Combined Tractability of Query Evaluation via Tree Automata and Cycluits (Extended Version)

Antoine Amarilli[1], Pierre Bourhis[2], Mikaël Monet[1,4], and Pierre Senellart[3,4]

1 **LTCI, Télécom ParisTech, Université Paris-Saclay; Paris, France**
2 **CRIStAL, CNRS & Université Lille 1; Lille, France**
3 **DI, École normale supérieure, PSL Research University; Paris, France**
4 **Inria Paris; Paris, France**

Abstract

We investigate parameterizations of both database instances and queries that make query evaluation fixed-parameter tractable in combined complexity. We introduce a new Datalog fragment with stratified negation, intensional-clique-guarded Datalog (ICG-Datalog), with linear-time evaluation on structures of bounded treewidth for programs of bounded rule size. Such programs capture in particular conjunctive queries with simplicial decompositions of bounded width, guarded negation fragment queries of bounded CQ-rank, or two-way regular path queries. Our result is shown by compiling to alternating two-way automata, whose semantics is defined via cyclic provenance circuits (cycluits) that can be tractably evaluated. Last, we prove that probabilistic query evaluation remains intractable in combined complexity under this parameterization.

1 Introduction

Arguably the most fundamental task performed by database systems is *query evaluation*, namely, computing the results of a query over a database instance. Unfortunately, this task is well-known to be intractable in *combined complexity* [54] even for simple query languages.

To address this issue, two main directions have been investigated. The first is to restrict the class of *queries* to ensure tractability, for instance, to α-acyclic conjunctive queries [56], this being motivated by the idea that many real-world queries are simple and usually small. The second approach restricts the structure of database instances, e.g., requiring them to have bounded *treewidth* [51] (we call them *treelike*). This has been notably studied by Courcelle [23], to show the tractability of monadic-second order logic on treelike instances, but in *data complexity* (i.e., for fixed queries); the combined complexity is generally nonelementary [48].

This leaves open the main question studied in this paper: *Which queries can be efficiently evaluated, in combined complexity, on treelike databases?* This question has been addressed by Gottlob, Pichler, and Fei [35] by introducing *quasi-guarded Datalog*; however, an unusual feature of this language is that programs must explicitly refer to the tree decomposition of the instance. Instead, we try to follow Courcelle's approach and investigate which queries can be efficiently *compiled to automata*. Specifically, rather than restricting to a fixed class of "efficient" queries, we study *parameterized* query classes, i.e., we define an efficient class of queries for each value of the parameter. We further make the standard assumption that the signature is fixed; in particular, its arity is constant. This allows us to aim for low combined complexity for query evaluation, namely, fixed-parameter tractability with linear time complexity in the product of the input query and instance, called *FPT-linear* complexity.

Surprisingly, we are not aware of further existing work on tractable combined query evaluation for parameterized instances and queries, except from an unexpected angle: the

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

compilation of restricted query fragments to tree automata on treelike instances was used in the context of *guarded logics* and other fragments, to decide *satisfiability* [12] and *containment* [10]. To do this, one usually establishes a *treelike model property* to restrict the search to models of low treewidth (but dependent on the formula), and then compiles the formula to an automaton, so that the problems reduce to emptiness testing: expressive automata formalisms, such as *alternating two-way automata*, are typically used. One contribution of our work is to notice this connection, and show how query evaluation on treelike instances can benefit from these ideas: for instance, as we show, some queries can only be compiled efficiently to such concise automata, and not to the more common bottom-up tree automata.

From there, the first main contribution of this paper is to define the language of *intensional-clique-guarded Datalog* (ICG-Datalog), and show an efficient FPT-linear compilation procedure for this language, parameterized by the body size of rules: this implies FPT-linear combined complexity on treelike instances. While we present it as a Datalog fragment, our language shares some similarities with guarded logics; yet, its design incorporates several features (fixpoints, clique-guards, guarded negation, guarding positive subformulae) that are not usually found together in guarded fragments, but are important for query evaluation. We show how the tractability of this language captures the tractability of such query classes as two-way regular path queries [9] and α-acyclic conjunctive queries.

Already for conjunctive queries, we show that the treewidth of queries is not the right parameter to ensure efficient compilability. In fact, a second contribution of our work is a lower bound: we show that bounded treewidth queries cannot be efficiently compiled to automata at all, so we cannot hope to show combined tractability for them via automata methods. By contrast, ICG-Datalog implies the combined tractability of bounded-treewidth queries with an additional requirement (interfaces between bags must be clique-guarded), which is the notion of *simplicial decompositions* previously studied by Tarjan [52]. To our knowledge, our paper is the first to introduce this query class and to show its tractability on treelike instances. ICG-Datalog can be understood as an extension of this fragment to disjunction, clique-guarded negation, and inflationary fixpoints, that preserves tractability.

To derive our main FPT-linear combined complexity result, we define an operational semantics for our tree automata by introducing a notion of *cyclic provenance circuits*, that we call *cycluits*. These cycluits, the third contribution of our paper, are well-suited as a provenance representation for alternating two-way automata encoding ICG-Datalog programs, as they naturally deal with both recursion and two-way traversal of a treelike instance, which is less straightforward with provenance formulae [36] or circuits [25]. While we believe that this natural generalization of Boolean circuits may be of independent interest, it does not seem to have been studied in detail, except in the context of integrated circuit design [44, 50], where the semantics often features feedback loops that involve negation; we prohibit these by focusing on *stratified* circuits, which we show can be evaluated in linear time. We show that the provenance of alternating two-way automata can be represented as a stratified cycluit in FPT-linear time, generalizing results on bottom-up automata and circuits from [5].

Since cycluits directly give us a provenance representation of the query, we then investigate *probabilistic query evaluation*, which we showed in [5] to be linear-time in data complexity through the use of provenance circuits. We show how to remove cycles, so as to apply message-passing methods [41], yielding a 2EXPTIME upper bound for the combined complexity of probabilistic query evaluation. While we do not obtain tractable probabilistic query evaluation in combined complexity, we give lower bounds showing that this is unlikely.

Outline. We give preliminaries in Section 2, and then position our approach relative to existing work in Section 3. We then present our tractable fragment, first for bounded-

simplicial-width conjunctive queries in Section 4, then for our ICG-Datalog language in Section 5. We then define our automata and compile ICG-Datalog to them in Section 6, before introducing cycluits and showing our provenance computation result in Section 7. We last study the conversion of cycluits to circuits, and probability evaluation, in Section 8. Full proofs are provided in appendix.

2 Preliminaries

A *relational signature* σ is a finite set of relation names written R, S, T, ..., each with its associated *arity* $\mathrm{arity}(R) \in \mathbb{N}$. Throughout this work, *we always assume the signature σ to be fixed*: hence, its *arity* $\mathrm{arity}(\sigma)$ (the maximal arity of relations in σ) is constant, and we further assume it is > 0. A *(σ-)instance* I is a finite set of *ground facts* on σ, i.e., $R(a_1, \ldots, a_{\mathrm{arity}(R)})$ with $R \in \sigma$. The *active domain* $\mathrm{dom}(I)$ consists of the elements occurring in I.

We study query evaluation for several *query languages* that are subsets of first-order (FO) logic (e.g., conjunctive queries) or of second-order (SO) logic (e.g., Datalog). Unless otherwise stated, we only consider queries that are *constant-free*, and *Boolean*, so that an instance I either *satisfies* a query q ($I \models q$), or *violates* it ($I \not\models q$), with the standard semantics [1].

We study the *query evaluation problem* (or *model checking*) for a query class $\mathcal{Q}$ and instance class $\mathcal{I}$: given an instance $I \in \mathcal{I}$ and query $Q \in \mathcal{Q}$, check if $I \models Q$. Its *combined complexity* for $\mathcal{I}$ and $\mathcal{Q}$ is a function of I and Q, whereas *data complexity* assumes Q to be fixed. We also study cases where $\mathcal{I}$ and $\mathcal{Q}$ are *parameterized*: given infinite sequences $\mathcal{I}_1, \mathcal{I}_2, \ldots$ and $\mathcal{Q}_1, \mathcal{Q}_2, \ldots$, the *query evaluation problem parameterized by k_I, k_Q* applies to $\mathcal{I}_{k_\mathrm{I}}$ and $\mathcal{Q}_{k_\mathrm{Q}}$. The parameterized problem is *fixed-parameter tractable* (FPT), for $(\mathcal{I}_n)$ and $(\mathcal{Q}_n)$, if there is a constant $c \in \mathbb{N}$ and computable function f such that the problem can be solved with combined complexity $O\left(f(k_\mathrm{I}, k_\mathrm{Q}) \cdot (|I| \cdot |Q|)^c\right)$. For $c = 1$, we call it *FPT-linear* (in $|I| \cdot |Q|$). Observe that calling the problem FPT is more informative than saying that it is in PTIME for fixed k_I and k_Q, as we are further imposing that the polynomial degree c does not depend on k_I and k_Q: this follows the usual distinction in parameterized complexity between FPT and classes such as XP [29].

Query languages. We first study fragments of FO, in particular, *conjunctive queries* (CQ), i.e., existentially quantified conjunctions of atoms. The *canonical model* of a CQ Q is the instance built from Q by seeing variables as elements and atoms as facts. The *primal graph* of Q has its variables as vertices, and connects all variable pairs that co-occur in some atom.

Second, we study *Datalog with stratified negation*. We summarize the definitions here, see [1] for details. A *Datalog program* P (without negation) over σ (called the *extensional signature*) consists of an *intensional signature* σ_int disjoint from σ (with the arity of σ_int being possibly greater than that of σ), a 0-ary *goal predicate* Goal in σ_int, and a set of *rules*: those are of the form $R(\mathbf{x}) \leftarrow \psi(\mathbf{x}, \mathbf{y})$, where the *head* $R(\mathbf{x})$ is an atom with $R \in \sigma_\mathrm{int}$, and the *body* ψ is a CQ over $\sigma_\mathrm{int} \sqcup \sigma$ where each variable of $\mathbf{x}$ must occur. The *semantics* $P(I)$ of P over an input σ-instance I is defined by a least fixpoint of the interpretation of σ_int: we start with $P(I) := I$, and for any rule $R(\mathbf{x}) \leftarrow \psi(\mathbf{x}, \mathbf{y})$ and tuple $\mathbf{a}$ of $\mathrm{dom}(I)$, when $P(I) \models \exists \mathbf{y}\, \psi(\mathbf{a}, \mathbf{y})$, then we *derive* the fact $R(\mathbf{a})$ and add it to $P(I)$, where we can then use it to derive more facts. We have $I \models P$ iff we derive the fact Goal(). The *arity* of P is $\max(\mathrm{arity}(\sigma), \mathrm{arity}(\sigma_\mathrm{int}))$. P is *monadic* if σ_int has arity 1.

Datalog with stratified negation [1] allows negated *intensional* atoms in bodies, but requires P to have a *stratification*, i.e., an ordered partition $P_1 \sqcup \cdots \sqcup P_n$ of the rules where:

(i) Each $R \in \sigma_\mathrm{int}$ has a *stratum* $\zeta(R) \in \{1, \ldots, n\}$ such that all rules with R in the head are in $P_{\zeta(R)}$;

(ii) For any $1 \leqslant i \leqslant n$ and σ_{int}-atom $R(\mathbf{z})$ in a body of a rule of P_i, we have $\zeta(R) \leqslant i$;
(iii) For any $1 \leqslant i \leqslant n$ and negated σ_{int}-atom $R(\mathbf{z})$ in a body of P_i, we have $\zeta(R) < i$.
The stratification ensures that we can define the semantics of a stratified Datalog program by computing its interpretation for strata $P_1, \ldots, P_n$ in order: atoms in bodies always depend on a lower stratum, and negated atoms depend on strictly lower strata, whose interpretation was already fixed. Hence, there is a unique least fixpoint and $I \models P$ is well-defined.

▶ **Example 1.** The following stratified Datalog program, with $\sigma = \{R\}$ and $\sigma_{\text{int}} = \{T, \text{Goal}\}$, and strata P_1, P_2, tests if there are two elements that are not connected by a directed R-path:

$$P_1 : T(x,y) \leftarrow R(x,y), \quad T(x,y) \leftarrow R(x,z) \wedge T(z,y) \qquad P_2 : \text{Goal}() \leftarrow \neg T(x,y)$$

Treewidth. Treewidth is a measure quantifying how far a graph is to being a tree, which we use to restrict instances and conjunctive queries. The *treewidth* of a CQ is that of its canonical instance, and the *treewidth* of an instance I is the smallest k such that I has a *tree decomposition* of *width* k, i.e., a finite, rooted, unranked tree T, whose nodes b (called *bags*) are labeled by a subset $\text{dom}(b)$ of $\text{dom}(I)$ with $|\text{dom}(b)| \leqslant k+1$, and which satisfies:
(i) for every fact $R(\mathbf{a}) \in I$, there is a bag $b \in T$ with $\mathbf{a} \subseteq \text{dom}(b)$;
(ii) for all $a \in \text{dom}(I)$, the set of bags $\{b \in T \mid a \in \text{dom}(b)\}$ is a connected subtree of T.
A family of instances is *treelike* if their treewidth is bounded by a constant.

3 Approaches for Tractability

We now review existing approaches to ensure the tractability of query evaluation, starting by query languages whose evaluation is tractable in combined complexity on *all* input instances. We then study more expressive query languages which are tractable on *treelike* instances, but where tractability only holds in data complexity. We then present the goals of our work.

3.1 Tractable Queries on All Instances

The best-known query language to ensure tractable query complexity is *α-acyclic queries* [27], i.e., those that have a tree decomposition where the domain of each bag corresponds exactly to an atom: this is called a *join tree* [33]. With Yannakakis's algorithm [56], we can evaluate an α-acyclic conjunctive query Q on an arbitrary instance I in time $O(|I| \cdot |Q|)$.

Yannakakis's result was generalized in two main directions. One direction [32], moving from linear time to PTIME, has investigated more general CQ classes, in particular CQs of bounded treewidth [28], *hypertreewidth* [33], and *fractional hypertreewidth* [37]. Bounding these query parameters to some fixed k makes query evaluation run in time $O((|I| \cdot |Q|)^{f(k)})$ for some function f, hence in PTIME; for treewidth, since the decomposition can be computed in FPT-linear time [18], this goes down to $O(|I|^k \cdot |Q|)$. However, query evaluation on arbitrary instances is unlikely to be FPT when parameterized by the query treewidth, since it would imply that the exponential-time hypothesis fails (Theorem 5.1 of [46]). Further, even for treewidth 2 (e.g., triangles), it is not known if we can achieve linear data complexity [2].

In another direction, α-acyclicity has been generalized to queries with more expressive operators, e.g., disjunction or negation. The result on α-acyclic CQs thus extends to the *guarded fragment* (GF) of first-order logic, which can be evaluated on arbitrary instances in time $O(|I| \cdot |Q|)$ [43]. Tractability is independently known for FO^k, the fragment of FO where subformulae use at most k variables, with a simple evaluation algorithm in $O(|I|^k \cdot |Q|)$ [55].

Another important operator are *fixpoints*, which can be used to express, e.g., reachability queries. Though FO^k is no longer tractable when adding fixpoints [55], query evaluation is tractable [16, Theorem 3] for μGF, i.e., GF with some restricted least and greatest fixpoint

operators, when *alternation depth* is bounded; without alternation, the combined complexity is in $O(|I| \cdot |Q|)$. We could alternatively express fixpoints in Datalog, but, sadly, most known tractable fragments are nonrecursive: nonrecursive stratified Datalog is tractable [28, Corollary 5.26] for rules with restricted bodies (i.e., strictly acyclic, or bounded strict treewidth). This result was generalized in [34] when bounding the number of guards: this nonrecursive fragment is shown to be equivalent to the k-guarded fragment of FO, with connections to the bounded-hypertreewidth approach. One recursive tractable fragment is Datalog LITE, which is equivalent to alternation-free μGF [31]. Fixpoints were independently studied for graph query languages such as reachability queries and *regular path queries* (RPQ), which enjoy linear combined complexity on arbitrary input instances: this extends to two-way RPQs (2RPQs) and even strongly acyclic conjunctions of 2RPQs (SAC2RPQs), which are expressible in alternation-free μGF. Tractability also extends to acyclic RPQs but with PTIME complexity [9].

3.2 Tractability on Treelike Instances

We now study another approach for tractable query evaluation: this time, we restrict the shape of the *instances*, using treewidth. This ensures that we can translate them to a tree for efficient query evaluation. Informally, having fixed the signature σ, for a fixed treewidth $k \in \mathbb{N}$, there is a finite tree alphabet Γ_σ^k such that σ-instances of treewidth $\leqslant k$ can be translated in FPT-linear time (parameterized by k), following the structure of a tree decomposition, to a Γ_σ^k*-tree*, i.e., a rooted full ordered binary tree with nodes labeled by Γ_σ^k, which we call a *tree encoding*. We omit the formal construction: see Appendix C.1 for more details.

We can then evaluate queries on treelike instances by running *tree automata* on the tree encoding that represents them. Formally, given an alphabet Γ, a *bottom-up nondeterministic tree automaton* on Γ-trees (or Γ-bNTA) is a tuple $A = (Q, F, \iota, \Delta)$, where:

(i) Q is a finite set of *states*;
(ii) $F \in Q$ is a subset of *accepting states*;
(iii) $\iota : \Gamma \to 2^Q$ is an *initialization function* determining the state of a leaf from its label;
(iv) $\Delta : \Gamma \times Q^2 \to 2^Q$ is a *transition function* determining the possible states for an internal node from its label and the states of its two children.

Given a Γ-tree $\langle T, \lambda\rangle$ (where $\lambda : T \to \Gamma$ is the *labeling function*), we define a *run* of A on $\langle T, \lambda\rangle$ as a function $\varphi : T \to Q$ such that (1) $\varphi(l) \in \iota(\lambda(l))$ for every leaf l of T; and (2) $\varphi(n) \in \Delta(\lambda(n), \varphi(n_1), \varphi(n_2))$ for every internal node n of T with children n_1 and n_2. The bNTA A *accepts* $\langle T, \lambda\rangle$ if it has a run on T mapping the root of T to a state of F.

We say that a bNTA A *tests* a query Q for treewidth k if, for any Γ_σ^k-encoding $\langle E, \lambda\rangle$ coding an instance I (of treewidth $\leqslant k$), A accepts $\langle E, \lambda\rangle$ iff $I \models Q$. By a well-known result of Courcelle [23] on graphs (extended to higher-arity in [28]), we can use bNTAs to evaluate all queries in *monadic second-order logic* (MSO), i.e., first-order logic with second-order variables of arity 1. MSO subsumes in particular CQs and monadic Datalog (but not general Datalog). Courcelle showed that MSO queries can be compiled to a bNTA that tests them:

▶ **Theorem 2 [23, 28].** *For any MSO query Q and treewidth $k \in \mathbb{N}$, we can compute a bNTA that tests Q for treewidth k.*

This implies that evaluating any MSO query Q has FPT-linear *data complexity* when parameterized by Q and the instance treewidth [23, 28], i.e., it is in $O\left(f(|Q|, k) \cdot |I|\right)$ for some computable function f. However, this tells little about the combined complexity, as f is generally nonelementary in Q [48]. A better combined complexity bound is known for

unions of conjunctions of two-way regular path queries (UC2RPQs) that are further required to be acyclic and to have a constant number of edges between pairs of variables: these can be compiled into polynomial-sized alternating two-way automata [10].

3.3 Restricted Queries on Treelike Instances

Our approach combines both ideas: we use instance treewidth as a parameter, but also restrict the queries to ensure tractable compilability. We are only aware of two approaches in this spirit. First, Gottlob, Pichler, and Wei [35] have proposed a *quasiguarded* Datalog fragment on *relational structures and their tree decompositions*, with query evaluation is in $O(|I| \cdot |Q|)$. However, this formalism requires queries to be expressed in terms of the tree decomposition, and not just in terms of the relational signature. Second, Berwanger and Grädel [16] remark (after Theorem 4) that, when alternation depth and *width* are bounded, μCGF (the *clique-guarded* fragment of FO with fixpoints) enjoys FPT-linear query evaluation when parameterized by instance treewidth. Their approach does not rely on automata methods, and subsumes the tractability of α-acyclic CQs and alternation-free μGF (and hence SAC2RPQs), on treelike instances. However, μCGF is a restricted query language (the only CQs that it can express are those with a chordal primal graph), whereas we want a richer language, with a parameterized definition.

Our goal is thus to develop an expressive parameterized query language, which can be compiled in *FPT-linear time* to an automaton that tests it (with the treewidth of instances also being a parameter). We can then evaluate the automaton, and obtain FPT-linear combined complexity for query evaluation. Further, as we will show, the use of tree automata will yield *provenance representations* for the query as in [5] (see Section 7).

4 Conjunctive Queries on Treelike Instances

To identify classes of queries that can be efficiently compiled to tree automata, we start by the simplest queries: *conjunctive queries*.

α-acyclic queries. A natural candidate for a tractable query class via automata methods would be *α-acyclic* CQs, which, as we explained in Section 3.1, can be evaluated in time $O(|I| \cdot |Q|)$ on all instances. Sadly, we show that such queries cannot be compiled efficiently to bNTAs, so our compilation result (Theorem 2) does not extend directly:

▶ **Proposition 3.** *There is an arity-two signature σ and an infinite family $Q_1, Q_2, \ldots$ of α-acyclic CQs such that, for any $i \in \mathbb{N}$, any bNTA that tests Q_i for treewidth 1 must have $\Omega(2^{|Q_i|^{1-\varepsilon}})$ states for any $\varepsilon > 0$.*

The intuition of the proof is that bNTAs can only make one traversal of the encoding of the input instance. Faced by this, we propose to use different tree automata formalisms, which are generally more concise than bNTAs. There are two classical generalizations of nondeterministic automata, on words [17] and on trees [22]: one goes from the inherent existential quantification of nondeterminism to *quantifier alternation*; the other allows *two-way* navigation instead of imposing a left-to-right (on words) or bottom-up (on trees) traversal. On words, both of these extensions independently allow for exponentially more compact automata [17]. In this work, we combine both extensions and use *alternating two-way tree automata* [22, 19], formally introduced in Section 6, which leads to tractable combined complexity for evaluation. Our general results in the next section will then imply:

▶ **Proposition 4.** *For any treewidth bound $k_{\mathrm{I}} \in \mathbb{N}$, given an α-acyclic CQ Q, we can compute in FPT-linear time in $O(|Q|)$ (parameterized by k_{I}) an alternating two-way tree automaton that tests it for treewidth k_{I}.*

Hence, if we are additionally given a relational instance I of treewidth $\leqslant k_{\mathrm{I}}$, one can determine whether $I \models Q$ in FPT-linear time in $|I| \cdot |Q|$ (parameterized by k_{I}).

Bounded-treewidth queries. Having re-proven the combined tractability of α-acyclic queries (on bounded-treewidth instances), we naturally try to extend to *bounded-treewidth* CQs. Recall from Section 3.1 that these queries have PTIME combined complexity on all instances, but are unlikely to be FPT when parameterized by the query treewidth [46]. Can they be efficiently evaluated on treelike instances by compiling them to automata? We answer in the negative: that bounded-treewidth CQs *cannot* be efficiently compiled to automata to test them, even when using the expressive formalism of alternating two-way tree automata [22]:

▶ **Theorem 5.** *There is an arity-two signature σ for which there is no algorithm $\mathcal{A}$ with exponential running time and polynomial output size for the following task: given a conjunctive query Q of treewidth $\leqslant 2$, produce an alternating two-way tree automaton A_Q on Γ_σ^5-trees that tests Q on σ-instances of treewidth $\leqslant 5$.*

This result is obtained from a variant of the 2EXPTIME-hardness of monadic Datalog containment [11]. We show that efficient compilation of bounded-treewidth CQs to automata would yield an EXPTIME containment test, and conclude by the time hierarchy theorem.

