Provenance Circuits for Trees and Treelike Instances

Antoine Amarilli[1(✉)], Pierre Bourhis[2], and Pierre Senellart[1,3]

[1] Institut Mines-Télécom, Télécom ParisTech, CNRS LTCI, Paris, France
antoine.amarilli@telecom-paristech.fr
[2] CNRS CRIStAL, Université Lille 1, INRIA Lille, Lille, France
pierre.bourhis@univ-lille1.fr
[3] National University of Singapore, CNRS IPAL, Singapore, Singapore
pierre.senellart@telecom-paristech.fr

Abstract. Query evaluation in monadic second-order logic (MSO) is tractable on trees and treelike instances, even though it is hard for arbitrary instances. This tractability result has been extended to several tasks related to query evaluation, such as counting query results [2] or performing query evaluation on probabilistic trees [8]. These are two examples of the more general problem of computing augmented query output, that is referred to as *provenance*. This article presents a provenance framework for trees and treelike instances, by describing a linear-time construction of a circuit provenance representation for MSO queries. We show how this provenance can be connected to the usual definitions of semiring provenance on relational instances [17], even though we compute it in an unusual way, using tree automata; we do so via intrinsic definitions of provenance for general semirings, independent of the operational details of query evaluation. We show applications of this provenance to capture existing counting and probabilistic results on trees and treelike instances, and give novel consequences for probability evaluation.

1 Introduction

A celebrated result by Courcelle [9] has shown that evaluating a fixed monadic second-order (MSO) query on relational instances, while generally hard in the input instance, can be performed in linear time on input instances of *bounded treewidth* (or *treelike* instances), by encoding the query to an automaton on tree encodings of instances. This idea has been extended more recently to monadic Datalog [14]. In addition to query evaluation, it is also possible to *count* in linear time the number of query answers over treelike instances [2,22].

However, query evaluation and counting are special cases of the more general problem of capturing *provenance information* [7,17] of query results, which describes the link between input and output tuples. Provenance information can be expressed through various formalisms, such as *provenance semirings* [17] or Boolean formulae [26]. Besides counting, provenance can be exploited for practically important tasks such as answering queries in incomplete databases, maintaining access rights, or computing query probability [26]. To our knowledge,

M.M. Halldórsson et al. (Eds.): ICALP 2015, Part II, LNCS 9135, pp. 56–68, 2015.
DOI: 10.1007/978-3-662-47666-6_5

no previous work has looked at the general question of efficient evaluation of expressive queries on treelike instances while keeping track of provenance.

Indeed, no proper definition of provenance for queries evaluated via tree automata has been put forward. The first contribution of this work (Section 3) is thus to introduce a general notion of *provenance circuit* [11] for tree automata, which provides an efficiently computable representation of all possible results of an automaton over a tree with uncertain annotations. Of course, we are interested in the provenance of *queries* rather than automata; however, in this setting, the provenance that we compute has an intrinsic definition, so it does not depend on which automaton we use to compute the query.

We then extend these results in Section 4 to the provenance of queries on treelike relational instances. We propose again an intrinsic definition of provenance capturing the subinstances that satisfy the query. We then show that, in the same way that queries can be evaluated by compiling them to an automaton on tree encodings, we can compute a provenance circuit for the query by compiling it to an automaton, computing a tree decomposition of the instance, and performing the previous construction, in linear time overall in the input instance. Our intrinsic definition of provenance ensures the provenance only depends on the logical query, not on the choice of query plan, of automaton, or of tree decomposition.

Our next contribution in Section 5 is to extend such definitions of provenance from Boolean formulae to $\mathbb{N}[X]$, the *universal provenance semiring* [17]. This poses several challenges. First, as semirings cannot deal satisfactorily with negation [1], we must restrict to *monotone* queries, to obtain monotone provenance circuits. Second, we must keep track of the multiplicity of facts, as well as the multiplicity of matches. For this reason, we restrict to unions of conjunctive queries (UCQ) in that section, as richer languages do not directly provide notions of multiplicity for matched facts. We generalize our notion of provenance circuits for automata to instances with unknown multiplicity annotations, using arithmetic circuits. We show that, for UCQs, the standard provenance for the universal semiring [17] matches the one defined via the automaton, and that a provenance circuit for it can be computed in linear time for treelike instances.

Returning to the non-monotone Boolean provenance, we show in Section 6 how the tractability of provenance computation on treelike instances implies that of two important problems: determining the probability of a query, and counting query matches. We show that probability evaluation of fixed MSO queries is tractable on probabilistic XML models with local uncertainty, a result already known in [8], and extend it to trees with event annotations that satisfy a condition of having bounded *scopes*. We also show that MSO query evaluation is tractable on treelike block-independent-disjoint (BID) relational instances [26]. These tractability results for provenance are achieved by applying message passing [20] on our provenance circuits. Last, we show the tractability of counting query matches, using a reduction to the probabilistic setting, capturing a result of [2].

2 Preliminaries

We introduce basic notions related to trees, tree automata, and Boolean circuits.

