Semiring-Annotated Data: Queries and Provenance*

Grigoris Karvounarakis
LogicBlox, Inc
1349 W Peachtree St NW
Atlanta, GA 30309
grigoris.karvounarakis@logicblox.com

Todd J. Green
LogicBlox, Inc
1349 W Peachtree St NW
Atlanta, GA 30309
todd.green@logicblox.com

ABSTRACT

We present an overview of the literature on *querying semiring-annotated data*, a notion we introduced five years ago in a paper with Val Tannen. First, we show that positive relational algebra calculations for various forms of annotated relations, as well as provenance models for such queries, are particular cases of the same general algorithm involving commutative semirings. For this reason, we present a formal framework for answering queries on data with annotations from commutative semirings, and propose a comprehensive provenance representation based on semirings of *polynomials*. We extend these considerations to XQuery views over annotated, unordered XML data, and show that the semiring framework suffices for a large positive fragment of XQuery applied to such data. Finally, we conclude with a brief overview of the large body of work that builds upon these results, including both extensions to the theoretical foundations and uses in practical applications.

1. INTRODUCTION

Various forms of *annotated data models* have appeared over the years in the database theory literature, ranging from the study of incomplete [30, 20, 21] and probabilistic [13, 33] databases, to query answering under bag (multiset) semantics and to various models to record the provenance of query results in data sharing and warehousing settings [10, 6, 9, 18, 8]. In a 2007 paper with Val Tannen [19], we showed that many of the mechanisms for evaluating queries over such annotated relations can be unified in a general framework based on *K-relations*, which are relations whose tuples are annotated with elements from a commutative semiring K.

The semantics of positive relational algebra ($\mathcal{RA}^+$) queries extends to K-relations via definitions in terms of the abstract "+" and "·" operations of K. Intuitively, "+" corresponds to alternative use of data in producing a query result, while "·" corresponds to joint use. For $K = \mathbb{B}$, the Boolean semiring, this specializes to the usual set semantics, while for $K = \mathbb{N}$, the semiring of natural numbers, it is bag semantics.

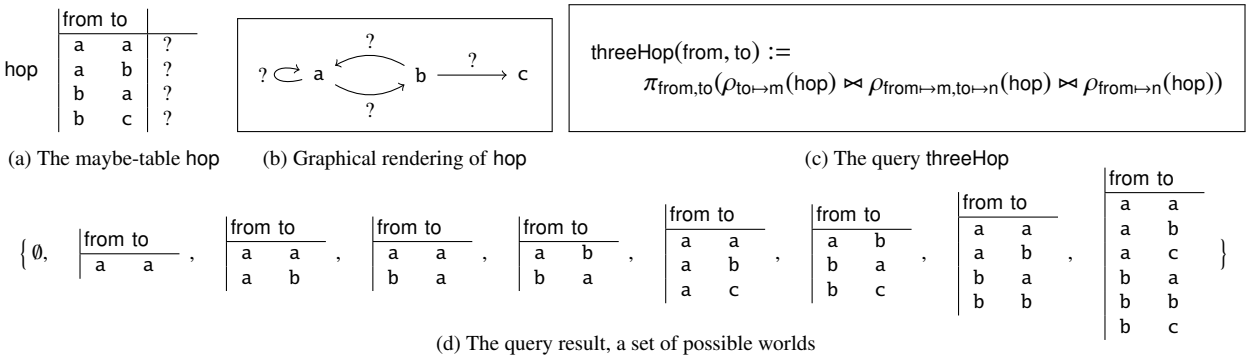
In fact, it turns out that the laws of commutative semirings are *forced* by certain expected identities of $\mathcal{RA}^+$. As additional evidence for the robustness of semiring annotations, we showed with Tannen [19] that the framework extends naturally to recursive Datalog queries (for semirings in which infinite sums are well-defined), and with Nate Foster [12], we gave an extension to a large fragment of positive XQuery over annotated unordered XML data. In each case, we proved a *fundamental theorem* stating that the semantics of queries commutes with the application of semiring homomorphisms. This theorem provides the formal backbone for applications in incomplete and probabilistic databases, trust evaluation, security applications, and others.

A corollary of the fundamental theorem is that, in all these cases, we can use a symbolic *provenance polynomial* representation of semiring calculations to record, document and track the provenance (lineage) of query answers. Using such "most general" semirings as provenance models has the benefit that annotation computations on various semirings factor through them. The resulting provenance expressions can be used at any time to compute result annotations from various semirings as well as for different assignments of values from those semirings to source data (e.g., corresponding to different beliefs of various users about the trustworthiness or quality of source data).

Since the publication of these two papers on semiring-annotated relations and XML, a number of followup studies have investigated semantics and provenance models for additional query operators [14, 3, 4, 17] and data models [31], the interaction of provenance information with query optimization [15, 26], minimization [2] and factorization [29] of provenance expressions, and other topics. Moreover, K-relations have been incorporated

*Portions of this column were adapted from joint work with Val Tannen [19] and Nate Foster and Val Tannen [12].

Database Principles Column. Column editor: Pablo Barceló, Department of Computer Science, University of Chile. E-mail: pbarcelo@dcc.uchile.cl.



(a) The maybe-table hop (b) Graphical rendering of hop (c) The query threeHop

(d) The query result, a set of possible worlds

Figure 1: Maybe-tables and query result. Table hop represents possible links between nodes in a network. The query threeHop computes pairs (x, y) of nodes such that y is reachable from x in exactly three hops. Operator ρ is the rename operator, used in conjunction with the natural join operator $\bowtie$ to express equijoins.

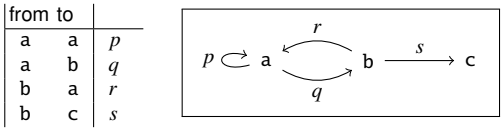
into research systems prototypes to support features including provenance-based trust evaluation [18, 25] and other forms of provenance querying [25], view update [24], maintenance [18, 16] and adaptation strategies [16]. Finally, they have been incorporated into the LogicBlox engine [28] (a commercial declarative programming platform supporting applications in retail planning and analytics), where provenance is used to aid Datalog program analysis and debugging.

The rest of this column is organized as follows. In Section 2, we reprise some original motivations of our work with Tannen [19] by illustrating the similarities of query answering on annotated relations through an example. Next, we define the semantics of $\mathcal{RA}^+$ queries on relations annotated with elements from a semiring K (Section 3), and propose *polynomials*, the *most general* commutative semiring, as a suitable provenance model for $\mathcal{RA}^+$ queries (Section 4). We sketch the extension to annotated XML data [12] in Section 5. Finally (Section 6), we discuss further extensions to incorporate negation, aggregation, and other considerations.