Bounded simplicial width. We have shown that we cannot compile bounded-treewidth queries to automata efficiently. We now show that efficient compilation can be ensured with an additional requirement on tree decompositions. As it turns out, the resulting decomposition notion has been independently introduced for graphs:

▶ **Definition 6 [26].** A *simplicial decomposition of a graph G* is a tree decomposition T of G such that, for any bag b of T and child bag b' of b, letting S be the intersection of the domains of b and b', then the subgraph of G induced by S is a complete subgraph of G.

We extend this notion to CQs, and introduce the *simplicial width* measure:

▶ **Definition 7.** A *simplicial decomposition of a CQ Q* is a simplicial decomposition of its primal graph. Note that any CQ has a simplicial decomposition (e.g., the trivial one that puts all variables in one bag). The *simplicial width* of Q is the minimum, over all simplicial tree decompositions, of the size of the largest bag minus 1.

Bounding the simplicial width of CQs is of course more restrictive than bounding their treewidth, and this containment relation is strict: cycles have treewidth $\leqslant 2$ but have unbounded simplicial width. This being said, bounding the simplicial width is less restrictive than imposing α-acyclicity: the join tree of an α-acyclic CQ is in particular a simplicial decomposition, so α-acyclic CQs have simplicial width at most $\mathrm{arity}(\sigma) - 1$, which is constant as σ is fixed. Again, the containment is strict: a triangle has simplicial width 2 but is not α-acyclic.

To our knowledge, simplicial width for CQs has not been studied before. Yet, we show that bounding the simplicial width ensures that CQs can be efficiently compiled to automata. This is unexpected, because the same is not true of treewidth, by Theorem 5. Hence:

▶ **Theorem 8.** *For any $k_{\mathrm{I}}, k_{\mathrm{Q}} \in \mathbb{N}$, given a CQ Q and a simplicial decomposition T of simplicial width k_{Q} of Q, we can compute in FPT-linear in $|Q|$ (parameterized by k_{I} and k_{Q}) an alternating two-way tree automaton that tests Q for treewidth k_{I}.*

Hence, if we are additionally given a relational instance I of treewidth $\leqslant k_{\mathrm{I}}$, one can determine whether $I \models Q$ in FPT-linear time in $|I| \cdot (|Q| + |T|)$ (parameterized by k_{I} and k_{Q}).

Notice the technicality that the simplicial decomposition T must be provided as input to the procedure, because it is not known to be computable in FPT-linear time, unlike tree decompositions. While we are not aware of results on the complexity of this specific task, quadratic time algorithms are known for the related problem of computing the *clique-minimal separator decomposition* [42, 15].

The intuition for the efficient compilation of bounded-simplicial-width CQs is as follows. The *interface* variables shared between any bag and its parent must be "clique-guarded" (each pair is covered by an atom). Hence, consider any subquery rooted at a bag of the query decomposition, and see it as a non-Boolean CQ with the interface variables as free variables. Each result of this CQ must then be covered by a clique of facts of the instance, which ensures [30] that it occurs in some bag in the instance tree decomposition and can be "seen" by a tree automaton. This intuition can be generalized, beyond conjunctive queries, to design an expressive query language featuring disjunction, negation, and fixpoints, with the same properties of efficient compilation to automata and FPT-linear combined complexity of evaluation on treelike instances. We introduce such a Datalog variant in the next section.

5 ICG-Datalog on Treelike Instances

To design a Datalog fragment with efficient compilation to automata, we must of course impose some limitations, as we did for CQs. In fact, we can even show that the full Datalog language (even without negation) *cannot* be compiled to automata, no matter the complexity:

▶ **Proposition 9.** *There is a signature σ and Datalog program P such that the language of Γ^1_σ-trees that encode instances satisfying P is not a regular tree language.*

Hence, there is no bNTA or alternating two-way tree automaton that tests P for treewidth 1. To work around this problem and ensure that compilation is possible and efficient, the key condition that we impose on Datalog programs, pursuant to the intuition of simplicial decompositions, is that intensional predicates in rule bodies must be *clique-guarded*, i.e., their variables must co-occur in *extensional* predicates of the rule body. We can then use the *body size* of the program rules as a parameter, and will show that the fragment can then be compiled to automata in FPT-linear time.

▶ **Definition 10.** Let P be a stratified Datalog program. An intensional literal $A(\mathbf{x})$ or $\neg A(\mathbf{x})$ in a rule body ψ of P is *clique-guarded* if, for any two variables $x_i \neq x_j$ of $\mathbf{x}$, x_i and x_j co-occur in some extensional atom of ψ. P is *intensional-clique-guarded* (ICG) if, for any rule $R(\mathbf{x}) \leftarrow \psi(\mathbf{x}, \mathbf{y})$, every *intensional* literal in ψ is clique-guarded in ψ. The *body size* of P is the maximal number of atoms in the body of its rules, multiplied by its arity.

The main result of this paper is that evaluation of ICG-Datalog is *FPT-linear* in *combined complexity*, when parameterized by the body size of the program and the instance treewidth.

▶ **Theorem 11.** *Given an ICG-Datalog program P of body size k_{P} and a relational instance I of treewidth k_{I}, checking if $I \models P$ is FPT-linear time in $|I| \cdot |P|$ (parameterized by k_{P} and k_{I}).*

We will show this result in the next section by compiling ICG-Datalog programs in FPT-linear time to a special kind of tree automata (Theorem 22), and showing in Section 7 that we can efficiently evaluate such automata and even compute *provenance representations*. The rest of this section presents consequences of our main result for various languages.

Conjunctive queries. Our tractability result for bounded-simplicial-width CQs (Theorem 8), including α-acyclic CQs, is shown by rewriting to ICG-Datalog of bounded body size:

▶ **Proposition 12.** *There is a function f_σ (depending only on σ) such that for all $k \in \mathbb{N}$, for any conjunctive query Q and simplicial tree decomposition T of Q of width at most k, we can compute in $O(|Q| + |T|)$ an equivalent ICG-Datalog program with body size at most $f_\sigma(k)$.*

This implies that ICG-Datalog can express any CQ up to increasing the body size parameter, unlike, e.g., μCGF. Conversely, we can show that bounded-simplicial-width CQs *characterize* the queries expressible in ICG-Datalog when disallowing negation, recursion and disjunction. Specifically, a Datalog program is *positive* if it contains no negated atoms. It is *nonrecursive* if there is no cycle in the directed graph on σ_{int} having an edge from R to S whenever a rule contains R in its head and S in its body. It is *conjunctive* [13] if each intensional relation R occurs in the head of at most one rule. We can then show:

▶ **Proposition 13.** *For any positive, conjunctive, nonrecursive ICG-Datalog program P with body size k, there is a CQ Q of simplicial width $\leqslant k$ that is equivalent to P.*

However, our ICG-Datalog fragment is still exponentially more *concise* than such CQs:

▶ **Proposition 14.** *There is a signature σ and a family $(P_n)_{n\in\mathbb{N}}$ of ICG-Datalog programs with body size at most 9 which are positive, conjunctive, and nonrecursive, such that $|P_n| = O(n)$ and any CQ Q_n equivalent to P_n has size $\Omega(2^n)$.*

Guarded negation fragments. Having explained the connections between ICG-Datalog and CQs, we now study its connections to the more expressive languages of guarded logics, specifically, the *guarded negation fragment* (GNF), a fragment of first-order logic [8]. Indeed, when putting GNF formulae in *GN-normal form* [8] or even *weak GN-normal form* [14], we can translate them to ICG-Datalog, and we can use the *CQ-rank* parameter [14] (that measures the maximal number of atoms in conjunctions) to control the body size parameter.

▶ **Proposition 15.** *There is a function f_σ (depending only on σ) such that, for any weak GN-normal form GNF query Q of CQ-rank r, we can compute in time $O(|Q|)$ an equivalent nonrecursive ICG-Datalog program P of body size $f_\sigma(r)$.*

In fact, the efficient compilation of bounded-CQ-rank normal-form GNF programs (using the fact that subformulae are "answer-guarded", like our guardedness requirements) has been used recently (e.g., in [12]), to give efficient procedures for GNF *satisfiability*, compiling them to automata (to a treewidth which is not fixed, unlike in our context, but depends on the formula). ICG-Datalog further allows clique guards (similar to CGNFO [8]), can reuse subformulae (similar to the idea of DAG-representations in [14]), and supports recursion (similar to GNFP [8], or GN-Datalog [7] but whose combined complexity is intractable — P^{NP}-complete). ICG-Datalog also resembles μCGF [16], but remember that it is not a guarded *negation* logic, so, e.g., μCGF cannot express all CQs.

Hence, the design of ICG-Datalog, and its compilation to automata, has similarities with guarded logics. However, to our knowledge, the idea of applying it to query evaluation is new, and ICG-Datalog is designed to support all relevant features to capture interesting query languages (e.g., clique guards are necessary to capture bounded-simplicial-width queries).

Recursive languages. The use of fixpoints in ICG-Datalog, in particular, allows us to capture the combined tractability of interesting recursive languages. First, observe that our guardedness requirement becomes trivial when all intensional predicates are monadic (arity-one), so our main result implies that *monadic Datalog* of bounded body size is tractable in combined complexity on treelike instances. This is reminiscent of the results of [35]:

▶ **Proposition 16.** *The combined complexity of monadic Datalog query evaluation on bounded-treewidth instances is FPT when parameterized by instance treewidth and body size (as in Definition 10) of the monadic Datalog program.*

Second, ICG-Datalog can capture *two-way regular path queries* (2RPQs) [20, 9], a well-known query language in the context of graph databases and knowledge bases:

▶ **Definition 17.** We assume that the signature σ contains only binary relations. A *regular path query* (RPQ) Q_L is defined by a regular language L on the alphabet Σ of the relation symbols of σ. Its semantics is that Q_L has two free variables x and y, and $Q_L(a, b)$ holds on an instance I for $a, b \in \mathrm{dom}(I)$ precisely when there is a directed path π of relations of σ from a to b such that the label of π is in L. A *two-way regular path query* (2RPQ) is an RPQ on the alphabet $\Sigma^{\pm} := \Sigma \sqcup \{R^- \mid R \in \Sigma\}$, which holds whenever there is a path from a to b with label in L, with R^- meaning that we traverse an R-fact in the reverse direction. A *Boolean 2RPQ* is a 2RPQ which is existentially quantified on its two free variables.

▶ **Proposition 18 [47, 9].** *2RPQ query evaluation (on arbitrary instances) has linear time combined complexity.*

ICG-Datalog allows us to capture this result for Boolean 2RPQs on treelike instances. In fact, the above result extends to SAC2RPQs, which are trees of 2RPQs with no multi-edges or loops. We can prove the following result, for Boolean 2RPQs and SAC2RPQs, which further implies compilability to automata (and efficient compilation of provenance representations). We do not know whether this extends to the more general classes studied in [10].

▶ **Proposition 19.** *Given a Boolean SAC2RPQ Q, we can compute in time $O(|Q|)$ an equivalent ICG-Datalog program P of body size 4.*

6 Compilation to Automata

In this section, we study how we can compile ICG-Datalog queries on treelike instances to tree automata, to be able to evaluate them efficiently. As we showed with Proposition 3, we need more expressive automata than bNTAs. Hence, we use instead the formalism of *alternating two-way automata* [22], i.e., automata that can navigate in trees in any direction, and can express transitions using Boolean formulae on states. Specifically, we introduce for our purposes a variant of these automata, which are *stratified* (i.e., allow a form of stratified negation), and *isotropic* (i.e., no direction is privileged, in particular order is ignored).

As in Section 3.2, we will define tree automata that run on Γ-*trees* for some alphabet Γ: a Γ-tree $\langle T, \lambda\rangle$ is a finite rooted ordered tree with a labeling function λ from the nodes of T to Γ. The *neighborhood* $\mathrm{Nbh}(n)$ of a node $n \in T$ is the set which contains n, all children of n, and the parent of n if it exists.

Stratified isotropic alternating two-way automata. To define the transitions of our alternating automata, we write $\mathscr{B}(X)$ the set of propositional formulae (not necessarily monotone) over a set X of variables: we will assume w.l.o.g. that *negations are only applied to variables*, which we can always enforce using de Morgan's laws. A *literal* is a propositional variable $x \in X$ (*positive* literal) or the negation of a propositional variable $\neg x$ (*negative* literal).

A *satisfying assignment* of $\varphi \in \mathscr{B}(X)$ consists of two *disjoint* sets $P, N \subseteq X$ (for "positive" and "negative") such that φ is a tautology when substituting the variables of P with 1 and those of N with 0, i.e., when we have $\nu(\varphi) = 1$ for every valuation ν of X such that $\nu(x) = 1$ for all $x \in P$ and $\nu(x) = 0$ for all $x \in N$. Note that we allow satisfying assignments with $P \sqcup N \subsetneq X$, which will be useful for our technical results. We now define our automata:

▶ **Definition 20.** A *stratified isotropic alternating two-way automata* on Γ-trees (Γ-SATWA) is a tuple $A = (\mathcal{Q}, q_{\mathrm{I}}, \Delta, \zeta)$ with $\mathcal{Q}$ a finite set of *states*, q_{I} the *initial state*, Δ the *transition function* from $\mathcal{Q} \times \Gamma$ to $\mathscr{B}(\mathcal{Q})$, and ζ a *stratification function*, i.e., a surjective function from $\mathcal{Q}$ to $\{0, \ldots, m\}$ for some $m \in \mathbb{N}$, such that for any $q, q' \in \mathcal{Q}$ and $f \in \Gamma$, if $\Delta(q, f)$ contains q' as a positive literal (resp., negative literal), then $\zeta(q') \leqslant \zeta(q)$ (resp. $\zeta(q') < \zeta(q)$).

We define by induction on $0 \leqslant i \leqslant m$ an *i-run* of A on a Γ-tree $\langle T, \lambda\rangle$ as a finite tree $\langle T_{\mathrm{r}}, \lambda_{\mathrm{r}}\rangle$, with labels of the form (q, w) or $\neg(q, w)$ for $w \in T$ and $q \in \mathcal{Q}$ with $\zeta(q) \leqslant i$, by the following recursive definition for all $w \in T$:

- For $q \in \mathcal{Q}$ such that $\zeta(q) < i$, the singleton tree $\langle T_{\mathrm{r}}, \lambda_{\mathrm{r}}\rangle$ with one node labeled by (q, w) (resp., by $\neg(q, w)$) is an i-run if there is a $\zeta(q)$-run of A on $\langle T, \lambda\rangle$ whose root is labeled by (q, w) (resp., if there is no such run);
- For $q \in \mathcal{Q}$ such that $\zeta(q) = i$, if $\Delta(q, \lambda(w))$ has a satisfying assignment (P, N), if we have a $\zeta(q')$-run T_{q^-} for each $q^- \in N$ with root labeled by $\neg(q^-, w)$, and a $\zeta(q^+)$-run T_{q^+} for each $q^+ \in P$ with root labeled by (q^+, w_{q^+}) for some w_{q^+} in $\mathrm{Nbh}(w)$, then the tree $\langle T_{\mathrm{r}}, \lambda_{\mathrm{r}}\rangle$ whose root is labeled (q, w) and has as children all the T_{q^-} and T_{q^+} is an i-run.

A *run* of A starting in a state $q \in \mathcal{Q}$ at a node $w \in T$ is a m-run whose root is labeled (q, w). We say that A *accepts* $\langle T, \lambda\rangle$ (written $\langle T, \lambda\rangle \models A$) if there exists a run of A on $\langle T, \lambda\rangle$ starting in the initial state q_{I} at the root of T.

Observe that the internal nodes of a run starting in some state q are labeled by states q' in the same stratum as q. The leaves of the run may be labeled by states of a strictly lower stratum or negations thereof, or by states of the same stratum whose transition function is tautological, i.e., by some (q', w) such that $\Delta(q', \lambda(w))$ has $\emptyset, \emptyset$ as a satisfying assignment. Intuitively, if we disallow negation in transitions, our automata amount to the alternating two-way automata used by [19], with the simplification that they do not need parity acceptance conditions (because we only work with finite trees), and that they are *isotropic*: the run for each positive child state of an internal node may start indifferently on *any* neighbor of w in the tree (its parent, a child, or w itself), no matter the direction. (Note, however, that the run for negated child states must start on w itself.)

We will soon explain how the compilation of ICG-Datalog is performed, but we first note that evaluation of Γ-SATWAs is in linear time:

▶ **Proposition 21.** *For any alphabet Γ, given a Γ-tree T and a Γ-SATWA A, we can determine whether $T \models A$ in time $O(|T| \cdot |A|)$.*

In fact, this result follows from the definition of provenance cycluits for SATWAs in the next section, and the claim that these cycluits can be evaluated in linear time.

Compilation. We now give our main compilation result: we can efficiently compile any ICG-Datalog program of bounded body size into a SATWA that *tests* it (in the same sense as for bNTAs). This is our main technical claim, which is proven in Appendix C.3.

▶ **Theorem 22.** *Given an ICG-Datalog program P of body size k_{P} and $k_{\mathrm{I}} \in \mathbb{N}$, we can build in FPT-linear time in $|P|$ (parameterized by $k_{\mathrm{P}}, k_{\mathrm{I}}$) a SATWA A_P testing P for treewidth k_{I}.*

Proof sketch. The idea is to have, for every relational symbol R, states of the form $q^{\nu}_{R(\mathbf{x})}$, where ν is a partial valuation of $\mathbf{x}$. This will be the starting state of a run if it is possible to navigate the tree encoding from some starting node and build in this way a total valuation ν' that extends ν and such that $R(\nu'(\mathbf{x}))$ holds. When R is intensional, once ν' is total on $\mathbf{x}$, we go into a state of the form $q_r^{\nu', \mathcal{A}}$ where r is a rule with head relation R and $\mathcal{A}$ is the set of atoms in the body of r (whose size is bounded by the body size). This means that

we choose a rule to prove $R(\nu'(\mathbf{x}))$. The automaton can then navigate the tree encoding, build ν' and coherently partition $\mathcal{A}$ so as to inductively prove the atoms of the body. The clique-guardedness condition ensures that, when there is a match of $R(\mathbf{x})$, the elements to which $\mathbf{x}$ is mapped can be found together in a bag. The fact that the automaton is isotropic relieves us from the syntactic burden of dealing with directions in the tree, as one usually has to do with alternating two-way automata. ◀

7 Provenance Cycluits

In the previous section, we have seen how ICG-Datalog programs could be compiled efficiently to tree automata (SATWAs) that test them on treelike instances. To show that SATWAs can be evaluated in linear time (stated earlier as Proposition 21), we will introduce an operational semantics for SATWAs based on the notion of *cyclic circuits*, or *cycluits* for short.

We will also use these cycluits as a new powerful tool to compute (Boolean) *provenance information*, i.e., a representation of how the query result depends on the input data:

▶ **Definition 23.** A (Boolean) *valuation* of a set S is a function $\nu : S \to \{0, 1\}$. A *Boolean function* φ on variables S is a mapping that associates to each valuation ν of S a Boolean value in $\{0, 1\}$ called the *evaluation* of φ according to ν; for consistency with further notation, we write it $\nu(\varphi)$. The *provenance* of a query Q on an instance I is the Boolean function φ, whose variables are the facts of I, which is defined as follows: for any valuation ν of the facts of I, we have $\nu(\varphi) = 1$ iff the subinstance $\{F \in I \mid \nu(F) = 1\}$ satisfies Q.

We can represent Boolean provenance as Boolean formulae [38, 36], or (more recently) as Boolean circuits [25, 5]. In this section, we first introduce *monotone cycluits* (monotone Boolean circuits with cycles), for which we define a semantics (in terms of the Boolean function that they express); we also show that cycluits can be evaluated in linear time, given a valuation. Second, we extend them to *stratified cycluits*, allowing a form of stratified negation. We conclude the section by showing how to construct the *provenance* of a SATWA as a cycluit, in FPT-linear time. Together with Theorem 22, this claim implies our main provenance result:

▶ **Theorem 24.** *Given an ICG-Datalog program P of body size k_{P} and a relational instance I of treewidth k_{I}, we can construct in FPT-linear time in $|I| \cdot |P|$ (parameterized by k_{P} and k_{I}) a representation of the provenance of P on I as a stratified cycluit. Further, for fixed k_{I}, this cycluit has treewidth $O(|P|)$.*

Of course, this result implies the analogous claims for query languages that are captured by ICG-Datalog parameterized by the body size, as we studied in Section 5. When combined with the fact that cycluits can be tractably evaluated, it yields our main result, Theorem 11. The rest of this section formally introduces cycluits and proves Theorem 24.

Cycluits. We define *cycluits* as Boolean circuits without the acyclicity requirement, as in [50]. To avoid the problem of feedback loops, however, we first study *monotone cycluits*, and then cycluits with stratified negation.

▶ **Definition 25.** A *monotone Boolean cycluit* is a directed graph $C = (G, W, g_0, \mu)$ where G is the set of *gates*, $W \subseteq G^2$ is the set of directed edges called *wires* (and written $g \to g'$), $g_0 \in G$ is the *output gate*, and μ is the *type* function mapping each gate $g \in G$ to one of inp (input gate, with no incoming wire in W), $\wedge$ (AND gate) or $\vee$ (OR gate).

We now define the semantics of monotone cycluits. A (Boolean) *valuation* of C is a function $\nu : C_{\mathsf{inp}} \to \{0,1\}$ indicating the value of the input gates. As for standard monotone circuits, a valuation yields an *evaluation* $\nu' : C \to \{0,1\}$, that we will define shortly, indicating the value of each gate under the valuation ν: we abuse notation and write $\nu(C) \in \{0,1\}$ for the *evaluation result*, i.e., $\nu'(g_0)$ where g_0 is the output gate of C. The Boolean function *captured* by a cycluit C is thus the Boolean function φ on C_{inp} defined by $\nu(\varphi) := \nu(C)$ for each valuation ν of C_{inp}. We define the evaluation ν' from ν by a least fixed-point computation (see Algorithm 1 in Appendix D.1): we set all input gates to their value by ν, and other gates to 0. We then iterate until the evaluation no longer changes, by evaluating OR-gates to 1 whenever some input evaluates to 1, and AND-gates to 1 whenever all their inputs evaluate to 1. The Knaster–Tarski theorem [53] gives an equivalent characterization:

▶ **Proposition 26.** *For any monotone cycluit C and Boolean valuation ν of C, the set $S := \{g \in C \mid \nu'(g) = 1\}$ is* the *minimal set of gates (under inclusion) such that:*
(i) S contains the true input gates, i.e., it contains $\{g \in C_{\mathsf{inp}} \mid \nu(g) = 1\}$;
(ii) for any g such that $\mu(g) = \vee$, if some input gate of g is in S, then g is in S;
(iii) for any g such that $\mu(g) = \wedge$, if all input gates of g are in S, then g is in S.

We show that this definition is computable in linear time (Algorithm 2 in Appendix D.1):

▶ **Proposition 27.** *Given any monotone cycluit C and Boolean valuation ν of C, we can compute the evaluation ν' of C in linear time.*

Stratified cycluits. We now move from monotone cycluits to general cycluits featuring negation. However, allowing arbitrary negation would make it difficult to define a proper semantics, because of possible cycles of negations. Hence, we focus on *stratified cycluits*:

▶ **Definition 28.** A *Boolean cycluit* C is defined like a *monotone cycluit*, but further allows NOT-gates ($\mu(g) = \neg$), which are required to have a single input. It is *stratified* if there exists a *stratification function* ζ mapping its gates surjectively to $\{0, \ldots, m\}$ for some $m \in \mathbb{N}$ such that $\zeta(g) = 0$ iff $g \in C_{\mathsf{inp}}$, and $\zeta(g) \leqslant \zeta(g')$ for each wire $g \to g'$, the inequality being strict if $\mu(g') = \neg$.

