Universal provenance for regular path queries

Gösta Grahne
grahne@cs.concordia.ca
Concordia University
Montreal, Quebec, Canada

Tianyi Liu
tinjiklauglobal@gmail.com
Concordia University
Montreal, Quebec, Canada

Nematollaah Shiri
shiri@cs.concordia.ca
Concordia University
Montreal, Quebec, Canada

ABSTRACT

We study the question of *universal provenance* for regular path queries in graph databases. Universal provenance was introduced by Green, Karvounarakis, and Tannen in 2007 as the most general form of semiring provenance for positive relational algebra query results. According to this methodology, the universal provenance is to be computed during query evaluation, and any *application provenance*, such as e.g. the multiplicity of a tuple in the answer, can then be obtained by a homomorphic projection from the universal provenance. The methodology has subsequently been applied to SPARQL-queries and queries expressed in first order and fixed point logics, but for regular path queries only application provenance has been considered. We remedy the situation in this paper, by elucidating the classical theoretical framework and results that enable us to show that certain *regular expressions* form the universal provenance for regular path queries. In addition, we show experimentally that computing the universal provenance, as opposed to computing the application provenance directly, does not incur any significant overhead.

CCS CONCEPTS

• **Theory of computation → Regular languages**; • **Information systems → Data management systems**.

KEYWORDS

regular path queries, graph databases, semiring provenance, universal provenance

ACM Reference Format:
Gösta Grahne, Tianyi Liu, and Nematollaah Shiri. 2022. Universal provenance for regular path queries. In *14th International Theory and Practice of Provenance (TaPP'22), June 17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3530800.3534531

1 PROVENANCE OF RELATIONAL ALGEBRA QUERY RESULTS

Suppose we have a database D, a query Q, and a tuple $t \in Q(D)$, the answer to Q on D. We might be interested in questions such as *"how was the tuple t derived to be in $Q(D)$?"* and *"what tuples from D contributed to the derivation of the tuple t?"* Answers to these questions form the *provenance* of the tuple $t \in Q(D)$. Various forms and models of provenance were proposed and studied by database researchers in the beginning of the millennium. The area was put on a solid theoretical footing in 2007, when Green, Karvounarakis, and Tannen [13] showed that for all queries expressed in the positive relational algebra, all proposed forms of provenance could be uniformly captured within the algebraic framework of *semirings*. Furthermore, [13] showed that there is a most general form of semiring provenance, called *provenance polynomials*, from which all other considered forms of provenance can be obtained through projections by homomorphisms.

For an illustration, let R be a relation with attributes A, B, C as shown below left, and let Q be the relational algebra query $\pi_{AC}\big(\pi_{AC}(R) \bowtie \pi_{BC}(R) \cup \pi_{AB}(R) \bowtie \pi_{BC}(R)\big)$. For definitions of the relational model and algebra, see [1]. The the result $Q(R)$ is shown below right.

A	B	C	
a	b	c	$\mathtt{t}_1$
d	e	c	$\mathtt{t}_2$

A	C	
a	c	$\mathtt{t}_1 \cdot \mathtt{t}_1 + \mathtt{t}_1 \cdot \mathtt{t}_1 + \mathtt{t}_1 \cdot \mathtt{t}_2$
d	c	$\mathtt{t}_2 \cdot \mathtt{t}_2 + \mathtt{t}_2 \cdot \mathtt{t}_1 + \mathtt{t}_2 \cdot \mathtt{t}_2$

Each tuple in R is labelled by a unique *tuple identifier* ($\mathtt{t}_1$ and $\mathtt{t}_2$), and the expressions on the tuples in the output relation $Q(R)$ are *provenance polynomials* representing how the tuple was derived using Q and the input tuples. In general, these expressions are *formal polynomials*, or *formal terms* using the tuple identfiers and operators $\cdot$ and $+$, as well as coefficients from $\mathbb{N} = \{0, 1, 2, \ldots\}$. Thus for instance the provenance $\mathtt{t}_1 \cdot \mathtt{t}_1 + \mathtt{t}_1 \cdot \mathtt{t}_1 + \mathtt{t}_1 \cdot \mathtt{t}_2$ of (a, c) can be read as "the tuple (a, c) can be obtained in three ways: twice using $\mathtt{t}_1$ followed by a second use of $\mathtt{t}_1$, and once by using $\mathtt{t}_1$ followed by the use of $\mathtt{t}_2$." This can now be formalized by saying the the provenance polynomials should come from a structure $\mathsf{K}_{\mathbb{N}[X]} = (\mathbb{N}[X], +, \cdot, 0, 1)$, the *commutative semiring of polynomials* with variables from X and coefficients from $\mathbb{N}$. The requirement that $\mathsf{K}_{\mathbb{N}[X]}$ should be a commutative semiring comes from the axioms for adding and multiplying formal polynomials, and from the requirement of being able to embed the positive relational algebra into $\mathsf{K}_{\mathbb{N}[X]}$. [1]

On the other hand, if our database D is a multiset of tuples, we could annotate each tuple with the number of times it occurs in D. If, for instance $\mathtt{t}_1$ occurs 3 times in R and $\mathtt{t}_2$ occurs 5 times, we can infer that tuple (a, c) occurs 43 times in $Q(R)$. In this scenario we have been working in the *multiplicity semiring*, which

TaPP'22, June 17, 2022, Philadelphia, PA, USA

ACM ISBN 978-1-4503-9349-2/22/06…$15.00
https://doi.org/10.1145/3530800.3534531

[1] In general, a *semiring* K is a structure $(K, +, \cdot, 0, 1)$, where $(K, +, 0)$ is a commutative monoid with identity element 0, $(K, \cdot, 1)$ is a monoid with identity element 1, $(x+y)\cdot z = (x\cdot z)+(y\cdot z)$, $x\cdot(y+z) = (x\cdot y)+(x\cdot z)$, and $0\cdot x = x\cdot 0 = 0$, for all $x, y, z \in K$. A semiring is *commutative*, if $x \cdot y = y \cdot x$, for all $x, y \in K$. We also need the notion of a semiring homomorphism: Let $\mathsf{K} = (K, +, \cdot, 0, 1)$ and $\mathsf{K}' = (K', +', \cdot', 0', 1')$ be semirings. A mapping $h : K \to K'$ is said to be a *semiring homomorphism* if $h(0) = 0'$, $h(1) = 1'$, $h(x + y) = h(x) +' h(y)$, and $h(x \cdot y) = h(x) \cdot' h(y)$.

is the commutative semiring $\mathsf{K}_{\mathsf{bag}} = (\mathbb{N}^\infty, +, \cdot, 0, 1)$, and the provenance/multiplicity of tuple (a, c) can be obtained from the $\mathbb{N}[X]$ provenance $\mathtt{t}_1 \cdot \mathtt{t}_1 + \mathtt{t}_1 \cdot \mathtt{t}_1 + \mathtt{t}_1 \cdot \mathtt{t}_2$ and homomorphism h, where $h(\mathtt{t}_1) = 3$, $h(\mathtt{t}_2) = 5$, and $h(2 \cdot (\mathtt{t}_1 \cdot \mathtt{t}_1) + \mathtt{t}_1 \cdot \mathtt{t}_2) = 2 \cdot (3 \cdot 3) + 5 \cdot 5 = 43$.