Given a fixed *alphabet* Γ, we define a *Γ-tree* $T = (V, L, R, \lambda)$ as a set of *nodes* V, two partial mappings $L, R : V \to V$ that associate an internal node with its left and right child, and a *labeling function* $\lambda : V \to \Gamma$. Unless stated otherwise, the trees that we consider are rooted, directed, ordered, binary, and full (each node has either zero or two children). We write $n \in T$ to mean $n \in V$. We say that two trees T_1 and T_2 are *isomorphic* if there is a bijection between their node sets preserving children and labels (we simply write it $T_1 = T_2$); they have *same skeleton* if they are isomorphic except for labels.

A *bottom-up nondeterministic tree automaton* on Γ-trees, or *Γ-bNTA*, is a tuple $A = (Q, F, \iota, \delta)$ of a set Q of *states*, a subset $F \subseteq Q$ of *accepting states*, an *initial relation* $\iota : \Gamma \to 2^Q$ giving possible states for leaves from their label, and a *transition relation* $\delta : Q^2 \times \Gamma \to 2^Q$ determining possible states for internal nodes from their label and the states of their children. A *run* of A on a Γ-tree $T = (V, L, R, \lambda)$ is a function $\rho : V \to Q$ such that for each leaf n we have $\rho(n) \in \iota(\lambda(n))$, and for every internal node n we have $\rho(n) \in \delta(\rho(L(n)), \rho(R(n)), \lambda(n))$. A run is *accepting* if, for the root n_r of T, $\rho(n_\mathrm{r}) \in F$; and A *accepts* T (written $T \models A$) if there is some accepting run of A on T. Tree automata capture usual query languages on trees, such as MSO [27].

A *Boolean circuit* is a directed acyclic graph $C = (G, W, g_0, \mu)$ where G is a set of *gates*, $W \subseteq G \times G$ is a set of *wires* (edges), $g_0 \in G$ is a distinguished *output gate*, and μ associates each *gate* $g \in G$ with a *type* $\mu(g)$ that can be inp (*input gate*, with no incoming wire in W), $\neg$ (NOT-gate, with exactly one incoming wire in W), $\wedge$ (AND-gate) or $\vee$ (OR-gate). A *valuation* of the *input gates* C_inp of C is a function $\nu : C_\mathsf{inp} \to \{0, 1\}$; it defines inductively a unique *evaluation* $\nu' : C \to \{0, 1\}$ as follows: $\nu'(g)$ is $\nu(g)$ if $g \in C_\mathsf{inp}$ (i.e., $\mu(g) = \mathsf{inp}$); it is $\neg\nu'(g')$ if $\mu(g) = \neg$ (with $(g', g) \in W$); otherwise it is $\bigodot_{(g',g)\in W} \nu'(g')$ where $\odot$ is $\mu(g)$ (hence, $\wedge$ or $\vee$). Note that this implies that AND- and OR-gates with no inputs always evaluate to 1 and 0 respectively. We will abuse notation and use valuations and evaluations interchangeably, and we write $\nu(C)$ to mean $\nu(g_0)$. The *function* captured by C is the one that maps any valuation ν of C_inp to $\nu(C)$.

3 Provenance Circuits for Tree Automata

We start by studying a notion of provenance on trees, defined in an uncertain tree framework. Fixing a finite alphabet Γ throughout this section, we view a Γ-tree T as an *uncertain tree*, where each node carries an unknown Boolean annotation in $\{0, 1\}$, and consider all possible *valuations* that choose an annotation for each node of T, calling $\overline{\Gamma}$ the alphabet of annotated trees:

Definition 3.1. *We write $\overline{\Gamma} := \Gamma \times \{0, 1\}$. For any Γ-tree $T = (V, L, R, \lambda)$ and valuation $\nu : V \to \{0, 1\}$, $\nu(T)$ is the $\overline{\Gamma}$-tree with same skeleton where each node n is given the label $(\lambda(n), \nu(n))$.*

We consider automata on *annotated* trees, namely, $\overline{\Gamma}$-bNTAs, and define their *provenance* on a Γ-tree T as a Boolean function that describes which valuations of T are accepted by the automaton. Intuitively, provenance keeps track of the dependence between Boolean annotations and acceptance or rejection of the tree.

Definition 3.2. *The* provenance *of a $\overline{\Gamma}$-bNTA A on a Γ-tree $T = (V, L, R, \lambda)$ is the function* $\mathrm{Prov}(A, T)$ *mapping any valuation $\nu : V \to \{0, 1\}$ to 1 or 0 depending on whether $\nu(T) \models A$.*

We now define a *provenance circuit* of A on a Γ-tree T as a circuit that captures the provenance of A on T, $\mathrm{Prov}(A, T)$. Formally:

Definition 3.3. *Let A be a $\overline{\Gamma}$-bNTA and $T = (V, L, R, \lambda)$ be a Γ-tree. A* provenance circuit *of A on T is a Boolean circuit C with $C_{\mathsf{inp}} = V$ that captures the function* $\mathrm{Prov}(A, T)$.