Equivalently, C contains no cycle of gates involving a $\neg$-gate. If C is stratified, we can compute a stratification function in linear time by a topological sort, and use it to define the evaluation of C (which will clearly be independent of the choice of stratification function):

▶ **Definition 29.** Let C be a stratified cycluit with stratification function $\zeta : C \to \{0, \ldots, m\}$, and let ν be a Boolean valuation of C. We inductively define the *i-th stratum evaluation* ν_i, for i in the range of ζ, by setting $\nu_0 := \nu$, and letting ν_i extend the ν_j ($j < i$) as follows:
1. For g such that $\zeta(g) = i$ with $\mu(g) = \neg$, set $\nu_i(g) := \neg\nu_{\zeta(g')}(g')$ for its one input g'.
2. Evaluate all other g with $\zeta(g) = i$ as for monotone cycluits, considering the $\neg$-gates of point **1.** and all gates of lower strata as input gates fixed to their value in ν_{i-1}.
Letting g_0 be the output gate of C, the Boolean function φ *captured* by C is then defined as $\nu(\varphi) := \nu_m(g_0)$ for each valuation ν of C_{inp}.

▶ **Proposition 30.** *We can compute $\nu(C)$ in linear time in the stratified cycluit C and in ν.*

Building provenance cycluits. Having defined cycluits as our provenance representation, we compute the provenance of a query on an instance as the *provenance* of its SATWA on a tree encoding. To do so, we must give a general definition of the provenance of SATWAs. Consider a Γ-tree $\mathcal{T} := \langle T, \lambda \rangle$ for some alphabet Γ, as in Section 6. We define a (Boolean)

valuation ν of $\mathcal{T}$ as a mapping from the nodes of T to $\{0,1\}$. Writing $\overline{\Gamma} := \Gamma \times \{0,1\}$, each valuation ν then defines a $\overline{\Gamma}$-tree $\nu(\mathcal{T}) := \langle T, (\lambda \times \nu)\rangle$, obtained by annotating each node of $\mathcal{T}$ by its ν-image. As in [5], we define the provenance of a $\overline{\Gamma}$-SATWA A on $\mathcal{T}$, which intuitively captures all possible results of evaluating A on possible valuations of $\mathcal{T}$:

▶ **Definition 31.** The *provenance* of a $\overline{\Gamma}$-SATWA A on a Γ-tree $\mathcal{T}$ is the Boolean function φ defined on the nodes of T such that, for any valuation ν of $\mathcal{T}$, $\nu(\varphi) = 1$ iff A accepts $\nu(\mathcal{T})$.

We then show that we can efficiently build provenance representations of SATWAs on trees as stratified cycluits:

▶ **Theorem 32.** *For any fixed alphabet Γ, given a $\overline{\Gamma}$-SATWA A and a Γ-tree $\mathcal{T}$, we can build a stratified cycluit capturing the provenance of A on $\mathcal{T}$ in time $O(|A| \cdot |\mathcal{T}|)$. Moreover, this stratified cycluit has treewidth $O(|A|)$.*

Proof sketch. The construction generalizes Proposition 3.1 of [5] from bNTAs and circuits to SATWAs and cycluits. For each node w of T and state q, we create a gate g_w^q which, following the transitions of A, is connected to those created for the neighbors of w; g_w^q is such that for every Boolean valuation $\nu : T \to \{0,1\}$ of the inputs of C, there exists a run ρ of A on $\nu(T)$ starting at w in state q if and only if $\nu(g_w^q) = 1$. The reason why we need cycluits rather than circuits is because SATWAs may loop back on previously visited nodes. ◀

Note that the proof can be easily modified to make it work for standard alternating two-way automata rather than our isotropic automata. This result allows us to prove Theorem 24, by applying it to the SATWA obtained from the ICG-Datalog program (Theorem 22), slightly modified so as to extend it to the alphabet $\overline{\Gamma}$. Recalling that nodes of the tree encodings each encode at most one fact of the instance, we use the second coordinate of $\overline{\Gamma}$ to indicate whether the fact is actually present or should be discarded. This allows us to range over possible subinstances, and thus to compute the provenance. This concludes the proof of our main result (Theorem 11 in Section 5): we can evaluate an ICG-Datalog program on a treelike instance in FPT-linear time by computing its provenance by Theorem 24 and evaluating the provenance in linear time (Proposition 30).

8 From Cycluits to Circuits and Probability Bounds

We have proven our main result on ICG-Datalog, Theorem 11, in the previous section, introducing stratified cycluits in the process as a way to capture the provenance of ICG-Datalog. In this section, we study how these stratified cycluits can be transformed into *equivalent* acyclic Boolean circuits, and we then show how we can use this to derive bounds for the *probabilistic query evaluation* problem (PQE).

From cycluits to circuits. We call two cycluits or circuits C_1 and C_2 *equivalent* if they have the same set of inputs C_{inp} and, for each valuation ν of C_{inp}, we have $\nu(C_1) = \nu(C_2)$. A first result from existing work is that we can remove cycles in cycluits and convert them to circuits, with a quadratic blowup, by creating linearly many copies to materialize the fixpoint computation. This allows us to remain FPT in combined complexity, but not FPT-linear:

▶ **Proposition 33 ([50], Theorem 2).** *For any stratified cycluit C, we can compute in time $O(|C|^2)$ a Boolean circuit C' which is equivalent to C.*

In addition to being quadratic rather than linear, another disadvantage of this approach is that bounds on the treewidth of the cycluit (which we will need later for probability

computation) are generally not preserved on the output. Hence, we prove a second cycle removal result, that proceeds in FPT-linear time when parameterized by the treewidth of the input cycluit. When we use this result, we no longer preserve FPT combined complexity of the overall computation, because the stratified cycluits produced by Theorem 24 generally have treewidth $\Omega(|P|)$. On the other hand, we obtain an FPT-linear data complexity bound, and a bounded-treewidth circuit as a result.

▶ **Theorem 34.** *There is an $\alpha \in \mathbb{N}$ s.t., for any stratified cycluit C of treewidth k, we can compute in time $O(2^{k^\alpha}\,|C|)$ a circuit C' which is equivalent to C and has treewidth $O(2^{k^\alpha})$.*

Proof sketch. The proof is technical and proceeds stratum by stratum, so it focuses on monotone cycluits. For such circuits, we rewrite tree decompositions to a normal form that ensures that gate definitions occur only in leaf bags, and with all relevant input gates in scope. We then perform our rewriting bottom-up on the tree decomposition, by creating gates at each bag to code the behavior of the sub-circuit in terms of the values of the yet-undefined internal gates propagated from the parent bag: this requires a fixpoint computation, coded by iterating in the circuits for the two subtrees rooted at the children of the current bag. We show it is equivalent to standard cycluit evaluation, and that the number of required iterations is bounded by the bag size. Further, as higher strata may depend on any gate (not just on one single output), we use a second top-down pass to perform the previous process for each rooting of the tree decomposition in overall linear time. ◀

Probabilistic query evaluation. We can then apply the above result to the probabilistic query evaluation (PQE) problem, which we now define:

▶ **Definition 35.** A *TID instance* is a relational instance I and a function π mapping each fact $F \in I$ to a rational probability $\pi(F)$. A TID instance (I, π) defines a probability distribution Pr on $I' \subseteq I$, where $\Pr(I') := \prod_{F \in I'} \pi(F) \times \prod_{F \in I \setminus I'} (1 - \pi(F))$.

The *probabilistic query evaluation* (PQE) problem asks, given a Boolean query Q and a TID instance (I, π), the probability that the query Q is satisfied in the distribution Pr of (I, π). Formally, we want to compute $\sum_{I' \subseteq I \text{ s.t. } I' \models Q} \Pr(I')$. The *data complexity* of PQE is its complexity when Q is fixed and the TID instance (I, π) is given as input. Its *combined complexity* is its complexity when both the query and TID instance are given as input.

Earlier work [24] showed that PQE has #P-hard data complexity even for some CQs of a simple form, but [5, 4] shows that PQE is tractable in data complexity for any Boolean query in monadic second-order (MSO) if the input instances are required to be treelike.

We now explain how to use Theorem 34 for PQE. Let P be an ICG-Datalog program of body size k_{P}. Given a TID instance (I, π) of treewidth k_{I}, we compute a provenance cycluit for P on I of treewidth $O(|P|)$ in FPT-linear time in $|I| \cdot |P|$ by Theorem 24. By Theorem 34, we compute in $O(2^{|P|^\alpha}\,|I|\,|P|)$ an equivalent circuit of treewidth $O(2^{|P|^\alpha})$. Now, by Theorem D.2 of [4], we can solve PQE for P and (I, π) in $O(2^{2^{|P|^\alpha}}\,|I|\,|P| + |\pi|)$ up to PTIME arithmetic costs. Linear-time data complexity was known from [5], but 2EXPTIME combined complexity is novel, as [5] only gave non-elementary combined complexity bounds.

Acyclic queries on tree TIDs. A natural question is then to understand whether better bounds are possible. In particular, is PQE tractable in *combined complexity* on treelike instances? We show that, unfortunately, treewidth bounds are *not* sufficient to ensure this. The proof draws some inspiration from earlier work [39] on the topic of tree-pattern query evaluation in *probabilistic XML* [40].

▶ **Proposition 36.** *There is a fixed arity-two signature on which PQE is #P-hard even when imposing that the input instances have treewidth 1 and the input queries are α-acyclic CQs.*

Path queries on tree TIDs. We must thus restrict the query language further to achieve combined tractability. One natural restriction is to go from *α-acyclic queries* to *path queries*, i.e., Boolean CQs of the form $R_1(x_1, x_2), \ldots, R_n(x_{n-1}, x_n)$, where each R_i is a binary relation of the signature. For instance, $R(x, y), S(y, z), T(z, w)$ is a path query, but $R(x, y), S(z, y)$ is not (we do not allow inverse relations). We can strengthen the previous result to show:

▶ **Proposition 37.** *There is a fixed arity-two signature on which PQE is #P-hard even when imposing that the input instances have treewidth* 1 *and the input queries are path queries.*

Tractable cases. In which cases, then, could PQE be tractable in combined complexity? One example is in [21]: PQE is tractable in combined complexity over *probabilistic XML*, when queries are written as *deterministic* tree automata. In this setting, that the edges of the XML document are *directed* (preventing, e.g., the inverse construction used in the proof of Proposition 37). Further, as the result works on *unranked* trees, it is important that children of a node are *ordered* as well (see [3] for examples where this matters).

We leave open the question of whether there are some practical classes of instances and of queries for which such a deterministic tree automaton can be obtained from the query in polynomial time to test the query for a given treewidth. As we have shown, path queries and instances of treewidth 1, even though very restricted, do not suffice to ensure this. Note that, in terms of data complexity, we have shown in [6] that treelike instances are essentially the only instances for which first-order tractability is achievable.

9 Conclusion

We introduced ICG-Datalog, a new stratified Datalog fragment whose evaluation has FPT-linear complexity when parameterized by instance treewidth and program body size. The complexity result is obtained via compilation to alternating two-way automata, and via the computation of a provenance representation in the form of stratified cycluits, a generalisation of provenance circuits that we hope to be of independent interest.

We believe that ICG-Datalog can be further improved by removing the guardedness requirement on negated atoms, which would make it more expressive and step back from the world of guarded negation logics. In particular, we conjecture that our FPT-linear tractability result generalizes to *frontier-guarded Datalog*, and its extensions with clique-guards and stratified (but unguarded) negation, taking the rule body size and instance treewidth as the parameters. We further hope that our results could be used to derive PTIME combined complexity results on instances of arbitrary treewidth, e.g., XP membership when parametrizing by program size; this could in particular recapture the tractability of bounded-treewidth queries. Last, we intend to extend our cycluit framework to support more expressive provenance semirings than Boolean provenance (e.g., formal power series [36]).

We leave open the question of practical implementation of the methods we developed, but we have good hopes that this approach can give efficient results in practice, in part from our experience with a preliminary provenance prototype [49]. Optimization is possible, for instance by not representing the full automata but building them on the fly when needed in query evaluation. Another promising direction supported by our experience, to deal with real-world datasets that are not treelike, is to use partial tree decompositions [45].

Acknowledgements. This work was partly funded by the Télécom ParisTech Research Chair on Big Data and Market Insights.

References

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

2 N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3), 1997.

3 A. Amarilli. The possibility problem for probabilistic XML. In *AMW*, 2014.

4 A. Amarilli, P. Bourhis, and P. Senellart. Provenance circuits for trees and treelike instances (extended version). *CoRR*, abs/1511.08723, 2015. Extended version of [5].

5 A. Amarilli, P. Bourhis, and P. Senellart. Provenance circuits for trees and treelike instances. In *ICALP*, volume 9135 of *LNCS*, 2015.

6 A. Amarilli, P. Bourhis, and P. Senellart. Tractable lineages on treelike instances: Limits and extensions. In *PODS*, 2016.

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

8 V. Bárány, B. ten Cate, and L. Segoufin. Guarded negation. *J. ACM*, 62(3), 2015.

9 P. Barceló. Querying graph databases. In *PODS*, 2013.

10 P. Barceló, M. Romero, and M. Y. Vardi. Does query evaluation tractability help query containment? In *PODS*, 2014.

11 M. Benedikt, P. Bourhis, and P. Senellart. Monadic datalog containment. In *ICALP*, 2012.

12 M. Benedikt, P. Bourhis, and M. Vanden Boom. A step up in expressiveness of decidable fixpoint logics. In *LICS*, 2016.

13 M. Benedikt and G. Gottlob. The impact of virtual views on containment. *PVLDB*, 3(1-2), 2010.

14 M. Benedikt, B. ten Cate, and M. Vanden Boom. Effective interpolation and preservation in guarded logics. In *LICS*, 2014.

15 A. Berry, R. Pogorelcnik, and G. Simonet. An introduction to clique minimal separator decomposition. *Algorithms*, 3(2), 2010.

16 D. Berwanger and E. Grädel. Games and model checking for guarded logics. In *LPAR*, 2001.

17 J.-C. Birget. State-complexity of finite-state devices, state compressibility and incompressibility. *Mathematical systems theory*, 26(3), 1993.

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

19 T. Cachat. Two-way tree automata solving pushdown games. In *Automata logics, and infinite games*, chapter 17. Springer, 2002.

20 D. Calvanese, G. De Giacomo, M. Lenzeniri, and M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In *KR*, 2000.

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

22 H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata: Techniques and applications, 2007. Available from `http://tata.gforge.inria.fr/`.

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

24 N. Dalvi and D. Suciu. Management of probabilistic data: foundations and challenges. In *PODS*, 2007.

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

26 R. Diestel. Simplicial decompositions of graphs: A survey of applications. *Discrete Math.*, 75(1), 1989.

27 R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM*, 30(3), 1983.

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

29 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

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

31 G. Gottlob, E. Grädel, and H. Veith. Datalog LITE: A deductive query language with linear time model checking. *ACM Trans. Comput. Log.*, 3(1), 2002.

32 G. Gottlob, G. Greco, and F. Scarcello. Treewidth and hypertree width. In L. Bordeaux, Y. Hamadi, and P. Kohli, editors, *Tractability: Practical Approaches to Hard Problems*, chapter 1. Cambridge University Press, 2014.

33 G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. *JCSS*, 64(3), 2002.

34 G. Gottlob, N. Leone, and F. Scarcello. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. *JCSS*, 66(4), 2003.

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

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

37 M. Grohe and D. Marx. Constraint solving via fractional edge covers. *TALG*, 11(1), 2014.

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

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

40 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.

41 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.

42 H.-G. Leimer. Optimal decomposition by clique separators. *Discrete Math.*, 113(1-3), 1993.

43 D. Leinders, M. Marx, J. Tyszkiewicz, and J. V. den Bussche. The semijoin algebra and the guarded fragment. *Journal of Logic, Language and Information*, 14(3), 2005.

44 S. Malik. Analysis of cyclic combinational circuits. In *ICCAD*, 1993.

45 S. Maniu, R. Cheng, and P. Senellart. ProbTree: A query-efficient representation of probabilistic graphs. In *BUDA*, June 2014. Workshop without formal proceedings.

46 D. Marx. Can you beat treewidth? *Theory of Computing*, 6(1), 2010.

47 A. O. Mendelzon and P. T. Wood. Finding regular simple paths in graph databases. In *VLDB*, 1989.

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

49 M. Monet. Probabilistic evaluation of expressive queries on bounded-treewidth instances. In *SIGMOD/PODS PhD Symposium*, June 2016.

50 M. D. Riedel and J. Bruck. Cyclic Boolean circuits. *Discrete Applied Mathematics*, 160(13-14), 2012.

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

52 R. E. Tarjan. Decomposition by clique separators. *Discrete Math.*, 55(2), 1985.

53 A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5, 1955.

54 M. Y. Vardi. The complexity of relational query languages. In *STOC*, 1982.
55 M. Y. Vardi. On the complexity of bounded-variable queries. In *PODS*, pages 266–276, 1995.
56 M. Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, 1981.

A Proofs for Section 4 (Conjunctive Queries on Treelike Instances)

▶ **Proposition 3.** *There is an arity-two signature σ and an infinite family $Q_1, Q_2, \ldots$ of α-acyclic CQs such that, for any $i \in \mathbb{N}$, any bNTA that tests Q_i for treewidth 1 must have $\Omega(2^{|Q_i|^{1-\varepsilon}})$ states for any $\varepsilon > 0$.*

Proof. We fix the signature σ to consist of binary relations S, S_0, S_1, and C. We will code binary numbers as gadgets on this fixed signature. The coding of $i \in \mathbb{N}$ at length k, with $k \geqslant 1 + \lceil \log_2 i \rceil$, consists of an S-chain $S(a_1, a_2), \ldots, S(a_{k-1}, a_k)$, and facts $S_{b_j}(a_{j+1}, a'_{j+1})$ for $1 \leqslant j \leqslant k-1$ where a'_{j+1} is a fresh element and b_j is the j-th bit in the binary expression of i (padding the most significant bits with 0). We now define the query family Q_i: each Q_i is formed by picking a root variable x and gluing 2^i chains to x; for $0 \leqslant j \leqslant 2^i - 1$, we have one chain that is the concatenation of a chain of C of length i and the length-$(i+1)$ coding of j using a gadget. Clearly the size of Q_i is $O(i \times 2^i)$ and thus $2^i = \Omega(|Q_i|^{1-\varepsilon/2})$.

Fix $i > 0$. Let A be a bNTA testing Q_i on instances of treewidth 1. We will show that A must have at least $\binom{2^i}{2^{i-1}} = \Omega\left(2^{2^i - \frac{i}{2}}\right)$ states (the lower bound is obtained from Stirling's formula), from which the claim follows. In fact, we will consider a specific subset $\mathcal{I}$ of the instances of treewidth $\leqslant 1$, and a specific set $\mathcal{E}$ of tree encodings of instances of $\mathcal{I}$, and show the claim on $\mathcal{E}$, which suffices to conclude.

To define $\mathcal{I}$, let $\mathcal{S}_i$ be the set of subsets of $\{0, \ldots, 2^i - 1\}$ of cardinality 2^{i-1}, so that $|\mathcal{S}_i|$ is $\binom{2^i}{2^{i-1}}$. We will first define a family $\mathcal{I}'$ of instances indexed by $\mathcal{S}_i$ as follows. Given $S \in \mathcal{S}_i$, the instance I'_S of $\mathcal{I}'$ is obtained by constructing a full binary tree of the C-relation of height $i-1$, and identifying, for all j, the j-th leaf node with element a_1 of the length-$(i+1)$ coding of the j-th smallest number in S. We now define the instances of $\mathcal{I}$ to consist of a root element with two C-children, each of which are the root element of an instance of $\mathcal{I}'$ (we call the two the *child instances*). It is clear that instances of $\mathcal{I}$ have treewidth 1, and we can check quite easily that an instance of $\mathcal{I}$ satisfies Q_i iff the child instances I'_{S_1} and I'_{S_2} are such that $S_1 \cup S_2 = \{1, \ldots, 2^i\}$.

We now define $\mathcal{E}$ to be tree encodings of instances of $\mathcal{I}$: refer to Appendix C.1 for details of how they are defined. First, define $\mathcal{E}'$ to consist of tree encodings of instances of $\mathcal{I}'$, which we will also index with $\mathcal{S}_i$, i.e., E_S is a tree encoding of I'_S. We now define $\mathcal{E}$ as the tree encodings E constructed as follows: given an instance $I \in \mathcal{I}$, we encode it as a root bag with domain $\{r\}$, where r is the root of the tree I, and no fact, the first child n_1 of the root bag having domain $\{r, r_1\}$ and fact $C(r, r_1)$, the second child n_2 of the root being defined in the same way. Now, n_1 has one dummy child with empty domain and no fact, and one child which is the root of some tree encoding in $\mathcal{E}$ of one child instance of I. We define n_2 analogously with the other child instance.

For each $S \in \mathcal{S}_i$, letting $\bar{S}$ be the complement of S relative to $\{0, \ldots, 2^i - 1\}$, we call $I_S \in \mathcal{I}$ the instance where the first child instance is I'_S and the second child instance is $I'_{\bar{S}}$, and we call $E_S \in \mathcal{E}$ the tree encoding of I_S according to the definition above. We then call $\mathcal{Q}_S$ the set of states q of A such that there exists a run of A on E_S where the root of the encoding of the first child instance is mapped to q. As each I_S satisfies Q, each E_S should be accepted by the automaton, so each $\mathcal{Q}_S$ is non-empty.

Further, we show that the $\mathcal{Q}_S$ are pairwise disjoint: for any $S_1 \neq S_2$ of $\mathcal{S}_i$, we show that $\mathcal{Q}_{S_1} \cap \mathcal{Q}_{S_2} = \emptyset$. Assume to the contrary the existence of q in the intersection, and let ρ_{S_1} and ρ_{S_2} be runs of A respectively on I_{S_1} and I_{S_2} that witness respectively that $q \in \mathcal{Q}_{S_1}$ and $q \in \mathcal{Q}_{S_2}$. Now, consider the instance $I \in \mathcal{I}$ where the first child instance is I_1, and the second child instance is $\bar{I}_2$, and let $E \in \mathcal{E}$ be the tree encoding of I. We can construct a run ρ of A on E by defining ρ according to ρ_{S_2} except that, on the subtree of E rooted at

the root r' of the tree encoding of the first child instance, ρ is defined according to ρ_{S_1}: this is possible because ρ_{S_1} and ρ_{S_2} agree on r'_1 as they both map r' to q. Hence, ρ witnesses that A accepts E. Yet, as $I_1 \neq I_2$, we know that I does not satisfy Q, so that, letting $E \in \mathcal{E}$ be its tree encoding, A rejects E. We have reached a contradiction, so indeed the $\mathcal{Q}_S$ are pairwise disjoint.

As the $\mathcal{Q}_S$ are non-empty, we can construct an mapping from $\mathcal{S}_i$ to the state set of A by mapping each $S \in \mathcal{S}_i$ to some state of $\mathcal{Q}_S$: as the $\mathcal{Q}_S$ are pairwise disjoint, this mapping is injective. We deduce that the state set of A has size at least $|\mathcal{S}_i|$, which concludes from the bound on the size of $\mathcal{S}_i$ that we showed previously. ◀

▶ **Proposition 4.** *For any treewidth bound $k_{\mathrm{I}} \in \mathbb{N}$, given an α-acyclic CQ Q, we can compute in FPT-linear time in $O(|Q|)$ (parameterized by k_{I}) an alternating two-way tree automaton that tests it for treewidth k_{I}.*

Hence, if we are additionally given a relational instance I of treewidth $\leqslant k_{\mathrm{I}}$, one can determine whether $I \models Q$ in FPT-linear time in $|I| \cdot |Q|$ (parameterized by k_{I}).

