Provenance Semirings

Todd J. Green
tjgreen@cis.upenn.edu

Grigoris Karvounarakis
gkarvoun@cis.upenn.edu

Val Tannen
val@cis.upenn.edu

Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104, USA

ABSTRACT

We show that relational algebra calculations for incomplete databases, probabilistic databases, bag semantics and why-provenance are particular cases of the same general algorithms involving semirings. This further suggests a comprehensive provenance representation that uses semirings of polynomials. We extend these considerations to datalog and semirings of formal power series. We give algorithms for datalog provenance calculation as well as datalog evaluation for incomplete and probabilistic databases. Finally, we show that for some semirings containment of conjunctive queries is the same as for standard set semantics.

Categories and Subject Descriptors

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

General Terms

Theory, Algorithms

Keywords

Data provenance, data lineage, incomplete databases, probabilistic databases, semirings, datalog, formal power series

1. INTRODUCTION

Several forms of *annotated relations* have appeared in various contexts in the literature. Query answering in these settings involves generalizing the relational algebra ($\mathcal{RA}$) to perform corresponding operations on the annotations.

The seminal paper in incomplete databases [19] generalized $\mathcal{RA}$ to c-tables, where relations are annotated with Boolean formulas. In probabilistic databases, [17] and [33] generalized $\mathcal{RA}$ to event tables, also a form of annotated relations. In data warehousing, [12] and [13] compute lineages for tuples in the output of queries, in effect generalizing $\mathcal{RA}$ to computations on relations annotated with sets of contributing tuples. Finally, $\mathcal{RA}$ on bag semantics can be viewed as a generalization to annotated relations, where a tuple's annotation is a number representing its multiplicity.

PODS'07, June 11–14, 2007, Beijing, China.

We observe that in all four cases, the calculations with annotations are strikingly similar. This suggests looking for an algebraic structure on annotations that captures the above as particular cases. We propose using commutative semirings for this purpose. In fact, we can show that the laws of commutative semirings are *forced* by certain expected identities in $\mathcal{RA}$. Having identified commutative semirings as the right algebraic structure, we argue that a symbolic representation of semiring calculations is just what is needed to record, document, and track $\mathcal{RA}$ querying from input to output for applications which require rich *provenance* information. It is a standard philosophy in algebra that such symbolic representations form *the most general* such structure. In the case of commutative semirings, just as for rings, the symbolic representation is that of polynomials. We therefore propose to use polynomials to capture provenance. Next we look to extend our approach to recursive datalog queries. To achieve this we combine semirings with fixed point theory.

The contributions of the paper are as follows:

- We introduce **K-relations**, in which tuples are annotated (tagged) with elements from K. We define a generalized positive algebra on K-relations and argue that K must be a **commutative semiring** (Section 3).
- For **provenance semirings** we propose polynomials with integer coefficients, and we show that positive algebra semantics for any commutative semirings **factors** through the provenance semantics (Section 4).
- We extend these results to **datalog** queries by considering semirings with **fixed points** (Section 5).
- For the (possibly infinite) provenance in datalog query answers we propose semirings of **formal power series** that are shown to be generated by finite **algebraic systems** of fixed point equations (Section 6).
- We give **algorithms** for deciding the finiteness of these formal power series, for computing them when finite, and for computing the coefficient of an arbitrary monomial otherwise (Section 7).
- We show how to specialize our algorithms for computing full datalog answers when K is a **finite distributive lattice**, in particular for **incomplete** and **probabilistic** databases (Section 8).
- We consider query **containment** wrt K-relation semantics and we show that for unions of conjunctive queries and when K is a **distributive lattice**, query

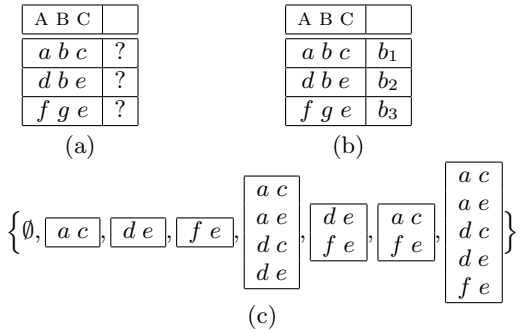

Figure 1: A maybe-table and a query result

containment is the same as that given by standard set semantics. (Section 9).

2. QUERIES ON 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, 18], in which optional tuples are annotated with a '?', as in the example of Figure 1(a). Such a table represents a set of *possible worlds*, and the answer to a query over such a table is the set of instances obtained by evaluating the query over each possible world. Thus, given a query like

$$q(R) \stackrel{\text{def}}{=} \pi_{\text{AC}}\big(\pi_{\text{AB}}R \bowtie \pi_{\text{BC}}R \cup \pi_{\text{AC}}R \bowtie \pi_{\text{BC}}R\big)$$

the query result is the set of possible worlds shown in Figure 1(c). Unfortunately, this set of possible worlds cannot itself be represented by a maybe-table, intuitively because whenever the tuples (a, e) and (d, c) appear, then so do (a, c) and (d, e), and maybe-tables cannot represent such a dependency.

To overcome such limitations, Imielinski and Lipski [19] 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 1(b). In contrast to weaker representation systems, *c*-tables are expressive enough to be *closed* under $\mathcal{RA}$ queries, and the main result of [19] 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 shown in Figure 2(a), which can be simplified to the *c*-table shown in Figure 2(b); this *c*-table represents exactly the set of possible worlds shown in Figure 1(c).

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

For example, consider the multiset shown in Figure 3(a). Then $q(R)$, where q is the same query from before, is the multiset shown in Figure 3(b). 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.

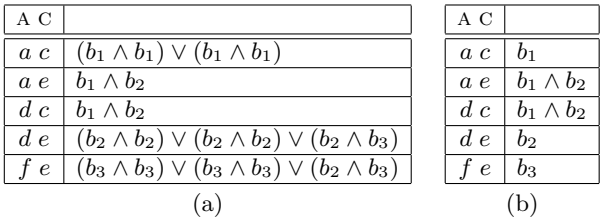

Figure 2: Result of Imielinski-Lipski computation

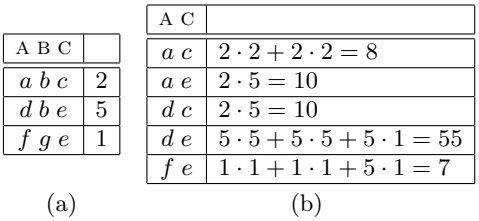

Figure 3: Bag semantics example

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 [17] and Zimányi [33] 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.[1]

Figure 4(a) shows an example of an event table with associated *event probabilities* (e.g., X represents the event that (a, b, c) appears in the instance, and X, Y, Z are assumed independent). Considering again the same query q as above, the Fuhr-Rölleke-Zimányi query answering algorithm produces the event table shown in Figure 4(b). 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 X and Y.