The observation of the relationship between $\mathsf{K}_{\mathbb{N}[X]}$ and $\mathsf{K}_{\mathsf{bag}}$ is a special case of the following theorem, stating that $\mathsf{K}_{\mathbb{N}[X]}$ is *free* for the class of commutative semirings.

Theorem 1.1. *Let $\mathsf{K} = (K, +, \cdot, 0, 1)$ be a commutative semiring and X a (finite) set. For any mapping $h : X \to K$ there is a unique extension to a semiring homomorphism $h : \mathsf{K}_{\mathbb{N}[X]} \to \mathsf{K}$.*

Significantly, Green, Karvounarakis, and Tannen [13] show that the positive relational algebra can be extended to construct the provenance polynomials recursively based on the structure of the query Q. Let $\mathsf{K} = (K, +, \cdot, 0, 1)$. be a commutative semiring. Green, Karvounarakis, and Tannen [13] introduce (in different notation) $\mathsf{prov}_t^{\mathsf{K},h}(Q(D))$, the K-provenance of tuple $t \in Q(D)$, where h maps each tuple in D to an element in K. The provenance is defined[2] recursively as $\mathsf{prov}_t^{\mathsf{K},h}(R(D)) = h(t)$, and

$$\begin{aligned}
\mathsf{prov}_t^{\mathsf{K},h}(\pi(R)(Q(D))) &= \textstyle\sum_{\{u \in Q(R) : \pi(u) = t\}} \mathsf{prov}_u^{\mathsf{K},h}(Q(D)) \\
\mathsf{prov}_t^{\mathsf{K},h}(Q(D) \cup Q'(D)) &= \mathsf{prov}_t^{\mathsf{K},h}(Q(DB)) + \mathsf{prov}_t^{\mathsf{K},h}(Q'(DB)) \\
\mathsf{prov}_{t \cdot u}^{\mathsf{K},h}(Q(DB) \bowtie Q'(DB)) &= \mathsf{prov}_t^{\mathsf{K},h}(Q(DB)) \cdot \mathsf{prov}_u^{\mathsf{K},h}(Q'(DB)).
\end{aligned}$$

Green, Karvounarakis, and Tannen [13] show that the freeness of $\mathsf{K}_{\mathbb{N}[X]}$ carries over to the provenance of query results:

Theorem 1.2. *Let $\mathsf{K} = (K, +, \cdot, 0, 1)$ be a commutative semiring, D a relational database, ι mapping each tuple of D to a unique identfier in a set X. For every mapping h that maps tuples of D to semiring elements in K, there is a mapping $g : X \to K$, such that $g \circ \iota = h$, and*

$$g\Big(\mathsf{prov}_t^{\mathsf{K}_{\mathbb{N}[X]},\iota}\big(Q(D)\big)\Big) = \mathsf{prov}_t^{\mathsf{K},h}\big(Q(D)\big) \tag{1}$$

for all positive relational algebra queries Q.

The two previous theorems justify [13] naming provenances in $\mathbb{N}[X]$ as *universal provenance*. Note that the operators $\cdot$ and $+$ are used in $\mathsf{K}_{\mathbb{N}[X]}$ to build terms, whereas in $\mathsf{K}_{\mathsf{bag}}$ they stand for the familiar arithmetic operations actually carried out on the elements of $\mathbb{N}$. Semirings like $\mathsf{K}_{\mathbb{N}[X]}$ will be called *syntactic* semiring, while $\mathsf{K}_{\mathsf{bag}}$ and the like will be called *semantic* semirings. The latter are called *application* semirings in [13]. For examples of syntactic and semantic provenance semirings, see next sections and [11, 13]. Green, Karvounarakis, and Tannen [13] argue that the provenance polynomials in $\mathbb{N}[X]$ should be computed at query evaluation time. Any application provenance can then be obtained from the polynomials by the corresponding homomorphism. The purpose of the present paper is to apply this approach to regular path queries in graph databases.

2 RELATED WORK

Green et al. [13] also considered provenance for datalog queries. We briefly return to this topic in Section 4. Their seminal paper [13] has spawned a multitude of results building upon it. We refer to the current survey [11]. Most closely related to our work is the work of Ramusat et al. [22, 23]. They study semiring provenance of regular path queries in graph databases, but only consider evaluation under various application semirings, and are not concerned with universal provenance. In [10], Geerts and Poggi define universal provenance for semirings reflecting relational algebra that includes the difference operator, and Geerts et al. [9] do the same for SPARQL queries. Tannen et al. have further broadened the scope by considering universal provenance semirings for first order logic [12, 26], for fixed point logics [4], and for linear algebra operators [28].

As for the present paper, one could argue that everything presented here is contained in the Kleene-Schützenberger Theorem (see e.g. [8] or [7]). However, the purpose of the current paper is to place the provenance of regular path queries solidly in the context of universal data provenance, outlined above.

3 PROVENANCE FOR REGULAR PATH QUERIES

Graph databases. A *graph database* $G = (V, E, \Sigma)$ is an edge-labelled directed graph, where $V = \{1, \ldots, n\}$ is a finite set of vertices, Σ is a finite set of (predicate) labels, and $E \subseteq V \times \Sigma \times V$ is a set of (directed) labelled edges. The vertices are to be thought of as objects, and the edges as relations between the objects. In other words, an edge (i, a, j), where $i, j \in V$ and $a \in \Sigma$, can be viewed as a tuple (i, j) in a binary relation a.

