Probabilities and Provenance via Tree Decompositions

Antoine Amarilli
Institut Mines–Télécom
Télécom ParisTech
CNRS LTCI
Paris, France

Pierre Bourhis
CNRS LIFL
Université Lille 1
INRIA Lille
Lille, France

Pierre Senellart
Institut Mines–Télécom
Télécom ParisTech; CNRS LTCI
& NUS; CNRS IPAL
Paris, France & Singapore

ABSTRACT

Query evaluation is hard on probabilistic databases, even on very simple probabilistic data frameworks and fairly simple queries, except for limited classes of safe queries. We study the problem from a different angle: rather than restricting the queries, at which conditions on the *data* can we tractably evaluate expressive queries on probabilistic instances? More specifically, we restrict the data *treewidth*, which we define on a circuit-based generalization of c-tables, in a natural way that restricts both the underlying instance and the annotations. We then leverage known tree-automata constructions to evaluate queries on bounded-treewidth instances, for such logical fragments as monadic second-order logic or frontier-guarded Datalog. We prove that we can compute in linear time a *bounded-treewidth* lineage circuit for automaton runs on tree decompositions of bounded-treewidth instances, so that the probability of the query can then be evaluated in linear-time data complexity (assuming unit-cost arithmetic). We also show that a similar construction can yield a circuit representation of the semiring provenance for absorptive semirings in the case of monotone queries. For known probabilistic data frameworks, this results implies bounded-treewidth tractability of query evaluation on BID relational models, and sufficient tractability conditions for probabilistic XML models.

1. INTRODUCTION

While relational database management systems are useful to query exact data, real-world data is often approximate, outdated, or incomplete: think of data extracted from unreliable sources, or using noisy techniques. These problems are the focus of uncertain and probabilistic data management, a topic investigated both from a theoretical angle [41, 50] and through practical implementations [34].

However, a limit of these works is that query evaluation on such instances is often hard. As an example, consider the simple *tuple-independent* (TID) model, with uncertainty at the tuple level: each tuple is present or absent with some probability, and we assume independence between all tuples. It is already #P-hard [18] to compute the probability that possible worlds of an input TID instance satisfies the simple, fixed CQ (conjunctive query) $\exists x \exists y\, R(x) \wedge S(x,y) \wedge T(y)$; by contrast, this query has PTIME (even AC^0) data complexity on usual relational instances.

Existing works have studied two main ways to mitigate this. The first one is to compute approximate probabilities, e.g., through sampling [36]. The second is to restrict the fixed query to *safe* query classes [39, 50] that guarantee tractable data complexity on all instances.

This works investigates a third approach, for which much less is understood: to restrict the *input instances* to ensure tractability no matter the query. We believe this approach has practical relevance (real-world data is not arbitrary) and has proven fruitful in other theoretical contexts. Consider the example of monadic second-order (MSO) queries, which are NP-hard to evaluate on (non-probabilistic) instances [6], but have linear-time complexity on instances whose *treewidth* is bounded [17], intuitively restricting them to be close to trees. Such results also apply, e.g., to counting and reliability calculations [6], which suggests a natural question: can we adapt them to query evaluation on probabilistic instances and show tractability assuming bounded treewidth?

Two obstacles make this question harder to answer. First, there are many probabilistic frameworks (TID, BID, probabilistic c-tables, probabilistic XML...), so it is difficult to define a general notion of treewidth for all of them. Second, probabilistic models such as pc-tables have probabilistic *correlations* which can also cause hardness even for a trivial underlying instance: it is not clear how to bound simultaneously the instances and the correlations.

This work presents a solution to both of these problems. We introduce the probabilistic framework of *pcc-instances*, a straightforward extension of pc-tables with tuple annotations given by a circuit rather than by formulae. We then show how to define a very natural notion of tree decompositions for pcc-tables, intuitively encoding the annotating circuit in the instance. This allows us to prove our desired result: query answering has tractable data complexity when the instance treewidth is bounded; in fact it is even linear assuming constant-time arithmetic operations.

While our result applies to pcc-tables, which may be of independent interest as a concise representation for pc-tables, we show that pcc-tables capture other formalisms, and the bounded-treewidth condition neatly translates to them. So a first strength of our result is that it implies tractability corollaries for all the probabilistic frameworks mentioned so far, assuming a constant bound on a natural notion of width for each model. For instance, fixed CQs are tractable to evaluate on input TID instances if they have bounded treewidth in the usual sense, ignoring probabilities.

A second strength of our results is that they cover very expressive queries languages. In fact, they generalize from CQs to all queries that can be rewritten to tree automata on relational instances of bounded treewidth. This query class, as we show, includes MSO, and in fact covers many expressive query languages, such as frontier-guarded Datalog, which are fragments of guarded second-order logic; we study the tradeoff between expressiveness and query complexity. While the tractability of such expressive languages is not so surprising as it matches known results in the non-probabilistic setting, it is in stark contrast with the narrow classes of queries which are tractable on arbitrary probabilistic instances.

A third strength is that our approach does not depend either on the specifics of probability computation. In fact, as we show, determining whether the query holds on a possible world of the input instance reduces to determining whether a certain circuit representation of the problem evaluates to true, and the circuit can be computed in linear time and has bounded treewidth. This allows us to apply probabilistic inference algorithms [9, 11, 43] as a black box on the circuit representation, which isolates neatly the “symbolical” aspects of the reduction, and the “numerical” aspects of probability computation, yielding a very modular proof.

What is more, we show that these circuits are not a mere technical tool, but have independent interest: they represent concisely which valuations of the probabilistic events satisfy the query. In this sense, we study how they relate to semiring provenance [30], a very general framework to understand the link between data instance and query result. More specifically, we show that a variant of our circuit construction can be used as a provenance circuit [20], if we assume that the query is monotone, and that the semiring is absorptive. This study of provenance is novel in at least two respects. First, following our focus on logical queries (rather than relational algebra or Datalog), we give an intrinsic (rather than operational) definition of provenance for queries on both sets and multisets of tuples, that we connect to standard definitions [30]. Second, our work provides (to our knowledge) the first general study of provenance for bounded-treewidth data, with interesting technical connections between tree automata and the resulting provenance circuit.

Paper structure. We state our problem in Section 2 with the minimal definitions, introduce some required technical tools in Section 3, and present our main definitions and results in Section 4, and their proof in Section 5. Sections 6 and 7 investigate respectively the ramifications of this result in terms of query languages, and in terms of probabilistic evaluation. Section 8 spells out the consequences for various existing probabilistic formalisms, and Section 9 investigates the connections to semiring provenance. We finally discuss related work in Section 10. Our results are provided with complete proofs which are deferred to the appendix because of space constraints.

2. PROBLEM STATEMENT

In this section, we define instances and queries [1] and the problems we study on uncertain and probabilistic data [50].

Instances. We fix a countable domain $\mathcal{D} = \{a_k \mid k \geqslant 0\}$. A *signature* σ is a finite set of *relation names* (e.g., R) with associated *arity* $\operatorname{arity}(R) \geqslant 1$. A *relational instance* (or *structure*) I over σ 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 $\operatorname{arity}(R)$ elements of $\mathcal{D}$. The *domain* $\operatorname{dom}(I) \subseteq \mathcal{D}$ of I is the finite set of domain elements used in I. Two instances I and I' are *isomorphic* if there is a bijection φ from $\operatorname{dom}(I)$ to $\operatorname{dom}(I')$ such that $\varphi(I) = I'$. We say that an instance J is a *subinstance*[1] of I, written $J \subseteq I$, if it contains some subset of the facts of I, which implies $\operatorname{dom}(J) \subseteq \operatorname{dom}(I)$.

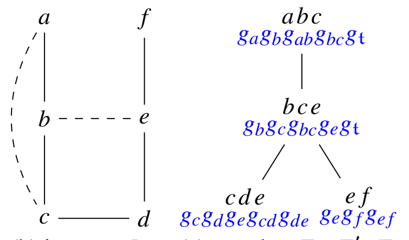
Queries. We focus on queries expressed by (function-free) *first-order* (FO) and *second-order* (SO) formulae. We omit definitions on FO [1], noting that we allow the equality predicate; SO is the extension of FO that allows second-order quantification over predicates (of arbitrary arity) which can then be used as part of the signature. *Monadic second-order* (MSO) is the SO fragment where all second-order variables have arity 1 (i.e., sets).

We mainly focus on *Boolean queries*, i.e., formulae with no free variables. For other queries, we assume that all free-variables are first-order and that query results are always from the domain of the instance (e.g., the query is *safe* [1]).

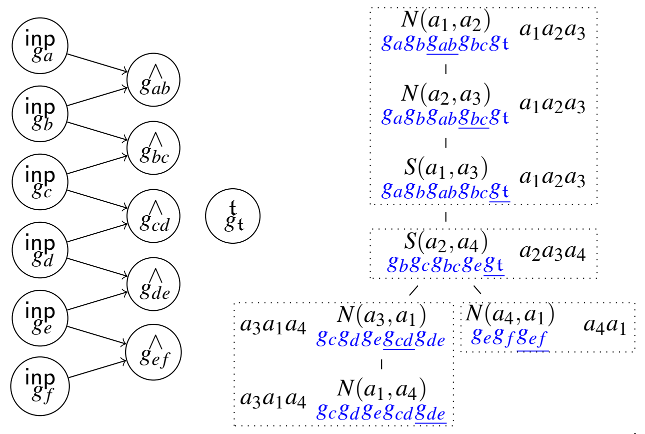
Uncertain and probabilistic instances. A *probability distribution* is a pair $(\mathcal{U}, \Pr)$ of a finite *universe* $\mathcal{U}$ (whose elements are called *possible worlds*) and a *probability measure* $\Pr : \mathcal{U} \to [0,1]$ such that $\sum_{I \in \mathcal{U}} \Pr(I) = 1$.

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

$N(a,b)$	$x_a \wedge x_b$
$N(b,c)$	$x_b \wedge x_c$
$N(c,d)$	$x_c \wedge x_d$
$N(d,e)$	$x_d \wedge x_e$
$N(e,f)$	$x_e \wedge x_f$
$S(a,c)$	$\mathfrak{t}$
$S(b,e)$	$\mathfrak{t}$


a
f
b
e
c
d
abc
$g_a g_b g_{ab} g_{bc} g_\mathfrak{t}$
bce
$g_b g_c g_{bc} g_e g_\mathfrak{t}$
cde
$g_c g_d g_e g_{cd} g_{de}$
ef
$g_e g_f g_{ef}$

(a) c-instance J_U (b) instance I_U (c) tree dec. T_U, T'_U, T_C


inp g_a
inp g_b
inp g_c
inp g_d
inp g_e
inp g_f
$\wedge$ g_{ab}
$\wedge$ g_{bc}
$\wedge$ g_{cd}
$\wedge$ g_{de}
$\wedge$ g_{ef}
$\mathfrak{t}$ $g_\mathfrak{t}$
$N(a_1,a_2)$ $g_a g_b \underline{g_{ab}} g_{bc} g_\mathfrak{t}$ $a_1a_2a_3$
$N(a_2,a_3)$ $g_a g_b g_{ab} \underline{g_{bc}} g_\mathfrak{t}$ $a_1a_2a_3$
$S(a_1,a_3)$ $g_a g_b g_{ab} g_{bc} \underline{g_\mathfrak{t}}$ $a_1a_2a_3$
$S(a_2,a_4)$ $g_b g_c g_{bc} g_e \underline{g_\mathfrak{t}}$ $a_2a_3a_4$
$a_3a_1a_4$ $N(a_3,a_1)$ $g_c g_d g_e \underline{g_{cd}} g_{de}$
$N(a_4,a_1)$ $g_e g_f \underline{g_{ef}}$ a_4a_1
$a_3a_1a_4$ $N(a_1,a_4)$ $g_c g_d g_e g_{cd} \underline{g_{de}}$

(d) circuit C_U (e) tree encoding E_U, cc-tree-encoding E'_U

An *uncertainty framework* gives to objects O in some language a semantics $[\![O]\!]$ which is a universe of instances (which may be much larger than O). Likewise, a *probabilistic framework* gives to O a semantics $[\![O]\!]$ which is a probability distribution over instances.

A fundamental example of an uncertainty and probabilistic framework are *c-instances* and *pc-instances*:

DEFINITION 2.1 [31, 34, 35]. *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 $\{\mathfrak{t}, \mathfrak{f}\}$, the possible world $\nu(J)$ is obtained by retaining exactly the tuples whose annotation evaluates to $\mathfrak{t}$ under ν; $[\![J]\!]$ is the set*[2] *of all these possible worlds.*

A pc-instance *$J = (J', \pi)$ is defined as a c-instance J' and a* probabilistic valuation[3] *$\pi : X \to [0,1]$ for the variables used in J'. The probability distribution $[\![J]\!]$ defined by J has universe $[\![J']\!]$ and probability measure $\Pr_J(I) := \sum_{\nu \mid \nu(J')=I} \Pr_J(\nu)$ with the product distribution on valuations:*

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

EXAMPLE 2.2. *The instance I_U of Figure (b), or Figure (a) ignoring annotations, represents people seated around a U-shaped dinner table: N indicates neighbors which can whisper to one another, and S (dashed in Figure (b)) indicates married couples. We will use it as our running example.*

We add uncertainty with one Boolean event x_i per person i indicating whether they are available to talk (rather than busy, e.g., eating). Hence, a neighbor pair can communicate only when both people are available. This yields the c-instance J_U of Figure (a).

Problem statement. In this paper, we study generalizations of the standard problem of evaluating a query on an instance, for the setting of uncertain and probabilistic instances:

DEFINITION 2.3. *Given a fixed query q and an uncertainty*

[2]Different valuations may yield the same possible world.

[3]Like all probabilities in this paper, the values of π are rationals.

framework, the probabilistic query evaluation problem *is to determine, for an input probabilistic instance J with $(\mathcal{U}, \Pr) = [\![J]\!]$, the probability that the query holds in a possible world of J, namely:* $\sum_{\substack{I\in\mathcal{U}\\ I\models q}} \Pr(I)$

We are considering the *data complexity* of these problems: q is fixed and we measure complexity as a function of $|J|$. In the whole paper, we use $|A|$ to denote an estimation of the size needed to represent an object A: for example, for a set of abstract objects, this is its cardinality in the usual sense, for an integer the length of its binary representation, and for a function $f : S \mapsto T$, $|f|$ is $|S|\log|T|$.

EXAMPLE 2.4. *We consider the MSO query $q_U : \exists xy\, S(x,y) \wedge N^*(x,y)$, where $N^*(x,y)$ is the (MSO-expressible) transitive closure of $N(x,y) \vee N(y,x)$; it asks whether there are two married people who can communicate by passing whispered messages along table neighbors. So our problem is, given the input pc-instance $J = (J_U, \pi)$ with some π indicating the probability that every person is busy (assuming independence), to compute the probability that q_U holds in possible worlds of J.*

3. TECHNICAL PRELIMINARIES

We now introduce technical tools: tree decompositions [22], tree automata [52], and circuits [53].

Trees. Given a fixed *alphabet* Γ, we will define a Γ*-tree* $(V, Ch_\swarrow, Ch_\searrow, \lambda)$ as a set of *nodes* V, two partial mappings $Ch_\swarrow, Ch_\searrow : V \to V$ that associate an internal node with its left and right child, and a *labeling function* $\lambda : V \to \Gamma$. Unless stated otherwise, the trees are rooted, directed, ordered, binary, and full (each node has either zero or two children). We write $n \in T$ to mean $n \in V$. When Γ is finite, the *size* $|T|$ of T is $O(|V| \cdot (\log|\Gamma| + \log|V|))$. We say that two trees T_1 and T_2 have *same skeleton* if they are the same tree up to labels.

Tree decompositions. A *tree decomposition* T of a relational instance I is a $\mathcal{T}$-tree $T = (B, Ch_\swarrow, Ch_\searrow, \mathrm{dom})$ 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 $w(T) := k - 1$ where $k := \max_{b\in T} |\mathrm{dom}(b)|$. The *treewidth* (or *width*) of an instance I, written $w(I)$, is the minimal width $w(T)$ of a tree decomposition T of I. It is NP-hard, given an instance I, to determine $w(I)$. 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 [10].

EXAMPLE 3.1. *Figure (c), ignoring for now the g_i elements, is a tree decomposition T_U of width $k = 2$ of I_U. (For brevity, when drawing trees we omit any additional nodes that would be needed to make them full.)*

Tree encodings. Structures of bounded treewidth can be represented by a variant of their tree decomposition which is a tree labeled on a finite alphabet. The representation is up to isomorphism, namely, the identities of constants are lost.

Our finite alphabet is the set of possible facts on an instance of fixed size; we will use element co-occurrences between one node and its parent in the tree as a way to encode element reuse[4]:

DEFINITION 3.2. *The set of k*-facts *of σ, written* $\mathrm{fct}_k(\sigma)$*, 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 $\mathcal{D}$, $a_1, \ldots, a_{2k+2}$;*
- *the* structure *s is a zero- or single-fact structure over σ such that* $\mathrm{dom}(s) \subseteq d$.

A tree encoding *E of width k is a* $\mathrm{fct}_k(\sigma)$*-tree. We say that $n \in E$ is* empty *if its label is of the form $(d, \emptyset)$.*

One can verify that, for n_σ the number of relations of σ and a_σ the maximal arity of σ, we have:
$|\mathrm{fct}_k(\sigma)| = \sum_{i=0}^{k+1} \binom{2k+2}{i} \left(1 + \sum_{R\in\sigma} i^{\mathrm{arity}(R)}\right) = O\left(2^{2k} n_\sigma k^{a_\sigma}\right)$.

We first explain how 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 3.3. *We call a* $\mathrm{fct}_k(\sigma)$*-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 $w(I) \leqslant k$. 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 [15]).

LEMMA 3.4. *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 sketch. 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 [22]. We then encode each node of T as a chain of nodes in E, one for each fact assigned to T. ☐

EXAMPLE 3.5. *A tree encoding E_U of I_U following T_U is given as Figure (e). (Ignore for now the g_i elements.) Each node is labeled by a domain and a fact using elements of the domain. The node clusters represent chains of facts created for each bag of T_U. Note that the domains of E_U correspond to those of T_U; yet, e.g., the two leaves use a_1 in the encoding, to refer to both d and f in T_U.*

Tree automata. A *bottom-up deterministic tree automaton* on (binary full) Γ-trees, or Γ*-bDTA*, is a tuple $A = (Q, F, \iota, \delta)$ of a set Q of *states*, a subset $F \subseteq Q$ of *accepting states*, an *initial function* $\iota : \Gamma \to Q$ determining the state of leaves from their label, and a *transition function* $\delta : Q^2 \times \Gamma \to Q$ determining the state of internal nodes from their label and the states of their children. $|A|$ is $|Q|^2\,|\mathrm{fct}_k(\sigma)|$ up to polylogarithmic factors (intuitively the size of a table for δ).

The *run* of A on a Γ-tree E is a function $\rho : E \to Q$ such that for each leaf node n, $\rho(n) = \iota(\lambda(n))$, and for every internal node n, $\rho(n) = \delta(\rho(Ch_\swarrow(n)), \rho(Ch_\searrow(n)), \lambda(n))$. The run is *accepting* if, for the root n_{r} of E, $\rho(n_{\mathrm{r}}) \in F$; and *$A$ accepts E* (written $E \models A$) if its run on E is accepting.

Circuits. Let $\mathcal{V} = \{V_1, \ldots, V_m\}$ be a finite set of *value sets* where each V_i is countable, and $\mathcal{F} = \{f_1, \ldots, f_k\}$ a finite set of *gate functions* where each f_i is a function from a product of value sets $V_n^{\mathrm{i}}(f)$ $(0 \leqslant n < \mathrm{arity}(f))$ to a value set V^{o}, where $0 \leqslant \mathrm{arity}(f) \leqslant 3$. A $(\mathcal{V}, \mathcal{F})$-circuit[5] is a directed acyclic graph

[4]This idea is from [15], where tree encodings are called *proof trees*.

[5]The dependence on $\mathcal{V}$ and $\mathcal{F}$ is omitted from notation when clear from context; we also simply write K-circuit when $\mathcal{V} = \{K\}$.