An important result is that provenance circuits can be tractably constructed:

Proposition 3.1. *A provenance circuit of a $\overline{\Gamma}$-bNTA A on a Γ-tree T can be constructed in time $O(|A| \cdot |T|)$.*

The proof is by creating one gate in C per state of A per node of T, and writing out in C all possible transitions of A at each node n of T, depending on the input gate that indicates the annotation of n. In fact, we can show that C is treelike for fixed A; we use this in Section 6 to show the tractability of tree automaton evaluation on probabilistic XML trees from $\mathsf{PrXML}^{\mathsf{mux,ind}}$ [19].

It is not hard to see that this construction gives us a way to capture the provenance of any *query* on trees that can be expressed as an automaton, no matter the choice of automaton. A *query* q is any logical sentence on $\overline{\Gamma}$-trees which a $\overline{\Gamma}$-tree T can *satisfy* (written $T \models q$) or *violate* ($T \not\models q$). An automaton A_q *tests* query q if for any $\overline{\Gamma}$-tree T, we have $T \models A_q$ iff $T \models q$. We define $\mathrm{Prov}(q, T)$ for a Γ-tree T as in Definition 3.2, and run circuits for queries as in Definition 3.3. It is immediate that Proposition 3.1 implies:

Proposition 3.2. *For any fixed query q on $\overline{\Gamma}$-trees for which we can compute an automaton A_q that tests it, a provenance circuit of q on a Γ-tree T can be constructed in time $O(|T|)$.*

Note that provenance does not depend on the automaton used to test the query.

4 Provenance on Tree Encodings

We lift the previous results to the setting of *relational instances*.

A *signature* σ is a finite set of *relation names* (e.g., R) with associated *arity* $\mathrm{arity}(R) \geqslant 1$. Fixing a countable domain $\mathcal{D} = \{a_k \mid k \geqslant 0\}$, a *relational instance* I over σ (or σ-instance) is a finite set I of *ground facts* of the form $R(\mathbf{a})$ with $R \in \sigma$, where $\mathbf{a}$ is a tuple of $\mathrm{arity}(R)$ elements of $\mathcal{D}$. The *active domain* $\mathrm{dom}(I) \subseteq \mathcal{D}$ of I is the finite set of elements of $\mathcal{D}$ used in I. Two instances I

and I' are *isomorphic* if there is a bijection φ from $\mathrm{dom}(I)$ to $\mathrm{dom}(I')$ such that $\varphi(I) = I'$. We say that an instance I' is a *subinstance* of I, written $I' \subseteq I$, if it is a subset of the facts of I, which implies $\mathrm{dom}(I') \subseteq \mathrm{dom}(I)$.

A *query* q is a logical formula in (function-free) first- or second-order logic on σ, without free second-order variables; a σ-instance I can *satisfy* it ($I \models q$) or *violate* it ($I \not\models q$). For simplicity, unless stated otherwise, we restrict to *Boolean queries*, that is, queries with no free variables, that are *constant-free*. This limitation is inessential for *data complexity*, namely complexity for a fixed query: we can handle non-Boolean queries by building a provenance circuit for each possible output result (there are polynomially many), and we encode constants by extending the signature with fresh unary predicates for them.

As before, we consider unknown Boolean annotations on the facts of an instance. However, rather than annotating the facts, it is more natural to say that a fact annotated by 1 is kept, and a fact annotated by 0 is deleted. Formally, given an instance σ, a *valuation* ν is a function from the facts of I to $\{0, 1\}$, and we define $\nu(I)$ as the subinstance $\{F \in I \mid \nu(F) = 1\}$ of I. We then define:

Definition 4.1. *The* provenance *of a query q on a σ-instance I is the function* $\mathrm{Prov}(q, I)$ *mapping any valuation $\nu : I \to \{0, 1\}$ to* 1 *or* 0 *depending on whether $\nu(I) \models q$. A* provenance circuit *of q on I is a Boolean circuit C with $C_{\mathsf{inp}} = I$ that captures* $\mathrm{Prov}(q, I)$.

We study provenance for treelike instances (i.e., bounded-treewidth instances), encoding queries to automata on tree encodings. Let us first define this. The *treewidth* $w(I)$ of an instance I is a standard measure [23] of how close I is to a tree: the treewidth of a tree is 1, that of a cycle is 2, and that of a k-clique or k-grid is $k - 1$; further, we have $w(I') \leqslant w(I)$ for any $I' \subseteq I$. It is known [9,12] that for any fixed $k \in \mathbb{N}$, there is a finite alphabet Γ_σ^k such that any σ-instance I of treewidth $\leqslant k$ can be encoded in linear time [4] to a Γ_σ^k-tree T_I, called the *tree encoding*, which can be decoded back to I up to isomorphism (i.e., up to the identity of constants). Each fact in I is encoded in a node for this fact in the tree encoding, where the node label describes the fact.