Proof. Given the α-acyclic CQ Q, we can compute in linear time in Q a chordal decomposition T (equivalently, a join tree) of Q (Theorem 5.6 of [FFG02], attributed to [TY84]). As T is in particular a simplicial decomposition of Q of width $\leqslant \mathrm{arity}(\sigma) - 1$, i.e., of constant width, we use Proposition 12 to compile in linear time in $|Q|$ an ICG-Datalog program P with body size bounded by a constant k_{P}.

We now use Theorem 22 to construct, in FPT-linear time in $|P|$ (hence, in $|Q|$), parameterized by k_{I} and the constant k_{P}, a SATWA A testing P for treewidth k_{I}.

We now observe that, thanks to the fact that Q is monotone, the SATWA A does not actually feature any negation: the translation in the proof of Proposition 12 does not produce any negated atom, and the compilation in the proof of Theorem 22 only produces a negated state within a Boolean formula when there is a corresponding negated atom in the Datalog program. Hence, A is actually an alternating two-way tree automaton, which proves the first part of the claim.

For the second part of the claim, we use Theorem 11 to evaluate P on I in FPT-linear time in $|I| \cdot |P|$, parameterized by the constant k_{P} and k_{I}. This proves the claim. ◀

▶ **Theorem 5.** *There is an arity-two signature σ for which there is no algorithm $\mathcal{A}$ with exponential running time and polynomial output size for the following task: given a conjunctive query Q of treewidth $\leqslant 2$, produce an alternating two-way tree automaton A_Q on Γ_σ^5-trees that tests Q on σ-instances of treewidth $\leqslant 5$.*

To prove this theorem, we need some notions and lemmas from [BBGS16], an extended version of [BBS12]. Since [BBGS16] is currently unpublished, we provide in Appendix F the complete relevant material from this paper, in particular Lemma 68, Theorem 70, and their proofs.

Proof of Theorem 5. Let σ be $\mathcal{S}^{\mathsf{Bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child},\mathsf{Child}^?}$ as in Theorem 70. We pose $c = 3$, $k_{\mathrm{I}} = 2 \times 3 - 1 = 5$. Assume by way of contradiction that there exists an algorithm $\mathcal{A}$ satisfying the prescribed properties. We will describe an algorithm to solve any instance of the containment problem of Theorem 70 in singly exponential time. As Theorem 70 states that it is 2EXPTIME-hard, this yields a contradiction by the time hierarchy theorem.

Let P and Q be an instance of the containment problem of Theorem 70, where P is a monadic Datalog program of var-size $\leqslant 3$, and Q is a CQ of treewidth $\leqslant 2$. We will show how to solve the containment problem, that is, decide whether there exists some instance I satisfying $P \wedge \neg Q$.

Using Lemma 68, compute in singly exponential time the $\Gamma_\sigma^{k_\mathrm{I}}$-bNTA A_P. Using the putative algorithm $\mathcal{A}$ on Q, compute in singly exponential time an alternating two-way automaton A_Q of polynomial size. As A_P describes a family $\mathcal{I}$ of canonical instances for P, there is an instance satisfying $P \wedge \neg Q$ iff there is an instance in $\mathcal{I}$ satisfying $P \wedge \neg Q$. Now, as $\mathcal{I}$ is described as the decodings of the language of A_P, all instances in $\mathcal{I}$ have treewidth $\leqslant k_\mathrm{I}$. Furthermore, the instances in $\mathcal{I}$ satisfy P by definition of $\mathcal{I}$. Hence, there is an instance satisfying $P \wedge \neg Q$ iff there is an encoding E in the language of A_P whose decoding satisfies $\neg Q$. Now, as A_Q tests Q on instances of treewidth k_I, this is the case iff there is an encoding E in the language of A_P which is not accepted by A_Q. Hence, our problem is equivalent to the problem of deciding whether there is a tree accepted by A_P but not by A_Q.

We now use Theorem A.1 of [CGKV88] to compute in EXPTIME in A_Q a bNTA A'_Q recognizing the complement of the language of A_Q. Remember that A_Q was computed in EXPTIME and is of polynomial size, so the entire process so far is EXPTIME. Now we know that we can solve the containment problem by testing whether A_P and A'_Q have non-trivial intersection, which can be done in PTIME by computing the product automaton and testing emptiness [CDG+07]. This solves the containment problem in EXPTIME. As we explained initially, we have reached a contradiction, because it is 2EXPTIME-hard. ◀

▶ **Theorem 8.** *For any $k_\mathrm{I}, k_\mathrm{Q} \in \mathbb{N}$, given a CQ Q and a simplicial decomposition T of simplicial width k_Q of Q, we can compute in FPT-linear in $|Q|$ (parameterized by k_I and k_Q) an alternating two-way tree automaton that tests Q for treewidth k_I.*

Hence, if we are additionally given a relational instance I of treewidth $\leqslant k_\mathrm{I}$, one can determine whether $I \models Q$ in FPT-linear time in $|I| \cdot (|Q| + |T|)$ (parameterized by k_I and k_Q).

Proof. We use Proposition 12 to compile the CQ Q to an ICG-Datalog program P with body size at most $k_\mathrm{P} := f_\sigma(k_\mathrm{Q})$, in FPT-linear time in $|Q| + |T|$ parameterized by k_Q.

We now use Theorem 22 to construct, in FPT-linear time in $|P|$ (hence, in $|Q|$), parameterized by k_I and k_P, hence in k_I and k_Q, a SATWA A testing P for treewidth k_I. For the same reasons as in the proof of Proposition 4, it is actually an two-way alternating tree automaton, so we have shown the first part of the result.

To prove the second part of the result, we now use Theorem 11 to evaluate P on I in FPT-linear time in $|I| \cdot |P|$, parameterized by k_P and k_I, hence again by k_Q and k_I. This proves the claim. ◀

B Proofs for Section 5 (ICG-Datalog on Treelike Instances)

▶ **Proposition 9.** *There is a signature σ and Datalog program P such that the language of Γ_σ^1-trees that encode instances satisfying P is not a regular tree language.*

We will use in this proof the notion of tree encoding and the associated concepts, which can be found in Appendix C.1.

Proof. Let σ be the signature containing two binary relations Y and Z and two unary relations Begin and End. Consider the following program P:

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

Let L be the language of the tree encodings of instances of treewidth 1 that satisfy P. We will show that L is not a regular tree language, which clearly implies the second claim, as a

bNTA or an alternating two-way tree automaton can only recognize regular tree languages [CDG+07]. To show this, let us assume by contradiction that L is a regular tree language, so that there exists a Γ^1_σ-bNTA A that accepts L, i.e., 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 end is the only node labeled End. This condition on instances can clearly be expressed in MSO, so that by Theorem 2 there exists a bNTA A_{chain} on Γ^1_σ that tests this property. In particular, we can build the bNTA A' which is the intersection of A and A_{chain}, which tests whether instances are of the prescribed form and are accepted by the program P.

We now observe that such instances must be the instance

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

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 chain in I of the form I_k, which by the restrictions on the shape of I means that $I = I_k$.

We further restrict our attention to tree encodings that form a single branch of a specific form, namely, their contents are as follows (given from leaf to root) for some integer $n \geqslant 0$: $(\{a_1\}, \text{Begin}(a_1))$, $(\{a_1, a_2\}, X(a_1, a_2))$, $(\{a_2, a_3\}, X(a_2, a_3))$, $(\{a_3, a_1\}, X(a_3, a_1))$, $\ldots$, $(\{a_n, a_{n+1}\}, X(a_n, a_{n+1}))$, $(\{a_{n+1}\}, \text{End}(a_{n+1}))$, where we write X to mean that we may match either Y or Z, where addition is modulo 3, and where we add dummy nodes $(\bot, \bot)$ as left children of all nodes, and as right children of the leaf node $(\{a_1\}, \text{Begin}(a_1))$, to ensure that the tree is full. It is clear that we can design a bNTA A_{encode} which recognizes tree encodings of this form, and we define A'' to be the intersection of A' and A_{encode}. In other words, A'' further enforces that the Γ^1_σ-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.

Now, 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 a bNTA which is forced to operate on chains (completed to full binary trees by a specific addition of binary nodes). Thus, 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 $\delta(\iota((\bot, \bot)), \iota((\bot, \bot)))$ and transition function $\delta(q, x) = \delta(\iota((\bot, \bot)), q, (s, f))$ 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'', the automaton A''' on words 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, which establishes the desired result. ◀

▶ **Theorem 11.** *Given an ICG-Datalog program P of body size k_{P} and a relational instance I of treewidth k_{I}, checking if $I \models P$ is FPT-linear time in $|I| \cdot |P|$ (parameterized by k_{P} and k_{I}).*

Proof. Anticipating on the results of later sections, we use Theorem 24 to compute a representation C of the provenance of P on I as a stratified cycluit, in FPT-linear time

(parameterized by k_{P} and k_{I}). We let ν be the valuation of C that sets every input to true (reflecting that all the facts of I are indeed there), and we then use Proposition 30 to evaluate $\nu(C)$ in linear time. By definition of the provenance, we have $\nu(C) = 1$ iff $I \models P$, which concludes the proof. ◀

▶ **Proposition 12.** *There is a function f_σ (depending only on σ) such that for all $k \in \mathbb{N}$, for any conjunctive query Q and simplicial tree decomposition T of Q of width at most k, we can compute in $O(|Q| + |T|)$ an equivalent ICG-Datalog program with body size at most $f_\sigma(k)$.*

We first prove the following lemma about simplicial tree decompositions:

▶ **Lemma 38.** *For any simplicial decomposition T of width k of a query Q, we can compute in linear time a simplicial decomposition T_{bounded} of Q such that each bag has degree at most 2^{k+1}.*

Proof. Fix Q and T. We construct the simplicial decomposition T_{bounded} of Q in a process which shares some similarity with the routine rewriting of tree decompositions to make them binary, by creating copies of bags. However, the process is more intricate because we need to preserve the fact that we have a *simplicial* tree decomposition, where interfaces are guarded.

We go over T bottom-up: for each bag b of T, we create a bag b' of T_{bounded} with same domain as b. Now, we partition the children of b depending on their intersection with b: for every subset S of the domain of b such that b has some children whose intersection with b is equal to S, we write these children $b_{S,1}, \ldots, b_{S,n_S}$ (so we have $S = \mathrm{dom}(b) \cap \mathrm{dom}(b_{S,j})$ for all $1 \leqslant j \leqslant n_S$), and we write $b'_{S,1}, \ldots, b'_{S,n_S}$ for the copies that we already created for these bags in T_{bounded}. Now, for each S, we create n_S fresh bags $b'_{=S,j}$ in T_{bounded} (for $1 \leqslant j \leqslant n_S$) with domain equal to S, and we set $b'_{=S,1}$ to be a child of b', $b'_{=S,j+1}$ to be a child of $b'_{=S,j}$ for all $1 \leqslant j < n_S$, and we set each $b'_{S,i}$ to be a child of $b'_{=S,i}$.

This process can clearly be performed in linear time. Now, the degree of the fresh bags in T_{bounded} is at most 2, and the degree of the copies of the original bags is at most 2^{k+1}, as stated. Further, it is clear that the result is still a tree decomposition (each fact is still covered, the occurrences of each element still form a connected subtree because they are as in T with the addition of some paths of the fresh bags), and the interfaces in T_{bounded} are the same as in T, so they still satisfy the requirement of simplicial decompositions. ◀

We can now prove Proposition 12. In fact, as will be easy to notice from the proof, our construction further ensures that the equivalent ICG-Datalog program is positive, nonrecursive, and conjunctive.

Proof of Proposition 12. Using Lemma 38, we can start by rewriting in linear time the input simplicial decomposition to ensure that each bag has degree at most 2^{k+1}. Hence, let us assume without loss of generality that T has this property. We further add an empty root bag if necessary to ensure that the root bag of T is empty and has exactly one child.

We start by using Lemma 3.1 of [FFG02] to annotate in linear time each node b of T by the set of atoms $\mathcal{A}_b$ of Q whose free variables are in the domain of b and such that for each atom A of $\mathcal{A}_b$, b is the topmost bag of T which contains all the variables of A. As the signature σ is fixed, note that we have $|\mathcal{A}_b| \leqslant g_\sigma(k)$ for some function g_σ depending only on σ.

Once this is done, we *scrub* T, namely, we remove variables from bags of T to ensure that, for each bag b, for each $x \in \mathrm{dom}(b)$, either $\mathcal{A}_b$ contains an atom where x appears, or there is a child of b where x appears. We can do so in linear time by traversing T bottom-up, keeping track of which variables must be kept, and removing all other variables. Intuitively,

this scrubbing transformation will ensure that, when constructing our ICG-Datalog program, all variables in the head of a rule also appear in the body of this rule.

We now perform a process similar to Lemma 3.1 of [FFG02]. We start by precomputing in linear time a mapping μ that associates, to each pair $\{x, y\}$ of variables of Q, the set of all atoms in Q where $\{x, y\}$ co-occur. We can compute μ in linear time by processing all atoms of Q and adding each atom as an image of μ for each pair of variables that it contains (remember that the arity of σ is constant). Now, we do the following computation: for each bag b which is not the root of T, letting S be its interface with its parent bag, we annotate b by a set of atoms $\mathcal{A}_b^{\text{guard}}$ defined as follows: for all $x, y \in S$ with $x \neq y$, letting $A(\mathbf{z})$ be an atom of Q where x and y appear (which must exist, by the requirement on simplicial decompositions, and which we retrieve from μ), we add $A(\mathbf{w})$ to $\mathcal{A}_b^{\text{guard}}$, where, for $1 \leqslant i \leqslant |\mathbf{z}|$, we set $w_i := z_i$ if $z_i \in \{x, y\}$, and w_i to be a fresh variable otherwise. In other words, $\mathcal{A}_b^{\text{guard}}$ is a set of atoms that ensures that the interface S of b with its parent is covered by a clique, and we construct it by picking atoms of Q that witness the fact that it is guarded (which it is, because T is a simplicial decomposition), and replacing their irrelevant variables to be fresh. Note that $\mathcal{A}_b^{\text{guard}}$ consists of at most $k \times (k+1)/2$ atoms, but the domain of these atoms is not a subset of $\text{dom}(b)$ (because they include fresh variables). This entire computation is performed in linear time.

We now define the function $f_\sigma(k)$ as follows, remembering that $\text{arity}(\sigma)$ denotes the arity of the *extensional* signature:

$$f_\sigma(k) := \max(\text{arity}(\sigma), k+1) \times \left(g_\sigma(k) + 2^{k+1}(k(k+1)/2+1)\right).$$

We now build our ICG-Datalog program P of body size $f_\sigma(k)$ which is equivalent to Q. We define the intensional signature σ_{int} by creating one intensional predicate P_b for each non-root bag b of T, whose arity is the size of the intersection of b with its parent. As we ensured that the root bag b_{r} of T is empty and has exactly one child b'_{r}, we use $P_{b'_{\text{r}}}$ as our 0-ary Goal() predicate (because its interface with its parent b_{r} is necessarily empty). We now define the rules of P by processing T bottom-up: for each bag b of T, we add one rule ρ_b with head $P_b(\mathbf{x})$, defined as follows:

- If b is a leaf, then ρ_b is $P_b \leftarrow \bigwedge \mathcal{A}_b$.
- If b is an internal node with children $b_1, \ldots, b_m$ (remember that $m \leqslant 2^{k+1}$), then ρ_b is $P_b \leftarrow \bigwedge \mathcal{A}_b \wedge \bigwedge_{1 \leqslant i \leqslant m} (\bigwedge \mathcal{A}_{b_i}^{\text{guard}} \wedge P_{b_i})$.

We first check that P is intensional-clique-guarded, but this is the case because the only use of intensional atoms in rules is the P_{b_i}, whose free variables are the intersection S of b and b_i, but by construction the conjunction of atoms $\bigwedge \mathcal{A}_{b_i}^{\text{guard}}$ is a suitable guard for S: for each $\{x, y\} \in S$, it contains an atom where both x and y occur).

Second, we check that, pursuant to the definition of Datalog, each head variable occurs in the body, but this is the case thanks to the fact that T is scrubbed: any variable of an intensional predicate P_b is a variable of $\text{dom}(b)$, which must occur either in $\mathcal{A}_b$ or in the intersection of b with one of its children b_i.

Third, we check that the body size of P is indeed $f_\sigma(k)$. It is clear that $\text{arity}(\sigma_{\text{int}}) \leqslant k+1$, so that $\text{arity}(P) \leqslant \max(\text{arity}(\sigma), k+1)$. Further, the maximal number of atoms in the body of a rule is $g_\sigma(k) + 2^{k+1}(k(k+1)/2+1))$, so we obtain the desired bound.

What is left to check is that P is equivalent to Q. It will be helpful to reason about P by seeing it as the conjunctive query Q' obtained by recursively inlining the definition of rules: observe that this a conjunctive query, because P is conjunctive, i.e., for each intensional

atom P_b, the rule ρ_b is the only one where P_b occurs as head atom. It is clear that P and Q' are equivalent, so we must prove that Q and Q' are equivalent.

For the forward direction, it is obvious that $Q' \implies Q$, because Q' contains every atom of Q by construction of the $\mathcal{A}_b$. For the backward direction, noting that the only atoms of Q' that are not in Q are those added in the sets $\mathcal{A}_b^{\text{guard}}$, we observe that there is a homomorphism from Q' to Q defined by mapping each atom $A(\mathbf{w})$ occurring in some $\mathcal{A}_b^{\text{guard}}$ to the atom $A(\mathbf{z})$ of Q used to create it; this mapping is the identity on the two variables x and y used to create $A(\mathbf{w})$, and maps each fresh variables w_i to z_i: the fact that these variables are fresh ensures that this homomorphism is well-defined. This shows Q and Q', hence P, to be equivalent, which concludes the proof. ◀

▶ **Proposition 13.** *For any positive, conjunctive, nonrecursive ICG-Datalog program P with body size k, there is a CQ Q of simplicial width $\leqslant k$ that is equivalent to P.*

To prove Proposition 13, we will use the notion of the *call graph* of a Datalog program. This is the graph G on the relations of σ_{int} which has an edge from R to S whenever a rule contains relation R in its head and S in its body. From the requirement that P is nonrecursive, we know that this graph G is a DAG.

We now prove Proposition 13:

Proof of 13. We first check that every intensional relation reachable from Goal in the call graph G of P appears in the head of a rule of P (as P is conjunctive, this rule is then unique). Otherwise, it is clear that P is not satisfiable (it has no derivation tree), so we can simply rewrite P to the query False. We also assume without loss of generality that each intensional relation except Goal() occurs in the body of some rule, as otherwise we can simply drop them and all rules where they appear as the head relation.

In the rest of the proof we will consider the rules of P in some order, and create an equivalent ICG-Datalog program P' with rules $r'_0, \ldots, r'_{m'}$. We will ensure that P' is also positive, conjunctive, and nonrecursive, and that it further satisfies the following additional properties:

1. Every intensional relation other than Goal appears in the body of exactly one rule of P', and appears there exactly once;
2. For every $0 \leqslant i \leqslant m$, for every variable z in the body of rule r'_i that does not occur in its head, then for every $0 \leqslant j < i$, z does not occur in r'_j.

We initialize a queue that contains only the one rule that defines Goal in P, and we do the following until the queue is empty:

- Pop a rule r from the queue. Let r' be defined from r as follows: for every intensional relation R that occurs in the body of r', letting $R(\mathbf{x^1}), \ldots, R(\mathbf{x^n})$ be its occurrences, rewrite these atoms to $R^1(\mathbf{x^1}), \ldots, R^n(\mathbf{x^n})$, where the R^i are *fresh* intensional relations.
- Add r' to P'.
- For each intensional atom $R^i(\mathbf{x})$ of r', letting R be the relation from which R^i was created, let r_R be the rule of P that has R in its head (by our initial considerations, there is one such rule, and as the program is conjunctive there is exactly one such rule). Define r'_{R^i} from r_R by replacing its head relation from R to R^i, and renaming its head and body variables such that the head is exactly $R^i(\mathbf{x})$. Further rename all variables that occur in the body but not in the head, to replace them by fresh new variables. Add r'_{R^i} to the queue.

We first argue that this process terminates. Indeed, considering the graph G, whenever we pop from the queue a rule with head relation R (or a fresh relation created from a relation R), we add to the queue a finite number of rules for head relations created from relations R' such that the edge (R, R') is in the graph G. The fact that G is acyclic ensures that the process terminates (but note that its running time may generally be exponential in the input). Second, we observe that, by construction, P satisfies the first property, because each occurrence of an intensional relation in a body of P' is fresh, and satisfies the second property, because each variable which is the body of a rule but not in its head is fresh, so it cannot occur in a previous rule

Last, we verify that P and P' are equivalent, but this is immediate, because any derivation tree for P can be rewritten to a derivation tree for P' (by renaming relations and variables), and vice-versa.

We define Q to be the conjunction of all extensional atoms occurring in P'. To show that it is equivalent to P', the fact that Q implies P' is immediate as the leaves are sufficient to construct a derivation tree, and the fact that P' implies Q is because, letting G' be the call graph of P', by the first property of P' we can easily observe that it is a tree, so the structure of derivation trees of G' also corresponds to P, and by the second property of P' we know that two variables are equal in two extensional atoms iff they have to be equal in any derivation tree. Hence, P' and Q are indeed equivalent.

We now justify that Q has simplicial width at most k. We do so by building from P' a simplicial decomposition T of Q of width $\leqslant k$. The structure of T is the same as G' (which is actually a tree). For each bag b of T corresponding to a node of G' standing for a rule r of P', we set the domain of b to be the variables occurring in r. It is clear that T is a tree decomposition of Q, because each atom of Q is covered by a bag of T (namely, the one for the rule whose body contained that atom) and the occurrences of each variable form a connected subtree (whose root is the node of G' standing for the rule where it was introduced, using the second condition of P'). Further, T is a simplicial decomposition because P' is intensional-clique-guarded; further, from the second condition, the variables shared between one bag and its child are precisely the head variables of the child rule. The width is $\leqslant k$ because the body size of an ICG-Datalog program is an upper bound on the maximal number of variables in a rule body. ◀

▶ **Proposition 14.** *There is a signature σ and a family $(P_n)_{n\in\mathbb{N}}$ of ICG-Datalog programs with body size at most 9 which are positive, conjunctive, and nonrecursive, such that $|P_n| = O(n)$ and any CQ Q_n equivalent to P_n has size $\Omega(2^n)$.*

To prove Proposition 14, we will introduce the following notion:

▶ **Definition 39.** A *match* of a CQ Q in an instance I is a subset M of facts of I which is an image of a homomorphism from the canonical instance of Q to I, i.e., M witnesses that $I \models Q$, in particular $M \models Q$ as a subset of I.

Our proof will rely on the following elementary observation:

▶ **Lemma 40.** *If a CQ Q has a match M in an instance I, then necessarily $|Q| \geqslant |M|$.*

Proof. As M is the image of Q by a homomorphism, it cannot have more atoms than M has facts. ◀

We are now ready to prove Proposition 14:

Proof of Proposition 14. Fix σ to contain a binary relation R and a ternary relation G. Consider the rule $\rho_0 : R_0(x, y) \leftarrow R(x, y)$ and define the following rules, for all $i > 0$:

$$\rho_i : R_i(x, y) \leftarrow G(x, z, y), R_{i-1}(x, z), R_{i-1}(z, y)$$

For each $i > 0$, we let P_i consist of the rules ρ_j for $1 \leqslant j \leqslant i$, as well as ρ_0 and the rule $\mathrm{Goal}() \leftarrow R_i(x, y)$. It is clear that each P_i is positive, conjunctive, and non-recursive; further, the predicate G ensures that it is an ICG-Datalog program. The arity is 3 and the maximum number of atoms is the body is 3, so the body size is indeed 9.

