Provenance Circuits for Trees and Treelike Instances (Extended Version)

Antoine Amarilli
Télécom ParisTech; Institut Mines–Télécom; CNRS LTCI
antoine.amarilli@telecom-paristech.fr

Pierre Bourhis
CNRS CRIStAL; Université Lille 1; INRIA Lille
pierre.bourhis@univ-lille1.fr

Pierre Senellart
Institut Mines-Télécom; Télécom ParisTech; CNRS LTCI
National University of Singapore; CNRS IPAL
pierre.senellart@telecom-paristech.fr

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 [3] or performing query evaluation on probabilistic trees [10]. 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 [20], 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 [11] has shown that evaluating a fixed monadic second-order (MSO) query on relational instances, while generally hard in the input instance for any level

of the polynomial hierarchy [1], 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 [17]. In addition to query evaluation, it is also possible to *count* in linear time the number of query answers over treelike instances [3, 29].

However, query evaluation and counting are special cases of the more general problem of capturing *provenance information* [9,20] of query results, which describes the link between input and output tuples. Provenance information can be expressed through various formalisms, such as *provenance semirings* [20] or Boolean formulae [34]. Besides counting, provenance can be exploited for practically important tasks such as answering queries in incomplete databases [21], maintaining access rights [28], or computing query probability [34]. To our knowledge, 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* [13] 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* [20]. This poses several challenges. First, as semirings cannot deal satisfactorily with negation [2, 16], 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 [20] 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 [10], 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 [34]. These tractability results for provenance are achieved by applying message passing [25] on our provenance circuits. Last, we

show the tractability of counting query matches, using a reduction to the probabilistic setting, capturing a result of [3].

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 [35] and tree-pattern queries [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 $\mathsf{PrXML}^{\mathsf{mux,ind}}$ [23].

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*[1] 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.

[1] Subinstances are not necessarily "induced" by a subset of the domain, they can be arbitrary subsets of facts.

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* $\mathrm{w}(I)$ of an instance I is a standard measure [31] 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 $\mathrm{w}(I') \leqslant \mathrm{w}(I)$ for any $I' \subseteq I$. It is known [11,14] that for any fixed $k \in \mathbb{N}$, there is a finite alphabet Γ^k_σ such that any σ-instance I of treewidth $\leqslant k$ can be encoded in linear time [6] to a Γ^k_σ-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^k_σ-bNTA A^k_q* tests *a query q for treewidth k if, for any Γ^k_σ-tree T, we have $T \models A^k_q$ iff T decodes to an instance I such that $I \models q$.*

Theorem 4.1 [11]. *For any $k \in \mathbb{N}$, for any MSO query q, one can compute a Γ^k_σ-bNTA A^k_q 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 [19]) and many common database query languages, e.g., *guarded Datalog* [18] or *frontier-guarded Datalog* [4]. 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 of the instance.

Theorem 4.2. *For any fixed $k \in \mathbb{N}$ and GSO query q, for any σ-instance I such that $\mathrm{w}(I) \leqslant k$, 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 $\overline{\Gamma^k_\sigma}$-bNTAs rather than Γ^k_σ-bNTAs, but the difference is inessential: we can easily map any $\overline{\Gamma^k_\sigma}$-tree T to a Γ^k_σ-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 $\overline{\Gamma^k_\sigma}$-bNTA A' such that $T \models A'$ iff $\epsilon(T) \models A$ for any $\overline{\Gamma^k_\sigma}$-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 [26]. 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 [20]:

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 [20]). This general setting has many specific applications:

Example 5.1. *For any variable set* X*, the monotone Boolean functions over* X *form a semiring* $(\mathrm{PosBool}[X], \vee, \wedge, 0, 1)$*. We write them as propositional formulae, but two equivalent formulae (e.g.,* $x \vee x$ *and* x*) denote the same* $\mathrm{PosBool}[X]$ *object. 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 security semiring *[15]* $\mathbb{S}$ *is defined on the ordered set* $1_{\mathbb{S}} < C < S < T < 0_{\mathbb{S}}$ *(respectively:* always available*,* confidential*,* secret*,* top secret*,* never available*) as* $(\{1, C, S, T, 0\}, \min, \max, 0, 1)$*. The provenance of a query for* $\mathbb{S}$ *denotes the minimal level of security clearance required to see that it holds. The* fuzzy semiring *[2] is* $([0, 1], \max, \min, 0, 1)$*. The provenance of a query for this semiring is the minimal fuzziness value that has to be tolerated for facts so that the query is satisfied.*

The tropical semiring *[13] 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 [2, 16] 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 [13]: 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 $\mathrm{w}(I) \leqslant k$, 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 [20] 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 [20], specialized to the semiring 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 PosBool$[X]$-provenance (represented as a circuit) on treelike instances.

Of course, the definitions of [20] go beyond 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 [20], 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 [20], 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 [20]. 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 $\mathrm{PosBool}[X]$ 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* $\mathrm{w}(I) \leqslant k$*, 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 [11] 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 [18], 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 [23] 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 [10]; 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 [10]. 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 [10]. *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 [34], or [5, 30] for formal definitions). 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.*

This implies the same claim for tuple-independent databases [12, 24].

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 [25] 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 [3]:

Theorem 6.3 [3]. *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

Bounded treewidth. From the original results [11, 14] on the linear-time data complexity of MSO evaluation on treelike structures, works such as [3] have investigated counting problems, including applications to probability computation (on graphs). A recent paper [7] 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 [29] on treelike structures, based on monadic Datalog (and not on MSO as the other works), that are limited to counting.

Probabilistic databases. The *intensional* approach [34] 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 [22]. Many works in this field provide sufficient conditions for lineage tractability, only a few based on the data [32, 33] but most based on the query [12, 22]. 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.

Provenance. Our provenance study is inspired by the usual definitions of semiring provenance for the relational algebra and Datalog [20]. Another notion of provenance, for XQuery queries on trees, has been introduced in [15]. Both [20] and [15] provide *operational* definitions of provenance, which cannot be directly connected to tree automata. A different relevant work on provenance is [13], which introduces provenance circuits, but uses them for Datalog and only on *absorptive* semirings. Last, other works study provenance for *transducers* [8], 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 [17,29]; this requires defining and computing provenance in this setting.

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

References

[1] M. Ajtai, R. Fagin, and L. J. Stockmeyer. The closure of monadic NP (extended abstract). In *STOC*, 1998.

[2] Y. Amsterdamer, D. Deutch, and V. Tannen. On the limitations of provenance for queries with difference. In *TaPP*, 2011.

[3] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2), 1991.

[4] J. Baget, M. Leclère, and M. Mugnier. Walking the decidability line for rules with existential variables. In *KR*, 2010.

[5] D. Barbará, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *TKDE*, 4(5), 1992.

[6] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6), 1996.

[7] M. H. L. Bodlaender. Probabilistic inference and monadic second order logic. In *IFIP TCS*, 2012.

[8] M. Bojańczyk. Transducers with origin information. In *ICALP*, 2014.

[9] J. Cheney, L. Chiticariu, and W. C. Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4), 2009.

[10] S. Cohen, B. Kimelfeld, and Y. Sagiv. Running tree automata on probabilistic XML. In *PODS*, 2009.

[11] B. Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inf. Comput.*, 85(1), 1990.

[12] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDBJ*, 16(4), 2007.

[13] D. Deutch, T. Milo, S. Roy, and V. Tannen. Circuits for Datalog provenance. In *ICDT*, 2014.

[14] J. Flum, M. Frick, and M. Grohe. Query evaluation via tree-decompositions. *J. ACM*, 49(6), 2002.

[15] J. N. Foster, T. J. Green, and V. Tannen. Annotated XML: queries and provenance. In *PODS*, 2008.

[16] F. Geerts and A. Poggi. On database query languages for k-relations. *J. App. Log.*, 8(2), 2010.

[17] G. Gottlob, R. Pichler, and F. Wei. Monadic Datalog over finite structures of bounded treewidth. *TOCL*, 12(1), 2010.

[18] E. Grädel. Efficient evaluation methods for guarded logics and Datalog LITE. In *LPAR*, 2000.

[19] E. Grädel, C. Hirsch, and M. Otto. Back and forth between guarded and modal logics. *TOCL*, 3(3), 2002.

[20] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, 2007.

[21] T. Imielinski and W. Lipski, Jr. Incomplete information in relational databases. *J. ACM*, 31(4), 1984.

[22] A. K. Jha and D. Suciu. On the tractability of query compilation and bounded treewidth. In *ICDT*, 2012.

[23] B. Kimelfeld and P. Senellart. Probabilistic XML: Models and complexity. In Z. Ma and L. Yan, editors, *Advances in Probabilistic Databases for Uncertain Information Management*. Springer, 2013.

[24] L. V. S. Lakshmanan, N. Leone, R. B. Ross, and V. S. Subrahmanian. ProbView: A flexible probabilistic database system. *TODS*, 22(3), 1997.

[25] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *J. Royal Statistical Society. Series B*, 1988.

[26] A. R. Meyer. Weak monadic second order theory of succesor is not elementary-recursive. In *Logic Colloquium*, 1975.

[27] F. Neven. Automata, logic, and XML. In *CSL*, 2002.

[28] J. Park, D. Nguyen, and R. S. Sandhu. A provenance-based access control model. In *PST*, 2012.

[29] R. Pichler, S. Rümmele, and S. Woltran. Counting and enumeration problems with bounded treewidth. In *Logic for Programming, Artificial Intelligence, and Reasoning*, 2010.

[30] C. Ré and D. Suciu. Materialized views in probabilistic databases: for information exchange and query optimization. In *VLDB*, 2007.

[31] N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7(3), 1986.

[32] S. Roy, V. Perduca, and V. Tannen. Faster query answering in probabilistic databases using read-once functions. In *ICDT*, 2011.

[33] P. Sen, A. Deshpande, and L. Getoor. Read-once functions and query evaluation in probabilistic databases. *PVLDB*, 3(1-2), 2010.

[34] D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases*. Morgan & Claypool, 2011.

[35] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Math. systems theory*, 2(1), 1968.

A. Proofs for Section 3 (Provenance Circuits for Tree Automata)

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

Throughout the appendix, we will call *0-gates* (resp. *1-gates*) OR-gates (resp. AND-gates) with no inputs; they always evaluate to 0 (resp. to 1).

We first prove this result in the specific case of a *monotone* $\overline{\Gamma}$-bNTA A (see Definition C.4), showing the same result but for provenance circuits which are taken to be *monotone* Boolean circuits (i.e., they do not feature NOT-gates).

Proposition A.1. *A monotone provenance circuit C of a monotone $\overline{\Gamma}$-bNTA A on a Γ-tree T can be constructed in time $O(|A| \cdot |T|)$.*

Proof. Fix $T = (V, L, R, \lambda)$, $A = (Q, F, \iota, \delta)$, and construct the provenance circuit $C = (G, W, g_0, \mu)$. For each node n of T, create one input gate g_n^{i} in C (which we identify to n, so that we have $C_{\mathsf{inp}} = V$), and create one gate g_n^q for every $q \in Q$. If n is a leaf node, for $q \in Q$, set g_n^q to be:

- if $q \in \iota(\lambda(n), 0)$, a 1-gate;
- if $q \in \iota(\lambda(n), 1)$ but $q \notin \iota(\lambda(n), 0)$, an OR-gate with sole input g_n^{i};
- if $q \notin \iota(\lambda(n), 1)$, a 0-gate.

If n is an internal node, create gates $g_n^{q_\mathrm{L}, q_\mathrm{R}}$ and $g_n^{q_\mathrm{L}, q_\mathrm{R}, \mathsf{i}}$ for every pair $q_\mathrm{L}, q_\mathrm{R} \in Q$ (that appears as input states of a transition of δ), the first one being an AND-gate of $g_{L(n)}^{q_\mathrm{L}}$ and $g_{R(n)}^{q_\mathrm{R}}$, the second one being an AND-gate of $g_n^{q_\mathrm{L}, q_\mathrm{R}}$ and of g_n^{i}. Now, for $q \in Q$, set g_n^q to be an OR-gate of all the $g_n^{q_\mathrm{L}, q_\mathrm{R}}$ such that $q \in \delta(q_\mathrm{L}, q_\mathrm{R}, (\lambda(n), 0))$ and of all the $g_n^{q_\mathrm{L}, q_\mathrm{R}, \mathsf{i}}$ such that $q \in \delta(q_\mathrm{L}, q_\mathrm{R}, (\lambda(n), 1))$.

Add gate g_0 to be an OR-gate of all the g_r^q such that $q \in F$, where r is the root of T.

This construction is in time $O(|A| \cdot |T|)$: more precisely, for every node of the tree T, we create a number of states that is linear in the number of states in Q and in the number of transitions of δ.

Now we show that C is indeed a provenance circuit of A on T. Let $\nu : V \to \{0, 1\}$ be a valuation that we extend to an evaluation of C. We show by induction on $n \in T$ that for any $q \in Q$, we have $\nu(g_n^q) = 1$ iff, letting T_n be the subtree of T rooted at n, there is a run ρ of A on T_n such that $\rho(n) = q$.

For a leaf node n, choosing $q \in Q$, if $\nu(n) = 0$ then $\nu(g_n^q) = 1$ iff $q \in \iota(\lambda(n), 0)$, and if $\nu(n) = 1$ then $\nu(g_n^q) = 1$ iff $q \in \iota(\lambda(n), 1)$, so in both cases we can define a run ρ as $\rho(n) := q$. Conversely, the existence of a run clearly ensures that $\nu(g_n^q) = 1$.

For an internal node n, choosing $q \in Q$, if $\nu(n) = 0$ then $\nu(g_n^q) = 1$ iff there are some $q_\mathrm{L}, q_\mathrm{R} \in Q$ such that $q \in \delta(q_\mathrm{L}, q_\mathrm{R}, (\lambda(n), 0))$, $\nu(g_{\mathrm{L}(n)}^{q_\mathrm{L}}) = 1$, and $\nu(g_{\mathrm{R}(n)}^{q_\mathrm{R}}) = 1$. By induction hypothesis this implies the existence of a run ρ_L of A on $T_{L(n)}$ such that $\rho_\mathrm{L}(\mathrm{L}(n)) = q_\mathrm{L}$ and a run ρ_R of A on $T_{R(n)}$ such that $\rho_\mathrm{R}(\mathrm{R}(n)) = q_\mathrm{R}$, from which we construct a run ρ of A on T_n such that $\rho(n) = q$, by setting $\rho(n) := q$ and setting $\rho(n')$ either to $\rho_\mathrm{L}(n')$ or to $\rho_\mathrm{R}(n')$ depending on whether $n' \in T_{L(n)}$ or $n' \in T_{R(n)}$. Conversely, the existence of such a run ρ implies the existence of two such runs ρ_L and ρ_R, from which we deduce that $\nu(g_n^q) = 1$.

If $\nu(n) = 1$ then $\nu(g_n^q) = 1$ iff there are some $q_\mathrm{L}, q_\mathrm{R} \in Q$ such that $\nu(g_{\mathrm{L}(n)}^{q_\mathrm{L}}) = 1$, $\nu(g_{\mathrm{R}(n)}^{q_\mathrm{R}}) = 1$, and either $q \in \delta(q_\mathrm{L}, q_\mathrm{R}, (\lambda(n), 0))$ or $q \in \delta(q_\mathrm{L}, q_\mathrm{R}, (\lambda(n), 1))$. By monotonicity of A, this is equivalent to $q \in \delta(q_\mathrm{L}, q_\mathrm{R}, (\lambda(n), 1))$. The rest is analogous to the previous case.

The claim proven by induction clearly justifies that C is a provenance circuit, as, applying it to the root of T, we deduce that, for any valuation ν, we have $\nu(C) = 1$ iff there is an accepting run of A on $\nu(T)$. □

We now generalize this result to automata and provenance circuits which are not necessarily monotone.

Proof. We adapt the construction of Proposition A.1. The only difference is that we add, for every node $n \in T$, a gate $g_n^{\neg \mathrm{i}}$ which is a NOT-gate of g^{i}, and we modify the definition of the following nodes:

- for leaf nodes n, for any state q, we set g_n^q to be an OR-gate of g^{i} if $q \in \iota(\lambda(n), 1)$ (and a 0-gate otherwise), and $g^{\neg \mathrm{i}}$ if $q \in \iota(\lambda(n), 0)$ (and a 0-gate otherwise).
- for internal nodes n, for every pair $q_{\mathrm{L}}, q_{\mathrm{R}} \in Q$ that appears as input states of a transition of δ, create a gate $g_n^{q_{\mathrm{L}}, q_{\mathrm{R}}, \neg \mathrm{i}}$ which is an AND-gate of $g_n^{q_{\mathrm{L}}, q_{\mathrm{R}}}$ and of $g_n^{\neg \mathrm{i}}$. Now, for any state q, we set g_n^q as before except that we use $g_n^{q_{\mathrm{L}}, q_{\mathrm{R}}}$ instead of $g_n^{q_{\mathrm{L}}, q_{\mathrm{R}}, \neg \mathrm{i}}$.

We show correctness as before, showing by induction on $n \in T$ that for any $q \in Q$, $\nu(g_n^q) = 1$ iff A_q accepts T_n, where A_q is obtained from A by letting q be the only final state. The property is clearly true on leaf nodes, and at internal nodes, if $\nu(n) = 0$ we have $\nu(g_n^q) = 1$ iff there exist $q_{\mathrm{L}}, q_{\mathrm{R}} \in Q$ such that $q \in \delta(q_{\mathrm{L}}, q_{\mathrm{R}}, (\lambda(n), 0))$, $\nu(g_{\mathrm{L}(n)}^{q_{\mathrm{L}}}) = 1$, which by induction hypothesis implies the existence of sub-runs on $T_{L(n)}$ and $T_{R(n)}$ that we combine as before. If $\nu(n) = 0$ we have $\nu(g_n^q) = 1$ iff there exist $q_{\mathrm{L}}, q_{\mathrm{R}} \in Q$ such that $q \in \delta(q_{\mathrm{L}}, q_{\mathrm{R}}, (\lambda(n), 1))$ (this time we cannot have $q \in \delta(q_{\mathrm{L}}, q_{\mathrm{R}}, (\lambda(n), 0)))$ so we conclude in the same way. We conclude by justifying that g_0 is correctly defined, as before. □

B. Proofs for Section 4 (Provenance on Tree Encodings)

B.1. Formal preliminaries

We start by giving the omitted formal definitions:

Definition B.1. *A* tree decomposition *of an instance I is a $\mathcal{T}$-tree $T = (B, L, R, \mathrm{dom})$ where $\mathcal{T}$ is the set of subsets of* $\mathrm{dom}(I)$*. The nodes of T are called* bags *and their label is written* $\mathrm{dom}(b)$*. We require:*

1. *for every $a \in \mathrm{dom}(I)$, letting $B_a := \{b \in B \mid a \in \mathrm{dom}(b)\}$, for every two bags $b_1, b_2 \in B_a$, all bags on the (unique) undirected path from b_1 to b_2 are also in B_a;*
2. *for every fact $R(\mathbf{a})$ of I, there exists a bag $b_{\mathbf{a}} \in B$ such that $\mathbf{a} \subseteq \mathrm{dom}(b_{\mathbf{a}})$.*

The width *of T is $\mathrm{w}(T) := k - 1$ where $k := \max_{b \in T} |\mathrm{dom}(b)|$. The* treewidth *(or* width*) of an instance I, written $\mathrm{w}(I)$, is the minimal width $\mathrm{w}(T)$ of a tree decomposition T of I.*

It is NP-hard, given an instance I, to determine $\mathrm{w}(I)$. However, given a fixed width k, one can compute in linear time in I a tree decomposition of width $\leqslant k$ of I if one exists [Bod96].