$C = (G, W)$, where each *gate* $g \in G$ is associated with a *value set* $V(g) \in \mathcal{V}$ and a *type* which is either $f \in \mathcal{F}$ (we then call g a *function gate* and require $V^{\mathrm{o}}(f) = V(g)$) or inp (we call g an *input gate*). Edges, or *wires*, of W are triples (g', g, i) where (g', g) is an edge in the usual sense, g is a function gate with function f, and $0 \leqslant i < \mathrm{arity}(f)$ is the *position* of the wire with respect to g. We require that a gate g with function f have exactly $\mathrm{arity}(f)$ incoming wires with distinct positions (and input gates have no incoming wires). We may write $g \in C$ for $g \in G$.

A *valuation* of the inputs C_{inp} of a circuit C is a function from its input gates to a value of their domain. Each such valuation ν defines a unique *evaluation* $\nu(C)$ of C by ν, namely, a function mapping each $g \in C$ to $\nu(C)(g) \in V(g)$, which is defined inductively: $\nu(C)(g)$ is $\nu(g)$ for $g \in C_{\mathsf{inp}}$, and $\nu(C)(g)$ for $g \notin C_{\mathsf{inp}}$ is the value of the function f of gate g on the sequence $(\nu(C)(g'))$ for $(g', g, i) \in W$, $0 \leqslant i < \mathrm{arity}(f)$.

A common example of circuits are *Boolean circuits*, where we have $\mathcal{V} = \{\{\mathfrak{t}, \mathfrak{f}\}\}$, $\mathcal{F} = \{\wedge, \vee, \neg, \mathfrak{t}, \mathfrak{f}\}$ with $\mathrm{arity}(\wedge) = \mathrm{arity}(\vee) = 2$, $\mathrm{arity}(\neg) = 1$, $\mathrm{arity}(\mathfrak{t}) = \mathrm{arity}(\mathfrak{f}) = 0$, and the expected truth tables. Any propositional formula can be translated in linear time to a Boolean circuit by creating one input gate per variable and one function gate per constant and per subexpression (using associativity to ensure that all $\wedge$ and $\vee$ operators are binary).

Tree decompositions of circuits. To a set $\mathcal{F}$ of gate functions, we can associate a relational signature $\sigma_{\mathcal{F}}$ with one unary relation Inp and one relation R^f of arity $\mathrm{arity}(f) + 1$ for every $f \in \mathcal{F}$. We can thus encode in linear time a $(\mathcal{V}, \mathcal{F})$-circuit $C = (G, W)$ to a $\sigma_{\mathcal{F}}$ instance I_C with $\mathrm{dom}(I_C) = G$ that contains one fact $\mathrm{Inp}(g)$ for each $g \in C_{\mathsf{inp}}$, and one fact $R^f(g, g_1, \ldots, g_{\mathrm{arity}(f)})$ for each $g \notin C_{\mathsf{inp}}$ with function $f \in \mathcal{F}$ and inputs $g_1, \ldots, g_{\mathrm{arity}(f)}$. The *treewidth* $w(C)$ of C is $w(I_C)$ (but we talk of tree decompositions of C as shorthand).

EXAMPLE 3.6. *Looking only at the g_i nodes, Figure (c) represents a tree decomposition T_C (of non-minimal width) of the Boolean circuit C_U of Figure (d).*

4. GENERAL MODEL AND MAIN RESULT

Our goal is to investigate the tractability of query evaluation on probabilistic instances under a suitable notion of bounded treewidth. Rather than investigating separately all existing probabilistic frameworks, we introduce new frameworks for uncertain and probabilistic instances, namely *cc-instances* and *pcc-instances*. These models are a straightforward generalization of (p)c-instances annotated by circuits rather than formulae. We next define a notion of tree decomposition and treewidth for them.

This being done, we focus on cc-instances only, and state our main result: the evaluation of some query classes on bounded-treewidth cc-instances can be represented in linear time by a bounded-treewidth circuit. We first state a simple version of this result (Theorem 4.8), and then a more elaborate version which accounts for combined complexity (Theorem 4.13).

Those results are proven in the next section. Section 6 then studies to which query classes they apply, and Section 7 points out their consequences for our original problem of probabilistic query evaluation.

4.1 CC-Instances

We define the *circuit c-instance* (cc-instance) uncertainty framework, and the corresponding pcc-instance probabilistic framework:

DEFINITION 4.1. *A* cc-instance *is a triple $J = (I, C, \varphi)$ of a relational σ-instance I, a 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 $\nu(C)(\varphi(F)) = \mathfrak{t}$, 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: $\Pr_J(\nu) = \prod_{\substack{g \in J_{\mathsf{inp}} \\ \nu(g) = \mathfrak{t}}} \pi(g) \prod_{\substack{g \in J_{\mathsf{inp}} \\ \nu(g) = \mathfrak{f}}} (1 - \pi(g))$.

Naturally, (p)cc-instances are very related to (p)c-instances, except that their annotations are represented as circuits. It is straightforward that (p)cc-instances capture (p)c-instances:

PROPOSITION 4.2. *For any (p)c-instance J, one can compute in linear time a (p)cc-instance J' whose inputs are the variables X of J, such that for any valuation ν of X, $\nu(J) = \nu(J')$ (and, for the probabilistic version, $\Pr_J(\nu) = \Pr_{J'}(\nu)$).*

EXAMPLE 4.3. *The cc-instance J'_U corresponding to J_U maps each fact $N(x, y)$ by φ to the gate g_{xy} of circuit C_U, and maps the S facts to $g_{\mathfrak{t}}$.*

Conversely, while (p)c-instances do capture (p)cc-instances (they can clearly represent any universe or probability distribution), the naïve translation from cc-instances to c-instances leads to a blowup, intuitively because c-instances cannot share common subexpressions between tuple annotations.

EXAMPLE 4.4. *We temporarily leave our running example aside. Fix $n \in \mathbb{N}$ and a universe $\mathcal{U}_n$ whose possible worlds are $\emptyset$, $\{R(a_1, a_2)\}$, $\{R(a_1, a_2), R(a_2, a_3)\}$, ..., $\{R(a_1, a_2), \ldots, R(a_{n-1}, a_n)\}$. For instance, this could be a representation of the possible worlds of a chain in an uncertain XML document, where the removal of an edge implies that all descendant edges are also removed.*

It is easy to represent $\mathcal{U}_n$ as a cc-instance J with underlying instance $I := \{R(a_1, a_2), \ldots, R(a_{n-1}, a_n)\}$, C defined by $g_1 = g_1^{\mathrm{i}}$, and $g_j := g_{j-1} \wedge g_j^{\mathrm{i}}$ for $1 < j \leqslant n - 1$, with the g_j^{i} being inputs, and $\varphi(R(a_i, a_{i+1})) = g_i$. This J has size linear in n.

By contrast, it is harder to represent $\mathcal{U}_n$ as a c-instance, as the annotation of every fact "depends" on that of the previous fact in the chain. If we consider the natural encoding J' where each fact $R(a_j, a_{j+1})$ is annotated by the conjunction $e_1 \wedge \cdots \wedge e_j$, we see that J' has size quadratic in n.

We leave open the question of whether (p)cc-instances are strictly more compact than (p)c-instances no matter the translation, noting that it is related to the problem of comparing the compactness of Boolean formulae and circuits [53]. In that context, some linear-size circuits are known [32] to have no equivalent formulae shorter than $O(n^{3-\varepsilon})$; but this does not directly translate to a bound for our setting.

4.2 Tree Decompositions

We now introduce our notion of *tree decomposition* of a cc-instance. Intuitively, it is a *simultaneous* tree decomposition of its underlying relational instance and of the relational encoding of its circuit, which *respects* the mapping φ between the two. However, we can just formalize it as a tree decomposition in the usual sense for a natural notion of *relational encoding* of the cc-instance.

DEFINITION 4.5. *Let σ_{b} be the signature of the relational encoding of Boolean circuits. 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 $(\sigma_{\mathrm{b}} \sqcup \sigma^+)$-*

instance containing both the σ_{b}-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).*

As the relational encoding of a (p)cc-instance can clearly be computed in linear time, all existing algorithms and complexity bounds [10] for the computation of tree decompositions of relational instances also apply to (p)cc-instances. In particular, for a fixed k, we can compute in linear time in the input cc-instance J a tree decomposition of J of width $\leqslant k$ if one exists.

Note that for a cc-instance $J = (I, C, \varphi)$, we have $w(J) \geqslant w(I)$, and $w(J) \geqslant w(C)$, as a tree decomposition of width k of J doubles as a tree decomposition of width $\leqslant k$ of both I and C; but conversely $w(C) \leqslant k$ and $w(I) \leqslant k$ does not imply that $w(J)$ is bounded. Note also that any possible world I' of J has treewidth at most k (as $I' \subseteq I$, $w(I') \leqslant w(I)$).

EXAMPLE 4.6. *Returning to our running example, Figure (c) is a tree decomposition T'_U of the cc-instance J'_U; it combines both the tree decomposition T_U of I_U and T_C of C_U, in a* compatible *way (the facts of I_U are covered, including their image by φ).*

4.3 Main Result

We now state our main result, but we must first define the query class to which it applies.

DEFINITION 4.7. *A $\mathrm{fct}_k(\sigma)$-bDTA A* tests *a Boolean query q for treewidth k if for any $\mathrm{fct}_k(\sigma)$-tree E, $E \models A$ iff $\langle E \rangle \models q$. (In particular, if $\langle E \rangle = \perp$ then A rejects E.)*

We say that a Boolean query q is fixed-treewidth automata-rewritable *(FTAR) if, for every k, one can compute a bDTA A_k on $\mathrm{fct}_k(\sigma)$ that tests q for treewidth k.*

Many important query classes, such as UCQs, MSO, or guarded Datalog, are FTAR: we review them in Section 6.

The key point is that a FTAR query can be tested in linear-time data complexity on a bounded-treewidth relational instance I by computing an instance-independent automaton A, computing a tree encoding E of I, and running A on E. Our main result shows that something similar can be done when the instance and tree encodings are uncertain:

THEOREM 4.8. *For any fixed integer k and FTAR query q, one can compute in linear time, from a cc-instance J with $w(J) \leqslant k$, a Boolean circuit C on J_{inp} with a distinguished gate g such that for every valuation ν of J_{inp}, $\nu(C)(g) = \mathfrak{t}$ iff $\nu(J) \models q$, with $w(C)$ depending only on k and q.*

In other words, for a bounded-treewidth cc-instance J, a FTAR query q can be "instrumented" in linear time data complexity, in the sense of producing a circuit C describing which valuations of J satisfy q; furthermore, the treewidth of C is bounded. We show in Section 7 the consequences of this fact in terms of *probability*, by an application of belief propagation to justify that the probability that q holds on J is tractable to evaluate. In terms of *provenance*, we study in Section 9 how the resulting circuit relates to semiring provenance.

EXAMPLE 4.9. *The Boolean circuit C' obtained for q_U on J_U would represent the Boolean function describing when q_U holds on a possible world of J_U; in this case, it would be equivalent to $x_b \wedge x_c \wedge (x_a \vee (x_d \wedge x_e))$.*

A limitation of the result is that FTAR queries are by definition Boolean; and they must be invariant under isomorphism, which forbids constants. Can our theorem apply to such queries? We see this by introducing the required notions:

DEFINITION 4.10. *We say that a Boolean query q over σ is* quasi-FTAR *if there exists a linear-time computable FTAR query q' over some σ' and set of facts I_q such that for any instance I over σ, $I \models q$ iff $I \cup I_q \models q'$.*

We say that a non-Boolean query q is non-Boolean quasi-FTAR *if for every tuple $\mathbf{a}$ with the output arity of q, there exists a linear-time computable Boolean quasi-FTAR query $q_{\mathbf{a}}$ such that for any instance I, $I \models q(\mathbf{a})$ iff $I \models q_{\mathbf{a}}$.*

Quasi-FTAR queries encompass in particular queries that contain constants when the underlying constant-free query language is FTAR: by replacing constants in the query as unary predicates that contain a single fact, one can obtain a constant-free query. Section 6 proves this formally for our query languages. Now, it is clear that our theorem also applies to Boolean quasi-FTAR queries, with the same data complexity. Also, since non-Boolean queries are assumed to only return elements from the domain of the instance, a fixed non-Boolean query has only polynomially many possible outputs, so Theorem 4.8 applies to them as well, but with polynomial rather than linear complexity.

4.4 A Finer Result

Theorem 4.8 is useful in terms of data complexity, but does not give any insight about combined complexity. We now give a stronger (but more complex) statement which highlights the contribution of the query and of the cc-instance's treewidth, both in terms of instance and circuit.

First, we need to give a technical definition that we use to distinguish the contribution of the instance and circuit:

DEFINITION 4.11. *Given a cc-instance J and a tree decomposition T of J, we let $k_I := (\max_{b \in T} |\mathrm{dom}(b) \cap \mathrm{dom}(I)|) - 1$ and likewise for k_C. We call (k_I, k_C) the* mixed width *of T.*

Clearly it holds that $\max(k_I, k_C) \leqslant w(J) \leqslant k_I + k_C + 1$.

Our stronger result considers the mixed width of a tree decomposition provided as input with the cc-instance: it cannot talk about the mixed width of the input instance, as we can no longer justify that a tree decomposition with the prescribed mixed width can be computed in linear time.

There is a second technical definition to give: as our more elaborate result separates the (instance-independent) cost of compiling the query to a bDTA, it takes an automaton as input rather than a query. So we need to impose a condition on this automaton (which is respected in particular by any automaton testing some FTAR query q):

DEFINITION 4.12. *A $\mathrm{fct}_k(\sigma)$-bDTA A is said to be* encoding-invariant *if for any two $\mathrm{fct}_k(\sigma)$-trees E and E' such that $\langle E \rangle$ and $\langle E' \rangle$ are isomorphic, $E \models A$ iff $E' \models A$.*

Let us now state the theorem, that we prove in Section 5:

THEOREM 4.13. *Let T be a tree decomposition of mixed width (k_I, k_C) of a cc-instance J, and A an encoding-invariant $\mathrm{fct}_k(\sigma)$-bDTA with state set Q. We pose $\mathcal{V} = \{\{\mathfrak{t}, \mathfrak{f}\}, Q\}$, There is some $\mathcal{F}$ depending only on A such that one can compute, in time $O(|A| \cdot (|T| + |J|))$, a $(\mathcal{V}, \mathcal{F})$-circuit C and distinguished gate g with domain $\{\mathfrak{t}, \mathfrak{f}\}$ such that for every valuation ν of J_{inp}, $\nu(C)(g) = \mathfrak{t}$ iff $\nu(J) \models q$; and compute a tree decomposition T of C of width $k_C + 5$ where each bag has at most $k_C + 6$ nodes with value set $\{\mathfrak{t}, \mathfrak{f}\}$ and 5 nodes with value set Q.*

Up to polylogarithmic factors, our combined complexity is thus in $O\left(|Q|^2 2^{2k} n_\sigma k^{a_\sigma} (|J| + |T|)\right)$, using the bound on $|\mathrm{fct}_{k_I}(\sigma)|$ established in Section 3. Note that this result does not account for the time needed for the (separate) tasks of compiling the query to A (see Section 6) or of computing probabilities from C (see Section 7). Also note that this theorem has no clear extension to (non-Boolean) quasi-FTAR queries, as it deals with an input automaton, not a query.

We conclude this section by observing that, while bounding the mixed width suffices (as we show) to ensure tractable data complexity for probabilistic query evaluation, bounding the width of I and of C *in isolation* does *not* suffice. Of course, it is already known that for simple queries, probabilistic query evaluation is #P-hard in an input cc-instance J for trivial C and arbitrary I ([18], Theorem 7), or for trivial I and arbitrary C (immediate reduction from #SAT). Yet, we show that even when I and C have constant treewidth, hardness may ensue; intuitively, it is crucial that we have a tree decomposition of J which *simultaneously* decomposes I and C:

PROPOSITION 4.14. *There is a fixed CQ q such that the probabilistic query evaluation for q is #P-hard even when input pcc-instances $J = (I, C, \varphi, \pi)$ are restricted by imposing $w(I) = w(C) = 1$.*

5. PROOF OF MAIN RESULT

In this section, we prove Theorem 4.13. It will be more convenient to work with K-cc-instances, that is, cc-instances annotated with an arbitrary set K, which we now fix.

DEFINITION 5.1. *A* K-instance *is an instance where every fact F is annotated by an element $\alpha(F)$ from K.*

A K-tree-encoding *of width k is a $(\mathrm{fct}_k(\sigma) \times K)$-tree. The* decoding *$\langle E \rangle$ of a K-tree-encoding E is a K-instance obtained in the expected fashion, by labeling each created fact with the annotation*[6] *in the node label. Given a function $f : E \to K$, we write $f(E)$ for the K-tree-encoding obtained from E in the expected way.*

A K-cc-instance $J = (I, C, \varphi)$ is a cc-instance where C is a K-circuit. For any valuation ν of C_{inp}, $\nu(J)$ is the K-instance where each fact F is annotated by $\nu(C)(\varphi(F))$.

For usual cc-instances, the value set K is $\{\mathfrak{t}, \mathfrak{f}\}$, so in $\nu(J)$ facts are annotated with $\mathfrak{t}$ if they must be kept and $\mathfrak{f}$ if they must be discarded. There is no important difference between annotating facts with $\mathfrak{f}$ and actually discarding them, as automata can be rewritten from one case to the other (see Appendix for details).

CC-encodings. We first explain how K-tree-encodings can be generalized for K-cc-instances: we encode the underlying instance while retaining the original circuit to provide the (yet unknown) annotations.

DEFINITION 5.2. *A* K-cc-encoding *of width (k_I, k_C) is a tuple $E' = (E, C, T, \chi)$ of an (unannotated) tree encoding E of width k_I, a K-circuit C, a tree decomposition T of C of width k_C with same skeleton as E, and a mapping $\chi : T \to C$ selecting a* distinguished 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 see $\nu(C)$ as an annotation function on E by setting $\nu(C)(n) := \nu(C)(\chi(b))$ for the bag b of T corresponding to n in E, and write $\nu(E')$ the resulting K-tree-encoding.

We can compute a K-cc-encoding of our K-cc-instance $J = (I, C, \varphi)$ by "splitting" its tree decomposition T in a tree decomposition of C and a tree encoding 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 5.3. *Given a K-cc-instance $J = (I, C, \varphi)$ and a tree decomposition T of J of mixed width (k_I, k_C), one can compute a K-cc-encoding $E' = (E, C', T', \chi)$ of width (k_I, k_C), with $C = C'$, such that for any valuation ν of C_{inp}, $\nu(C)(E')$ is an encoding of $\nu(J)$. The computation is in $O(|T| + |C|)$.*

[6]The annotations are lost for nodes with 0-fact structures.

EXAMPLE 5.4. *Figure (e) represents a $\{\mathfrak{t}, \mathfrak{f}\}$-cc-encoding E'_U: it combines the tree encoding E_U of I_U and a tree decomposition T'_C of C constructed from T_C (the g_i nodes), which has same skeleton as E_U. Note that the image by φ of the instance facts for the nodes of E_U are present (and underlined) in the corresponding bag of T'_C.*

Run circuits. Forgetting temporarily the original circuit C, we now describe how the run of a $(\mathrm{fct}_k(\sigma) \times K)$-bDTA A on the encoding E, with its yet-unknown annotations, can be "instrumented" as a *run circuit* C'' following E, with C''_{inp} waiting to receive each node's annotation.

DEFINITION 5.5. *Let E be a $\mathrm{fct}_k(\sigma)$-tree, and A be a bDTA on $(\mathrm{fct}_k(\sigma) \times K)$ with states Q. A* run circuit *of A on E is a tuple $(C, T, \xi, g^{\mathrm{o}})$ where C is a $\{K, Q, \{\mathfrak{t}, \mathfrak{f}\}\}$-circuit, T is a tree decomposition of C with same skeleton as E, ξ is a bijection from T to C_{inp}, and g^{o} is a distinguished gate of C with value set $\{\mathfrak{t}, \mathfrak{f}\}$.*

For any annotation function $f : E \to K$, letting ν be the corresponding valuation of C_{inp} (for every bag b of T, $\nu(\xi(b)) = f(n)$ where n is the node of E corresponding to $\xi(b)$), we require that $\nu(C)(g^{\mathrm{o}}) = \mathfrak{t}$ iff $f(E) \models A$.

