multiplicity that is the substraction of its multiplicity in S from its multiplicity in S; the resulting multiplicity may be negative. Consequently, the semantics also entails equivalence axioms that are different from those that we have here for $\mathbb{Z}$-relations:

PROPOSITION 5.6. *$Q \equiv_{\mathbb{Z}} Q'$ does not imply $Q \equiv Q'$ under $\mathbb{Z}$ semantics [9], and vice versa.*

PROOF. Under $\mathbb{Z}$ semantics it was shown in [20] that $(A-(B-C)) \equiv (A \cup C)-B$. This does not hold for our semantics. In contrast $A-(B\cup B) \equiv_{\mathbb{Z}} A-B$, but this equivalence does not hold under $\mathbb{Z}$ semantics. □

Deciding Query Equivalence. We conclude with a note on the decidability of equivalence of queries using our semantics. It turns out that for semirings such as $\mathbb{B}, \mathbb{N}$ for which we can interpret the results in $\widehat{\mathbb{B}}$ (in the sense of proposition 5.1 above), query equivalence is undecidable.

PROPOSITION 5.7. *Let K be such that $K^{\widehat{\mathbb{B}}} \otimes \widehat{\mathbb{B}}$ is isomorphic to $\widehat{\mathbb{B}}$. Equivalence of Relational Algebra queries on K-relations is undecidable.*

6. RELATED WORK

Provenance information has been extensively studied in the database literature. Different provenance management techniques are introduced in [13, 6, 7, 4, 31, 5], etc., and it was shown in [22, 19] that these approaches can be compared in the semiring framework. The usefulness of using the general framework of provenance semirings rather than e.g. simple logging of query evaluation has been demonstrated in this paper via applications such as deletion propagation. To our knowledge, this work is the first to study aggregate queries in the context of provenance semirings. Provenance information has a variety of applications (see introduction) and we believe that our novel framework for aggregate queries will benefit all of these. Specifically, workflow provenance has been so far managed mainly in a coarse-grained manner (i.e. at a flow level rather than individual data items level). Since aggregate queries play a key role in workflows, we believe that the framework presented here can be an important step towards fine-grained provenance management for workflows.

[9] As defined in [20].

Aggregate queries have been extensively studied in e.g. [11, 12] for bag and set semantics. As explained in [11], such queries are fundamental in many applications: OLAP queries, mobile computing, the analysis of streaming data, etc. We note that Monoids are used to capture general aggregation operators in [12], but our paper seems to be the first to study their interaction with provenance.

Several semantics of *difference* on relations with annotations have been proposed, starting with the c-tables of [26]. The semirings with monus of [17] generalize this as well as bag-semantics. Difference on relations with annotations from $\mathbb{Z}$ are considered in [20] and from $\mathbb{Z}[X]$ in [18].

There are interesting connections between provenance management and query evaluation on uncertain (and probabilistic) databases (e.g. [30, 14, 4, 3]), as observed in [22]. Evaluation of aggregate queries on probabilistic databases has been studied in e.g. [37, 34]. Optimizing aggregate query evaluation on probabilistic databases via provenance management is an intriguing research challenge.

7. CONCLUSION

We have studied in this paper provenance information for queries with *aggregation* in the semiring framework. We have identified three desiderata for the assessment of candidate approaches: compatibility with the usual set/bag semantics, commutation with semiring homomorphisms and polysize overhead with respect to the database size, and have suggested a semantics that satisfies all three desiderata.

We have exemplified in the paper the application of our approach for deletion propagation and security annotations, and have mentioned other areas in which provenance is useful. A practical, compact representation of the obtained provenance expressions is key to their applicability to these areas. Such compact representation of provenance was realized for the positive relational algebra via the notion of provenance graphs, and their applicability was proven in the context of Orchestra [28]. Finding similar compact representation for the provenance expressions obtained for aggregates, and applying the construction in the context of practical systems, is an intriguing future research.

Acknowledgements. Val Tannen is grateful to Christoph Koch for discussions on the relationship between aggregates and difference and to Vittorio Perduca for discussions on the tensor product constructions.

8. REFERENCES

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

[2] F. N. Afrati and A. Vasilakopoulos. Managing lineage and uncertainty under a data exchange setting. In *SUM*, 2010.

[3] L. Antova, C. Koch, and D. Olteanu. $10^{(10^6)}$ worlds and beyond: efficient representation and processing of incomplete information. *VLDB J.*, 18(5), 2009.

[4] O. Benjelloun, A.D. Sarma, A.Y. Halevy, M. Theobald, and J. Widom. Databases with uncertainty and lineage. *VLDB J.*, 17, 2008.

[5] P. Buneman, J. Cheney, and S. Vansummeren. On the expressiveness of implicit provenance in query and update languages. In *ICDT*, 2007.

[6] P. Buneman, J. Cheney, and S. Vansummeren. On the expressiveness of implicit provenance in query and update languages. *ACM Trans. Database Syst.*, 33(4), 2008.