To represent bounded-treewidth instances as trees on a finite alphabet, we introduce the notion of *tree encodings*. The representation is up to isomorphism, i.e., it loses the identity of constants. Our finite alphabet Γ_σ^k is the set of possible facts on an instance of width fixed

to k; following the definition of proof trees in [CV92] we use element co-occurrences between one node and its parent in the tree as a way to encode element reuse. Formally, we take Γ^k_σ to be the set defined as follows:

Definition B.2. *The set of* k-facts *of the signature* σ*, written* Γ^k_σ*, is the set of pairs* $\tau = (d, s)$ *where:*

- *the* domain d *is a subset of size at most* $k+1$ *of the first* $2k+2$ *elements of the countable domain* $\mathcal{D}$*,* $a_1, \ldots, a_{2k+2}$*;*
- *the* structure s *is a zero- or single-fact structure over* σ *such that* $\mathrm{dom}(s) \subseteq d$*.*

A tree encoding is just a Γ^k_σ-tree. We first explain how such a tree encoding E can be decoded to a structure $I = \langle E \rangle$ (defined up to isomorphism) and a tree decomposition T of width k of I. Process E top-down. At each (d, s)-labeled node of E that is child of a (d', s')-labeled node, pick fresh elements in $\mathcal{D}$ for the elements of $d \backslash d'$ (at the root, pick all fresh elements), add the fact of s to I (replacing the elements in d by the fresh elements, and by the old elements of $\mathrm{dom}(I)$ for $d \cap d'$), and add a bag to T with the elements of I matching those in d. If we ever attempt to create a fact that already exists, we abort and set $\langle E \rangle := \bot$ (we say that E is *invalid*).

We can now define tree encodings in terms of decoding:

Definition B.3. *We call a* Γ^k_σ*-tree* T *a* tree encoding *of width* k *of a* σ*-structure* I *if* $\langle T \rangle$ *is isomorphic to* I*.*

We note that clearly if a structure I has a tree encoding of width k, then $\mathrm{w}(I) \leqslant k$. Further, observe that there is a clear injective function from $\langle T \rangle$ to T, which maps each fact of $\langle T \rangle$ to the node of T which encoded this fact. This function is not total, because some nodes in T contain no fact (their s is a zero-fact structure).

We now justify that one can efficiently compute a tree encoding of width k of I from a tree decomposition of width k of I (this result is implicit in [CV92]).

Lemma B.1. *From a tree decomposition* T *of width* k *of a* σ*-structure* I*, one can compute in linear time a tree encoding* E *of width* k *of* I *with a bijection from the facts of* I *to the non-empty nodes of* E*.*

Proof. The intuition is that we assign each fact $R(\mathbf{a})$ of I to a bag $b \in T$ such that $\mathbf{a} \subseteq \mathrm{dom}(b)$, which can be done in linear time [FFG02]. We then encode each node of T as a chain of nodes in E, one for each fact assigned to T.

Fix the σ-structure I and its tree decomposition T of width k. Informally, we build E by walking through the decomposition T and copying it by enumerating the new facts in the domain of each bag of T as a chain of nodes in E, picking the labels in Γ^k_σ so that the elements shared between a bag and its parent in T are retained, and the new elements are chosen so as not to overlap with the parent node. Overlaps between one node and a non-parent or non-child node are irrelevant.

Formally, we proceed as follows. We start by precomputing a mapping that indicates, for every tuple $\mathbf{a}$ of I such that some fact $R(\mathbf{a})$ holds in I, the topmost bag $\mathrm{node}(\mathbf{a})$ of T such that $\mathbf{a} \subseteq \mathrm{dom}(\mathrm{node}(\mathbf{a}))$. This can be performed in linear time by Lemma 3.1 of [FFG02]. Then, we label the tree decomposition T with the facts of I as follows: for each fact $F = R(\mathbf{a})$ of I, we add F to the label of $\mathrm{node}(\mathbf{a})$.

Now, to encode a bag b of T, consider b_p the parent of b in T and partition $\mathrm{dom}(b) = d_\mathrm{o} \sqcup d_\mathrm{n}$ where d_o are the *old elements* already present in $\mathrm{dom}(b_\mathrm{p})$, and d_n are the *new elements* that did not appear in $\mathrm{dom}(b_\mathrm{p})$. (If b is the root, then $d_\mathrm{o} = \emptyset$ and $d_\mathrm{n} = \mathrm{dom}(b)$.) Under a specific node

$(d_\mathrm{p}, n_\mathrm{p})$ in E, with a bijection f_p from $\mathrm{dom}(b_\mathrm{p})$ to d_p, choose a domain d of size $|\mathrm{dom}(b)|$ over the fixed $a_1, \ldots, a_{2k+2}$ whose intersection with $f_\mathrm{p}(\mathrm{dom}(b_\mathrm{p}))$ is exactly $f_\mathrm{p}(d_\mathrm{o})$ (this is possible, as there are $2k+2$ elements to choose from and $|\mathrm{dom}(b_\mathrm{p})| \leqslant k+1$) and extend the bijection f_p to f so that it maps $\mathrm{dom}(b)$ to d. At the root, choose an arbitrary bijection. Now, encode b as a chain of nodes in E labeled with (d, s_i) where each s_i encodes one of the facts in the label of b (thus defining the bijection from I to the non-empty nodes of E). If there are zero such facts, create a $(d, \emptyset)$ zero-fact node instead, rather than creating no node. Recursively encode the children of b (if any) in T, under this chain of nodes in E. Add zero-fact $(\emptyset, \emptyset)$ child nodes so that each non-leaf node has exactly two children. We assume that all arbitrary choices are done in a consistent manner so that the process is deterministic. □

Hence, when restricting to instances whose width is bounded by a constant k, one can equivalently work with Γ^k_σ-trees which are encoding of these instances, instead of working with the instances themselves.

We redefine properly the notion of an automaton testing a query:

Definition B.4. *A Γ^k_σ-bNTA A* tests *a Boolean query q for treewidth k if for any Γ^k_σ-tree E, $E \models A$ iff $\langle E \rangle \models q$. (In particular, if $\langle E \rangle = \bot$ then A rejects E.)*

B.2. Proof of Theorem 4.1

We can now state and prove the theorem that says that MSO sentences can be tested by automata for any treewidth. The main problem is to justify that MSO sentences can be rewritten to MSO sentences on our tree encodings, as we can then use [TW68] to compile them to a bNTA. We rely on [FFG02] for that result, but we must translate between our tree encodings and theirs. We do so by a general technique of justifying that certain product trees annotated with both encodings can be recognized by a bNTA. Let us state and prove the result:

Theorem 4.1 [Cou90]. *For any $k \in \mathbb{N}$, for any MSO query q, one can compute a Γ^k_σ-bNTA A^k_q that tests q for treewidth $\leqslant k$.*

Proof. In the context of this proof, we define a *bDTA* (bottom-up deterministic tree automaton) as a bNTA but where ι and δ, rather than returning sets of reachable states, return a single state. This implies that the automaton has a unique run on any tree.

Let us fix $k \in \mathbb{N}^*$. We denote by $[m]$ the set $\{1, \ldots, m\}$ and by n_σ the number of relations in σ. Lemma 4.10 of [FFG02] shows that for a certain finite alphabet $\Gamma(\sigma, k)$, for any MSO formula φ over the signature σ, there exists an MSO formula φ^* such that for any $\Gamma(\sigma, k)$-tree t representing an instance I, t satisfies φ^* iff I satisfies φ. More precisely, one can define a partial $\langle\cdot\rangle'$ function on $\Gamma(\sigma, k)$-trees such that for every instance I of treewidth $\leqslant k$ there is a $\Gamma(\sigma, k)$-tree t such that $\langle T \rangle'$ is well-defined and isomorphic to I and for every $\Gamma(\sigma, k)$-tree T, we have $T \models \varphi^*$ iff $\langle T \rangle'$ is well-defined and $\langle T \rangle' \models \varphi$.

We first describe the alphabet $\Gamma(\sigma, k)$. A letter of $\Gamma(\sigma, k)$ is of the form $(\gamma_1, \gamma_2, \ldots, \gamma_{n_\sigma+2})$. The element γ_1 belongs to $2^{[k]^2}$ and describes the equalities between the elements inside a bag; the element γ_2 belongs to $2^{[k]^2}$ and describes the equalities between the elements from this bag and its parents; For $i \geqslant 3$, γ_i belongs to $2^{k^{\mathrm{arity}(R_i)}}$ and describes the tuples belonging to the relation R_i. In the $\Gamma(\sigma, k)$-trees, the encoding of equalities between values of the bags and its parents are described explicitly (by γ_2) rather than implicitly (by element reuse between parent and child, as in our encoding).

We next describe for which $\Gamma(\sigma, k)$ trees T is their encoding operation $\langle T \rangle'$ well-defined, and how it is then computed. We say that T is *well-formed* if $\langle T \rangle'$ is well-defined. We accordingly say that a Γ_σ^k-tree T is *well-formed* if $\langle T \rangle$ is *well-defined*, namely, different from $\bot$.

For a $\Gamma(\sigma, k)$-tree T, $\langle T \rangle'$ is well-defined iff for each node n with $\gamma := \lambda(n)$ and each child $n' \in \{L(n), R(n)\}$ with $\gamma' := \lambda(n')$:

- γ_1 is closed by transitive closure, i.e., if (i, j) and (j, e) belong to γ_1 then (i, e) belongs to γ_1
- γ_2 is closed by transitive closure (to check this, we need to consider paths, rather than the mere pair n and n') γ_2 is closed by transitive closure.
- for each (i, j) in γ_1 and (j, e) in γ_2' then (i, e) belongs to γ_2' (and symmetrically, reversing the roles of n and n')
- for each pair $(j_1, \ldots, j_l)$ in γ_m' and if for each b (i_b, j_b) in γ_2', then $(i_1, \ldots, i_l)$ is in γ_m (and vice-versa, reversing the roles of n and n'); a similar condition holds with γ_1

Note that these conditions are clearly expressible in MSO. While [FFG02] does not precisely describe the behavior of φ^* on $\Gamma(\sigma, k)$-trees which are not well-formed, the above justifies our assumption that φ^* tests well-formedness and rejects the trees which are not well-formed.

We now define $\langle T \rangle'$ as follows, if T is well-formed. Process E top-down. At each node $n \in E$ with $\gamma := \lambda(n)$ with parent node $n' \in E$ with $\gamma' := \lambda(n')$, pick fresh elements in $\mathcal{D}$ for the positions j such that there is no pair (i, j) in γ_2' (at the root, pick all fresh elements) and if (j_1, j_2) belongs to γ_1' then the same fresh element is assigned for the elements at both positions; if there is such a pair, pick the existing elements used when decoding n'. These choices define a mapping ν from the positions to the fresh elements and to existing elements. Now, for each $(j_1, \ldots, j_m)$ in γ_i, then the fact $R_i(\nu(j_1), \ldots, \nu(j_m))$ is added to I. If we ever attempt to create a fact that already exists, we ignore it.

We have reviewed the alphabet $\Gamma(\sigma, k)$ of [FFG02], the conditions for the well-definedness of $\langle T \rangle'$ and the semantics of this operation. Now, following [TW68, FFG02], with an additional step to determinize the resulting automaton to a bDTA, the formula φ^* from [FFG02] can be translated into a bDTA A_{theirs} on $\Gamma(\sigma, k)$-trees such that for any $\Gamma(\sigma, k)$-tree T, A_{theirs} accepts T iff $T \models \varphi^*$, that is, iff $\langle T \rangle'$ is well-defined and satisfies φ. Note that this implies that A_{theirs} is encoding-invariant. We now explain how to translate A_{theirs} to our desired bDTA A_{ours} over Γ_σ^k such that for every Γ_σ^k-tree T, A_{ours} accepts T iff $\langle T \rangle \models \varphi$.

We consider the alphabet $\Sigma = \Gamma_\sigma^k \times \Gamma(\sigma, k)$, and call π_1 and π_2 the operations on Σ-trees that map them respectively to Γ_σ^k and $\Gamma(\sigma, k)$ trees with same skeleton by keeping the first or second component of the labels. Given a Γ_σ^k-tree T_1 and a $\Gamma(\sigma, k)$-tree T_2 with same skeleton, we will write $T_1 \times T_2$ the Σ-tree obtained from them.

We will do this by building a Σ-bDTA A_{t} with the following properties:

1. If A_{t} accepts T then $\langle \pi_1(T) \rangle$ and $\langle \pi_2(T) \rangle'$ are well-defined and isomorphic.
2. For every Γ_σ^k-tree T_1 such that $\langle T_1 \rangle$ is well-defined, there exists a $\Gamma(\sigma, k)$-tree T_2 such that A_{t} accepts $T_1 \times T_2$.

Then, we can notice that from A_{theirs}, we can build an Σ-bDTA A'_{theirs} such that T is recognized by A'_{theirs} iff $\pi_2(T)$ is accepted by A'_{theirs}, and build A_{ours} as the conjunction of A_{t} and A'_{theirs}, projected to the first component (accept a Γ_σ^k-tree T_1 iff there is some $\Gamma(\sigma, k)$-tree T_2 such that

$T_1 \times T_2$ is accepted, which is possible using non-determinism, and then determinizing). It is now clear that A_{ours} thus defined accepts a Γ_σ^k-tree T_1 iff the Σ-tree $T_1 \times T_2$ is accepted, for some $\Gamma(\sigma, k)$-tree T_2, by A'_{theirs} and A_{t}: if this happens then T_2 is accepted by A_{theirs} and $\langle T_1 \rangle$ is isomorphic to $\langle T_2 \rangle'$ so $\langle T_2 \rangle \models \varphi$; and conversely, if $\langle T_1 \rangle$ models φ, there is some $\Gamma(\sigma, k)$-tree T_2 such that A_{t} accepts $T_1 \times T_2$, and this implies that $\langle T_2 \rangle'$ is isomorphic to $\langle T_1 \rangle$ so (as φ, being a constant-free MSO query, is invariant under isomorphisms) $\langle T_2 \rangle$ satisfies φ^* and A'_{theirs} accepts $T_1 \times T_2$. So it suffices to build the Σ-bDTA A_{t} with the desired properties.

We now define a simple encoding from Γ_σ^k to $\Gamma(\sigma, k)$ describing what is the tree T_2, given a well-formed tree T_1, such that A_{t} accepts $T_1 \times T_2$. It will then suffice to see that it is possible, with a MSO formula ψ, to check on a Σ-tree T whether $\pi_2(T)$ is the encoding of $\pi_1(T)$ in this sense. Indeed, we can then compile ψ to a Γ-bDTA using [TW68].

Consider a node $n_1 \in T_1$ and let $(d, s) := \text{dom}(n)$. We define the label of the corresponding node $n_2 \in T_2$. We define γ_1 so that the $k+1-\text{dom}(d)$ last elements are all equal to the $\text{dom}(d)$-th element (i.e., we complete $\text{dom}(d)$ to always have $k+1$ elements, by "repeating" the last element, where "last" is according to an arbitrary order on domain elements). We define γ_2 to indicate which elements of n_1 were shared with its parent node, completing it to be consistent with respect to γ_1. Last, we define $\gamma_3, \ldots, \gamma_{n_\sigma+2}$ to be the tuples of elements in the various relations of σ in $\langle T_1 \rangle$, with repetitions to be consistent according to γ_1.

It is clear that this encoding maps every tree T_1 such that $\langle T_1 \rangle$ is well-defined to a tree T_2 such that $\langle T_2 \rangle$ is well-defined and isomorphic to $\langle T_1 \rangle$. Now, to justify the existence of ψ, observe that the only non-local condition to check on T is the definition of the $\gamma_3, \ldots, \gamma_{n_\sigma+2}$; but we can clearly define by an MSO formula, for a node $n \in T$ with $(d, s) := \lambda(n)$, the exact set of facts stated by T for the elements represented by d (there are only a finite number of such "types"): they are defined to check, for all facts of the putative type, whether a node with the right fact is reachable following an undirected path where the same elements are kept along the path. So we can define an MSO formula checking for each node $n \in T$ whether the type of $\pi_1(n)$ in $\pi_1(T)$ in this sense matches the graph stated in $\pi_2(n)$. □

B.3. Proof of Theorem 4.2

We first give the formal definition of the treewidth of a circuit, which we omitted. To do so, we must first give a normal form for circuits:

Definition B.5. *Let $C = (G, W, g_0, \mu)$ be a Boolean circuit. The* fan-in *of a gate $g \in G$ is the number of gates $g' \in G$, such that $(g', g) \in W$. Note that our definitions of circuits impose that the fan-in of input gates is always 0 and the fan-in of NOT-gates is always 1. We say C is* arity-two *if the fan-in of AND- and OR-gates is always 2, where we allow constant 0- and 1-gates as gate types of their own (but require that such gates have fan-in of 0).*

Clearly this restriction is inessential as circuits can be rewritten in linear-time to an arity-two circuit by merging AND- and OR-gates with fan-in of 1 with their one input, replacing those with fan-in of 0 by a 0- or 1-gate, and rewriting those with fan-in > 2 to a chain of gates of the same type with fan-in 2.

We now define tree decompositions of circuits, and their relational encoding:

Definition B.6. *The relational signature σ_{Circuit} features one unary relation R^{i} which applies to input gates, two unary relations R^0 and R^1 which apply to constant 0- and 1-gates, one binary relation $R^\neg(g_{\text{o}}, g_{\text{i}})$ which applies to NOT-gates (the first element is the output and the second is the input), and two ternary relations $R^\wedge(g_{\text{o}}, g_{\text{i}}, g'_{\text{i}})$ and $R^\vee(g_{\text{o}}, g_{\text{i}}, g'_{\text{i}})$ which apply respectively*

to AND- and OR-gates, with the first element being the input and the second and third being the inputs. The relational encoding *of an (arity-two non-monotone) Boolean circuit C is the σ_{Circuit}-instance I_C obtained in the expected way; we can clearly construct I_C from C in linear time. The* treewidth $\mathrm{w}(C)$ *of C is $\mathrm{w}(I_C)$ (but we talk of tree decompositions of C as shorthand).*

Now, to prove the theorem, we first take care of a technical issue. Given an instance I and its tree encoding T_I, we want to construct provenance circuits on T_I, meaning that we wish to consider Boolean-annotated versions of T_I. However, the tree encodings of subinstances of I are not annotated versions of T_I. We need to justify that they can be taken to have the same structure as T_I, with some facts having been removed in nodes containing no fact.