2. ANNOTATED RELATIONS

We motivate our study by considering three important examples of query answering on annotated relations and highlight the similarities between them.

The first example comes from the study of *incomplete databases*, where a simple representation system is the *maybe-table* [30, 20], in which optional tuples are annotated with a '?'. Such a table represents a set of *possible worlds*, capturing the fact that some of the information in the database may be missing or incorrect. Table hop in Figure 1a is a maybe-table representing possible links between nodes in a network (rendered graphically in Figure 1b). The answer to a query over such tables is the set of instances obtained by evaluating the query over each possible world. For example, the result of the query threeHop in Figure 1c is the set of possible worlds shown in Figure 1d. Unfortunately, this set of possible



(a) C-table hop and its graphical rendering

from to	
a a	$(p \wedge p \wedge p) \vee (p \wedge q \wedge r) \vee (p \wedge q \wedge r) \equiv p$
a b	$(p \wedge p \wedge q) \vee (q \wedge q \wedge r) \equiv (p \wedge q) \vee (q \wedge r)$
a c	$p \wedge q \wedge s$
b a	$(p \wedge r \wedge r) \vee (q \wedge r \wedge r) \equiv (p \wedge r) \vee (q \wedge r)$
b b	$p \wedge q \wedge r$
b c	$q \wedge r \wedge s$

(b) C-table query result threeHop

Figure 2: Example of data annotated with Boolean variables, and result of Imielinski-Lipski computation.

worlds cannot itself be represented by a maybe-table. For instance, observe that whenever the tuple (a, c) appears in the result, so does (a, b) and (a, a), and maybe-tables cannot represent such a dependency.

To overcome such limitations, Imielinski and Lipski [21] introduced *c-tables*, where tuples are annotated with Boolean formulas called *conditions*. A maybe-table is a simple kind of c-table, where the annotations are distinct Boolean variables, as shown in Figure 2a. In contrast to weaker representation systems, c-tables are expressive enough to be *closed* under $\mathcal{RA}$ queries, and the main result of Imielinski and Lipski [21] is an algorithm for answering $\mathcal{RA}$ queries on c-tables, producing another c-table as a result. On our example, this algorithm produces the c-table threeHop in Figure 2b; this c-table represents exactly the set of possible worlds shown in Figure 1d. As shown in Figure 2b, some of the Boolean formulas in this c-table can be simplified by applying standard Boolean algebra identities.

Another kind of table with annotations is a *multiset* or *bag*. In this case, the annotations are natural numbers which represent the multiplicity of the tuple in the multiset. (A tuple not listed in the table has multiplicity 0.)

from	to	
a	a	1
a	b	4
b	a	2
b	c	3

(a) Bag table hop

from	to	
a	a	$1\cdot1\cdot1+1\cdot4\cdot2+1\cdot4\cdot2=17$
a	b	$1\cdot1\cdot4+4\cdot4\cdot2=36$
a	c	$1\cdot4\cdot3=12$
b	a	$1\cdot1\cdot2+4\cdot2\cdot2=18$
b	b	$1\cdot4\cdot2=8$
b	c	$4\cdot2\cdot3=24$

(b) Bag query result threeHop

Figure 3: Bag semantics example

from	to	
a	a	x
a	b	y
b	a	z
b	c	u

E	Pr
x	0.6
y	0.5
z	0.1
u	0.4

(a) Probabilistic event table hop

from	to	
a	a	x
a	b	$(x\cap y)\cup(y\cap z)$
a	c	$x\cap y\cap u$
b	a	$(x\cap z)\cup(y\cap z)$
b	b	$x\cap y\cap z$
b	c	$y\cap z\cap u$

(b) Event table threeHop

Figure 4: Probabilistic example

Query answering on such tables involves calculating not just tuples in the output, but also their multiplicities.

For example, consider the multiset shown in Figure 3a. The result of the query threeHop from Figure 1c is the multiset shown in Figure 3b. Note that for projection and union we add multiplicities, while for join we multiply them. There is a striking similarity between the arithmetic calculations we do here for multisets, and the Boolean calculations for the c-table (before the simplifications due to Boolean algebra identities).

A third example comes from the study of *probabilistic databases*, where tuples are associated with values from $[0, 1]$ which represent the probability that the tuple is present in the database. Answering queries over probabilistic tables requires computing the correct probabilities for tuples in the output. To do this, Fuhr and Röllecke [13] and Zimányi [33] (FRZ) introduced *event tables*, where tuples are annotated with probabilistic events, and they gave a query answering algorithm for computing the events associated with tuples in the query output.

Figure 4a shows an example of an event table, with associated *event probabilities* (e.g., z represents the event that the link (b, a) exists). Considering again the same query threeHop as above, the FRZ query answering algorithm produces the event table shown in Figure 4b. Note again the similarity between this table and the example earlier with c-tables. The probabilities of tuples in the output of the query can be computed from this table using the independence of various events that appear together in each expression.

3. POSITIVE RELATIONAL ALGEBRA

In this section we attempt to unify the examples above by considering generalized relations in which the tuples are annotated (*tagged*) with information of various kinds. Then, we will define a generalization of the positive relational algebra ($\mathcal{RA}^+$) to such tagged-tuple relations. The examples in Section 2 will turn out to be particular cases.

We use here the named perspective [1] of the relational model in which tuples are functions $t : U \to \mathbb{D}$ with U a finite set of attributes and $\mathbb{D}$ a domain of values. We fix the domain $\mathbb{D}$ for the time being and we denote the set of all such U-tuples by U-Tup. (Usual) relations over U are subsets of U-Tup.

A notationally convenient way of working with tagged-tuple relations is to model tagging by a function on all possible tuples, with those tuples not considered to be "in" the relation tagged with a special value. For example, the usual set-theoretic relations correspond to functions that map U-Tup to $\mathbb{B} = \{\mathsf{true}, \mathsf{false}\}$ with the tuples in the relation tagged by true and those not in the relation tagged by false.

Definition 3.1. Let K be a set containing a distinguished element 0. A **K-relation** over a finite set of attributes U is a function $R : U\text{-Tup} \to K$ such that its **support** defined by $\mathsf{supp}(R) \stackrel{\text{def}}{=} \{t \mid R(t) \neq 0\}$ is finite.

In generalizing $\mathcal{RA}^+$ we will need to assume more structure on the set of tags. To deal with selection we assume that the set K contains two distinct values $0 \neq 1$ which denote "out of" and "in" the relation, respectively. To deal with union and projection and therefore to combine different tags of the same tuple into one tag we assume that K is equipped with a binary operation "+". To deal with natural join (hence intersection and selection) and therefore to combine the tags of joinable tuples we assume that K is equipped with another binary operation "$\cdot$".