A *path* from vertex i to vertex j in a graph database G is a sequence $\pi = (i, a_1, i_1)(i_1, a_2, i_2) \ldots (i_{k-1}, a_k, j)$ of adjacent edges of G. We define $\mathsf{first}(\pi) = i$ and $\mathsf{last}(\pi) = j$. The empty path is denoted ϵ. The *word* represented by the path π is defined as $\mathsf{word}(\pi) = a_1 a_2 \ldots a_k$. The set of all paths from i to j in G is denoted $\Pi_{i,j}(G)$, and the set of all paths in G is denoted $\Pi(G)$.

A *regular path query* on a graph database G is a regular expression Q over Σ, defining a regular language $L(Q) \subseteq \Sigma^*$. Let G_Q be a graph such that

$$\Pi(G_Q) = \{\pi \in \Pi(G) : \mathsf{word}(\pi) \in L(Q)\}.$$

Note that G_Q can be computed with standard automata theoretic techniques. The *answer to Q on G* is

$$Q(G) = \big\{\big(\mathsf{first}(\pi), \mathsf{last}(\pi)\big) : \pi \in \Pi(G_Q)\big\}.$$

Let $(i, j) \in Q(G)$. Then $(i, j) = (\mathsf{first}(\pi), \mathsf{last}(\pi))$, for some path $\pi \in G_Q$. This tells us that π should belong to the provenance of (i, j). If $\pi = (i, a_1, i_1)(i_1, a_2, i_2) \ldots (i_{k-1}, a_k, j)$ with edge identifiers $\mathtt{t}_1, \mathtt{t}_2, \ldots, \mathtt{t}_k$, then the term $\mathtt{t}_1 \cdot \mathtt{t}_2 \cdot \ldots \cdot \mathtt{t}_k$ should be a term in the provenance of (i, j). If we think of an edge (i_{m-1}, a_m, i_m) as a tuple (i_{m-1}, i_m) in relation a_m, our provenance agrees with the relational provenance of [13]. The class of semirings relevant to graph databases and regular path queries is called *complete star-semirings*, and is defined next.

Complete star-semirings. A *semiring* $\mathsf{K} = (K, +, \cdot, 0, 1)$ is *complete* if $\Sigma_{i \in I} x_i \in K$, $x \cdot (\Sigma_{i \in I} x_i) = \Sigma_{i \in I}(x \cdot x_i)$, and $(\Sigma_{i \in I} x_i) \cdot x = \Sigma_{i \in I}(x_i \cdot x)$, for all $x, x_i \in K$ and infinite I. A *complete star-semiring (cs-semiring)* K is a structure $(K, +, \cdot, *, 0, 1)$, where $(K, +, \cdot, 0, 1)$ is a complete semiring with idempotent $+$, such that $x^* = \Sigma_{j \geq 0} x^j$, where $x^0 = 1$, $x^{j+1} = x \cdot x^j = x^j \cdot x$ for all $j \geq 0$. Let $\mathsf{K} = (K, +, \cdot, *, 0, 1)$ and $\mathsf{K}' = (K', +', \cdot', *', 0', 1')$ be cs-semirings. A mapping $h : K \to K'$ is said to be a *cs-semiring homomorphism* if h

[2] In [13] the reader can find the definitions of the provenance for the selection and renaming operators.

is a semiring homomorphism and $h(x^*) = h(x)^{*'}$. The following are some examples of cs-semirings (for more details, see e.g. [11]). Note that all of them are semantic, or application semirings. We'll encounter a syntactic cs-semiring later.

- The Boolean cs-semiring $\mathsf{K}_{\mathsf{bool}} = (\{0,1\}, \vee, \wedge, *, 0, 1)$ with $0^* = 1^* = 1$.
- The Tropical cs-semiring $\mathsf{K}_{\mathsf{trop}} = (\mathbb{R}_+^\infty, \min, +, *, \infty, 0)$ with $x^* = 0$ for all $x \in \mathbb{R}_+^\infty$.
- The Bag cs-semiring $\mathsf{K}_{\mathsf{bag}} = (\mathbb{N}^\infty, +, \cdot, *, 0, 1)$ with $0^* = 1$ and $x^* = \infty$ for $x \neq 0$.
- The Access Control cs-semiring $\mathsf{K}_{\mathsf{acc}} = (\mathbb{A}, \min, \max, U, P)$, where $\mathbb{A} = \{P, C, S, T, U\}$ totally ordered as $P < C < S < T < U$, and where $x^* = P$, for all $x \in \mathbb{A}$.
- The Influence cs-semiring $\mathsf{K}_{\mathsf{infl}} = ([0,1], \max, \times, *, 0, 1)$, where $x^* = 1$, for all $x \in [0,1]$.

The free cs-semiring. We now look for a (syntactic) cs-semiring that is free for the class. Not surprisingly the familiar regular expressions fit the bill perfectly. Let X be a finite set, and $\mathsf{rex}[X]$ the set of all regular expressions over alphabet X. Then the freely generated semiring

$$\mathsf{K}_{\mathsf{rex}[X]} = (\mathsf{rex}[X], +, \cdot, *, \emptyset, \epsilon)$$

is a universal cs-semiring, as witnessed by the next theorem. [3] The proof is a standard exercise in Universal Algebra (see e.g. [3]).

Theorem 3.1. *Let $\mathsf{K} = (K, +, \cdot, *, 0, 1)$ be a complete star-semiring and X a (finite) set. For any mapping $h : X \to K$ there is a unique extension to a semiring homomorphism $h : \mathsf{K}_{\mathsf{rex}[X]} \to \mathsf{K}$.*

For regular expressions $\mathsf{e} \in \mathsf{rex}[X]$, the language of e, that is $L(\mathsf{e}) \subseteq X^*$, is defined in the usual way. Note that L is a cs-semiring homomorphism[4] $L : \mathsf{K}_{\mathsf{rex}[X]} \to \mathsf{K}_{\mathsf{reg}[X]}$, where

$$\mathsf{K}_{\mathsf{reg}[X]} = (\mathsf{reg}[X], \cup, \cdot, *, \emptyset, \{\epsilon\})$$

is the cs-semiring of regular languages over alphabet X. Here $\mathsf{reg}[X]$ is the set of all regular langauges generated from X.