The point of tree encodings is that queries in *monadic second-order logic*, the extension of first-order logic with second-order quantification on sets, can be encoded to automata which are then evaluated on tree encodings. Formally:

Definition 4.2. *For $k \in \mathbb{N}$, we say that a Γ_σ^k-bNTA A_q^k* tests *a query q for treewidth k if, for any Γ_σ^k-tree T, we have $T \models A_q^k$ iff T decodes to an instance I such that $I \models q$.*

Theorem 4.1 [9]. *For any $k \in \mathbb{N}$, for any MSO query q, one can compute a Γ_σ^k-bNTA A_q^k that tests q for treewidth $\leqslant k$.*

Our results apply to any query language that can be rewritten to tree automata under a bound on instance treewidth. Beyond MSO, this is also the case of *guarded second-order logic* (GSO). GSO extends first-order logic with second-order quantification on arbitrary-arity relations, with a semantic restriction to *guarded tuples*

(already co-occurring in some instance fact); it captures MSO (it has the same expressive power on treelike instances [16]) and many common database query languages, e.g., *frontier-guarded Datalog* [3]. We use GSO in the sequel as our choice of query language that can be rewritten to automata. Combining the result above with the results of the previous section, we claim that provenance for GSO queries on treelike instances can be tractably computed, and that the resulting provenance circuit has treewidth independent on the instance.

Theorem 4.2. *For any fixed $k \in \mathbb{N}$ and GSO query q, for any σ-instance I such that $w(I) \leqslant k$, one can construct a provenance circuit C of q on I in time $O(|I|)$. The treewidth of C only depends on k and q (not on I).*

The proof is by encoding the instance I to its tree encoding T_I in linear time, and compiling the query q to an automaton A_q that tests it, in constant time in the instance. Now, Section 3 worked with $\overline{\Gamma_\sigma^k}$-bNTAs rather than Γ_σ^k-bNTAs, but the difference is inessential: we can easily map any $\overline{\Gamma_\sigma^k}$-tree T to a Γ_σ^k-tree $\epsilon(T)$ where any node label $(\tau, 1)$ is replaced by τ, and any label $(\tau, 0)$ is replaced by a dummy label indicating the absence of a fact; and we straightforwardly translate A to a $\overline{\Gamma_\sigma^k}$-bNTA A' such that $T \models A'$ iff $\epsilon(T) \models A$ for any $\overline{\Gamma_\sigma^k}$-tree T. The key point is then that, for any valuation $\nu : T \to \{0, 1\}$, $\epsilon(\nu(T))$ is a tree encoding of $\nu(I)$ (defined in the expected way), so we conclude by applying Proposition 3.1 to A' and T. As in Section 3, our definition of provenance is intrinsic to the query and does not depend on its formulation, on the choice of tree decomposition, or on the choice of automaton to evaluate the query on tree encodings.

Note that tractability holds only in data complexity. For combined complexity, we incur the cost of compiling the query to an automaton, which is nonelementary in general [21]. However, for some restricted query classes, such as *unions of conjunctive queries* (UCQs), the compilation phase has lower cost.

5 General Semirings

In this section we connect our previous results to the existing definitions of *semiring provenance* on arbitrary relational instances [17]:

Definition 5.1. *A* commutative semiring $(K, \oplus, \otimes, 0_K, 1_K)$ *is a set K with binary operations $\oplus$ and $\otimes$ and distinguished elements 0_K and 1_K, such that $(K, \oplus)$ and $(K, \otimes)$ are commutative monoids with identity element 0_K and 1_K, $\otimes$ distributes over $\oplus$, and $0_K \otimes a = 0_K$ for all $a \in K$.*

Provenance for semiring K is defined on instances where each fact is annotated with an element of K. The provenance of a query on such an instance is an element of K obtained by combining fact annotations following the semantics of the query, intuitively describing how the query output depends on the annotations (see exact definitions in [17]). This general setting has many specific applications:

Example 5.1. For any variable set X, the monotone Boolean functions over X form a semiring $(\mathrm{PosBool}[X], \vee, \wedge, 0, 1)$. On instances where each fact is annotated by its own variable in X, the $\mathrm{PosBool}[X]$-provenance of a query q is a monotone Boolean function on X describing which subinstances satisfy q. As we will see, this is what we defined in Section 4, using circuits as compact representations.

The *natural numbers* $\mathbb{N}$ with the usual $+$ and $\times$ form a semiring. On instances where facts are annotated with an element of $\mathbb{N}$ representing a multiplicity, the provenance of a query describes its number of matches under the bag semantics.

The *tropical semiring* [11] is $(\mathbb{N} \sqcup \{\infty\}, \min, +, \infty, 0)$. Fact annotations are costs, and the tropical provenance of a query is the minimal cost of the facts required to satisfy it, with multiple uses of a fact being charged multiple times.

For any set of variables X, the *polynomial semiring* $\mathbb{N}[X]$ is the semiring of polynomials with variables in X and coefficients in $\mathbb{N}$, with the usual sum and product over polynomials, and with $0, 1 \in \mathbb{N}$.