Definition 3.2. Let $(K, +, \cdot, 0, 1)$ be an algebraic structure with two binary operations and two distinguished elements. The main operations of the **positive algebra** are defined as follows:

union If $R_1, R_2 : U\text{-Tup} \to K$ then $R_1 \cup R_2 : U\text{-Tup} \to K$ is defined by: $(R_1 \cup R_2)(t) \stackrel{\text{def}}{=} R_1(t) + R_2(t)$

projection If $R : U\text{-Tup} \to K$ and $V \subseteq U$ then $\pi_V R : V\text{-Tup} \to K$ is defined by:

$$(\pi_V R)(t) \stackrel{\text{def}}{=} \sum_{t=t' \text{ on } V \text{ and } R(t')\neq 0} R(t')$$

natural join If $R_i : U_i\text{-Tup} \to K$, $(i = 1, 2)$, then $R_1 \bowtie R_2$ is the K-relation over $U_1 \cup U_2$ defined by:

$$(R_1 \bowtie R_2)(t) \stackrel{\text{def}}{=} R_1(t_1) \cdot R_2(t_2)$$

where $t_1 = t$ on U_1 and $t_2 = t$ on U_2

In our paper with Tannen [19] we also provided definitions for the remaining operators, such as renaming ρ and selection σ, and showed that the resulting definition gives us an algebra on K-relations and generalizes the definitions of $\mathcal{RA}^+$ for the examples in Section 2.

Indeed, for $(\mathbb{B}, \vee, \wedge, \mathsf{false}, \mathsf{true})$ we obtain the usual $\mathcal{RA}^+$ with set semantics. For $(\mathbb{N}, +, \cdot, 0, 1)$ it is $\mathcal{RA}^+$ with bag semantics. For the Imielinski-Lipski algebra on c-tables we consider the semiring $(\mathsf{PosBool}(B), \vee, \wedge, \mathsf{false}, \mathsf{true})$ of *positive* Boolean expressions over some set B of variables, i.e., $\neg$ is disallowed. (We identify those expressions that yield the same truth-value for all Boolean assignments of the variables in B.) Applying Definition 3.2 to the structure produces exactly the Imielinski-Lipski algebra. Finally, for $(\mathcal{P}(\Omega), \cup, \cap, \emptyset, \Omega)$ we obtain the FRZ $\mathcal{RA}^+$ on event tables.

These four structures are examples of *commutative semirings*, i.e., algebraic structures $(K, +, \cdot, 0, 1)$ such that $(K, +, 0)$ and $(K, \cdot, 1)$ are commutative monoids, $\cdot$ is distributive over $+$, and for all a, we have $0 \cdot a = a \cdot 0 = 0$.

Other examples of commutative semirings include:

- $(\mathbb{N}^\infty, \min, +, \infty, 0)$, the *tropical semiring* [27], where we add ∞ to the natural numbers. In our running example, we might annotate the edges of hop with elements of $\mathbb{N}^\infty$ representing costs; then, a tuple (x, y) in threeHop would be annotated by the cost of the cheapest path of length three from x to y.
- $([0, 1], \max, \min, 0, 1)$ which is related to *fuzzy sets* [32], and could be called the *fuzzy semiring*.
- The semiring of *confidentiality policies* [12] $(C, \min, \max, \mathsf{P}, 0)$, where the total order $C = \mathsf{P} < \mathsf{C} < \mathsf{S} < \mathsf{T} < 0$ describes levels of security clearance: P public, C confidential, S secret, and T top-secret.

Further evidence for requiring K to form such a semiring is given by a result in our paper with Tannen [19] stating that certain standard $\mathcal{RA}^+$ identities (such as commutativity of joins and unions) hold for the positive algebra on K-relations iff $(K, +, \cdot, 0, 1)$ is a commutative semiring. The list of relational identities does not include the idempotence of unions and joins, since these fail for bag semantics, an important special case in our treatment.

Any function $h\colon K \to K'$ can be used to transform K-relations to K'-relations simply by applying h to each tag (note that the support may shrink but never increase). Abusing the notation a bit we denote the resulting transformation from K-relations to K'-relations also by h. We can now state the following *fundamental theorem*, showing that the $\mathcal{RA}^+$ operations work nicely with semiring structures.

Theorem 3.3 (Fundamental Theorem)**.** *Let $h\colon K \to K'$ and assume that K, K' are commutative semirings. The transformation given by h from K-relations to K'-relations commutes with any $\mathcal{RA}^+$ query (i.e., $q(h(R)) = h(q(R))$) iff h is a semiring homomorphism.*

Example 3.4 (Incomplete databases). The c-table hop of Figure 2a represents the set of possible worlds

$$\{h_\nu(\mathsf{hop}) \mid \nu : B \to \mathbb{B}\}$$

where $h_\nu : \mathsf{PosBool}B \to \mathbb{B}$ is the semiring homomorphism which "plugs in" values for the Boolean variables according to ν then simplifies the result to true or false. Theorem 3.3 tells us that the c-table threeHop of Figure 2b correctly represents the possible worlds of Figure 1d. □

Example 3.5 (Non-interference). Recall the semiring C of confidentiality policies. For a user with a given security clearance level c, we want to present a view of the database containing only tuples of clearance level $c' \leq c$. "Erasing" tuples exceeding a given clearance level c corresponds to applying the semiring homomorphism $h_c : C \to C$ which maps $c' \leq c$ to c' and to 0 otherwise. Theorem 3.3 implies that we get the same result by evaluating the query first, then erasing unauthorized tuples from the result, that we get by erasing first, then evaluating the query. □

4. POLYNOMIALS FOR PROVENANCE

Apart from the kinds of annotations we have discussed until now, an important category involves *provenance models* [8], which have been defined as a way of relating the tuples in a query output to the tuples in the input that "contribute" to them. For this reason, in our paper with Tannen [19] we proposed using the different operations of the semiring from Definition 3.2, which appear to fully "document" how an output tuple is produced. To record the documentation as tuple tags we need to use a semiring of symbolic expressions. In the case of semirings, like in ring theory, these are the *polynomials*.