The cs-semiring of paths in a graph database. We consider the following operations on sets of paths. For singleton sets define $\{\pi\} \circ \{\pi'\} = \{\pi''\}$ where π'' is the concatenation of π and π' provided $\mathsf{last}_\pi = \mathsf{first}_{\pi'}$, and $\{\pi\} \circ \{\pi'\} = \emptyset$ otherwise. Also, $\{\epsilon\} \circ \{\pi\} = \{\pi\} \circ \{\epsilon\} = \{\pi\}$. For sets of paths P and P', their composition is $P \circ P' = \bigcup_{\pi \in P, \pi' \in P'} \{\pi\} \circ \{\pi'\}$. Let $P^* = \bigcup_{j \geq 0} P^j$, where $P^0 = \{\epsilon\}$, $P^{j+1} = P \circ P^j$, for $j \geq 0$. We note that it is easily verified that

$$\mathsf{K}_{\Pi(G)} = (2^{\Pi(G)}, \cup, \circ, *, \emptyset, \{\epsilon\})$$

is a cs-semiring. Recall that $\Pi(G)$ denotes the set of all paths in a graph database G with vertex set $\{1, \ldots, n\}$. Furthermore, let $\Pi^k_{i,j}(G)$ be the set of paths from i to j with intermediate vertices in $\{1, \ldots, k\}$, where $k \leq n$. Then clearly

$$\Pi^k_{i,j}(G) = \Pi^{k-1}_{i,j}(G) \ \cup \ \Pi^{k-1}_{i,k}(G) \circ \Pi^{k-1}_{k,k}(G)^* \circ \Pi^{k-1}_{k,j}(G), \qquad (2)$$

and $\Pi_{i,j}(G) = \Pi^n_{i,j}(G)$.

[3] To be more precise, $\mathsf{K}_{\mathsf{rex}[X]}$ is an absolutely free term algebra. It becomes a cs-semiring by forming the quotient algebra $\mathsf{K}_{\mathsf{rex}[X]}/_{\equiv}$, where $\mathsf{e}_1 \equiv \mathsf{e}_2$ if $\mathsf{e}_1 \in \mathsf{rex}[X]$ can be transformed into $\mathsf{e}_2 \in \mathsf{rex}[X]$ using the axioms a Kleene-algebra, or equivalently if $L(\mathsf{e}_1) = L(\mathsf{e}_2)$, see [16].

[4] The mapping $h : \mathsf{K}_{\mathsf{rex}[X]}/_{\equiv} \to \mathsf{K}_{\mathsf{reg}[X]}$ is actually an isomorphism.

Kleene's algorithm for regular expressions over edges. Let $G = (V, E, \Sigma)$ be a graph database with $V = \{1, \ldots, n\}$, let X be a finite set with $|X| = |E|$, and $\iota : E \to X$ a bijection. We want to compute expressions $\mathsf{e}_{i,j}(G) \in \mathsf{rex}[X]$, such that $\iota^{-1}(L(\mathsf{e}_{i,j})) = \Pi_{i,j}(G)$, where $\iota^{-1} \circ L : \mathsf{K}_{\mathsf{rex}[X]} \to \mathsf{K}_{\Pi(G)}$. Note that in particular $\iota^{-1}(\mathsf{t}) = \{(i, a, j)\}$, where $\iota(i, a, j) = \mathsf{t}$, and $\iota^{-1}(u \cdot v) = \iota^{-1}(u) \circ \iota^{-1}(v)$, for $u, v \in X^*$.

Regular expressions $\mathsf{e}_{i,j}(G)$ —simply denoted $\mathsf{e}_{i,j}$ when G is understood from the context— are computed recursively by *Kleene's algorithm* [20] as $\mathsf{e}^0_{i,j}, \mathsf{e}^1_{i,j}, \ldots, \mathsf{e}^n_{i,j} = \mathsf{e}_{i,j}$ as follows:

Let $(i, a_1, j), (i, a_2, j), \ldots, (i, a_k, j)$ be all the edges from i to j in V. Then $\mathsf{e}^0_{i,j} = \sum_{i=1}^k \iota(i, a_i, j)$ and $\mathsf{e}^0_{i,i} = \epsilon + \sum_{i=1}^k \iota(i, a_i, i)$ Recursively we define

$$\mathsf{e}^k_{i,j} = \mathsf{e}^{k-1}_{i,j} \ + \ \mathsf{e}^{k-1}_{i,k} \cdot (\mathsf{e}^{k-1}_{k,k})^* \cdot \mathsf{e}^{k-1}_{k,j}. \qquad (3)$$

The next proposition is proved by induction, using equations (2) and (3).

Theorem 3.2. *Let $\mathsf{e}_{i,j}(G) \in \mathsf{rex}[X]$ be computed by Kleene's algorithm, and $P = \iota^{-1} \circ L : \mathsf{rex}[X] \to \Pi(G)$. Then $P(\mathsf{e}_{i,j}(G)) = \Pi_{i,j}(G)$.*

Proof: We show by induction on k that $P(\mathsf{e}^k_{i,j}(G)) = \Pi^k_{i,j}(G)$. For the base case, suppose $\mathsf{e}^0_{i,j}(G) = \iota(i, a_1, j) + \iota(i, a_2, j) + \cdots + \iota(i, a_k, j)$. Then $P(\mathsf{e}^0_{i,j}(G)) = \{(i, a_1, j), (i, a_2, j), \ldots, (i, a_k, j)\} = \Pi^0_{i,j}(G)$. For the inductive step,

$$\begin{aligned}
P\big(\mathsf{e}^k_{i,j}(G)\big) &= \\
P\big(\mathsf{e}^{k-1}_{i,j}(G) \ + \ \mathsf{e}^{k-1}_{i,k}(G) \cdot (\mathsf{e}^{k-1}_{k,k}(G))^* \cdot \mathsf{e}^{k-1}_{k,j}(G)\big) &= \\
P\big(\mathsf{e}^{k-1}_{i,j}(G)\big) \ \cup \ P\big(\mathsf{e}^{k-1}_{i,k}(G)\big) \circ P\big(\mathsf{e}^{k-1}_{k,k}(G)\big)^* \circ P\big(\mathsf{e}^{k-1}_{k,j}(G)\big) &= \\
\Pi^{k-1}_{i,j}(G) \ \cup \ \Pi^{k-1}_{i,k}(G) \circ \Pi^{k-1}_{k,k}(G)^* \circ \Pi^{k-1}_{k,j}(G) &= \\
\Pi^k_{i,j}(G). &
\end{aligned}$$

The above reasoning relies on the idempotence of $+$ in all cs-semirings $\mathsf{K} = (K, +, \cdot, *, 0, 1)$, and on the fact that

$$x \cdot y^* \cdot z \ = \ x \cdot z \ + \ x \cdot y \cdot z \ + \ x \cdot y \cdot y \cdot z \ + \ \ldots,$$

for all $x, y, z \in K$.

