Regular Expressions for Provenance

Michael Luttenberger
TU München
luttenbe@model.in.tum.de

Maximilian Schlund
TU München
schlund@model.in.tum.de

Abstract

As noted by Green et al. several provenance analyses can be considered a special case of the general problem of computing formal polynomials resp. power-series as solutions of an algebraic system. Specific provenance is then obtained by means of evaluating the formal polynomial under a suitable homomorphism.

Recently, we presented the idea of approximating the least solution of such algebraic systems by means of unfolding the system into a sequence of simpler algebraic systems. Similar ideas are at the heart of the semi-naive evaluation algorithm for datalog.

We apply these results to provenance problems: Semi-naive evaluation can be seen as a particular implementation of fixed point iteration which can only be used to compute (finite) provenance polynomials. Other unfolding schemes, e.g. based on Newton's method, allow us to compute a regular expression which yields a finite representation of (possibly infinite) provenance power series in the case of commutative and idempotent semirings. For specific semirings (e.g. Why(X)) we can then, in a second step, transform these regular expressions resp. power series into polynomials which capture the provenance. Using techniques like subterm sharing both the regular expressions and the polynomials can be succinctly represented.

Keywords Provenance, Datalog, Semirings

1. Introduction

As noted e.g. by Green et al. in [6], several provenance analyses can be seen as particular instances of the problem of solving algebraic systems (systems of polynomial equations) over semirings. These different analyses can be unified by considering the algebraic system over the semiring $\overline{\mathbb{N}}\langle\langle\mathfrak{A}^{\oplus}\rangle\rangle$ of all formal power-series in the commuting variables $\mathfrak{A}$ and coefficients in $\overline{\mathbb{N}} = \mathbb{N} \cup \{\infty\}$. W.r.t. $\overline{\mathbb{N}}\langle\langle\mathfrak{A}^{\oplus}\rangle\rangle$ a least solution always exists which captures how the elements of $\mathfrak{A}$ influence the solution, and it specializes to the results of specific provanence analyses, like why-provenance, by means of unique homomorphisms. Already in the 1960s Schützenberger initialized the study of algebraic systems in formal language theory as generalization of the concept of context-free grammar, see e.g. [1], but in contrast to provenance analyses one also considers the setting where the elements of $\mathfrak{A}$ do not commute, i.e. the monomials of the formal power-series become words over $\mathfrak{A}$. This semiring is denoted by $\overline{\mathbb{N}}\langle\langle\mathfrak{A}^{*}\rangle\rangle$. For instance, the ambiguity of a context-free grammar is the formal power-series where the coefficient of a word $w \in \mathfrak{A}^*$ is the number of derivation trees yielding w, and this power-series is the least solution of an algebraic system induced by the context-free grammar. In particular, the problem studied in [6] can be restated as the problem of computing the commutative ambiguity of a context-free grammar, i.e. the coefficient of a monomial (commutative word) m over $\mathfrak{A}$ is the number of derivation trees of a certain context-free grammar which yield a word $w \in \mathfrak{A}^*$ that is equivalent to m up to commutativity. We recall this connection in the following section in greater detail.

Lately [4, 5, 7], we have used this close connection between context-free grammars and algebraic systems in order to derive several results on the computation of the least solution of an algebraic system over specific semirings: these results allow to approximate the (least) solution of an algebraic system at least as fast as algorithms based on the classical fixed-point iteration (algorithms like "Gauss-Seidel", "chaotic iteration using a work-list", "semi-naive evaluation") and at times even converge to the solution within a finite number of steps while classical fixed-point iteration does not.

Contributions In this article, we provide the following insights:

- Exploiting the connection between context-free grammars, Datalog programs, and algebraic systems we show how iteration schemes based on grammar unfolding – like Newton's method – can be applied to provenance problems for recursive Datalog.
- Under various forms of idempotence, this allows us to compute regular expressions which represent in a finite way the solution of the provenance equations over a very general semiring.
- For specific provenance analyses, these regular expressions can be converted to polynomials (without sacrificing their concise representation using subterm sharing). We discuss this in detail for the Why-provenance.