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

[1]The Fuhr-Röllecke-Zimányi algorithm is a general-purpose *intensional* algorithm. Dalvi and Suciu [14] give a sound and complete algorithm which returns a *safe query plan*, if one exists, which may be used to answer the query correctly via a more efficient *extensional* algorithm. Their results do not apply to our example query.

A B C	
a b c	X
d b e	Y
f g e	Z

(a)

E	Pr
X	0.6
Y	0.5
Z	0.1

A C	
a c	X
a e	X ∩ Y
d c	X ∩ Y
d e	Y
f e	Z

(b)

Figure 4: Probabilistic example

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 operations of the* **positive algebra** *are defined as follows:*

empty relation *For any set of attributes U, there is $\emptyset : U\text{-Tup} \to K$ such that $\emptyset(t) = 0$.*

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*

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

(here $t = t'$ on V means t' is a U-tuple whose restriction to V is the same as the V-tuple t; note also that the sum is finite since R has finite support)

selection *If $R : U\text{-Tup} \to K$ and the selection predicate $\mathbf{P}$ maps each U-tuple to either 0 or 1 then $\sigma_{\mathbf{P}} R : U\text{-Tup} \to K$ is defined by*

$$(\sigma_{\mathbf{P}} R)(t) \stackrel{\text{def}}{=} R(t) \cdot \mathbf{P}(t)$$

Which $\{0,1\}$-valued functions are used as selection predicates is left unspecified, except that we assume that false*—the constantly 0 predicate, and* true*—the constantly 1 predicate, are always available.*

natural join *If $R_i : U_i\text{-Tup} \to K \quad i = 1, 2$ 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 (recall that t is a $U_1 \cup U_2$-tuple).

renaming *If $R : U\text{-Tup} \to K$ and $\beta : U \to U'$ is a bijection then $\rho_\beta R$ is a K-relation over U' defined by*

$$(\rho_\beta R)(t) \stackrel{\text{def}}{=} R(t \circ \beta)$$

PROPOSITION 3.3. *The operations of $\mathcal{RA}^+$ preserve the finiteness of supports therefore they map K-relations to K-relations. Hence, Definition 3.2 gives us an algebra on K-relations.*

This definition generalizes the definitions of $\mathcal{RA}^+$ for the motivating examples we saw. 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 set of Boolean expressions over some set B of variables which are *positive*, i.e., they involve only disjunction, conjunction, and constants for true and false. Then we identify those expressions that yield the same truth-value for all boolean assignments of the variables in B.[2] Denoting by $\mathsf{PosBool}(B)$ the result and applying Definition 3.2 to the structure $(\mathsf{PosBool}(B), \vee, \wedge, \mathsf{false}, \mathsf{true})$ produces exactly the Imielinski-Lipski algebra. Finally, for $(\mathcal{P}(\Omega), \cup, \cap, \emptyset, \Omega)$ we obtain the Fuhr-Rölleke-Zimányi $\mathcal{RA}^+$ 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 $\forall a,\ 0 \cdot a = a \cdot 0 = 0$. Further evidence for requiring K to form such a semiring is given by

PROPOSITION 3.4. *The following $\mathcal{RA}$ identities:*

- *union is associative, commutative and has identity $\emptyset$;*
- *join is associative, commutative and distributive over union;*
- *projections and selections commute with each other as well as with unions and joins (when applicable);*
- *$\sigma_{\mathsf{false}}(R) = \emptyset$ and $\sigma_{\mathsf{true}}(R) = R$.*

hold for the positive algebra on K-relations if and only if $(K, +, \cdot, 0, 1)$ is a commutative semiring.

Glaringly absent from the list of relational identities are the idempotence of union and of (self-)join. Indeed, these fail for the bag semantics, an important particular case of our general treatment.

Any function $h : K \to K'$ can be used to transform K-relations to K'-relations simply by applying h to each

[2] in order to permit simplifications; it turns out that this is the same as transforming using the axioms of *distributive lattices* [11]

A	B	C	
a	b	c	p
d	b	e	r
f	g	e	s

(a)

A	C	
a	c	$\{p\}$
a	e	$\{p, r\}$
d	c	$\{p, r\}$
d	e	$\{r, s\}$
f	e	$\{r, s\}$

(b)

A	C	
a	c	$2p^2$
a	e	pr
d	c	pr
d	e	$2r^2 + rs$
f	e	$2s^2 + rs$

(c)

Figure 5: Why-prov. and provenance polynomials

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. The $\mathcal{RA}$ operations we have defined work nicely with semiring structures:

PROPOSITION 3.5. *Let $h : K \to K'$ 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 (for queries of one argument) $q(h(R)) = h(q(R))$ if and only if h is a semiring homomorphism.*

4. POLYNOMIALS FOR PROVENANCE

Lineage/why-provenance was defined in [12, 13, 6] as a way of relating the tuples in a query output to the tuples in the query input that "contribute" to them. The why-provenance of a tuple t in a query output is in fact the *set* of all contributing input tuples.

Computing the why-provenance for queries in $\mathcal{RA}^+$ turns out to be exactly Definition 3.2 for the semiring $(\mathcal{P}(X), \cup, \cup, \emptyset, \emptyset)$ where X consists of the ids of the tuples in the input instance. For example, we consider the same tuples as in relation R used in the examples of Section 2 but now we tag them with their own ids p,r,s, as shown in Figure 5(a). The resulting R can be seen as a $\mathcal{P}(\{p, r, s\})$-relation by replacing p with $\{p\}$, etc. Applying the query q from Section 2 to R we obtain according to Definition 3.2 the $\mathcal{P}(\{p, r, s\})$-relation shown in Figure 5(b).

This example illustrates the limitations of why-provenance (also recognized in [8]). For example, in the query result in Figure 5(b) (f, e) and (d, e) have the same why-provenance, the input tuples with id r and s. However, the query can also calculate (f, e) from s alone and (d, e) from r alone. In a provenance application in which one of r or s is perhaps less trusted or less usable than the other the effect can be different on (f, e) than on (d, e) and this cannot be detected by why-provenance. It seems that we need to know not just *which* input tuples contribute but also *how* they contribute.[3]

On the other hand, by using the different operations of the semiring, Definition 3.2 appears 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*[4]*: $(\mathbb{N}[X], +, \cdot, 0, 1)$.*

Example of provenance computation. Start again from the relation R in Figure 5(a) in which tuples are tagged with their own id. R can be seen as an $\mathbb{N}[p, r, s]$-relation. Applying to R the query q from Section 2 and doing the calculations in the provenance semiring we obtain the $\mathbb{N}[p, r, s]$-relation shown in Figure 5(c). The provenance of (f, e) is $2s^2 + rs$ which can be "read" as follows: (f, e) is computed by q in three different ways; two of them use the input tuple s *twice*; the third uses input tuples r and s. We also see that the provenance of (d, e) is different and we see *how* it is different! □

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

PROPOSITION 4.2. *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 the one-variable monomials we have $\mathsf{Eval}_v(x) = v(x)$.