Semiring provenance does not support negation well [1] and is therefore only defined for *monotone* queries: a query q is *monotone* if, for any instances $I \subseteq I'$, if $I \models q$ then $I' \models q$. Provenance circuits for semiring provenance are *monotone* circuits [11]: they do not feature NOT-gates. We can show that, adapting the constructions of Section 3 to work with a notion of *monotone* bNTAs, Theorem 4.2 applied to monotone queries yields a *monotone* provenance circuit:

Theorem 5.1. *For any fixed $k \in \mathbb{N}$ and monotone GSO query q, for any σ-instance I such that $w(I) \leqslant k$, one can construct in time $O(|I|)$ a monotone provenance circuit of q on I whose treewidth only depends on k and q (not on I).*

Hence, for monotone GSO queries for which [17] defines a notion of semiring provenance (e.g., those that can be encoded to *Datalog*, a recursive query language that subsumes UCQs), our provenance $\mathrm{Prov}(q, I)$ is easily seen to match the provenance of [17], specialized to the semiring $\mathrm{PosBool}[X]$ of monotone Boolean functions. Indeed, both provenances obey the same intrinsic definition: they are the function that maps to 1 exactly the valuations corresponding to subinstances accepted by the query. Hence, we can understand Theorem 5.1 as a tractability result for $\mathrm{PosBool}[X]$-provenance (represented as a circuit) on treelike instances.

Of course, the definitions of [17] go beyond $\mathrm{PosBool}[X]$ and extend to arbitrary commutative semirings. We now turn to this more general question.

$\mathbb{N}[X]$-provenance for UCQs. First, we note that, as shown by [17], the provenance of Datalog queries for *any* semiring K can be computed in the semiring $\mathbb{N}[X]$, on instances where each fact is annotated by its own variable in X. Indeed, the provenance can then be *specialized* to K, and the actual fact annotations in K, once known, can be used to replace the variables in the result, thanks to a *commutation with homomorphisms* property. Hence, *we restrict to $\mathbb{N}[X]$-provenance* and to instances of this form, which covers all the examples above.

Second, in our setting of treelike instances, we evaluate queries using tree automata, which are compiled from logical formulae with no prescribed execution plan. For the semiring $\mathbb{N}[X]$, this is hard to connect to the general definitions of provenance in [17], which are mainly designed for positive relational algebra operators or Datalog queries. Hence, to generalize our constructions to $\mathbb{N}[X]$-provenance, *we now restrict our query language to UCQs*, assuming without loss of generality that they contain no equality atoms, We comment at the end of this section on the difficulties arising for richer query languages.

We formally define the $\mathbb{N}[X]$-*provenance* of UCQs on relational instances by encoding them straightforwardly to Datalog and using the Datalog provenance definition of [17]. The resulting provenance can be rephrased as follows:

Definition 5.2. *The* $\mathbb{N}[X]$-provenance *of a UCQ* $q = \bigvee_{i=1}^{n} \exists \mathbf{x}_i \, q_i(\mathbf{x}_i)$ *(where* q_i *is a conjunction of atoms with free variables* $\mathbf{x}_i$*) on an instance* I *is defined as:*

$$\mathrm{Prov}_{\mathbb{N}[X]}(q, I) := \bigoplus_{i=1}^{n} \bigoplus_{f : \mathbf{x}_i \to \mathrm{dom}(I) \text{ such that } I \models q_i(f(\mathbf{x}_i))} \bigotimes_{A(\mathbf{x}_i) \in q_i} A(f(\mathbf{x}_i)).$$

In other words, we sum over each disjunct, and over each match of the disjunct; for each match, we take the product, over the atoms of the disjunct, of their image fact in I*, identifying each fact to the one variable in* X *that annotates it.*

We know that $\mathrm{Prov}_{\mathbb{N}[X]}(q, I)$ enjoys all the usual properties of provenance: it can be specialized to PosBool$[X]$, yielding back the previous definition; it can be evaluated in the $\mathbb{N}$ semiring to count the number of matches of a query; etc.

Example 5.2. Consider the instance $I = \{F_1 := R(a, a), F_2 := R(b, c), F_3 := R(c, b)\}$ and the CQ $q : \exists xy \, R(x, y)R(y, x)$. We have $\mathrm{Prov}_{\mathbb{N}[X]}(q, I) = F_1^2 + 2F_2F_3$ and $\mathrm{Prov}(q, I) = F_1 \vee (F_2 \wedge F_3)$. Unlike PosBool$[X]$-provenance, $\mathbb{N}[X]$-provenance can describe that multiple atoms of the query map to the same fact, and that the same subinstance is obtained with two different query matches. Evaluating in the semiring $\mathbb{N}$ with facts annotated by 1, q has $1^2 + 2 \times 1 \times 1 = 3$ matches.

Provenance circuits for trees. Guided by this definition of $\mathbb{N}[X]$-provenance, we generalize the construction of Section 3 of provenance on trees to a more expressive provenance construction, before we extend it to treelike instances as in Section 4.

Instead of considering $\overline{\Gamma}$-trees, we consider $\overline{\Gamma}^p$-trees for $p \in \mathbb{N}$, whose label set is $\Gamma \times \{0, \ldots, p\}$ rather than $\Gamma \times \{0, 1\}$. Intuitively, rather than uncertainty about whether facts are present or missing, we represent uncertainty about the *number of available copies* of facts, as UCQ matches may include the same fact multiple times. We impose on $\overline{\Gamma}$ the partial order $<$ defined by $(\tau, i) < (\tau, j)$ for all $\tau \in \Gamma$ and $i < j$ in $\{0, \ldots, p\}$, and call a $\overline{\Gamma}^p$-bNTA $A = (Q, F, \iota, \delta)$ *monotone* if for every $\tau < \tau'$ in $\overline{\Gamma}^p$, we have $\iota(\tau) \subseteq \iota(\tau')$ and $\delta(q_1, q_2, \tau) \subseteq \delta(q_1, q_2, \tau')$ for every $q_1, q_2 \in Q$. We write $\mathrm{Val}^p(T)$ for the set of all *p-valuations* $\nu : V \to \{0, \ldots, p\}$ of a Γ-tree T. We write $|\mathrm{aruns}(A, T)|$ for a $\overline{\Gamma}^p$-tree T and $\overline{\Gamma}^p$-bNTA A to denote the number of accepting runs of A on T. We can now define:

Definition 5.3. *The* $\mathbb{N}[X]$-provenance *of a* $\overline{\Gamma}^p$*-bNTA* A *on a* Γ*-tree* T *is*

$$\mathrm{Prov}_{\mathbb{N}[X]}(A, T) := \bigoplus_{\nu \in \mathrm{Val}^p(T)} |\mathrm{aruns}(A, \nu(T))| \bigotimes_{n \in T} n^{\nu(n)}$$

where each node $n \in T$ *is identified with its own variable in* X*. Intuitively, we*

sum over all valuations ν of T to $\{0, \ldots, p\}$, and take the product of the tree nodes to the power of their valuation in ν, with the number of accepting runs of A on $\nu(T)$ as coefficient; in particular, the term for ν is 0 if A rejects $\nu(T)$.

This definition specializes in PosBool$[X]$ to our earlier definition of $\mathrm{Prov}(A, T)$, but extends it with the two features of $\mathbb{N}[X]$: multiple copies of the same nodes (represented as $n^{\nu(n)}$) and multiple derivations (represented as $|\mathrm{aruns}(A, \nu(T))|$). To construct this general provenance, we need *arithmetic circuits*:

Definition 5.4. *A* K-circuit *for semiring $(K, \oplus, \otimes, 0_K, 1_K)$ is a circuit with $\oplus$- and $\otimes$-gates instead of OR- and AND-gates (and no analogue of NOT-gates), whose input gates stand for elements of K. As before, the constants 0_K and 1_K can be written as $\oplus$- and $\otimes$-gates with no inputs. The element of K* captured *by a K-circuit is the element captured by its distinguished gate, under the recursive definition that $\oplus$- and $\otimes$-gates capture the sum and product of the elements captured by their operands, and input gates capture their own value.*

We now show an efficient construction for such provenance circuits, generalizing the monotone analogue of Proposition 3.1. The proof technique is to replace AND- and OR-gates by $\otimes$- and $\oplus$-gates, and to consider possible annotations in $\{0, \ldots, p\}$ instead of $\{0, 1\}$. The correctness is proved by induction via a general identity relating the provenance on a tree to that of its left and right subtrees.

Theorem 5.2. *For any fixed $p \in \mathbb{N}$, for a $\overline{\Gamma}^p$-bNTA A and a Γ-tree T, a $\mathbb{N}[X]$-circuit capturing $\mathrm{Prov}_{\mathbb{N}[X]}(A, T)$ can be constructed in time $O(|A| \cdot |T|)$.*

Provenance circuit for instances. Moving back to provenance for UCQs on bounded-treewidth instances, we obtain a linear-time provenance construction:

Theorem 5.3. *For any fixed $k \in \mathbb{N}$ and UCQ q, for any σ-instance I such that $w(I) \leqslant k$, one can construct a $\mathbb{N}[X]$-circuit that captures $\mathrm{Prov}_{\mathbb{N}[X]}(q, I)$ in time $O(|I|)$.*

The proof technique is to construct for each disjunct q' of q a $\overline{\Gamma}^p$-bNTA $A_{q'}$, where $\Gamma := \Gamma_\sigma^k$ is the alphabet for tree encodings of width k, and p is the maximum number of atoms in a disjunct of q. We want $A_{q'}$ to test q' on tree encodings over Γ, *while preserving multiplicities*: this is done by enumerating all possible self-homomorphisms of q', changing σ to make the multiplicity of atoms part of the relation name, encoding the resulting queries to automata as usual [9] and going back to the original σ. We then apply a variant of Theorem 5.2 to construct a $\mathbb{N}[X]$-circuit capturing the provenance of $A_{q'}$ on a tree encoding of I but for valuations that sum to the number of atoms of q'; this restricts to bag-subinstances corresponding exactly to matches of q'. We obtain a $\mathbb{N}[X]$-circuit that captures $\mathrm{Prov}_{\mathbb{N}[X]}(q, I)$ by combining the circuits for each disjunct, the distinguished gate of the overall circuit being a $\oplus$-gate of that of each circuit.

Remember that an $\mathbb{N}[X]$-circuit can then be specialized to a circuit for an arbitrary semiring (in particular, if the semiring has no variable, the circuit can be used for evaluation); thus, this provides provenance for q on I for any semiring.

Going beyond UCQs. To compute $\mathbb{N}[X]$-provenance beyond UCQs (e.g., for monotone GSO queries or their intersection with Datalog), the main issue is fact multiplicity: multiple uses of facts are easy to describe for UCQs (Definition 5.2), but for more expressive languages we do not know how to define them and connect them to automata.