K-annotated graph databases and provenance. Let $G = (V, E, \Sigma)$ a graph database and $\mathsf{K} = (K, +, \cdot, *, 0, 1)$ a cs-semiring. A K-*annotation* of G is a mapping $h : E \to K$ from the edges of G to elements of K. The mapping h is homomorphically extended to paths $\pi \in \Pi(G)$ by $h(\pi) = h(e_1) \cdot h(e_2) \cdot \ldots \cdot h(e_n)$, where $\pi = e_1 e_2 \ldots e_n$, and to sets of paths P by $h(P) = \sum_{\pi \in P} h(\pi)$. We can now define the (K, h)-*provenance* of a pair $(i, j) \in Q(G)$ as

$$\mathsf{prov}^{\mathsf{K},h}_{i,j}\big(Q(G)\big) = h\big(\Pi_{i,j}(G_Q)\big).$$

Let X, with $|X| = |E|$, be a finite set of *abstract edge identifiers* and $\iota : E \to X$ an bijection. The next theorem shows that $\mathsf{prov}^{\mathsf{K}_{\mathsf{rex}[X]},\iota}_{i,j}\big(Q(G)\big)$ is the universal provenance for regular path queries. Note that $\mathsf{prov}^{\mathsf{K}_{\mathsf{rex}[X]},\iota}_{i,j}\big(Q(G)\big) \in \mathsf{rex}[X]$.

Theorem 3.3. *Let* $\mathsf{K} = (K, +, \cdot, *, 0, 1)$ *be a cs-semiring, G a graph database, and* ι *mapping each edge in G to a unique identifier in a set X. For every* K*-annotation* $h : E \to K$ *of G there is a mapping* $g : X \to K$*, such that* $g \circ \iota = h$*, and*

$$g\Big(\mathrm{prov}_{i,j}^{\mathsf{K}_{\mathrm{rex}[X]},\iota}\big(Q(G)\big)\Big) = \mathrm{prov}_{i,j}^{\mathsf{K},h}\big(Q(G)\big) \tag{4}$$

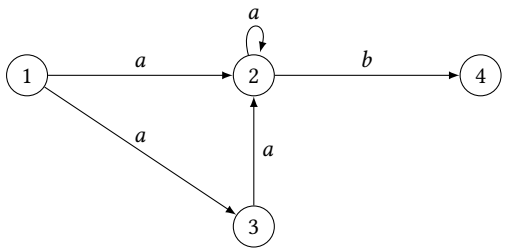
for all regular path queries Q.

Proof: In order to prove (4) it clearly is sufficient to show that

$$g\Big(\mathrm{prov}_{i,j}^{\mathsf{K}_{\mathrm{rex}[X]},\iota}\big(G\big)\Big) = \mathrm{prov}_{i,j}^{\mathsf{K},h}\big(G\big). \tag{5}$$

Let $g = \iota^{-1} \circ h$. Then $g \circ \iota = h$. To verify (5) we first note that $\mathrm{prov}_{i,j}^{\mathsf{K}_{\mathrm{rex}[X]},\iota}(G) = \iota(\Pi_{i,j}(G))$. Consequently $g(\mathrm{prov}_{i,j}^{\mathsf{K}_{\mathrm{rex}[X]},\iota}(G)) = g(\iota(\Pi_{i,j}(G))) = h(\Pi_{i,j}(G)) = \mathrm{prov}_{i,j}^{\mathsf{K},h}\big(G\big)$.

Examples. We illustrate the universal and application provenances next. Suppose that a query is $(a+b)^*b$ and that G_Q is the following graph.



Assume that edges are mapped to elements of cs-semirings by homomorphisms h_1 – h_4, as in the table below.

	ι	h_1	h_2	h_3	h_4
E	X	Σ	$\mathbb{R}_+^\infty$	$\mathbb{N}^\infty$	$\mathbb{A}$
edge	id	language	distance	multiplicity	access
$(1,a,2)$	$\mathtt{t}_1$	$\{a\}$	4	2	P
$(1,a,3)$	$\mathtt{t}_2$	$\{a\}$	1	4	C
$(3,a,2)$	$\mathtt{t}_3$	$\{a\}$	2	3	P
$(2,a,2)$	$\mathtt{t}_4$	$\{a\}$	3	5	P
$(2,b,4)$	$\mathtt{t}_5$	$\{b\}$	2	2	S

We then have the following provenances of answer $(1,4)$.

$$\begin{aligned}
\mathrm{prov}_{1,4}^{\mathsf{K}_{\mathrm{rex}[X]},\iota}(G_Q) &= (\mathtt{t}_1 + \mathtt{t}_2 \cdot \mathtt{t}_3) \cdot \mathtt{t}_4^* \cdot \mathtt{t}_5 \\
\mathrm{prov}_{1,4}^{\mathsf{K}_{\mathrm{reg}[\Sigma]},h_1}(G_Q) &= \{a\}^+ \cdot \{b\} \\
\mathrm{prov}_{1,4}^{\mathsf{K}_{\mathrm{trop}},h_2}(G_Q) &= \min(4, 1+2) + 0 + 2 = 5 \\
\mathrm{prov}_{1,4}^{\mathsf{K}_{\mathrm{bag}},h_3}(G_Q) &= (2 + 4 \cdot 3) \cdot \infty \cdot 2 = \infty \\
\mathrm{prov}_{1,4}^{\mathsf{K}_{\mathrm{acc}},h_4}(G_Q) &= \max(\max(\min(P, \max(C, P)), P), S) = S
\end{aligned}$$

4 REGULAR PATH QUERIES AND DATALOG

Green et al. also considered provenance for datalog queries in [13]. They propose that the provenance of fact Q in a program P is the semiring sum of the semiring product of the abstract identifiers of the leaves in all derivation trees of Q in P. As a fact can have an infinite number of derivations the provenances will be *formal power series* over $\mathbb{N}$ and $[X]$, and opposed to polynomials $\mathbb{N}[X]$. Alternatively, the provenance can be characterized as a solution to a system of algebraic equations[5] corresponding to Q and P.