As the notation suggests, $\mathsf{Eval}_v(P)$ evaluates the polynomial P in K given a valuation for its variables. In calculations with the 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 3.5 and 4.2 we obtain Theorem 4.3 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. For example, in Figure 7(d) we show an abstractly-tagged version of the relation in Figure 7(b). 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 theorem for queries of one argument (but the generalization is immediate):

THEOREM 4.3. *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 $2r^2 + rs$ of the tuple (d, e) in Figure 5(c). Evaluating it in $\mathbb{N}$ for $p = 2, r = 5, s = 1$ we get 55 which is indeed the multiplicity of (d, e) in Figure 3(a).

[3]In contrast to why-provenance, the notion of provenance we propose could justifiably be called **how-provenance**.

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

$$Q(x,y) \text{ :- } R(x,z), R(z,y)$$

(a)

$a\ a$	2
$a\ b$	3
$b\ b$	4

(b)

$a\ a$	$2 \cdot 2 = 4$
$a\ b$	$2 \cdot 3 + 3 \cdot 4 = 18$
$b\ b$	$4 \cdot 4 = 16$

(c)

Figure 6: Datalog with bag semantics

5. DATALOG ON K-RELATIONS

We now seek to give semantics on K-relations to datalog queries. It is more convenient to use the unnamed perspective here. We also consider only "pure" datalog rules in which all subgoals are relational atoms. First observe that for conjunctive queries over K-relations the semantics in Definition 3.2 simplifies to computing tags as sums of products, each product corresponding to a valuation of the query variables that makes the query body hold. For example, consider the conjunctive query and $\mathbb{N}$-relation shown in Figure 6(a) and (b), respectively.

There are two valuations that produce the answer $Q(a,b)$: $\{x=a, y=a, z=b\}$ yields the body $R(a,a), R(a,b)$ while $\{x=a, y=b, z=b\}$ yields the body $R(a,b), R(b,b)$. The sum of products of tags is $2\cdot 3 + 3\cdot 4$ which is exactly what the equivalent $\mathcal{RA}^+$ query yields according to Definition 3.2. If we think of this conjunctive query as a datalog program, the two valuations above correspond to the two *derivation trees* of the tuple $Q(a,b)$.

This suggests the following generalized semantics for datalog on K-relations: the tag of an answer tuple is the sum over all its derivation trees of the product of the tags of the leaves of each tree. Indeed, this generalizes the bag semantics of datalog considered in [26, 27] *when the number of derivation trees is finite.* In general, a tuple can have infinitely many derivation trees (an algorithm for detecting this appears in [28]) hence we need to work with semirings in which infinite sums are defined.

Closed semirings [31] have infinite sums but their "+" is idempotent which rules out the bag and provenance semantics. We will adopt the approach used in formal languages [22] and later show further connections with how semirings and formal power series are used for context-free languages. By assuming that $\mathbb{D}$ is countable, it will suffice to define countable sums.

Let $(K, +, \cdot, 0, 1)$ be a semiring and define $a \leq b \overset{\text{def}}{\Leftrightarrow} \exists x\ a + x = b$. When $\leq$ is a partial order we say that K is **naturally ordered**. $\mathbb{B}$, $\mathbb{N}$, $\mathbb{N}[X]$ and the other semiring examples we gave so far are all naturally ordered.

We say that K is an **ω-complete** semiring if it is naturally ordered and $\leq$ is such that ω-chains $x_0 \leq x_1 \leq \cdots \leq x_n \leq \cdots$ have least upper bounds. In such semirings we can define *countable sums*:

$$\sum_{n\in\mathbb{N}} a_n \overset{\text{def}}{=} \sup_{m\in\mathbb{N}} \Big(\sum_{i=0}^{m} a_i\Big)$$

Note that if $\exists N$ s.t. $\forall n > N, a_n = 0$ then $\sum_{n\in\mathbb{N}} a_n = \sum_{i=0}^{N} a_i$. All the semiring examples we gave so far are ω-complete with the exception of $\mathbb{N}$ and $\mathbb{N}[X]$.

An **ω-continuous** semiring is an ω-complete semiring in which the operations $+$ and $\cdot$ are ω-continuous in each argument. It follows that countable sums are associative and commutative, that $\cdot$ distributes over countable sums and that countable sums are monotone in each addend.

Examples of commutative ω-continuous semirings:

- $(\mathbb{B}, \vee, \wedge, \mathsf{false}, \mathsf{true})$
- $(\mathbb{N}^\infty, +, \cdot, 0, 1)$ where we add ∞ to the natural numbers and define $\infty + n = n + \infty = \infty$ and $\infty \cdot n = n \cdot \infty = \infty$ except for $\infty \cdot 0 = 0 \cdot \infty = 0$. We can think of $\mathbb{N}^\infty$ as the ω-continuous "completion" of $\mathbb{N}$.
- $(\mathsf{PosBool}(B), \vee, \wedge, \mathsf{false}, \mathsf{true})$ with B finite. This commutative semiring is in fact a *distributive lattice* [11] and the natural order is the lattice order. Since we identify those expressions that yield the same truth-value for all boolean assignments for the variables, B finite makes $\mathsf{PosBool}(B)$ finite, hence ω-continuous.
- $(\mathcal{P}(\Omega), \cup, \cap, \emptyset, \Omega)$, used for event tables which is also an example of distributive lattice.
- $(\mathbb{N}^\infty, \min, +, \infty, 0)$, the *tropical semiring* [22]
- $([0,1], \max, \min, 0, 1)$ is related to *fuzzy sets* [32] so we will call it the *fuzzy semiring.*

DEFINITION 5.1. *Let $(K, +, \cdot, 0, 1)$ be a commutative ω-continuous semiring. To keep notation simple let q be a datalog query with one argument (it is easy to generalize to multiple arguments). For any K-relation R define*

$$q(R)(t) = \sum_{\tau \text{ yields } t} \Big(\prod_{t' \in leaves(\tau)} R(t') \Big)$$

where τ ranges over all q-derivation trees for t and t' ranges over all the leaves of τ.