Definition 4.1. Let X be the set of tuple ids of a (usual) database instance I. The *positive algebra provenance semiring* for I is the semiring of polynomials with variables (a.k.a. indeterminates) from X and coefficients from $\mathbb{N}$, with the operations defined as usual[1]:

$$(\mathbb{N}[X], +, \cdot, 0, 1)$$

Example 4.2. Start again from our running example, with tuples tagged with their own id. These relations can be seen as $\mathbb{N}[p, q, r, s]$-relations. Applying the query threeHop from Section 2 and performing the calculations in the provenance semiring we obtain the annotations in the last column of Figure 5. The provenance of (a, a) is $p^3 + 2pqr$ which can be "read" as follows: (a, a) is derived by threeHop in three different ways; one of them uses the input tuple annotated with p *three* times; the other two both involve joining input tuples annotated with p, q and r. □

The following property of polynomials captures the intuition that $\mathbb{N}[X]$ is as "general" as any semiring.

[1]These are polynomials in commutative variables so their operations are the same as in middle-school algebra, except that subtraction is not allowed.

from	to	Lin(X)	Why(X)	Trio(X)	$\mathbb{B}[X]$	$\mathbb{N}[X]$
a	a	pqr	$p + pqr$	$p + 2pqr$	$p^3 + pqr$	$p^3 + 2pqr$
a	b	pqr	$pq + qr$	$pq + qr$	$p^2q + q^2r$	$p^2q + q^2r$
a	c	pqs	pqs	pqs	pqs	pqs
b	a	pqr	$pr + qr$	$pr + qr$	$p^2r + qr^2$	$p^2r + qr^2$
b	b	pqr	pqr	pqr	pqr	pqr
b	c	qrs	qrs	qrs	qrs	qrs

Figure 5: Five kinds of provenance for threeHop

Proposition 4.3. *Let K be a commutative semiring and X a set of variables. For any valuation $v : X \to K$ there exists a unique homomorphism of semirings*

$$\mathsf{Eval}_v : \mathbb{N}[X] \to K$$

such that for one-variable monomials $\mathsf{Eval}_v(x) = v(x)$.

$\mathsf{Eval}_v(P)$ evaluates the polynomial P in K given a valuation for its variables. In calculations with integer coefficients, na where $n \in \mathbb{N}$ and $a \in K$ is the sum in K of n copies of a. Note that $\mathbb{N}$ is embedded in K by mapping n to the sum of n copies of 1_K.

Using the Eval notation, for any $P \in \mathbb{N}[x_1, \ldots, x_n]$ and any K the *polynomial function* $f_P : K^n \to K$ is given by:

$$f_P(a_1, \ldots, a_n) \stackrel{\text{def}}{=} \mathsf{Eval}_v(P) \quad v(x_i) = a_i,\ i = 1..n$$

Putting together Propositions 3.3 and 4.3 we obtain Theorem 4.4 below, a conceptually important fact that says, informally, that the semantics of $\mathcal{RA}^+$ on K-relations for any semiring K *factors* through the semantics of the same in provenance semirings.

Indeed, let K be a commutative semiring, let R be a K-relation, and let X be the set of tuple ids of the tuples in $\mathsf{supp}(R)$. There is an obvious valuation $v : X \to K$ that associates to a tuple id the tag of that tuple in R.

We associate to R an "abstractly tagged" version, denoted $\bar{R}$, which is an $X \cup \{0\}$-relation. $\bar{R}$ is such that $\mathsf{supp}(\bar{R}) = \mathsf{supp}(R)$ and the tuples in $\mathsf{supp}(\bar{R})$ are tagged by their own tuple id. Note that as an $X \cup \{0\}$-relation, $\bar{R}$ is a particular kind of $\mathbb{N}[X]$-relation.

To simplify notation we state the following corollary of Theorem 3.3 for queries of one argument (but the generalization is immediate):

Corollary 4.4. *For any $\mathcal{RA}^+$ query q we have*

$$q(R) \;=\; \mathsf{Eval}_v \circ q(\bar{R})$$

To illustrate an instance of this theorem, consider the provenance polynomial $p^3 + 2pqr$ of the tuple (a, a) in the last column of the table in Figure 5. Evaluating it in $\mathbb{N}$ for $p = 1, q = 4, r = 2, s = 3$ we get 17 which is indeed the multiplicity of (a, a) in Figure 3.

4.1 Provenance hierarchy

Apart from the various forms of annotations described earlier, it turns out that various provenance models can also be captured by semirings, and thus provenance polynomials generalize those models as well. In fact, these

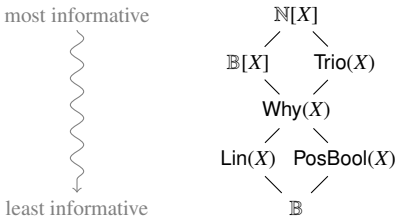


Figure 6: Provenance hierarchy. A path downward from K_1 to K_2 indicates that there exists a surjective semiring homomorphism from K_1 to K_2.

models of provenance can be neatly arranged in the hierarchy of Figure 6 [15].

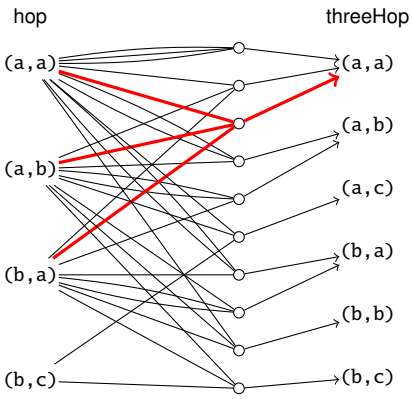
In this figure, $\mathbb{B}[X]$ denotes the *Boolean provenance polynomials semiring* [15] over variables X, Trio(X) denotes a semiring capturing the form of provenance used in the *Trio* system [15, 5], and Why(X) corresponds to *why-provenance* [6]. Formal definitions can be found in the paper by Green [15], but intuitively, annotations in $\mathbb{B}[X]$ (Trio(X), respectively) can be obtained from the provenance polynomials by dropping coefficients (exponents, resp.); while annotations from Why(X) can be obtained by dropping both coefficients and exponents. Annotations from the semiring Lin(X) capturing *lineage* [10] collapse all variables appearing in the polynomial into a single monomial. The corresponding provenance expressions for these models in our running example are shown in Figure 5.

Coupled with Proposition 3.3, this hierarchy gives a clear picture of the relative "*informativeness*" of the various provenance models, since provenance computations for models lower in the hierarchy can always be factored through computations involving models above them in the hierarchy. This additional informativeness of provenance polynomials, compared to all other provenance models in the literature, allows to overcome some of their limitations [19] (also recognized in SPIDER [9] for lineage) to support a variety of applications on annotated data.

4.2 Provenance as a graph

We conclude our discussion of annotated relations by presenting an alternative *graphical* interpretation of provenance polynomials. This viewpoint is especially useful in practical systems needing provenance support such as Orchestra [18] and LogicBlox [28].