It is well known that regular path queries can be represented in datalog. As pointed out by Vardi [27], regular path queries have the advantage over datalog (and context-free grammars) that decision problems are decidable. On the other hand, a datalog formulation of a regular path query will give a more fine grained provenance, as the provenance reflects the evaluation of the program, whereas the provenance of regular path queries represent the higher level concept of paths in a graph database. For an example, suppose the query is a^* and the database has one edge $(1, a, 1)$ abstractly labelled $\mathtt{t}$. Then the provenance of answer $(1, 1)$ is $\mathtt{t}^*$. If the query is formulated in datalog with rules $tr(x, y) \leftarrow a(x, y)$ and $tr(x, y) \leftarrow tr(x, z), tr(z, y)$, then answer $tr(1, 1)$ has as provenance the formal power series $\sum_{n=0}^{\infty} C_n \mathtt{t}^{n+1}$, where C_n is the n:th *Catalan number*, representing the number of ways a term with $n + 1$ factors can be parenthesized. Projecting this formal power series to the $WHY(X)$ provenance of [13] would yield the provenance $\mathtt{t}$, which is less informative than the regular path query provenance $\mathtt{t}^*$. Of course, for instance for program debugging purposes the datalog provenance could be useful.

5 EXPERIMENTAL EVALUATION

In recent studies, Ramusat et al. [22, 23], have undertaken a fairly thorough experimental analysis of various algorithms for computing the provenance of answers to regular path queries in graph databases. One of their main conclusions is that the best algorithm depends on the semiring in question. However, all the semirings experimentally considered in [22, 23] are application semirings, such as $\mathsf{K}_{\mathsf{bag}}$. Following the dictum *"compute provenance as generally as (computationally) feasible, then specialize via homomorphisms to coarser-grain provenance, or to specific domains, e.g., count, trust, cost or access control"* by Grädel and Tannen [12], we want to determine if computing the universal provenance in $\mathrm{rex}[X]$ is indeed computationally feasible.

The first task is to compute G_Q for a given graph database G and regular path query Q. This can be achieved by the standard product construction for the intersection of the langauges of finite state automata (see e.g. [15]). Here we have an automaton for Q, and G can be be viewed as a finite state automaton where all states (vertices) are both initial and final. We then project the product automaton to the second component, that is the states (vertices) of G. The result is then a graph database G_Q, such that $\Pi(G_Q) = \{\pi \in \Pi(G) : \mathrm{word}(\pi) \in L(Q)\}$.

Having G_Q, the next step for a given pair of vertices (i, j) is to compute a regular expression e representing the set of all (abstractly labelled) paths from i to j in G_Q. It is well known that Kleene's algorithm for computing $\mathsf{e}_{i,j}$ is not feasible in practice, due to the fact the size of the expressions $\mathsf{e}_{i,j}^k$ grow exponentially. We note however that if e is a regular expression such that $L(\mathsf{e}) = L(\mathsf{e}_{i,j})$, and we substitute $\mathrm{prov}_{i,j}^{\mathsf{K}_{\mathrm{rex}[X]},\iota}\big(Q(G)\big)$ with e, then Theorem 3.3 still holds. In our experiments we used the *state elimination procedure* [2] for obtaining a regular expression describing the language

[5] In this vein, Luttenberger and Schlund [18] show that under various assumptions of idempotence of the product operator in the semirings considered, the solution is regular and can be represented by regular expressions.

of a given finite state automaton. Let $A = (P, \Sigma, \delta, p_s, \{p_f\})$ be a finite state automaton.[6] The procedure is based on viewing the label a in a transition $(p, a, q) \in \delta$ as a regular expression denoting the language $\{a\}$. We then construct a regular expression $\mathtt{e}$ by removing, one by one, all states $q \in P \setminus \{p_s, p_f\}$. For each pair $(r, \mathtt{e}_1, q), (q, \mathtt{e}_2, s)$ of transitions, we replace them with the transition $(r, \mathtt{e}_1 \cdot \mathtt{e}_2, s)$. If there is a self-loop $(q, \mathtt{e}_3, q)$, the replaced transition is $(r, \mathtt{e}_1 \cdot \mathtt{e}_3^* \cdot \mathtt{e}_2, s)$. If there is another transition $(r, \mathtt{e}_4, s)$ this is merged with the new transition into $(r, \mathtt{e}_1 \cdot \mathtt{e}_3^* \cdot \mathtt{e}_2 + \mathtt{e}_4, s)$. After all pairs $(r, \mathtt{e}_1, q), (q, \mathtt{e}_2, s)$ have been processed, the possible self-loop $(q, \mathtt{e}_3, q)$ along with the state q is removed. When all states in $P \setminus \{p_s, p_f\}$ are removed and transitions are replaced, we are left with transitions $(p_s, \mathtt{e}_1, p_s)$, $(p_s, \mathtt{e}_2, p_f)$, $(p_f, \mathtt{e}_3, p_f)$, $(p_f, \mathtt{e}_4, p_s)$, for some regular expressions $\mathtt{e}_1, \mathtt{e}_2, \mathtt{e}_3$, and $\mathtt{e}_4$. This yields regular expression $\mathtt{e} = (\mathtt{e}_1 + \mathtt{e}_2 \cdot \mathtt{e}_3^* \cdot \mathtt{e}_4)^* \cdot \mathtt{e}_2 \cdot \mathtt{e}_3^*$. It then follows (see e.g. [15]) that $L(\mathtt{e}) = L(A)$.

In order to apply the state elimination procedure to $G_Q = (V, E, \Sigma)$, with $V = \{1, \ldots, n\}$ and answer pair (i, j), we consider G_Q as the finite state automaton $(A_{G_Q})_{i,j} = (V, X, \hat{E}, i, \{j\})$, where $\hat{E} = \{(r, \iota(r, a, s), s) : (r, a, s) \in E\}$. Here ι is the bijection from E to X assigning each edge its abstract identitifier. We have the following state of affairs:

Theorem 5.1. *Let $\mathtt{f}_{i,j}$ be the regular expression obtained by the state elimination procedure from $(A_{G_Q})_{i,j}$, and $P = \iota^{-1} \circ L$. Then $P(\mathtt{f}_{i,j}) = \Pi_{i,j}(G_Q)$.*

To summarize, given a database graph G_Q, a K-annotation h of the edges of G_Q, and a pair (i, j) of vertices of G_Q, we compute the application provenance $h(\Pi_{i,j}(G_Q))$ by computing $\mathtt{f}_{i,j}$ using the state elimination procedure, and then obtaining $h \circ P(\mathtt{f}_{i,j}) = h(\Pi_{i,j}(G_Q))$.