The next result shows that Definition 5.1 does indeed give us a semantics for datalog queries on K-relations.

PROPOSITION 5.2. *For any K-relation R, $q(R)$ has finite support and is therefore a K-relation.*

PROOF. (sketch) Let S be the set of tuples t s.t. $R(t) \neq 0$ and let t' be a tuple s.t. $q(R)(t') \neq 0$. By Definition 5.1, this implies that there is a derivation tree for t (s.t. the tags of the tuples in its leaves are non-zero and correspond to this product) i.e., $t' \in q(S)$. Since $q(S)$ is finite, $q(R)$ has finite support. □

As an example, consider the datalog program q with output predicate Q defined by the rules shown in Figure 7(c), applied on the $\mathbb{N}$-relation R shown in Figure 7(a). Since any $\mathbb{N}$-relation is also a $\mathbb{N}^\infty$-relation and $\mathbb{N}^\infty$ is ω-continuous we can answer this query[5] and we obtain the table shown in Figure 7(b).

A couple of sanity checks follow.

PROPOSITION 5.3. *Let q be an $\mathcal{RA}^+$ query in which the selection predicates only test for attribute equality and let q' be the (non-recursive) datalog query obtained by standard translation from q. Then q and q' produce the same answer when applied to the same instance of a database of K-relations.*

[5]This is transitive closure with bag semantics.

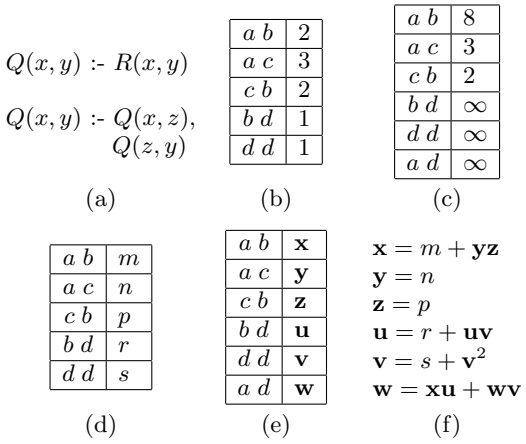

Figure 7: Datalog example

PROPOSITION 5.4. *For any datalog query q and any $\mathbb{B}$-relation R, $\mathsf{supp}(q(R))$ is the same as the result of applying q to the standard relation $\mathsf{supp}(R)$.*

The definition of datalog semantics given above is not so useful computationally. However, we can think of it as the proof-theoretic definition, and as with standard datalog, it turns out that there is an equivalent fixpoint-theoretic definition that is much more workable.

Intuitively, this involves representing the possibly infinite sum of products above as a system of fixpoint equations that reflect all the ways that a tuple can be produced as a result of applying the *immediate consequence operator* T_q (for a datalog query q) on other tuples. Since such an immediate consequence can involve other tuples in idb relations, that may themselves have infinitely many derivations, we introduce a new variable for each tuple in the idb relation and use that variable to refer to that tuple when calculating its immediate consequences. Thus, for every tuple there is an equation between the variable for that tuple and a polynomial over all the variables.

To make this precise we consider **polynomials** with coefficients in an arbitrary commutative semiring K. If the set of variables is X we denote the set of polynomials by $K[X]$. We have already used $\mathbb{N}[X]$ for provenance but $K[X]$ also forms a commutative semiring. We saw in Section 4 that because $\mathbb{N}$ can be embedded in any semiring K the polynomials in $\mathbb{N}[X]$ define polynomial functions over K. Similarly, if $X = \{x_1, \ldots, x_n\}$ then any polynomial $P \in K[X]$ defines a polynomial function $f_P : K^n \to K$. Most importantly, if K is ω-continuous then f_P is ω-continuous in each argument.

DEFINITION 5.5. *Let K be a commutative ω-continuous semiring. An* **algebraic system** *over K with variables $X = \{x_1, \ldots, x_n\}$ consists of a list of polynomials $P_1, \ldots, P_n \in K[X]$ and is written*

$$\begin{array}{rcl} x_1 & = & P_1(x_1, \ldots, x_n) \\ & \cdots & \\ x_n & = & P_n(x_1, \ldots, x_n) \end{array}$$

Together, $f_{P_1}, \ldots, f_{P_n}$ define a function $f_{\mathbf{P}} : K^n \to K^n$. K^n has a component-wise commutative ω-continuous semiring structure such that $f_{\mathbf{P}}$ is ω-continuous. Hence, the least fixed point

$$\mathrm{lfp}(f_{\mathbf{P}}) \;=\; \sup_{m \in \mathbb{N}} f_{\mathbf{P}}^m(0, \ldots, 0)$$

exists, and we call it the **solution** *of the algebraic system above.*

As an example, consider the one-variable equation $\mathbf{x} = a\mathbf{x} + b$ with $a, b \in K$. This is closely related to regular language theory and its solution is $\mathbf{x} = a^*b$ where

$$a^* \stackrel{\text{def}}{=} 1 + a + a^2 + a^3 + \cdots$$

For example, in $\mathbb{N}^\infty$ we have $1^* = \infty$ while in $\mathsf{PosBool}(B)$ we have $e^* = \mathsf{true}$ for any e.

Consider a datalog program q and to simplify notation assume just one edb predicate R and one idb-and-output predicate Q. Given an edb K-relation of finite support R we can effectively construct an algebraic system over K as follows. Denote by Q also the K-relation that is the output of the program and let $\bar{Q}$ be the abstractly-tagged (as in Theorem 4.3) version of Q where X is the set of ids of the tuples in $\mathsf{supp}(Q)$. Since $\bar{Q}$ is a $X \cup \{0\}$-relation and R is a K-relation both can be seen also as $K[X]$-relations. The immediate consequence operator T_q is in fact a union of conjunctive queries, hence Definition 3.2 shows how to calculate effectively $T_q(R, \bar{Q})$ as a $K[X]$-relation of finite support. By equating the tags of $\bar{Q}$ with those of $T_q(R, \bar{Q})$ we obtain the promised algebraic system. We will denote this system as $\bar{Q} = T_q(R, \bar{Q})$ (although it only involves the tags of these relations).