In fact, we can build a query P, in guarded Datalog [15], such that the smallest number of occurrences of a fact in a derivation tree for P cannot be bounded independently from the instance. We thus cannot rewrite P to a fixed finite bNTA testing multiplicities on all input instances. However, as guarded Datalog is monotone and GSO-expressible, we can compute the PosBool$[X]$-provenance of P with Theorem 4.2, hinting at a difference between PosBool$[X]$ and $\mathbb{N}[X]$-provenance computation for queries beyond UCQs.

6 Applications

In Section 5 we have shown a $\mathbb{N}[X]$-provenance circuit construction for UCQs on treelike instances. This construction can be specialized to any provenance semiring, yielding various applications: counting query results by evaluating in $\mathbb{N}$, computing the cost of a query in the tropical semiring, etc. By contrast, Section 4 presented a provenance construction for arbitrary GSO queries, but only for a Boolean representation of provenance, which does not capture multiplicities of facts or derivations. The results of both sections are thus incomparable. In this section we show applications of our constructions to two important problems: *probability evaluation*, determining the probability that a query holds on an uncertain instance, and *counting*, counting the number of answers to a given query. These results are consequences of the construction of Section 4.

Probabilistic XML. We start with the problem of probabilistic query evaluation, beginning with the setting of *trees*. We use the framework of *probabilistic XML*, denoted $\mathsf{PrXML}^{\mathsf{fie}}$, to represent probabilistic trees as trees annotated by propositional formulas over independent probabilistic events (see [19] for the formal definitions), and consider the *data complexity* of the *query evaluation* problem for a MSO query q on such trees (i.e., computing the probability that q holds).

This problem is intractable in general, which is not surprising: it is harder than determining the probability of a single propositional annotation. However, for the less expressive *local* PrXML model, $\mathsf{PrXML}^{\mathsf{mux,ind}}$, query evaluation has tractable data complexity [8]; this model restricts edges to be annotated by only one event literal that is only used on that edge (plus a form of mutual exclusivity).

We can use the provenance circuits of Section 4 to justify that query evaluation is tractable for $\mathsf{PrXML}^{\mathsf{mux,ind}}$ and capture the data complexity tractability result of [8]. We say that an algorithm runs in *ra-linear time* if it runs in linear time assuming that arithmetic operations over rational numbers take constant time and rationals are stored in constant space, and runs in polynomial time without this assumption. We can show:

Theorem 6.1 [8]. *MSO query evaluation on* $\mathsf{PrXML}^{\mathsf{mux,ind}}$ *has ra-linear data complexity.*

We can also show extensions of this result. For instance, on $\mathsf{PrXML}^{\mathsf{fie}}$, defining the *scope* of event e in a document D as the smallest subtree in the left-child-right-sibling encoding of D covering nodes whose parent edge mentions e, and the *scope size* of a node n as the number of events with n in their scope, we show:

Proposition 6.1. *For any fixed* $k \in \mathbb{N}$*, MSO query evaluation on* $\mathsf{PrXML}^{\mathsf{fie}}$ *documents with scopes assumed to have size* $\leqslant k$ *has ra-linear data complexity.*

BID instances. We move from trees to relational instances, and show another bounded-width tractability result for *block-independent disjoint* (BID) instances (see [26]). We define the *treewidth* of a BID instance as that of its underlying relational instance, and claim the following (remember that query evaluation on a probabilistic instance means determining the probability that the query holds):

Theorem 6.2. *For any fixed* $k \in \mathbb{N}$*, MSO query evaluation on an input BID instance of treewidth* $\leqslant k$ *has ra-linear data complexity.*

All probabilistic results are proven by rewriting to a formalism of relational instances with a circuit annotation, such that instance and circuit have a bounded-width joint decomposition. We compute a treelike provenance circuit for the instance using Theorem 4.2, combine it with the annotation circuit, and apply existing message passing techniques [20] to compute the probability of the circuit.

Counting. We turn to the problem of counting query results, and reduce it in ra-linear time to query evaluation on treelike instances, capturing a result of [2]:

Theorem 6.3 [2]. *For any fixed MSO query* $q(\mathbf{x})$ *with free first-order variables and* $k \in \mathbb{N}$*, the number of matching assignments to* $\mathbf{x}$ *on an input instance* I *of width* $\leqslant k$ *can be computed in ra-linear data complexity.*

7 Related Work

From the original results [9,12] on the linear-time data complexity of MSO evaluation on treelike structures, works such as [2] have investigated counting problems, including applications to probability computation (on graphs). A recent paper [5] also shows the linear-time data complexity of evaluating an MSO query on a treelike probabilistic network (analogous to a circuit). Such works, however, do not decouple the computation of a treelike *provenance* of the query and the *application* of probabilistic inference on this provenance, as we do. We also note results from another approach [22] on treelike structures, based on monadic Datalog (and not on MSO as the other works), that are limited to counting.

The *intensional* approach [26] to query evaluation on probabilistic databases is to compute a lineage of the query and evaluate its probability via general purpose methods; tree-like lineages allow for tractable probabilistic query evaluation

[18]. Many works in this field provide sufficient conditions for lineage tractability, only a few based on the data [24,25] but most based on the query [10,18]. For treelike instances, as we show, we can *always* compute treelike lineages, and we can do so for expressive queries (beyond UCQs considered in these works), or alternatively generalize Boolean lineages to connect them to more expressive semirings.