Intuitively, a run circuit computes the result of running A on E depending on the annotation of E (which are fed to the gates indicated by ξ). We now justify that run circuits of bounded treewidth can be constructed:

LEMMA 5.6. *Assume that K is finite. Let $k \in \mathbb{N}^*$, and A be $(\mathrm{fct}_k(\sigma) \times K)$-bDTA with state space Q. Let $\mathcal{V} = \{K, Q, \{\mathfrak{t}, \mathfrak{f}\}\}$. There exists a function set $\mathcal{F}$ such that, for any tree encoding E of width k, one can compute in $O(|E| \cdot |A|)$ a run $(\mathcal{V}, \mathcal{F})$-circuit $(C, T, \xi, g^{\mathrm{o}})$ of A on E such that $w(T) = 4$.*

Proof sketch. The circuit has the same structure as E with one input gate per $n \in E$ representing the annotation of n, and one state gate with domain Q representing the state of the automaton at n, whose inputs are the input gate and the state gate of children (if any). The function set $\mathcal{F}$ encodes the transitions of the automaton. □

Circuit stitching. We now need to "stitch together" the cc-instance circuit C and the run circuit C''.

DEFINITION 5.7. *Let $C = (G, W)$ and $C' = (G', W')$ 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 $(G \cup G', W \cup W')$. In particular, $(C \circ C')_{\mathsf{inp}} = C_{\mathsf{inp}}$.*

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:

LEMMA 5.8. *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 $C \circ C'$.*

Concluding. We now sketch the overall proof. We compute from the cc-instance J and its decomposition T a cc-encoding $E' = (E, C, T', \chi)$, and from the $(\mathrm{fct}_{k_I}(\sigma) \times K)$-bDTA A (having added $K = \{\mathfrak{t}, \mathfrak{f}\}$) and the unannotated encoding E we compute a run circuit C'' of E on A and tree decomposition T'' of C'. Now C and C'' being essentially stitchable and having tree decompositions T' and T'' with same skeleton, we stitch them, yielding the desired circuit and tree decomposition. Correctness follows from the fact that C'' correctly instruments the run on A on E', that is, on E following the annotations given by the original circuit C.

Theorem 4.13 implies Theorem 4.8 as the tree decomposition can be computed in linear time even if not given in the input, and we can rewrite the gates with domain Q representing the automaton state by Boolean gates, which does not change data complexity or the treewidth beyond a multiplicative constant.

6. REWRITING QUERIES TO AUTOMATA

In this section, we give a survey of query languages which are FTAR (recall Definition 4.7). We show that the very expressive language of *guarded second-order logic* is FTAR, and then study the complexity of compiling the automaton, as a function of the query and treewidth, for more restricted query languages: unions of conjunctive queries (UCQ), and frontier-guarded datalog. We close by an example of a non-FTAR language, namely Datalog.

We first look at MSO queries, well-known to be rewritable to tree automata on bounded treewidth instances [22,51]. We restate this result in our setting:

THEOREM 6.1. *Constant-free MSO sentences are FTAR.*

Proof sketch. The main problem is to justify that MSO sentences can be rewritten to MSO sentences on our tree encodings, as we can then use [51] to compile them to a bDTA. We rely on [22] 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 bDTA. □

While MSO captures many useful query languages (in particular, all FO queries), it does not capture languages such as Datalog or its various restrictions, which we now define:

DEFINITION 6.2. *A* Datalog query *P over the signature σ consists of a signature σ_{int} of* intensional predicates *with a special relation* Goal() *(exceptionally with arity 0) 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. The query is called* monadic *[1] if all the intentional predicates are unary,* guarded *[28] if all the variables of the body occur together in some body atom, and* frontier-guarded *[7] if all the variables of the head atom appear together in some body atom. The semantics of P is that $I \models P$ whenever the fact* Goal() *can be derived on I by applying the rules of P (see [1] for formal definitions).*

While monadic Datalog is captured by MSO, is it true that guarded Datalog is FTAR? What about the more expressive frontier-guarder Datalog? In fact, it is known that these languages can be expressed in an SO fragment called *guarded second-order logic* (GSO). We define GSO and show that it captures frontier-guarded Datalog (and thus guarded Datalog):

DEFINITION 6.3. Guarded Second-Order logic *(GSO) is the restriction of SO where quantification on second-order variables is* semantically *restricted so that it only applies to the* guarded tuples*, namely, the tuples formed by elements which occur together in some fact of the instance.*

PROPOSITION 6.4. *Any frontier-guarded Datalog program can be expressed as a GSO formula.*

We now show the following, using a result of [29] about GSO queries being rewritable to MSO queries on an *incidence instance* which has roughly the same treewidth:

THEOREM 6.5. *Constant-free GSO sentences are FTAR.*

Hence, in terms of data complexity, Theorem 4.8 applies to any fragment of GSO. The result generalizes to GSO formulae with free variables and constants:

COROLLARY 6.6. *GSO sentences (resp., queries) are quasi-FTAR (resp., non-Boolean quasi-FTAR).*

However, when we are concerned with combined complexity, we must account for the (data-independent) step of rewriting the query to a bDTA, and the complexity of this task for MSO (and hence GSO) is nonelementary [25]. We therefore turn to studying the complexity of this task for less expressive fragments of GSO, to prove better complexity bounds.

DEFINITION 6.7. *Let σ be a fixed signature. The* rewriting complexity *of a FTAR fragment is the complexity of rewriting a query q from this fragment to a $\text{fct}_k(\sigma)$-bDTA A_q that tests the query, as a function of the query size $|q|$ and treewidth k.*

First, we consider *unions of conjunctive queries*. A *conjunctive query* is an existentially quantified conjunction of positive atoms and unions of conjunctive queries (UCQs) are disjunctions of CQs.

PROPOSITION 6.8. *[15] The rewriting complexity of UCQs is in 2-EXPTIME in $|q|$ and k.*

Second, we look at frontier-guarded Datalog:

PROPOSITION 6.9. *[12] The rewriting complexity of frontier-guarded datalog is in 3-EXPTIME in $|P|$ and k.*

We conclude by an example of a non-FTAR language:

PROPOSITION 6.10. *Datalog is not FTAR.*

7. PROBABILITY EVALUATION

In this section, we show how probability evaluation can be tractably performed on bounded-treewidth circuits. Together with Theorem 4.8, this implies that FTAR query evaluation is tractable on bounded-treewidth pcc-instances. The result is based on an existing technique to determine marginal probabilities in bounded-treewidth Bayesian networks, through *belief propagation.*

Recall our definition of circuit from Section 3, and the notion of tree decomposition of a circuit.

DEFINITION 7.1. *The* probability evaluation problem *for a circuit C, distinguished gate g, and* probabilistic valuation *π associating each $g' \in C_{\mathsf{inp}}$ to a probability distribution $\pi_{g'}$ on its value set $V(g')$, is to compute the probability distribution of g under the product distribution for the inputs (i.e., assuming independence), that is, $\Pr_g$ mapping $d \in V(g)$ to*

$$\sum_{\substack{\text{valuation } \nu \\ \nu(C)(g)=d}} \prod_{g' \in C_{\mathsf{inp}}} \pi_{g'}(\nu(g')).$$

As probability evaluation requires computation over rational probabilities, we carefully define a notion of tractability:

DEFINITION 7.2. *An algorithm runs in* ra-linear time *(for* rational-arithmetic linear time*) if it runs in linear time assuming that arithmetic operations over rationals take constant time and rationals are stored in constant space, and runs in polynomial time without this assumption.*

We can now state our main result for this section:

THEOREM 7.3. *Let $\mathcal{V}$ be a set of* finite *value sets, and let $k_1, \ldots, k_{|\mathcal{V}|}$ be integers. Given a tree decomposition T of a $\mathcal{V}$-circuit C such that the number of gates of value set V_i in any bag of T is less than k_i for all $1 \leqslant i \leqslant |\mathcal{V}|$, and given a probabilistic valuation $\pi_{g'}$ for every $g' \in C_{\mathsf{inp}}$, the probability evaluation problem for C, a gate g, and π can be solved in time* ra-linear *in $|T| \times h(\mathbf{k}) + |\pi| + |C|$, where $h(\mathbf{k}) = \prod_i |V_i|^{k_i}$.*

Proof sketch. The result is by the known technique of *belief propagation* (or *sum-product message passing*) [9,11,43] applied to bounded treewidth circuits. We introduce a potential function for every bag and edge of T, initialize them using the distribution π and the constraints imposed by the functions of $\mathcal{F}$, and apply belief propagation (amounting to one bottom-up and one top-down pass), so our desired distribution is obtained by marginalizing the potentials. $\square$

As for Theorem 4.13 we must assume that T is given because we can no longer argue it is computable in PTIME; but this problems disappears if we are only interested in data complexity, yielding the simpler corollary:

COROLLARY 7.4. *Fix the value sets $\mathcal{V}$ and $k \in \mathbb{N}^*$. Given a circuit C with $w(C) \leqslant k$, $g \in C$ and a probabilistic valuation π, the probability evaluation problem can be solved in time ra-linear in $|C| + |\pi|$.*

We can now combine Theorem 4.8 with Corollary 7.4 and finally obtain the desired tractability result for FTAR query evaluation over bounded-treewidth pcc-instances:

THEOREM 7.5. *The probabilistic query evaluation problem for FTAR queries on bounded-treewidth pcc-instances can be solved in ra-linear time data complexity.*

Of course, this result has immediate consequences for the *possibility* and *certainty* problem, namely, that of determining whether a query is satisfied in some possible world or in all possible worlds of an input cc-instance:

COROLLARY 7.6. *The query certainty and possibility problems for FTAR queries on bounded-treewidth cc-instances can be solved in ra-linear time.*

Note that belief propagation additionally allows efficient computation of other properties [33] of the probability distribution over circuit gates: conditional distributions, arbitrary marginals, etc.

8. APPLICATIONS

We now investigate the consequences of Theorem 4.8 for probabilistic query evaluation on various existing models. All results are stated for Boolean MSO queries for readability; however, they actually hold for all quasi-FTAR query languages (see Section 6), and for non-Boolean queries if linear-time claims are demoted to PTIME (see Section 4.3).

8.1 Relational Models

We first study relational probabilistic models: pc-instances and block-independent disjoint instances (BID) [8,46].

pc-instances. Recall the definition of (p)c-instances and pc-instances, which intuitively are (p)cc-instances where each tuple is mapped to a tree-shaped circuit representation of its formula, with no sharing between trees except the inputs.

Ignoring the structure of the annotation formulae (except event occurrences), we can accordingly define a notion of bounded treewidth for (p)c-instances, which implies the analogue of Theorem 4.8 for (p)c-instances.

DEFINITION 8.1. *Let $\sigma^{\text{o}} = \sigma \cup \{\text{Occ}, \text{Cooc}\}$, 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 $a_e \in \text{dom}(J)$, and where we add a fact* $\text{Occ}(a, a_e)$ *in I_J whenever $a \in \text{dom}(J)$ is used in a fact annotated by a formula involving e, and* $\text{Cooc}(a_e, a_f)$ *whenever events e and f co-occur in the formula of some fact.*

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

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 8.2. *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 in the sense of Proposition 4.2 and has treewidth depending only on k.*

Combining this with Theorem 4.8, we deduce:

THEOREM 8.3. *For bounded-treewidth pc-instances, the probabilistic query evaluation problem for Boolean MSO queries can be solved in ra-linear time data complexity.*

This clearly implies the same claim for tuple-independent databases (TID) [18,42], with the treewidth of a TID being that of the underlying relational instance, as it is straightforward to encode in linear time a TID instance to a pc-instance with the same treewidth up to an additive constant.

BID instances. Following [8,46], we define:

DEFINITION 8.4. *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 8.5. *We define the* treewidth *$w(I)$ of a BID instance I as that of the underlying relational instance, forgetting about the probabilities.*

We are able to show the tractability of MSO query evaluation on BID through Theorem 4.8 thanks to the following:

LEMMA 8.6. *For any fixed $k \in \mathbb{N}^*$, given a BID instance J with $w(J) \leqslant k$, we can compute in ra-linear time an equivalent pcc-instance J' where $w(J')$ depends only on k.*

However, the proof of this result is subtler. 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.

Combining Lemma 8.6 and Theorem 4.8, we can conclude:

THEOREM 8.7. *The probabilistic query evaluation problem for Boolean MSO queries on bounded-treewidth BID instances can be solved in ra-linear time data complexity.*

8.2 Probabilistic XML

We now turn to probabilistic XML models [41].

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

DEFINITION 8.8. *An* XML document *with* label set Λ *(or* Λ-document*) is an* unranked Λ-*tree.*

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 8.9. *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 $Ch_{\swarrow}(n) = n_1$, $Ch_{\searrow}(n_1) = n_2$, ..., $Ch_{\searrow}(n_{k-1}) = n_k$; we complete by nodes labeled $\perp \notin \Lambda$ to make the tree full.*

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

DEFINITION 8.10. *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.*

Unlike other relational encodings of XML [2], we have:

LEMMA 8.11. *The relational encoding I_D of an XML document D has treewidth* 1 *and can be computed in linear time.*

Importantly, the language of MSO queries[7] on XML documents [44], which we now define, can be easily translated to queries on the relational encoding:

DEFINITION 8.12. *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 8.13. *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 $I_D \models q'$.*

Probabilistic XML. We now turn to the setting of *probabilistic* XML documents, to present tractability results about probabilistic query evaluation on classes of such documents. The main variant that we study is [41]:

DEFINITION 8.14. *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 $\nu(\Phi) = \mathfrak{t}$ (the others, and their descendants, are discarded). We require the root to have label in Λ.*

We encode $\mathsf{PrXML}^{\mathsf{fie}}$ documents into pc-instances:

DEFINITION 8.15. *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 $\mathfrak{t}$. $P_\lambda(n)$-facts are annotated with the annotation Φ of the edge from the parent of n to n, if Φ exists, with $\mathfrak{t}$ otherwise.*

PROPOSITION 8.16. *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 $\nu'(J_D) \models q'$.*

Of course we cannot hope 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 ([40], Theorem 5.2). A first notion of tractability for a $\mathsf{PrXML}^{\mathsf{fie}}$ document D is the treewidth (following Definition 8.1) of the pc-encoding of D. Indeed, a direct consequence of Proposition 8.16 is:

COROLLARY 8.17. *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.*

However, it is not so easy to compute the width of the relational encoding directly from the $\mathsf{PrXML}^{\mathsf{fie}}$ document, even though it can be computed in linear time if we assume that it is bounded by a constant. We now give a simpler sufficient condition for tractability, whose corresponding notion of width can be evaluated in PTIME on $\mathsf{PrXML}^{\mathsf{fie}}$ documents, even without a-priori bounds. Intuitively, the gist of this criterion is that, given a $\mathsf{PrXML}^{\mathsf{fie}}$ document D, we are looking for tree decompositions of J_D in a certain form: those which mimic exactly the LCRS representation of J_D, with the addition of events in this fixed decomposition.

DEFINITION 8.18. *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 $w_{\mathrm{s}}(D) := \max_{n \in D} |S(n)|$.*

PROPOSITION 8.19. *For any* $\mathsf{PrXML}^{\mathsf{fie}}$ *document D, we have $w(J_D) \leqslant w_{\mathrm{s}}(D) + 1$.*

COROLLARY 8.20. *For* $\mathsf{PrXML}^{\mathsf{fie}}$ *documents with bounded event scope, the MSO probabilistic query evaluation problem on* $\mathsf{PrXML}^{\mathsf{fie}}$ *has ra-linear data complexity.*

Bounded event scope width is a strictly weaker condition than bounded treewidth of the pc-encoding: in some situations, the minimal-width tree decomposition of the XML document has smaller width than the straightforward one.

PROPOSITION 8.21. *There exists a family of* $\mathsf{PrXML}^{\mathsf{fie}}$ *documents D_n such that $w(J_{D_n}) \leqslant 4$ but $w_{\mathrm{s}}(D_n)$ is not bounded.*

As a consequence of Corollary 8.20, we obtain the MSO tractability over the $\mathsf{PrXML}^{\mathsf{mux,ind}}$ model [41] (definitions and details in appendix), a result that was previously proved in [16]:

PROPOSITION 8.22. *The MSO query evaluation problem on* $\mathsf{PrXML}^{\mathsf{mux,ind}}$ *has ra-linear time data complexity.*

9. SEMIRING PROVENANCE

Our main theorem in Section 4 described how to build, from a Boolean query q and cc-instance J, a Boolean circuit C on J_{inp} which describes exactly the possible worlds of J which satisfy q. So, in a sense, C records the dependency of the query result on J_{inp}; we can use it, e.g., to test whether q still holds when changing the valuation.

This is reminiscent of *semiring provenance* [30], which provides a framework to annotate query results with an expression describing how they depend from the input tuples. This section studies under which hypotheses we can generalize our results to a semiring-based provenance construction.

For simplicity, we limit ourselves to constant-free Boolean queries, but the distinction is inessential: for a non-Boolean query, for instance, we can always consider each possible output tuple (there are polynomially many) and build a provenance representation for that tuple.

9.1 Definitions

The main difference between our setting and that of semiring provenance is our use of *negation*. Negation can occur in the circuit of the cc-instance, and it is also needed in our construction of the run circuit, intuitively because the query itself may use negation. Yet, negation is hard to incorporate to semiring-based provenance constructions [4, 26].

We will accordingly require that the circuit of cc-instances

[7]MSO subsumes other languages such as tree-pattern queries [3].

use only semiring operations (in the case of Boolean circuits, $\wedge$ and $\vee$, but no negation), and restrict to queries which are *monotone* in the following sense:

DEFINITION 9.1. *A Boolean query q is* monotone *if for every instance I and subinstance $J \subseteq I$, if $I \models q$ then $J \models q$. Given a monotone query q and instance I, we can define the (possibly empty) set $\widehat{q}(I)$ of minimal subinstances $J \subseteq I$ such that $J \models q$.*

Examples of monotone queries are CQs and UCQs.