THEOREM 5.6. *With the notation above, for any tuple t, the tag $Q(t)$ given by Definition 5.1, when not 0, equals the component of the solution (Definition 5.5) of the algebraic system $\bar{Q} = T_q(R, \bar{Q})$ corresponding to the id of t.*

To illustrate with an example, consider again the datalog program in Figure 7(a) applied to the same $\mathbb{N}$-relation, R shown in Figure 7(b). In Figure 7(e) we have the abstractly-tagged version of the output relation, $\bar{Q}$ in which the tuples are tagged with their own ids. The corresponding algebraic system is the one obtained from Figure 7(f) by replacing $m = 2, n = 3, p = 2, r = 1, s = 1$. (Note that $T_q(R, \bar{Q}) = R \cup \bar{Q} \bowtie \bar{Q}$.) Calculating its solution we get after two fixed point iterations $\mathbf{x} = 8, \mathbf{y} = 3, \mathbf{z} = 2, \mathbf{u} = 2, \mathbf{v} = 2, \mathbf{w} = 2$. In further iterations $\mathbf{x}, \mathbf{y}, \mathbf{z}$ remain the same while $\mathbf{u}, \mathbf{v}, \mathbf{w}$ grow unboundedly (in Section 7 we show how unbounded growth can be detected). Hence the solution is the one shown in Figure 7(c).

Note that semiring homomorphisms are monotone with respect to the natural order. However, to work well with the datalog semantics more is needed.

PROPOSITION 5.7. *Let K, K' be commutative ω-continuous semirings and let $h : K \to K'$ be an ω-continuous semiring homomorphism. Then, the transformation given by h from K-relations to K'-relations commutes with any datalog query (for queries of one argument $q(h(R)) = h(q(R))$).*

6. FORMAL POWER SERIES FOR PROVENANCE

In Section 4 we showed how to use $\mathbb{N}[X]$-relations to capture an expressive notion of provenance for the tuples in the output of an $\mathcal{RA}^+$ query. However, polynomials will not suffice for the provenance of tuples in the output of datalog queries because the semiring $\mathbb{N}[X]$ does not define infinite sums. As with the transition from $\mathbb{N}$ to $\mathbb{N}^\infty$ we wish to

"complete" $\mathbb{N}[X]$ to a commutative ω-continuous semiring. This problem has been tackled in formal language theory and it led to the study of *formal power series* [22].

Note that when we try to apply naively Definition 5.1 to datalog queries on $\mathbb{N}[X]$-relations we encounter two kinds of infinite summations. First, it is possible that we have to sum infinitely many *distinct* monomials. This leads directly to formal power series. Second, it is possible that we have to sum infinitely many copies of the *same* monomial. This means that we need coefficients from $\mathbb{N}^\infty$, not just $\mathbb{N}$.

Let X be a set of variables. Denote by $X^\oplus$ the set of all possible monomials over X. For example, if $X = \{x, y\}$ then $X^\oplus = \{x^m y^n \mid m, n \geq 0\} = \{\epsilon, x, y, x^2, xy, y^2, x^3, x^2y, \ldots\}$ where ϵ is the monomial in which both x and y have exponent 0.

Let K be a commutative semiring. A **formal power series** with variables from X and coefficients from K is a mapping that associates to each monomial in $X^\oplus$ a coefficient in K. A formal power series S is traditionally written as a possibly infinite sum

$$S \;=\; \sum_{\mu \in X^\oplus} S(\mu)\, \mu$$

and we denote the set of formal power series by $K[[X]]$. As with $K[X]$, there is a commutative semiring structure on $K[[X]]$ given by the usual way of adding and multiplying, for example

$$(S_1 \cdot S_2)(\mu) = \sum_{\mu_1 \mu_2 = \mu} S_1(\mu_1) \cdot S_2(\mu_2)$$

But the real reason we use formal power series is the fact that if K is ω-continuous then $K[[X]]$ *is also* ω-continuous (see [22], for example).

DEFINITION 6.1. *Let X be the set of tuple ids of a database instance I. The* **datalog provenance semiring** *for I is the commutative ω-continuous semiring of formal power series $\mathbb{N}^\infty[[X]]$.*

Let us calculate, using the fixed point semantics, the provenances for the output of the datalog query in Figure 7(a). We now take as input the relation, call it $\bar{R}$, in Figure 7(d) which is the abstractly-tagged (tagged with tuple ids) version of the relation R in Figure 7(b). Note that we have *two sets of variables* here. The tuple ids of $\bar{R}$ form one set of variables and the provenance semiring in which we compute is $\mathbb{N}^\infty[[m, n, p, r, s]]$. At the same time, the ids of the tuples in $\bar{Q}$ in Figure 7(e) are used as variables in the algebraic system, whose right-hand sides belong to

$$\mathbb{N}^\infty[[m, n, p, r, s]][\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{u}, \mathbf{v}, \mathbf{w}]$$

i.e., they are polynomials in the variables $\{\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{u}, \mathbf{v}, \mathbf{w}\}$, with coefficients in the semiring of formal power series $\mathbb{N}^\infty[[m, n, p, r, s]]$. The $\mathbf{v}$ component of the solution can be calculated separately:[6]

$$\mathbf{v} = s + s^2 + 2s^3 + 5s^4 + 14s^5 + \cdots$$

Also, one can see that $\mathbf{x} = m + np$, $\mathbf{u} = r\mathbf{v}^*$, $\mathbf{w} = r(m + np)(\mathbf{v}^*)^2$. For example the coefficient of $rnps^3$ in the provenance $\mathbf{w}$ of $Q(a, d)$ is 5, which means this tuple can be obtained in 5 distinct ways using $R(a, c), R(c, b)$ and $R(b, d)$ once and $R(d, d)$ three times.

[6] [9] shows that the coefficient of s^{n+1} is $\frac{2n!}{n!(n+1)!}$.

Algebra provenance, $\mathbb{N}[X]$, is embedded in datalog provenance, $\mathbb{N}^\infty[X]$, by regarding polynomials as formal power series in which all but finitely many coefficients are 0. Here is the corresponding sanity check:

PROPOSITION 6.2. *Let q be an $\mathcal{RA}^+$ query (of one argument, to simplify notation) in which the selection predicates only test for attribute equality, let q' be the (non-recursive) datalog query obtained by standard translation from q and let R be a $\mathbb{N}[X]$-relation. Modulo the embedding of $\mathbb{N}[X]$ in $\mathbb{N}^\infty[X]$ we have $q'(R) = q(R)$*

Formal power series can be evaluated in commutative ω-continuous semirings:

PROPOSITION 6.3. *Let K be a commutative ω-continuous semiring and X a set of variables. For any valuation $v : X \to K$ there exists a unique ω-continuous homomorphism of semirings*

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

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

Therefore, just like polynomials, formal power series define **series functions** on any commutative ω-continuous semiring. Finally, we have the analog of Theorem 4.3.