Definition B.7. *For any k-fact $\tau = (d, s) \in \Gamma_\sigma^k$, we define the* neutered k-fact $\underline{\tau}$ *as $(d, \emptyset)$. In particular, if $s = \emptyset$ then $\underline{\tau} = \tau$. For $\tau \in \Gamma_\sigma^k$ and $b \in \{0, 1\}$, we write $\tau_{[b]}$ to be τ if b is 1 and $\underline{\tau}$ if b is 0.*

Given a $\overline{\Gamma_\sigma^k}$-tree E, we define its evaluation $\epsilon(E)$ *as the Γ_σ^k-tree that has same skeleton, where for every node $n \in E$ with corresponding node n' in $\epsilon(E)$, letting $\lambda(n) = (\tau, b) \in \Gamma_\sigma^k \times \{0, 1\}$, we have $\lambda(n') = \tau_{[b]}$.*

This definition allows us to lift Γ_σ^k-bNTAs, intuitively testing a query, to $\overline{\Gamma_\sigma^k}$-bNTAs that test the same query on Boolean-annotated tree encodings, seen via ϵ as the tree encoding of a subinstance. Formally:

Lemma B.2. *For any Γ_σ^k-bNTA A, one can compute in linear time a $\overline{\Gamma_\sigma^k}$-bNTA A' on such that $E \models A'$ iff $\epsilon(E) \models A$.*

Proof. Let $A = (Q, F, \iota, \delta)$. We construct the bNTA $A' = (Q, F, \iota', \delta')$ according to the following definition: $\iota'((\tau, b)) := \iota(\tau_{[b]})$ and $\delta'((\tau, b), q_1, q_2) := \delta(\tau_{[b]}, q_1, q_2)$ for all $b \in \{0, 1\}$, $\tau \in \Gamma_\sigma^k$, and $q_1, q_2 \in Q$. The process is clearly in linear time in $|A|$. Now, it is immediate that $E \models A'$ iff $\epsilon(E) \models A$, because a run of A' on E is a run of A on $\epsilon(E)$, and vice-versa. □

We are now ready to state and prove the result:

Theorem 4.2. *For any fixed $k \in \mathbb{N}$ and GSO query q, for any σ-instance I such that $\mathrm{w}(I) \leqslant k$, 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).*

Proof. Fix $k \in \mathbb{N}$ and the GSO query q. Using Theorem 4.1, let A be a Γ_σ^k-bNTA A_q^k that tests q for treewidth k (remember that Theorem 4.1 extends from MSO to GSO because both collapse on treelike instances [GHO02]). We lift A to a bNTA A' on $\overline{\Gamma_\sigma^k}$ using Lemma B.2. This is performed in constant time in the instance.

Now, given the input instance I such that $\mathrm{w}(I) \leqslant k$, compute in linear time [Bod96] a tree decomposition of I, and, from this, compute in linear time a tree encoding E_I of I using Lemma B.1. We now use Proposition 3.1 to construct a provenance circuit C of A' on E_I. Consider now the injective function f that maps the facts of I to the nodes of E_I where those facts are encoded. We modify C to replace the input gate g_n^{i} for any $n \in E_I$ not in the image of f, setting it to be a 1-gate; and renaming the input gates g_n^{i} for any $n \in E_I$ to be F, for F the fact such that $f(F) = n$. Let C' be the result of this process. C' is thus a Boolean circuit such that $C'_{\mathsf{inp}} = I$, and it was computed in linear time from I. We claim that it captures $\mathrm{Prov}(q, I)$.

To check that it does, let $\nu : I \to \{0, 1\}$ be a valuation of I. We show that $\nu(C') = 1$ iff $\nu(I) \models q$. To do so, the key point is to observe that, letting ν' be the valuation of E_I defined

by $\nu'(n) = \nu(F)$ if there is $F \in I$ such that $f(F) = n$, and $\nu'(n) = 1$ otherwise, we have that $\epsilon(\nu'(E_I))$ is a tree encoding of $\nu(I)$. Indeed, $\epsilon(\nu'(E_I))$ and E_I have same skeleton, the elements that constitute the domains of node labels are the same, and a fact $F \in I$ is encoded in $\epsilon(\nu'(E_I))$ iff $f(F)$ is annotated by 1 in $\nu'(E_I)$ iff we have $F \in \nu(I)$. (Note that our choice to extend ν' by setting it to be 1 on nodes that encode no facts makes no difference, as the annotation of such nodes is projected to the same label by ϵ, i.e., for such labels τ in E_I, we have $\tau_{[0]} = \tau_{[1]}$.)

Having observed this, we know that, because A tests q, $\nu(I) \models q$ iff $\nu'(E_I) \models A$. Now, by definition of Lemma B.2, we have $\epsilon(\nu'(E_I)) \models A$ iff $\nu'(E_I) \models A'$, which by definition of the provenance circuit C is the case iff $\nu'(C) = 1$, which by definition of C' is the case iff $\nu(C') = 1$. Hence, C' is indeed a provenance circuit of q on I.

The only point left to justify is that the treewidth of the circuit C is indeed bounded. Indeed, the number of gates that we create in C for each node n of E only depends on the automaton A that tests q, and wires in C only go from gates for one node n to gates for nodes $L(n)$ and $R(n)$, so that the tree decomposition T for C is obtained by putting, in each bag of the tree decomposition corresponding to node n of E, the gates for node n, and $L(n)$ and $R(n)$ if they exist. The additional distinguished gate g_0 is added to the bag of the root node of E. This construction is described on the circuit before translating to arity-two, but as the fan-in of the gates of the original circuit is bounded by a constant, clearly rewriting to arity-two preserves the property that the number of gates per bag of the decomposition is bounded by a constant. This proves the claim. □

B.4. Proof of EXPTIME rewriting of UCQs

Proposition B.1 [CV92]. *For any UCQ q and $k \in \mathbb{N}$, a Γ_σ^k-bNTA that tests q for treewidth $\leqslant k$ can be computed in EXPTIME in q and k.*

Proof. Let q be a UCQ and k be an integer.

This proof relies on the notion of proof trees introduced in [CV92]. The proof trees are intuitively tree encodings of an *unfolding*, or *expansion tree*, of a Datalog query P (refer to Definition C.1 for the definition of Datalog). An *expansion tree* of P is a ranked tree (not binary in general) defined as follows: the node labels are pairs of a fact F from an intentional predicate of σ_{int} and an instantiation of the body of a rule $r \in P$ (i.e., the variables are mapped to elements of the instance in a way that satisfies the body of r) such that the corresponding instantiation of the head of r is F.

Such a tree is *well-formed* if for any node n labeled by (F, x) there is a bijection f between the children of n and the intensional facts of the instantiation x such that for any node n, $f(n)$ is exactly the head fact of n. (In particular, if the same intensional fact is used multiple times in the rule, then there are as many children as there are occurrences of this fact). We will require that in the rules of the query P, every body contains either 0 or 2 intensional facts, so that expansion trees are full binary trees.

From an expansion tree, it is possible to derive a *proof tree*, which is a $\Sigma(P)$-tree for some finite set $\Sigma(P)$ (for fixed P), as follows: the alphabet $\Sigma(P)$ is the set of pairs of tuples over some fixed set of $2\,|P|$ values and of a rule of P, and the intuition of a $\Sigma(P)$-tree, just like for our notion of k-facts, is that sharing an element between one node and its parent encodes that it is the same element, but elements shared between, e.g., siblings, are not necessarily the same element. Note that proof trees, as expansion trees, are full binary trees. In this proof we

use proof trees to mean this, as [CV92], and we do not mean the notion of proof tree used for Datalog provenance in Definition C.1.

Having described how to encode an expansion tree to a proof tree, we describe the *decoding* $\langle T \rangle'$ of a $\Sigma(P)$-tree T: first, apply a process analogous to our own notion of decoding, to obtain an expansion tree T'; second, consider the extensional facts that appear in the instantiation of the bodies in the labels of T', and define $\langle T \rangle'$ to be the instance formed of those facts. Of course, if any of these processes fails, or if the intermediate expansion tree is not well-formed, we abort and set $\langle T \rangle' = \bot$.

Our goal is now to define a Datalog query P such that there is a surjective homomorphism from $\Sigma(P)$ to Γ_σ^k. Fix σ_{int} to have intensional relations $P_0, \ldots, P_{k+1}$ of arity $0, \ldots, k+1$. (We technically disallowed predicates of arity 0 in our definition of instances, but there is clearly no problem in this context.) For every tuples of variables $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z_1}$, $\mathbf{z_2}$ taken from a set of $3k + a_\sigma$ variables denoted by S_X (where a_σ is the arity of σ), with the condition $\mathbf{y} \subseteq \mathbf{x}$, for every relation R of σ, and $0 \leqslant i, j_1, j_2 \leqslant k+1$, create the rules in P:

$$P_i(\mathbf{x}) \leftarrow R(\mathbf{y}) P_{j_1}(\mathbf{z_1}) P_{j_2}(\mathbf{z_2})$$

and

$$P(\mathbf{x}) \leftarrow R(\mathbf{y})$$

Finally, we create the rules

$$P_i(\mathbf{x}) \leftarrow P_{j_1}(\mathbf{z_1}) P_{j_2}(\mathbf{z_2})$$

In terms of size, each rule of the query P contains a number of variables polynomial in k and σ, and the overall size of the query is exponential in a polynomial of k and σ. Last, the size of $\Sigma(P)$ is in $O(|P| \cdot |P|^{a(P)})$, where $a(P)$ is the maximal arity of the intentional relations. Let β be a set of values of cardinality equal to $2k+2$. Then, $\Sigma(P)$ is equal to the pairs $P(\mathbf{a}), r$ where r is a rule. We define the following homomorphism h from $\Sigma(P)$ to Γ_σ^k. Let $(P_i(\mathbf{a}), r)$ be a element of $\Sigma(P)$. If r does not have an extensional fact then $h(P_i(\mathbf{a}), r)$ is equal to $(\mathbf{a}, \emptyset)$. Otherwise, the atom $R(\mathbf{y})$ occurs in the body of r, let ν be the valuation from the variables of r defined according to the head atom $P_i(\mathbf{a})$ (as we imposed $\mathbf{y} \subseteq \mathbf{x}$ above) such that $\nu(\mathbf{x})$ is equal to $\mathbf{a}$: $h(P(\mathbf{a}), r)$ is equal to $(\mathbf{a}, R(\nu(\mathbf{y}))$. h is thus defined from $\Sigma(P)$-trees to the Γ_σ^k, and it is clearly surjective. Furthermore, it is clear that this application extends to a surjective mapping h' from $\Sigma(P)$-trees to Γ_σ^k-trees, with the property that whenever $\langle h'(T) \rangle$ is defined then T is well-formed and $\langle h'(T) \rangle$ and $\langle T \rangle'$ are isomorphic.

We now explain how we construct our automaton for the query q. Let us first assume that q is a conjunctive query (CQ). We consider the Datalog query P that we constructed above. From the proof of Proposition 5.10 of [CV92], we deduce that we can construct, in time polynomial in its size, a bNTA A_P on $\Sigma(P)$ whose number of states is in is in $O(|\Sigma(P)| \cdot 2^{|q| + V_q * V_P})$, where V_P (resp., V_q) is the maximal number of variables in a rule of P (resp., in q) such that A_P recognizes the language of the well-formed $\Sigma(P)$-trees T such that $\langle T \rangle'$ satisfies q. For our query P, the size of A_P is therefore exponential in a polynomial of k, σ and $|q|$.

Because h' is an surjective homomorphism from $\Sigma(P)$-trees to Γ_σ^k-trees and A_P is on $\Sigma(P)$ with the Property 1.4.3 of [CDG+07] that shows that bNTA are closed by homomorphism, we compute in polynomial time in A_P a bNTA A'_P on Γ_σ^k that has size exponential in a polynomial of σ, $|q|$ and k. We intersect it with a bNTA (clearly constructible in EXPTIME) that checks whether a Γ_σ^k-tree is a valid encoding, and rejects otherwise. This yields the final automaton A.

We now check that A tests the query q. Let T be a Γ_σ^k-tree. If $\langle T \rangle$ satisfies q, then it is well-defined, Let T' be a preimage of T by h'. By our condition on h', $\langle T' \rangle'$ is well-defined and isomorphic to $\langle T \rangle$, so (as q features no constants and is thus preserved by isomorphisms) it satisfies q, and therefore T' was accepted by A_P, so T is accepted by A. Conversely, if A accepts T, then let T' be a preimage of T by h such that A' accepts T'. As $\langle T \rangle$ is well-defined, T' is well-defined and $\langle T' \rangle'$ and $\langle T \rangle$ are isomorphic; but as T' is accepted by A_P, we must have $\langle T' \rangle' \models q$, so $\langle T \rangle \models q$.

The result can be extended to an UCQ q by applying the result to every CQ and taking the union of the resulting automata (whose size is the sum of the input automata). □

C. Proofs for Section 5 (General Semirings)

Definition C.1. *A* Datalog query P *over the signature* σ *consists of a signature* σ_{int} *of* intensional predicates *with a special* 0*-ary relation* Goal *and a finite set of* rules *of the form* $R(\mathbf{x}) \leftarrow R_1(\mathbf{y_1}), \ldots, R_k(\mathbf{y_k})$ *where* $R \in \sigma_{\text{int}}$, $R_i \in \sigma \sqcup \sigma_{\text{int}}$ *for* $1 \leqslant i \leqslant k$, *and each variable in the tuple* $\mathbf{x}$ *also occurs in some tuple* $\mathbf{y_i}$. *The left-hand (resp., right-hand) side of a Datalog rule is called the* head *(resp.,* body*) of the rule.*

A proof tree T *of a Datalog query* P *over an instance* I *is a (non-binary) ordered tree with nodes annotated by facts over* $\sigma \cup \sigma_{\text{int}}$ *on elements of* $\text{dom}(I)$ *and internal nodes annotated by rules, such that the fact of the root of* T *is* Goal*, and, for every internal node* n *in* T *with children* $n_1, \ldots, n_m$, *the indicated rule* $R(\mathbf{x}) \leftarrow \Psi(\mathbf{y})$ *on* n *in* P *is such that there is a homomorphism* h *mapping* $R(\mathbf{x})$ *to the fact of* n *and mapping* $\Psi(\mathbf{y})$ *to the facts of the* n_i. *(Note that this definition implies that internal nodes are necessarily annotated by a fact of* σ_{int}*.) We write* $I \models P$ *if* P *has a proof tree on* I.

The *semiring provenance* of a Datalog query on an instance is defined as follows:

Definition C.2. *Given a semiring* K *and an instance* I *where each fact* F *carries an annotation* $\alpha(F) \in K$, *the* provenance *of a Datalog query* P *on* I *is the following [GKT07]:*

$$\bigoplus_{T \text{ proof tree of } P} \bigotimes_{n \text{ leaf of } T} \alpha(n).$$

Note that this expression may not always be defined depending on the query, instance, and semiring. In particular, the number of terms in the sum may be infinite, so that the result cannot necessarily be represented in the semiring.

We now use this to define Datalog queries associated to conjunctive queries (CQs) and unions of CQs (UCQs), which will be useful for provenance.

Definition C.3. *We assume that CQs and UCQs contain no equality atoms. The Datalog query* P_q associated *to a CQ* q *has only one rule,* $\text{Goal} \leftarrow q$. *The Datalog query* P_q *associated to a UCQ* $q = \bigvee_i q_i$ *has rules* $\text{Goal} \leftarrow q_1$, ..., $\text{Goal} \leftarrow q_n$.

Observe that in this case the provenance of P_q *in the sense of Definition C.2 is always defined, no matter the semiring, as the number of possible derivation trees is clearly finite.*

Theorem 5.1. *For any fixed* $k \in \mathbb{N}$ *and monotone GSO query* q, *for any* σ*-instance* I *such that* $\text{w}(I) \leqslant k$, *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*).*

We first define our notion of monotonicity for $\overline{\Gamma}$-bNTAs. Intuitively, it implies that if a $\overline{\Gamma}$-tree is accepted by the automaton, it will not be rejected when changing annotations from 0 to 1.

Formally:

Definition C.4. *We consider the partial order $<$ on $\overline{\Gamma}$ defined by $(\tau, 0) < (\tau, 1)$ for all $\tau \in \Gamma$. We say that a $\overline{\Gamma}$-bNTA $A = (Q, F, \iota, \delta)$ is* monotone *if for every $\tau \leqslant \tau'$ in $\overline{\Gamma}$, we have $\iota(\tau) \subseteq \iota(\tau')$ and $\delta(q_1, q_2, \tau) \subseteq \delta(q_1, q_2, \tau')$ for every $q_1, q_2 \in Q$.*

It is easy to see that the provenance of a *monotone* $\overline{\Gamma}$-bNTA A on any tree T is a *monotone* function in the following sense: for any valuations ν and ν', if $\nu(g) = 1$ implies $\nu'(g) = 1$ for all $g \in C_{\mathsf{inp}}$ (which we write $\nu \leqslant \nu'$), then $(\mathrm{Prov}(A, T))(\nu) = 1$ implies $(\mathrm{Prov}(A, T))(\nu') = 1$. Indeed, for any monotone Γ-tree T and valuations $\nu \leqslant \nu'$, $\nu(T) \models A$ implies $\nu'(T) \models A$.

We already know that the results of Section 3 also hold for monotone bNTAs and monotone provenance circuits (see Proposition A.1). Hence, if we know that the monotone GSO sentence q of interest can be tested (Definition B.4) by a *monotone* bNTA, i.e., if the analogue of Theorem 4.1 holds for monotone queries and bNTAs, then clearly the reduction from treelike instances to trees (Theorem 4.2) generalizes. We first prove an auxiliary lemma:

Lemma C.1. *For every $\overline{\Gamma^k_\sigma}$-trees E and E', if $E \leqslant E'$ then $\langle \epsilon(E) \rangle \subseteq \langle \epsilon(E') \rangle$.*

Proof. We follow the decoding process and notice that, as the domains of the Γ^k_σ node labels in E and E' are the same, the same fresh elements are used throughout, so the only difference between $\langle \epsilon(E) \rangle$ and $\langle \epsilon(E') \rangle$ is about the annotation of the created facts; and we notice that whenever $E \leqslant E'$ then every fact created in E is also created in E'. ☐

We now prove the analogue of Theorem 4.1 for monotone queries and automata. We immediately generalize the notion of an automaton testing a query (Definition B.4) to $\overline{\Gamma^k_\sigma}$-bNTAs and trees, with $\langle T \rangle$ for a $\overline{\Gamma^k_\sigma}$-tree T defined as above.

Lemma C.2. *Let $k \in \mathbb{N}$, q be a monotone query, and A be a $\overline{\Gamma^k_\sigma}$-bNTA that tests q for treewidth k. One can compute in linear time from A a $\overline{\Gamma^k_\sigma}$-bNTA A' that tests q for treewidth k and is monotone for the partial order on $\overline{\Gamma^k_\sigma}$.*

Proof. Fix k, q, and $A = (Q, F, \iota, \delta)$ the $\overline{\Gamma^k_\sigma}$-bNTA. We build the bNTA $A' = (Q, F, \iota', \delta')$ by setting, for all $(\tau, i) \in \overline{\Gamma^k_\sigma}$, $\iota'((\tau, i)) := \bigcup_{0 \leqslant j \leqslant i} \iota((\tau, j))$ and, for all $q_1, q_2 \in Q$, we pose: $\delta'(q_1, q_2, (\tau, i)) := \bigcup_{0 \leqslant j \leqslant i} \delta(q_1, q_2, (\tau, j))$.