Missing proofs can be found in the appendix of the extended version of this article available at `http://www7.in.tum.de/~schlund/tapp14-ext.pdf`.

2. Preliminaries

By $\mathbb{N}$ we denote the natural numbers including zero. The extended natural numbers $\overline{\mathbb{N}} = \mathbb{N} \cup \{\infty\}$ include a greatest element ∞. A *semiring* $\langle S, +, \cdot, 0, 1\rangle$ is an algebraic structure satisfying:

$$x + y = y + x \qquad x + 0 = x$$
$$x \cdot 1 = 1 \cdot x = x \qquad x \cdot 0 = 0 \cdot x = 0$$
$$x \cdot (y + z) = x \cdot y + x \cdot z \qquad (x + y) \cdot z = x \cdot z + y \cdot z$$

As usual we write xy for $x \cdot y$. If $xy = yx$ holds, the semiring is called *commutative*; if $1 + 1 = 1$, it is *idempotent*. We are mostly concerned with the semirings $\overline{\mathbb{N}}\langle\langle\mathfrak{A}^{\oplus}\rangle\rangle$ and $\overline{\mathbb{N}}\langle\langle\mathfrak{A}^{*}\rangle\rangle$: Their elements are the functions from the monomials (commuting words) resp.

TaPP '14, June 12–13, 2014, Cologne, Germany.
Copyright © 2014 ACM 978-1-nnnn-nnnn-n/yy/mm. . . $15.00.
http://dx.doi.org/10.1145/nnnnnnn.nnnnnnn

words over $\mathfrak{A}$ to $\overline{\mathbb{N}}$; addition is defined pointwise, multiplication is defined via the Cauchy product. Due to the page limit, we will forgo most other formal definitions and instead try to convey the most important ideas in an extended example. For details on the theory of semirings and algebraic systems we refer the reader to [3].

Datalog, provenance, context-free grammars, and algebraic systems We recall the basic connection between Datalog evaluation, provenance analysis, context-free grammars, and algebraic systems by means of a simply example summarized in Fig. 1: We compute the kinsman-relation K via the ancestor-relation A from an extensional parent-relation P. We will assume for the rest of this section that $\mathsf{P}(x, y)$ is true if and only if there is an edge from x to y in the graph shown in Fig. 1. The program then computes e.g. that "Clothar and Theuderich are kinsmen" ($\mathsf{K}(\text{Chlothar}, \text{Theuderich})$). We obtain from this program a context-free grammar by replacing ": −" by "→" and removing "," (Fig. 1, middle left). A predicate symbol like P then gives rise to a family $\mathsf{P}(x, y)$ of nonterminals. For every EDB fact, we introduce a unique identifier $\mathfrak{A} = \{a, b, \ldots, g\}$ (as hinted in the graph show on the right of Fig. 1) and add corresponding rewrite rules for the nonterminals associated with the EDB predicates, thereby implicitly defining the domain of the parameters x, y, z. We then have that the Datalog program derives $\mathsf{K}(x, y)$ if and only if the nonterminal $\mathsf{K}(x, y)$ is productive. Deciding whether a nonterminal is productive can be restated as the problem of computing the least solution of the algebraic system depicted on the bottom left of Fig. 1 when interpreting the system over the Boolean semiring $\langle \mathbb{B}, \vee, \wedge, 0, 1\rangle$: That is, addition becomes logical disjunction, multiplication becomes logical conjunction, and we assign each identifier in $\mathfrak{A}$ the value 1 (true). The least solution of this system then assigns the value 1 to the variable $\mathsf{K}(x, y)$ if and only if the nonterminal $\mathsf{K}(x, y)$ is productive. Thus, computing the semantics of the Datalog program w.r.t. the given EDB means to compute the least solution of this algebraic system over the Boolean semiring. As all ascending chains in the Boolean semiring eventually terminate, this can efficiently be done using algorithms based on standard fixed-point iteration $f(0), f(f(0)), \ldots$ (e.g. Gauss-Seidel, semi-naive evaluation, constant propagation).