We now recall the usual definition of *commutative semirings* (all semirings we consider are commutative), and a corresponding notion of circuits:

DEFINITION 9.2. *A* commutative semiring *is a set K with two binary operations $\oplus_K$ and $\otimes_K$ and two distinguished elements 0_K and 1_K, such that $(K, \oplus_K)$ is a commutative monoid with identity element 0_K, $(K, \otimes_K)$ is a commutative monoid with identity element 1_K, multiplication distributes over addition, and $0_K \otimes_K a = 0_K$ for all $a \in K$. We often drop subscripts for brevity.*

*A K-*circuit *for semiring $(K, \oplus, \otimes, 0_K, 1_K)$ is a circuit with one domain K (that is, $\mathcal{D}_K = \{K\}$), and whose gates are binary gates $\otimes$ and $\oplus$ corresponding to the two operations of the semiring (seen as functions, i.e., $\oplus, \otimes : K \times K \to K$) and constant gates 0 and 1.*

The most general possible semiring to consider for semiring provenance is:

DEFINITION 9.3 [30]. *We define the* positive algebra provenance semiring $\mathbb{N}[X]$ *for a set X of variables as the semiring $(\mathbb{N}[X], +, \times, 0, 1)$ of polynomials with variables in X, with addition and product on polynomials in the usual sense.*

However, by contrast, our definitions and results will be limited to *absorptive semirings*:

DEFINITION 9.4. *A commutative semiring K is* absorptive *if, for every $a, b \in K$, we have $a \oplus (a \otimes b) = a$.*

We now define a notion of semiring provenance of a query:

DEFINITION 9.5. *For a semiring K, we define K-instances as instances annotated by K in the usual way (Definition 5.1).*

Given an absorptive semiring K, a monotone query q and a K-instance I, the provenance *of q on I for K is defined as the following value (in K): $W_K(q, I) = \bigoplus_{J \in \widehat{q}(I)} \bigotimes_{F \in J} \alpha(F)$.*

This definition applies to *any* monotone query, no matter the language. It contrasts with the more "operational" semiring provenance constructions which apply to the positive relational algebra, to Datalog, etc. Yet this definition, though natural, has limitations:

EXAMPLE 9.6. *Consider the CQ $\exists xyz\, R(x,y)R(x,z)$, the instance $I = \{R(a,b)\}$ whose fact is annotated by x, and $K = \mathbb{N}[x]$. We have $\widehat{q}(I) = \{I\}$ and $W_{\mathbb{N}[x]}(q, I) = x$ rather than $x \otimes x$ as would be expected.*

To work around this problem, we generalize Definition 9.5 to minimal subinstances which are *multisets* of facts of the original instance. This allows us to represent that multiple "copies" of a fact may be needed to satisfy a query, while remaining fully general with respect to the query language. This requires several definitions:

DEFINITION 9.7. *A* multiset *is a function M from a finite* domain *(or support)* $\mathrm{dom}(M)$ *to* $\mathbb{N}$. *We define the relation $M \subseteq M'$ if* $\mathrm{dom}(M) \subseteq \mathrm{dom}(M')$ *and for all $s \in \mathrm{dom}(M)$ we have $M(s) \leqslant M'(s)$. We write $x \in M$ to mean that $M(x) > 0$.*

A multi-instance *I is a multiset of facts on* $\mathrm{dom}(I)$. *Where necessary to avoid confusion, we call the ordinary instances* set-instances. *A K-*multi-instance *I is a multi-instance where every fact F of the support of I is annotated by some value $\alpha(F) \in K$ (in addition to the* multiplicity $I(F)$*). A* multi-query *q (as opposed to a* set-query*) is a function from multi-instances to $\{\mathfrak{t}, \mathfrak{f}\}$; given a multi-instance I, we write $I \models q$ if $q(I) = \mathfrak{t}$.*

We say that J is a multi-subinstance *of a multi-instance I if $J \subseteq I$ (as multisets). We say that J is a* multi-subinstance *of a set-instance I if* $\mathrm{dom}(J)$ *is a subset of I. In other words, the multiplicities of the facts of set-instances should be seen as arbitrarily large (*not *equal to* 1*).*

A multi-query q is monotone *if for all multi-instances I and J, $I \models q$ and $J \supseteq I$ imply $J \models q$. For an instance or multi-instance I, we define the set (of multi-instances) $\widehat{q}(I)$ as the minimal multi-subinstances $J \subseteq I$ such that $J \models q$.*

We can show using Dickson's lemma [21]:

LEMMA 9.8. *For every (multi-)instance I and monotone multi-query q, $\widehat{q}(I)$ is finite.*

In this context, we can restate our definition of provenance:

DEFINITION 9.9. *Given an absorptive semiring K, a monotone multi-query q and a K-(multi-)instance I, the* provenance *of q on I for K is $W_K(I, q) = \bigoplus_{J \in \widehat{q}(I)} \bigotimes_{F \in J} \alpha_J(F)^{J(F)}$, where the exponent denotes iterated multiplication by $\otimes_K$.*

Note that the sum is finite by Lemma 9.8 (and the product is finite as well) so there is no need to make assume anything more about the semiring. The definition of provenance for multi-queries allows us to fix the problem of Example 9.6 if we see q as a multi-query which accepts the multi-subinstance $\{R(a,b), R(a,b)\}$, but not $\{R(a,b)\}$. (We stress that the multiplicity of $R(a,b)$ in I must be thought of as infinite, not 1.) However, our restriction to absorptive semirings still makes our provenance unable to keep track of how many times, e.g., the query can be derived from the same multi-subinstance, because absorptivity implies the involutivity of addition ($a \oplus a = a$, taking $b = 1$ in the absorptivity axiom).

Of course, if we are willing to impose the involutivity of product in our semiring K ($a \otimes a = a$ for all $a \in K$) then the problem of Example 9.6 no longer occurs; and the above definition for a multi-query q becomes essentially equivalent to Definition 9.5 for an related set-query q.

We now return to an operational definition of provenance and explain the connection with Definition 9.9. To our knowledge, the most expressive query language that has a well-established semiring provenance construction is Datalog (see [30], Definition 5.1); and that definition matches ours:

PROPOSITION 9.10. *For any Datalog program P, one can compute a monotone multi-query q_P such that, for every absorptive semiring K and (multi-)instance I, $W_K(q_P, I)$ is the provenance of P on I for K in the sense of [30].*

Of course, this result generalizes to more restricted languages, such as CQs or UCQs, that can be seen as Datalog programs. However our definition of provenance, while restricted to absorptive semirings, can represent the provenance of *arbitrary* monotone multi-queries.

9.2 Main provenance result and consequences

We now prove that our notion of provenance can be efficiently computed for FTAR multi-queries and absorptive semirings. Informally (see Appendix for details), FTAR multi-queries are the queries testable by automata on $\mathrm{fct}_k(\sigma)$-trees with nodes annotated with multiplicity *up to p*, for some fixed p, ensuring that the alphabet is finite.

THEOREM 9.11. *Recall Definition 5.1 of K-cc-instances. Let q be a monotone FTAR multi-query, K an absorptive semiring, and $k \in \mathbb{N}^*$. For any K-cc-instance J of treewidth $\leqslant k$, one can compute in time linear in $|J|$ a K-circuit C with $C_{\mathsf{inp}} = J_{\mathsf{inp}}$ and a distinguished gate g of C such that for any valuation ν of J_{inp}, $\nu(C)(g) = W_K(q, \nu(J))$, and the treewidth* *of C only depends on q and k.*

In particular, this result implies that the provenance of q on a K-*instance* I can be represented as a bounded-treewidth circuit (take the K-cc-instance J with a trivial circuit that maps each fact of I to a dedicated input gate).

Proof sketch. We first connect the monotonicity of q to a notion of monotonicity for non-deterministic tree automata, connect multi-subinstances to a notion of subencodings, and introduce provenance for automata on encodings matching that of queries on instances. We then construct a *provenance circuit*, an analogue of a run circuit that uses only semiring operations, intuitively introducing one gate per automaton state and node. We last use Lemmas 5.8 and 5.3 to connect it with the original cc-instance circuit. □

We conclude by answering two natural questions:

Which monotone multi-queries are FTAR? We defer to future work a more general study, but prove that at least UCQs are, either when seen as a Datalog program or (equivalently) seeing them as the multi-query where a CQ match requires an instance fact to appear at least as many times as the number of atoms which match to it. So Theorem 9.11 applies to the provenance of UCQs (for any absorptive semiring).

PROPOSITION 9.12. *Let q be a UCQ. The corresponding multi-query defined via Proposition 9.10 through a direct encoding of q to Datalog is FTAR.*

However, when we are not interested in multiplicity and are ready to impose $x \otimes x = x$ in K, for instance for $K = \mathrm{PosBool}[X]$ [30], it is clear that all FTAR set-queries are FTAR when seen as multi-queries that only distinguish multiplicities 0 and > 0. Hence, Theorem 9.11 also applies to monotone FTAR set-queries, for the provenance of Definition 9.5. For relational instances, this relates to *why-provenance* [13, 14], as the resulting circuit intuitively describes the minimal subinstances of I where the query holds.

Which semirings are absorptive? Examples of absorptive semirings, in addition to $\mathrm{PosBool}[X]$, are the *security semiring* [5, 23] computing the security clearance required to see query results, the *fuzzy semiring* (see e.g. [4]) computing the minimal fuzziness value that must be tolerated in a fact, and the *tropical semiring* (see e.g. [20]) computing the minimal cost of a multi-subinstance satisfying the query. In all but the last one, $a \otimes a = a$ holds for all $a \in K$.

A key insight [20] is that provenance for any absorptive semiring can be obtained as the specialization, by a unique semiring homomorphism, of the provenance for the most general semiring $\mathrm{Sorp}[X]$, defined as the quotient of $\mathbb{N}[X]$ by the smallest equivalence relation that performs absorptivity simplifications. (See Appendix for details.)

10. RELATED WORK

Bounded treewidth. From the original results [17, 22] on linear-time data complexity of MSO evaluation on bounded treewidth structures, work such as [6] has investigated counting problems, including applications to probability computation (there, on graphs). In this context, a recent paper [11] is perhaps closest to ours, as it shows the linear-time data complexity of evaluating an MSO query on a bounded-treewidth probabilistic network (analogous to a circuit). The angle is different, however, as the work thinks of queries *over the network*, not over an instance annotated with the network, so the connection to probabilistic database frameworks is not immediately obvious. Perhaps more importantly, such works do not decouple the computation of a bounded-treewidth *lineage* of the query and the *application* of probabilistic inference on this lineage, as we do. We argue that this makes our proof more modular, and that the circuit representation of the lineage has independent interest, as is shown by our link with provenance. We also note counting results from another approach [45] on bounded-treewidth structures, based on monadic Datalog, but they do not address probability computation.

Probabilistic databases. A well-known line of work has investigated the tractability of probabilistic query evaluation on the tuple-independent (TID) relational model. Following [37], a first approach is the "extensional" one, which evaluates probabilities on the query plan; it is restricted to *safe* (or *hierarchical*) queries [24], which were later characterized [18, 19]. The "intensional" approach is closer to ours: it computes a lineage of the query and evaluates its probability via general purpose methods. While applicable to all databases, it is slower in practice [37], and tractability depends on the resulting lineage. Example of tractable lineage classes are read-once formulas [49], or more generally those with bounded *expression treewidth* [38]; the latter relates to our setting as the expression treewidth of a formula is the minimal treewidth of a circuit that represents it. Such works give conditions to ensure tractability of the lineage [38, 39], but only based on the query (e.g., the safe queries), not the instance; and they are limited to the TID model, and to CQs or UCQs. Of course, if the lineage of an arbitrary query happens to be tractable on the instance, this can be useful in practice [49]; both [49] and [47] propose characterizations, dependent on the data, for CQ without self-joins to have read-once lineage on the TID model. Our approach deals with a much larger class of queries, and a much richer probabilistic model, and proposes efficient probability evaluation even when the lineage is not read-once but bounded-treewidth. We leave for future work to investigate how our results relate to those of [49] and [47].

Other works [48] have considered probabilistic database models annotated with graphical models. Those are very related to pcc-instances, as circuits can be seen as a special case of graphical models; but once again there are no guarantees of tractability of the lineage from conditions on the data.

Provenance. Our study of provenance is inspired by the usual context of semiring provenance [30]. The closest work that we are aware of, which introduces the $\mathrm{Sorp}[X]$ semiring and circuits for provenance, is [20]. However, to our knowledge, our definition of absorptive provenance for expressive query languages is new, as is our study of provenance for bounded-treewidth instances.

11. CONCLUSION

The main lesson of the current work is that, by the simple mechanism of automaton instrumentation, we are able to combine existing techniques for tree decomposition of instances, compilation of queries to automata, and probabilistic inference on bounded-treewidth networks. Further, we have illustrated a connection between provenance in absorptive semirings and our circuit representation of the possible worlds where the query holds, shedding light on how provenance can be understood when evaluating queries on tree decompositions using tree automata.

Future work. A first direction for future work would be to investigate whether boundedness for our notion of treewidth is in some sense "necessary" for tractability, or if it is also implied by weaker assumptions such as bounded clique-width or hypertree width [27].

Our tractability results could be refined in multiple ways: a more subtle analysis of the complexity of enumeration for non-Boolean queries, as in [22], or finer combined complexity results if we follow existing work and assume restrictions on the query [39]. We could also, e.g., study combined complexity for XML documents and automata, to try to capture and generalize the corresponding results of [16].

Last, the practical relevance of our work should be investigated. To which extent is the treewidth real-world probabilistic data smaller than the instance size? Is it possible to combine existing practical techniques to compile query to automata, compute tree decompositions of instances, and perform fast, possibly approximate, belief propagation? We expect that such techniques would in practice run much faster than the theoretical bounds (i.e., not compute the entire automaton, find quickly a tree decomposition of non-minimal width, use sampling, etc.); it would be interesting to determine whether "on the fly" automaton instrumentation could be performed in such contexts.

12. REFERENCES

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

[2] A. Amarilli and P. Senellart. On the connections between relational and XML probabilistic data models. In *BNCOD*, 2013.

[3] S. Amer-Yahia, S. Cho, L. V. Lakshmanan, and D. Srivastava. Minimization of tree pattern queries. *SIGMOD Record*, 30(2), 2001.

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

[5] Y. Amsterdamer, D. Deutch, and V. Tannen. Provenance for aggregate queries. In *PODS*, 2011.

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

[7] V. Bárány, B. ten Cate, and M. Otto. Queries with guarded negation. *PVLDB*, 5(11), 2012.

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

[9] C. M. Bishop. Graphical models. In *Pattern Recognition and Machine Learning*, chapter 8. Springer, 2006.

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

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

[12] P. Bourhis, M. Krötzsch, and S. Rudolph. Query containment for highly expressive datalog fragments. *CoRR*, abs/1406.7801, 2014.

[13] P. Buneman. Curated databases. In *ECDL*, 2009.

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

[15] S. Chaudhuri and M. Y. Vardi. On the equivalence of recursive and nonrecursive datalog programs. In *PODS*, 1992.

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

[17] B. Courcelle. Graph rewriting: An algebraic and logic approach. In *Handbook of Theoretical Computer Science*. Elsevier, 1990.

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

[19] N. Dalvi and D. Suciu. The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM*, 59(6), 2012.

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

[21] L. E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *American J. Math.*, 1913.

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

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

[24] N. Fuhr and T. Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *TOIS*, 15(1), 1997.

[25] F. Gécseg and M. Steinby. *Tree Automata*. Akadéniai Kiadó, Budapest, Hungary, 1984.

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

[27] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions: A survey. In *Mathematical Foundations of Computer Science*. Springer, 2001.

[28] E. Grädel. Efficient evaluation methods for guarded logics and datalog lite. In *LPAR*, 2000.

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

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

[31] T. J. Green and V. Tannen. Models for incomplete and probabilistic information. In *IIDB*, 2006.

[32] J. Hastad. The shrinkage exponent of de Morgan formulas is 2. *SIAM J. Comput.*, 27(1), 1998.

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

[34] J. Huang, L. Antova, C. Koch, and D. Olteanu. MayBMS: a probabilistic database management system. In *SIGMOD*, 2009.

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

[36] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. M. Jermaine, and P. J. Haas. MCDB: a Monte Carlo approach to managing uncertain data. In *SIGMOD*, 2008.

[37] A. K. Jha, D. Olteanu, and D. Suciu. Bridging the gap between intensional and extensional query evaluation in probabilistic databases. In *EDBT*, 2010.

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

[39] A. K. Jha and D. Suciu. Knowledge compilation meets database theory: Compiling queries to decision diagrams. *Theory Comput. Syst.*, 52(3), 2013.

[40] B. Kimelfeld, Y. Kosharovsky, and Y. Sagiv. Query efficiency in probabilistic XML models. In *SIGMOD*, 2008.

[41] 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.

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

[43] 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.

[44] F. Neven and T. Schwentick. Query automata over finite trees. *TCS*, 275(1), 2002.

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

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

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

[48] P. Sen and A. Deshpande. Representing and querying correlated tuples in probabilistic databases. In *ICDE*, 2007.

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

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

[51] 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.

[52] W. Thomas. "Languages, Automata, and Logic". In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, chapter 7. Springer, 1997.

[53] I. Wegener. *The Complexity of Boolean Functions*. Wiley, 1991.

APPENDIX

A. PROOFS FOR SECTION 3 (TECHNICAL PRELIMINARIES)

LEMMA 3.4. *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. 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 $\mathrm{fct}_k(\sigma)$ 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 $\left|\mathrm{dom}(b_\mathrm{p})\right| \leqslant k+1$) 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. $\square$

B. PROOFS FOR SECTION 4 (GENERAL MODEL AND MAIN RESULT)

PROPOSITION 4.2. *For any (p)c-instance J, one can compute in linear time a (p)cc-instance J' whose inputs are the variables X of J, such that for any valuation ν of X, $\nu(J) = \nu(J')$ (and, for the probabilistic version, $\Pr_J(\nu) = \Pr_{J'}(\nu)$).*

Proof. We first show the result for c-instances. Create one input gate g_x per variable x, and create for each tuple a Boolean circuit that represents the formula that annotates that tuple (in linear time in the annotation of the tuple). Map the tuple to the distinguished function gate of this circuit. It is clear that for a valuation ν the possible world of J and J' is the same. The same argument applies to pc-instances, taking $\pi'_J(g_x) := \pi_J(x)$ for every variable x. $\square$

PROPOSITION 4.14. *There is a fixed CQ q such that the probabilistic query evaluation for q is #P-hard even when input pcc-instances $J = (I, C, \varphi, \pi)$ are restricted by imposing $w(I) = w(C) = 1$.*

Proof. We show a reduction from the problem of determining the probability of a propositional formula given as a monotone 2-DNF (disjunctive normal form with only positive literals, and two variables per disjunct). This problem itself is hard by an immediate (Turing) reduction from #MONOTONE-2SAT, the same problem for monotone 2-CNF formulae: to compute the probability of a 2-CNF F, build its 2-DNF negation using de Morgan's rule, replace each literal $\neg x$ by x and change the probability of each variable from p to $1-p$. The resulting negation has probability $1-P$, where P is the probability of the original formula F, and it is a 2-DNF of positive literals.

Consider a monotone 2-DNF formula $F = \bigvee_{1 \leqslant i \leqslant n} (c_i^1 \wedge c_i^2)$, and the fixed conjunctive query $q : \exists xyz\, A(x,y) \wedge B(y,z)$. We encode F to a pc-instance: let $\mathrm{dom}(\widehat{I})$ be a, b_i and c_i for $1 \leqslant i \leqslant n$, create one Boolean variable e_x per variable x in F with the same probability as the variable, and let $\widehat{I}$ be one fact $A(a, b_i)$ with annotation $e_{c_i^1}$ and one fact $B(b_i, c_i)$ with annotation $e_{c_i^2}$ for $1 \leqslant i \leqslant n$. It is immediate that to each valuation ν of the variables of F corresponds a valuation ν' of the events of $\widehat{I}$ with the same probability, and we have $\nu(F) = \mathfrak{t}$ iff $c_i^1 \wedge c_i^2$ is true for some i which occurs iff some $A(a, b_i), B(b_i, c_i)$ occurs in $\nu'(I_F)$, i.e., $\nu'(\widehat{I}) \models q$, so that the probability of F is $q(\widehat{I})$. Now, use the construction of the proof of Proposition 4.2 to encode $\widehat{I}$ into a pcc-instance $J = (I, C, \varphi, \pi)$. The circuit C has treewidth 1: indeed, it does not contain any wire, just input gates. The instance I has treewidth 1 as it is a tree. $\square$

C. PROOFS FOR SECTION 5 (PROOF OF MAIN RESULT)

Lifting to annotated relations. Theorem 4.13 is stated for cc-instances, but for the proof we see them as $\{\mathfrak{t}, \mathfrak{f}\}$-cc-instances. However, those two notions of instances differ as in the first case the "false" facts are removed whereas in the second they are kept (and annotated with "false"). We start by showing that this difference is inessential.