We define a *provenance graph* for a query as having two kinds of nodes, *tuple nodes*, one for each source or derived tuple, and *join nodes*. Edges connect tuple nodes to join nodes. Intuitively, each join node corresponds to the instantiation of an n-way join in the query, with incoming edges from tuple nodes participating in the join, and a single outgoing edge to the tuple node output by the join. Output tuple nodes may have multi-



(a)

prThreeHop

from	m	n	to
a	a	a	a
a	a	b	a
a	b	a	a
a	a	a	b
a	b	a	b
a	a	b	c
b	a	a	a
b	a	b	a
b	a	a	b
b	a	b	c

(b)

```
threeHop(X,Y) :- hop(X,M), hop(M,N), hop(N,Y).
```

(c)

```
prThreeHop(X,M,N,Y) :- hop(X,M), hop(M,N), hop(N,Y).
threeHop(X,Y) :- prThreeHop(X,_,_,Y).
```

(d)

Figure 7: Graphical representation of provenance polynomials, along with the graph's relational encoding, and the Datalog rules which construct the graph.

ple incoming edges, representing alternative derivations.

For example, Figure 7a shows the graphical representation of the provenance polynomials in Figure 5. The left column contains tuple nodes for hop, the right column has tuple nodes for threeHop, and the center column has connecting join nodes. The highlighted edges in the graph correspond to (one copy of) the monomial pqr in the annotation $p^3 + 2pqr$ for output tuple (a, a).

This graph representation has served as the basis for a compact storage scheme for provenance. Indeed, with some extensions described in Section 6, this graph model has been used for provenance storage and querying [18, 25] in the ORCHESTRA collaborative data sharing system (CDSS). In ORCHESTRA data is propagated from various sources through paths of *schema mappings* (akin to Datalog rules) and materialized in data warehouses, and provenance is used for view maintenance [18] and update [24], as well as trust evaluation [18, 25]. A similar provenance storage scheme, as well as provenance querying functionality, is supported by the LogicBlox [28] Datalog engine. Figure 8 illustrates the output of a couple of provenance queries issued through the command-line LogicBlox client for a Datalog program encoding our running example.

We illustrate how these systems store provenance graphs in relations on our running example. A query such as threeHop (shown in Figure 7c in Datalog syntax) is translated into a pair of Datalog rules shown in Figure 7d. From the first rule, the relation prThreeHop contains one tuple for every join node in the provenance graph. The resulting provenance relation for the graph of Figure 7a is shown in Figure 7b (where we follow the

```
$ bloxbatch -interactive
<blox>: provenance <doc>
          +provenance[`threeHop]("a","a").
        </doc>
Provenance information:
threeHop(a,a) <- hop(a,a),hop(a,a),hop(a,a).
threeHop(a,a) <- hop(a,a),hop(a,b),hop(b,a).
threeHop(a,a) <- hop(a,b),hop(b,a),hop(a,a).
<blox>: provenance <doc>
          +provenance[`threeHop]("a","b").
        </doc>
Provenance information:
threeHop(a,b) <- hop(a,a),hop(a,a),hop(a,b).
threeHop(a,b) <- hop(a,b),hop(b,a),hop(a,b).
<blox>:
```

Figure 8: Provenance queries in LogicBlox command-line client, for a Datalog program encoding our running example.

order of the join nodes in Figure 7a, and we use horizontal lines to distinguish derivations producing the same result tuple, e.g., the first three tuples correspond to the top three join nodes, that encode derivations for the tuple (a,a)). The highlighted tuple in the table encodes the highlighted edges in the graph.

The storage scheme described above avoids storing redundantly some common provenance subexpressions [25, 23], which are ubiquitous in settings with multiple queries, such as large Datalog programs or complex data sharing settings. Finally, as we explain in Section 6, the graph—as well as its relational encoding—provides a finite representation for the (possibly infinite) provenance of the results of recursive Datalog queries. Even though the graph representation does not store provenance polynomials directly, one can generate them by traversing the graph recursively backwards along its edges (this can be achieved by joining provenance relations) [25].

5. ANNOTATED XML

In this section, we sketch a generalization of K-relations beyond flat relational data by Foster et al. [12], to encompass hierarchically-structured XML data. We present first the annotated XML data model, then illustrate the operation of queries on such data (for a fragment of XQuery) via several examples. Key properties of K-relations (in particular the natural analogue of the fundamental theorem 3.3) continue to hold in the generalized setting, which can be interpreted as evidence for the "robustness" of K-relations.

We fix a commutative semiring K and consider XML data modified so that instead of lists of trees (sequences of elements) there are *sets* of trees. Moreover, each tree belonging to such a set is decorated with an annotation $k \in K$. Since *bags* of elements can be obtained by interpreting the annotations as multiplicities (by picking K to be $(\mathbb{N}, +, \cdot, 0, 1)$), the only difference compared to standard XML is the absence of ordering between sib-

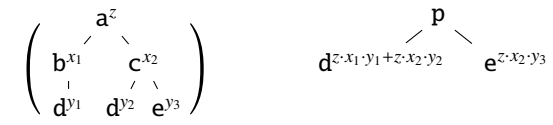


Figure 9: Simple `for` Example.

lings.[2] We call such data *K-annotated unordered XML*, or simply *K-UXML*. Given a domain $\mathcal{L}$ of *labels*, the usual mutually recursive definition of XML data naturally generalizes to *K-UXML*:[3]

- A *value* is either a label in $\mathcal{L}$, a tree, or a K-set of trees;
- A *tree* consists of a label together with a finite (possibly empty) K-set of trees as its "children";
- A finite *K-set of trees* is a function from trees to K such that all but finitely many trees map to 0.

In examples, we illustrate K-UXML data by adding annotations as a superscript notation on the label at the root of the (sub)tree. By convention *omitted annotations* correspond to the "neutral" element $1 \in K$.[4] Note that a tree gets an annotation only as a member of a K-set. To annotate a single tree, we place it in a singleton K-set. When the semiring of annotations is $(\mathbb{B}, \vee, \wedge, \mathsf{false}, \mathsf{true})$ we have essentially unannotated unordered XML; we write UXML instead of $\mathbb{B}$-UXML.

In Figure 9, two K-UXML data values are displayed as trees. The source value can be written in document style as: <a^z> <b^{x_1}> d^{y_1} </> <c^{x_2}> d^{y_2} e^{y_3} </> </>

where we have abbreviated leaves <*l*></> as *l*.