If we are interested in those EDB facts which are necessary in order to deduce that "Clothar and Theuderich are kinsmen" we need to study how the bound variables of the algebraic system depend on the free variables (induced by the EDB facts). Using conventional fixed-point iteration (setting all bound variables to 0 initially) and simplifying obtained terms using only the equalities holding in any semiring we obtain polynomials in (non-commuting) variables $\mathfrak{A}$ as solutions, e.g.:

$$\mathsf{K}(\text{Chlothar}, \text{Theuderich}) = dg + cdcg + bcdbcg + abcdabcg.$$

We deliberately have not assumed that the elements of $\mathfrak{A}$ commute to point out that these polynomials simply are the sum of all words (non-commutative monomials) encoding the paths which connect x and y to a common ancestor z in above graph; implicitly, every word has the coefficient 1 in this example. Yet another way to describe this solution is that it is the sum of all words generated by the associated context-free grammar. This in turn means that the solution of $\mathsf{K}(x, y)$ is the sum of all words for which there is a derivation tree w.r.t. above context-free grammar when taking $\mathsf{K}(x, y)$ as the start symbol; the coefficient of a word counts the number of distinct derivation trees yielding the word. Thus, what we actually computed is the ambiguity function of the grammar. In our example this grammar only generates a finite language and the grammar itself is unambiguous – thus conventional fixed-point iteration suffices for computing the solution; but in general the context-free grammars resp. algebraic systems represent infinite languages and are ambiguous; in this case, the solution will be a (formal) power-series where words can take arbitrary coefficients in $\overline{\mathbb{N}}$ and fixed-point iteration does not suffice anymore (see Section 3).[1] If we further allow the elements of $\mathfrak{A}$ to commute we can simplify words to monomials (e.g. $cdcg$ to c^2dg) and we obtain the *commutative ambiguity* of the grammar [7].[2] In the following, we discuss how to compute regular expressions which represent these power-series under the restrictions that elements of $\mathfrak{A}$ commute and (some form of generalized) idempotence holds – which is the case for several provenance analyses [6]:

In the case of why-provenance, we are only interested in those EDB facts which are necessary for a derivation to exist, but we only need to know whether an EDB fact is used at least once. Hence, we would like to simplify a word like $bcdbcg$ to $bcdg$. Algebraically this amounts to the identities $xy = yx$ and $x^2 = x$ $x, y \in \mathfrak{A}^*$:

$$\begin{aligned}\mathsf{K}(\text{Chlothar}, \text{Theuderich}) &= dg + c^2dg + b^2c^2dg + a^2b^2c^2dg \\ &= dg + cdg + bcdg + abcdg\end{aligned}$$

To further reduce the polynomials to those monomials which encode the minimal sets of EDB facts required to deduce $\mathsf{K}(x, y)$ we can further add the equality $1 + x = 1$ (obtaining the PosBool semiring):[3]

$$\mathsf{K}(\text{Chlothar}, \text{Theuderich}) = dg(1 + c + bc + abc) = dg$$

As mentioned above, we propose to represent the solution of an algebraic system (up to commutativity and idempotence) as a regular expression when the actual solution is not a polynomial. Still, modulo the additional equalities underlying specific provenance analyses these power-series might reduce to polynomials. Hence, we will also discuss how we can efficiently reduce these regular expressions to polynomials modulo the additional equalities.

3. Solving algebraic systems: Newton's method for Datalog programs

We assume in the following that elements of $\mathfrak{A}$ commute. As sketched in the preceding section, provenance analyses resp. semantics for Datalog programs can be reduced to the problem of solving algebraic systems in the most generic way, namely over the semiring $\overline{\mathbb{N}}\langle\langle \mathfrak{A}^{\oplus}\rangle\rangle$ of formal power-series. While in our example of Section 2 the solution of the algebraic system is only a polynomial, in general this is of course not true. Just consider the example in Fig. 2 taken from [6]: The program depicted there computes the transitive closure T of an extensional edge relation E. Here we assume w.l.o.g. that we already know the resulting IDB facts, i.e which variables of the algebraic system are productive, and thus reduce the algebraic system to these variables. In contrast to the preceding example, fixed-point iteration now does not terminate within a finite number of steps, e.g. consider the equation $V = r + V^2$. Here, fixed-point iteration yields the sequence

$$0, \quad r, \quad r + r^2, \quad r + r^2 + 2r^3 + r^4, \quad \ldots \quad \to \sum_{n=0}^{\infty} C_n r^{n+1}$$

which converges to the generating series of binary trees whose coefficients are the Catalan numbers C_n, i.e. the number of binary trees which have exactly $n + 1$ leaves. In fact, the h-th approximation generated by the fixed-point iteration can be characterized