DEFINITION C.1. *For any k-fact $\tau = (d, s) \in \mathrm{fct}_k(\sigma)$, we define the* neutered k-fact $\underline{\tau} = (d, \emptyset)$. *In particular, if $s = \emptyset$ then $\underline{\tau} = \tau$. For $\tau \in \mathrm{fct}_k(\sigma)$ and $b \in \{\mathfrak{t}, \mathfrak{f}\}$, we write τ_b to be τ if b is $\mathfrak{t}$ and $\underline{\tau}$ if b is $\mathfrak{f}$.*

Given a $\{\mathfrak{t}, \mathfrak{f}\}$-tree-encoding E, we define its evaluation $\varepsilon(E)$ *as the (non-annotated) tree encoding that has same skeleton, where for every node $n \in E$ with corresponding node n' in $\varepsilon(E)$, letting $\lambda(n) = (\tau, b) \in \mathrm{fct}_k(\sigma) \times \{\mathfrak{t}, \mathfrak{f}\}$, we have $\lambda(n') = \tau_b$.*

LEMMA C.2. *For any bDTA A on $\mathrm{fct}_k(\sigma)$, one can compute in linear time a bDTA A' on $\mathrm{fct}_k(\sigma) \times \{\mathfrak{t}, \mathfrak{f}\}$ such that $E \models A'$ iff $\varepsilon(E) \models A$.*

Proof. Let $A = (Q, F, \iota, \delta)$. We construct the bDTA $A' = (Q, F, \iota', \delta')$ as follows: $\iota'((\tau, b)) := \iota(\tau_b)$ while $\delta'((\tau, b), q_1, q_2) := \delta(\tau_b, q_1, q_2)$ for all $b \in \{\mathfrak{t}, \mathfrak{f}\}$, $\tau \in \mathrm{fct}_k(\sigma)$, and $q_1, q_2 \in Q$. The process is clearly in linear time in $|A|$. Now, it is immediate that both automata are in the same state when reading E and $\varepsilon(E)$ respectively. □

CC-encodings

LEMMA 5.3. *Given a K-cc-instance $J = (I, C, \varphi)$ and a tree decomposition T of J of mixed width (k_I, k_C), one can compute a K-cc-encoding $E' = (E, C', T', \chi)$ of width (k_I, k_C), with $C = C'$, such that for any valuation ν of C_{inp}, $\nu(C)(E')$ 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 construction described in Lemma 3.4.

Whenever we process a bag $b \in T$, the mapping precomputed with J (see Lemma 3.4) 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 distinguished 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_C of C and that E is a tree encoding of width k_I of I. By construction T' and E have same skeleton, and clearly the process is in linear time in $|T| + |J|$.

It remains to check the last condition. Consider a valuation ν of the inputs of C with domain K. Consider the K-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 which is exactly E except that each node n has an additional annotation in K which is that of the corresponding node in $\nu(C)$. Hence, the result is proven. □

Run circuits

LEMMA 5.6. *Assume that K is finite. Let $k \in \mathbb{N}^*$, and A be $(\mathrm{fct}_k(\sigma) \times K)$-bDTA with state space Q. Let $\mathcal{V} = \{K, Q, \{\mathfrak{t}, \mathfrak{f}\}\}$. There exists a function set $\mathcal{F}$ such that, for any tree encoding E of width k, one can compute in $O(|E| \cdot |A|)$ a run $(\mathcal{V}, \mathcal{F})$-circuit $(C, T, \xi, g^{\mathrm{o}})$ of A on E such that $w(T) = 4$.*

Proof. Fix k, $A = (Q, F, \iota, \delta)$, and $\mathcal{D}$.

We define $\mathcal{F}$ to consist of the function $\mathrm{Final} : Q \to \{\mathfrak{t}, \mathfrak{f}\}$ and of the functions $\mathrm{Init}^\tau : K \to Q$ and $\mathrm{Trans}^\tau : Q \times Q \times K \to Q$ for all $\tau \in \mathrm{fct}_k(\sigma)$. The function Final maps $q \in Q$ to $\mathfrak{t}$ if $q \in F$ and $\mathfrak{f}$ otherwise. The function Init^τ maps $k \in K$ to $\iota(\tau, k)$. The function Trans^τ maps (q_1, q_2, k) to $\delta((\tau, k), q_1, q_2)$.

We create circuit C to have, for every node n of E, one input gate g^{i}_n with value set K and one gate g^{q}_n with value set Q which is a function gate of type $\mathrm{Init}^{\lambda(n)}$ with input g^{i}_n if n is a leaf, and a function gate of type $\mathrm{Trans}^{\lambda(n)}$ with inputs $(g^{\mathrm{i}}_n, g^{\mathrm{q}}_{Ch_{\swarrow}(n)}, g^{\mathrm{q}}_{Ch_{\searrow}(n)})$ if n is an internal node. We add one more gate g^{o} to be a function gate of type Final connected to g^{q}_r, where r is the root node of E. This construction can clearly be performed in linear time: for every node n of E, the number of operations that we perform is in $O(|A|)$.

We first build a tree decomposition T of circuit C with same skeleton as E and with width 4. Indeed, take T to have same skeleton as E and:

- for every node n of E with corresponding bag b in T, add gates $\xi(b) := g^{\mathrm{i}}_n$ and g^{q}_n to b;
- for every non-root node n of E with parent n' with corresponding bag b' in T, add gate g^{q}_n to b';
- for the bag b_{r} of T corresponding to the root node of E, add g^{o} to b_{r}.

Clearly each bag b of T corresponding to node n of E contains at most 5 nodes (g^{i}_n, g^{q}_n, optionally $g^{\mathrm{q}}_{Ch_{\swarrow}(n)}$ and $g^{\mathrm{q}}_{Ch_{\searrow}(n)}$, and optionally g^{o}). We now check that every function gate of C is covered in T: g^{o} is covered at the root bag, g^{i}_n is covered in the node where it occurs, and, for every node n of E, g^{q}_n is covered in the bag b of T corresponding to n as all inputs of g^{q}_n are present in this bag). Last, it is clear that the occurrences of all gates in T form a connected subtree (the g^{i}_n's and g^{o} only occur in one bag, the g^{q}_n's occur in one bag or in two parent-child bags).

We then argue that C is indeed a run circuit of A on E. Consider a function $f : E \to K$, and ν the corresponding valuation of the inputs of C. Consider the run of A on $f(E)$ and, for every node n of E, call q_n the state of A at the corresponding node of $f(E)$. We show by a bottom-up induction on E that $\nu(C)(g^{\mathrm{q}}_n) = q_n$ for all $n \in E$.

Base case: if n is a leaf, then the corresponding node of $f(E)$ has label $(\lambda(n), f(n))$ so that $q_n = \iota(\lambda(n), f(n))$ and, because $\nu(C)(g^{\mathrm{i}}_n) = \nu(g^{\mathrm{i}}_n) = f(n)$, we have $\nu(C)(g^{\mathrm{q}}_n) = \mathrm{Init}^{\lambda(n)}(\nu(C)(g^{\mathrm{i}}_n)) = \iota(\lambda(n), f(n)) = q_n$ so the result holds.

Induction: if n is an internal node, then the corresponding node of $f(E)$ has label $(\lambda(n), f(n))$ and we have:

$$q_n = \delta((\lambda(n), f(n)), q_{Ch_{\swarrow}(n)}, q_{Ch_{\searrow}(n)}).$$

Now:

$$\nu(C)(g^{\mathrm{q}}_n) = \mathrm{Trans}^{\lambda(n)}(\nu(g^{\mathrm{i}}_n), \nu(C)(g^{\mathrm{q}}_{Ch_{\swarrow}(n)}), \nu(C)(g^{\mathrm{q}}_{Ch_{\searrow}(n)}))$$

By induction hypothesis, we have $\nu(C)(g^{\mathrm{q}}_{Ch_{\swarrow}(n)}) = q_{Ch_{\swarrow}(n)}$ and $\nu(C)(g^{\mathrm{q}}_{Ch_{\searrow}(n)}) = q_{Ch_{\searrow}(n)}$, and we have $\nu(g^{\mathrm{i}}_n) = f(n)$, so that by unfolding the definition of $\mathrm{Trans}^{\lambda(n)}$ we get $\nu(C)(g^{\mathrm{q}}_n) = q_n$, so the result holds.

We conclude by observing that, because $\nu(C)(g^{\mathrm{q}}_r) = q_r$, by definition of Final, we have $\nu(C)(g^{\mathrm{o}}) = \mathfrak{t}$ iff $q_r \in F$, that is, iff A accepts $f(E)$. This concludes the proof. □

Circuit stitching. The fundamental property of stitching is:

LEMMA C.3. *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. □

DEFINITION C.4. *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 C.5. *Given two tree decompositions with same skeleton T and T' of fixed width k and k' for a $\mathcal{V}$-circuit C and a $\mathcal{V}'$-circuit C', $T + T'$ can be computed in linear time in T and T' and has width $\leqslant k + k' + 1$ with $\leqslant k + 1$ nodes from $\mathcal{V}$ and $\leqslant k' + 1$ nodes from $\mathcal{V}'$ in each bag.*

LEMMA 5.8. *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:

1. 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.

2. 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'. □

Concluding. We are now ready to prove Theorem 4.13. Fix the input signature σ, the input cc-instance $J = (I, C, \varphi)$ and its tree decomposition T of width (k_I, k_C). Set $K = \{\mathfrak{t}, \mathfrak{f}\}$ (which is finite). Fix the bDTA $A = (Q, F, \iota, \delta)$. Lift it in linear time to a bDTA A' on $\mathrm{fct}_{k_I}(\sigma) \times \{\mathfrak{t}, \mathfrak{f}\}$ by Lemma C.2. Fix $\mathcal{V} = \{K, Q\}$, and see J as a K-cc-instance.

Use Lemma 5.3 on J and T to compute in time $O(|T| + |J|)$ a cc-encoding $E' = (E, C, T', \chi)$ of width (k_I, k_C) of J such that for any valuation ν of C_{inp}, $\nu(E')$ is a K-tree-encoding of $\nu(J)$.

Now, use Lemma 5.6 on A' and E to compute, in time $O(|E| \cdot |A'|)$, a function set $\mathcal{F}$ depending only on A', a run $(\mathcal{V}, \mathcal{F})$-circuit $(C'', T'', \xi, g^{\mathrm{o}})$ of A' on E.

Now, rename C''_{inp}: for every node n of E with corresponding bag b' in T' and b'' in T'' (remember that both T' and T'' have same skeleton as E), rename $\xi(b'')$ to $\chi(b')$. This renaming is performed both in C'' and in T''. This ensures that C and C'' are stitchable, so we compute in time linear in $|C|$ and $|C''|$ the stitching $C''' = C \circ C''$ (which is a $(\mathcal{V}, \mathcal{F})$-circuit), and, because the tree decompositions T' and T'' satisfy the conditions of Lemma 5.8, $T''' = T' + T''$ is a tree decomposition of C'''. By Lemma C.5 it has width $k_C + 5$, and each bag contains at most $k_C + 6$ nodes of value set K and 5 nodes of value set Q (the counting by value set requires an immediate strengthening of Lemma C.5).

It remains to check that, picking as distinguished gate of C''' the distinguished function gate g^{o} of C'', for any valuation ν of J_{inp}, A' accepts a tree encoding E_ν of $\nu(J)$ iff $\nu(C''')(g) = \mathfrak{t}$.

Fix such a valuation ν. By Lemma C.3, fixing ν'' to be the restriction of $\nu(C)$ to $C''_i np$, we have $\nu(C''')(g^{\mathrm{o}}) = \nu''(C'')(g^{\mathrm{o}})$. Now, by definition of run circuits, this value describes whether $f(E) \models A'$ where f is the annotation function corresponding to ν''. So we have $\nu''(C'')(g^{\mathrm{o}}) = \mathfrak{t}$ iff $f(E) \models A'$, that is, iff $\varepsilon(f(E)) \models A$. Now we know by Lemma 5.3 that $f(E)$ is an annotated encoding of $\nu(J)$ (seen as a K-cc-instance), so that $\varepsilon(f(E))$ is an annotated encoding of $\nu(J)$ (seen as a cc-instance). Now as A is encoding-invariant, for any tree encoding E_ν of $\nu(J)$, $E_\nu \models A$ iff $\nu(C''')(g^{\mathrm{o}}) = \mathfrak{t}$. Hence, Theorem 4.13 is proven.

It is now easy to see that Theorem 4.13 implies its simplified version, Theorem 4.8. The first thing to notice is that from an input cc-instance J of treewidth assumed to be k, we can compute in time linear in $|J|$ a tree decomposition T of width (k, k) of J.

The second thing to notice is that multivalued gates over the domain Q (which does not depend on the instance) can be encoded as a tuple of gates with domain $\{\mathfrak{t}, \mathfrak{f}\}$, encoding explicitly with Boolean gates the functions of $\mathcal{F}$. As this is instance-independent, it can be performed in linear-time in the instance, and the treewidth remains independent on the instance.

Note that the complexity bound of $|A| \cdot |J|$ in Theorem 4.13 could be tightened by noticing that Lemma 5.6 performs for each node of E a number of operations which is in $O(|Q|^2)$ (up to polylogarithmic factors); the size of $\mathrm{fct}_{k_I}(\sigma)$ does not intervene. Hence, performing the lifting operation of Lemma C.2 online rather than on the entire automaton, it could be shown that the exponential blowup k_I only intervenes in the automaton compilation phase, but that, if the number of states of the result is small, it does not intervene in the instrumentation result. In fact, the $|Q|^2$ factor could further be bounded by the number of states which are "useful" (reachable) on the provided encoding. We leave these considerations for future work.

D. PROOFS FOR SECTION 6 (REWRITING QUERIES TO AUTOMATA)

THEOREM 6.1. *Constant-free MSO sentences are FTAR.*

Proof. Let us fix $k \in \mathbb{N}^*$. We denote by $[m]$ be the the set $\{1, \cdots, m\}$ 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 $(\gamma_1, \gamma_2, \cdots, \gamma_{n_\sigma+2})$. 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 $2^{k^{\text{arity}(R_i)}}$ 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 explicitely (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 $\text{fct}_k(\sigma)$-tree T is *well-formed* if $\langle T\rangle$ is *well-defined*, namely, different from $\perp$.

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 $n' \in \{Ch_\swarrow(n), Ch_\searrow(n)\}$ 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 $(j_1, \cdots, j_l)$ in γ'_m and if for each b (i_b, j_b) in γ'_2, then $(i_1, \cdots, i_l)$ 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 $(j_1, \cdots, j_m)$ in γ_i, then the fact $R_i(\nu(j_1), \cdots, \nu(j_m))$ 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_Γ on $\Gamma(\sigma,k)$-trees such that for any $\Gamma(\sigma,k)$-tree T, A_Γ accepts T iff $T \models \varphi^*$, that is, iff $\langle T\rangle'$ is well-defined and satisfies φ. Note that this implies that A_Γ is encoding-invariant. We now explain how to translate A_Γ to our desired bDTA A_{fct} over $\text{fct}_k(\sigma)$ such that for every $\text{fct}_k(\sigma)$-tree T, A_{fct} accepts T iff $\langle T\rangle \models \varphi$.

We consider the alphabet $\Sigma = \text{fct}_k(\sigma) \times \Gamma(\sigma,k)$, and call π_1 and π_2 the operations on Σ-trees that map them respectively to $\text{fct}_k(\sigma)$ and $\Gamma(\sigma,k)$ trees with same skeleton by keeping the first or second component of the labels. Given a $\text{fct}_k(\sigma)$-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 $\text{fct}_k(\sigma)$-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_Γ, we can build an Σ-bDTA A'_Γ such that T is recognized by A'_Γ iff $\pi_2(T)$ is accepted by A'_Γ, and build A_{fct} as the conjunction of A_{t} and A'_Γ, projected to the first component (accept a $\text{fct}_k(\sigma)$-tree T_1 iff there is some Γ-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_{fct} thus defined accepts a $\text{fct}_k(\sigma)$-tree T_1 iff the Σ-tree $T_1 \times T_2$ is accepted, for some $\Gamma(\sigma,k)$-tree T_2, by A'_Γ and A_{t}: if this happens then T_2 is accepted by A_Γ 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'_Γ 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 $\text{fct}_k(\sigma)$ 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, to 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 $\pi_2(n)$.

□

PROPOSITION 6.4. *Any frontier-guarded Datalog program can be expressed as a GSO formula.*

Proof. We can translate a frontier-guarded Datalog query to an equivalent formula in the guarded negation fragment (GNFP) [BtCS11], as has been explained in [BtCO12], and the translation of GNFP to GSO is directly inspired of the translation of the guarded fixed point fragment [GHO02].

Alternatively, for an intuitive argument, notice that any tuple created in an intensional relation when running a frontier-guarded Datalog program must be a guarded tuple, as it is covered by the match of the body atom that guards the head variables of the rule where it was created. So a frontier-guarded Datalog program only considers guarded tuples. □

THEOREM 6.5. *Constant-free GSO sentences are FTAR.*

Proof. We recall the definition of [GHO02] of an *incidence instance* I' on a signature σ' for an instance I on signature σ:

- the signature σ' includes a unary relation A_R and a binary relation R_i for every relation R in the original signature and $1 \leqslant i \leqslant \text{arity}(R)$
- for any fact $R(\mathbf{a})$ of I, we create a new element e and fact $A_R(e)$ in I', and for each $1 \leqslant i \leqslant \text{arity}(R)$, we create the fact $R_i(e, a_i)$ in I'

Fix the signature σ, $k \in \mathbb{N}^*$, and fix q a constant-free GSO sentence. By Proposition 7.1 of [GHO02], there exists a MSO sentence q' such that, for any instance I of treewidth k, $I \models q$ iff $I' \models q'$, where I' is the incidence instance of I; besides, from the proof of this result, q' is computable from q.

Now, it is clear (this is also mentioned in [GHO02]) that if I has treewidth of k then I' has treewidth of $k+1$, as from a tree encoding of I of width k one can deduce a tree decomposition of I' with same skeleton with width $k+1$, creating for each node a bag containing the domain of the node and the fresh element representing the fact of the node. Moreover, the only propagated values from a node to another are the ones belonging to the domain of I.

We will now go back to the notion of encoding of [FFG02] as it is more convenient to represent in one bag all the facts of I' created for a fact of I. In so doing, we will reuse the notations of the proof of Theorem 6.1. Let $\text{fct}_k(\sigma)$ and $\Gamma_{k+1}(\sigma')$ be the two alphabets as defined here and in [FFG02]. We build a bDTA A_t on $\Sigma := \text{fct}_k(\sigma) \times \Gamma_{k+1}(\sigma')$ such that:

1. if A_t accepts a Σ-tree T then $I_1 = \langle \pi_1(T) \rangle$ and $I_2 = \langle \pi_2(T) \rangle'$ are well-defined and I_2 is isomorphic to the incidence instance of I_1.
2. for any $\text{fct}_k(\sigma)$-tree T, there exists a $\Gamma_{k+1}(\sigma')$ tree T' such that $T \times T'$ is accepted by A_t

Having built such an automaton A_t, as in the proof of Theorem 6.1, we will then be able to perform the intersection of A_t with a bDTA that enforces that a tree $T \times T'$ is accepted only if $\langle T' \rangle' \models q'$ (this bDTA being obtained from a bDTA on $\Gamma_{k+1}(\sigma')$ testing this, which we obtain by encoding q' to a bDTA using the process of [FFG02] (Lemma 4.10) and encoding to a bDTA [TW68]), and then build an automaton A_q that tests q as the projection of A_t on $\text{fct}_k(\sigma)$, proving that query q is FTAR.

The automaton A_t is defined, as in the proof of Theorem 6.1, to check for a certain encoding from $\text{fct}_k(\sigma)$ to $\Gamma_{k+1}(\sigma')$, which we now define: it must be possible to check in MSO whether a Σ-tree follows this encoding, and the encoding must ensure that the encoding of a $\text{fct}_k(\sigma)$-tree T decodes (with $\langle \cdot \rangle'$) to a σ'-instance which is isomorphic to the incidence instance of the decoding of T.

Letting n be a node of the $\text{fct}_k(\sigma)$-tree and $(d,s) := \lambda(n)$, the label of the corresponding node n' of the $\Gamma_{k+1}(\sigma')$-tree is set by picking its domain to be elements representing the elements of d, plus, if s is a non-zero fact, an element representing the fresh element e introduced to represent that fact. The values γ_1 and γ_2 of $\lambda(n')$ are set as in the proof of Theorem 6.1, except for the fresh element (whose index never occurs in them), and $\gamma_3, \ldots, \gamma_{n_\sigma+2}$ are defined in the expected way, noting that the A_R and R_i facts that code the fact of s will be reflected in them. This encoding clearly satisfies the desired properties. □

COROLLARY 6.6. *GSO sentences (resp., queries) are quasi-FTAR (resp., non-Boolean quasi-FTAR).*

Proof. Given a GSO sentence q with constants, we can compute the set C of all constants occurring in q, and consider the set of facts I' which are $P_c(c)$ for every $c \in C$, where P_c is a fresh unary predicate. We can then reduce the problem of evaluating q on an instance I to evaluating the constant-free GSO sentence q' on instance $I \sqcup I'$, where q' is $\forall \mathbf{y} q''(\mathbf{y}) \wedge \bigwedge_{c \in C} P_c(y_c)$, $\mathbf{y}$ is a $|C|$-tuple of variables, and q'' is obtained from q by replacing every constant $c \in C$ by the variable y_c. Hence, q is quasi-FTAR.