We first prove by an immediate induction that, for each $i \geqslant 0$, considering the rules of P_i and the intensional predicate R_i, whenever an instance I satisfies $R_i(a, b)$ for two elements $a, b \in \mathrm{dom}(I)$ then there is an R-path of length 2^i from a to b. Now, fixing $i \geqslant 0$, this clearly implies there is an instance I_i of size (number of facts) $\geqslant 2^i$, namely, an R-path of this length with the right set of additional G-facts, such that $I_i \models P_i$ but any strict subset of I_i does not satisfy P_i.

Now, let us consider a CQ Q_i which is equivalent to P_i, and let us show the desired size bound. By equivalence, we know that $I_i \models Q_i$, hence Q_i has a match M_i in I_i, but any strict subset of I_i does not satisfy Q_i, which implies that, necessarily, $M_i = I_i$ (indeed, otherwise M_i would survive as a match in some strict subset of I_i). Now, by Lemma 40, we deduce that $|Q_i| \geqslant |M_i|$, and as $|M_i| = |I_i| \geqslant 2^i$, we obtain the desired size bound, which concludes the proof. ◀

▶ **Proposition 15.** *There is a function f_σ (depending only on σ) such that, for any weak GN-normal form GNF query Q of CQ-rank r, we can compute in time $O(|Q|)$ an equivalent nonrecursive ICG-Datalog program P of body size $f_\sigma(r)$.*

Proof. We recall from [BtCV14], Appendix B.1, that a weak GN-normal form formulae is a φ-formula in the inductive definition below:

- A disjunction of existentially quantified conjunctions of ψ-formulae is a φ-formula;
- An atom is a ψ-formula;
- The conjunction of a φ-formula and of a guard is a ψ-formula;
- The conjunction of the negation of a φ-formula and of a guard is a ψ-formula.

We define $f_\sigma : n \mapsto \mathrm{arity}(\sigma) \times n$.

We consider an input Boolean GN-normal form formula Q of CQ-rank r, and call T its abstract syntax tree. We rewrite T in linear time to inline in φ-formulae the definition of their ψ-formulae, so all nodes of T consist of φ-formulae, in which all subformulae are guarded (but they can be used positively or negatively).

We now process T bottom-up. We introduce one intensional Datalog predicate R_n per node n in T: its arity is the number of variables that are free at n. We then introduce one rule $\rho_{n,\delta}$ for each disjunct δ of the disjunction that defines n in T: the head of $\rho_{n,\delta}$ is an R_n-atom whose free variables are the variables that are free in n, and the body of $\rho_{n,\delta}$ is the conjunction that defines δ, with each subformula replaced by the intensional relation that codes it. Of course, we use the predicate R_r for the root r of T as our goal predicate; note that it must be 0-ary, as Q is Boolean so there are no free variables at the root of T. This process defines our ICG-Datalog program P: it is clear that this process runs in linear time.

We first observe that body size for an intensional predicate R_n is less than the CQ-rank of the corresponding subformula: recall that the *CQ-rank* is the overall number of conjuncts occurring in the disjunction of existentially quantified conjunctions that defines this subformula. Hence, as the arity of σ is bounded, clearly P has body size $\leqslant f_\sigma(r)$. We

next observe that intentional predicates in the bodies of rules of P are always guarded, thanks to the guardedness requirement on Q. Further, it is obvious that P is nonrecursive, as it is computed from the abstract syntax tree T. Last, it is clear that P is equivalent to the original formula Q, as we can obtain Q back simply by inlining the definition of the intensional predicates. ◀

▶ **Proposition 16.** *The combined complexity of monadic Datalog query evaluation on bounded-treewidth instances is FPT when parameterized by instance treewidth and body size (as in Definition 10) of the monadic Datalog program.*

Proof. This is simply by observing that any monadic Datalog program is an ICG-Datalog program with the same body size, so we can simply apply Theorem 11. ◀

▶ **Proposition 19.** *Given a Boolean SAC2RPQ Q, we can compute in time $O(|Q|)$ an equivalent ICG-Datalog program P of body size 4.*

Proof. We first show the result for 2RPQs, and then explain how to extend it to SAC2RPQs.

We have not specified how RPQs are provided as input. We assume that they are provided as a regular expression, from which we can use Thompson's construction [ALSU06] to compute in linear time an equivalent NFA A (with ε-transitions) on the alphabet $\Sigma^{\pm}$. Note that the result of Thompson's construction has exactly one final state, so we may assume that each automaton has exactly one final state.

We now define the intensional signature of the ICG-Datalog program to consist of one unary predicate P_q for each state q of the automaton, in addition to Goal(). We add the rule $\text{Goal}() \leftarrow P_{q_\text{f}}(x)$ for the final state q_f, and for each extensional relation $R(x, y)$, we add the rules $P_{q_0}(x) \leftarrow R(x, y)$ and $P_{q_0}(y) \leftarrow R(x, y)$, where q_0 is the initial state. We then add rules corresponding to automaton transitions:

- for each transition from q to q' labeled with a letter R, we add the rule $P_{q'}(y) \leftarrow P_q(x), R(x, y)$;
- for each transition from q to q' labeled with a negative letter R^-, we add the rule $P_{q'}(y) \leftarrow P_q(x), R(y, x)$;
- for each ε-transition from q to q' we add the rule $P_{q'}(x) \leftarrow P_q(x)$

This transformation is clearly in linear time, and the result clearly satisfies the desired body size bound. Further, as the result is a monadic Datalog program, it is clearly an ICG-Datalog program. Now, it is clear that, in any instance I where Q holds, from two witnessing elements a and b and a path $\pi : a = c_0, c_1, \ldots, c_n = b$ from a to b satisfying Q, we can build a derivation tree of the Datalog program by deriving $P_{q_0}(a), P_{q_1}(c_1), \ldots, P_{q_n}(c_n)$, where q_0 is the initial state and q_n is final, to match the accepting path in the automaton A that witnesses that π is a match of Q. Conversely, any derivation tree of the Datalog program P that witnesses that an instance satisfies P can clearly be used to extract a path of relations which corresponds to an accepting run in the automaton.

We now extend this argument to SAC2RPQs. Recall from [Bar13] that a C2RPQ is a conjunction of 2RPQs, i.e., writing a 2RPQ as $Q(x, y)$ with its two free variables, a C2RPQ is a CQ built on RPQs. An AC2RPQ is a C2RPQ where the undirected graph on variables defined by co-occurrence between variables is acyclic, and a SAC2RPQ further imposes that there are no loops (i.e., atoms of the C2RPQ of the form $Q(x, x)$) and no multiedges (i.e., for each variable pair, there is at most one atom where it occurs).

We will also make a preliminary observation on ICG-Datalog programs: any rule of the form (*) $A(x) \leftarrow A_1(x), \ldots, A_n(x)$, where A and each A_i is a unary atom, can be rewritten

in linear time to rules with bounded body size, by creating unary intensional predicates A'_i for $1 \leqslant i \leqslant n$, writing the rule $A'_n(x) \leftarrow A_n(x)$, writing the rule $A'_i(x) \leftarrow A'_{i+1}(x), A_i(x)$ for each $1 \leqslant i < n$, and writing the rule $A(x) \leftarrow A'_1(x)$. Hence, we will write rules of the form (*) in the transformation, with unbounded body size, being understood that we can finish the process by rewriting out each rule of this form to rules of bounded body size.

Given a SAC2RPQ Q, we compute in linear time the undirected graph G on variables, and its connected components. Clearly we can rewrite each connected component separately, by defining one $\mathrm{Goal}_i()$ 0-ary predicate for each connected component i, and adding the rule $\mathrm{Goal}() \leftarrow \mathrm{Goal}_1(), \ldots, \mathrm{Goal}_n()$: this is a rule of form (*), which we can rewrite. Hence, it suffices to consider each connected component separately.

Hence, assuming that the graph G is connected, we root it at an arbitrary vertex to obtain a tree T. For each node n of T (corresponding to a variable of the SAC2RPQ), we define a unary intensional predicate P'_n which will intuitively hold on elements where there is a match of the sub-SAC2RPQ defined by the subtree of T rooted at n, and one unary intensional predicate $P''_{n,n'}$ for all non-root n and children n' of n in T which will hold whenever there is a match of the sub-SAC2RPQ rooted at n which removes all children of n except n'. Of course we add the rule $\mathrm{Goal}() \leftarrow P'_{n_\mathrm{r}}(x)$, where n_r is the root of T.

Now, we rewrite the SAC2RPQ to monadic Datalog by rewriting each edge of T independently, as in the argument for 2RPQs above. Specifically, we assume that the edge when read from bottom to top corresponds to a 2RPQ; otherwise, if the edge is oriented in the wrong direction, we can clearly compute an automaton for the reverse language in linear time from the Thompson automaton, by reversing the direction of transitions in the automaton, and swapping the initial state and the final state. We modify the previous construction by replacing the rule for the initial state P_{q_0} by $P_{q_0}(x) \leftarrow P'_{n'}(x)$ where n' is the lower node of the edge that we are rewriting, and the rule for the goal predicate in the head is replaced by a rule $P''_{n,n'}(x) \leftarrow P_{q_\mathrm{f}}(x)$, where n is the upper node of the edge, and q_f is the final state of the automaton for the edge: this is the rule that defines the $P''_{n,n'}$.

Now, we define each P'_n as follows:

- If n is a leaf node of T, we define P'_n by the same rules that we used to define P_{q_0} in the previous construction, so that P'_n holds of all elements in the active domain of an input instance.
- If n is an internal node of T, we define $P'_n(x) \leftarrow P''_{n,n_1}(x), \ldots, P''_{n,n_m}(x)$, where $n_1, \ldots, n_m$ are the children of n in T: this is a rule of form (*).

Now, given an instance I satisfying the SAC2RPQ, from a match of the SAC2RPQ as a rooted tree of paths, it is easy to see by bottom-up induction on the tree that we derive P_v with the desired semantics, using the correctness of the rewriting of each edge. Conversely, a derivation tree for the rewriting can be used to obtain a rooted tree of paths with the correct structure where each path satisfies the RPQ corresponding to this edge. ◀

C Proofs for Section 6 (Compilation to Automata)

C.1 Details on Tree Encodings

We first explain how we encode and decode structures of bounded treewidth to trees whose alphabet size depends only on the treewidth bound and on the signature. Having fixed a signature σ and a treewidth $k \in \mathbb{N}$, we define a domain $\mathcal{D}_k = \{a_1, \ldots, a_{2k+2}\}$ and a finite alphabet Γ^k_σ whose elements are pairs (d, s), with d being a subset of up to $k+1$ elements of $\mathcal{D}_k$, and s being either the empty set or an instance consisting of a single σ-fact over some

subset of d: in the latter case, we will abuse notation and identify s with the one fact that it contains. A (σ, k)-*tree encoding* is simply a rooted, binary, ordered, full Γ_σ^k-tree $\langle E, \lambda\rangle$; the fact that $\langle E, \lambda\rangle$ is ordered is merely for technical convenience when running bNTAs, but it is otherwise inessential.

Intuitively, a tree encoding $\langle E, \lambda\rangle$ can be decoded (up to isomorphism) to an instance $\mathrm{dec}(\langle E, \lambda\rangle)$ with the elements of $\mathcal{D}$ being decoded to the domain elements: each occurrence of an element $a_i \in \mathcal{D}$ in an a_i-*connected subtree of* E, i.e., a maximal connected subtree where a_i appears in the first component of each node, is decoded to a fresh element. In other words, reusing the same a_i in adjacent nodes in $\langle E, \lambda\rangle$ mean that they stand for the same element, and using a_i elsewhere in the tree creates a new element. It is easy to see that $\mathrm{dec}(\langle E, \lambda\rangle)$ has treewidth $\leqslant k$, as a tree decomposition for it can be constructed from $\langle E, \lambda\rangle$. Conversely, any instance I of treewidth $\leqslant k$ has a (σ, k)-*encoding*, i.e., a Γ_σ^k-tree $\langle E, \lambda\rangle$ such that $\mathrm{dec}(\langle E, \lambda\rangle)$ is I up to isomorphism: we can construct it from a tree decomposition, replicating each bag of the decomposition to code each fact in its own node of the tree encoding. What matters is that this process is FPT-linear for k, so that we will use the following claim:

▶ **Lemma 41 [FFG02] (see [Ama16] for our type of encodings).** *The problem, given an instance I of treewidth $\leqslant k$, of computing a tree encoding of I, is FPT-linear for k.*

C.2 Evaluation

▶ **Proposition 21.** *For any alphabet Γ, given a Γ-tree T and a Γ-SATWA A, we can determine whether $T \models A$ in time $O(|T| \cdot |A|)$.*

Proof. We use Theorem 32 to compute a provenance cycluit C of the SATWA (modified to be a $\overline{\Gamma}$-SATWA by simply ignoring the second component of the alphabet) in time $O(|T| \cdot |A|)$. Then we conclude by evaluating the resulting provenance cycluit (for an arbitrary valuation of that circuit) in time $O(|C|)$ using Proposition 30.

Note that, intuitively, the fixpoint evaluation of the cycluit can be understood as a least fixpoint computation to determine which pairs of states and tree nodes (of which there are $O(|T| \cdot |A|)$) are reachable. ◀

C.3 Compilation

▶ **Theorem 22.** *Given an ICG-Datalog program P of body size k_P and $k_\mathrm{I} \in \mathbb{N}$, we can build in FPT-linear time in $|P|$ (parameterized by $k_\mathrm{P}, k_\mathrm{I}$) a SATWA A_P testing P for treewidth k_I.*

First, we introduce some useful notations to deal with valuations of variables as constants of the encoding alphabet. Recall that $\mathcal{D}_{k_\mathrm{I}}$ is the domain for treewidth k_I, used to define the alphabet of tree encodings of width k_I.

▶ **Definition 42.** Given a tuple $\mathbf{x}$ of variables, a *partial valuation* of $\mathbf{x}$ is a function ν from $\mathbf{x}$ to $\mathcal{D}_{k_\mathrm{I}} \sqcup \{?\}$. The set of *undefined* variables of ν is $U(\nu) = \{x_j \mid \nu(x_j) = ?\}$: we say that the variables of $U(\nu)$ are *not defined* by ν, and the other variables are *defined* by ν.

A *total valuation* of $\mathbf{x}$ is a partial valuation ν of $\mathbf{x}$ such that $U(\nu) = \emptyset$. We say that a valuation ν *extends* another valuation ν' if the domain of ν' is a superset of that of ν, all variables defined by ν are defined by ν' and are mapped to the same value. For $\mathbf{y} \subseteq \mathbf{y}$, we say that ν is *total on* $\mathbf{y}$ if its restriction to $\mathbf{y}$ is a total valuation.

For any two partial valuations ν of $\mathbf{x}$ and ν' of $\mathbf{y}'$ if we have $\nu(x) = \nu'(x)$ for all x in $(\mathbf{x} \cap \mathbf{y}') \setminus (U(\nu) \cup U(\nu'))$, we write $\nu \cup \nu'$ for the valuation on $\mathbf{x} \cup \mathbf{y}'$ that maps every x to $\nu(x)$ or $\nu'(x)$ if one is defined, and to "?" otherwise.

When ν is a partial valuation of $\mathbf{x}$ with $\mathbf{x} \subseteq \mathbf{x}'$ and we define a partial valuation ν' of $\mathbf{x}'$ with $\nu' := \nu$, we mean that ν' is defined like ν on $\mathbf{x}$ and is undefined on $\mathbf{x}' \setminus \mathbf{x}$.

▶ **Definition 43.** Let $\mathbf{x}$ and $\mathbf{y}$ be two tuples of variables of same arity (note that some variables of $\mathbf{x}$ may be repeated, and likewise for $\mathbf{y}$). Let $\nu : \mathbf{x} \to \mathcal{D}_{k_\mathrm{I}}$ be a total valuation of $\mathbf{x}$. We define $\mathrm{Hom}_{\mathbf{y},\mathbf{x}}(\nu)$ to be the (unique) homomorphism between the tuple $\mathbf{y}$ and the tuple $\nu(\mathbf{x})$, if such a homomorphism exists; otherwise, $\mathrm{Hom}_{\mathbf{y},\mathbf{x}}(\nu)$ is null.

The rest of this section proves Theorem 22 in two steps. First, we build a SATWA A'_P and we prove that A'_P tests P for treewidth k_I; however, the construction of A'_P that we present is not FPT-linear. Second, we explain how to modify the construction to construct an equivalent SATWA A_P while respecting the FPT-linear time bound.

Construction of A'_P. We construct the SATWA A'_P by describing its states and transitions. First, for every extensional atom $S(\mathbf{x})$ appearing in (the body) of a rule of P and partial valuation ν of $\mathbf{x}$, we introduce a state $q^\nu_{S(\mathbf{x})}$. This will be the starting state of a run if it is possible to navigate the tree encoding from some starting node and build this way a total valuation ν' that extends ν and such that $S(\nu'(\mathbf{x}))$ holds in the tree encoding, in a node whose domain elements that are in the image of ν' will decode to the same element as they do in the node where the automaton can reach state $q^\nu_{S(\mathbf{x})}$. In doing so, one has to be careful not to leave the occurrence subtree of the values of the valuation (the "allowed subtree"). Indeed, in a tree encoding, an element $a \in \mathcal{D}_{k_\mathrm{I}}$ appearing in two bags that are separated by another bag not containing a is used to encode two distinct elements of the original instance, rather than the same element. We now formally define the transitions needed to implement this.

Let $(d, s) \in \Gamma_\sigma^{k_\mathrm{I}}$ be a symbol; we have the following transitions:

- If there is a j such that $\nu(x_j) \neq ?$ and $\nu(x_j) \notin d$, then $\Delta(q^\nu_{S(\mathbf{x})}, (d, s)) := \bot$. This is to prevent the automaton from leaving the allowed subtree.
- Else if ν is not total, then $\Delta(q^\nu_{S(\mathbf{x})}, (d, s)) := q^\nu_{S(\mathbf{x})} \vee \bigvee_{a \in d, x_j \in U(\nu)} q^{\nu \cup \{x_j \mapsto a\}}_{S(\mathbf{x})}$. That is, either we continue navigating in the same state, or we guess a value for some undefined variable.
- Else if ν is total but $s \neq S(\nu(\mathbf{x}))$, then $\Delta(q^\nu_{S(\mathbf{x})}, (d, s)) := q^\nu_{S(\mathbf{x})}$: if the fact s of the node is not a match, then we continue searching.
- Else, the only remaining possibility is that ν is total and that $s = S(\nu(\mathbf{x}))$, in which case we set $\Delta(q^\nu_{S(\mathbf{x})}, (d, s)) := \top$, i.e., we have found a node containing the desired fact.

For every rule r of P, subset $\mathcal{A}$ of the literals in the body of r, and partial valuation ν of the variables in $\mathcal{A}$ that is total for the variables in the head of r, we introduce a state $q^{\nu,\mathcal{A}}_r$. This state is intended to prove the literals in $\mathcal{A}$ with the partial valuation ν. We will describe the transitions for those states later.

For every intensional predicate $R(\mathbf{x})$ appearing in a rule of P and total valuation ν of $\mathbf{x}$, we have a state $q^\nu_{R(\mathbf{x})}$. This state is intended to prove $R(\mathbf{x})$ with the total valuation ν. Let $(d, s) \in \Gamma_\sigma^{k_\mathrm{I}}$ be a symbol; we have the following transitions:

- If there is a j such that $\nu(x_j) \notin d$, then $\Delta(q^\nu_{R(\mathbf{x})}, (d, s)) := \bot$. This is again in order to prevent the automaton from leaving the allowed subtree.
- Else, $\Delta(q^\nu_{R(\mathbf{x})}, (d, s))$ is defined a disjunction of all the $q^{\nu',\mathcal{A}}_r$ for each rule r such that the head of r is $R(\mathbf{y})$, $\nu' := \mathrm{Hom}_{\mathbf{y},\mathbf{x}}(\nu)$ is not null and $\mathcal{A}$ is the set of all literals in the body of r. Notice that because ν was total on $\mathbf{x}$, ν' is also total on $\mathbf{y}$. This transition simply means that we need to chose an appropriate rule to prove $R(\mathbf{x})$. We point out here that

these transitions are the ones that make the construction quadratic instead of linear in $|P|$, but this will be handled later.

It is now time to describe transitions for the states $q_r^{\nu,\mathcal{A}}$. Let $(d, s) \in \Gamma_\sigma^{k_\mathrm{I}}$, then:

- If there is a variable z in $\mathcal{A}$ such that z is defined by ν and $\nu(z) \notin d$, then $\Delta(q_r^{\nu,\mathcal{A}}, (d, s)) := \bot$.
- Else, if $\mathcal{A}$ contains at least two literals, then $\Delta(q_r^{\nu,\mathcal{A}}, (d, s))$ is defined as a disjunction of $q_r^{\nu,\mathcal{A}}$ and of $\left[\text{a disjunction over all the non-empty sets } \mathcal{A}_1, \mathcal{A}_2 \text{ that partition } \mathcal{A} \text{ of } \left[\text{a disjunction over all the total valuations } \nu' \text{ of } U(\nu) \cap \mathrm{vars}(\mathcal{A}_1) \cap \mathrm{vars}(\mathcal{A}_2) \text{ with values in } d \text{ of } \left[q_r^{\nu\cup\nu',\mathcal{A}_1} \wedge q_r^{\nu\cup\nu',\mathcal{A}_2}\right]\right]\right]$. This transition means that we allow to split in two partitions the literals that need to be proven, and for each partition we launch one run that will have to prove it. In doing so, we have to take care that the two runs will build valuations that are consistent. This is why we fix the value of the variables that they have in common with a total valuation ν'.
- Else, if $\mathcal{A} = \{S(\mathbf{y})\}$ where S is an extensional relation, then $\Delta(q_r^{\nu,\mathcal{A}}, (d, s)) := q^\nu_{S(\mathbf{y})}$.
- Else, if $\mathcal{A} = \{R'(\mathbf{y})\}$ or $\{\neg R'(\mathbf{y})\}$ where R' is an intensional relation, and if $|\mathbf{y}| = 1$, and if $\nu(y)$ is undefined (where we write y the one element of $\mathbf{y}$), then $\Delta(q_r^{\nu,\mathcal{A}}, (d, s)) := q_r^{\nu,\mathcal{A}} \vee \bigvee_{a\in d} q_r^{\nu\cup\{y\mapsto a\},\mathcal{A}}$.
- Else, if $\mathcal{A} = \{R'(\mathbf{y})\}$ where R' is an intensional relation, then we will only define the transitions in the case where ν is total on $\mathbf{y}$, in which case we set $\Delta(q_r^{\nu,\mathcal{A}}, (d, s)) := q^\nu_{R'(\mathbf{y})}$. It is sufficient to define the transitions in this case, because $q_r^{\nu,\{R'(\mathbf{y})\}}$ can only be reached if ν is total on $\mathbf{y}$. Indeed, if $|\mathbf{y}| = 1$, then ν must be total on $\mathbf{y}$ because we would have applied the previous bullet point otherwise. If $|\mathbf{y}| > 1$, the only way we could have reached the state $q_r^{\nu,\{R'(\mathbf{y})\}}$ is by a sequence of transitions involving $q_r^{\nu_0,\mathcal{A}_0}, \ldots, q_r^{\nu_m,\mathcal{A}_m}$, where $\mathcal{A}_0$ are all the literals in the body of r, $\mathcal{A}_m$ is $\{R'(\mathbf{y})\}$ and ν_m is ν. We can then see that, during the partitioning process, $R'(\mathbf{y})$ must have been separated from all the extensional atoms that formed its guard, hence all its variables have been assigned a valuation.
- Else, if $\mathcal{A} = \{\neg R'(\mathbf{y})\}$ with R' intensional, then $\Delta(q_r^{\nu,\mathcal{A}}, (d, s)) := \neg q^\nu_{R'(\mathbf{y})}$. Again, we can show that it suffices to consider the case where ν is total on $\mathbf{y}$, for the same reasons as in the previous bullet point.