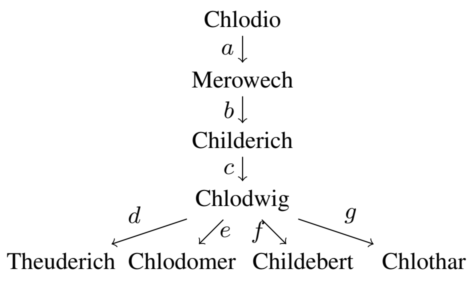
[1] Only if the grammar allows for cyclic derivations which do not produce any letters, a coefficient can be ∞.

[2] Unambiguity resp. finite commutative ambiguity can be enforced by extending every rule of the grammar by a terminal uniquely identifying the rule (resulting in a proper grammar resp. proper algebraic system).

[3] I.e. 1 is the greatest element. Semirings satisfying this property have been called 1-*bounded* in [4]. In [2] also the term *absorptive* is used.

$$\begin{array}{lll}
\mathsf{K}(x,y) :- \mathsf{A}(z,x),\mathsf{A}(z,y) & \mathsf{K}(x,y) :- \mathsf{A}(x,y) & \mathsf{K}(x,y) :- \mathsf{A}(y,x) \\
\mathsf{A}(x,y) :- \mathsf{A}(x,z),\mathsf{A}(z,y) & \mathsf{A}(x,y) :- \mathsf{P}(x,y) & \\
\hline
\mathsf{K}(x,y) \to \mathsf{A}(z,x),\mathsf{A}(z,y) & \mathsf{K}(x,y) \to \mathsf{A}(x,y) & \mathsf{K}(x,y) \to \mathsf{A}(y,x) \\
\mathsf{A}(x,y) \to \mathsf{A}(x,z),\mathsf{A}(z,y) & \mathsf{A}(x,y) \to \mathsf{P}(x,y) & \\
\mathsf{P}(\text{Chlodio},\text{Merowech}) \to a & \dots & \mathsf{P}(\text{Chlodwig},\text{Chlothar}) \to g \\
\hline
\end{array}$$

$$\begin{aligned}
&\mathsf{K}(x,y) = \textstyle\sum_z \mathsf{A}(z,x)\mathsf{A}(z,y) + \mathsf{A}(x,y) + \mathsf{A}(y,x) \\
&\mathsf{A}(x,y) = \textstyle\sum_z \mathsf{A}(x,z)\mathsf{A}(z,y) + \mathsf{P}(x,y) \\
&\mathsf{P}(\text{Chlodio},\text{Merowech}) = a \quad \dots \quad \mathsf{P}(\text{Chlodwig},\text{Chlothar}) = g
\end{aligned}$$



$$\begin{tikzcd}
 & & \text{Chlodio} \arrow[d, "a"'] & & \\
 & & \text{Merowech} \arrow[d, "b"'] & & \\
 & & \text{Childerich} \arrow[d, "c"'] & & \\
 & & \text{Chlodwig} \arrow[dll, "d"'] \arrow[dl, "e"] \arrow[d, "f"'] \arrow[dr, "g"] & & \\
\text{Theuderich} & \text{Chlodomer} & \text{Childebert} & \text{Chlothar} &
\end{tikzcd}$$

Figure 1. *Left*: A Datalog program (top), its associated context-free grammar (middle), and algebraic system (bottom). *Right*: The graph underlying the EDB instance used in the example.