Clearly A' is monotone by construction for $\overline{\Gamma^k_\sigma}$. Besides, for any $\overline{\Gamma^k_\sigma}$-tree T, if A accepts T then A' accepts T, so to prove the correctness of A' it suffices to prove the converse implication.

Let us consider such a T, and consider an accepting run ρ of A' on T. We build a new tree T' whose skeleton is that of T and where for any leaf (resp. internal node) $n' \in T'$ with corresponding node $n \in T$ with $\lambda(n) = (\tau, j)$, we set $\lambda(n')$ in T' to be (τ, i) for some i such that $\rho(n) \in \iota((\tau, i))$ (resp. $\rho(n) \in \delta(\rho(L(n)), \rho(R(n)), (\tau, i))$), the existence of such an i being guaranteed by the definition of ι' (resp. δ').

We now observe that, by construction, ρ is a run of A on T', and it is still accepting, so that T' is accepted by A. Hence, $\langle T' \rangle \models q$. But now we observe that, once again by construction, for every node n' of T' with label τ' and with corresponding node n in T with label τ, it holds that $\tau' \leqslant \tau$. Hence we have $T' \leqslant T$, for which we can easily prove that $\langle T' \rangle \subseteq \langle T \rangle$, and thus, by monotonicity of q, we must have $\langle T \rangle \models P$. Thus, A accepts T, proving the desired result. ☐

From our earlier explanations, this proves Theorem 5.1.

C.1. $\mathbb{N}[X]$-provenance for UCQs

Proposition C.1. *For any UCQ q, $\mathrm{Prov}_{\mathbb{N}[X]}(q, I)$, is the $\mathbb{N}[X]$-provenance in the sense of [GKT07] of the associated Datalog query P_q on I.*

Proof. The proof trees of each CQ within q have a fixed structure, the only unspecified part being the assignment of variables. It is then clear that each variable assignment gives a proof tree, and this mapping is injective because all variables in the assignment occur in the proof tree. So for each CQ, we are summing on the same thing, and each term of the sum is the leaves of the proof tree, which is what we imposed. Further, the set of proof trees of q is the union of the set of proof trees of each CQ, which means we can separate the sum for each CQ. □

C.2. Provenance circuits for trees

To prove Theorem 5.3, we need a generalization of $\mathbb{N}[X]$-provenance of automata on trees. Indeed, while Definition 5.3 is natural for trees, using it to define the provenance of queries on treelike instances would lead to a subtle problem. The reason is that this provenance describes *all* valuations of the tree for which the automaton accepts (up to the maximal multiplicity p), and not the *minimal* ones. For UCQs, this would intuitively mean that the resulting $\mathbb{N}[X]$-provenance would reflect *all* subinstances satisfying the query, not the minimal ones. This does not match Definition 5.2 and is undesirable: for instance, the specialization of such a provenance to $\mathbb{N}$ would have nothing to do with the number of query matches. The reason why this problem did not occur before is because both choices of definition collapse in the PosBool$[X]$ setting of the previous sections; so this problem is specific to the setting of general semirings such as $\mathbb{N}[X]$, which are not necessarily *absorptive* [DMRT14].

However, in the case of UCQs, we can introduce a generalization of Definition 5.3, for which we can show the analogue of Theorem 5.2, and that will give us the right provenance once lifted to treelike instances as in Section 4. For a Γ-tree T and $p, l \in \mathbb{N}$, we introduce for $l \in \mathbb{N}$ the set of p-valuations that sum to l: $\mathrm{Val}^p_l(T) := \{\nu \in \mathrm{Val}^p(T) \mid \sum_{n \in T} \nu(n) = l\}$. We take $\mathrm{Val}^p_{\mathrm{all}}(T)$ to be $\mathrm{Val}^p(T)$. We can now generalize Definition 5.3 as follows:

Definition C.5. *Let $l \in \mathbb{N} \cup \{\mathrm{all}\}$. The $\mathbb{N}[X]$-l-*provenance *of a $\overline{\Gamma}^p$-bNTA A on a Γ-tree T is:*

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

Note that $\mathrm{Prov}_{\mathbb{N}[X]}(A, T) = \mathrm{Prov}_{\mathbb{N}[X]}(A, T, \mathrm{all})$, so this definition indeed generalizes Definition 5.3.

We first prove the key lemma about the propagation of provenance throughout encodings, which will be used in the inductive step of our correctness proof of provenance circuits:

Lemma C.3. *For any $l, p \in \mathbb{N}$, $l \leqslant p$, for any non-singleton Γ-tree $T = (V, L, R, \lambda)$, letting T_L and T_R be its left and right subtrees and n_{r} be its root node, for any $\overline{\Gamma}^p$-bNTA $A = (Q, F, \iota, \delta)$, writing A_q for all $q \in Q$ the bNTA obtained from A by making q the only final state, we have:*

$$\mathrm{Prov}_{\mathbb{N}[X]}(A, T, l) = \bigoplus_{\substack{l_1 + l_2 + l' = l \\ q_L, q_R \in Q \\ q \in \delta(q_L, q_R, (\lambda(n_{\mathrm{r}}), l'))}} \mathrm{Prov}_{\mathbb{N}[X]}(A_{q_L}, T_L, l_1) \otimes \mathrm{Prov}_{\mathbb{N}[X]}(A_{q_R}, T_R, l_2) \otimes n_{\mathrm{r}}^{l'}$$

Proof. We first observe the following identity, for any $\nu \in \mathrm{Val}_l^p(T)$ and any $q \in Q$, by definition of automaton runs:

$$|\mathrm{aruns}(A_q, \nu(T))| = \sum_{\substack{q_\mathrm{L}, q_\mathrm{R} \in Q \\ q \in \delta(q_\mathrm{L}, q_\mathrm{R}, (\lambda(n_\mathrm{r}), \nu(n_\mathrm{r})))}} |\mathrm{aruns}(A_{q_\mathrm{L}}, \nu(T_\mathrm{L}))| \cdot |\mathrm{aruns}(A_{q_\mathrm{R}}, \nu(T_\mathrm{R}))|$$

We then observe that $\mathrm{Val}_l^p(T)$ can be decomposed as

$$\bigsqcup_{l_1 + l_2 + l' = l} \mathrm{Val}_{l_1}^p(T_\mathrm{L}) \times \mathrm{Val}_{l_2}^p(T_\mathrm{R}) \times \{n_\mathrm{r} \mapsto l'\}$$

as a valuation of T summing to l can be chosen as a valuation of its left and right subtree and of n_r by assigning the possible weights. We also observe that the product over $n \in T$ can be split in a product on n_r, on $n \in T_\mathrm{L}$ and on $n \in T_\mathrm{R}$. We can thus rewrite as follows:

$$\mathrm{Prov}_{\mathbb{N}[X]}(A, T, l) = \bigoplus_{\substack{\nu_\mathrm{L} \in \mathrm{Val}_{l_1}^p(T_\mathrm{L}) \\ \nu_\mathrm{R} \in \mathrm{Val}_{l_2}^p(T_\mathrm{R}) \\ l_1 + l_2 + l' = l}} \bigoplus_{\substack{q_\mathrm{L}, q_\mathrm{R} \in Q \\ q \in \Delta}} m_\mathrm{L} \cdot m_\mathrm{R} \left(\bigotimes_{n \in T_\mathrm{L}} n^{\nu_\mathrm{L}(n)} \right) \left(\bigotimes_{n \in T_\mathrm{R}} n^{\nu_\mathrm{R}(n)} \right) n_\mathrm{r}^{l'}$$

where we abbreviated $m_\mathrm{L} := |\mathrm{aruns}(A_{q_\mathrm{L}}, \nu(T_\mathrm{L}))|$, $m_\mathrm{R} := |\mathrm{aruns}(A_{q_\mathrm{R}}, \nu(T_\mathrm{R}))|$, and $\Delta := \delta(q_\mathrm{L}, q_\mathrm{R}, (\lambda(n_\mathrm{r}), l'))$.

Reordering sums and performing factorizations, we obtain:

$$\mathrm{Prov}_{\mathbb{N}[X]}(A, T, l) = \bigoplus_{\substack{q_\mathrm{L}, q_\mathrm{R} \in Q \\ l_1 + l_2 + l' = l \\ q \in \Delta}} \left(\bigoplus_{\nu_\mathrm{L} \in \mathrm{Val}_{l_1}^p(T_\mathrm{L})} m_\mathrm{L} \bigotimes_{n \in T_\mathrm{L}} n^{\nu_\mathrm{L}(n)} \right) \left(\bigoplus_{\nu_\mathrm{R} \in \mathrm{Val}_{l_2}^p(T_\mathrm{R})} m_\mathrm{R} \bigotimes_{n \in T_\mathrm{R}} n^{\nu_\mathrm{R}(n)} \right) n_\mathrm{r}^{l'}.$$

Plugging back the definition of provenance yields the desired claim. □

We then prove the following variant of Theorem 5.2:

Theorem C.1. *For any fixed $p \in \mathbb{N}$ and $l \in \mathbb{N} \cup \{\mathrm{all}\}$, a $\mathbb{N}[X]$-l-provenance circuit for a $\overline{\Gamma}^p$-bNTA A and a Γ-tree T (i.e., a $\mathbb{N}[X]$-circuit capturing $\mathrm{Prov}_{\mathbb{N}[X]}(A, T, l)$) can be constructed in time $O(|A| \cdot |T|)$.*

Proof. We modify the proof of Proposition A.1.

We fix l_0 to be the l provided as input. We will first assume $l_0 \in \mathbb{N}$, we explain at the end of the proof how to handle the (simpler) case $l_0 = \mathrm{all}$.

For every node n of the tree T, we create one input gate g_n^i in C (identified to n), and for $j \in \{0, \ldots, p\}$, we create a gate $g_n^{\mathrm{i},j}$ which is a $\otimes$-gate of j copies of the input gate g_n^i. (By "copies" we mean $\otimes$- or $\oplus$-gates whose sole input is g_n^i, this being a technical necessity as K-circuits are defined as graphs and not multigraphs.) In particular, $g_n^{\mathrm{i},0}$ is always a 1-gate.

We create one gate $g_n^{q,l}$ for $n \in T$, $q \in Q$, and $0 \leqslant l \leqslant l_0$.

For leaf nodes n, for $q \in Q$, we set $g_n^{q,l}$ to be $g_n^{\mathrm{i},l}$ if $q \in \iota(\lambda(n), l)$ and a 0-gate otherwise.

For internal nodes n, for every pair $q_\mathrm{L}, q_\mathrm{R} \in Q$ (that appears as input states of a transition of δ) and $0 \leqslant l_1, l_2 \leqslant l_0$ such that $l_1 + l_2 \leqslant l_0$, we create the gate $g_n^{q_\mathrm{L}, l_1, q_\mathrm{R}, l_2}$ as an $\otimes$-gate of $g_{L(n)}^{q_\mathrm{L}, l_1}$ and $g_{R(n)}^{q_\mathrm{R}, l_2}$, and, for $0 \leqslant l' \leqslant l_0$ such that $l_1 + l_2 + l' \leqslant l_0$, we create one gate $g_n^{q_\mathrm{L}, l_1, q_\mathrm{R}, l_2, l'}$ as

the $\otimes$-gate of $g_n^{q_\mathrm{L},l_1,q_\mathrm{R},l_2}$ and $g^{\mathrm{i},l'}$. For $0 \leqslant l \leqslant l_0$, we set $g_n^{q,l}$ to be a $\oplus$-gate of all the $g_n^{q_\mathrm{L},l_1,q_\mathrm{R},l_2,l'}$ such that $q \in \delta(q_\mathrm{L}, q_\mathrm{R}, (\lambda(n), l'))$ and $l_1 + l_2 + l' = l$.

We define the distinguished gate g_0 as an $\otimes$-gate of the $g_{n_\mathrm{r}}^{q,l_0}$ where n_r is the root of T. The construction is again in $O(|A| \cdot |T|)$ for fixed l and p.

To prove correctness, we show by induction that the element captured by $g_n^{q,l}$ is $\mathrm{Prov}_{\mathbb{N}[X]}(A_q, T_n, l)$ where A_q is A with q as the only final state, and T_n is T rooted at n.

As a general property, note that for any node n, the value captured by $g_n^{\mathrm{i},j}$ for $0 \leqslant j \leqslant p$ is n^j.

For a leaf node n, $\mathrm{Prov}_{\mathbb{N}[X]}(A_q, T_n, l) = n^l$ if $q \in \iota(\lambda(n), l)$ and 0 otherwise, which is the value captured by $g_n^{q,l}$.

For an internal node n, the claim follows immediately by Lemma C.3, applying the induction hypothesis to $g_{L(n)}^{q_L,l_1}$ and $g_{R(n)}^{q_R,l_2}$.

We conclude because clearly we have $\mathrm{Prov}_{\mathbb{N}[X]}(A, T, l_0) = \bigoplus_{q \in F} \mathrm{Prov}_{\mathbb{N}[X]}(A_q, T, l_0)$, so the value captured by g_0 is indeed correct.

Now, if $l_0 = \mathrm{all}$, we do the same construction, but we only need a single node g_n^q for $n \in T$ and $q \in Q$ instead of $l_0 + 1$ nodes $g_n^{q,l}$. For leaf nodes, g_n^q is the $\oplus$-node of the $g_n^{\mathrm{i},l}$; for internal nodes, g_n^q is simply the $\oplus$-gate of all $g_n^{q_L,q_R,l}$ gates with $q \in \delta(q_L, q_R, (\lambda(n), l))$, each of them being an $\otimes$-gate of the $g_n^{q_L,q_R}$ gate and the $g^{\mathrm{i},l}$ gate. Finally, $g_n^{q_L,q_R}$ is the $\otimes$-gate of $g_{L(n)}^{q_L}$ and $g_{R(n)}^{q_R}$. Correctness is shown using a variant of Lemma C.3 on $\mathrm{Prov}_{\mathbb{N}[X]}(A, T, \mathrm{all})$ which replaces $l_1 + l_2 + l' = l$ in the sum subscript by $0 \leqslant l' \leqslant p$. □

C.3. Provenance circuit for instances

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

We first give some preliminary definitions. We need to introduce *bag-instances*, to materialize the possibility that a fact is used multiple times in a UCQ:

Definition C.6. *A* multiset *is a function M from a finite* support $\mathrm{supp}(M)$ *to $\mathbb{N}$. We define the relation $M \subseteq M'$ if $\mathrm{supp}(M) \subseteq \mathrm{supp}(M')$ and for all $s \in \mathrm{supp}(M)$ we have $M(s) \leqslant M'(s)$. We write $x \in M$ to mean that $M(x) > 0$. Given a set S and multiset M, we write $M \sqsubseteq S$ to mean that $\mathrm{supp}(M) \subseteq S$, and for $p \in \mathbb{N}$ we write $M \sqsubseteq^p S$ to mean that $M \sqsubseteq S$ and $M(a) \leqslant p$ for all $a \in \mathrm{supp}(M)$.*

A bag-instance *J is a multiset of facts on $\mathrm{dom}(J)$. Where necessary to avoid confusion, we call the ordinary instances* set-instances*. For two bag-instances J and J', we say that J is a* bag-subinstance *of J' if $J \subseteq J'$ holds (as multisets). We say that J is a* bag-subinstance *of a set-instance I if $J \sqsubseteq I$, and a p-*bag-subinstance *of I if $J \sqsubseteq^p I$. In other words, set-instances are understood as bag-instances where facts have an arbitrarily large multiplicity (and* not *multiplicity equal to 1). The* truncation to p *of a bag-instance J is $J^{\leqslant p}(F) := \min(J(F), p)$ for all $F \in \mathrm{supp}(J)$.*

A bag-homomorphism *h from a bag-instance J to a bag-instance J' is a mapping from $\mathrm{supp}(J)$ to $\mathrm{supp}(J')$ with the following condition: for each $F \in \mathrm{supp}(J')$, letting $F_1, \ldots, F_n$ be the facts of $\mathrm{supp}(J)$ such that $h(F_i) = F$ for $1 \leqslant i \leqslant n$, we have $\sum_{i=1}^n J(F_i) \leqslant J'(F)$.*

We accordingly define *bag-queries* as queries on such bag-instances. Intuitively, bag-queries are like regular Boolean queries on instances, except that they can "see" the multiplicity of

facts. This is crucial to talk about the required multiplicity of facts in matches, which we need to talk about the $\mathbb{N}[X]$-provenance of UCQs.

Definition C.7. *A* bag-query *q is a query on bag-instances. The bag-query q'* associated *to a CQ $\exists \mathbf{x}\, q(\mathbf{x})$ is defined as follows. A* match *of q in a bag-instance J is a bag-homomorphism from q (seen as a bag-instance of facts over $\mathbf{x}$) to J. We say that $J \models q'$ if q has a match in J.*

The bag-query associated to a UCQ $q = \bigvee_{i=1}^{n} \exists \mathbf{x}_i\, q_i(\mathbf{x}_i)$ is the disjunction of the bag-queries for each CQ in the disjunction.

Alternatively it is easily seen that $J \models q'$, for q' the bag-query associated to a UCQ q, iff J contains a bag of facts that can be used as the leaves of a derivation tree for the Datalog query P_q associated to q.

We notice that the bag-query associated to a UCQ q is *bounded*, namely, the fact that it holds or not cannot depend on the multiplicity of facts beyond a certain maximal value (the maximal number of atoms in a disjunct of the UCQ):

Definition C.8. *A bag-query q is* bounded *by $p \in \mathbb{N}$ if, for any bag-instance J, if $J \models q$, then the truncation $J^{\leqslant p}$ of J is such that $J^{\leqslant p} \models q$. A bag-query is* bounded *if it is bounded by some $p \in \mathbb{N}$.*

We now extend our definitions of tree encodings and automaton compilation to bag-queries. First, tree encodings simply generalize to bag-instances as tree encodings annotated with the multiplicities of facts, that is, $\overline{\Gamma_\sigma^k}^p$-trees:

Definition C.9. *Let $k, p \in \mathbb{N}$ and let J be a bag-instance such that $J(F) \leqslant p$ for all $F \in J$. Let $I := \mathrm{supp}(J)$ be the underlying instance of J, and let T_I be its tree encoding (a Γ_σ^k-tree). We define the (k,p)-*tree-encoding *T_J of J as the tree with same skeleton as T_I where any node n encoding a fact F of I is given the label $(\lambda(n), J(F))$ and other nodes are given the label $(\lambda(n), 1)$. We accordingly define $\langle\cdot\rangle$ on $\overline{\Gamma_\sigma^k}^p$-trees to yield bag-instances in the expected way, again returning $\bot$ whenever two nodes code the same fact (rather than summing up their multiplicity).*

We then define what it means for a $\overline{\Gamma_\sigma^k}^p$-bNTA to *test* a bag-query q. Note that the definition implies that the automaton cannot "see" multiplicities beyond p, so we require that the query be p-bounded so that the limitation does not matter.