For a GSO query q, clearly, for any tuple $\mathbf{a}$ whose arity is the number of free variables of q, $q(\mathbf{a})$ is a GSO sentence, so it is FTAR, so we can take $q_\mathbf{a}$ to be $q(\mathbf{a})$. □

PROPOSITION 6.8. *[CV92] The rewriting complexity of UCQs is in 2-EXPTIME in $|q|$ and k.*

DEFINITION D.1. *A* bottom-up nondeterministic tree automaton *on (binary full)* 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 $\iota : \Gamma \to 2^Q$ *determining 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.* $|A|$ *is* $|Q|^3 \, |\text{fct}_k(\sigma)|$ *up to polylogarithmic factors (intuitively the size of a table for* δ*).*

A run *of* A *on a* Γ*-tree* E *is a function* $\rho : E \to Q$ *such that for each leaf node* n *we have* $\rho(n) \in \iota(\lambda(n))$*, and for every internal node* n *we have* $\rho(n) \in \delta(\rho(Ch_\swarrow(n)), \rho(Ch_\searrow(n)), \lambda(n))$*. A run is* accepting *if, for the root* n_r *of* E*,* $\rho(n_\text{r}) \in F$*; and* A accepts E *(written* $E \models A$*) if there exists some accepting run of* A *on* E*.*

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 program P. 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 of an intentional predicate of σ_{int} and an instanciation 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 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 instanciation 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 program 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 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.

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 instanciation 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 $\langle T\rangle' = \perp$.

Our goal is now to define a datalog program P such that there is a surjective homomorphism from $\Sigma(P)$ to $\text{fct}_k(\sigma)$. Fix σ_{int} to have intensional relations $P_0,\dots,P_{k+1}$ of arity [8] $0,\dots,k+1$. 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 program P contains a number of variables polynomial in k and σ, and the overall size of the program is exponential in a polynomial of k and σ. Last, the size of $\Sigma(P)$ is in $O(|P| * |P|^{a(P)})$, 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 $\text{fct}_k(\sigma)$. 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 $\text{fct}_k(\sigma)$, and it is clearly surjective. Furthermore, it is clear that this application extends to a surjective mapping h' from $\Sigma(P)$-trees to $\text{fct}_k(\sigma)$-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 CQ. We consider the Datalog program 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 program 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 $\text{fct}_k(\sigma)$-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 $\text{fct}_k(\sigma)$ that has size exponential in a polynomial of σ, $|q|$ and k.

Finally, the determinization of A'_P into an equivalent bDTA is performed in exponential time in that bDTA, yielding our 2-EXPTIME bound, and we intersect it with an bDTA (clearly constructible in 2-EXPTIME) that checks whether a $\text{fct}_k(\sigma)$-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 $\text{fct}_k(\sigma)$-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) before we perform the determinization. □

PROPOSITION 6.9. *[BKR14] The rewriting complexity of frontier-guarded datalog is in 3-EXPTIME in $|P|$ and k.*

Proof. [BKR14] defines a query language GQ which is more expressive than frontier-guarded datalog (with global constants appearing in the rules). The satisfaction of a GQ query q_1 on I is defined by first given a valuation ν over the special constants

[8]While we technically disallowed predicates of arity 0 in our definition of instances, there is clearly no problem in this context.

over $\mathrm{dom}(I)$, applying ν to the special constants of GQ to obtain a frontier-guarded Datalog query q_2 and determining as usual whether $I \models q_2$.

Now, Proposition 9 of [BKR14] states that for any GQ query q_1 and a valuation ν of the special constant λ, there exists an alternating two-way tree automata [CGKV88]. which recognizes proof trees whose decoding (as an instance) satisfies the frontier-guarded datalog program obtained from q_1 by applying the valuation. Moreover, the proof of Proposition 9 of [BKR14] shows that this construction is exponential in q'. Frontier-guarded datalog programs are GQ without any special constants and therefore we can conclude from Proposition 9 [BKR14] that for any frontier-guarded Datalog q there exists an exponential alternating two-way tree automata A which recognizes proof trees whose decoding satisfies q, and (from the proof) that this construction is polynomial in the size of A. The translation [CGKV88] of this two-way alternating tree automaton A to an equivalent bNTA A' is in exponential time in $|A|$. Therefore, by the same arguments as in the proof of Proposition 6.8, there exists an bDTA of triple exponential size in $|P|$ and k that recognizes the $\mathrm{fct}_k(\sigma)$-trees T such that $\langle T \rangle$ satisfies the given frontier-guarded Datalog program (the third exponential coming from the determinization). □

PROPOSITION 6.10. *Datalog is not FTAR.*

Proof. Let σ be the signature including the two binary relations Y and Z and the unary relations Begin and End. Consider the following program P, where $I \models P$ for an instance I means that the Goal() predicate is derived:

$$\mathrm{Goal}() \leftarrow S(x,y), \mathrm{Begin}(x), \mathrm{End}(y)$$
$$S(x,y) \leftarrow Y(x,w), S(w,u), Z(u,y)$$
$$S(x,y) \leftarrow Y(x,w), Z(w,y)$$

By way of contradiction, let us assume that the program P is FTAR. Fix the treewidth of instances to be 1, and let A be a $\mathrm{fct}_1(\sigma)$-automaton that tests P.

We consider instances which are chains of facts which are either Y- or Z-facts, and where the first end is the only node labeled Begin and the other is the only node labeled End. This condition on instances can clearly be expressed in MSO, known to be FTAR by Theorem 6.1, so there exists an automaton A_{chain} that tests this property. In particular, we can build the automaton A' which is the intersection of A and A_{chain}, which tests whether instances are of the prescribed form and satisfy property P.

We now observe that such instances must be the instance

$$I_k = \{\mathrm{Begin}(a_1), Y(a_1,a_2), \ldots, Y(a_{k-1},a_k), Y(a_k,a_{k+1}), Z(a_{k+1},a_{k+2}), \ldots, Z(a_{2k-1},a_{2k}), Z(a_{2k},a_{2k+1}), \mathrm{End}(a_{2k+1})\}$$

for some $k \in \mathbb{N}$. Indeed, it is clear that I_k satisfies P for all $k \in \mathbb{N}$, as we derive the facts

$$S(a_k,a_{k+2}), S(a_{k-1},a_{k+3}), \ldots, S(a_{k-(k-1)},a_{k+2+(k-1)}), \text{ that is, } S(a_1,a_{2k+1}),$$

and finally Goal(). Conversely, for any instance I of the prescribed shape that satisfies P, it is easily seen that the derivation of Goal justifies the existence of an I_k-chain in I, which by the restrictions on the shape of I means that $I = I_k$.

We further design a tree automaton A_{encode} which recognizes tree encodings that form a single branch as follows (given from leaf to root), for any integer $n \geqslant 0$ and where X may match either Y or Z: $(\{a_1\}, \mathrm{Start}(a_1))$, $(\{a_1,a_2\}, X(a_1,a_2))$, $(\{a_2,a_3\}, X(a_2,a_3))$, $(\{a_3,a_1\}, X(a_3,a_1))$, $\ldots$, $(\{a_n,a_{n+1}\}, X(a_n,a_{n+1}))$, $(\{a_{n+1}\}, \mathrm{End}(a_{n+1}))$, with addition modulo 3, with dummy nodes $(\perp,\perp)$ added as left children, and left and right children of the leaf node $(\{a_1\}, \mathrm{Start}(a_1))$ to ensure that the tree is full. In other words, A_{encode} enforces that the $\mathrm{fct}_1(\sigma)$-tree encodes the input instance as a chain of consecutive facts with a certain prescribed alternation pattern for elements, with the Begin end of the chain at the top and the End end at the bottom. We define A'' to be the intersection of A' and A_{encode}.

It is easily seen that there is exactly one tree encoding of every I_k which is accepted by A'', namely, the one of the form tested by A_{encode} where $n = 2k$, the first k X are matched to Y and the last k X are matched to Z.

Now, we observe that as A'' is an automaton which is forced to operate on chains (actually, chains completed to full binary trees by a specific addition of binary nodes), we can translate it to a deterministic automaton A''' on words on the alphabet $\Sigma = \{B,Y,Z,E\}$, by looking at its behavior in terms of the X-facts. Formally, A''' has same state space as A'', same final states, initial state $\delta(\iota((\perp,\perp)), \iota((\perp,\perp)))$ and transition function $\delta(q,x) = \delta(\iota((\perp,\perp)), q, (s,f))$ for every domain s, where f is a fact corresponding to the letter $x \in \Sigma$ (B stands here for Begin, and E for End). By definition of A'', A''' recognizes the language $\{BY^kZ^kE \mid k \in \mathbb{N}\}$. As this language is not regular, we have reached a contradiction. This contradicts our hypothesis about the existence of automaton A; so the program P is not FTAR. □

E. PROOFS FOR SECTION 7 (PROBABILITY EVALUATION)

THEOREM 7.3. *Let $\mathcal{V}$ be a set of* finite *value sets, and let $k_1, \ldots, k_{|\mathcal{V}|}$ be integers. Given a tree decomposition T of a $\mathcal{V}$-circuit C such that the number of gates of value set V_i in any bag of T is less than k_i for all $1 \leqslant i \leqslant |\mathcal{V}|$, and given a probabilistic valuation $\pi_{g'}$ for every $g' \in C_{\mathsf{inp}}$, the probability evaluation problem for C, a gate g, and π can be solved in time* ra-linear *in $|T| \times h(\mathbf{k}) + |\pi| + |C|$, where $h(\mathbf{k}) = \prod_i |V_i|^{k_i}$.*

Proof. Fix $T = (B, Ch_{\swarrow}, Ch_{\searrow}, \mathrm{dom})$ a tree decomposition of a $(\mathcal{V},\mathcal{F})$-circuit $C = (G,W)$ (so that for any $b \in B$, $\mathrm{dom}(b)$ is a set of gates of G). We define $E := Ch_{\swarrow} \cup Ch_{\searrow}$ 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 $V(b) = V(g_1) \times \cdots \times V(g_m)$ (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 $h(\mathbf{k})$ 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 function gate, let $\kappa(g)$ be the tuple formed of g and all gates with a wire to g, ordered by $<$. Let $f \in \mathcal{F}$ be the function of g. 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 $(\mathrm{arity}(f)+1)$-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 $h(\mathbf{k})$.

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 $\Phi^{\beta(g)}$:

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

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

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

Remark E.1. *Theorem 7.3 means that we see circuit C as a "lineage" of the query, we can compute its probability efficiently, by computing a tree decomposition, without remembering anything in this query. This is in contrast with other settings [BLRS14] where probability evaluation on the lineage ("grounded inference"; or "intensional approach" [JOS10]) is harder than probability evaluation using the query ("lifted inference").*

Theorem 7.5. *The probabilistic query evaluation problem for FTAR queries on bounded-treewidth pcc-instances can be solved in ra-linear time data complexity.*

Proof. Let $J = (I, C, \varphi, \pi)$ be a pcc-instance of treewidth k and q a query. We use Theorem 4.8 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 $\mathrm{Pr}_g(\mathrm{t})$. We conclude as Theorem 7.3 states that this can be computed in ra-linear time in $|C'| + |\pi|$ for fixed k'. ☐

Definition E.2. *Given a fixed query q and uncertainty framework, the* query certainty problem *(resp.* query possibility problem*) is to determine, given an input uncertain instance J, whether all possible worlds of J satisfy q (resp. whether some possible world of J satisfies q).*

Corollary 7.6. *The query certainty and possibility problems for FTAR queries on bounded-treewidth cc-instances can be solved in ra-linear time.*

Proof. See the cc-instance as a pcc-instance with all probabilities set to $\frac{1}{2}$. The probability of a query is 1 (resp., not 0) on the pcc-instance if the query is certain (resp. possible) on the underlying cc-instance. We can then apply Theorem 7.5. ☐

F. PROOFS FOR SECTION 8 (APPLICATIONS)

F.1 Relational Models (Section 8.1)

pc-instances.

Proposition 8.2. *For any fixed k, given a (p)c-instance J of width ⩽ k, we can compute in linear time a (p)cc-instance J which is equivalent in the sense of Proposition 4.2 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 $\Pr_J(\nu) = \Pr_{J'}(\nu)$), and the annotations of J' have size depending only on k.

Indeed, we observe that by our assumption that $w(J) \leqslant k$, 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 $w(I_J) \leqslant k$ (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. ☐

THEOREM 8.3. *For bounded-treewidth pc-instances, the probabilistic 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 8.2 and Theorem 4.8 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 $w(J) \leqslant k$, 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. ☐

BID instances.

LEMMA 8.6. *For any fixed $k \in \mathbb{N}^*$, given a BID instance J with $w(J) \leqslant k$, we can compute in ra-linear time an equivalent pcc-instance J' where $w(J')$ depends only on k.*

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

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 3.4, 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 3.4, 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 3.4). 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 $w(b) = \pi(F)$ (the probability of F) for $F \in I_{\mathbf{a}}$ and $\beta_{\mathbf{a}}(F) = b$, if any such F exists; $w(b) = 0$ otherwise. We define bottom-up a cumulative weight function w' on $T_{\mathbf{a}}$ as $w(b)$, plus $w'(Ch_{\swarrow}(b))$ if $Ch_{\swarrow}(b) \in T_{\mathbf{a}}$, plus $w'(Ch_{\searrow}(b))$ if $Ch_{\searrow}(b) \in T_{\mathbf{a}}$. For notational convenience we also extend w' to anything by saying that $w'(b) = 0$ if $b \notin T_{\mathbf{a}}$ or b does not exist.

Observe now that for a non-interesting bag b, $w(b)$ and $w'(b)$ can be represented either as 0 or as a pointer to some $w(b')$ or $w'(b')$ for an interesting bag b'. Indeed, if b is non-interesting then we must have $w(b) = 0$. 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 $w'(b) = 0$, or it does exist and all descendants of b that are in the image of $\beta_{\mathbf{a}}$ are descendants of b', so that $w'(b) = w'(b')$ and we can just make $w'(b)$ a pointer to $w'(b')$.

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_b^{i}; for the root bag b it is an input gate with probability $w'(b)$; for other bags it is a gate whose value is defined by the parent bag. Intuitively, g_b^{i} describes whether to choose a fact from $F_{\mathbf{a}}$ within the subtree rooted at b.

For every interesting bag b, writing $w'(b) = k'/d$ and $w(b) = k/d$ with d the common denominator, create one input gate g_b^{h} with probability $\frac{1}{w'(b)} w(b) = k/k'$, 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 $w'(b) > w(b)$ (intuitively: there is still the possibility to generate fact at child nodes), we create one input gate $g_b^{\leftrightarrow}$ which has probability $\frac{1}{w'(b) - w(b)} w'(Ch_{\swarrow}(b))$. Once again, this probability simplifies to a rational whose numerator and denominator are $< d$. We create a gate $g_b^{\swarrow}$ 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 $g_b^{\searrow}$ to be $g_b^{\mathrm{i}} \wedge \neg g_b^{\mathrm{h}} \wedge \neg g_b^{\leftrightarrow}$, setting them to be $g_{Ch_{\swarrow}(b)}^{\mathrm{i}}$ and $g_{Ch_{\searrow}(b)}^{\mathrm{i}}$ where applicable (i.e., if $Ch_{\swarrow}(b)$ and $Ch_{\searrow}(b)$ exist and are in $T_{\mathbf{a}}$).

By contrast, non-interesting bags b just set $g_{Ch_{\swarrow}(b)}^{\mathrm{i}}$ and $g_{Ch_{\searrow}(b)}^{\mathrm{i}}$ (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 $\mathfrak{t}$ is $w'(b)$, and $g_b^{\mathrm{h}\wedge}$, $g_b^{\swarrow}$ and $g_b^{\searrow}$ are either all $\mathfrak{f}$ if g_b^{i} is $\mathfrak{f}$ or, if g_b^{i} is $\mathfrak{t}$, exactly one is true and they respectively have marginal probabilities $w(b)$, $w'(Ch_{\swarrow}(b))$, and $w'(Ch_{\searrow}(b))$. 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. ☐

F.2 Probabilistic XML (Section 8.2)

XML documents.

LEMMA 8.11. *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. ☐

LEMMA 8.13. *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 F.1. *Given label set Λ, we say that an XML document D on $\Lambda \sqcup \{\bot, \mathsf{det}\}$ 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 $\sigma_{\Lambda \sqcup \{\mathsf{det}\}}$ 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 F.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 $\Lambda \sqcup \{\bot, \mathsf{det}\}$ 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 8.13.

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.

PROPOSITION 8.16. *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 F.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 anotation of its parent edge in $\nu(D)$ evaluates to $\mathfrak{t}$; 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 $\mathfrak{t}$ 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 $\bot$ 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 F.3. *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.*

PROPOSITION 8.19. *For any* $\mathsf{PrXML}^{\mathsf{fie}}$ *document D, we have $w(J_D) \leqslant w_{\mathrm{s}}(D) + 1$.*

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. ☐

PROPOSITION 8.21. *There exists a family of* $\mathsf{PrXML}^{\mathsf{fie}}$ *documents D_n such that $w(J_{D_n}) \leqslant 4$ but $w_{\mathrm{s}}(D_n)$ is not bounded.*

Proof. Consider the collection of $\mathsf{PrXML}^{\mathsf{fie}}$ documents D_n of the following form: a root r with two children c_1 and c_2 (and no annotations), and each c_i having children a^i_j for $1 \leqslant j \leqslant n$ with an edge annotated with some event x_i. The family D_n does not have event scope bounded by any finite value, yet the relational encoding of each pc-encoding J_{D_n} admits a tree decomposition of fixed width $\leqslant 4$: we have a root bag with r, c_1, c_2, and n other bags with c_1, c_2, a^1_j, a^2_j, and x_j for $1 \leqslant j \leqslant n$. Hence MSO query evaluation is in ra-linear time for documents of D_n even though they are not covered by Corollary 8.20. □

REMARK F.4. *We observe that a limit of both the bounded event scope and the bounded correlation width conditions is that they can only show tractability for documents where we assume that each edge uses at most a constant number of events in its annotation (although of course there are tractable documents that would not satisfy this condition).*

The $\mathsf{PrXML}^{\mathsf{mux,ind}}$ *model.*

DEFINITION F.5. *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 TID and BID probabilistic relational formalisms. As we show later, this helps ensure the tractability of query evaluation.

We use Corollary 8.17 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 F.6. *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 F.7. *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 F.8. *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 F.9. *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 F.10. *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 $\mathsf{PrXML}^{\mathsf{mux,ind}}$:

PROPOSITION 8.22. *The MSO query evaluation problem on* $\mathsf{PrXML}^{\mathsf{mux,ind}}$ *has ra-linear time data complexity.*

G. PROOFS FOR SECTION 9 (SEMIRING PROVENANCE)

G.1 Material for Section 9.1 and 9.2

LEMMA 9.8. *For every (multi-)instance I and monotone multi-query q, $\widehat{q}(I)$ is finite.*

Proof. The claim is trivial if I is a multi-instance as the number of multi-subinstances of I is clearly finite. We now consider the claim where I is a set-instance with n facts.

We consider the order relation $\subseteq$ on the multi-subinstances of I, in other words, the infinite set $\mathcal{S}$ of multisets whose domain S is a subset of the facts of I, which is finite. Now, an alternative way to see the partial order $(\mathcal{S}, \subseteq)$ is as the component-wise ordering ($\mathbf{s} \leqslant \mathbf{t}$ iff $s_i \leqslant t_i$ for all $1 \leqslant i \leqslant n$) on tuples of natural numbers (where the i-th element of the tuple indicates the multiplicity of the i-th element in some arbitrary ordering on the facts of I).