Our provenance study is inspired by the usual definitions of semiring provenance for the relational algebra and Datalog [17]. Another notion of provenance, for XQuery queries on trees, has been introduced in [13]. Both [17] and [13] provide *operational* definitions of provenance, which cannot be directly connected to tree automata. A different relevant work on provenance is [11], which introduces provenance circuits, but uses them for Datalog and only on *absorptive* semirings. Last, other works study provenance for *transducers* [6], but with no clear connections to semiring provenance or provenance for Boolean queries.

8 Conclusion

We have shown that two provenance constructions can be computed in linear time on trees and treelike instances: one for UCQs on arbitrary semirings, the other for arbitrary GSO queries as non-monotone Boolean expressions. A drawback of our results is their high combined complexity, as they rely on non-elementary encoding of the query to an automaton. One approach to fix this is monadic Datalog [14,22]; this requires defining and computing provenance in this setting.

Acknowledgments. This work was partly supported by a financial contribution from the Fondation Campus Paris-Saclay and the French ANR Aggreg project.

References

1. Amsterdamer, Y., Deutch, D., Tannen, V.: On the limitations of provenance for queries with difference. In: TaPP (2011)
2. Arnborg, S., Lagergren, J., Seese, D.: Easy problems for tree-decomposable graphs. J. Algorithms **12**(2) (1991)
3. Baget, J., Leclère, M., Mugnier, M.: Walking the decidability line for rules with existential variables. In: KR (2010)
4. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput. **25**(6) (1996)
5. Bodlaender, H.L.: Probabilistic inference and monadic second order logic. In: Baeten, J.C.M., Ball, T., de Boer, F.S. (eds.) TCS 2012. LNCS, vol. 7604, pp. 43–56. Springer, Heidelberg (2012)
6. Bojańczyk, M.: Transducers with origin information. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014, Part II. LNCS, vol. 8573, pp. 26–37. Springer, Heidelberg (2014)
7. Cheney, J., Chiticariu, L., Tan, W.C.: Provenance in databases: Why, how, and where. Foundations and Trends in Databases **1**(4) (2009)

8. Cohen, S., Kimelfeld, B., Sagiv, Y.: Running tree automata on probabilistic XML. In: PODS (2009)
9. Courcelle, B.: The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. Inf. Comput. **85**(1) (1990)
10. Dalvi, N., Suciu, D.: Efficient query evaluation on probabilistic databases. VLDBJ **16**(4) (2007)
11. Deutch, D., Milo, T., Roy, S., Tannen, V.: Circuits for Datalog provenance. In: ICDT (2014)
12. Flum, J., Frick, M., Grohe, M.: Query evaluation via tree-decompositions. J. ACM **49**(6) (2002)
13. Foster, J.N., Green, T.J., Tannen, V.: Annotated XML: queries and provenance. In: PODS (2008)
14. Gottlob, G., Pichler, R., Wei, F.: Monadic Datalog over finite structures of bounded treewidth. TOCL **12**(1) (2010)
15. Grädel, E.: Efficient evaluation methods for guarded logics and Datalog LITE. In: LPAR (2000)
16. Grädel, E., Hirsch, C., Otto, M.: Back and forth between guarded and modal logics. TOCL **3**(3) (2002)
17. Green, T.J., Karvounarakis, G., Tannen, V.: Provenance semirings. In: PODS (2007)
18. Jha, A.K., Suciu, D.: On the tractability of query compilation and bounded treewidth. In: ICDT (2012)
19. Kimelfeld, B., Senellart, P.: Probabilistic XML: models and complexity. In: Ma, Z., Yan, L. (eds.) Advances in Probabilistic Databases. STUDFUZZ, vol. 304, pp. 39–66. Springer, Heidelberg (2013)
20. Lauritzen, S.L., Spiegelhalter, D.J.: Local computations with probabilities on graphical structures and their application to expert systems. J. Royal Statistical Society, Series B (1988)
21. Meyer, A.R.: Weak monadic second order theory of succesor is not elementary-recursive. In: Logic Colloquium (1975)
22. Pichler, R., Rümmele, S., Woltran, S.: Counting and enumeration problems with bounded treewidth. Artificial Intelligence, and Reasoning, In Logic for Programming (2010)
23. Robertson, N., Seymour, P.D.: Graph minors. II. Algorithmic aspects of tree-width. J. Algorithms **7**(3) (1986)
24. Roy, S., Perduca, V., Tannen, V.: Faster query answering in probabilistic databases using read-once functions. In: ICDT (2011)
25. Sen, P., Deshpande, A., Getoor, L.: Read-once functions and query evaluation in probabilistic databases. PVLDB **3**(1–2) (2010)
26. Suciu, D., Olteanu, D., Ré, C., Koch, C.: Probabilistic Databases. Morgan & Claypool (2011)
27. Thatcher, J.W., Wright, J.B.: Generalized finite automata theory with an application to a decision problem of second-order logic. Math. systems theory **2**(1) (1968)