Finally, the initial state of A'_P is $q^\emptyset_{\mathrm{Goal}}$.

We describe the stratification function ζ' of A'_P. Let ζ be that of P. For any state q of the form $q^\nu_{T(\mathbf{x})}$ or $q_r^{\nu,\mathcal{A}}$ with r having as head relation T, then $\zeta'(q)$ is 0 if T is extensional and $\zeta(T)$ (which is $\geqslant 1$) if T is intensional. Notice that then only states corresponding to extensional relations are in the first stratum. It is then clear from the transitions that ζ' is a valid stratification function for A'_P.

As previously mentioned, the construction of A'_P is not FPT-linear, but we will explain at the end of the proof how to construct in FPT-linear time a SATWA A_P equivalent to A'_P.

***A'_P* tests *P* on tree encodings of width $\leqslant k_\mathrm{I}$.** To show this claim, let $\langle T, \lambda_E\rangle$ be a (σ, k_I)-tree encoding. Let I be the instance obtained by decoding $\langle T, \lambda_E\rangle$; we know that I has treewidth $\leqslant k_\mathrm{I}$ and that we can define from $\langle T, \lambda_E\rangle$ a tree decomposition $\langle T, \mathrm{dom}\rangle$ of I whose underlying tree is also T. For each node $n \in T$, let $\mathrm{dec}_n : \mathcal{D}_{k_\mathrm{I}} \to \mathrm{dom}(n)$ be the function that decodes the elements in node n of the encoding to the elements of I that are in

the corresponding bag of the tree decomposition, and let $\mathrm{enc}_n : \mathrm{dom}(n) \to \mathcal{D}_{k_\mathrm{I}}$ be the inverse function that encodes back the elements, so that we have $\mathrm{dec}_n \circ \mathrm{enc}_n = \mathrm{enc}_n \circ \mathrm{dec}_n = \mathrm{Id}$. We will denote elements of $\mathcal{D}_{k_\mathrm{I}}$ by a and elements in the domain of I by c.

We recall some properties of tree decompositions and tree encodings:

▶ **Property 44.** *Let n_1, n_2 be nodes of T and $a \in \mathcal{D}_{k_\mathrm{I}}$ be an (encoded) element that appears in the λ_E-image of n_1 and n_2. Then the element a appears in every node in the path from n_1 to n_2 if and only if $\mathrm{dec}_{n_1}(a) = \mathrm{dec}_{n_2}(a)$.*

▶ **Property 45.** *Let n_1, n_2 be nodes of T and c be an element of I that appears in $\mathrm{dom}(n_1) \cap \mathrm{dom}(n_2)$. Then for every node n' on the path from n_1 to n_2, c is also in $\mathrm{dom}(n')$, and moreover $\mathrm{enc}_{n'}(c) = \mathrm{enc}_{n_1}(c)$.*

We start with the following lemma about extensional facts:

▶ **Lemma 46.** *For every extensional relation S, node $n \in T$, variables $\mathbf{y}$, and partial valuation ν of $\mathbf{y}$, there exists a run ρ of A'_P starting at node n in state $q^\nu_{S(\mathbf{y})}$ if and only if there exists a fact $S(\mathbf{c})$ in I such that we have $\mathrm{dec}_n(\nu(y_j)) = c_j$ for every y_j defined by ν. We call this a match $\mathbf{c}$ of $S(\mathbf{y})$ in I that is* compatible with ν at node n.

Proof. We prove each direction in turn.

Forward direction. Suppose there exists a run ρ of A'_P starting at node n in state $q^\nu_{S(\mathbf{y})}$. First, notice that by design of the transitions starting in a state of that form, states appearing in the labeling of the run can only be of the form $q^{\nu'}_{S(\mathbf{y})}$ for an extension ν' of ν. We will show by induction on the run that for every node π of the run labeled by $(q^{\nu'}_{S(\mathbf{y})}, m)$, there exists $\mathbf{c}'$ such that $S(\mathbf{c}') \in I$ and $\mathbf{c}'$ is compatible with ν' at node m. This will conclude the proof of the forward part of the lemma, by taking $m = n$.

The base case is when π is a leaf of ρ. The node π is then labeled by $(q^{\nu'}_{S(\mathbf{y})}, m)$ such that $\Delta(q^{\nu'}_{S(\mathbf{y})}, \lambda_E(m)) = \top$. Let $(d, s) = \lambda_E(m)$. By construction of the automaton we have that ν' is total and $s = S(\nu'(\mathbf{y}))$. We take $\mathbf{c}'$ to be $\mathrm{dec}_m(\nu'(\mathbf{y}))$, which satisfies the compatibility condition by definition and is such that $S(\mathbf{c}') = S(\mathrm{dec}_m(\nu'(\mathbf{y}))) = \mathrm{dec}_m(s) \in I$.

When π is an internal node of ρ, we write $(q^{\nu'}_{S(\mathbf{y})}, m)$ its label. By definition of the transitions of the automaton, we have $\Delta(q^{\nu'}_{S(\mathbf{y})}, (d, s)) = q^{\nu'}_{S(\mathbf{y})} \vee \bigvee_{a \in d, y_j \in U(\nu')} q^{\nu' \cup \{y_j \mapsto a\}}_{S(\mathbf{y})}$. Hence, the node π has exactly one child π', the first component of its label is some $m' \in \mathrm{Nbh}(m)$, and we have two cases depending on the first component of its label:

- π' may be labeled by $(q^{\nu'}_{S(\mathbf{y})}, m')$. Then by induction on the run there exists $\mathbf{c}''$ such that $S(\mathbf{c}'') \in I$ and $\mathbf{c}''$ is compatible with ν' at node m'. We take $\mathbf{c}'$ to be $\mathbf{c}''$, so that we only need to check the compatibility condition, i.e., that for every y_j defined by ν', $\mathrm{dec}_m(\nu'(y_j)) = c_j = \mathrm{dec}_{m'}(\nu'(y_j))$. This is true by Property 44. Indeed, for every y_j defined by ν', we must have $\nu'(y_j) \in m'$, otherwise π' would have a label that cannot occur in a run.
- π' is labeled by $(q^{\nu' \cup \{y_j \mapsto a\}}_{S(\mathbf{y})}, m')$ for some $a \in d$ and for some $y_j \in U(\nu')$. Then by induction on the run there exists $\mathbf{c}''$ such that $S(\mathbf{c}'') \in I$ and $\mathbf{c}''$ is compatible with $\nu' \cup \{y_j \mapsto a\}$ at node m'. We take $\mathbf{c}'$ to be $\mathbf{c}''$, which again satisfies the compatibility condition thanks to Property 44.

Backward direction. Now, suppose that there exists $\mathbf{c}$ such that $S(\mathbf{c}) \in I$ and $\mathbf{c}$ is compatible with ν at node n. The fact $S(\mathbf{c})$ is encoded somewhere $\langle T, \lambda_E \rangle$, so there exists a node m such that, letting (d, s) be $\lambda_E(m)$, we have $\mathrm{dec}_m(s) = S(\mathbf{c})$. Let $n =$

$m_1, m_2, \ldots, m_p = m$ be the nodes on the path from n to m, and (d_i, s_i) be $\lambda_E(m_i)$ for $1 \leqslant i \leqslant p$. By compatibility, for every y_j defined by ν we have $\mathrm{dec}_n(\nu(y_j)) = c_j$. But $\mathrm{dec}_n(\nu(y_j)) \in \mathrm{dom}(n)$ and $c_j \in \mathrm{dom}(m)$ so by Property 45, for every $1 \leqslant i \leqslant p$ we have $c_j \in \mathrm{dom}(m_i)$ and $\mathrm{enc}_{m_i}(c_j) = \mathrm{enc}_n(c_j) = \mathrm{enc}_n(\mathrm{dec}_n(\nu(y_j))) = \nu(y_j)$, so that $\nu(y_j) \in d_i$. We can then construct a run ρ starting at node n in state $q^\nu_{S(\mathbf{y})}$ as follows. The root π_1 is labeled by $(q^\nu_{S(\mathbf{y})}, n)$, and for every $2 \leqslant i \leqslant p$, π_i is the unique child of π_{i-1} and is labeled by $(q^\nu_{S(\mathbf{y})}, m_i)$. This part is valid because we just proved that for every i, there is no j such that y_j is defined by ν and $\nu(y_j) \notin d_j$. Now from π_m, we continue the run by staying at node m and building up the valuation, until we reach a total valuation ν_f such that $\nu_\mathrm{f}(\mathbf{y}) = \mathrm{enc}_m(\mathbf{c})$. Then we have $s = S(\nu_\mathrm{f}(\mathbf{y}))$ and the transition is $\top$, which completes the definition of the run. ◀

The preceding lemma concerns the base case of extensional relations. We now prove a similar *equivalence lemma* for intensional relations. This lemma allows us to conclude the correctness proof, by applying it to the Goal() predicate and to the root of the tree-encoding.

▶ **Lemma 47.** *For every relation R, node $n \in T$ and total valuation ν of $\mathbf{x}$, there exists a run ρ of A'_P starting at node n in state $q^\nu_{R(\mathbf{x})}$ if and only if $R(\mathrm{dec}_n(\nu(\mathbf{x}))) \in P(I)$.*

Proof. We will prove this equivalence by induction on the stratum $\zeta(R)$ of the relation R. The base case ($\zeta(R) = 0$, so R is an extensional relation) was shown in Lemma 46. For the inductive case, where R is an intensional relation, we prove each direction separately.

Forward direction. First, suppose that there exists a run ρ of A'_P starting at node n in state $q^\nu_{R(\mathbf{x})}$. We show by induction on the run (from bottom to top) that for every node π of the run the following implications hold:

(i) If π is labeled with $(q^{\nu'}_{R'(\mathbf{y})}, m)$, then there exists $\mathbf{c}$ such that $R'(\mathbf{c}) \in P(I)$ and $\mathbf{c}$ is compatible with ν' at node m.
(ii) If π is labeled with $\neg(q^{\nu'}_{R'(\mathbf{y})}, m)$ with ν' total, then $R'(\mathrm{dec}_m(\nu'(\mathbf{y}))) \notin P(I)$.
(iii) If π is labeled with $(q^{\nu', \mathcal{A}}_r, m)$, then there exists a mapping $\mu : \mathrm{vars}(\mathcal{A}) \to \mathrm{Dom}(I)$ that is compatible with ν' at node m and such that:
- For every positive literal $S(\mathbf{z})$ in $\mathcal{A}$, then $S(\mu(\mathbf{z})) \in P(I)$.
- For every negative literal $\neg S(\mathbf{z})$ in $\mathcal{A}$, then $S(\mu(\mathbf{z})) \notin P(I)$.

The base case is when π is a leaf. Notice that in this case, and by construction of A'_P, the node π cannot be labeled by states corresponding to rules of P: indeed, there are no transition for these states leading to a tautology, and all transitions to such a state are from a state in the same stratum, so π could not be a leaf. Thus, we have three subcases:

- π may be labeled by $(q^{\nu'}_{R'(\mathbf{y})}, m)$, where R' is extensional. We must show (i), but this follows from Lemma 46.
- π may be labeled by $(q^{\nu'}_{R'(\mathbf{y})}, m)$, where R' is intensional and verifies $\zeta(R') < i$, and where ν' is total. Again we need to show (i). By definition of the run ρ, this implies that there exists a run of A'_P starting at m in state $q^{\nu'}_{R'(\mathbf{y})}$. But ν' is total, so by induction on the strata we have (using the forward direction of the equivalence lemma) that $R'(\mathrm{dec}_m(\nu'(\mathbf{y}))) \in P(I)$. We take $\mathbf{c}$ to be $\mathrm{dec}_m(\nu'(\mathbf{y}))$, which satisfies the required conditions.
- π may be labeled by $\neg(q^{\nu'}_{R'(\mathbf{y})}, m)$, where R' is intensional and verifies $\zeta(R') < i$, and where ν' is total. We need to show (ii). By definition of the run ρ there exists no run of A'_P starting at m in state $q^{\nu'}_{R'(\mathbf{y})}$. But ν' is total, so by induction on the strata we have

(using the backward direction of the equivalence lemma) that $R'(\mathrm{dec}_m(\nu'(\mathbf{y}))) \notin P(I)$, which is what we needed to show.

For the induction case, where π is an internal node and letting (d, s) be $\lambda_E(m)$ in what follows, we have five subcases:

- π may be labeled by $(q^{\nu'}_{R'(\mathbf{y})}, m)$ with R' extensional. We must show (i), but this follows from Lemma 46.
- π may be labeled by $(q^{\nu'}_{R'(\mathbf{y})}, m)$ with R' intensional and ν' total. We need to prove (i). In that case, given the definition of $\Delta(q^{\nu'}_{R'(\mathbf{y})}, (d, s))$ and by induction (on the run), there exists a child π' of π labeled by $(q_r^{\nu'', \mathcal{A}}, m')$, where $m' \in \mathrm{Nbh}(m)$, where r is a rule with head $R'(\mathbf{z})$, where $\nu'' = \mathrm{Hom}_{\mathbf{z},\mathbf{y}}(\nu')$ is a partial valuation which is not null, and where $\mathcal{A}$ is the set of literals of r. Then, by induction on the run, there exists a mapping $\mu : \mathrm{vars}(\mathcal{A}) \to \mathrm{Dom}(I)$ that verifies (iii). Thus by definition of the semantics of P we have that $R'(\mu(\mathbf{z})) \in P(I)$, and we take $\mathbf{c}$ to be $\mu(\mathbf{z})$. What is left to check is that the compatibility condition holds. We need to prove that $\mathrm{dec}_m(\nu'(\mathbf{y})) = \mathbf{c}$, i.e., that $\mathrm{dec}_m(\nu'(\mathbf{y})) = \mu(\mathbf{z})$. We know, by definition of μ, that $\mathrm{dec}_{m'}(\nu''(\mathbf{z})) = \mu(\mathbf{z})$. So our goal is to prove $\mathrm{dec}_m(\nu'(\mathbf{y})) = \mathrm{dec}_{m'}(\nu''(\mathbf{z}))$, i.e., by definition of ν'' we want $\mathrm{dec}_m(\nu'(\mathbf{y})) = \mathrm{dec}_{m'}(\mathrm{Hom}_{\mathbf{z},\mathbf{y}}(\nu')(\mathbf{z}))$. By definition of $\mathrm{Hom}_{\mathbf{z},\mathbf{y}}(\nu')$, we know that $\nu'(\mathbf{y}) = \mathrm{Hom}_{\mathbf{z},\mathbf{y}}(\nu')(\mathbf{z})$, and this implies the desired equality by applying Property 44 to m and m'.
- π may be labeled by $(q_r^{\nu', \mathcal{A}}, m)$, where $\mathcal{A} = \{R''(\mathbf{y})\}$ or $\{\neg R''(\mathbf{y})\}$, where $|\mathbf{y}| = 1$, where the head of r uses relation R'., and where $y \in U(\nu')$ (writing y the one element of $\mathbf{y}$). We need to prove (iii). By construction we have $\Delta(q_r^{\nu', \mathcal{A}}, (d, s)) = q_r^{\nu', \mathcal{A}} \vee \bigvee_{a \in d} q_r^{\nu' \cup \{y \mapsto a\}, \mathcal{A}}$. So by definition of a run there is $m' \in \mathrm{Nbh}(m)$ and a child π' of π such that π' is labeled by $(q_r^{\nu', \mathcal{A}}, m')$ or by $(q_r^{\nu' \cup \{y \mapsto a\}, \mathcal{A}}, m')$ for some $a \in d$. In both cases it is easily seen that we can define an appropriate ν from the valuation ν' that we obtain by induction on the run (more details are given in the next bullet point).
- π may be labeled by $(q_r^{\nu', \mathcal{A}}, m)$, with $\mathcal{A} = \{R''(\mathbf{y})\}$ and ν' total on $\mathbf{y}$, the head of r using relation R'. We need to prove (iii). By construction we have $\Delta(q_r^{\nu', \mathcal{A}}, (d, s)) = q^{\nu'}_{R''(\mathbf{y})}$, so that by definition of the run there is $m' \in \mathrm{Nbh}(m)$ and a child π' of π such that π' is labeled by $(q^{\nu'}_{R''(\mathbf{y})}, m')$. Thus by induction on the run there exists $\mathbf{c}$ such that $R''(\mathbf{c}) \in P(I)$ and $\mathbf{c}$ compatible with ν' at node m'. By Property 44, $\mathbf{c}$ is also compatible with ν' at node m. We define μ by $\mu(\mathbf{y}) := \mathbf{c}$, which effectively defines it because in this case $\mathrm{vars}(r) = \mathbf{y}$, and this choice satisfies the required properties.
- π may be labeled by $(q_r^{\nu', \mathcal{A}}, m)$, with $\mathcal{A} = \{\neg R''(\mathbf{y})\}$ and ν' total on $\mathbf{y}$ and the head of r has relation R'. We again need to prove (iii). By construction we have $\Delta(q_r^{\nu', \mathcal{A}}, (d, s)) = \neg q^{\nu'}_{R''(\mathbf{y})}$ and then by definition of the automaton there exists a child π' of π labeled by $\neg(q^{\nu'}_{R''(\mathbf{y})}, m)$ with $\zeta(R'') < i$ and there exists no run starting at node m in state $q^{\nu'}_{R''(\mathbf{y})}$. So by induction on the strata (using the backward direction of the equivalence lemma) we have $R''(\mathrm{dec}_m(\nu'(\mathbf{y}))) \notin P(I)$. We define μ by $\mu(\mathbf{y}) = \mathrm{dec}_m(\nu'(\mathbf{y}))$, which effectively defines it because $\mathrm{vars}(r) = \mathbf{y}$, and the compatibility conditions are satisfied.
- π may be labeled by $(q_r^{\nu', \mathcal{A}}, m)$, with $|\mathcal{A}| \geqslant 2$. We need to prove (iii). Given the definition of $\Delta(q_r^{\nu', \mathcal{A}}, (d, s))$ and by definition of the run, one of the following holds:
  - There exists $m' \in \mathrm{Nbh}(m)$ and a child π' of π such that π' is labeled by $(q_r^{\nu', \mathcal{A}}, m')$. By induction there exists $\mu' : \mathrm{vars}(\mathcal{A}) \to \mathrm{Dom}(I)$ satisfying (iii) for node m'. We can take μ to be μ', which satisfies the required properties.
  - There exist $m_1, m_2 \in \mathrm{Nbh}(m)^2$ and π_1, π_2 children of π and non-empty sets $\mathcal{A}_1, \mathcal{A}_2$ that partition $\mathcal{A}$ and a total valuation ν'' of $\mathrm{vars}(\mathcal{A}_1) \cap \mathrm{vars}(\mathcal{A}_2)$ with values in d such

that π_1 is labeled by $(q_r^{\nu' \cup \nu'', \mathcal{A}_1}, m_1)$ and π_2 is labeled by $(q_r^{\nu' \cup \nu'', \mathcal{A}_2}, m_2)$. By induction there exists $\mu_1 : \mathrm{vars}(\mathcal{A}_1) \to \mathrm{Dom}(I)$ and similarly μ_2 that satisfy (iii). Thanks to the compatibility conditions for μ_1 and μ_2 and to Property 44 applied to m_1 and m_2 via m, we can define $\mu : \mathrm{vars}(\mathcal{A}) \to \mathrm{Dom}(I)$ with $\mu = \mu_1 \cup \mu_2$. One can check that μ satisfies the required properties.

Hence, the forward direction of our equivalence lemma is proven.

Backward direction. We now prove the backward direction of the induction case of our main equivalence lemma (Lemma 47). From the induction hypothesis on strata, we know that, for every relation R with $\zeta(R) \leqslant i-1$, for every node $n \in T$ and total valuation ν of $\mathbf{x}$, there exists a run ρ of A'_P starting at node n in state $q^{\nu}_{R(\mathbf{x})}$ if and only if we have $R(\mathrm{dec}_n(\nu(\mathbf{x}))) \in P(I)$. Let R be a relation with $\zeta(R) = i$, $n \in T$ be a node and ν be a total valuation of $\mathbf{x}$ such that $R(\mathrm{dec}_n(\nu(\mathbf{x}))) \in P(I)$. We need to show that there exists a run ρ of A'_P starting at node n in state $q^{\nu}_{R(\mathbf{x})}$. We will prove this by induction on the smallest $j \in \mathbb{N}$ such that $R(\mathrm{dec}_n(\nu(\mathbf{x}))) \in \Xi^j_P(P_{i-1}(I))$, where Ξ^j_P is the j-th application of the immediate consequence operator for the program P (see [AHV95]) and P_{i-1} is the restriction of P with only the rules up to strata $i-1$. The base case, when $j = 0$, is in fact vacuous since $R(\mathrm{dec}_n(\nu(\mathbf{x}))) \in \Xi^0_P(P_{i-1}(I)) = P_{i-1}(I)$ implies that $\zeta(R) \leqslant i-1$, whereas we assumed $\zeta(R) = i$. For the inductive case ($j \geqslant 1$), we have $R(\mathrm{dec}_n(\nu(\mathbf{x}))) \in \Xi^j_P(P_{i-1}(I))$ so by definition of the semantics of P, there is a rule r of the form $R(\mathbf{z}) \leftarrow L_1(\mathbf{y}_1) \dots L_t(\mathbf{y}_t)$ of P and a mapping $\mu : \mathbf{y}_1 \cup \dots \cup \mathbf{y}_t \to \mathrm{Dom}(I)$ such that $\mu(\mathbf{z}) = \mathrm{dec}_n(\nu(\mathbf{x}))$ and, for every literal L_l in the body of r:

- If $L_l(\mathbf{y}_l) = R_l(\mathbf{y}_l)$ is a positive literal, then $R_l(\mu(\mathbf{y}_l)) \in \Xi^{j-1}_P(P_{i-1}(I))$
- If $L_l(\mathbf{y}_l) = \neg R_l(\mathbf{y}_l)$ is a negative literal, then $R_l(\mu(\mathbf{y}_l)) \notin P_{i-1}(I)$

To achieve our goal of building a run starting at node n in state $q^{\nu}_{R(\mathbf{x})}$, we will construct a run starting at node n in state $q_r^{\nu', \{L_1, \dots, L_t\}}$, with $\nu' = \mathrm{Hom}_{\mathbf{z},\mathbf{x}}(\nu)$. The first step is to take care of the literals of the rule and to prove that:

(i) If $L_l(\mathbf{y}_l) = R_l(\mathbf{y}_l)$ is a positive literal, then there exists a node m_l and a valuation ν_l such that there exists a run ρ_l starting at node m_l in state $q^{\nu_l}_{R_l(\mathbf{y}_l)}$ and such that $\mathrm{dec}_{m_l}(\nu_l(\mathbf{y}_l)) = \mu(\mathbf{y}_l)$.

(ii) If $L_l(\mathbf{y}_l) = \neg R_l(\mathbf{y}_l)$ is a negative literal, then for every node m_l such that $\mathrm{dec}_{m_l}(\nu_l(\mathbf{y}_l)) = \mu(\mathbf{y}_l)$, there exists no run starting at node m_l in state $q^{\nu_l}_{R_l(\mathbf{y}_l)}$.