We propose a query language for K-UXML called *K-UXQuery*. Its syntax (see Foster et al. [12] for details) corresponds to a core fragment of XQuery [11] with one exception: the new construct `annot` k p allows queries to modify the annotations on sets. With `annot` k p any K-UXML value can be built with the K-UXQuery constructs. We use the following types for K-UXML and K-UXQuery:

$$t \quad ::= \quad \mathit{label} \mid \mathit{tree} \mid \{\mathit{tree}\}$$

where *label* denotes $\mathcal{L}$, *tree* denotes the set of all trees and {*tree*} denotes the set of all finite K-sets of trees.

In the rest of this section, we illustrate the syntax of K-UXQuery on K-UXML informally on some simple examples to introduce the basic ideas. We start with very simple queries demonstrating how the individual operators work, and build up to a larger example corresponding to a translation of a relational algebra query.

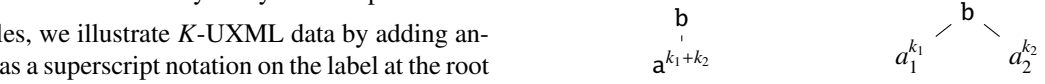
As a first example, define the following queries:

$$p_1 \stackrel{\text{def}}{=} \texttt{element}\ a_1\ \{()\} \quad \text{and} \quad p_2 \stackrel{\text{def}}{=} \texttt{element}\ a_2\ \{()\}$$

Each p_i constructs a tree labeled with a_i and having no children—i.e., a leaf node. The query `(`p_1`)` produces the singleton K-set in which p_1 is annotated with $1 \in K$. The query `annot` k_1 `(`p_1`)` produces the singleton K-set in which p_1 is annotated with $k_1 \cdot 1 = k_1$. We can also construct a union of K-sets:

$$q \stackrel{\text{def}}{=} \texttt{annot}\ k_1\ (p_1)\texttt{,annot}\ k_2\ (p_2)$$

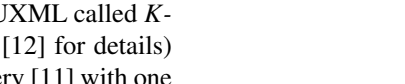
The result computed by q depends on whether a_1 and a_2 are the same label or different labels. If $a_1 = a_2 = \mathsf{a}$, then p_1 and p_2 are the same tree and thus the query `element b {`q`}` produces the left tree below. If $a_1 \neq a_2$, then the same query produces the tree on the right.



Next, let us examine a query that uses iteration:

```
p = element p { for $t in $S return
                  for $x in ($t)/* return
                    ($x)/* }
```

If `$S` is the (source) set on the left side of Figure 9, then the answer produced by p is the tree on the right in the same figure.[5] Operationally, the query works as follows. First, the outer `for`-clause iterates over the set given by `$S`. As `$S` is a singleton in our example, `$t` is bound to the tree whose root is labeled `a` and annotation in `$S` is z. Next, the inner `for`-clause iterates over the set of trees given by `($t)/*`:



It binds `$x` to each of these trees, evaluates the `return`-clause in this extended context, and multiplies the resulting set by the annotation on `$x`. For example, when `$x` is bound to the `b` child, the `return`-clause produces the singleton set `(`d^{y_1}`)`. Multiplying this set by the annotation x_1 yields `(`$d^{x_1 \cdot y_1}$`)`. After combining all the sets returned by iterations of this inner `for`-clause, we obtain the set `(`$d^{x_1 \cdot x_1 + x_2 \cdot y_2}$`, `$e^{x_2 \cdot y_3}$`)`. The final answer for p is obtained by multiplying this set by z. Note that the annotation on each child in the answer is the sum, over all paths that lead to that child in `$t`, of the product of the annotations from the root of `$t` to that child, thus recording *how* it arises from subtrees of `$S`.

Next we illustrate the semantics of XPath `descendant` navigation (shorthand `//`). Consider the query:

[2] For simplicity, we also omit attributes and model atomic values as the labels on trees having no children.

[3] In the XQuery data model, sets of labels are also values; it is straightforward to extend our formal treatment to include this.

[4] Items annotated with 0 are allowed by the definition but are useless because our semantics interprets 0 as "not present/available".

[5] Actually this query is equivalent to the shorter "grandchildren" XPath query `$S/*/*`; we use the version with a `for`-clauses to illustrate the semantics of iteration.

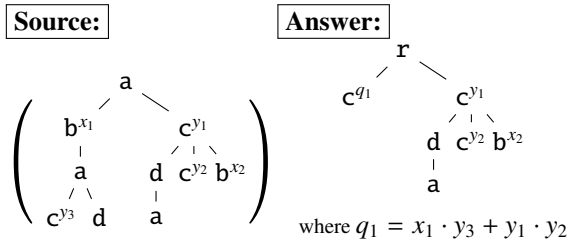


Figure 10: XPath Example.

$$r = \texttt{element r \{ \$T//c \}}$$

which picks out the set of subtrees of elements of `$T` whose label is `c`. A sample source and corresponding answer computed by r are shown in Figure 10. Foster et al. [12] defines the semantics of the `descendant` operator using structural recursion and iteration. It has the property that the annotation for each subtree in the answer is the sum of the products of annotations for each path from the root to an occurrence of that subtree in the source, like the answer shown here.

Now we turn to an example which demonstrates how K-UXQuery behaves on an encoding of a database of relations whose tuples are annotated with elements of K, i.e., the *K-relations* of Section 3. Consider again the relational algebra query from Figure 1c evaluated over the table of Figure 2, viewed as an $\mathbb{N}[X]$-relation.

Figure 11 shows the $\mathbb{N}[X]$-UXML tree that is obtained by encoding the relation hop, the corresponding translation of the view definition into $\mathbb{N}[X]$-UXQuery, and the $\mathbb{N}[X]$-UXML view that is computed using $\mathbb{N}[X]$-UXQuery. Observe that the result is the encoding of the $\mathbb{N}[X]$-relation of Figure 5. This equivalence holds in general: Foster et al. [12] give a systematic translation of positive relational algebra queries into K-UXQuery which agrees with the definitions of Section 3.

In a K-relation, annotations *only* appear on tuples. In our model for annotated UXML data, however, every internal node carries an annotation (recall that, according to our convention, every node in Figure 11 depicted with no annotation carries the "neutral" element $1 \in K$). Therefore, we have more flexibility in how we annotate source values—besides tuples, we can place annotations on the values in individual fields, on attributes on the relations themselves, and even on the whole database! It is interesting to see how, even for a query that is essentially relational, these extra annotations participate in the calculations. See Foster et al. [12] for an example.

6. FURTHER CONSIDERATIONS

In the few years since the publication of our paper with Tannen introducing semiring-annotated relations [19], there has been a flowering of work in the area. Various researchers have investigated different aspects of annotated data and provenance. We conclude this column with a brief overview of these works.