$$\begin{aligned}
T(x,y) &= E(x,y) + T(x,z)T(z,y) \\
\hline
T^{<1}(x,y) &= T^{=0}(x,y) = E(x,y) \\
T^{<h}(x,y) &= T^{=h-1}(x,y) + T^{<h-1}(x,y) \\
T^{=h}(x,y) &= T^{=h-1}(x,y)T^{<h-1}(x,y) + T^{<h}(x,y)T^{=h-1}(x,y) \\
\hline
T^{<1}(x,y) &= T^{=0}(x,y) = E(x,y) \\
T(x,y)^{<d} &= T^{=d-1}(x,y) + T^{<d-1}(x,y) \\
T(x,y)^{=d} &= T^{<d}(x,z)T^{=d}(z,y) + T^{=d}(x,z)T^{<d}(z,y) \\
&\quad + T^{=d-1}(x,z)T^{=d-1}(z,y)
\end{aligned}$$

a	b	m
a	c	n
c	b	p
b	d	q
d	d	r
a	b	X
a	c	Y
c	b	Z
b	d	U
d	d	V
a	d	W

$$\begin{array}{ll}
X = m + YZ & U = q + UV \\
Y = n & V = r + V^2 \\
Z = p & W = XU + WV \\
\hline
\end{array}$$

$$\begin{aligned}
V^{<1} &= V^{=0} = r \\
V^{<d} &= V^{<d-1} + V^{=d-1} \\
V^{=d} &= \left(V^{=d-1}\right)^2 + V^{=d}V^{<d} + V^{<d}V^{=d} \\
&\overset{\text{comm.}}{=} \left(V^{=d-1}\right)^2 + 2 \cdot V^{=d}V^{<d} \\
&\overset{\text{Kleene star}}{=} \left(2 \cdot V^{<d}\right)^* \cdot \left(V^{=d-1}\right)^2
\end{aligned}$$

Figure 2. From left to right: Datalog program (represented as parametrized algebraic system) computing the transitive closure T of E, its unfolding w.r.t. height as done by the semi-naive evalution, and its unfolding w.r.t. dimension as done by Newton's method; EDB facts tagged with free variables $\mathfrak{A} = \{m,n,p,q,r\}$, and IDB facts computed by the program with abbrevations ($X := T(a,b)$); algebraic system reduced to the productive (non-zero) variables, and Newton's method applied to $V = r + V^2$.

as the sum where the coefficient of r^{n+1} is exactly the number of derivation trees yielding r^{n+1} but whose height is less than h. Thus any algorithm based on the conventional fixed-point iteration (Gauss-Seidl, semi-naive, work list, ...) can only produce finite approximations of $\sum_{n\geq 0} C_n r^{n+1}$.

In [5] we have shown that we can suitably generalize Newton's method, whose approximations of the least solution can be represented by means of rational expressions (i.e. regular expressions with cofficients in $\mathbb{N}$); in [7] we discuss its rate of convergence and how to determine those monomials in the least solution whose coefficient is equal to a given $k \in \overline{\mathbb{N}}$. While conventional fixed-point iteration is connected to the height of derivation trees, Newton's method is connected to the *Strahler number (dimension)* [5, 9] of a derivation tree. Using this connection, we can rewrite above Datalog program to a linear program (presented for succinctness as an algebraic system) as show in Fig. 2 (middle left). Like semi-naive evaluation, Newton's method also ensures that all derivations are considered (in the limit) exactly once; but while semi-naive evaluation can be understood as unfolding the Datalog program (grammar, algebraic system) w.r.t. the height of the derivation trees[4] Newton's method unravels the Datalog program w.r.t. the dimension d. On the bottom right of Fig. 2 we have applied Newton's method to $V := T(d,d)$ and used the Kleene star $x^* = \sum_{k=0}^{\infty} x^k$ to get rid of the linear recursion w.r.t. the variable $V^{=d}$ thereby obtaining again a recursion-free (acyclic) algebraic system which now can be easily solved bottom-up. Most notably, it can be shown that modulo idempotence ($1 + 1 = 1$) resp. some generalization of it (there is some $k \in \mathbb{N}$ s.t. $k + 1 = k$) the number of productive variables of the algebraic system bounds the dimension d up to which we need to unravel the program. For instance modulo $1 + 1 = 1$:

$$\begin{aligned}
V^{<2} &= V^{<1} + V^{=1} = V^{<1} + (2V^{=0})^*(V^{=0})^2 \\
&= r + (2r)^* r^2 \overset{1+1=1}{=} rr^* \overset{1+1=1}{=} \sum_{k=0}^{\infty} C_n r^{n+1}.
\end{aligned}$$