On the other hand, to compute $h(\Pi_{i,j}(G_Q))$ "directly" without first generating the universal provenance $\mathtt{f}_{i,j}$, we can use a result of Sakarovitch [24, 25] who has shown that if states are eliminated in order $1, 2, \ldots, n$, then the state elimination algorithm is isomorphic with Kleene's algorithm, and each expression $e_{r,s}^k$ obtained at the *k-th* stage of the latter algorithm is equal to the expression, call it $\mathtt{f}_{r,s}^k$, that represents the regular expression between states r and s after removing state k in the state elimination procedure.[7] In other words, $\mathtt{e}_{r,s}^k = \mathtt{f}_{rs}^k$, for all $k, r, s \in \{1, 2, \ldots, n\}$. In each stage, we evaluate

$$\begin{aligned} h \circ P(f_{i,j}^k) &= \\ h \circ P(e_{i,j}^k) &= \\ h \circ P(e_{i,j}^{k-1} + e_{i,k}^{k-1} \cdot (e_{k,k}^{k-1})^* \cdot e_{k,j}^{k-1}) &= \\ h \circ P(f_{i,j}^{k-1} + f_{i,k}^{k-1} \cdot (f_{k,k}^{k-1})^* \cdot f_{k,j}^{k-1}) &= \\ h \circ P(f_{i,j}^{k-1}) + h \circ P(f_{i,k}^{k-1}) \cdot h \circ P(f_{k,k}^{k-1})^* \cdot h \circ P(f_{k,j}^{k-1}). & \end{aligned}$$

Thus, when using the state elimination procedure for computing the application provenance $h(\Pi_{i,j}(G_Q))$, we never need to store (unwieldy large) regular expressions, only application semiring values $h(P(\mathtt{f}_{r,s}^k))$.

[6] There is no loss of generality assuming that the set of final states is a singleton.
[7] Note that we can choose any order of eliminating states simply by relabelling the states accordingly.

Determining State Removal Order. The size of the regular expression resulting from the state removal procedure heavily depends on the order in which states are removed. Several heuristics for determining the removal order have been studied [6, 14, 23]. Experiments show that the heuristics proposed by Delgado and Morais [6], the DM algorithm, produce the most compact regular expressions. The DM algorithm repeatedly computes the weight of each state and eliminates the state with the smallest weight. For states r and s, let $\mathrm{len}(r, s)$ denote the length of the label of the transition from r to s. For a state q, let $r_1, \ldots, r_m$ be the states from which there is a transition into q, and $s_1, \ldots, s_p$ states into which there is an outgoing transition from q. At each intermediate step of the state elimination procedure, the weight of state q is computed as $\mathrm{weight}(q) =$

$$\sum_{i=1}^{m} \mathrm{len}(r_i, q) \times (p-1) + \sum_{i=1}^{p} \mathrm{len}(q, s_i) \times (m-1) + \mathrm{len}(q, q) \times ((m \times p) - 1).$$

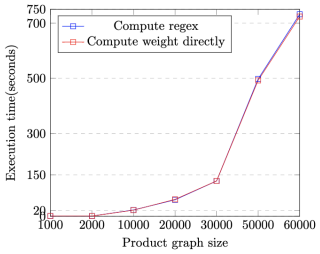
Ramusat et al. [23] proposed a new state removal heuristic based on the degree of each node. The degree of each node is defined as the sum of incoming and outgoing edges from the node, and recomputed after a node with the minimum degree is eliminated. Experiments reported in [17] showed that the algorithm [23] is slightly faster, whereas the DM algorithm tend to produce shorter regular expressions. In the experiments reported here we used the DM algorithm.

We compared experimentally the time difference between computing the universal provenance and then applying a homomorphism to obtain the application provenance as opposed to computing the application provenance directly. The computer we used for the experiments was a commodity PC which has 16GB of RAM, 256GB SSD, and runs an M1 processor.

In the experiments we used sparse graph databases to measure the performance of our algorithms. Most real-world networks are sparse, such as social, computer, and biological networks, as well as transportation and citation networks, etc. The chosen databases G were the *Retweet* network [5] from SNAP[8], and the *Yeast* protein-to-protein interaction network [19]. The *Retweet* network has 256,491 nodes and 328,132 edges, and the *Yeast* graph database has 2,361 nodes and 7,182 edges. We generated Σ-labels and tropical semiring weights to the edges of these datasets as they were missing annotations. The queries Q we used to generate product graphs were of sizes 3 to 10, where the size counts the number of occurrences of alphabet symbols in the query. As computing the product graph G_Q is standard, we didn't include this component in the times reported. We considered the databases as *single source*, and each line in Figures 1 and 2 represent 10 different trials. For each trial we chose one vertex in G randomly as single source and computed the product graph G_Q. The product graph size is the approximate average of the ten trials; the other columns show the average running times and average lengths of the resulting regular expressions.

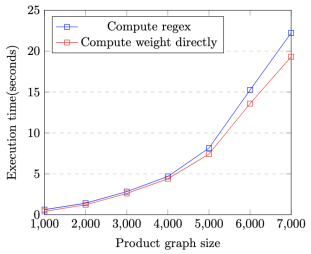
Figure 1 shows the comparison of two methods running on the *Retweet* network, and Figure 2 shows the experiment results for the *Yeast* network. The product graph size ranges from 1,000 to 60,000 vertices for the *Retweet* network, and from 1,000 to 7,000 vertices for the *Yeast* network. The execution times of both methods

[8] http://snap.stanford.edu/data/index.html



Product graph size	Time for regex (seconds)	Time for weight (seconds)	Average length of regex
1,000	0.79	0.631	1,601
2,000	0.79	0.63	1,654
10,000	22.79	22.22	28,432
20,000	59.51	61.17	64,500
30,000	127.95	128.16	83,766
50,000	497.62	492.37	413,666
60,000	732.10	723.90	477,862

Figure 1: Experimental results for Retweet Network



Product graph size	Time for regex (seconds)	Time for weight (seconds)	Average length of regex
1,000	0.63	0.42	3,905
2,000	1.42	1.22	10,704
3,000	2.83	2.61	34,938
4,000	4.69	4.41	112,952
5,000	8.15	7.42	392,192
6,000	15.25	13.59	1,255,144
7,000	22.23	19.31	1,907,156

Figure 2: Experimental results for Yeast Network

increase non-linearly as the product graph size grows, which is expected since the complexity of the state elimination procedure is exponential. For the *Retweet* network, we could not finish the experiments with a product graph size of 70,000 and above because of the limitation of hardware memory. As for the *Yeast* network, because it is small, and contains a large number of disconnect nodes, it was not possible to generate larger product graphs.