We straightforwardly get (ii) by using the induction on the strata of our equivalence lemma. We now prove (i). Suppose first that R_l is an extensional relation. We define m_l to be the node in which $R_l(\mu(\mathbf{y}_l))$ appears (in the tree decomposition), and we define ν_l to be $\mathrm{enc}_{m_l}(\mu(\mathbf{y}_l))$. We then have $\mathrm{dec}_{m_l}(\nu_l(\mathbf{y}_l)) = \mathrm{dec}_{m_l}(\mathrm{enc}_{m_l}(\mu(\mathbf{y}_l))) = \mu(\mathbf{y}_l)$, so by Lemma 46 there exists a run ρ_l starting at node m_l in state $q^{\nu_l}_{R_l(\mathbf{y}_l)}$.

Suppose now that R_l is intensional. By (syntactical) definition of our fragment, $R_l(\mathbf{y}_l)$ is clique-guarded by some extensional relations in the body of r, say $R_{l_1}(\mathbf{y}_{l_1}), \dots, R_{l_c}(\mathbf{y}_{l_c})$. Moreover, there exist nodes $m_{l_1}, \dots, m_{l_c}$ such that for every $1 \leqslant p \leqslant c$, $R_{l_p}(\mu(\mathbf{y}_{l_p}))$ is in m_{l_p}. By a well-known property of tree decompositions, this implies that there exists a node in which all the elements of $\mu(\mathbf{y}_l)$ appear (see Lemma 1 of [Gav74], Lemma 2 of [BK10]). We define m_l to be this node. We define $\nu_l : \mathbf{y}_l \to \mathcal{D}_{k_\mathrm{I}}$ to be $\bigsqcup_{1 \leqslant p \leqslant c} \nu_{l_p}$, where the ν_{l_p} are the valuations obtained when proving (i) for extensional relations in the case above. This definition makes sense. Indeed, let v be a variable in $\mathbf{y}_{l_p} \cap \mathbf{y}_{l_{p'}}$. Because we defined ν_{l_p} by $\mathrm{enc}_{m_l}(\mu)$, we have that $\mu(v)$ appears in the bag (of the tree decomposition) of m_{l_p}, and similarly in that of $m_{l_{p'}}$.

Then by Property 45, we have that $\mathrm{enc}_{m_{l_p}}(\mu(v)) = \mathrm{enc}_{m_{l_{p'}}}(\mu(v))$, and thus $\nu_{l_p}(v) = \nu_{l_{p'}}(v)$. We now show that $\mathrm{dec}_{m_l}(\nu_l(\mathbf{y}_l)) = \mu(\mathbf{y}_l)$, which will imply by induction hypothesis (on the number j of applications of the immediate consequence operator) that there exists a run ρ_l starting at node m_l in state $q^{\nu_l}_{R_l(\mathbf{y}_l)}$. Pick $v \in \mathbf{y}_l$. It is in some $\mathbf{y}_{l_p}$ for some $1 \leqslant p \leqslant c$, so by definition of ν_l and of ν_{l_p} we only need to prove $\mathrm{dec}_{m_l}(\mathrm{enc}_{m_{l_p}}(\mu(v))) = \mu(v)$. But we have $\mu(v)$ is in the bag of m_{l_p} (by definition of ν_{l_p}), and in that of m_l (by definition of m_l), so that again by Property 45 we get $\mathrm{enc}_{m_{l_p}}(\mu(v)) = \mathrm{enc}_{m_l}(\mu(v))$, which gives us what we wanted because we can decode. Hence, (i) and (ii) are proven.

The second step is, from the runs ρ_l that we just constructed, to construct a run starting at node n in state $q_r^{\nu', \{L_1, \ldots, L_t\}}$. We describe in a high-level manner how we build the run. Starting at node n, we partition the literals to prove (i.e., the atoms of the body of the rule that we are applying), in the following way:

- We create one class in the partition for each positive literal R_l (which can be intentional or extensional) such that m_l is n, which we prove directly at the current node. Specifically, we handle these literals one by one, by splitting the remaining literals in two using the transition formula corresponding to the rule and by staying at node n and building the valuations according to $\mathrm{dec}_n(\mu)$.
- We create one class in the partition for each negative literal $\neg R_l(\mathbf{y}_l)$ such that all its variables $\mathbf{y}_l$ are defined by the valuation: we use (ii) to know that there will be no run for there literals.
- For the remaining literals, considering all neighbors of n in the tree encoding, we split the literals into one class per neighbor n', where each literal L_l is mapped to the neighbor that allows us to reach its node m_l. We ignore the empty classes. If there is only one class, i.e., we must go in the same direction to prove all facts, we simply go to the right neighbor n', remaining in the same state. If there are multiple classes, we partition the facts and prove each class on the correct neighbor.
  One must then argue that, when we do so, we can indeed choose the image by ν' of all elements that were shared between literals in two different classes and were not yet defined in ν'. The reason why this is possible is because we are working on a tree encoding: if two facts of the body share a variable x, and the two facts will be proved in two different directions, then the variable x must be mapped to the same element in the two direction, which implies that it must occur in the node m where we split. Hence, we can indeed choose the image of x at the moment when we split.

◀

FPT-linear time construction. Finally, we justify that we can construct in FPT-linear time the automaton A_P which recognizes the same language as A'_P. The size of $\Gamma_\sigma^{k_\mathrm{I}}$ only depends on k_I and on the extensional signature, which are fixed. As the number of states is linear in $|P|$, the number of transitions is linear in $|P|$. Most of the transitions are of constant size, and in fact one can check that the only problematic transitions are those for states of the form $q^\nu_{R(\mathbf{x})}$ with R intensional, specifically the second bullet point. Indeed, we have defined a transition from $q^\nu_{R(\mathbf{x})}$, for each valuation ν of a rule body, to the $q_r^{\nu', \mathcal{A}}$ for linearly many rules, so in general there are quadratically many transitions.

However, it is easy to fix this problem: instead of having one state $q^\nu_{R(\mathbf{x})}$ for every occurrence of an intensional predicate $R(\mathbf{x})$ in a rule body of P and total valuation ν of this rule body, we can instead have a constant number of states $q_{R(\mathbf{a})}$ for $\mathbf{a} \in \mathcal{D}_{k_\mathrm{I}}^{\mathrm{arity}(R)}$. In other words, when we have decided to prove a single intensional atom in the body of a rule, instead of remembering the entire valuation of the rule body (as we remember ν in $q^\nu_{R(\mathbf{x})}$), we can

simply forget all other variable values, and just remember the tuple which is the image of $\mathbf{x}$ by ν, as in $q_{R(\mathbf{a})}$. Remember that the number of such states is only a function of k_{P} and k_{I}, because bounding k_{P} implies that we bound the arity of P, and thus the arity of intensional predicates.

We now redefine the transitions for those states :

- If there is a j such that $a_j \notin d$, then $\Delta(q_{R(\mathbf{a})}, (d, s)) = \bot$.
- Else, $\Delta(q_{R(\mathbf{a})}, (d, s))$ is a disjunction of all the $q_r^{\nu', \mathcal{A}}$ for each rule r such that the head of r is $R(\mathbf{y})$, $\nu'(\mathbf{y}) = \mathbf{a}$ and $\mathcal{A}$ is the set of all literals in the body of r.

The key point is that a given $q_r^{\nu', \mathcal{A}}$ will only appear in rules for states of the form $q_{R(\mathbf{a})}$ where R is the predicate of the head of r, and there is a constant number of such states.

We also redefine the transitions that used these states:

- Else, if $\mathcal{A} = \{R'(\mathbf{y})\}$ with R' intensional, then $\Delta(q_r^{\nu, \mathcal{A}}, (d, s)) = q_{R'(\nu(\mathbf{y}))}$.
- Else, if $\mathcal{A} = \{\neg R'(\mathbf{y})\}$ with R' intensional, then $\Delta(q_r^{\nu, \mathcal{A}}, (d, s)) = \neg q_{R'(\nu(\mathbf{y}))}$.

A_P recognizes the same language as A'_P. Indeed, consider a run of A'_P, and replace every state $q^{\nu}_{R(\mathbf{x})}$ with R intensional by the state $q_{R(\nu(\mathbf{x}))}$: we obtain a run of A_P. Conversely, being given a run of A_P, observe that every state $q_{R(\mathbf{a})}$ comes from a state $q_r^{\nu, \{R(\mathbf{y})\}}$ with $\nu(\mathbf{y}) = \mathbf{a}$. We can then replace $q_{R(\mathbf{a})}$ by the state $q^{\nu}_{R(\mathbf{x})}$ to obtain a run of A'_P.

D Proofs for Section 7 (Provenance Cycluits)

D.1 Cycluits

The semantics of monotone cycluits is formally defined by Algorithm 1.

Algorithm 1: Semantics of monotone cycluits

Input: Monotone cycluit $C = (G, W, g_0, \mu)$, Boolean valuation $\nu : C_{\mathsf{inp}} \to \{0, 1\}$
Output: $\{g \in C \mid \nu(g) = 1\}$

$S_0 := \{g \in C_{\mathsf{inp}} \mid \nu(g) = 1\}$
$i := 0$
do
  i++
  $S_i := S_{i-1} \cup \left\{g \in C \mid (\mu(g) = \vee), \exists g' \in S_{i-1}, g' \to g \in W\right\} \cup$
    $\left\{g \in C \mid (\mu(g) = \wedge), \{g' \mid g' \to g \in W\} \subseteq S_{i-1}\right\}$
While $S_i \neq S_{i-1}$
return S_i

▶ **Proposition 26.** *For any monotone cycluit C and Boolean valuation ν of C, the set $S := \{g \in C \mid \nu'(g) = 1\}$ is the minimal set of gates (under inclusion) such that:*
(i) S contains the true input gates, i.e., it contains $\{g \in C_{\mathsf{inp}} \mid \nu(g) = 1\}$;
(ii) for any g such that $\mu(g) = \vee$, if some input gate of g is in S, then g is in S;
(iii) for any g such that $\mu(g) = \wedge$, if all input gates of g are in S, then g is in S.

Proof. The operator used in Algorithm 1 is clearly monotone, so by the Knaster–Tarski theorem, the outcome of the computation is the intersection of all set of gates satisfying the conditions in Proposition 26. ◀

Algorithm 1 is a naive fixpoint algorithm running in quadratic time, but we show that the same output can be computed in linear time with Algorithm 2.

▶ **Proposition 27.** *Given any monotone cycluit C and Boolean valuation ν of C, we can compute the evaluation ν' of C in linear time.*

Proof. We use Algorithm 2. We first prove the claim about the running time. The preprocessing to compute M is linear-time in C (we enumerate at most once every wire), and the rest of the algorithm is clearly in linear time as it is a variant of a DFS traversal of the graph, with the added refinement that we only visit nodes that evaluate to 1 (i.e., OR-gates with some input that evaluates to 1, and AND-gates where all inputs evaluate to 1).

We now prove correctness. We use the characterization of Proposition 26. We first check that S satisfies the properties:

(i) S contains the true input gates by construction.
(ii) Whenever an OR-gate g' has an input gate g in S, then, when we added g to S, we have necessarily followed the wire $g \to g'$ and added g' to Q, and later added it to S.
(iii) Whenever an AND-gate g' has all its input gates g' in S, there are two cases. The first case in when g has no input gates at all, in which case S contains it by construction. The second case is where such input gates exist: in this case, observe that $M[g']$ was initially equal to the degree of g', and that we decrement it for each input gate g of g' that we add to S. Hence, considering the last input gate g of g' that we add to S, it must be the case that $M[g']$ reaches zero when we decrement it, and then we add g to Q, and later to S.

Second, we check that S is minimal. Assume by contradiction that it is not the case, and consider the first gate g which is added to S while not being in the minimal Boolean valuation S'. It cannot be the case that g was added when initializing S, as we initialize S to contain true input gates and AND-gates with no inputs, which must be true also in S' by the characterization of Proposition 26. Hence, we added g to S in a later step of the algorithm. However, we notice that we must added g to S because of the value of its input gates. By minimality of g, these input gates have the same value in S and in S'. This yields a contradiction, because the gates that we add to S are added following the characterization of Proposition 26. This concludes the proof. ◀

D.2 Stratified cycluits

We show the claim that a Boolean cycluit is stratified iff it contains no cycle of gates involving a $\neg$-gate, and that a stratification function can be computed in linear time.

▶ **Proposition 48.** *Any Boolean cycluit C is stratified iff it it contains no cycle of gates involving a $\neg$-gate. Moreover, a stratification function can be computed in linear time from C.*

Proof. To see why a stratified Boolean cycluit C cannot contain a cycle of gates involving a $\neg$-gate, assume by contradiction that it has such a cycle $g_1 \to g_2 \to \cdots \to g_n \to g_1$. As C is stratified, there exists a stratification function ζ. From the properties of a stratification function, we know that $\zeta(g_1) \leqslant \zeta(g_2) \leqslant \cdots \leqslant \zeta(g_1)$, so that we must have $\zeta(g_1) = \cdots = \zeta(g_n)$. However, letting g_i be such that $\mu(g_i) = \neg$, we know that $\zeta(g_{i-1}) < \zeta(g_i)$ (or, if $i = 1$, $\zeta(g_n) < \zeta(g_1)$), so we have a contradiction.

We now prove the converse direction of the claim, i.e., that any Boolean cycluit which does not contain a cycle of gates involving a $\neg$-gate must have a stratification function, and

Algorithm 2: Linear-time evaluation of monotone cycluits

Input: Monotone cycluit $C = (G, W, g_0, \mu)$, Boolean valuation $\nu : C_{\mathsf{inp}} \to \{0, 1\}$
Output: $\{g \in C \mid \nu(g) = 1\}$

```
/* Precompute the in-degree of ∧ gates                                  */
for g ∈ C s.t. μ(g) = ∧ do
    M[g] := |{g' ∈ C | g' → g}|
Q := {g ∈ C_inp | ν(g) = 1} ∪ {g ∈ C | (μ(C) = ∧) ∧ M[g] = 0}   /* as a stack */
S := ∅                                                  /* as a bit array */
while Q ≠ ∅ do
    pop g from Q
    if g ∉ S then
        add g to S
        for g' ∈ C | g → g' do
            if μ(g') = ∨ then
                push g' into Q
            if μ(g') = ∧ then
                M[g'] := M[g'] − 1
                if M[g'] = 0 then
                    push g' into Q
return S
```

show how to compute such a function in linear time. Compute in linear time the strongly connected components (SCCs) of C, and a topological sort of the SCCs. As the input gates of C do not themselves have inputs, each of them must have their own SCC, and each such SCC must be a leaf, so we can modify the topological sort by merging these SCCs corresponding to input gates, and putting them first in the topological sort. We define the function ζ to map each gate of C to the index number of its SCC in the topological sort, which ensures in particular that the input gates of C are exactly the gates assigned to 0 by ζ. This can be performed in linear time. Let us show that the result ζ is a stratification function:

- For any edge $g \to g'$, we have $\zeta(g) \leqslant \zeta(g')$. Indeed, either g and g' are in the same strongly connected component and we have $\zeta(g) = \zeta(g')$, or they are not and in this case the edge $g \to g'$ witnesses that the SCC of g precedes that of g', whence, by definition of a topological sort, it follows that $\zeta(g) < \zeta(g')$.
- For any edge $g \to g'$ where $\mu(g') = \neg$, we have $\zeta(g) < \zeta(g')$. Indeed, by adapting the reasoning of the previous bullet point, it suffices to show that g and g' cannot be in the same SCC. Indeed, assuming by contradiction that they are, by definition of a SCC, there must be a path from g' to g, and combining this with the edge $g \to g'$ yields a cycle involving a $\neg$-gate, contradicting our assumption on C. ◀

▶ **Proposition 30.** *We can compute $\nu(C)$ in linear time in the stratified cycluit C and in ν.*

Proof. Compute in linear time a stratification function ζ of C using Proposition 48, and compute the evaluation following Definition 29. This can be performed in linear time.

To see why this evaluation is independent from the choice of stratification, observe that any stratification function must clearly assign the same value to all gates in an SCC. Hence,

the choosing a stratification function amounts to choosing the stratum that we assign to each SCC. Further, when an SCC S precedes another SCC S', the stratum of S must be no higher than the stratum of S'. So in fact the only freedom that we have is to choose a topological sort of the SCCs, and optionally to assign the same stratum to consecutive SCCs in the topological sort: this amounts to "merging" some SCCs, and is only possible when there are no $\neg$-gates between them. Now, in the evaluation, it is clear that the order in which we evaluate the SCCs makes no difference, nor does it matter if some SCCs are evaluated simultaneously. Hence, the evaluation of a stratified cycluit is well-defined. ◀

D.3 Building provenance cycluits

▶ **Theorem 32.** *For any fixed alphabet Γ, given a $\overline{\Gamma}$-SATWA A and a Γ-tree $\mathcal{T}$, we can build a stratified cycluit capturing the provenance of A on $\mathcal{T}$ in time $O(|A| \cdot |\mathcal{T}|)$. Moreover, this stratified cycluit has treewidth $O(|A|)$.*

To prove Theorem 32, we construct a cycluit C_T^A as follows, generalizing the construction of [ABS15b]. For each node w of T, we create an input node g_w^{i}, a $\neg$-gate $g_w^{\neg\mathrm{i}}$ defined as $\mathrm{NOT}(g_w^{\mathrm{i}})$, and an OR-gate g_w^q for each state $q \in Q$. Now for each g_w^q, for $b \in \{0,1\}$, we consider the propositional formula $\Delta(q, (\lambda(w), b))$, and we express it as a circuit that captures this formula: we let $g_w^{q,b}$ be the output gate of that circuit, we replace each variable q' occurring positively by an OR-gate $\bigvee_{w' \in \mathrm{Nbh}(w)} g_{w'}^{q'}$, and we replace each variable q' occurring negatively by the gate $g_w^{q'}$. We then define g_w^q as $\mathrm{OR}(\mathrm{AND}(g_w^{\mathrm{i}}, g_w^{q,0}), \mathrm{AND}(g_w^{\neg\mathrm{i}}, g_w^{q,1}))$. Finally, we let the output gate of C be $g_r^{q_\mathrm{I}}$, where r is the root of T.

It is clear that this process runs in linear time in $|A| \cdot |T|$. Moreover, for every node w of T, we create $O(|A|)$ gates, and those gates can only be connected between them or between gates created for the neighbors of w. Hence, the treewidth of C_T^A is $O(|A|)$: we can compute a tree decomposition of C_T^A where the underlying tree is T and where each bag b corresponding to a node w of T contains the gates defined for the node w and for the neighbors of w. The proof of Theorem 32 then follows from the following claim:

▶ **Lemma 49.** *The cycluit C_T^A is a stratified cycluit capturing the provenance of A on $\langle T, \lambda \rangle$.*

Proof. We first show that $C := C_T^A$ is a stratified cycluit. Let ζ be the stratification function of the $\overline{\Gamma}$-SATWA A and let $\{0, \ldots, m\}$ be its range. We use ζ to define ζ' as the following function from the gates of C to $\{0, \ldots, m+1\}$:

- For any input gate g_w^{i}, we set $\zeta'(g_w^{\mathrm{i}}) := 0$ and $\zeta'(g_w^{\neg\mathrm{i}}) := 1$.
- For an OR gate $g := \bigvee_{w' \in \mathrm{Nbh}(w)} g_{w'}^{q'}$, we set $\zeta'(g) := \zeta(q')$.
- For any state g_w^q, we set $\zeta'(g_w^n) := \zeta(q) + 1$, and do the same for the intermediate AND-gates used in its definition, as well as the gates in the two circuits that capture the transitions $\Delta(q, (\lambda(w), b))$ for $b \in \{0,1\}$, except for the input gates of that circuit (i.e., gates of the form $\bigvee_{w' \in \mathrm{Nbh}(w)} g_{w'}^{q'}$, which are covered by the previous point, or $g_w^{q'}$, which are covered by another application of that point).

Let us show that ζ' is indeed a stratification function for C. We first observe that it is the case that the gates in stratum zero are precisely the input gates. We then check the condition for the various possible wires:

- $g_w^{\mathrm{i}} \to g_w^{\neg\mathrm{i}}$: by construction, we have $\zeta(g_w^{\mathrm{i}}) < \zeta'(g_w^{\neg\mathrm{i}})$.
- $g \to g'$ where g' is a gate of the form g_w^q and g is an intermediate AND-gate in the definition of a g_w^q: by construction we have $\zeta'(g) = \zeta'(g')$, so in particular $\zeta'(g) \leqslant \zeta'(g')$.

- $g \to g'$ where g' is an intermediate AND-gate in the definition of a gate of the form g^q_w, and g is g^{i}_w or $g^{\neg\mathrm{i}}_w$: by construction we have $\zeta'(g) \in \{0,1\}$ and $\zeta'(g') \geqslant 1$, so $\zeta'(g) \leqslant \zeta'(g')$.
- $g \to g'$ where g is a gate in a circuit capturing the propositional formula of some transition of $\Delta(q,\cdot)$ without being an input gate or a NOT-gate of this circuit, and g' is also such a gate, or is an intermediate AND-gate in the definition of g^q_w: then g' cannot be a NOT-gate (remembering that the propositional formulae of transitions only have negations on literals), and by construction we have $\zeta'(g) = \zeta'(g')$.
- $g \to g'$ where g is of the form $\bigvee_{w' \in \mathrm{Nbh}(w)} g^q_{w'}$, and g' is a gate in a circuit describing $\Delta(q',\cdot)$ or an intermediate gate in the definition of $g^{q'}_w$. Then we have $\zeta'(g) = \zeta(q)$ and $\zeta'(g') = \zeta(q')$, and as q occurs as a positive literal in a transition of q', by definition of ζ being a transition function, we have $\zeta(q) \leqslant \zeta(q')$. Now we have $\zeta'(g) = \zeta(q)$ and $\zeta'(g') = \zeta'(q')$ by definition of ζ', so we deduce that $\zeta'(g) \leqslant \zeta'(g')$.
- $g \to g'$ where g' is of the form $\bigvee_{w' \in \mathrm{Nbh}(w)} g^{q'}_{w'}$, and g is one of the $g^{q'}_{w'}$. Then by definition of ζ' we have $\zeta'(g) = \zeta(q')$ and $\zeta'(g') = \zeta(q')$, so in particular $\zeta'(g) \leqslant \zeta'(g')$.
- $g \to g'$ where g is a NOT-gate in a circuit capturing a propositional formula $\Delta(q', (\lambda(w), b))$, and g is then necessarily a gate of the form g^q_w: then clearly q' was negated in φ so we had $\zeta(q) < \zeta(q')$, and as by construction we have $\zeta'(g) = \zeta(q)$ and $\zeta'(g') = \zeta(q')$, we deduce that $\zeta'(g) < \zeta'(g')$.

We now show that C indeed captures the provenance of A on $\langle T, \lambda\rangle$. Let $\nu : T \to \{0,1\}$ be a Boolean valuation of the inputs of C, that we extend to an evaluation $\nu' : C \to \{0,1\}$ of C. We claim the following **equivalence**: for all q and w, there exists a run ρ of A on $\nu(T)$ *starting at* w in state q if and only if $\nu(g^q_w) = 1$.

We prove this claim by induction on the stratum $\zeta(q)$ of q. Up to adding an empty first stratum, we can make sure that the base case is vacuous. For the induction step, we prove each implication separately.