[7] P. Buneman, S. Khanna, and W.C. Tan. Why and where: A characterization of data provenance. In *ICDT*, 2001.

[8] P. Buneman, S. Naqvi, V. Tannen, and L. Wong. Principles of programming with complex objects and collection types. *TCS*, 149(1), 1995.

[9] J. Cheney, L. Chiticariu, and W. C. Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4), 2009.

[10] J. Cheney, S. Chong, N. Foster, M. I. Seltzer, and S. Vansummeren. Provenance: a future history. In *OOPSLA Companion*, 2009.

[11] S. Cohen. Containment of aggregate queries. *SIGMOD Record*, 34(1), 2005.

[12] S. Cohen, W. Nutt, and Y. Sagiv. Rewriting queries with arbitrary aggregation functions using views. *ACM Trans. Database Syst.*, 31(2), 2006.

[13] Y. Cui, J. Widom, and J.L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Transactions on Database Systems*, 25(2), 2000.

[14] N. N. Dalvi, C. Ré, and D. Suciu. Probabilistic databases: diamonds in the dirt. *Commun. ACM*, 52(7), 2009.

[15] J.N. Foster, T.J. Green, and V. Tannen. Annotated XML: queries and provenance. In *PODS*, 2008.

[16] N. Fuhr and T. Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.*, 15(1), 1997.

[17] F. Geerts and A. Poggi. On database query languages for k-relations. *J. Applied Logic*, 8(2), 2010.

[18] T.J. Green. *Collaborative data sharing with mappings and provenance*. PhD thesis, University of Pennsylvania, 2009.

[19] T.J. Green. Containment of conjunctive queries on annotated relations. In *ICDT*, 2009.

[20] T.J. Green, Z. Ives, and V. Tannen. Reconcilable differences. In *ICDT*, 2009.

[21] T.J. Green, G. Karvounarakis, Z. Ives, and V. Tannen. Update exchange with mappings and provenance. In *VLDB*, 2007.

[22] T.J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, 2007.

[23] T.J. Green, G. Karvounarakis, N. E. Taylor, O. Biton, Z. Ives, and V. Tannen. Orchestra: facilitating collaborative data sharing. In *Proc. of SIGMOD*, 2007.

[24] A. Gupta, I.S. Mumick, and V.S. Subrahmanian. Maintaining views incrementally. In *SIGMOD*, 1993.

[25] J. Huang, T. Chen, A. Doan, and J. F. Naughton. On the provenance of non-answers to queries over extracted data. *Proc. VLDB*, 1, 2008.

[26] T. Imielinski and W. Lipski. Incomplete information in relational databases. *J. ACM*, 31(4), 1984.

[27] S. Issam, F. Adrian, and S. Vladimiro. A formal model of provenance in distributed systems. In *First workshop on on Theory and practice of provenance*, 2009.

[28] G. Karvounarakis, Z. G. Ives, and V. Tannen. Querying data provenance. In *SIGMOD Conference*, 2010.

[29] C. Koch. Incremental query evaluation in a ring of databases. In *PODS*, 2010.

[30] C. Koch and D. Olteanu. Conditioning probabilistic databases. *PVLDB*, 1(1), 2008.

[31] N. Kwasnikowska and J. Van den Bussche. Mapping the nrc dataflow model to the open provenance model. In *IPAW*, pages 3–16, 2008.

[32] J. Lechtenbörger, H. Shu, and G. Vossen. Aggregate queries over conditional tables. *J. Intell. Inf. Syst.*, 19(3), 2002.

[33] S.K. Lellahi and V. Tannen. A calculus for collections and aggregates. In *Cat. Theory and Comp. Science*, 1997.

[34] J. Li, B. Saha, and A. Deshpande. A unified approach to ranking in probabilistic databases. *Proc. VLDB*, 2, 2009.

[35] B. Liu, L. Chiticariu, V. Chu, H.V. Jagadish, and F. Reiss. Refining information extraction rules using data provenance. *IEEE Data Eng. Bull.*, 33(3), 2010.

[36] A. Meliou, W. Gatterbauer, K. F. Moore, and D. Suciu. The complexity of causality and responsibility for query answers and non-answers. *PVLDB*, 4(1), 2010.

[37] C. Ré and D. Suciu. Efficient evaluation of having queries on a probabilistic database. In *DBPL*, 2007.

[38] S. Vansummeren and J. Cheney. Recording provenance for sql queries and updates. *IEEE Data Eng. Bull.*, 30(4), 2007.

[39] W. Zhou, M. Sherr, T. Tao, X. Li, B. T. Loo, and Y. Mao. Efficient querying and maintenance of network provenance at internet-scale. In *SIGMOD*, 2010.

[40] E. Zimányi. Query evaluation in probabilistic relational databases. *Theor. Comput. Sci.*, 171(1-2), 1997.