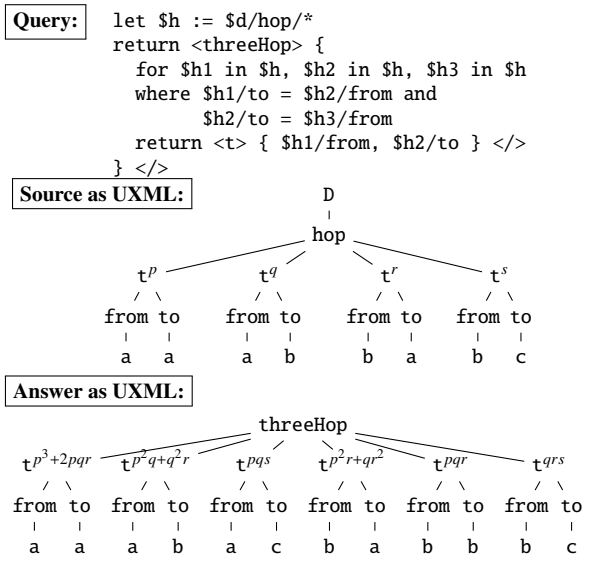


Figure 11: Relational (encoded) example

Recursive queries.

Apart from semantics for $\mathcal{RA}^+$ queries, in our paper with Tannen [19] we also gave semantics for recursive Datalog queries on K-relations. The high-level idea is that the provenance of a tuple in the result of a Datalog query is the sum over all its derivation trees of the product of the tags of the leaves of each tree. In general, however, a tuple can have infinitely many derivation trees. As a result, we focused on ω-continuous semirings, for which infinite sums can be defined. The most general such semiring, that can be used as the provenance model for Datalog queries, is the commutative ω-continuous semiring of **formal power series** with variables from X and coefficients from $\mathbb{N}^\infty$. Interestingly, even if the provenances of some tuples are infinite, they can be represented through a finite system of equations, whose solutions are the provenance expressions for each tuple. Moreover, the provenance graph we showed in the previous section can provide an alternative finite representation for the provenance of Datalog queries.

Query containment.

The introduction of annotations on relations presents new challenges in query reformulation and optimization, as queries that are semantically *equivalent* when posed over ordinary relations may become *inequivalent* when posed over K-relations. Indeed, this phenomenon was previously observed for the case of bag semantics [7, 22], where, e.g., adding a "redundant" self-join to a query actually changes the query's meaning. Green [15] studied containment and equivalence of (unions of) conjunctive queries under five difference kinds of provenance annotations—$\mathbb{N}[X]$, $\mathbb{B}[X]$, Trio(X), Why(X), and Lin(X)—and established decision procedures and complexity characterizations in each case. Interestingly, the provenance hierarchy of Figure 6, which organizes provenance semir-

ings in terms of their information content, also turns out to organize them in terms of "stronger" to "weaker" notions of containment/equivalence. Kostylev et al. [26] nicely generalize the containment results by studying how these containment procedures *axiomatize* various classes of semirings.

Negation.

Semirings do not contain any operation that can capture the semantics of relational *difference*. Note that, e.g., for bag semantics, "regular" subtraction does not work for this purpose, because a tuple cannot have a negative multiplicity. For this reason, Geerts et al. [14] identified a class of semirings that can be extended with a *monus* operator, that captures the semantics of relational difference, and showed that the fundamental theorem holds for all such extended structures (called *m*-semirings). However, they also showed that the structure obtained by extending polynomials with monus is not the most general *m*-semiring (e.g., does not satisfy a factorization theorem similar to Theorem 4.4). Thus, they proposed using as a provenance model for $\mathcal{RA}^+$ with difference ($\mathcal{RA}^+(\setminus)$) the free *m*-semiring, for which there is a standard (though not very practical, from a systems perspective) algebraic construction.

Amsterdamer et al. [3] have recently raised questions about the suitability of *m*-semirings for capturing the provenance of $\mathcal{RA}^+(\setminus)$ queries, by identifying an identity of $\mathcal{RA}^+(\setminus)$ (namely $R \bowtie (S \setminus T) = (R \bowtie S) \setminus (R \bowtie T)$) that fails for some important *m*-semirings (e.g., $\mathsf{PosBool}(X)$ or the semiring of confidentiality policies). However, they have left the study of a provenance model satisfying all identities of $\mathcal{RA}^+(\setminus)$ as an open problem for future work, as they also deemed the standard algebraic construction for the free structure satisfying these identities of $\mathcal{RA}^+(\setminus)$ to not be very practical.

RDF/SPARQL.

Similar challenges, stemming from a form of negation, are posed when trying to specify the semantics of SPARQL queries over RDF data and capture their provenance. Indeed, recent work on SPARQL provenance [31] has showed that provenance polynomials, and other relational provenance models, can be used to capture the semantics of a positive fragment of SPARQL. However, in that paper the authors also argue that queries employing the SPARQL OPTIONAL operator, whose semantics involves a form of difference, cannot be captured by these models. Unfortunately, *m*-semirings are not suitable for this purpose, either, due to subtle differences between the semantics of relational and SPARQL difference. The specification of formal semantics and provenance models for SPARQL queries is the subject of ongoing work.

Z-relations.

The paper by Green et al. [17] introduced $\mathbb{Z}$*-relations*, relations annotated with (positive or negative) integer values, where the difference operator of $\mathcal{RA}$ has a natural interpretation using the inverse operation of the ring, and proposed $\mathbb{Z}$-relations as a useful device for representing data and updates (insertion or deletion of tuples) in a uniform manner. $\mathbb{Z}$-relations turn out to be surprisingly "well-behaved" with respect to query optimization, in that equivalence of $\mathcal{RA}$ queries is decidable under the semantics (in contrast to set or bag semantics, where the problem is undecidable). The paper also proposed a unified perspective on *view maintenance* (propagating the effects of source data updates to materialized views) and *view adaptation* (recomputing materialized views after their definitions are changed), based on optimizing queries using materialized views. Under $\mathbb{Z}$-semantics, a sound and complete algorithm for this problem is possible (again in contrast to set or bag semantics, where any sound algorithm for the problem is necessarily incomplete). Green and Ives [16] present a practical implementation of these ideas.

Aggregates.

Beyond $\mathcal{RA}^+$, Amsterdamer et al. [4] investigated aggregate queries, and found that capturing their provenance requires annotating with provenance information not just tuples, but also individual values within tuples. Then, they provided 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, and commutes with semiring homomorphisms. The resulting provenance expressions involve tensor products between semiring annotations and possible aggregation values.

Minimization and factorization.