Note that modulo ($2 + 1 = 2$) only $V^{<3}$ would have given us the complete solution. We therefore propose to use Newton's method for actually solving the algebraic system over $\overline{\mathbb{N}}\langle\!\langle \mathfrak{A}^{\oplus} \rangle\!\rangle$ modulo some form of idempotence by means of rational expressions. In the following section, we describe in more detail how this can be done in general, how the rational expressions can be succinctly stored using subterm-sharing, and reduce them to polynomials when additional equalities hold w.r.t. a specific provenance analysis.

4. Representing Solutions by Regular Expression

Newton's method as described in Section 3 produces a linear (algebraic) system $X^{=d} = A \cdot X^{=d} + b$ for each "stage" d. A is a matrix containing only variables of lower index ($X^{=d-1}$ or $X^{<d}$) or semiring values. Inductively, we can assume that lower index variables are already represented as regular expressions.

We can solve such a linear system by computing $X = A^* b$, where the Kleene star for the matrix A can be computed via a modified Floyd-Warshall algorithm or via a recursive divide-and-

[4] The auxiliary IDB predicates introduced by semi-naive evaluation partition the derivations w.r.t. height exactly h resp. at most h.

conquer approach (cf. [3]): both algorithms reduce the Kleene star of a matrix to the Kleene star on the underlying semiring, thus yielding rational expressions. In the preceding section, we have already seen that modulo $1 + 1 = 1$ the solution of $V = r + V^2$ is $V = rr^* = \sum_{k=0}^{\infty} r^{k+1}$. This leads to the final solution of the system over the commutative, idempotent semiring freely generated by $\mathfrak{A}$:

$$\begin{aligned} X &= m + np & Y &= n \\ Z &= p & U &= qr^* \\ V &= rr^* & W &= (m + np)qr^* \end{aligned}$$

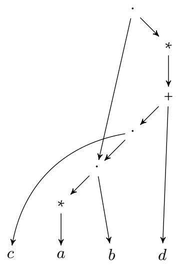


Figure 3. Representing $a^*b(ca^*b+d)^*$.

Concise Representation Since all semiring operations involve either terminals or previously computed expressions, we can store them concisely using a shared pointer structure. This sharing of common subexpressions is a standard technique from compiler construction and was also implemented in our tool described in [8]. The shared structure can also be seen as a generalized *provenance circuit* [2] having additional Kleene star nodes.

Complexity For algebraic systems with n variables over a commutative idempotent semiring we have shown that the $n+1$-st unfolding $X^{<n+1}$ is already equal to the least fixed-point. In each stage of the unfolding a system of n linear equations is to be solved which requires $\mathcal{O}(n^3)$ semiring operations (e.g. using Floyd-Warshall). Hence, the complexity of the solution procedure in terms of semiring operations is bounded by $\mathcal{O}(n^4)$. Since every semiring operation corresponds to a node (of type $+, \cdot, ^*$ or terminal symbol) in the shared structure described above, the space needed to store these regular expressions is also bounded by $\mathcal{O}(n^4)$.

4.1 Simplifying Representations over Special Semirings

General Commutative Idempotent Semirings So far, we have shown how to compute *regular* expressions that compactly represent provenance information. These expressions may contain Kleene stars which could be applied to complicated terms (i.e. not only to alphabet symbols),

$$x = b(ab^* + c)^*.$$

In many cases we may want to eliminate these stars to get "polynomial expressions" (i.e. involving only + and ·), which is not possible in general if the semiring admits infinite ascending chains. However, one can transform rational expressions into a finite sum of "linear" expressions, where stars are only applied to monomials. In this form it is usually trivial to eliminate the stars over a more specialized semiring. The transformation uses well-known identities which hold over every commutative, idempotent semiring (cf. [3]):

$$(x + y)^* = x^* \cdot y^* \qquad (xy^*)^* = 1 + xx^*y^* \qquad (x^*)^* = x^*$$

Applying these to our example yields $x = b(ab^* + c)^* = b(ab^*)^*c^* = b(1 + aa^*b^*)c^*$. Over semirings satisfying $x^* = 1 + x$ (see below), this expression simplifies to $x = b(1 + ab(1 + a)(1 + b))(1 + c)$.