Definition C.10. *For q a bag-query and $k, p \in \mathbb{N}$, a $\overline{\Gamma_\sigma^k}^p$-bNTA A* tests *q for treewidth k if q is bounded by p and for every $\overline{\Gamma_\sigma^k}^p$-tree E, we have $E \models A$ iff $\langle E\rangle \models q$.*

We then provide a general definition and prove a lemma about constructing the union of bNTAs:

Definition C.11. *Let Γ be a finite label set and let $A_i = (Q_i, F_i, \iota_i, \delta_i)$ be a family of Γ-bNTAs. Assume without loss of generality that the Q_i have been renamed so that they are pairwise disjoint. The* union bNTA *is the Γ-bNTA $A_\sqcup = (Q_\sqcup, F_\sqcup, \iota_\sqcup, \delta_\sqcup)$ defined by $Q_\sqcup := \bigsqcup_i Q_i$, $F_\sqcup := \bigsqcup_i F_i$, for every $\tau \in \Gamma$ $\iota_\sqcup(\tau) := \bigsqcup_i \iota_\sqcup(\tau)$, and $\delta_\sqcup$ is only defined for $q_1, q_2 \in Q_i$ for some Q_i, in which case it is defined as $\delta_\sqcup(q_1, q_2, \tau) := \delta_i(q_1, q_2, \tau)$.*

Lemma C.4. *For any family of Γ-bNTAs A_i, letting $A_\sqcup$ be the union bNTA of the A_i, for any Γ-tree T, we have $T \models A_\sqcup$ iff $T \models A_i$ for some A_i, and more precisely we have $|\mathrm{aruns}(A_\sqcup, T)| = \sum_i |\mathrm{aruns}(A_i, T)|$.*

Proof. The claim about acceptance and the number of runs is straightforward by noticing that the runs of $A_\sqcup$ on T are exactly the disjoint union of the runs of the A_i on T. ☐

Our last preliminary result is to show that every bag-query corresponding to a UCQ can be encoded to an automaton that tests it.

Proposition C.2. *Let q be a UCQ. There is $p \in \mathbb{N}$ such that, for any $k \in \mathbb{N}$, we can compute a $\overline{\Gamma_\sigma^k}^p$-bNTA A that tests q for treewidth k.*

Proof. We introduce some notation. We call $\mathrm{CQ}^{\neq}$ the language of CQs which can feature atoms of the form $x \neq y$, and $\mathrm{UCQ}^{\neq}$ the language of UCQs except the disjuncts are in $\mathrm{CQ}^{\neq}$. We write $\mathrm{Vars}(q)$ for the variables of a query q of $\mathrm{CQ}^{\neq}$. The bag-query associated to a query in $\mathrm{CQ}^{\neq}$ or $\mathrm{UCQ}^{\neq}$ is defined as for the corresponding query with no inequalities, but imposing the inequalities on matches. Formally, a match of a $\mathrm{CQ}^{\neq}$ query q' is a match h of q' such that $h(x) \neq h(y)$ for any two variables x and y such that $x \neq y$ occurs in q. (Note that the multiplicity of inequality atoms is irrelevant.)

We first note that, writing the UCQ q as the disjunction of CQs q_i, if we can show the claim for each q_i, then the result clearly follows from q by computing one bNTA A_i for each q_i that tests q_i for treewidth k and uses $p = \max_i p_i$, where p_i is the multiplicity for which the result was shown for each q_i (clearly if the claim holds for a value of p then it must hold for larger values by ignoring larger multiplicities). We then construct the union bNTA $A_\sqcup$ of these bNTAs to obtain a bNTA that tests q (Lemma C.4).

We see a CQ q as an existentially quantified multiset of atoms (the same atom, i.e., the same relation name applied to the same variables in the same order, can occur multiple times; in other words we distinguish, e.g., $\exists x R(x)$ and $\exists x R(x)R(x)$). Let $\mathrm{Vars}(q)$ be the set of the variables of q (which are all existentially quantified as q is Boolean). We call $\mathcal{E}_q$ the set of all equivalence classes on $\mathrm{Vars}(q)$ (which is of course finite), and for $\sim \in \mathcal{E}_q$ we let $q/\sim$ be the query in $\mathrm{CQ}^{\neq}$ obtained by choosing one representative variable in $\mathrm{Vars}(q)$ for each equivalence class of $\sim$ and mapping every $x \in \mathrm{Vars}(q)$ to the representative variable for the class of x (dropping in the result the useless existential quantifications on variables that do not occur anymore), and adding disequalities $x \neq y$ between each pair of the remaining variables.

We rewrite a CQ q to the UCQ $q' := \bigvee_{\sim \in \mathcal{E}_q} q/\sim$. We claim that for every bag-instance I, if $I \models q$ then $I \models q'$, which justifies that for an instance I'', considering the subinstances of I'', $W_K(q, I'') = W_K(q', I'')$. For the forward implication, assuming that $I \models q$, letting m be the witnessing match, we consider the $\sim_m$ relation defined by $x \sim_m y$ iff $m(x) = m(y)$, and it is easily seen that $I \models q/\sim_m$. For the backward implication, if $I \models q/\sim$ for some $\sim \in \mathcal{E}_q$, it is immediate that $I \models q$ with the straightforward match. Hence, using again Lemma C.4, it suffices to show the result for queries in $\mathrm{CQ}^{\neq}$ which include inequality axioms between all their variables. We call those *forced queries*.

We now show that the claim holds for forced queries. To see this, considering such a query q on signature σ, letting p be the sum of the multiplicities of all atoms in q (i.e., the number of atoms in the original CQ q), let σ_p be the signature obtained from σ by creating a relation R^i for $1 \leqslant i \leqslant p$, with arity $\mathrm{arity}(R)$, for every relation R of σ, and let q' be the rewriting of q obtained by replacing every atom $R(\mathbf{a})$ with multiplicity m by the disjunction $\bigvee_{m \leqslant j \leqslant p} R^j(\mathbf{a})$ (and keeping the inequalities), rewritten to a $\mathrm{UCQ}^{\neq}$. We now see q' as a $\mathrm{UCQ}^{\neq}$ in the usual sense (without multiplicities). We now claim that for any bag-instance I on σ where facts have multiplicity $\leqslant p$, letting I' be the set-instance obtained by replacing every fact $F = R(\mathbf{a})$ of I with multiplicity $m = I(F)$ by the fact $R^m(\mathbf{a})$, $I \models q$ iff $I' \models q'$. To see why, observe that, as q is a forced query, if q has a match m then every atom A of q must be mapped by m to a fact of I (written $m(A)$) and this mapping must be injective (because m is), so that the necessary

and sufficient condition is that $I(m(A)) \geqslant p_A$ (where p_A is the multiplicity of A in q) for every atom A of q; and this is equivalent to $I' \models q'$.

Now, q' is a UCQ$^{\neq}$ so it can be tested *in the sense of Definition 4.2* (as it is expressible in GSO, so we can apply Theorem 4.1); fix $k \in \mathbb{N}^*$ and let $A_{q'} = (Q, F, \iota, \delta)$ be a $\Gamma^k_{\sigma_p}$-bNTA that tests q' for width k. We build a $\overline{\Gamma^k_\sigma}^p$-bNTA $A_q = (Q, F, \iota', \delta')$ by relabeling $A_{q'}$ in the following way. Recall the definition of Γ^k_σ (Definition B.2). For every $((d, f), i) \in \overline{\Gamma^k_\sigma}^p$, set f' to be either f if $f = \emptyset$ and $f' = R^i(\mathbf{a})$ if $f = R(\mathbf{a})$, and set $\iota'(((d, f), i))$ to be $\iota((d, f'))$ and $\delta'(q_\mathrm{L}, q_\mathrm{R}, ((d, f), i))$ to be $\delta(q_\mathrm{L}, q_\mathrm{R}, (d, f'))$ for every $q_\mathrm{L}, q_\mathrm{R} \in Q$.

We now claim that A_q tests q for treewidth k. To see why, it suffices to observe that for any $\overline{\Gamma^k_\sigma}^p$-tree T, letting T' be the $\Gamma^k_{\sigma_p}$-tree obtained in the straightforward manner, then A_q accepts T iff $A_{q'}$ accepts T', which is immediate by construction. Now indeed, as we know that $A_{q'}$ accepts T' iff $\langle T' \rangle \models q'$ (as $A_{q'}$ tests q'), and (as immediately $\langle T' \rangle$ is the σ_p-instance corresponding to $\langle T \rangle$ as I' corresponds to I above) that $\langle T' \rangle \models q'$ iff $\langle T \rangle \models q$, we have the desired equivalence.

The only thing left is to observe that A_q does not only correctly test q on instances where each fact has multiplicity $\leqslant p$, but correctly tests q on all bag-instances. But this is straightforward: as q matches at most p fact occurrences in the instance I, we have $I \models q$ iff $I^{\leqslant p} \models q$. This concludes the proof. □

We are now ready to prove Theorem 5.3:

Proof. We show the proof for CQs, and then extend to UCQs.

Let $k \in \mathbb{N}$, $q : \exists \mathbf{x}\, q'(\mathbf{x})$ be the CQ. We rewrite q to $q'' : \exists \mathbf{x}\, q'(\mathbf{x}) \wedge \bigwedge_{x \in \mathbf{x}} P_x(x)$ for fresh unary predicates P_x. We apply Proposition C.2 to compile q'' to a $\overline{\Gamma^k_\sigma}^p$-bNTA A, where p is the number of atoms of q'', such that A tests q'' for treewidth k. We can clearly design a $\overline{\Gamma^k_\sigma}^p$-bNTA A' that checks on a $\overline{\Gamma^k_\sigma}^p$-tree whether, for all $x \in \mathbf{x}$, the input tree contains exactly one P_x-fact: this can be done with state space $2^{\mathbf{x}}$. We intersect A and A' to obtain a bNTA that recognizes all $\overline{\Gamma^k_\sigma}^p$-trees that satisfy the bag-query associated to q'' and have exactly one P_x-fact for all $x \in \mathbf{x}$, and determinize this bNTA to obtain an equivalent automaton A'' which is *deterministic*: if it has an accepting run then it has exactly one accepting run.

Let I be the input instance, and I' be the instance where we added one fact $P_x(a)$ for all $x \in \mathbf{x}$ and $a \in \mathrm{dom}(I)$: we call those the *additional facts*. We can clearly compute I' from I in linear time, and the treewidth is unchanged. Let $T_{I'}$ be a tree encoding of I', that is, a Γ^k_σ-tree.

We claim that we can construct, from A'', a bNTA A''' such that, for any valuation ν of $T_{I'}$ that gives multiplicity 1 to the additional facts, the number of accepting runs of A''' on $\nu(T_{I'})$ is the number of valuations ν' from the additional facts to $\{0, 1\}$ such that A'' accepts $\nu''(T_{I'})$, where ν'' follows ν' on nodes encoding additional facts and follows ν otherwise. We proceed as follows: first, duplicate the states of A'' so that every state q is replicated to two states q and q', q and q' being treated exactly the same way in terms of transitions in δ and in terms of being final (this preserves determinism). Now, ensure that for any two states q_1 and q_2 and labels $(\tau, 0)$ and $(\tau, 1)$ in $\overline{\Gamma^k_\sigma}^p$ that encode a present or absent additional fact, $\delta(q_1, q_2, (\tau, 0))$ and $\delta(q_1, q_2, (\tau, 1))$ are disjoint (as A'' is deterministic, those are single facts, so if they are the same fact, replace one of them by its equivalent copy). Now, modify the transitions of the automaton so that, for any states q_1 and q_2 and τ encoding an additional fact, $\delta(q_1, q_2, (\tau, 1))$ is $\delta(q_1, q_2, (\tau, 0)) \cup \delta(q_1, q_2, (\tau, 1))$. It is now clear that the resulting automaton A''' satisfies the desired property: for any valuation ν as above, there is a bijection between the accepting runs of A''' and the valuations ν' as above such that $\nu''(T_{I'})$ is accepted by A''.

We now apply Theorem C.1 with l the number of facts in the CQ q'' to obtain a $\mathbb{N}[X]$-circuit that captures the $\mathbb{N}[X]$-l-provenance of A''' on $T_{I'}$; and fix to 1 all inputs except those coding a fact of I (i.e., nodes coding additional facts are set to 1) and rename the remaining inputs to match the facts of I. Let l' be the number of facts in the original CQ q. Then the circuit captures:

$$\bigoplus_{\substack{J \sqsubseteq I \\ J \models q \\ \sum_{F \in \mathrm{supp}(J)} J(F) = l'}} \left|\{\, f : \mathbf{x} \to \mathrm{dom}(I) \mid J \models q'(f(\mathbf{x})) \,\}\right| \bigotimes_{F \in J} F^{J(F)}.$$

Now, $J \models q'(f(\mathbf{x}))$ just means $q'(f(\mathbf{x})) \subseteq J$ (as bag-instances), and as $q'(f(\mathbf{x}))$ and J have same total multiplicity, this actually means $J = q'(f(\mathbf{x}))$. Hence, the above is equal to:

$$\bigoplus_{\substack{f : \mathbf{x} \to \mathrm{dom}(I) \\ I \models q'(f(\mathbf{x}))}} \bigotimes_{A(\mathbf{x}) \in q'} A(f(\mathbf{x})).$$

and this is exactly $\mathrm{Prov}_{\mathbb{N}[X]}(q, I)$.

For UCQs, observe that the provenance we need to compute (Definition 5.2) is simply the sum of the provenance for each CQ. So we can just independently build a circuit for each CQ and combine the circuits into one (merging the input gates), while choosing as distinguished gate a $\oplus$-gate of each distinguished gate. ☐

C.4. Going beyond UCQs

Note that the proof of Theorem 5.3 implicitly relies on the fact that UCQs are bounded in the sense of Definition C.8, and we cannot hope to rewrite a query to a $\overline{\Gamma_\sigma^k}^p$-bNTA that sensibly tests it if it is not bounded. However, we can show:

Proposition C.3. *There is a guarded monadic Datalog query P whose associated bag-query q_P is not bounded.*

Proof. Consider the Datalog query P consisting of the rules $S(y) \leftarrow S(x), R(x, a, y), A(a)$, Goal $\leftarrow S(x), T(x)$. For all $n \in \mathbb{N}$, consider the instance $I_n = \{R(a_1, a, a_2), R(a_2, a, a_3), \ldots, R(a_{n-1}, a, a_n), S(a_1), T(a_n), A(a)\}$. It is easily verified that the only proof tree of P on I_n has $n - 1$ leaves with the fact $A(a)$. Hence, assuming that the bag-query q_P captured by P is bounded by p, considering the bag-instance J formed of the leaves of the sole proof tree of P on I_{p+2}, it is not the case that $J^{\leqslant p} \models q_P$, contradicting boundedness. ☐

D. Proofs for Section 6 (Applications)

D.1. Preliminaries

All proofs about probability evaluation in this section will use the notion of *cc-instances*, which we now introduce.

In this appendix, all Boolean circuits are non-monotone (i.e., they allow NOT-gates) and arity-two (Definition B.5), unless stated otherwise. We will first define the formalism of cc-instances, then state a result about the construction of circuits (the analogue of provenance circuits) for them, using Theorem 4.2, and finally explain how probability evaluation is performed using that result using message-passing. We conclude by presenting the similar formalism of *pc-instances*, and stating tractability results for them implied by the results on pcc-instances.

cc-instances. We define the formalism of *cc-instances*:

Definition D.1. *A* cc-instance *is a triple $J = (I, C, \varphi)$ of a relational σ-instance I, a (non-monotone arity-two) Boolean circuit C, and a mapping φ from the facts of I to gates of C. The* inputs J_{inp} *of J are C_{inp}. For every valuation ν of J_{inp}, the possible world $\nu(J)$ is the subinstance of I that contains the facts F of I such that $\nu(C)(\varphi(F)) = 1$, and, as for c-instances, $[\![J]\!]$ is the set of possible worlds of J.*

A pcc-instance is a 4-tuple $J = (I, C, \varphi, \pi)$ such that $J' = (I, C, \varphi)$ is a cc-instance (and $J_{\mathsf{inp}} := J'_{\mathsf{inp}}$) and $\pi : J_{\mathsf{inp}} \to [0, 1]$ gives a probability to each input. As for pc-instances, the probability distribution $[\![J]\!]$ has universe $[\![J']\!]$ and probability measure $\Pr_J(I') = \sum_{\nu \mid \nu(J)=I'} \Pr_J(\nu)$ with the product distribution:

$$\Pr_J(\nu) = \prod_{\substack{g \in J_{\mathsf{inp}} \\ \nu(g)=1}} \pi(g) \prod_{\substack{g \in J_{\mathsf{inp}} \\ \nu(g)=0}} (1 - \pi(g)).$$

We define relational encodings and treewidth for cc-instances:

Definition D.2. *Let $\sigma_{\mathrm{Circuit}}$ be the signature of the relational encoding of Boolean circuits (Definition B.6). Let σ be a signature and σ^+ be the signature with one relation R^+ of arity $\mathrm{arity}(R)+1$ for every relation R of σ. The* relational encoding I_J *of a cc-instance $J = (I, C, \varphi)$ over signature σ, is the $(\sigma_{\mathrm{Circuit}} \sqcup \sigma^+)$-instance containing both the $\sigma_{\mathrm{Circuit}}$-instance I_C encoding C and one fact $R^+(\mathbf{a}, \varphi(F))$ for every fact $F = R(\mathbf{a})$ in I.*

A tree decomposition *of a cc-instance J is a tree decomposition of I_J. Tree decompositions of pcc-instances are defined as a tree decomposition of the corresponding cc-instance (the probabilities are ignored).*

Circuits for cc-instances. We claim the following result about cc-instances, intuitively corresponding to the provenance circuits of Section 4 for them (combined with their circuit annotation):

Theorem D.1. *For any fixed integer k and GSO sentence q, one can compute in linear time, from a cc-instance J with $\mathrm{w}(J) \leqslant k$, a Boolean circuit C on J_{inp} such that for every valuation ν of J_{inp}, $\nu(C) = 1$ iff $\nu(J) \models q$, with $\mathrm{w}(C)$ depending only on k and q.*

We now prove this result, explaining later what it implies in terms of probability evaluation. We first introduce the notion of *cc-encoding*. Recall the definition of tree decompositions of circuits (Definition B.6):

Definition D.3. *A* cc-encoding *of width k is a tuple $E' = (E, C, T, \chi)$ of a Γ^k_σ-tree E of width k, a Boolean circuit C, a tree decomposition T of C of width k with same skeleton as E, and a mapping $\chi : T \to C$ selecting a* selected gate *such that $\chi(b) \in \mathrm{dom}(b)$ for all $b \in T$. The* inputs E'_{inp} *of E' are C_{inp}.*

Given a valuation ν of C_{inp}, we extend it to an evaluation of C, and see it as a Boolean valuation of E by setting $\nu(n) := \nu(\chi(b))$ for the bag b of T corresponding to n in E, and write $\nu(E')$ the resulting $\overline{\Gamma^k_\sigma}$-tree.