THEOREM 6.4. *The semantics of datalog on K-relations for any commutative ω-continuous semiring K* **factors** *through the semantics of the same in provenance semirings (of formal power series).*

Although the coefficients in the provenance series may be ∞, we can characterize exactly when this happens[7]:

THEOREM 6.5. *A datalog query q has provenance series in $\mathbb{N}[[X]]$ for some tuple t if and only if the instantiation of q has no cycle of* unit rules *(rules whose body consists of a single idb) s.t. t is part of the cycle (i.e., appears on the head of one of those unit rules and the body of another) and t is in the result of q.*

7. COMPUTING PROVENANCE SERIES

We show here that several natural questions that one can ask about the computability of formal power series in $\mathbb{N}^\infty[[X]]$ can in fact be decided and all finitely representable information can in fact be computed.

Given a datalog program q and a relational instance I, consider the formal power series provenance of some tuple t in the output $q(I)$, i.e., $q(I)(t)$ where the datalog semantics is taken in $\mathbb{N}^\infty[[X]]$ (X is the set of ids of the tuples in I). We show that it is decidable whether $q(I)(t)$ is in fact a polynomial in $N[X]$. The algorithm *All-Trees*, shown in Figure 8 (inspired by [28]) decides this for all output tuples and computes the polynomial when the answer is affirmative.

For an output tuple t for which the answer given by algorithm *All-Trees* is negative, we can use Theorem 6.5 to decide whether $q(I)(t)$ is in $\mathbb{N}[[X]]$. The remaining question is whether $q(I)(t)$ is in $\mathbb{N}^\infty[X]$, which we can decide by

[7] In this theorem, the *instantiation* of a datalog query is the set of rules obtained by considering all satisfying valuations for the variables in rules of q.

Algorithm *All-Trees*
Input: query q, instance I
Output: the power series $P(t)$ for every tuple $t \in q(I)$
1. Initialize $\mathbf{T} \leftarrow \{t() : t \in I\}$
2. Initialize $T^\infty \leftarrow \emptyset$
3. **repeat**
4. $\quad \mathbf{T}^\nu \leftarrow T_q^\nu(\mathbf{T}, T^\infty)$
5. $\quad$ **for** every tree $\tau \in \mathbf{T}^\nu$
6. $\quad\quad$ **do if** any child of root(τ) is in T^∞ **or** any proper descendant of root(τ) to a node associated with the same tuple
7. $\quad\quad\quad$ **then** $T^\infty \leftarrow T^\infty \cup \{\text{root}(\tau)\}$
8. $\quad\quad\quad$ **else** $\mathbf{T} \leftarrow \mathbf{T} \cup \{\tau\}$
9. **until** nothing added to either $\mathbf{T}$ or T^∞ in last iteration
10. **for** every $t \in q(I)$
11. $\quad$ **do if** $t \in T^\infty$
12. $\quad\quad$ **then** $P(t) \leftarrow \infty$
13. $\quad\quad$ **else** $P(t) \leftarrow \sum_{\substack{\tau \in \mathbf{T}: \\ \text{root}(\tau)=t}} \left(\prod_{l \in \text{fringe}(\tau)} l \right)$
14. **return** P

Figure 8: Algorithm *All-Trees*

Algorithm *Monomial-Coefficient*
Input: query q, monomial μ, tuple t
Output: C, the coefficient of μ in the power series $P(t)$
1. Initialize $\mathbf{T} \leftarrow \{t() : t \in M\}$
2. Initialize $P^\infty \leftarrow \emptyset$
3. **repeat**
4. $\quad \mathbf{T}^i \leftarrow T_q^i(\mathbf{T}, T^\infty)$
5. $\quad$ **for** every tree $\tau \in \mathbf{T}^i$
6. $\quad\quad$ **do if** for any child tree τ' of root(τ), $(\text{root}(\tau'), \text{fringe}(\tau')) \in P^\infty$ **or** there is a chain from root(τ) to a node associated with the same tuple
7. $\quad\quad\quad$ **then** $P^\infty \leftarrow P^\infty \cup \{(\text{root}(\tau), \text{fringe}(\tau))\}$
8. $\quad\quad\quad$ **else** $\mathbf{T} \leftarrow \mathbf{T} \cup \{\tau\}$
9. **until** nothing added to either $\mathbf{T}$ or T^∞ in last iteration
10. **if** $(t, \mu) \in P^\infty$
11. $\quad$ **then** $C \leftarrow \infty$
12. $\quad$ **else for** every $\tau \in \mathbf{T}$ s.t. root$(\tau) = t$ and fringe$(\tau) = \mu$
13. $\quad\quad$ **do** $C \leftarrow C + 1$
14. **return** C

Figure 9: Algorithm *Monomial-Coefficient*

checking if there is any cycle in the instantiation of the query involving at least one non-unit rule, s.t. t is part of that cycle; otherwise, $q(I)(t)$ is in $\mathbb{N}^\infty[[X]]$. In algorithm *All-Trees* shown in Figure 8:

- $\mathbf{T}$ is the set of derivation trees computed thus far; T^∞ is the set of tuples with infinitely many derivations; fringe(τ) is the *bag* of labels of leaves of the tree τ.
- $T_q^\nu(\mathbf{T}, T^\infty) = \{\tau \,|\, \tau \notin \mathbf{T} \wedge \tau \in T_q(\mathbf{T}) \wedge \text{root}(\tau) \notin T^\infty\}$, where $T_q(\mathbf{T})$ is the set of trees produced by applying a rule on tuples in $\{\text{root}(\tau) \mid \tau \in \mathbf{T}\} \cup T^\infty$.

Algorithm *All-Trees* terminates because at every iteration only trees which are not there already are produced and moreover, for every tuple that has infinitely many derivations, as soon as it is identified and inserted in T^∞, no more trees for it are produced. Note also that by Theorem 6.4 this algorithm will also give us, in particular, an algorithm for evaluating datalog queries on bag semantics, just like in [28].

If the answer of algorithm *All-Trees* for an output tuple t is negative, we can also use algorithm *Monomial-Coefficient*, shown in Figure 9, to compute the coefficient of a particular monomial μ in $q(I)(t)$, even when that coefficient is ∞. In this algorithm:

- M is the set of tuples whose ids appear in μ, a monomial represented as a bag of labels that appear in it; P^∞ is a set of pairs (t, μ), representing tuples t for which infinite derivation trees whose leaves are equal to the monomial μ have been found.
- $T_q^i(\mathbf{T}, P^\infty, \mu) = \{\tau \mid \tau \notin \mathbf{T} \wedge \tau \in T_q(\mathbf{T}) \wedge \text{fringe}(\tau) \leq \mu \wedge (\text{root}(\tau), m) \notin P^\infty\}$, where $T_q(\mathbf{T})$ is the set of trees that can be produced by applying a rule on tuples in $\{\text{root}(\tau) \mid \tau \in \mathbf{T}\} \cup \{t \mid (t, m) \in P^\infty\}$ it, where the multiplicity of each is the corresponding exponent in the monomial.