Now the set of multi-subinstances of I that satisfy q is a subset of $\mathcal{S}$, and using the alternative formulation and Dickson's lemma [Dic13] we conclude that it has finitely many minimal elements, that is, $\widehat{q}(I)$ is finite. □

PROPOSITION 9.10. *For any Datalog program P, one can compute a monotone multi-query q_P such that, for every absorptive semiring K and (multi-)instance I, $W_K(q_P, I)$ is the provenance of P on I for K in the sense of [GKT07].*

Proof. We recall the definition of Datalog proof trees from the proof of Proposition 6.8, which we amend slightly to require that nodes have as many children as the number of *facts* used in their rules (intensional or extensional), and node labels can be $(F, \emptyset)$ where F is an instance fact; in other words, the use of instance facts are visible in the proof tree, and the nodes labeled with instance facts are leaves. This definition matches that of the *derivation trees* of [GKT07].

We now recall the definition of provenance from [GKT07] (Definition 5.1): For any K-instance I, and Datalog program P, the provenance of P on I is:

$$\bigoplus_{\tau \text{ proof tree of } P} \quad \bigotimes_{l \text{ leaf of } \tau} I(l)$$

where $I(l)$ is the annotation of fact l in I. This expression is not always defined, but if we assume that K is absorptive, a variant of Lemma 9.8 would justify that it is.

We now define q_P as follows. Given a multi-instance I and Datalog program P, we say that I satisfies q_P if there exists a proof tree of P on I whose multiset of leaves is $\subseteq I$ (we call this the *multi-query defined by P*). Clearly this multi-query is always monotone and its provenance according to our definition matches that of [GKT07]. □

DEFINITION G.1. *The set of* p-multi k-facts *of σ, written $\mathrm{fct}_k^p(\sigma)$, is the set $\mathrm{fct}_k(\sigma) \times \{1, \ldots, p\}$. A* p-multi tree encoding *of width k is a $\mathrm{fct}_k^p(\sigma)$-tree and* p-multi K-tree-encodings *are $(\mathrm{fct}_k^p(\sigma) \times K)$-trees.*

A tree decomposition *of a (K-)multi-instance I is that of the instance formed by the facts of* $\mathrm{dom}(I)$ *with multiplicity > 0, and a* tree encoding *of a (K-)multi-instance of width $\leqslant k$ is a $(\mathrm{fct}_k(\sigma) \times \mathbb{N})$-tree (or $(\mathrm{fct}_k(\sigma) \times \mathbb{N} \times K)$-tree). Note that this alphabet is not finite. For $p \in \mathbb{N}^*$, we accordingly define the* p-truncation *of a (K-)multi-instance I as the instance $I^{\leqslant p} \subseteq I$ with $I^{\leqslant p}(F) = \min(I(F), p)$ for every F of I. The* p-truncation *of a $(\mathrm{fct}_k(\sigma) \times \mathbb{N})$-tree E (or $(\mathrm{fct}_k(\sigma) \times \mathbb{N} \times K)$-tree E) is the $\mathrm{fct}_k^p(\sigma)$-tree (or $(\mathrm{fct}_k^p(\sigma) \times K)$-tree) $E^{\leqslant p}$ obtained from E by replacing every label (τ, i) for $i > p$ by (τ, p). We generalize $\langle E \rangle$ for $\mathrm{fct}_k^p(\sigma)$-trees and $(\mathrm{fct}_k^p(\sigma) \times K)$-tree.*

We say that a $\mathrm{fct}_k^p(\sigma)$-bDTA (for some p) tests *a multi-query q for treewidth k if for any $(\mathrm{fct}_k(\sigma) \times \mathbb{N})$-tree E, $E^{\leqslant p} \models A$ iff $\langle E \rangle \models q$. We say that q is FTAR if there exists $p \in \mathbb{N}^*$ such that, for every $k \in \mathbb{N}^*$, one can compute a bDTA A_k on $\mathrm{fct}_k^p(\sigma)$ that tests q for treewidth k.*

LEMMA G.2. *For any semiring K and monotone FTAR multi-query q tested by a $\mathrm{fct}_k^p(\sigma)$-bDTA, for any K-instance I, we have $W_K(q, I) = W_K(q, I^{\leqslant p})$.*

Proof. It suffices to show that $\widehat{q}(I) = \widehat{q}(I^{\leqslant p})$. Let $I' \in \widehat{q}(I^{\leqslant p})$. Clearly it is a sub-multi-instance of I that satisfies q, and it is minimal as any $I'' \subsetneq I'$ such that $I'' \models q$ would contradict the minimality of I' in $\widehat{q}(I)$. So it suffices to show the converse inclusion.

Letting $I' \in \widehat{q}(I)$, we show that the multiplicity of every fact in I' is at most p. Indeed, as there is an automaton A on $\mathrm{fct}_k^p(\sigma)$ that tests q, letting E' be $\mathrm{fct}_k(\sigma) \times \mathbb{N}$ be a tree encoding of I', A accepts $E'^{\leqslant p}$. As A tests q, this implies that $\left\langle E'^{\leqslant p} \right\rangle \models q$, and it is clear that $\left\langle E'^{\leqslant p} \right\rangle \subseteq I'$ and that the multiplicity of every fact in $\left\langle E'^{\leqslant p} \right\rangle$ is at most p. This proves our claim by minimality of I', so that $I' \subseteq I^{\leqslant p}$. Since $I' \models q$, and any $I'' \subsetneq I'$ such that $I'' \models q$ would contradict the minimality of I' in $\widehat{q}(I)$, $I' \in \widehat{q}(I^{\leqslant p})$. □

G.2 Proof of Theorem 9.11

Our proof is connected to that of Section 5. We use Lemmas 5.8 and 5.3, and our main problem is to adapt Lemma 5.6 to compute a provenance; there, however, a different technique is needed.

Before we embark on this, however, we need first to connect the provenance of a multi-query on an instance to a notion of provenance for automata on tree encodings, for which we need to understand in which sense automata for monotone queries can be made monotone. We fix throughout the proof a semiring K.

We work with bNTAs rather than bDTAs (recall Definition D.1), as the monotonicity property is simpler to state for them. We now define our notion of a *monotone* Γ-bNTA given an order relation on Γ:

DEFINITION G.3. *Given a partial order $<$ on Γ, we say that a Γ-bNTA $A = (Q, F, \iota, \delta)$ is* monotone *if for every $l < l'$ in Γ, we have $\iota(l) \subseteq \iota(l')$, and for every $q_1, q_2 \in Q$ we have $\delta(q_1, q_2, l) \subseteq \delta(q_1, q_2, l')$.*

This syntactic definition of monotonicity implies a semantic consequence, in the following way:

DEFINITION G.4. *Any partial order $(\Gamma, <)$ can be extended to a partial order on Γ-trees by setting $T \leqslant T'$ if T and T' have same skeleton and for every node n in T with label l corresponding to node n' in T' with label l', we have $l \leqslant l'$.*

LEMMA G.5. *Given a Γ-bNTA A which is monotone for $(\Gamma, <)$, for any Γ-trees T and T', if $T \leqslant T'$ and $T' \models A$ then $T \models A$.*

Proof. Immediate as any run of A on T is a run of A on T'. □

In our context, for any $p \in \mathbb{N}$, we define the partial order $(\mathrm{fct}_k^p(\sigma), <)$ by $(\tau, i) < (\tau', j)$ iff $\tau = \tau'$ and $i < j$. We say that E' is a *subencoding* of E if $E' \leqslant E$, and then:

LEMMA G.6. *For every $\mathrm{fct}_k^p(\sigma)$-trees E and E', if $E \leqslant E'$ then $\langle E \rangle \subseteq \langle E' \rangle$.*

Proof. We follow the decoding process and notice that, as the domains of the $\mathrm{fct}_k(\sigma)$ node labels in E and E' are the same, the same fresh elements are used throughout, so the only difference between $\langle E \rangle$ and $\langle E' \rangle$ is about the multiplicity of the created facts; and we notice that whenever $E \leqslant E'$ then every fact created in E is created in E' with a higher multiplicity. □

We can now show that monotone multi-queries can be compiled to monotone bNTAs (and the complexity for this is no higher than compiling them to bDTAs):

LEMMA G.7. *Let $k \in \mathbb{N}$, q be a monotone multi-query, and A be a $\mathrm{fct}_k^p(\sigma)$-bDTA for some $p \in \mathbb{N}$ that tests q for treewidth k. One can compute in linear time from A a $\mathrm{fct}_k^p(\sigma)$-bNTA A' that tests q for treewidth k and is monotone for $(\mathrm{fct}_k^p(\sigma), <)$.*

Proof. Fix k, q, and $A = (Q, F, \iota, \delta)$ the $\mathrm{fct}_k^p(\sigma)$-bDTA. We build the bNTA $A' = (Q, F, \iota', \delta')$ by setting, for all $(\tau, i) \in \mathrm{fct}_k^p(\sigma)$, $\iota'((\tau, i)) := \{\iota((\tau, j)) \mid 0 \leqslant j \leqslant i\}$ and for all $q_1, q_2 \in Q$, $\delta'(q_1, q_2, (\tau, i)) := \{\delta(q_1, q_2, (\tau, j)) \mid 0 \leqslant j \leqslant i\}$.

Clearly A' is monotone by construction for $(\mathrm{fct}_k^p(\sigma), <)$. Besides, for any $\mathrm{fct}_k^p(\sigma)$-tree E, if A accepts E then A' accepts E, so to prove the correctness of A' it suffices to prove the converse implication.

Let us consider such an E, and consider an accepting run ρ of A' on E. We build a new encoding E' whose skeleton is that of E and where for any leaf (resp. internal node) $n' \in E'$ with corresponding node $n \in E$, we set $\lambda(n)$ in E' to be $(\lambda(n), i)$ for some i such that $\rho(n) = \iota((\tau, i))$ (resp. $\rho(n) = \delta(\rho(Ch_{\swarrow}(n)), \rho(Ch_{\searrow}(n)), (\tau, i))$), the existence of such an i being guaranteed by the definition of ι' (resp. δ').

We now observe that, by construction, f is a run of A on E', and it is still accepting, so that E' is accepted by A. Hence, $\langle E' \rangle \models q$. But now we observe that, once again by construction, for every node n' of E' with label τ' and with corresponding node n in E with label τ, it holds that $\tau' \leqslant \tau$. Hence we have $E' \subseteq E$, and, by Lemma G.6, $\langle E' \rangle \subseteq \langle E \rangle$, and thus, by monotonicity of q, we must have $\langle E \rangle \models P$. Thus, A accepts E, proving the desired result. □

We now define a notion of provenance for a monotone bNTA A on a $(\mathrm{fct}_k^p(\sigma) \times K)$-tree E, to mimic the notion of provenance of a monotone query on an instance.

DEFINITION G.8. *Given a $\mathrm{fct}_k^p(\sigma)$-bNTA A and $\mathrm{fct}_k^p(\sigma)$-tree E, we denote by $\widehat{A}(E)$ the set of minimal $\mathrm{fct}_k^p(\sigma)$-subtrees $E' \leqslant E$ (for $\mathrm{fct}_k^p(\sigma)$ following Definition G.4) such that $E \models A$.*

Given a monotone bNTA A and a p-multi K-tree-encoding E, the provenance *of A on E is:*

$$W_K(A, E) = \bigoplus_{E' \in \widehat{A}(E)} \bigotimes_{\substack{n \in E' \\ (\tau, i, \kappa) := \lambda(n)}} \kappa^i$$

Of course, we now justify that this notion of provenance for automata relative to encodings corresponds to our notion of provenance for queries relative to instances.

LEMMA G.9. *Let $k \in \mathbb{N}^*$, let I be a K-multi-instance of width $\leqslant k$ and E be its p-multi K-tree encoding (for sufficiently large p). Then for any monotone FTAR multi-query q and monotone $\mathrm{fct}_k^p(\sigma)$-bNTA A that tests q, we have $W_K(A, E) = W_K(q, I)$.*

Proof. Let ψ be the bijection from the support of I to the nodes of E with multiplicity > 0, as obtained in Lemma 3.4. This mapping defines a bijection $\widehat{\psi}$ from the set of multi-subinstances J of I to the set of subencodings E' of E, as follows: $\widehat{\psi}(J)$ is the encoding E' where, for every $n' \in E'$ with corresponding $n \in E$, writing $(\tau, i, \kappa) := \lambda(n)$, we set $\lambda(n') := \lambda(n)$ if n is not in the image of ψ, and $\lambda(n') := (\tau, J(\psi^{-1}(n)), \kappa)$ otherwise. It is immediate that for any $I' \subseteq I$, $\widehat{\psi}(I')$ is a tree encoding of I'.

Besides, as A tests q, for any $I' \subseteq I$, $I' \models q$ iff $\widehat{\psi}(I')$ is accepted by A. So $\widehat{A}(E) = \widehat{\psi}(\widehat{q}(I))$, and it only remains to see that for $I' \in \widehat{q}(I)$, we have $\bigotimes_{F \in I'} \alpha(F)^{I'(F)} = \bigotimes_{\substack{n \in \widehat{\psi}(I') \\ (\tau, i, \kappa) := \lambda(n)}} \kappa^i$ but this is clear by construction of $\widehat{\psi}$. □

We now turn to the proper analogue of Lemma 5.6, introducing the required definitions and stating the lemma. Compare the following to Definition 5.5:

DEFINITION G.10. *Let $k, p \in \mathbb{N}^*$, E be a $\mathrm{fct}_k^p(\sigma)$-tree, and A be a $\mathrm{fct}_k^p(\sigma)$-bNTA. A K-*provenance circuit *of A on E is a tuple $(C, T, \xi, g^{\mathrm{o}})$ where C is a K-circuit, T is a tree decomposition of C with same skeleton as E, ξ maps every $b \in T$ to $\xi(b) \in C_{\mathsf{inp}}$ and the image of ξ covers C_{inp}, and g^{o} is a distinguished gate of C. For any annotation function $f : E \to K$ defining a K-encoding $f(E)$, letting ν be the corresponding valuation of C_{inp} (for every bag b of T and node n of E corresponding to $\xi(b)$, $\nu(\xi(b)) = f(n)$), we require that $\nu(C)(g^{\mathrm{o}}) = W_K(A, f(E))$.*

The absorptivity of K is used as follows:

LEMMA G.11. *Letting $k, p \in \mathbb{N}$, for any monotone $\mathrm{fct}_k^p(\sigma)$-bNTA A and $\mathrm{fct}_k^p(\sigma)$-tree E, we have:*

$$W_K(A,E) = \bigoplus_{\substack{E' \subseteq E \\ E \models A}} \bigotimes_{\substack{n \in E' \\ (\tau,i,\kappa) := \lambda(n)}} \kappa^i$$

In other words, we can sum on all subencodings, not just minimal ones.

Proof. By definition of $\widehat{A}(E)$, for any $E' \subseteq E$ such that A accepts E', there exists $E'' \in \widehat{A}(E)$ such that $E'' \subseteq E'$. Now as $E'' \in \widehat{A}(E)$, A also accepts E'', so the sum in the right-hand size of the equality contains a term $T_{E''}$ for E'' as well as the term $T_{E'}$ for E'. We now argue that $T_{E''} \oplus T_{E'} = T_{E''}$ by absorptivity, so the result follows by induction on the number of non-minimal terms in the $\bigoplus$ (intuitively, they can be simplified one by one). But now we only have to observe that $T_{E'}$ and $T_{E''}$ are products and the set of factors of $T_{E'}$ is a superset of that of $T_{E''}$ because $E'' \subseteq E'$. This concludes the proof. $\square$

This is the analogue of Lemma 5.6:

LEMMA G.12. *Let $k, p \in \mathbb{N}^*$ and let A be a monotone $\mathrm{fct}_k^p(\sigma)$-bNTA with state space Q. If K is absorptive, for any p-multi tree encoding E of width k, one can compute in $O(|E| \cdot |A|)$ a K-provenance circuit C of A on E with a tree decomposition T of width depending only on k, p and A.*

Proof. Fix $k, p \in \mathbb{N}$, the automaton $A = (Q, F, \iota, \delta)$, and E. We construct the circuit C, distinguished gate g^{o} and function ξ. For the purposes of the construction, we temporarily relax the restriction on circuits that the in-degree of gates matches that of the function that they compute, in the case of gate functions which are associative (intuitively, those that represents the laws of the semiring). We later explain how to leverage associativity to rewrite the circuit to an equivalent circuit where such gates have in-degree two (intuitively, by rewriting each such gate to a chain of gates).

For every node n in E, letting $(\tau_n, m_n) := \lambda(n)$, we create $|Q|$ gates g_n^q, one input gate g_n^{i} in C, and at most $p \times |Q|$ gates $g_n^{q,i}$ for $0 \leqslant i \leqslant m_n$ which are $\otimes$-gates with input g_n^{i} repeated i times.

If n is a leaf node, for each $q \in Q$, we set g_n^q to be an $\oplus$-gate of all the $g_n^{q,i}$ such that $q \in \iota((\tau_n, i))$, for $0 \leqslant i \leqslant m_n$.

If n is an internal node, for each $q \in Q$:

- for every states $q_\swarrow, q_\searrow, q \in Q$, and $0 \leqslant i \leqslant m_n$, create an additional gate $g_n^{q_\swarrow, q_\searrow, q, i}$ which is an $\otimes$-gate of $g_{Ch_\swarrow(n)}^{q_\swarrow}$, $g_{Ch_\searrow(n)}^{q_\searrow}$, and $g^{q,i}$;
- for every states $q_\swarrow, q_\searrow, q \in Q$, create a gate $g_n^{q_\swarrow, q_\searrow, q}$ which is the $\oplus$ of all the $g_n^{q_\swarrow, q_\searrow, q, i}$ such that $q \in \delta(q_\swarrow, q_\searrow, (\tau_n, i))$ for $0 \leqslant i \leqslant m_n$;
- set g_n^q to be the $\oplus$ of all the $g_n^{q_\swarrow, q_\searrow, q}$ for $q_\swarrow, q_\searrow \in Q$.

We add a distinguished gate g^{o} which is the $\bigoplus$ of all the $g_{n_{\mathrm{r}}}^q$, for $q \in F$ and where n_{r} is the root node of E.

To prove the correctness of the construction, we define an additional notion: we define the *provenance of A at node n for state q*, for $n \in E$ and $q \in Q$, as the provenance of A^q for $E_{|n}$, where A^q is the automaton obtained from A by setting $F = \{q\}$ and $E_{|n}$ is the subtree of E rooted at n.

We now prove for any annotation function f of E, with ν the corresponding valuation of C_{inp}, the following invariant by a bottom-up induction on E: for every node $n \in E$, and state $q \in Q$, $\nu(C)(g_n^q)$ is the provenance of A at node n for state q: formally, $\nu(C)(g_n^q) = W_K(A^q, f(E)_{|n})$.

For the base case, let n be a leaf node of E. By Lemma G.11, the provenance of A on $f(E)_{|n}$ at node n for state q is:

$$W_K(A^q, f(E)_{|n}) = \bigoplus_{\substack{E' \subseteq f(E)_{|n} \\ E' \models A^q}} \bigotimes_{\substack{n' \in E' \\ (\tau,i,\kappa) := \lambda(n')}} \kappa^i$$

Now as $f(E)_{|n}$ contains only one node, letting $(\tau_n, m_n, \kappa_n) := \lambda(n')$ for n' the node corresponding to n in $f(E)$, this rewrites to:

$$W_K(A^q, f(E)_{|n}) = \bigoplus_{\substack{0 \leqslant i \leqslant m_n \\ q \in \iota((\tau_n, i))}} \kappa_n^i$$

Now, let $q \in Q$, we have $\nu(C)(g_n^q) = \bigoplus_{\substack{0 \leqslant i \leqslant m_n \\ q \in \iota((\tau,i))}} \nu(C)(g_n^{\mathrm{i}})$, and as $\kappa_n = \nu(C)(g_n^{\mathrm{i}})$, we have shown that $\nu(C)(g_n^q) = W_K(A^q, f(E)_{|n})$.