First, we explain how we can compute a cc-encoding of our cc-instance $J = (I, C, \varphi)$ by "splitting" its tree decomposition T in a tree decomposition of C and a Γ^k_σ-tree E of I with same skeleton, with χ keeping track of the gate of C to which each node $n \in E$ was mapped by φ. Formally:

Lemma D.1. *Recall the definition of ϵ (Definition B.7). Given a cc-instance $J = (I, C, \varphi)$ and a tree decomposition T of J of width k, one can compute a cc-encoding $E' = (E, C', T', \chi)$ of width k, with $C = C'$, such that for any valuation ν of C_{inp}, $\epsilon(\nu(E'))$ is an encoding of $\nu(J)$. The computation is in $O(|T| + |C|)$.*

Proof. We process the tree decomposition T of J to construct E and T'. We adapt the encoding construction described in Lemma B.1.

Whenever we process a bag $b \in T$, the mapping precomputed with J (see Lemma B.1) is used to obtain all facts F of I for which b is the topmost node where domain $\mathrm{dom}(F) \subseteq \mathrm{dom}(b)$ and $\varphi(F) \in \mathrm{dom}(b)$.

For every such fact F, we create one bag b' in T' labeled with all elements of $\mathrm{dom}(b)$ that are gates of G, and one node n in E which is the encoding of F (considering only the domain $\mathrm{dom}(b) \cap \mathrm{dom}(I)$) as for a normal relational instance. Set the selected gate $\chi(b') := \varphi(F)$ (which is in $\mathrm{dom}(b')$ by the condition according to which we chose to consider fact F).

Because T was a tree decomposition of J, it is immediate that the resulting tree T' is indeed a tree decomposition of width k of C and that E is a tree encoding of width k of I. By construction T' and E have same skeleton, and clearly the process is in linear time in $|T| + |J|$. We let $E' = (E, C', T', \chi)$.

It remains to check the last condition. Consider a Boolean valuation ν of the inputs of C. Consider the instance $\nu(J)$ and its tree decomposition derived from T. It is clear that when one computes a tree encoding of $\nu(J)$ following T, one obtains an encoding E'' which is exactly E except that the facts have been removed from the nodes which used to encode in E a fact that was removed from $\nu(J)$. Hence, E'' is exactly $\epsilon(\nu(E'))$. This concludes the proof. □

Second, we show the lemmas that will allow us to "glue together" the circuit C of the cc-instance, which annotates the cc-encoding, with a provenance circuit for an automaton on the tree encoding.

Definition D.4. *Let $C = (G, W, g_0, \mu)$ and $C' = (G', W', g'_0, \mu')$ be circuits such that $G \cap G' = C'_{\mathsf{inp}}$ (we say that C and C' are* stitchable*). The* stitching *of C and C', denoted $C \circ C'$, is the circuit $(G \cup G', W \cup W', g'_0, \mu'')$ where $\mu''(g)$ is defined according to μ for $g \in G$ and according to μ' otherwise. In particular, $(C \circ C')_{\mathsf{inp}} = C_{\mathsf{inp}}$.*

In the following lemmas about stitching, for clarity, we distinguish valuations ν to the inputs of a circuit and the evaluation on all circuit gates, which we write $\nu(C)$ where C is the circuit.

The fundamental property of stitching is:

Lemma D.2. *For any stitchable circuits C and C', for any gate g of C' and valuation ν of C_{inp}, letting ν' be the restriction of $\nu(C)$ to C'_{inp}, we have: $\nu'(C')(g) = \nu(C \circ C')(g)$.*

Proof. Fix C, C', g, and ν. As C and $C \circ C'$ share the same inputs, ν is a valuation for both of them. Now, first note that for any gate g of C, $\nu(C)(g) = \nu(C \circ C')(g)$. Hence, in particular, for any input gate g of C', as it is a gate of C because C and C' are stitchable, we have $\nu(C \circ C')(g) = \nu(C)(g) = \nu'(g)$. As this equality holds for any input gate g of C', it inductively holds for any gate of C', which proves the result. □

We show that a tree decomposition for $C \circ C''$ can be obtained from two tree decompositions T and T'' for C and C'' that have same skeleton, as the *sum* $T + T''$ with same skeleton where each bag b'' of $T + T'$ is the union of the corresponding bags b and b' in T and T'. Namely:

Definition D.5. *Given two tree decompositions T and T' with same skeleton, the* sum *of T and T' (written $T + T'$) is the tree decomposition T with same skeleton where every bag b'' is the union of the corresponding bags b and b' in T and T'.*

The following is immediate:

Lemma D.3. *Given two tree decompositions with same skeleton T and T' of fixed width k and k' for a Boolean circuit C and a Boolean circuit C', $T + T'$ can be computed in linear time in T and T' and has width $\leqslant k + k' + 1$.*

We now show:

Lemma D.4. *Let C and C' be stitchable circuits with tree decompositions T and T' with same skeleton (with witnessing bijection ψ). Assume that for any $g \in C'_{\mathsf{inp}}$ and bag b of T' with $g \in \mathrm{dom}(b)$, we have $g \in \mathrm{dom}(\psi^{-1}(b))$. Then $T + T'$ is a tree decomposition of $C \circ C'$.*

Proof. We consider $I_{C \circ C'}$ and show that $T + T'$ is a tree decomposition of it:

- Let g be a gate of $C \circ C'$. If g is not a gate of $C \cap C'$, then its occurrences in $T + T'$ are only its occurrences in T or in T', so that they form a connected subtree of $T + T'$ as they did in T or T'. If it is a gate of $C \cap C'$, then it is an input gate of C' because C and C' are stitchable, and by the hypothesis, its occurrences in T' are a subset of its occurrences in T, so its occurrences in $T + T'$ are its occurrences in T, and they also form a connected subtree.
- Let $\mathbf{g}$ be a tuple occurring in a fact of $I_{C \circ C'}$. Clearly $\mathbf{g}$ occurs either in I_C or in $I_{C'}$, so that it is covered by the bag $b_{\mathbf{g}}$ that covers all elements of $\mathbf{g}$ in T or in T'. □

Last, we conclude the proof of Theorem D.1:

Proof. Let $k \in \mathbb{N}$, q be the GSO sentence, and let A be a Γ^k_σ-bNTA that tests q for treewidth k according to Theorem 4.1, which we lift to a $\overline{\Gamma^k_\sigma}$ in the same way as in the proof of Theorem 4.2.

Construct in linear time in the input cc-instance $J = (I, C, \varphi)$ a tree decomposition of J of width $\leqslant k$, and a cc-encoding $E' = (E, C, T, \chi)$ of width k of J, according to Lemma D.1, satisfying the conditions of that lemma. Now, use Theorem 4.2 to compute an arity-two provenance circuit C' of A on E and with a tree decomposition T' whose width is constant in I. We further observe from the proof of the proposition that T' that has same skeleton as E (and T), and that for any node $n \in E$, the input gate for this node is in the bag corresponding to n in T'.

We then observe that C' and C are stitchable circuits, and that their tree decompositions T' and T have same skeleton and satisfy the conditions of Lemma D.4. We deduce from this lemma and Lemma D.3 that we can construct in linear time the stitching $C'' := C \circ C'$ and a tree decomposition of it, whose width does not depend on J. We now show that C'' satisfies the desired property, namely, $\nu(C'')$ is 1 iff $\nu(J) \models q$. For any valuation ν of J_{inp}, we have $\nu(C \circ C') = \nu'(C')$, by Lemma D.2, where ν' is the valuation of C'_{inp} obtained from $\nu(C)$. It is clear by definition of $\nu(J)$ that a fact F is present in $\nu(J)$ iff $\varphi(F)$ is true in $\nu(C)$. We conclude using the fact that ν' is a provenance circuit: $\nu'(C')$ holds iff $\{F \in I \mid \varphi(F) \text{ true in } \nu(C)\} \models q$. □

Probability evaluation. We now describe the consequences of Theorem D.1 in terms of probability evaluation. Here is what we want to show:

Corollary D.1. *The problem of computing the probability of a fixed GSO sentence on bounded-treewidth pcc-instances can be solved in ra-linear time data complexity.*

To prove this corollary, we need the following definition and key result:

Definition D.6. *Let $C = (G, W, g_0, \mu)$ be an (arity-two non-monotone) Boolean circuit and π be a* probabilistic valuation *of C associating each $g \in C_{\mathsf{inp}}$ to a probability distribution π_g on $\{0,1\}$, that is, one rational $v_0 = \pi_g(0)$ and one rational $v_1 = \pi_g(1)$ such that $v_0 + v_1 = 1$. The* probability evaluation problem *for C and π is to compute the probability distribution of g_0 under the product distribution for the inputs (i.e., assuming independence), that is, $\Pr_{g_0}$ mapping $v \in \{0,1\}$ to*

$$\sum_{\substack{\nu \in \mathrm{Val}(C_{\mathsf{inp}}) \\ \nu(g_0)=v}} \prod_{g \in C_{\mathsf{inp}}} \pi_g(\nu(g))$$

where $\mathrm{Val}(C_{\mathsf{inp}})$ denotes the set of Boolean valuations of C_{inp}.

Theorem D.2. *Given a tree decomposition T of width k of an arity-two Boolean circuit C, and given a probabilistic valuation π of C, the probability evaluation problem for C and π can be solved in time* ra-linear *in $2^k\,|T| + |\pi| + |C|$.*

With the above theorem, we can prove Corollary D.1 as follows:

Proof. Let $J = (I, C, \varphi, \pi)$ be a pcc-instance of treewidth k and q a query. We use Theorem D.1 to construct in linear time a Boolean circuit C' of treewidth k' dependent only on k and q, with distinguished gate g. We build from C' a tree decomposition of width k' in linear time. The probability that q is true in J is $\Pr_g(1)$. We conclude as Theorem D.2 states that this can be computed in ra-linear time in $|C'| + |\pi|$ for fixed k'. □

We now prove Theorem D.2.

Proof. Fix $T = (B, L, R, \mathrm{dom})$ a tree decomposition of a Boolean circuit $C = (G, W, g_0, \mu)$ (so that for any $b \in B$, $\mathrm{dom}(b)$ is a set of gates of G). We define $E := L \cup R$ and, for $g \in G$, $V(g)$ the value set of g. For $e = (b_1, b_2) \in E$, we define $\mathrm{dom}(e) := \mathrm{dom}(b_1) \cap \mathrm{dom}(b_2)$, the shared elements between a bag and its parent. We assume an arbitrary order $<$ over G and see $\mathrm{dom}(b)$ as a tuple by ordering elements of $\mathrm{dom}(b)$ with $<$ (this ordering taking constant time as the size of bags is bounded by a constant). If $\mathrm{dom}(b) = (g_1, \ldots, g_m)$, we note $V(b) = \{0,1\}^m$ (and similarly, for $e \in E$, $V(e)$ is the product over $\mathrm{dom}(e)$). For every $g \in G$, let $\beta(g) \in B$ be an arbitrary bag containing g and all gates that are inputs of g, that is, all gates g' such that $(g', g, i) \in W$ for some W: such a bag exists by definition of the tree decomposition of circuits (there is a fact in I_C regrouping g and the g') and we can precompute such a function in linear time by a traversal of T. In particular, if g is an input gate, then $\beta(g)$ is an arbitrary bag containing just g.

We associate to every bag $b \in B$ (resp., every edge $e \in E$) a *potential function* $\Phi^b : V(b) \to \mathbb{Q}^+$ (resp., $\Phi^e : V(e) \to \mathbb{Q}^+$), where $\mathbb{Q}^+$ denotes the nonnegative rational numbers, initialized to the constant 1 function. We will store for each bag and each edge the full table of values of Φ^e, i.e., at most 2^k values, each of which has size bounded by $|\pi|$.

The functions π_g for $g \in C_{\mathsf{inp}}$ are mappings from $V(g)$ to $\mathbb{R}^+$. For a bag $b \in B$ with $g \in \mathrm{dom}(b)$, we define π_g^b as the function that maps every tuple $\mathbf{d} \in V(b)$ to $\pi_g(d')$ where d' is the value assigned to g in $\mathbf{d}$.

For g a non-input gate, let $\kappa(g)$ be the tuple formed of g and all gates with a wire to g, ordered by $<$. Let $f := \mu(g)$ be the function of g, in $\{\neg, \vee, \wedge, 0, 1\}$. We see f as a subrelation

R_g of $V(\kappa(t))$ (the table of values of the function, with columns reordered by applying $<$ on g), that is, a set of $(\text{arity}(f)+1)$-tuples which represents the graph of the function.

We update the potential function by the following steps, where the product of two functions f and f' which have same domain D denotes pointwise multiplication, that is, $(f \times f')(x) = f(x) \times f'(x)$ for all $x \in D$:

1. For every $g \in C_{\mathsf{inp}}$, we set $\Phi^{\beta(g)} := \Phi^{\beta(g)} \times \pi_g^{\beta(g)}$.
2. For every $g \in G \backslash C_{\mathsf{inp}}$, we set $\Phi^{\beta(g)}(\mathbf{d}) := 0$ if the projection of $\mathbf{d}$ onto $\kappa(g)$ is not in R_g, we leave $\Phi^{\beta(g)}(t)$ unchanged otherwise.

Note that we have now initialized the potential functions in a way which exactly corresponds to that of [HD96], for a straightforward interpretation of our circuit with probabilistic inputs as a special case of a belief network where all non-root nodes are deterministic (i.e., have a conditional distribution with values in $\{0,1\}$).

We now apply as is the Global Propagation steps described in Section 5.3 of [HD96]: if we choose the root of the tree decomposition as the root cluster $\mathbf{X}$, this consists in propagating potentials from the leaves of the tree decomposition up to the root, then from the root down to the leaves of the tree. This process is linear in $|T|$ and, at every bag of T, requires a number of arithmetic operations linear in 2^k.

As shown in [LS88, HD96], at the end of the process, the desired probability distribution $\Pr_g$ for gate g can be obtained by marginalizing $\Phi^{\beta(g)}$:

$$\Pr_g(d') = \sum_{\substack{\mathbf{d} \in V(\beta(g)) \\ d_k = d'}} \Phi^{\beta(g)}(\mathbf{d})$$

where k is the position of g in $\mathrm{dom}(\beta(g))$.

The whole process is linear in $|T| \times 2^k + |C| + |\pi|$ under fixed-cost arithmetic; under real-cost arithmetic, belief propagation requires multiplying and summing linearly many times $O(|T| \times 2^k)$ probability values, each of size bounded by $|\pi|$, which is polynomial-time in $|T|$, 2^k, $|\pi|$. □

Consequences for pc-instances. We define existing the formalism of *(p)c-instances* [SORK11], which is analogous to (p)cc-instances, but annotates facts with propositional formulae rather than circuits:

Definition D.7 [HAKO09,GT06]. *A* c-instance J *is a relational instance where each tuple is labeled with a propositional formula of variables (or* events*) from a fixed set* X*. For a valuation* ν *of* X *mapping each variable to* $\{0,1\}$*, the possible world* $\nu(J)$ *is obtained by retaining exactly the tuples whose annotation evaluates to* 1 *under* ν*;* $[\![J]\!]$ *is the set of all these possible worlds. Observe that different valuations may yield the same possible world.*

A pc-instance $J = (J', \pi)$ *is defined as a c-instance* J' *and a* probabilistic valuation $\pi : X \to [0,1]$ *for the variables used in* J'*. Like all probabilities in this paper, the values of* π *are rationals. The probability distribution* $[\![J]\!]$ *defined by* J *has universe* $[\![J']\!]$ *and probability measure* $\Pr_J(I) := \sum_{\nu | \nu(J') = I} \Pr_J(\nu)$ *with the product distribution on valuations:*

$$\Pr_J(\nu) := \prod_{\substack{x \in X \\ \nu(x)=1}} \pi(x) \prod_{\substack{x \in X \\ \nu(x)=0}} (1 - \pi(x)).$$

We define a notion of treewidth for them:

Definition D.8. *Let* $\sigma^{\mathrm{o}} = \sigma \cup \{\mathrm{Occ}, \mathrm{Cooc}\}$*, where* Occ *and* Cooc *have arity two. From a pc-instance* J*, we define the* relational encoding I_J *of* J *as the* σ^{o}*-instance where each event* e *of* J *is encoded to a fresh* $a_e \in \mathrm{dom}(J)$*, and where we add a fact* $\mathrm{Occ}(a, a_e)$ *in* I_J *whenever* $a \in \mathrm{dom}(J)$ *is used in a fact annotated by a formula involving* e*, and* $\mathrm{Cooc}(a_e, a_f)$ *whenever events* e *and* f *co-occur in the formula of some fact.*

The treewidth $\mathrm{w}(J)$ *of a (p)c-instance* J *is* $\mathrm{w}(I_J)$*.*

This notion of treewidth, through event (co-)occurrences, can be connected to treewidth for (p)cc-instances, to ensure tractability of query evaluation on (p)c-instances of bounded treewidth in that sense. A technicality is that we must first rewrite annotations of the bounded-treewidth (p)c-instance to bound their size by a constant; but we can show:

Proposition D.1. *For any fixed* k*, given a (p)c-instance* J *of width* $\leqslant k$*, we can compute in linear time a (p)cc-instance* J *which is equivalent (has the same possible worlds with the same probabilities) and has treewidth depending only on* k*.*

Proof. We first justify that we can compute in linear time from J (p)c-instance J' with the same events such that for any valuation ν, we have $\nu(J) = \nu(J')$ (and $\mathrm{Pr}_J(\nu) = \mathrm{Pr}_{J'}(\nu)$), and the annotations of J' have size depending only on k.

Indeed, we observe that by our assumption that $\mathrm{w}(J) \leqslant k$, for any formula F in an annotation, the number of distinct events occurring in F is at most k. Indeed, there is a Cooc clique between these events in I_J, so that as $\mathrm{w}(I_J) \leqslant k$ (by Lemma 1 of [Gav74]) there must be less than k of them.

Now, we observe that any formula in J can be rewritten, in linear time in this formula for fixed k, to an equivalent formula whose size depends only on k. Indeed, for every valuation of the input events, which means at most 2^k valuations by the above, we can evaluate the formula in linear time; then we can rewrite the formula to the disjunction of all valuations that satisfy it, each valuation being tested as the conjunction of the right events and negation of events. So this overall process produces in linear time an equivalent (p)c-instance J' where the annotation size depends only on k. So we can assume without loss of generality that the size of the annotations of J is bounded by a constant.

Consider now the (p)c-instance J, its relational encoding I_J, and a tree decomposition T of I_J. We build a tree decomposition T' of a relational encoding I_J of a cc-instance $J' = (I, C, \varphi)$ designed to be equivalent to J. Start by adding to C the input gates, which correspond to the events of J.