Recent work has focused on alleviating practical challenges that arise in data management tools due to the size of provenance information, by proposing methods to reduce this size. In particular, Amsterdamer et al. [2] studied provenance *minimization*, and defined the *core* of provenance information, namely the part of provenance that appears independently of the query plan that is in use.[6] They also provided algorithms that, given a query, compute an equivalent one that realizes the core provenance for all tuples in its result, as well as algorithms to compute the core provenance of a particular tuple in a query result without re-evaluating the query.

Olteanu and Zavodny [29] considered *factorization* of provenance polynomials, which conceptually follows an approach akin to the relational provenance storage

[6] among plans that are equivalent under standard set semantics, not to be confused with containment and equivalence on annotated relations described above.

scheme described in Section 4.2, by representing polynomials as nested expressions. Indeed, such factorized polynomials of query results can be exponentially more succinct than their flat equivalents, by identifying and reusing parts of a polynomial that are common across multiple monomials. Moreover, they make explicit some of the structure in the query result. A downside of factorization is that the monomials of a factorized polynomial are not represented explicitly, but they can be enumerated (similarly to traversing a provenance graph backwards) with polynomial delay.

Recording schema mapping names in provenance.

In data sharing settings, such as addressed in the ORCHESTRA collaborative data sharing system [18], there is a need for provenance to also record the names of *schema mappings* (expressed as database queries) via which data are propagated. For this purpose, Karvounarakis [23] extended semirings with a set of unary functions, one for each mapping, and showed that they satisfy the fundamental theorem, as well as that the corresponding extension of provenance polynomials is the most general such structure. The provenance graph was similarly extended by labeling join nodes with the name of the corresponding mapping.

Querying provenance.

In Section 4.2 we explained how provenance graphs can be traversed to reconstruct provenance polynomials. In more complicated data sharing settings, involving nested and possibly recursive queries or schema mappings [18], it is also important to be able to isolate parts of the provenance of a tuple, e.g., relative to another derived tuple in the intermediate result of some other query, or only focus on parts of derivations involving certain mappings. The provenance query language ProQL [25] employs path expressions to simplify such operations, involving traversal and projection of parts of provenance graphs. Moreover, ProQL takes advantage of the factorization theorem for provenance polynomials, by supporting evaluation of provenance expressions (corresponding to matched graph parts) to compute annotations from various semirings.

Acknowledgments

We are grateful to Val Tannen and Nate Foster, our coauthors from the two papers [19, 12] on which much of this article is based. We also thank Zack Ives, whose ORCHESTRA project was the motivating context for the development of K-relations, and the Penn DB group and our colleagues at LogicBlox for many useful discussions.

7. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
[2] Y. Amsterdamer, D. Deutch, T. Milo, and V. Tannen. On provenance minimization. In *PODS*, 2011.
[3] Y. Amsterdamer, D. Deutch, and V. Tannen. On the limitations of provenance for queries with difference. In *TaPP*, 2011.
[4] Y. Amsterdamer, D. Deutch, and V. Tannen. Provenance for aggregate queries. In *PODS*, pages 153–164, 2011.
[5] O. Benjelloun, A. D. Sarma, A. Y. Halevy, and J. Widom. ULDBs: Databases with uncertainty and lineage. In *VLDB*, 2006.
[6] P. Buneman, S. Khanna, and W.-C. Tan. Why and where: A characterization of data provenance. In *ICDT*, 2001.
[7] S. Chaudhuri and M. Y. Vardi. Optimization of real conjunctive queries. In *PODS*, 1993.
[8] J. Cheney, L. Chiticariu, and W. C. Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4):379–474, 2009.
[9] L. Chiticariu and W.-C. Tan. Debugging schema mappings with routes. In *VLDB*, 2006.
[10] Y. Cui, J. Widom, and J. L. Wiener. Tracing the lineage of view data in a warehousing environment. *TODS*, 25(2), 2000.
[11] D. Draper, P. Fankhauser, M. Fernandez, A. Malhotra, M. Rys, J. Simeon, and P. Wadler. XQuery 1.0 formal semantics. Available from `http://www.w3.org/TR/xquery-semantics/`, 12 November 2003. W3C working draft.
[12] J. N. Foster, T. J. Green, and V. Tannen. Annotated XML: Queries and provenance. In *PODS*, 2008.
[13] N. Fuhr and T. Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *TOIS*, 14(1), 1997.
[14] F. Geerts and A. Poggi. On database query languages for K-relations. *J. Applied Logic*, 8(2), 2010.
[15] T. J. Green. Containment of conjunctive queries on annotated relations. *Theory Comput. Syst.*, 49(2), 2011.
[16] T. J. Green and Z. G. Ives. Recomputing materialized instances after changes to mappings and data. In *ICDE*, 2012.
[17] T. J. Green, Z. G. Ives, and V. Tannen. Reconciliable differences. *Theory of Computing Systems*, 49(2), 2011.
[18] T. J. Green, G. Karvounarakis, Z. G. Ives, and V. Tannen. Update exchange with mappings and provenance. In *VLDB*, 2007. Amended version available as Univ. of Pennsylvania report MS-CIS-07-26.
[19] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, 2007.
[20] T. J. Green and V. Tannen. Models for incomplete and probabilistic information. In *EDBT Workshops*, 2006.
[21] T. Imielinski and W. Lipski. Incomplete information in relational databases. *JACM*, 31(4), 1984.
[22] Y. Ioannidis and R. Ramakrishnan. Containment of conjunctive queries: beyond relations as sets. *TODS*, 20(3), 1995.
[23] G. Karvounarakis. *Provenance for Collaborative Data Sharing*. PhD thesis, University of Pennsylvania, July 2009.
[24] G. Karvounarakis and Z. G. Ives. Bidirectional mappings for data and update exchange. In *WebDB*, 2008.
[25] G. Karvounarakis, Z. G. Ives, and V. Tannen. Querying Data Provenance. In *SIGMOD*, 2010.
[26] E. V. Kostylev, J. L. Reutter, and A. Z. Salamon. Classification of annotation semirings over query containment. In *PODS*, 2012.
[27] W. Kuich. Semirings and formal power series. In *Handbook of formal languages*, volume 1. Springer, 1997.
[28] `http://www.logicblox.com`.
[29] D. Olteanu and J. Zavodny. Factorised representations of query results: Size bounds and readability. In *ICDT*, 2012.
[30] A. D. Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working models for uncertain data. In *ICDE*, 2006.
[31] Y. Theoharis, I. Fundulaki, G. Karvounarakis, and V. Christophides. On provenance of queries on semantic web data. *IEEE Internet Computing*, 15(1):31–39, 2011.
[32] L. A. Zadeh. Fuzzy sets. *Inf. Control*, 8(3), 1965.
[33] E. Zimányi. Query evaluation in probabilistic relational databases. *TCS*, 171(1-2), 1997.