If n is the length of the longest acyclic path of unit rules, then every $n+1$ iterations the algorithm *Monomial-Coefficient* either has produced a tree τ with larger fringe(τ) than the ones that were combined to produce it, or it has identified a pair (t', μ') whose derivation trees involve a cycle of unit rules. The algorithm then is guaranteed to terminate, because for all such tuples t' with infinitely many derivations, no trees for t' with fringe$(\tau) = \mu'$ are used in any subsequent derivations, and moreover, the set of trees τ s.t. τ does not involve nodes marked as infinite and fringe$(\tau) \leq \mu$ is finite.

8. INCOMPLETE / PROBABILISTIC DATABASES

Theorem 6.4 suggests using algorithm *All-Trees* for evaluating datalog queries in various semirings. We already noted in Section 7 that this will work fine for $\mathbb{N}^\infty$. How about $\mathsf{PosBool}(B)$, $\mathcal{P}(\Omega)$, and, as a sanity check, $\mathbb{B}$? When algorithm *All-Trees* returns ∞, the evaluation on these semirings will return a normal value! We show that we can compute this value in the more general case when the semiring K is a *finite distributive lattice*. We do so with some simple modifications to algorithm *All-Trees*:

- Redefine T_q^ν to take only $\mathbf{T}$ as a parameter, and return all τ in $T_q(\mathbf{T})$ such that for all τ' in $\mathbf{T}$, if root$(\tau) =$ root(τ'), then fringe$(\tau) <$ fringe(τ').

Thus a derivation tree for a tuple is considered "new" only when its associated monomial is smaller than any yet seen for that tuple. This modified algorithm always returns a polynomial for each tuple. Evaluating these polynomials in K gives the K-relation output.

The sanity check that for $K = \mathbb{B}$ the output tuples get the tag true is easy to check. For $K = \mathsf{PosBool}(B)$, after also checking that for any valuation $v : B \to \mathbb{B}$ we have $v(q(R)) = q(v(R))$, we get a **datalog on boolean c-tables semantics**. This is new for incomplete databases.

In probabilistic databases we restrict ourselves, as usual, to the case when the domain $\mathbb{D}$ is finite, hence the sample space Ω of all possible instances is finite. $K = \mathcal{P}(\Omega)$ is a

finite distributive lattice so we get an effective semantics for datalog on event tables. After checking that the resulting event tagging a tuple t does in fact say that t is in $q(R)$ for the random instance R, we conclude that our algorithm also generalizes that of [16].

9. QUERY CONTAINMENT

Here we present some results about query containment w.r.t. the general semantics in K-relations.

DEFINITION 9.1. *Let K be a naturally ordered commutative semiring and let q_1, q_2 be two queries defined on K-relations. We define containment with respect to K-relations semantics by*

$$q_1 \sqsubseteq_K q_2 \quad \stackrel{\text{def}}{\Leftrightarrow} \quad \forall R\, \forall t\; q_1(R)(t) \leq q_2(R)(t)$$

When K is $\mathbb{B}$ and $\mathbb{N}$ we get the usual notions of query containment with respect to set and bag semantics.

Some simple facts follow immediately. For example if $h : K \to K'$ is a semiring homomorphism such that $h(x) \leq h(y) \Rightarrow x \leq y$ and q_1, q_2 are $\mathcal{RA}^+$ queries it follows from Prop. 3.5 that $q_1 \sqsubseteq_{K'} q_2 \Rightarrow q_1 \sqsubseteq_K q_2$. If instead h is a surjective homomorphism then $q_1 \sqsubseteq_K q_2 \Rightarrow q_1 \sqsubseteq_{K'} q_2$. Similarly when K, K' and h are also ω-continuous and q_1, q_2 are datalog queries (via Prop. 5.7).

The following result allows us to use the decidability of containment of unions of conjunctive queries [7, 29].

THEOREM 9.2. *If K is a* distributive lattice *then for any q_1, q_2 unions of conjunctive queries we have*

$$q_1 \sqsubseteq_K q_2 \quad \textit{iff} \quad q_1 \sqsubseteq_{\mathbb{B}} q_2$$

PROOF. (sketch) One direction follows because $\mathbb{B}$ can be homomorphically embedded in K. For the other direction we use the existence of query body homomorphisms to establish mappings between monomials of provenance polynomials. Then we apply the factorization theorem (4.3) and the idempotence and absorption laws of K. □

Therefore, if K is a distributive lattice for (unions of) conjunctive queries containment with respect to K-relation semantics is decidable by the same procedure as for standard set semantics. $\mathsf{PosBool}(B)$, $\mathcal{P}(\Omega)$ and the fuzzy semiring are all distributive lattices. A theorem similar to the one above is shown in [20] but the class of algebraic structures used there does not include $\mathsf{PosBool}(B)$ or $\mathcal{P}(\Omega)$ (although it does include the fuzzy semiring).

10. RELATED WORK

Lineage/why-provenance was introduced in [12, 13, 6], (the last paper uses a tree data model) but the relationship with [19] was not noticed. The papers on probabilistic databases [17, 33, 23] note the similarities with [19] but do not attempt a generalization.

Datalog with bag semantics in which derivation trees are counted was considered in several papers, among them [25, 26, 27]. The evaluation algorithms presented in these papers do not terminate if some output tuple has infinite multiplicity. Datalog on incomplete and on probabilistic databases is considered in [15, 24], again with non-terminating algorithms. Later [28] gave an algorithm for detecting infinite multiplicities in datalog with bag semantics and [16] gave a terminating algorithm for datalog on probabilistic databases.

Two recent papers on provenance, although independent of our work, have a closer relationship to our approach. Like us, [8] identifies the limitations of why-provenance and proposes *route-provenance* which is also related to derivation trees. The issue of infinite routes in recursive programs is avoided by considering only *minimal* ones. [3] proposes a notion of lineage of tuples for a type of incomplete databases but does not consider recursive queries. It turns out that we can also describe the lineage in [3] by means of a special commutative semiring so our approach is more general. More significantly, we have provided evidence for the statement "the algorithm in [19] could already compute lineage" rather than just showing that incompleteness and provenance can co-exist.

The first attempt at a general theory of relations with annotations appears to be [20] where axiomatized *label systems* are introduced in order to study containment.