For the inductive case, let n be an internal node of E. By Lemma G.11, we have:

$$W_K(A^q, f(E)_{|n}) = \bigoplus_{\substack{E' \subseteq f(E)_{|n} \\ E' \models A^q}} \bigotimes_{\substack{n \in E' \\ (\tau', i, \kappa) := \lambda(n)}} \kappa^i$$

Now, let $(\tau_n, m_n) := \lambda(n)$ and partition the terms of the sum depending on the multiplicity of the node corresponding to n in E', which is $0 \leqslant i_0 \leqslant m_n$. For every such i_0, we know that we can factor out $\kappa_n^{i_0}$ from the $\bigotimes$, where $\kappa_n := \nu(C)(g_n^{\mathrm{i}})$. Further, we can clearly choose $E' \subseteq f(E)_{|n}$, under the condition that the node corresponding to n in E' has multiplicity i_0, by choosing two subencodings $E'_{\swarrow}$ and $E'_{\searrow}$ of $f(E)_{|Ch_{\swarrow}(n)}$ and $f(E)_{|Ch_{\searrow}(n)}$; to respect the condition that A accepts E', it is necessary and sufficient that $E'_{\swarrow}$ and $E'_{\searrow}$ are accepted by $A^{q_{\swarrow}}$ and $A^{q_{\searrow}}$ for some $q_{\swarrow}, q_{\searrow} \in Q$ such that $q \in \delta(q_{\swarrow}, q_{\searrow}, (\tau_n, i_0))$; from which we can split the $\bigotimes$ in two, and rewrite:

$$W_K(A^q, f(E)_{|n}) = \bigoplus_{\substack{0 \leqslant i_0 \leqslant m_n \\ q_{\swarrow}, q_{\searrow} \in Q \\ E'_{\swarrow} \subseteq f(E)_{|Ch_{\swarrow}(n)} \\ E'_{\searrow} \subseteq f(E)_{|Ch_{\searrow}(n)} \\ q \in \delta(q_{\swarrow}, q_{\searrow}, (\tau_n, i_0)) \\ E'_{\swarrow} \models A^{q_{\swarrow}} \\ E'_{\searrow} \models A^{q_{\searrow}}}} \kappa_n^{i_0} \otimes \left(\bigotimes_{\substack{n \in E'_{\swarrow} \\ (\tau', i, \kappa) := \lambda(n)}} \kappa^i \right) \otimes \left(\bigotimes_{\substack{n \in E'_{\searrow} \\ (\tau', i, \kappa) := \lambda(n)}} \kappa^i \right)$$

Of course the sum on $q_{\swarrow}, q_{\searrow} \in Q$ is not a partition (some terms of the corresponding sum may be counted multiple times), but this is not a problem as $\oplus$ is involutive because K is absorptive. Now we use distributivity to regroup the sums and apply Lemma G.11 (backwards) to rewrite this as:

$$W_K(A^q, f(E)_{|n}) = \bigoplus_{\substack{0 \leqslant i_0 \leqslant m_n \\ q_{\swarrow}, q_{\searrow} \in Q \\ q \in \delta(q_{\swarrow}, q_{\searrow}, (\tau_n, i_0))}} \kappa_n^{i_0} \otimes W_K(A^{q_{\swarrow}}, f(E)_{|n_{\swarrow}}) \otimes W_K(A^{q_{\searrow}}, f(E)_{|n_{\searrow}})$$

Now, fix $q \in Q$, and we show that this expression matches $\nu(C)(g_n^q)$ for $q \in Q$. We have:

$$\nu(C)(g_n^q) = \bigoplus_{\substack{q_{\swarrow}, q_{\searrow} \in Q \\ 0 \leqslant i_0 \leqslant m_n \\ q \in \delta(q_{\swarrow}, q_{\searrow}, (\tau_n, i))}} (\nu(C)(g_n^{\mathrm{i}}))^{i_0} \otimes \nu(C)(g_{Ch_{\swarrow}(n)}^{q_{\swarrow}}) \otimes \nu(C)(g_{Ch_{\searrow}(n)}^{q_{\searrow}})$$

Hence, as $\nu(C)(g_n^{\mathrm{i}}) = \kappa_n$, and using the induction hypothesis on $\nu(C)(g_{Ch_{\swarrow}(n)}^{q_{\swarrow}})$ and $\nu(C)(g_{Ch_{\searrow}(n)}^{q_{\searrow}})$, we have shown that $\nu(C)(g_n^q) = W_K(A^q, f(E)_{|n})$.

By applying the invariant to the $g_{n_r}^q$ where $n_r \in E$ is the root of E, we conclude the correctness proof by noticing that we can use Lemma G.11 and rewrite the resulting expression (using the involutivity of $\oplus$) as:

$$W_K(A, f(E)) = \bigoplus_{q_{\mathrm{f}} \in F} W_K(A^{q_{\mathrm{f}}}, f(E))$$

so that clearly $\nu(C)(g^{\mathrm{o}}) = W_K(A, f(E))$, which concludes the correctness proof.

We now take care of the following technical issue: we have not created a circuit where $\oplus$- and $\otimes$-gates necessarily have an in-degree of two. We show that we can rewrite in linear time such an arity-n circuit to an arity-two circuit that is equivalent.

We say that a circuit is *arity-two* if its $\otimes$- and $\oplus$-gates have in-degree exactly two. It is clear that for any semiring K, from our K-circuit $C = (G, W)$ with gates of in-degree higher than two, one can compute in linear time an arity-two K-circuit $C' = (G', W')$, using associativity and the axioms about $\otimes$ and $\oplus$, with only a constant-factor blowup. First, replace $\oplus$- and $\otimes$-gates with arity zero by a 0_K-gate, and a 1_K-gate, respectively. Next, add to $\otimes$- and $\oplus$-gates with in-degree 1 a new gate as input that is respectively a 1_K-gate or a 0_K-gate. Next, use associativity of the $\otimes$ and $\oplus$ Boolean operations: if g is an $\otimes$- or $\oplus$-gate of C with in-degree > 2, and $(g_i, g) \in W$ for $1 \leqslant i \leqslant n$, create new gates $g'_1, \ldots, g'_{n-2}$ in G' of the same type as g, remove (g_i, g) from W for $1 \leqslant i < n$, add (g'_i, g'_{i+1}) to W for $1 \leqslant i \leqslant n-3$, add (g'_{n-2}, g) to W, and add (g_1, g'_1), and (g_{i+1}, g'_i) to W for all $1 \leqslant i \leqslant n-2$, yielding W'. It is clear that C' is equivalent to C and that the process can be performed in linear time.

To justify the overall complexity claim, it suffices to check that at each node n of E, the number of operations performed, up to logarithmic factors, is in $O(|A|)$. Last, we justify the bound on treewidth by creating the tree decomposition in the same way as in the proof of Lemma 5.6, where the intermediate gates used to encode gates of the original circuit with more than two inputs are put in the same bag as the gates for which they were created (this resulting only in a blowup of the size of each bag by a constant multiplicative factor). We set $\xi(b)$ to be g_n^{i} for the node n of E corresponding to b, for every b in T. $\square$

We are now ready to conclude the proof of Theorem 9.11. Fix the monotone FTAR multi-query q, and $k \in \mathbb{N}^*$. Having computed (in constant time in the query) a bDTA that tests the FTAR query q, compile in constant time the query q to a monotone bNTA A on $\mathrm{fct}_k^p(\sigma)$, for some $p \in \mathbb{N}^*$, using Lemma G.7.

Given a K-cc-instance $J = (I, C, \varphi)$ of treewidth $\leqslant k$, first compute a tree decomposition T of J (in linear time in $|J|$), and apply Lemma 5.3 to J and T to obtain a cc-encoding $E' = (E, C, T)$ of width k of J. Let E'' be the $\mathrm{fct}_k^p(\sigma)$-tree obtained from E by giving each node multiplicity p. Apply Lemma G.12 to A to obtain in time linear in $|J|$ a provenance circuit C' of A on E'' with a tree decomposition T' of width depending only on q and k; rename C''_{inp} as in the proof of Theorem 4.13, yielding the circuit C'', so that C and C'' are stitchable. Now, build $C''' = C \circ C''$ with tree decomposition $T''' = T + T'$ of width depending only on q and k, with $C'''_{\mathsf{inp}} = C_{\mathsf{inp}}$ and whose distinguished gate is the gate g^{o} of C''. The circuit C''' has treewidth depending only on q and k and the overall process has linear complexity in J.

The only thing left to show is that C''' satisfies the desired property. Fix a valuation ν of J_{inp}, and consider $\nu(C''')(g^{\mathrm{o}})$. By Lemma C.3, fixing ν'' to be the restriction of $\nu(C)$ to C''_{inp}, we have $\nu(C''')(g^{\mathrm{o}}) = \nu''(C'')(g^{\mathrm{o}})$. Now, by Lemma G.12, this is $W_K(A, f(E''))$, where f is the annotation function corresponding to ν'', which by Lemma G.9 is $W_K(q, I')$ where I' is the K-multi-instance obtained from $\nu(J)$ by setting the multiplicity of every fact to p; by Lemma G.2, this is equal to $W_K(q, \nu(J))$, showing the desired result.

G.3 Material for consequences of Theorem 9.11

PROPOSITION 9.12. *Let q be a UCQ. The corresponding multi-query defined via Proposition 9.10 through a direct encoding of q to Datalog is FTAR.*

Proof. As a first step, we rewrite query q so that it contains no equality atoms.

We first need to introduce terminology that simplifies reasoning about UCQs seen as multi-queries.

Given a UCQ $q = \bigvee_i q_i$, the multi-query q' *associated* to q is defined as follows: for any multi-instance I, we say that $I \models q$ if there is a CQ q_i such that, seeing the atoms of q_i as a multiset (so not identifying, e.g., $R(x)R(x)$ and $R(x)$), there is a mapping m from the variables of q_i to $\mathrm{dom}(I)$ such that for every fact $R(\mathbf{a})$ of I occurring with multiplicity p, considering all the atoms $R(\mathbf{x}^j)$ of q_j with multiplicity p_j such that $m(x_i^j) = a_i$ for every i, we have $p \geqslant \sum_j p_j$. We call such a mapping a *match* of q.

This intuitive notion of a "match" of a UCQ with multiplicities is the same as the notion of match of the corresponding Datalog program. Indeed: for any UCQ $q = \bigvee_i \exists \mathbf{x} q_i(\mathbf{x})$, letting q' be the associated multi-query, letting P be the straightforward encoding of q to Datalog obtained by creating one rule $\mathrm{Goal}() \leftarrow q_i(\mathbf{x})$ for every CQ, and q_P be the multi-query associated to P, we have that q_P and q' are the same bag query (i.e., for any multi-instance I, $I \models q_P$ iff $I \models q'$).

We prove this by noticing that for a CQ q it is clear that the matches of q in an instance I are in bijective correspondence with the sets of leaves of derivation tree of the straightforward Datalog encoding of q. Now the result immediately generalizes to UCQs, as the derivation trees or matches of a UCQ are the union of derivation trees of matches of the component CQs.

We introduce some more 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 $\mathrm{CQ}^{\neq}$. We immediately generalize the above definition of matches to $\mathrm{UCQ}^{\neq}$ by adding the condition that the match m from $\mathrm{Vars}(q_i)$ to $\mathrm{dom}(I)$ respects that $m(x) \neq m(y)$ if $x \neq y$ occurs in q_i (the inequality atoms are otherwise ignored; in particular, their multiplicity is irrelevant).

We first note that, writing the UCQ q as the disjunction of CQs q_i, if we can show that the resulting multi-query for every q_i is FTAR, then the result clearly follows from q by a product construction on the automata that test q_i (to build a $\mathrm{fct}_k^p(\sigma)$-bDTA, where $p = \max_i p_i$ and each q_i is tested by a $\mathrm{fct}_k^{p_i}(\sigma)$-bDTA). So it suffices to show the result for CQs.

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 multi-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 the argument that we can perform a product construction on the automata, 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 these forced queries are FTAR. To see this, considering such a query q on signature σ, letting p be the sum of the multiplicities of all atoms in 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 $\bigvee_{m \geqslant j \geqslant p} R^j(\mathbf{a})$ (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 multi-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 is FTAR (as it is expressible in GSO, so we can apply Theorem 6.5); fix $k \in \mathbb{N}^*$ and let $A_{q'} = (Q, F, \iota, \delta)$ be a monotone $\mathrm{fct}_k(\sigma^p)$-bNTA that tests q' for width k (this is possible by Lemma G.7 as q', as a UCQ$^{\neq}$, is monotone). We build the following bNTA $A_q = (Q, F, \iota', \delta')$ on $\mathrm{fct}_k^p(\sigma)$: for every $((d, f), i) \in \mathrm{fct}_k^p(\sigma)$, 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 $\delta'(q_\swarrow, q_\searrow, ((d, f), i))$ to be $\delta(q_\swarrow, q_\searrow, (d, f'))$ for every $q_\swarrow, q_\searrow \in Q$.

We now claim that A_q tests q on $\mathrm{fct}_k^p(\sigma)$. To see why, it suffices to observe that for any $\mathrm{fct}_p^k(\sigma)$-tree E, letting E' be the $\mathrm{fct}^k(\sigma^p)$-tree obtained in the straightforward manner, then A_q accepts E iff $A_{q'}$ accepts E', which is immediate by construction. Now indeed, as we know that $A_{q'}$ accepts E' iff $\langle E' \rangle \models q'$ (as $A_{q'}$ tests q'), and (as immediately $\langle E' \rangle$ is the σ^p-instance corresponding to $\langle E \rangle$ as I' corresponds to I above) that $\langle E' \rangle \models q'$ iff $\langle E \rangle \models q$, so that we have the desired equivalence.

The only thing left is to observe that A_q not only tests q on instances where each fact has multiplicity $\leqslant p$. But a straightforward argument shows that because 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. $\square$

Of course, the construction of Proposition 9.12 incurs a blowup, but it does not depend on the instance, so does not influence data complexity. We conjecture that the same technique (with the same limitation) applies more generally to non-recursive Datalog programs.

DEFINITION G.13. *For any set of variables X, the* PosBool$[X]$ *semiring is the semiring* $(\mathrm{PosBool}[X], \vee, \wedge, \mathfrak{f}, \mathfrak{t})$ *whose elements are the monotone Boolean functions on variables X. We write the elements of* PosBool$[X]$ *as propositional formulae of the variables of X on the de Morgan basis, but it is important to understand that equivalent ways to express the same function (e.g., $x \vee x$ and x) denote the same object.*

DEFINITION G.14 [DMRT14]. *The* Sorp$[X]$ *semiring is the quotient of* $\mathbb{N}[X]$ *by the smallest equivalence relation $\sim$ which identifies polynomials according to absorption.*

DEFINITION G.15. *A* semiring homomorphism *is a mapping $h : K_1 \to K_2$ such that $h(0_{K_1}) = 0_{K_2}$, $h(1_{K_1}) = 1_{K_2}$, and for all $a, b \in K_1$ we have $h(a \oplus_{K_1} b) = h(a) \oplus_{K_2} h(b)$ and $h(a \otimes_{K_1} b) = h(a) \otimes_{K_2} h(b)$.*

The following is our formal claim, in the context of set-instances. A similar result holds for cc-instances (where inputs rather than facts are mapped to canonical annotations):

PROPOSITION G.16. *For any multi-instance I, let $\mathcal{A}$ be a finite set and $\overline{\alpha}$ be a bijection from the facts of I to $\mathcal{A}$, and let $\bar{I}$ be the* Sorp$[\mathcal{A}]$*-multi-instance obtained by annotating each fact F of I by $\overline{\alpha}(F)$. For any monotone query q, let $W(\bar{I}, q) := W_{\mathrm{Sorp}[\mathcal{A}]}(\bar{I}, q)$ be the provenance of q on $\bar{I}$. For any absorptive semiring K and valuation function ν to annotate the facts of I, there exists a unique semiring homomorphism h_ν :* Sorp$[\mathcal{A}] \to K$ *which extends ν (that is, $h_\nu(\overline{\alpha}(F)) = \nu(F)$ for every fact of I), and we have $W_K(\nu(I), q) = h_\nu(W(I', q))$.*

Proof. By Proposition 4.2 of [GKT07], there exists a unique homomorphism of semirings $h'_\nu : \mathbb{N}[\mathcal{A}] \to K$ satisfying the desired condition. We show that the same holds for Sorp$[\mathcal{A}]$, assuming that K is associative. Unicity is immediate by observing that the definition of h_ν on $\mathcal{A}$ can be extended in at most one way to a homomorphism on Sorp$[\mathcal{A}]$, as any element of Sorp$[\mathcal{A}]$ can be written in a minimal way as a (possibly empty) sum of (possibly empty) products of elements of $\mathcal{A}$. For existence, observe that for any sum $h_\nu(a \oplus b)$ for $a, b \in$ Sorp$[\mathcal{A}]$, any simplifications in $a \oplus b$ due to absorptivity can be mimicked in $h_\nu(a) \oplus h_\nu(b)$ as K is absorptive, and likewise for $\otimes$. This proves our claim about Sorp$[\mathcal{A}]$.

Now to prove that the desired property holds, write:

$$\begin{aligned} W_K(\nu(I), q) &= \bigoplus_{J \in \widehat{q}(\nu(I))} \bigotimes_{F \in J} \nu(F)^{J(F)} \\ &= \bigoplus_{J \in \widehat{q}(\nu(I))} \bigotimes_{F \in J} (h_\nu(\overline{\alpha}(F)))^{J(F)} \\ &= h_\nu \left(\bigoplus_{J \in \widehat{q}(\nu(I))} \bigotimes_{F \in J} (\overline{\alpha}(F))^{J(F)} \right) \\ &= h_\nu(W(I', q)) \end{aligned}$$

$\square$

DEFINITION G.17. *The* security semiring *[FGT08, ADT11b] $\mathbb{S}$ 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 $(\{0, 1, C, S, T\}, \max, \min, 0, 1)$. The provenance of a query for $\mathbb{S}$ denotes the minimal level of security clearance required to see that it holds.*

The fuzzy semiring *(see e.g. [ADT11a]) 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 *(see e.g. [DMRT14]) is $(\mathbb{N} \sqcup \{\infty\}, \min, +, \infty, 0)$. The provenance of a query for this semiring is the minimal cost of the tuples of a multi-subinstance such that the query holds.*

H. REFERENCES FOR THE APPENDIX

[ADT11a] Y. Amsterdamer, D. Deutch, and V. Tannen. On the limitations of provenance for queries with difference. In *TaPP*, 2011.

[ADT11b] Y. Amsterdamer, D. Deutch, and V. Tannen. Provenance for aggregate queries. In *PODS*, 2011.

[BKR14] P. Bourhis, M. Krötzsch, and S. Rudolph. Query containment for highly expressive datalog fragments. *CoRR*, abs/1406.7801, 2014.

[BLRS14] P. Beame, J. Li, S. Roy, and D. Suciu. Counting of query expressions: Limitations of propositional methods. In *ICDT*, 2014.

[BtCO12] V. Bárány, B. ten Cate, and M. Otto. Queries with guarded negation. *PVLDB*, 5(11), 2012.

[BtCS11] V. Bárány, B. ten Cate, and L. Segoufin. Guarded negation. In *ICALP*, 2011.

[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.

[CGKV88] S. S. Cosmadakis, H. Gaifman, P. C. Kanellakis, and M. Y. Vardi. Decidable optimization problems for database logic programs. In *STOC*, 1988.

[CKS09] S. Cohen, B. Kimelfeld, and Y. Sagiv. Running tree automata on probabilistic XML. In *PODS*, 2009.

[CV92] S. Chaudhuri and M. Y. Vardi. On the equivalence of recursive and nonrecursive datalog programs. In *PODS*, 1992.

[Dic13] L. E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *American J. Math.*, 1913.

[DMRT14] D. Deutch, T. Milo, S. Roy, and V. Tannen. Circuits for Datalog provenance. In *ICDT*, 2014.

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

[FGT08] J. N. Foster, T. J. Green, and V. Tannen. Annotated XML: queries and provenance. In *PODS*, 2008.

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

[GHO02] E. Grädel, C. Hirsch, and M. Otto. Back and forth between guarded and modal logics. *TOCL*, 3(3), 2002.

[GKT07] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, 2007.

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

[JOS10] A. K. Jha, D. Olteanu, and D. Suciu. Bridging the gap between intensional and extensional query evaluation in probabilistic databases. In *EDBT*, 2010.

[LS88] 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.

[TW68] 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.