Now, consider each fact $F = R(\mathbf{a})$ of J. Let $\mathbf{e}$ be the set of events used in the annotation A_F of F. Note that every pair of $S = \mathbf{a} \sqcup \mathbf{e}$ co-occurs in some fact of I_J: the elements of $\mathbf{a}$ co-occur within F, the elements of $\mathbf{e}$ co-occur in a Cooc fact, and any pair of elements from $\mathbf{a}$ and $\mathbf{e}$ co-occur in some Occ fact. Hence, by Lemma 1 of [Gav74], there is a bag $b_F \in T$ such that $S \subseteq \mathrm{dom}(b)$.

Let C_F be a circuit representation of the Boolean function A_F on E, whose size depends only on k. Add C_F to C, add F to I, and set $\varphi(F)$ to be the distinguished node of C_F. We have thus built J', which by construction is equivalent to J.

We now build T' by making it a copy of T. Now, for each fact F, considering its bag b_F, and b'_F the corresponding bag in T', we add all elements of C_F to b'_F. This decomposition clearly covers all facts of $I_{J'}$, and event occurrences form subtrees because they do in T and the elements that we added to T' are always in a single bag only. Last, it is clear that the bag size depends only on k, as the size of the C_F added to the bags depends only on k, and at most

k of them are added to each bag (because there are at most k elements per bag).

We have not talked about probabilities, but clearly if J is a pc-instance the probabilities of the inputs of the pcc-instance J' should be defined analogously. □

We can now combine the above with Theorem D.1, and deduce the tractability of query evaluation on bounded-treewidth pc-instances.

Theorem D.3. *For bounded-treewidth pc-instances, the probability query evaluation problem for Boolean MSO queries can be solved in ra-linear time data complexity.*

Proof. The result is an immediate consequence of Proposition D.1 and Theorem D.1 as long as we show that, for any fixed $k \in \mathbb{N}^*$, and for every (p)c-instance J of width $\leqslant k$, one can compute in linear time a (p)c-instance J' with the same events such that for any valuation ν, we have $\nu(J) = \nu(J')$ (and $\Pr_J(\nu) = \Pr_{J'}(\nu)$), and the annotations of J' have size depending only on k.

Fix k and J. We observe that by our assumption that $\mathrm{w}(J) \leqslant k$, for any formula Φ in an annotation, the number p_Φ of distinct events occurring in Φ is at most k. Indeed, there is a Cooc clique between these events in I_J and each of them is connected by the Occ relation to domain elements of the fact F annotated by Φ (there is at least one), so we have in total a $(p_\Phi+1)$-clique. By Lemma 1 of [Gav74], any tree decomposition must have one node containing all these $p_\Phi + 1$ elements, and therefore $p_\Phi \leqslant k$.

Now, we observe that any formula in J can be rewritten, in linear time in this formula for fixed k, to an equivalent formula whose size depends only on k. Indeed, for every valuation of the input events, which means at most 2^k valuations by the above, we can evaluate the formula in linear time; then we can rewrite the formula to the disjunction of all valuations that satisfy it, each valuation being tested as a conjunction of at most k literals. So this overall process produces in linear time an equivalent (p)c-instance where the annotation size depends only on k. □

D.2. Probabilistic XML

We will first prove the result on scopes (Proposition 6.1) and then prove the result on local models (Theorem 6.1).

XML and instances. We first describe XML documents and their connections to relational models.

Definition D.9. *An* XML document *with* label set Λ *(or* Λ-document*) is an* unranked Λ*-tree.*

Definition D.10. *A* PrXML*-tree T is an* unranked Λ*-tree, for a fixed alphabet Λ of* labels*, augmented with a set of Boolean events $\mathcal{E}$ where each event e_x has a probability $0 \leqslant p_x \leqslant 1$, and where each edge of the tree is labeled by a propositional formula over $\mathcal{E}$.*

We see T as defining a probability distribution over Λ-trees in the following fashion: for every valuation ν over $\mathcal{E}$, the possible world $\nu(T)$ is obtained by removing all edges whose annotation evaluates to false under ν, and all their descendent nodes and edges. The probability of a Λ-tree T' according to T is the sum of the probability of all valuations ν such that $\nu(T) = T'$, where the probability of a valuation is defined assuming that the events in $\mathcal{E}$ are drawn independently with their indicated probability.

Definition D.11. *To perform* query evaluation *on a* PrXML *document is to determine, for a fixed query over Λ-trees, given an input* PrXML *document T, what is the total probability of its possible worlds that satisfy q; we study its* data complexity*, i.e., its complexity as a function of T.*

We always assume that the label set Λ is fixed (not provided as input). As XML documents are unranked, it is often more convenient to manipulate their binary left-child-right-sibling representation:

Definition D.12. *The* left-child-right-sibling *(LCRS) representation of an unranked rooted ordered Λ-tree T is the following Λ-tree T': a node n whose children are the ordered sequence of siblings $n_1, \ldots, n_k$ is encoded as the node n with $L(n) = n_1$, $R(n_1) = n_2$, ..., $R(n_{k-1}) = n_k$; we complete by nodes labeled $\bot \notin \Lambda$ to make the tree full.*

We now define how XML documents can be encoded to the relational setting.

Definition D.13. *Given a Λ-document D, let σ_Λ be the relational signature with two binary predicates FC and NS (for "first child" and "next sibling"), and unary predicates P_λ for every $\lambda \in \Lambda$. The* relational encoding *I_D of D is the-σ_Λ instance with $\mathrm{dom}(I_D) = \mathrm{dom}(D)$, such that:*

- *for any consecutive siblings (n, n'), $NS(n, n')$ holds;*
- *for every pair (n, n') of a node $n \in D$ and its first child $n' \in D$ following sibling order, $FC(n, n')$ holds;*
- *for every node $n \in D$, the fact $P_{\lambda(n)}(n)$ holds.*

Lemma D.5. *The relational encoding I_D of an XML document D has treewidth 1 and can be computed in linear time.*

Proof. Immediate: the relational encoding is clearly computable in linear time and there is a width-1 tree decomposition of the relational encoding that has same skeleton as the LCRS representation of the XML document. □

Importantly, the language of MSO queries on XML documents [NS02], which we now define formally, can be easily translated to queries on the relational encoding:

Definition D.14. *An* MSO query *on XML documents is a MSO formula where first-order variables refer to nodes and where atoms are $\lambda(x)$ (x has label λ), $x \to y$ (x is the parent of y), and $x < y$ (x and y are siblings and x comes before y).*

Lemma D.6. *For any MSO query q on Λ-documents, one can compute in linear time an MSO query q' on σ_Λ such that for any Λ-document D, $D \models q$ iff $I_D \models q'$.*

Proof. We add a constant overhead to q by defining the predicates $\lambda(x)$ for $\lambda \in \Lambda$ as $P_\lambda(x)$, the predicate $x < y$ to be the transitive closure of NS ($\neg(x = y) \wedge \forall S(x \in S \wedge (\forall z z'(z \in S \wedge NS(z, z')) \Rightarrow z' \in S) \Rightarrow y \in S)$), and the predicate $x \to y$ to be $\exists z, FC(x, z) \wedge (z = y \vee z < y)$. It is clear that the semantics of those atoms on I_D match that of the corresponding atoms on D, so that a straightforward structural induction on the formula shows that q' satisfies the desired properties. □

Definition D.15. *Given label set Λ, we say that an XML document D on $\Lambda \sqcup \{\bot, \mathrm{det}\}$ is a* sparse representation *of an XML document D' on Λ if the root is labeled with an element of Λ, and the XML document obtained from D by removing every $\bot$ node and their descendants, and replacing every* det *node by the collection of its children, in order, is exactly D'.*

We say that a $\sigma_{\Lambda \sqcup \{\text{det}\}}$ instance I is a weak relational encoding *of an XML document D with label set Λ if there exists a sparse representation D' of D such that I is the relational encoding of D' except that $P_\bot$ facts are not written.*

Proposition D.2. *For any MSO query q on XML documents with (fixed) label set Λ, one can compute in linear time an MSO query q' on σ_Λ such that for any XML document D on label set Λ, if $D \models q$ then $I \models q'$ for any weak relational encoding I of D; and conversely if $D \not\models q$ then $I \not\models q'$ for any weak relational encoding I of D.*

Proof. We show that, for any MSO query q on XML documents with (fixed) label set Λ, one can compute in linear time an MSO query q' on documents with label in $\Lambda \sqcup \{\bot, \text{det}\}$ such that for any XML document D on label set Λ, if $D \models q$ then $D' \models q'$ for any sparse representation D' of D; and conversely if $D \not\models q$ then $D' \not\models q'$ for any sparse representation D' of D. The result then follows by Lemma D.6.

We call *regular* the nodes with label in Λ. Consider a document D and sparse representation D' of D with a mapping f from D to D' witnessing that D' is a sparse representation of D. Let us consider a node $n \in D$ with children $n_1, \ldots, n_k$ in order, and determine what is the relationship between $f(n)$ and the $f(n_i)$ in D'.

It is straightforward to observe that $f(n)$ is regular and the $f(n_i)$ are topmost regular descendants of $f(n)$ in D'; and for $i < j$, there is some node n' in D' (intuitively, their lowest common ancestor, which is a descendant of $f(n)$, possibly $f(n)$ itself) such that n' is both an ancestor of $f(n_i)$ and $f(n_j)$, n' is a descendant of $f(n)$, and n' has two children n'_1 and n'_2 such that $f(n_i)$ is a descendant of n'_1 (maybe they are equal), $f(n_j)$ is a descendant of n'_2 (maybe they are equal), and $n'_1 < n'_2$ in D'. Note that n', n'_1 and n'_2 are not necessarily regular nodes of D but can be det nodes. In addition, no $\bot$ node can be traversed in any of the ancestor–descendant chains discussed in this paragraph.

It is now clear that we can have MSO predicates $\to'$ and $<'$ in D' following these informal definitions (and not depending on D or D'), defined from predicates $\to$, $<$ and $\lambda(\cdot)$ on D', such that for every D and sparse encoding D' of D, for every nodes $n, n' \in D$, we have $n \to n'$ in D iff $f(n) \to f(n')$ in D' (which should only hold between regular nodes, so nodes in the image of f), and likewise for $<$. Last, it is clear that the predicates $\lambda(\cdot)$ of D can be encoded directly to the same predicates in D'. ☐

Probabilistic XML. We formally introduce probabilistic XML. We start by $\mathsf{PrXML}^{\mathsf{fie}}$, i.e., PrXML with events.

Definition D.16. *A $\mathsf{PrXML}^{\mathsf{fie}}$ probabilistic XML document $D = (D', \pi)$ is a $(\Lambda \sqcup \{\mathsf{fie}\})$-document D' where edges from* fie *nodes to their children are labeled with a propositional formula over some set of Boolean events X, and a probabilistic valuation π mapping each $e \in X$ used in D to an independent probability $\pi(e) \in [0, 1]$ of being true.*

The semantics $[\![D]\!]$ of D is obtained by extending π to a probability distribution on valuations ν of X as usual, and defining $\nu(D)$ for ν to be D' where all fie *nodes are replaced by the collection of their children with edge annotation Φ such that $\nu(\Phi) = 1$ (the others, and their descendants, are discarded). We require the root to have label in Λ.*

We will prove Proposition 6.1 via an encoding of $\mathsf{PrXML}^{\mathsf{fie}}$ to pc-instances:

Definition D.17. *The* pc-encoding *of a $\mathsf{PrXML}^{\mathsf{fie}}$ document $D = (D', \pi)$ in $\Lambda \sqcup \{\mathsf{fie}\}$ is the pc-instance $J_D = (J'_D, \pi')$ with same events, $\pi' = \pi$, and where the c-instance J'_D is the relational*

encoding of D' with the following annotations. NS- and FC-facts are annotated with 1. *$P_\lambda(n)$-facts are annotated with the annotation Φ of the edge from the parent of n to n, if Φ exists, with* 1 *otherwise.*

Proposition D.3. *For any MSO query q on Λ-documents, one can compute in linear time an MSO query q' on σ_Λ such that for any $\mathsf{PrXML}^{\mathsf{fie}}$ XML document D, for any valuation ν of D, letting ν' be the corresponding valuation of J_D, we have that $\nu(D) \models q$ iff $\nu'(J_D) \models q'$.*

Proof. We prove that for any valuation ν of D, letting ν' be the corresponding valuation of J_D, we have that $\nu'(J_D)$ is a weak encoding of $\nu(D)$ (we see P_{fie} facts in $\nu'(J_D)$ as if they were P_{det} facts). The result then follows by Proposition D.2.

We first show that for any valuation ν of D and corresponding valuation ν' of J_D, for every $\lambda \in \Lambda$, n is a node of $\nu'(J_D)$ that is retained in the XML document $\nu'(J_D)$ is a sparse representation of iff n is a node which is retained in $\nu(D)$, with same labels. Indeed, for the forward implication, observe that any fact $P_\lambda(n)$ is created for node n with label λ in n, and it is retained if and only if all its regular ancestors are retained and the annotation of its parent edge in $\nu(D)$ evaluates to 1; conversely, if n has label λ in D then a fact $P_\lambda(n)$ was created in I and if n is retained in $\nu(D)$ then all the conditions on edges in the chain from n to the root evaluate to 1 so $P_\lambda(n)$ does hold and n is retained in $\nu'(J_D)$.

We further know that by construction relations FC and NS correspond to the first-child and next-sibling relations in D no matter the valuation.

So we deduce that J_D is the relational encoding of the XML document obtained from D by replacing all nodes not kept in $\nu(D)$ by $\perp$ nodes, and removing all edge annotations. □

Observe that in this definition of pc-encoding, it is *not* the case that the possible worlds of J_D are the relational encodings of the possible worlds of D. For instance, the fie nodes are retained as is, and FC- and NS-facts are always retained even if the corresponding nodes are dropped. The following example shows that it would not be reasonable to ensure such a strong property:

Example D.1. *Consider an* fie *node with k children $n_1, \ldots, n_k$, all annotated with independent events with probability* 1/2. *In a straightforward attempt to encode this node and its descendants to a pc-instance J (or even to a pcc-instance J), we would create one domain element e_i for each of the n_i. But then we would need to account for the fact that, as any pair n_i, n_j may be retained individually, the fact $NS(e_i, e_j)$ would need to occur in a possible world of J, and thus would also occur in J. So this naïve attempt to ensure that the possible worlds of J are exactly the relational encodings of the possible worlds of D leads to a pcc-instance of quadratic size and linear treewidth.*

Tractability for $\mathsf{PrXML}^{\mathsf{fie}}$. Of course we cannot hope that the pc-encoding of a $\mathsf{PrXML}^{\mathsf{fie}}$ document always has constant treewidth for it is known that for $\mathsf{PrXML}^{\mathsf{fie}}$, evaluating MSO queries is almost always $\#P$-hard ([KKS08], Theorem 5.2). A first notion of tractability for a $\mathsf{PrXML}^{\mathsf{fie}}$ document D is the treewidth (following Definition D.8) of the pc-encoding of D. Indeed, Proposition D.3 and Theorem D.3 imply the following:

Corollary D.2. *For $\mathsf{PrXML}^{\mathsf{fie}}$ documents with bounded-treewidth pc-encoding, the MSO probabilistic query evaluation problem can be solved in ra-linear time data complexity.*

The condition on event scopes is a simpler sufficient condition for tractability. We give its formal definition:

Definition D.18. *Consider a* $\mathsf{PrXML}^{\mathsf{fie}}$ *document* D *with event set* X *and its LCRS representation* D'*. We say that an event* $e \in X$ occurs *in a node* n *of* D' *if* e *occurs in the annotation of the edge from the parent of* n *to* n*. For every* $e \in X$*, let* D'_e *be the smallest connected subtree of* D' *that covers all nodes where* e *occurs. The* event scope $S(n)$ *of a node* $n \in D'$ *is* $\{e \in X \mid n \in D'_e\}$*. The* event scope width *of* D *is* $w_{\mathrm{s}}(D) := \max_{n \in D} |S(n)|$*.*

We are now ready to prove the result on XML element scopes:

Proposition D.4. *For any* $\mathsf{PrXML}^{\mathsf{fie}}$ *document* D*, we have* $\mathrm{w}(J_D) \leqslant w_{\mathrm{s}}(D) + 1$*.*

Proof. We show how to build a tree decomposition of the relational encoding of J_D from the event scopes. Consider the tree decomposition T of I_D that is isomorphic to a LCRS encoding D' of D: the root node of D' is coded to an empty bag, and each node n of the LCRS encoding with parent n' is coded to $\{n', n\}$.

We now add to T, for each bag b corresponding to a node n, the events of $S(n)$. It is clear that T is of the prescribed width and that the occurrences of all nodes and events are connected subtrees.

We now argue that it is a tree decomposition of the relational encoding of J_D, but this is easily seen: it covers all *NS*- and *FC*- facts represented in J_D, and covers all occurrences and co-occurrences by construction of the scopes. □

This implies Proposition 6.1 because of Corollary D.2.

Tractability of $\mathsf{PrXML}^{\mathsf{mux,ind}}$. We now introduce the definitions and proofs for the local model, $\mathsf{PrXML}^{\mathsf{mux,ind}}$.

Definition D.19. *A* $\mathsf{PrXML}^{\mathsf{mux,ind}}$ *probabilistic document is an XML document* D *over* $\Lambda \sqcup \{\mathsf{ind}, \mathsf{mux}\}$*, where edges from* ind *and* mux *nodes to their children are labeled with a probability in* $[0, 1]$*, the annotations of outgoing edges of every* mux *node summing to* $\leqslant 1$*.*

The semantics $[\![D]\!]$ *of* D *is obtained as follows: for every* ind *node, decide to keep or discard each child according to the indicated probability, and replace the node by the (possibly empty) collection of its kept children; for every* mux *node, choose one child node to keep according to the indicated probabilities (possibly keep no node if they sum to* < 1*), and replace the* mux *node by the chosen child (or remove it if no child was chosen). All probabilistic choices are performed independently. When a node is not kept, its descendants are also discarded. We require the root to have label in* Λ*.*

Observe that in $\mathsf{PrXML}^{\mathsf{mux,ind}}$ all probabilistic choices are "local", in a similar fashion to the tuple-independent (TID) and BID probabilistic relational formalisms. As we show later, this helps ensure the tractability of query evaluation.

We use Corollary D.2 to show the tractability of query evaluation on the $\mathsf{PrXML}^{\mathsf{mux,ind}}$ local model, which was already proven in [CKS09]. We first rewrite input documents to a simpler form:

Definition D.20. *Two* $\mathsf{PrXML}^{\mathsf{mux,ind}}$ *documents* D_1 *and* D_2 *are equivalent if for every XML document* D*,* $\Pr_{D_1}(D) = \Pr_{D_2}(D)$*.*

Definition D.21. *We say that a* $\mathsf{PrXML}^{\mathsf{mux,ind}}$ *is in* binary form *if it is a full binary tree, and the sum of the outgoing probabilities of every* mux *node is equal to* 1*.*

The following definition is needed to ensure linear time execution for technical reasons:

Definition D.22. *A* $\mathsf{PrXML}^{\mathsf{mux,ind}}$ *document is* normalized *if for every* mux *nodes, the rational probabilities that annotate its child nodes all share the same denominator.*

Lemma D.7. *From any normalized* $\mathsf{PrXML}^{\mathsf{mux,ind}}$ *document D, we can compute in linear time in D an equivalent* $\mathsf{PrXML}^{\mathsf{mux,ind}}$ *document D' which is in binary form.*

Proof. In this proof, for brevity, we use det nodes to refer to ind nodes whose child edges are all annotated with probability 1.