Our paper borrows the machinery of semirings and formal power series from the theory of formal languages (see [22] and references in there). For example, (non-commutative) algebraic systems of equations can be associated to context-free grammars and the integer coefficients in the formal power series solutions count the "degree of ambiguity" of a string in the language [9] (their restriction to grammars without unit rules inspired our Theorem 6.5).

Context-free grammars have been used in the study of datalog but mainly *chain* datalog programs were considered (e.g., [2]) in order to capture the inherent order in strings. Closed semirings are used in [31, 10] but only in order to use Kleene's regular expression algorithm to optimize special classes of datalog programs.

We learned with interest that some of what we do is similar in spirit with a line of work is AI on *constraint satisfaction problems* (CSP) [4, 5]. Their constraints over semirings are in fact the same as our K-relations and the two operations on constraints correspond indeed to relational join and projection. CSP *solutions* are expressed as projection-join queries in [4] and as Prolog programs in [5]. Computing solutions is the same as the evaluation of join and projection in Section 3 and [5] also uses fixed points on semirings. There are some important differences though. The semirings used in [4, 5] are such that $+$ is idempotent and 1 is a top element in the resulting order. This rules out our semirings $\mathbb{N}, \mathbb{N}^\infty, \mathbb{N}[X], \mathbb{N}^\infty[[X]]$ hence the bag and provenance semantics. [8] More importantly, much of the focus in CSP is in choosing optimal solutions rather than how these solutions depend on the constraints.

11. CONCLUSION AND FURTHER WORK

Beyond the technical results, this paper can be regarded also as arguing that various forms of K-relations, even multisets, provide coarser forms of provenance while the polynomial and formal power series annotations are, by virtue of their "universality" (as illustrated by the factorization theorems) the most general form of annotation possible with the boundaries of semiring structures. This might be a perspective worth using when, in the future, we search for provenance structures for data models other than relational.

[8] Another difference is that for datalog semantics we require our semirings to be ω-continuous while [5] uses the less well-behaved fixed points given by Tarski's theorem for monotone operators on complete lattices. However, the semiring examples [5] appear to be in fact ω-continuous.

Although we only considered boolean c-tables, the results can be extended to arbitrary c-tables. Further, we would like to extend Definition 3.2 to include negation, which seems to require axiomatizing an additional operation akin to "proper subtraction" in $\mathbb{N}$. Algorithm *All-Trees* does not immediately give an effective way to evaluate datalog over the tropical semiring but we conjecture that such a procedure exists. We also conjecture that containment under bag semantics of conjunctive or unions of conjunctive queries implies (hence is equivalent to) containment under $\mathbb{N}[X]$-relation semantics. This would allow us to transfer to $\mathbb{N}[X]$ the undecidability result in [20].

Although in general database research and AI CSP research sadly tend to ignore each other (witness the original submission of this paper!), some deep connections have been exhibited in [21], involving, in particular, conjunctive query containment. Looking at the relationship between our work and that of [4, 5] from this perspective might provide interesting results.

Finally, we plan to investigate the application of our approach to database applications where a fine and detailed notion of provenance is needed, for example to propagate information back and forth between databases related by logical constraints of various kinds.

Acknowledgments We are grateful to Zack Ives for many useful discussions. We thank Nilesh Dalvi for pointing out [20], Jan Chomicki for letting us know about semirings being used in constraint satisfaction and an anonymous referee for pointing out [5]. The authors were partially supported by NSF grants IIS 0415810 and IIS 0513778.

12. REFERENCES

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

[2] F. N. Afrati and C. H. Papadimitriou. The parallel complexity of simple logic programs. *J. ACM*, 40(4):891–916, 1993.

[3] O. Benjelloun, A. D. Sarma, A. Y. Halevy, and J. Widom. ULDBs: Databases with uncertainty and lineage. In *VLDB*, 2006.

[4] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *JACM*, 44(2):201–236, 1997.

[5] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint logic programming: syntax and semantics. *ACM TOPLAS*, 23(1):1–29, 2001.

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

[7] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, 1977.

[8] L. Chiticariu and W.-C. Tan. Debugging schema mappings with routes. In *VLDB*, 2006.

[9] N. Chomsky and M.-P. Schützenberger. The algebraic theory of context-free languages. *Computer Programming and Formal Systems*, pages 118–161, 1963.

[10] M. P. Consens and A. O. Mendelzon. Low complexity aggregation in graphlog and datalog. In *ICDT*, 1990.

[11] P. Crawley and R. P. Dilworth. *Algebraic Theory of Lattices*. Prentice Hall, 1973.

[12] Y. Cui. *Lineage Tracing in Data Warehouses*. PhD thesis, Stanford University, 2001.

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

[14] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, 2004.

[15] F. Dong and L. V. S. Lakshmanan. Deductive databases with incomplete information. In *Symposium on Logic Programming*, 1992.

[16] N. Fuhr. Probabilistic datalog — a logic for powerful retrieval methods. In *SIGIR*, 1995.

[17] N. Fuhr and T. Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *TOIS*, 14(1), 1997.

[18] T. J. Green and V. Tannen. Models for incomplete and probabilistic information. In *EDBT Workshops*, 2006.

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

[20] Y. E. Ioannidis and R. Ramakrishnan. Containment of conjunctive queries: beyond relations as sets. *TODS*, 20(3), 1995.

[21] P. Kolaitis and M. Vardi. Conjunctive-query containment and constraint satisfaction. In *PODS*, 1998.

[22] W. Kuich. Semirings and formal power series. In *Handbook of formal languages*, volume 1. Springer, 1997.

[23] L. V. S. Lakshmanan, N. Leone, R. Ross, and V. S. Subrahmanian. Probview: a flexible probabilistic database system. *TODS*, 22(3), 1997.

[24] L. V. S. Lakshmanan and F. Sadri. Probabilistic deductive databases. In *Symposium on Logic Programming*, 1994.

[25] M. Maher and R. Ramakrishnan. Déjà vu in fixpoints of logic programs. In *NACLP*, 1989.

[26] I. S. Mumick. *Query Optimization in Deductive and Relational Databases*. PhD thesis, Stanford University, 1991.

[27] I. S. Mumick, H. Pirahesh, and R. Ramakrishnan. The magic of duplicates and aggregates. In *VLDB Journal*, 1990.

[28] I. S. Mumick and O. Shmueli. Finiteness properties of database queries. In *Fourth Australian Database Conference*, 1993.

[29] Y. Sagiv and M. Yannakakis. Equivalences among relational expressions with the union and difference operators. *J. ACM*, 27(4), 1980.

[30] A. D. Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working models for uncertain data. In *ICDE*, 2006.

[31] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume II. Computer Science Press, 1989.

[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.