The experimental results show that computing the universal provenance (regular expression) first does not cause significant overhead compared to computing the tropical semiring weights directly. This shows its potential usage in practical scenarios, where different application provenances need to be computed based on the same graph. We also observe that the size of the regular expressions generated from the node elimination procedure increases significantly when the product graph size grows. This is however an inherent problem due to the fact that the size of regular expressions increase four times after each node is eliminated. In [21], the authors have compared multiple node elimination heuristics, and report regular expressions of exponential size already for small automata (graphs).

Since we use the *Yeast* database, as did [23], it would be natural to compare their experimental results with ours. The problem studied in [23] was the reachability problem and they performed node elimination (among other experiments) to get the application provenance of paths between two random nodes. Their results show that the node elimination method is applicable on graphs with low tree width graphs. The tree width of a graph measures how far the graph is from being a tree. However, the experiments in [23] might involve unnecessary state removals because not every node is intermediate and relevant to the input pair of nodes. This makes it difficult to directly compare our results with theirs. We note however that the running times of our algorithms and the ones in [23] are of the same order of magnitude.

ACKNOWLEDGEMENTS

Work partially supported by grants from NSERC Canada and Concordia University. Many thanks to Sandeep Chowdary[9] for assisting in implementing and conducting the experiments.

REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.
[2] J. A. Brzozowski and E. J. McCluskey. Signal flow graph techniques for sequential circuit state diagrams. *IEEE Trans. Electron. Comput.*, 12(2):67–76, 1963.
[3] S. Burris and H. P. Sankappanavar. *A course in universal algebra*, volume 78 of *Graduate texts in mathematics.* Springer, 1981.
[4] K. M. Dannert, E. Grädel, M. Naaf, and V. Tannen. Semiring provenance for fixed-point logic. In C. Baier and J. Goubault-Larrecq, editors, *29th EACSL Annual Conference on Computer Science Logic, CSL 2021, January 25-28, 2021, Ljubljana, Slovenia (Virtual Conference)*, volume 183 of *LIPIcs*, pages 17:1–17:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
[5] M. De Domenico, A. Lima, P. Mougel, and M. Musolesi. The anatomy of a scientific rumor. *Scientific Reports*, 3(1), Oct 2013.
[6] M. Delgado and J. Morais. Approximation to the smallest regular expression for a given regular language. In M. Domaratzki, A. Okhotin, K. Salomaa, and S. Yu, editors, *Implementation and Application of Automata*, pages 312–314, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
[7] M. Droste, W. Kuich, and H. Vogler. *Handbook of Weighted Automata.* Springer Publishing Company, Incorporated, 1st edition, 2009.
[8] S. Eilenberg. *Automata, languages, and machines. A.* Pure and applied mathematics. Academic Press, 1974.
[9] F. Geerts, G. Karvounarakis, V. Christophides, and I. Fundulaki. Algebraic structures for capturing the provenance of SPARQL queries. In W. Tan, G. Guerrini, B. Catania, and A. Gounaris, editors, *Joint 2013 EDBT/ICDT Conferences, ICDT '13 Proceedings, Genoa, Italy, March 18-22, 2013*, pages 153–164. ACM, 2013.
[10] F. Geerts and A. Poggi. On database query languages for k-relations. *J. Appl. Log.*, 8(2):173–185, 2010.
[11] B. Glavic. Data provenance. *Found. Trends Databases*, 9(3-4):209–441, 2021.
[12] E. Grädel and V. Tannen. Semiring provenance for first-order model checking. *CoRR*, abs/1712.01980, 2017.
[13] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In L. Libkin, editor, *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11-13, 2007, Beijing, China*, pages 31–40. ACM, 2007.
[14] Y.-S. Han and D. Wood. Obtaining shorter regular expressions from finite-state automata. *Theoretical Computer Science*, 370(1):110–120, 2007.
[15] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.
[16] D. Kozen. A completeness theorem for kleene algebras and the algebra of regular events. *Inf. Comput.*, 110(2):366–390, 1994.
[17] T. Liu. *Polynomials for Multidimensional Provenance in Graph Databases.* Master's Thesis, Concordia University, 2021.
[18] M. Luttenberger and M. Schlund. Regular expressions for provenance. In A. Chapman, B. Ludäscher, and A. Schreiber, editors, *6th Workshop on the Theory and Practice of Provenance, TaPP'14, Cologne, Germany, June 12-13, 2014.* USENIX Association, 2014.

[9] email: avgchowdary@gmail.com

[19] S. Maniu, P. Senellart, and S. Jog. An experimental study of the treewidth of real-world graph data. In *ICDT 2019 – 22nd International Conference on Database Theory*, page 18, Lisbon, Portugal, Mar. 2019.

[20] R. McNaughton and H. Yamada. Regular expressions and state graphs for automata. *IRE Trans. Electron. Comput.*, 9(1):39–47, 1960.

[21] N. Moreira, D. Nabais, and R. Reis. State elimination ordering strategies: Some experimental results. In I. McQuillan and G. Pighizzini, editors, *Proceedings Twelfth Annual Workshop on Descriptional Complexity of Formal Systems, DCFS 2010, Saskatoon, Canada, 8-10th August 2010*, volume 31 of *EPTCS*, pages 139–148, 2010.

[22] Y. Ramusat, S. Maniu, and P. Senellart. Semiring provenance over graph databases. In M. Herschel, editor, *10th USENIX Workshop on the Theory and Practice of Provenance, TaPP 2018, London, UK, July 11-12, 2018*. USENIX Association, 2018.

[23] Y. Ramusat, S. Maniu, and P. Senellart. Provenance-based algorithms for rich queries over graph databases. In Y. Velegrakis, D. Zeinalipour-Yazti, P. K. Chrysanthis, and F. Guerra, editors, *Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021*, pages 73–84. OpenProceedings.org, 2021.

[24] J. Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.

[25] J. Sakarovitch. Automata and rational expressions. *CoRR*, abs/1502.03573, 2015.

[26] V. Tannen. Provenance analysis for FOL model checking. *ACM SIGLOG News*, 4(1):24–36, 2017.

[27] M. Y. Vardi. A theory of regular queries. In T. Milo and W. Tan, editors, *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 1–9. ACM, 2016.

[28] Z. Yan, V. Tannen, and Z. G. Ives. Fine-grained provenance for linear algebra operators. In S. C. Boulakia, editor, *8th USENIX Workshop on the Theory and Practice of Provenance, TaPP 2016, Washington, D.C., USA, June 8-9, 2016*. USENIX Association, 2016.