First, rewrite mux nodes whose outgoing probabilities sum up to < 1 by adding a det child for them with the remaining probability. This operation is in linear time because the corresponding number has same denominator as other children of the mux node (as the document is normalized), and the numerator is smaller than the denominator.

Next, use det nodes to rewrite the children of regular and ind nodes to a chain so that all regular and ind nodes have at most 2 children. This only causes a constant-factor blowup of the document.

Next, rewrite mux nodes with more than two children to a hierarchy of mux nodes in the obvious way: considering a mux node n with k children $n_1, \ldots, n_k$ and probabilities $p_1, \ldots, p_k$ summing to 1, we replace n by a hierarchy $n'_1, \ldots, n'_{k-1}$ of mux nodes: the children of each n'_i is n_i with probability $\frac{p_i}{\sum_{j<i} p_j}$ and n'_{i+1} with probability $1 - \frac{p_i}{\sum_{j<i} p_j}$; except for n'_{k-1} whose children are n_{k-1} and n_k (with the same probabilities). This operation can be performed in linear time as the denominators of the fractions simplify (by the assumption that the document is normalized), and the sum operations work on operands and results which are smaller than the numerator.

Now, replace mux nodes with < 2 children by ind nodes (the probabilities are unchanged).

Last, add det children to nodes so that the degree of every node is either 2 or 0.

This process can be performed in linear time and that the resulting document is in binary form; equivalence has been maintained through all steps. □

Now, we can show:

Proposition D.5. *For any* $\mathsf{PrXML}^{\mathsf{mux,ind}}$ *document D in binary form, one can compute in linear time an equivalent* $\mathsf{PrXML}^{\mathsf{fie}}$ *document whose scopes have size $\leqslant 1$.*

Proof. For every ind node n with two children n_1 and n_2 with probabilities p_1 and p_2, introduce two fresh events $e_n^{\mathsf{ind},1}$ and $e_n^{\mathsf{ind},2}$ with probabilities p_1 and p_2, and replace n by a fie node so that its first and second outgoing edges are annotated with $e_n^{\mathsf{ind},1}$ and $e_n^{\mathsf{ind},2}$.

Likewise, for every mux node n with two children n_1 and n_2 with probabilities p and $1-p$, introduce a fresh event e_n^{mux} with probability p and replace n by a fie node so that its first and second outgoing edges are annotated with e_n^{mux} and $\neg e_n^{\mathsf{mux}}$.

It is immediate that the resulting document D' is equivalent to D. Now, consider the scope of any node of this document. Only one event occurs in this node, and the only events that occur more than one time in the document occur exactly twice, on the edges of two direct sibling nodes, so they never occur in the scope of any other node. Hence all scopes in D have size $\leqslant 1$. □

From this, given that $\mathsf{PrXML}^{\mathsf{mux,ind}}$ document can be normalized in ra-linear time, we deduce the tractability of MSO query evaluation on $\mathsf{PrXML}^{\mathsf{mux,ind}}$, as claimed in the main text:

Theorem 6.1 [CKS09]. *MSO query evaluation on* $\mathsf{PrXML}^{\mathsf{mux,ind}}$ *has ra-linear data complexity.*

D.3. BID instances

Following [BGMP92, RS07], we define:

Definition D.23. *A* BID instance *I is a relational instance with each relation partitioned into* key *and* value *positions. For each valuation of the key positions, all matching facts (that form a* block*) are mutually exclusive, each has a probability > 0 and the probabilities of the block sum to $\leqslant 1$. The semantics is to keep, independently between blocks, one fact at random in each block, according to the indicated probabilities (or possibly no fact if probabilities sum to < 1).*

To ensure ra-linear time complexity, we assume that BID instances are given with facts regrouped per blocks; otherwise our bounds are PTIME as we first need to sort the facts.

Definition D.24. *We define the* treewidth $\mathrm{w}(I)$ *of a BID instance I as that of the underlying relational instance, forgetting about the probabilities.*

We show the tractability of MSO query evaluation on BID through Theorem D.1 and Corollary D.1, using the following result:

Lemma D.8. *For any fixed $k \in \mathbb{N}^*$, given a BID instance J with $\mathrm{w}(J) \leqslant k$, we can compute in ra-linear time an equivalent pcc-instance J' where $\mathrm{w}(J')$ depends only on k.*

However, the proof of this result is non-trivial. By an encoding to pc-instances, it is straightforward to show the result if we assume that the size of each block is bounded by a constant. But otherwise, we need to build a decision circuit for which value to pick for each key; we do so in a tree-like fashion following a decomposition of the BID instance.

Proof. Fix k and J. First, compute in linear time a tree decomposition T of J of width $\mathrm{w}(J) \leqslant k$.

Without loss of generality, we can assume that probabilities within each block of J are rationals with the same denominator (if this is not the case, we normalize these probabilities in ra-linear time).

As in the proof of Lemma B.1, we can assume that every fact of J has been assigned to a bag of T where it is covered (i.e., $F = R(\mathbf{a})$ with $\mathbf{a} \subseteq \mathrm{dom}(b)$ for b the covering bag). Actually, still in the spirit of the proof of Lemma B.1, we can modify the decomposition T by copying nodes to create chains, so that we can assume that at most one fact is assigned to each bag. This preprocessing can be performed in linear time. For every fact F of J we let $\beta(F)$ be the bag of T to which fact F was assigned.

We compute the pcc-instance $J' = (J, C, \varphi)$ by building C and φ and a tree decomposition T' for J' with same skeleton as T, which is initialized as a copy of T. We add the gates of C to T' to turn it into a tree decomposition of J'.

Let $\mathcal{B}$ be the set of blocks: a key $\mathbf{a} \in \mathcal{B}$ is a pair of a relation symbol and a tuple that is a key in J for that relation. We write $J_\mathbf{a}$ to refer to J restricted to the facts of block $\mathbf{a}$; and $|I_\mathbf{a}|$ is the size (*not* the number of facts!) of this part of the instance (the size of both the facts and the associated probabilities). It is then clear that $\sum_{\mathbf{a}\in\mathcal{B}} |I_\mathbf{a}| = |J|$, the size of the original instance.

Now, for every $\mathbf{a} \in \mathcal{B}$, consider the subset of bags $T_\mathbf{a}$ of T that cover $\mathbf{a}$; it is a connected subtree, as it is the intersection for every element $a \in \mathbf{a}$ of the occurrence subtree T_k of this element, which are connected subtrees, and it is not empty because the elements of $\mathbf{a}$ must occur together in some fact of J so they also do in some bag of T. What is more, we can precompute in linear time the roots of all the $T_\mathbf{a}$ (by the same precomputation as in the proof

of Lemma B.1). It is also clear that $\sum_{\mathbf{a}\in\mathcal{B}} |T_{\mathbf{a}}|$ is of size linear in $|J|$, as, for fixed σ and k, each bag of T can only occur in a constant number of $T_{\mathbf{a}}$.

So we prove the result in the following way: for each $\mathbf{a} \in \mathcal{B}$, we compute in time $O(|I_{\mathbf{a}}| + |T_{\mathbf{a}}|)$ a circuit $C_{\mathbf{a}}$ to annotate the facts of $I_{\mathbf{a}}$ in J', and we add the gates of $C_{\mathbf{a}}$ to T' to obtain a tree decomposition of J' so far, making sure that we add only a constant number of gates to each bag, and only to bags that are in $T_{\mathbf{a}}$. If we can manage this for every $\mathbf{a} \in \mathcal{B}$, then the result follows, as we can process the blocks in J in order (as they are provided); our final pcc-instance has width that is still constant (for each bag of T can only occur in a constant number of $T_{\mathbf{a}}$); and by the arguments about the sizes of the sums, the overall running time of the algorithm is linear in J.

So in what follows we fix $\mathbf{a} \in \mathcal{B}$ and describe the construction of $C_{\mathbf{a}}$ and the associated decomposition.

Using our preprocessed table to find the root of $T_{\mathbf{a}}$, we can label its nodes by going over it top-down, in time linear in $T_{\mathbf{a}}$. We now notice that for every fact $F = R(\mathbf{a}, \mathbf{v})$ of $I_{\mathbf{a}}$, the bag $\beta(F)$ covers F so it must be in $T_{\mathbf{a}}$. We write $\beta_{\mathbf{a}}$ for the restriction of the function β to the facts of $I_{\mathbf{a}}$.

We now say that a bag $b \in T_{\mathbf{a}}$ is an *interesting bag* either if it is in the image of $f_{\mathbf{a}}$ or if it is a lowest common ancestor of some subset of bags that are in the image of $f_{\mathbf{a}}$. We now observe that the number of interesting bags of $T_{\mathbf{a}}$ is linear in the number of facts of $I_{\mathbf{a}}$; indeed, the interesting bags form the internal nodes and leaves of a binary tree whose leaves must all be in the image of $\beta_{\mathbf{a}}$, so the number of leaves is at most the number of facts of $I_{\mathbf{a}}$, so the total number of nodes in the tree is linear in the number of leaves.

We now define a weight function w on $T_{\mathbf{a}}$ by $\mathrm{w}(b) = \pi(F)$ (the probability of F) for $F \in I_{\mathbf{a}}$ and $\beta_{\mathbf{a}}(F) = b$, if any such F exists; $\mathrm{w}(b) = 0$ otherwise. We define bottom-up a cumulative weight function w' on $T_{\mathbf{a}}$ as $\mathrm{w}(b)$, plus $\mathrm{w}'(L(b))$ if $L(b) \in T_{\mathbf{a}}$, plus $\mathrm{w}'(R(b))$ if $R(b) \in T_{\mathbf{a}}$. For notational convenience we also extend w' to anything by saying that $\mathrm{w}'(b) = 0$ if $b \notin T_{\mathbf{a}}$ or b does not exist.

Observe now that for a non-interesting bag b, $\mathrm{w}(b)$ and $\mathrm{w}'(b)$ can be represented either as 0 or as a pointer to some $\mathrm{w}(b')$ or $\mathrm{w}'(b')$ for an interesting bag b'. Indeed, if b is non-interesting then we must have $\mathrm{w}(b) = 0$. Now we show that if b has a topmost interesting descendant b' then it is unique: indeed, the lowest common ancestor of two interesting descendants of b is a descendant of b and it is also interesting, so there is a unique topmost one. Now this means that either b' does not exist and $\mathrm{w}'(b) = 0$, or it does exist and all descendants of b that are in the image of $\beta_{\mathbf{a}}$ are descendants of b', so that $\mathrm{w}'(b) = \mathrm{w}'(b')$ and we can just make $\mathrm{w}'(b)$ a pointer to $\mathrm{w}'(b')$.

Now this justifies that we can compute w and w' bottom-up in linear time in $|T_{\mathbf{a}}| + |I_{\mathbf{a}}|$: observe that we are working on rationals with the same denominator, so the sums that we perform are sums of integers, whose size always remains less than the common denominator; as there is a number of interesting bags which is linear in the number of facts of $I_{\mathbf{a}}$, and those are the only nodes for which a value (whose size is that of the probabilities in $I_{\mathbf{a}}$) actually needs to be computed and written, the computation is performed in time $O(|T_{\mathbf{a}}| + |I_{\mathbf{a}}|)$ overall.

We now justify that we can encode $T_{\mathbf{a}}$ to a circuit with the correct probabilities. For each bag $b \in T_{\mathbf{a}}$, we create a gate g^{i}_b; for the root bag b it is an input gate with probability $\mathrm{w}'(b)$; for other bags it is a gate whose value is defined by the parent bag. Intuitively, g^{i}_b describes whether to choose a fact from $F_{\mathbf{a}}$ within the subtree rooted at b.

For every interesting bag b, writing $\mathrm{w}'(b) = k'/d$ and $\mathrm{w}(b) = k/d$ with d the common

denominator, create one input gate g_b^{h} with probability $\frac{1}{\mathrm{w}'(b)}\mathrm{w}(b) = k/k'$, and one gate $g_b^{\mathrm{h}\wedge}$ which is the AND of g_b^{h} and g_b^{i}. Intuitively, this gate describes whether to generate the fact assigned at this node, if any. If there is such a fact, set its image by φ to be $g_b^{\mathrm{h}\wedge}$. Now if $\mathrm{w}'(b) > \mathrm{w}(b)$ (intuitively: there is still the possibility to generate fact at child nodes), we create one input gate $g_b^{\leftrightarrow}$ which has probability $\frac{1}{\mathrm{w}'(b)-\mathrm{w}(b)}\mathrm{w}'(L(b))$. Once again, this probability simplifies to a rational whose numerator and denominator are $< d$. We create a gate g_b^{L} to be $g_b^{\mathrm{i}} \wedge \neg g_b^{\mathrm{h}} \wedge g_b^{\leftrightarrow}$ (creating a constant number of intermediate gates as necessary), and g_b^{R} to be $g_b^{\mathrm{i}} \wedge \neg g_b^{\mathrm{h}} \wedge \neg g_b^{\leftrightarrow}$, setting them to be $g_{L(b)}^{\mathrm{i}}$ and $g_{R(b)}^{\mathrm{i}}$ where applicable (i.e., if $L(b)$ and $R(b)$ exist and are in $T_{\mathbf{a}}$).

By contrast, non-interesting bags b just set $g_{L(b)}^{\mathrm{i}}$ and $g_{R(b)}^{\mathrm{i}}$ (where applicable) to be g_b^{i}, with no input gates.

We now observe that by construction the resulting circuit has a tree decomposition that is compatible with T, so that we can add its events to T' and only add constant width to the nodes of $T_{\mathbf{a}}$ as required. It is also easy to see that the circuit gives the correct distribution on the facts of $F_{\mathbf{a}}$, with the following invariant: for any bag $b \in T_{\mathbf{a}}$, the probability that g_b^{i} is 1 is $\mathrm{w}'(b)$, and $g_b^{\mathrm{h}\wedge}$, g_b^{L} and g_b^{R} are either all 0 if g_b^{i} is 0 or, if g_b^{i} is 1, exactly one is true and they respectively have marginal probabilities $\mathrm{w}(b)$, $\mathrm{w}'(L(b))$, and $\mathrm{w}'(R(b))$. Now the circuit construction is once again in time $O(|I_{\mathbf{a}}| + |T_{\mathbf{a}}|)$, noting that interesting nodes are the only nodes where numbers need to be computed and written; and we have performed the entire computation in time $O(|I_{\mathbf{a}}| + |T_{\mathbf{a}}|)$, so the overall result is proven. □

Combining Lemma D.8 and Theorem D.1, we can conclude:

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.*

D.4. Counting

Theorem 6.3 [ALS91]. *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.*

Proof. Let $k \in \mathbb{N}$. Let $q(\mathbf{x})$ be the MSO query. We rewrite it to the following query: $q' : \exists \mathbf{x} \bigwedge_{x \in \mathbf{x}} P_x(x) \wedge q(\mathbf{x})$, where the P_x are fresh unary predicates. Consider an input instance I of width $\leqslant k$, and expand it to a BID instance I' by setting existing relations to be trivial BID tables (i.e., all attributes of the relation are keys, and all facts have probability 1) and adding tables P_x for every $x \in \mathbf{x}$ with one attribute, with the empty set as key, and with facts $P_x(a)$ for all $a \in \mathrm{dom}(I)$, with probability $1/\left|\mathrm{dom}(I)\right|$. This rewriting can clearly be performed in ra-linear time, and if I has treewidth $\leqslant k$ then so does I'. Intuitively, the possible worlds of I' are all the possible ways of extending I with facts $P_x(a)$ for $x \in \mathbf{x}$ and $a \in \mathrm{dom}(I)$, with only one fact $P_x(a)$ for every $x \in \mathbf{x}$, and each possible world has probability $1/\left|\mathrm{dom}(I)\right|^{|\mathbf{x}|}$.

We now make the immediate observation that for every such possible world $I'_{\mathbf{a}}$ of I' indexed by the $a_x \in \mathrm{dom}(I)$ for $x \in \mathbf{x}$, where we add the facts $P_x(a_x)$ for $x \in \mathbf{x}$, we have $I'_{\mathbf{a}} \models q'$ iff $I \models q(\mathbf{a})$. Hence, the number of matches of q in I is the number of possible worlds of I' where q' holds, that is, the probability of q' on I' multiplied by $M := \left|\mathrm{dom}(I)\right|^{|\mathbf{x}|}$.

We conclude by Theorem 6.2 that we can compute this probability in ra-linear time in I', that is, in I, and we compute the count from the probability by multiplying by M in ra-linear time, proving the result. □

REFERENCES FOR THE APPENDIX

[ALS91] Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2), 1991.

[BGMP92] Daniel Barbará, Hector Garcia-Molina, and Daryl Porter. The management of probabilistic data. *TKDE*, 4(5), 1992.

[Bod96] Hans L Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6), 1996.

[CDG+07] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata: Techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2007.

[CKS09] Sara Cohen, Benny Kimelfeld, and Yehoshua Sagiv. Running tree automata on probabilistic XML. In *PODS*, 2009.

[Cou90] Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inf. Comput.*, 85(1), 1990.

[CV92] Surajit Chaudhuri and Moshe Y Vardi. On the equivalence of recursive and nonrecursive Datalog programs. In *PODS*, 1992.

[DMRT14] Daniel Deutch, Tova Milo, Sudeepa Roy, and Val Tannen. Circuits for Datalog provenance. In *ICDT*, 2014.

[FFG02] Jörg Flum, Markus Frick, and Martin Grohe. Query evaluation via tree-decompositions. *J. ACM*, 49(6), 2002.

[Gav74] Fănică Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *J. Combinatorial Theory*, 16(1), 1974.

[GHO02] Erich Grädel, Colin Hirsch, and Martin Otto. Back and forth between guarded and modal logics. *TOCL*, 3(3), 2002.

[GKT07] Todd J Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, 2007.

[GT06] Todd J. Green and Val Tannen. Models for incomplete and probabilistic information. In *IIDB*, 2006.

[HAKO09] Jiewen Huang, Lyublena Antova, Christoph Koch, and Dan Olteanu. MayBMS: a probabilistic database management system. In *SIGMOD*, 2009.

[HD96] Cecil Huang and Adnan Darwiche. Inference in belief networks: A procedural guide. *Int. J. Approximate Reasoning*, 1996.

[KKS08] Benny Kimelfeld, Yuri Kosharovsky, and Yehoshua Sagiv. Query efficiency in probabilistic XML models. In *SIGMOD*, 2008.

[LS88] Steffen L Lauritzen and David J Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *J. Royal Statistical Society. Series B*, 1988.

[NS02] Frank Neven and Thomas Schwentick. Query automata over finite trees. *TCS*, 275(1), 2002.

[RS07] Christopher Ré and Dan Suciu. Materialized views in probabilistic databases: for information exchange and query optimization. In *VLDB*, 2007.

[SORK11] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Morgan & Claypool, 2011.

[TW68] James W. Thatcher and Jesse B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Math. systems theory*, 2(1), 1968.