This transformation can be carried out over every commutative idempotent semiring, but the resulting "semi-linear" expression may become exponentially larger in the worst case. In special cases, like why-provenance, eliminating the star-expressions can be done more efficiently in polynomial time.

Polynomial-time Star Elimination for k-Closed Semirings A semiring is k-closed ([3]) if $x^* = \sum_{i=0}^{k} x^i$. For any such semiring, we can use the truncation $x^* = \sum_{i=0}^{k} x^i$ to eliminate Kleene stars.

In the case of why-provenance the additional identity $x \cdot x = x$ for all $x \in \mathfrak{A}$ leads to a k-closed semiring: one immediately obtains from this identity that also $x^* = 1 + x$ holds for all $x \in \mathfrak{A}$. This, in turn, allows to prove for *all* elements of the semiring that

$$x^* = 1 + x^{N(x)}$$

where $N(x)$ is the number of alphabet symbols occuring in x. Thus, modulo $x \cdot x = x \; \forall x \in \mathfrak{A}$ the semiring becomes $|\mathfrak{A}|$-closed. For our example this yields $x = b(ab^* + c)^* = b(1 + a(1 + b) + c)^3$. Note that by sharing subexpressions, we can represent the power x^N of any regular expression by introducing at most $O(\log N)$ new multiplication nodes. Hence, we can obtain polynomial[5] sized representations of why-provenance.

Note that if we had required $x^2 = x$ for *all* semiring elements (idempotent multiplication) we would have obtained a different semiring, which might lose some explanations, as e.g. $(a + b)^2$ now simplifies to $a + b$, loosing the explanation ab.

5. Discussion

We have shown how to compute finite, concise regular expressions representing formal power series solutions to algebraic equations over commutative, idempotent semirings and how these relate to provenance for Datalog. We have sketched how to specialize these expressions (by eliminating Kleene stars and simplifying expressions) to semirings satisfying additional axioms like Why or PosBool.

Going beyond idempotence, our results on k-collapsed semirings [7] (where $k = k + 1$ holds) allow us to approximate provenance with bag-semantics (e.g. Trio). For special semirings (star-distributive, lossy, 1-bounded) [4] there exist improved unfoldings that produce even simpler solutions; e.g. idempotent multiplication yields a star-distributive semiring. We only note that star-distributive subsume tropical semirings like $\langle \overline{\mathbb{Z}}, \min, +, 0, \infty \rangle$, hence these results allow to solve algebraic system over these semirings, thereby answering an open question of [6].

References

[1] N. Chomsky and M. Schützenberger. *Computer Programming and Formal Systems*, chapter The Algebraic Theory of Context-Free Languages, pages 118 – 161. North Holland, 1963.

[2] D. Deutch, T. Milo, S. Roy, and V. Tannen. Circuits for datalog provenance. In *ICDT*, pages 201–212, 2014.

[3] M. Droste, W. Kuich, and H. Vogler. *Handbook of Weighted Automata*. Springer, 2009.

[4] J. Esparza, S. Kiefer, and M. Luttenberger. Derivation tree analysis for accelerated fixed-point computation. In *DLT*, pages 301–313, 2008.

[5] J. Esparza, S. Kiefer, and M. Luttenberger. Newtonian program analysis. *J. ACM*, 57(6):33, 2010.

[6] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.

[7] M. Luttenberger and M. Schlund. Convergence of Newton's Method over Commutative Semirings. In *LATA*, volume 7810 of *LNCS*, pages 407–418, 2013.

[8] M. Schlund, M. Terepeta, and M. Luttenberger. Putting Newton into Practice: A Solver for Polynomial Equations over Semirings. In *LPAR 2013*, volume 8312 of *LNCS*, pages 727–734, 2013.

[9] A. N. Strahler. Hypsometric (area-altitude) analysis of erosional topology. *Geol. Soc. Am. Bull.*, 63(11):1117–1142, 1952.

[5] Polynomial in the size of the equation system, which is determined by the EDB.