Forward direction. First, suppose that there exists a run ρ starting at w in state q, and let us show that $\nu'(g^q_w) = 1$. We show by induction on the run (from bottom to top) that for each node y of the run labeled by a *positive* state (q', w') we have $\nu'(g^{q'}_{w'}) = 1$, and for every node y of the run labeled by a *negative* state $\neg(q', w')$ we have $\nu'(g^{q'}_{w'}) = 0$. The base case concerns the leaves, where there are three possible subcases:

- We may have $\lambda_r(y) = (q', w')$ with $\zeta(q') = i$, so that $\Delta(q', (\lambda(w'), \nu(w')))$ is tautological. In this case, $g^{q'}_{w'}$ is defined as $\mathrm{OR}(\mathrm{AND}(g^{\mathrm{i}}_{w'}, g^{q',1}_{w'}), \mathrm{AND}(g^{\neg\mathrm{i}}_{w'}, g^{q',0}_{w'}))$. Hence, we know that $\nu(g^{q',\nu(w)}_{w'}) = 1$ because the circuit is also tautological, and depending on whether $\nu(w)$ is 0 or 1 we know that $\nu(g^{\neg\mathrm{i}}_{w'}) = 1$ or $\nu(g^{\mathrm{i}}_{w'}) = 1$, so this proves the claim.
- We may have $\lambda_r(y) = (q', w')$ with $\zeta(q') = j$ for $j < i$. By definition of the run ρ, this implies that there exists a run starting at w' in state q'. But then, by the induction on the strata (using the forward direction of the **equivalence**), we must have $\nu(g^{q'}_{w'}) = 1$.
- We may have $\lambda_r(y) = \neg(q', w')$ with $\zeta(q') = j$ for $j < i$. Then by definition there exists no run starting at w' in state q'. Hence again by induction on the strata (using the backward direction of the **equivalence**), we have that $\nu(g^{q'}_{w'}) = 0$.

For the induction case on the run, where y is an internal node, by definition of a run there is a subset $S = \{q_{P_1}, \cdots, q_{P_n}\}$ of positive literals and a subset $N = \{\neg q_{N_1}, \cdots, \neg q_{N_m}\}$ of negative literals that satisfy $\varphi_{\nu(w')} := \Delta(q', (\lambda(w'), \nu(w')))$ such that:

- For all $q_{P_k} \in P$, there exists a child y_k of y with $\lambda_r(y_k) = (q_{P_k}, w'_k)$ where $w'_k \in \mathrm{Nbh}(w')$;
- For all $\neg q_{N_k} \in N$ there is a child $y_{w'_k}$ of y with $\lambda_r(y_{w'}) = \neg(q_{N_k}, w')$.

Then, by induction on the run, we know that for all q_{P_k} we have $\nu(g^{q_{P_k}}_{w'_k}) = 1$ and for all $\neg q_{N_k}$ we have $\nu(g^{q_{N_k}}_{w'}) = 0$. By construction of C, we have $\nu(g^{q'}_{w'}) = 1$. There are two cases: either $\nu(w') = 1$ or $\nu(w') = 0$. In the first case, remember that the first input of the OR-gate $g^{q'}_{w'}$ is an AND-gate of $g^{\mathrm{i}}_{w'}$ and the output gate $g^{q',1}_{w'}$ of a circuit coding φ_1 on inputs including the $g^{q_{P_k}}_{w'_k}$ and $g^{q_{N_k}}_{w''}$. We have $\nu(g^{\mathrm{i}}_{w'}) = 1$ because $\nu(w') = 1$, and the second gate evaluates to 1 by construction of the circuit, as witnessed by the Boolean valuation of the $g^{q_{P_k}}_{w'_k}$ and $g^{q_{N_k}}_{w''}$. In the second case we follow the same reasoning but with the second input to $g^{q'}_{w'}$ instead, which is an AND-gate on $g^{\neg\mathrm{i}}_{w'}$ and φ_0.

By induction on the run, the claim is proven, and applying it to the root of the run concludes the proof of the first direction of the **equivalence** (for the induction step of the induction on strata).

Backward direction. We now prove the converse implication for the induction step of the induction on strata, i.e., letting i be the current stratum, for every node w and state q with $\zeta(q) = i$, if $\nu(g^q_w) = 1$ then there exists a run ρ of A starting at w. From the definition of the stratification function ζ' of the cycluit from ζ, we have $\zeta'(g^q_w) = \zeta(q)$, so as $\nu(g^q_w) = 1$ we know that $\nu_i(g^q_w) = 1$, where ν_i is the i-th stratum evaluation ν_i of C (remember Definition 29). By induction hypothesis on the strata, we know from the **equivalence** that, for any $j < i$, for any gate $g^{q''}_{w''}$ of C with $\zeta(g^{q''}_{w''}) = j$, we have $\nu_j(g^{q''}_{w''}) = 1$ iff there exists a run ρ of A on $\nu(T)$ *starting at* w'' in state q''.

We show the claim by an induction on the iteration in the application of Algorithm 1 for ν_i where the gate g^q_w was set to 1. The base case concerns gates that were initially true before applying the algorithm: by

Recall that the definition of ν_i according to Definition 29 proceeds in three steps. Initially, we fix the value in ν_i of gates of lower strata, so we can then conclude by induction hypothesis on the strata. We then set the value of all NOT-gates in ν_i, but these cannot be of the form $g^{q'}_{w'}$ so there is nothing to show. Last, we evaluate all other gates with Algorithm 1. We then show our claim by an induction on the iteration in the application of Algorithm 1 for ν_i where the gate g^q_w was set to 1. The base case, where g^q_w was initially true, was covered in the beginning of this paragraph.

For the induction step on the application of Algorithm 1, when a gate $\nu_i(g^{q'}_{w'})$ is set to true, as $\nu_i(g^{q'}_{w'})$ is an OR-gate by construction, from the workings of Algorithm 1, there are two possibilities: either its input AND-gate that includes $g^{\mathrm{i}}_{w'}$ was true, or its input AND-gate that includes $g^{\neg\mathrm{i}}_{w'}$ was true. We prove the first case, the second being analogous. From the fact that $g^{\mathrm{i}}_{w'}$ is true, we know that $\nu(w') = 1$. Consider the other input gate to that AND gate, which is the output gate of a circuit C' reflecting $\varphi := \Delta(q', (\lambda(w'), \nu(w')))$, with the input gates adequately substituted. We consider the value by ν_i of the gates that are used as input gates of C' in the construction of C (i.e., OR-gates, in the case of variables that occur positively, or directly $g^{q''}_{w'}$-gates, in the case of variables that occur negatively). By construction of C', the corresponding Boolean valuation ν' is a witness to the satisfaction of φ. By induction hypothesis on the strata (for the negated inputs to C'; and for the non-negated inputs to C' which are in a lower stratum) and on the step at which the gate was set to true by Algorithm 1 (for the inputs in the same stratum, which must be positive), the valuation of these inputs reflects the existence of the corresponding runs. Hence, we can assemble these (i.e., a leaf node in the first two cases, a run in the third case) to obtain a run starting at w' for state q' using the Boolean valuation ν' of the variables of φ; this valuation satisfies φ as we have argued.

This concludes the two inductions of the proof of the **equivalence** for the induction step

of the induction on strata, which concludes the proof. ◀

Putting it together. We now conclude the proof of Theorem 24 by explaining how this provenance construction for $\overline{\Gamma}$-SATWAs can be used to compute the provenance of an ICG-Datalog query on a treelike instance. This is again similar to [ABS15b].

Recall the definition of tree encodings from Appendix C.1, and the definition of the alphabet Γ^k_σ. To represent the dependency of automaton runs on the presence of individual facts, we will be working with $\overline{\Gamma^k_\sigma}$-trees, where the Boolean annotation on a node n indicates whether the fact coded by n (if any) is present or absent. The semantics is that we map back the result to Γ^k_σ as follows:

▶ **Definition 50.** We define the mapping ε from $\overline{\Gamma^k_\sigma}$ to Γ^k_σ by:

- $\varepsilon((d, s), 1)$ is just (d, s), indicating that the fact of s (if any) is kept;
- $\varepsilon((d, s), 0)$ is $(d, \emptyset)$, indicating that the fact of s (if any) is removed.

We abuse notation and also see ε as a mapping from $\overline{\Gamma^k_\sigma}$-trees to Γ^k_σ-trees by applying it to each node of the tree.

As our construction of provenance applies to automata on $\overline{\Gamma^k_\sigma}$, we show the following easy *lifting lemma* (generalizing Lemma 3.3.4 of [Ama16]):

▶ **Lemma 51.** *For any Γ^k_σ-SATWA A, we can compute in linear time a $\overline{\Gamma^k_\sigma}$-SATWA A' such that, for any $\overline{\Gamma^k_\sigma}$-tree E, we have that A' accepts E iff A accepts $\varepsilon(E)$.*

Proof. The proof is exactly analogous to that of Lemma 3.3.4 of [Ama16]. ◀

We are now ready to conclude the proof of our main provenance result (Theorem 24):

Proof of Theorem 24. Given the program P and instance I, use Theorem 22 to compute in FPT-linear time in $|P|$ a Γ^k_σ-SATWA A that tests it on tree encodings of width $\leqslant k_{\mathrm{I}}$, for k_{I} the treewidth bound. Compute also in FPT-linear time a tree encoding E of the instance I, i.e., a Γ^k_σ-tree E, using Lemma 41. Lift the Γ^k_σ-SATWA A in linear time using Lemma 51 to a $\overline{\Gamma^k_\sigma}$-SATWA A', and use Theorem 32 on A' and E to compute in FPT-linear time a stratified cycluit C' that captures the provenance of A' on E: the inputs of C' correspond to the nodes of E. The treewidth of C' is in $O(A')$, so it is FPT-linear in $|P|$. Let C be obtained from C' in linear time by changing the inputs of C' as follows: those which correspond to nodes n of E containing a fact (i.e., with label (d, s) for $|s| = 1$) are renamed to be an input gate that stands for the fact of I coded in this node; the nodes n of E containing no fact are replaced by a 0-gate, i.e., an OR-gate with no inputs. Clearly, C is still a stratified Boolean cycluit that satisfies the required treewidth bound, and C_{inp} is exactly the set of facts of I.

All that remains to show is that C captures the provenance of P on I in the sense of Definition 23. To see why, consider an arbitrary Boolean valuation ν mapping the facts of I to $\{0, 1\}$, and call $\nu(I) := \{F \in I \mid \nu(F) = 1\}$. We must show that $\nu(I)$ satisfies P iff $\nu(C) = 1$. By construction of C, it is obvious that $\nu(C) = 1$ iff $\nu'(C') = 1$, where ν' is the Boolean valuation of C'_{inp} defined by $\nu'(n) = \nu(F)$ when n codes some fact F in E, and $\nu'(n) = 0$ otherwise. By definition of the provenance of A' on E, we have $\nu'(C') = 1$ iff A' accepts $\nu'(E)$, that is, by definition of lifting, iff A accepts $\varepsilon(\nu'(E))$. Now all that remains to observe is that $\varepsilon(\nu'(E))$ is precisely a tree encoding of the instance $\nu(I)$: this is by definition of ν' from ν, and by definition of our tree encoding scheme (see "subinstance-compatibility" in [Ama16]). Hence, by definition of A testing P, the tree $\varepsilon(\nu'(E))$ is accepted by A iff $\nu(I)$ satisfies P. This finishes the chain of equivalences, and concludes the proof of Theorem 24. ◀

E Proofs for Section 8 (From Cycluits to Circuits and Probability Bounds)

E.1 From cycluits to circuits

▶ **Theorem 34.** *There is an $\alpha \in \mathbb{N}$ s.t., for any stratified cycluit C of treewidth k, we can compute in time $O(2^{k^\alpha}|C|)$ a circuit C' which is equivalent to C and has treewidth $O(2^{k^\alpha})$.*

The proof of Theorem 34 is quite technical and proceeds in several successive stages, corresponding to the following sections. In all sections but the last one, we focus on *monotone cycluits*. Throughout the whole proof, we assume without loss of generality that there are no *isolated gates*, i.e., gates that have no input gate and being the input to no gate (because such gates are not necessarily reflected in tree decompositions). We call a *0-gate* an OR-gate with no inputs, and a *1-gate* an AND-gate with no inputs.

E.1.1 Rewriting to arity-two cycluits

For technical convenience, it will be easier to work with *arity-two cycluits*:

▶ **Definition 52.** The *fan-in* of a gate g in a monotone cycluit C is the number of gates g' such that $g' \to g$ is a wire of C (note that, as we are working with cycluits, this may include $g' = g$, i.e., g may be its own input). For $c \in \mathbb{N}$, a monotone cycluit C is *arity-c* if each AND- and OR-gate has fan-in at most c, and if no gate is its own input (i.e., there is no wire of the form $g \to g$).

It is obvious that monotone cycluits, just like circuits, can be rewritten in linear time to an *arity-two* monotone cycluit, by taking advantage of the associativity of the OR and AND Boolean operations: we just rewrite each AND- and OR-gate with fan-in > 2 to a tree of gates of fan-in $\leqslant 2$ that computes the required Boolean operation on the original set of inputs.

What is more technical to show, however, is that this rewriting operation can be performed on a *treelike* cycluit, while ensuring that the treewidth of the result is still only a function of the treewidth of the original cycluit. Hence, the result that we wish to prove is the following:

▶ **Lemma 53.** *For any monotone cycluit C and tree decomposition T of width k of C, we can compute in time $O(|C| + |T|)$ a monotone arity-two cycluit C' which is equivalent to C, and a tree decomposition T' of width k^2 of C'.*

Proof. First, to ensure the condition that no gate of the circuit is its own input, observe that any wire of the form $g \to g$ can be dropped if g is an OR-gate with other inputs; if g is an OR-gate whose only input is itself, then it will never evaluate to 1 so we can replace it by a 0-gate, and if g is an AND-gate with a wire $g \to g$ then we can replace g by a 0-gate as it will never evaluate to 1. This can be done in linear time and does not increase the width of the tree decomposition.

We assume without loss of generality that the tree decomposition T has arity at most two (i.e., each bag has at most two children), as we can otherwise ensure it by rewriting T in linear time (by replacing each bag with more than two children with a hierarchy of bags with the same contents and with two children). We assign each wire $g \to g'$ of the circuit to some bag b of T such that $g, g' \in \mathrm{dom}(b)$: this is doable in linear time by Lemma 3.1 of [FFG02]. We now perform a bottom-up traversal of the tree decomposition T. In this process, we will annotate bags b of the tree decomposition with sets of pairs of gates $g \mapsto g'$, indicating that g is to be renamed to g' in the subtree of T rooted at b. During the bottom-up traversal, we

memorize, for each gate g in the current bag b, its fan-in for the wires that we have seen so far: it is initially 0, and is computed at bag b as the sum of the fan-in of g at all the child bags (if any) plus its fan-in for the wires assigned to bag b. Whenever the fan-in of a gate g at a bag b exceeds two, we do the following: we create one gate g_1 and label the first child bag of b (if it exists) with $g \mapsto g_1$, create one gate g_2 and label the second child bag of b (if it exists) with $g \mapsto g_2$, and create a gate g' for the wires of the current bag. We let the input gates of g' be the input gates present at the current bag, and rewrite this to arity-two, which introduces $\leqslant k-1$ new gates in total. Now, we set the input gates of g to be g'' and g_1, where g'' is a fresh gate for the same operation as g, and set the input gates of g'' to be g_2 and g'. We have introduced $\leqslant k$ new gates in the current bag per gate initially in b, so the new size of b is at most k^2.

Once this pass is done, we perform a top-down pass to perform the renamings indicated on the bags, which is doable in linear time because the number of renamings to perform at any stage is constant (it is bounded by the width of the bags).

It is clear that this process produces an arity-two cycluit equivalent to C, and the rewritten tree decomposition T' has width k^2. ◀

E.1.2 Regrouped tree decompositions

Now that we can rewrite monotone cycluits to arity-two monotone cycluits in linear time with only a quadratic blowup in the treewidth, we show that we can impose an additional condition on tree decompositions of arity-two circuits, which we call being *regrouped*:

▶ **Definition 54.** A tree decomposition T of a cycluit $C = (G, W, g_0, \mu)$ is *regrouped* if it satisfies the following property: for any gate $g \in G$, letting $\mathrm{ins}(g) := \{g' \in G \mid g' \to g \in W\}$, there is a bag b_g of T such that $\{g\} \cup \mathrm{ins}(g) \subseteq \mathrm{dom}(b)$.

In other words, a tree decomposition is regrouped if each gate has a witness bag which contains that gate and *all* of its inputs. This can be contrasted with the usual notion of tree decomposition, where each of the input wires can be witnessed by a different bag. We show that, for arity-two cycluits, we can assume that tree decompositions are regrouped in this sense:

▶ **Lemma 55.** *For any monotone arity-two cycluit C and tree decomposition T of C of width k, we can compute in time $O(|C| + |T|)$ a regrouped tree decomposition T' of C of width $\leqslant 3k$.*

Proof. We define T' as having same skeleton as T and define each $b' \in T'$, letting b be the corresponding bag in T, to have $\mathrm{dom}(b') = \mathrm{dom}(b) \cup \bigcup\{\mathrm{ins}(g) \mid g \in b\}$. In other words, we add to each bag in T' the input gates of the gates that occur there. It is clear that T' then satisfies the regrouped condition: take as witnessing bag b_g of a gate g any bag b' of T' corresponding to a bag $b \in T$ such that $g \in \mathrm{dom}(b)$. It is clear that T' has width at most $3k$, as each bag b' of T' contains at most the contents of the corresponding bag b of T plus two additional input gates for each gate occurring in b, thanks to the fact that C is arity-two.

As the regrouped condition subsumes the condition of tree decompositions that requires an occurrence of every wire, the only thing left to show is that the occurrences of any gate g in T' still forms a connected subtree. As T is a tree decomposition of C, let T_g be the connected subtree containing the occurrences of g in T, and, for every g' such that $g \in \mathrm{ins}(g')$, let $T_{g'}$ be the connected subtree of the occurrences of g' in T. For any g' such that $g \in \mathrm{ins}(g')$, as T is a tree decomposition, it must witness the wire $g \to g'$, so the subtrees T_g and $T_{g'}$ intersect in T. Now, the subtree T'_g of the occurrences of g in T' is by construction the

analogue in T' of the union of the connected subtrees T_g and $T_{g'}$ for g' such that $g \in \mathrm{ins}(g')$. We conclude because the union of connected subtrees that intersect is also a connected subtree. ◀

E.1.3 Normal-form tree decompositions

Having presented the notion of regrouped tree decompositions, we now introduce the notion of *normal-form* tree decompositions, which satisfy the regrouped conditions and satisfy other additional conditions. Unlike what we have done so far, for technical reasons that will become apparent in the next section, we will define these tree decompositions as *undirected* connected trees rather than *rooted* trees: this makes no difference, as we used to consider tree decompositions as rooted trees simply for convenience. We call T an *unrooted* tree decomposition if it is a tree decomposition whose underlying tree is unrooted, and, for $b \in T$, the *rooting* of T at bag b is a tree decomposition, in the usual rooted sense, obtained by picking b as the root in this tree; this always yields a valid tree decomposition according to our earlier definitions.

▶ **Definition 56.** An unrooted tree decomposition T of a monotone cycluit C is in *normal form* if it satisfies the following additional conditions:

1. Each bag b of T has (undirected) degree either 1 or 3; we call b a *leaf bag* if it has degree 1, and an *internal bag* if it has degree 3.
2. There is an injective mapping φ from each gate g of C to a leaf bag b_g with $\mathrm{dom}(b_g) = \{g\} \cup \mathrm{ins}(g)$, where $\mathrm{ins}(g)$ are the input gates of g. The *φ-occurrences* of g are the leaf bags $\{\varphi(g)\} \sqcup \{\varphi(g') \mid g' \in \mathrm{ins}(g)\}$.
3. The tree decomposition is *scrubbed*, meaning that:
   - if a gate g occurs in a leaf bag b, then b is a φ-occurrence of g;
   - if a gate g occurs in a non-leaf bag b then there are two φ-occurrences $b_1 \neq b_2$ of g and two neighbor bags $b'_1 \neq b'_2$ of b (note that we may have $b_i = b'_i$), such that, for $i \in \{1, 2\}$, the unique simple path from b to b_i goes through b'_i.

Observe that the first condition ensures that, for any leaf bag b of T, the rooting of T at bag b is a tree decomposition where the root bag b has exactly one child, and all other bags have exactly two children except the leaf bags which have zero. The second condition implies that T, or indeed any rooting of T, is regrouped, with the witnesses for regrouping being chosen at the leaf bags in an injective fashion. The third condition intuitively ensures that each gate g occurs in bags of T precisely in its φ-occurrences and on the paths between the φ-occurrences of g. In particular, the first point of the third condition implies that any leaf bag with non-empty domain must be in the image of φ. We also call a rooted tree decomposition *scrubbed* when it satisfies the analogous condition (with undirected paths).

We now show:

▶ **Lemma 57.** *For any monotone arity-two cycluit C and regrouped tree decomposition T of C of width k, we can compute in time $O(|C| + |T|)$ a normal-form tree decomposition T' of C of width k.*

Proof. We modify T in linear time in the following way to satisfy the second condition. For each gate g, letting b_g be the witness bag for g guaranteed by replace b_g by a new bag b'_g with $\mathrm{dom}(b'_g) = \mathrm{dom}(b_g)$, whose first child is b_g and whose second child is a bag b''_g whose domain is $(\{g\} \cup \mathrm{ins}(g))$ (note that this is a subset of $\mathrm{dom}(b_g)$). We can now set $\varphi(g)$ to be b''_g, with φ being injective.

We then rewrite T in linear time in the usual way to ensure that each bag has at most two children.

We next make T scrubbed in three linear time passes. First, we remove all gates g in leaf bags b when b is not a φ-occurrence of g; this removal does not affect the fact that T is a tree decomposition, because each wire is still covered at the φ-image of its target gate, and the occurrences of each gate still form a connected subtree as we are only removing occurrences at leaf bags. Second, we do a bottom-up pass to compute, for each occurrence of a gate g at a bag b, a value $n_{g,b}$ indicating how many different children of b have a φ-occurrence of g as a descendent (this number is between 0 and 2): if this number is 0 at an internal bag, then we remove g. The removal of g in this case does not violate the fact that T is a tree decomposition: indeed, it does not affect the fact that the wires are covered, because this is still the case at the φ-images, which are not internal bags; and gate occurrences still form connected subtrees, because whenever we remove a gate g from a bag b then there is only one neighbor of b at most (namely, its parent) where g also occurs. Third, we do a top-down pass where, whenever we have $n_{g,b} = 1$ for a gate g at a bag b when we encounter b top-down (i.e., $n_{g,b} = 1$ and $n_{g,b'} = 1$ for all ancestors of b' where g occurs), then we remove g from b. The result is still a tree decomposition, because again wires are still covered at the φ-images, and whenever we remove a gate g from a bag b then there is only one neighbor of b at most (namely, one of its children) where g also occurs.

We now let T' be the result of forgetting the orientation of T. It is clear that the first point of the third condition is satisfied, and the second point is satisfied for any occurrence of a gate g at an internal bag b: indeed, b either had two different children having a φ-occurrence of g as a descendant, or b had one such descendant and had an ancestor b' whose parent b'' had a φ-occurrence as a descendent of its other child. Hence, T' is scrubbed.

We last ensure that T' satisfies the first condition. Note that, as each bag of T had at most two children, the undirected degree of bags of T is at most 3, so we can simply enforce the condition by adding bags with empty domain. This concludes the proof. ◀

E.1.4 Rewriting monotone cycluits

Armed with normal-form tree decompositions and the lemmas required to obtain them, we are now ready to show our rewriting result for *monotone* cycluits:

▶ **Theorem 58.** *There is an $\alpha \in \mathbb{N}^*$ s.t., for any monotone cycluit C of treewidth k, we can compute in time $O(2^{k^\alpha} \cdot |C|)$ a circuit C' which is equivalent to C and has treewidth $\leqslant 2^{k^\alpha}$.*

To prove the theorem, we will rely on a notion of *partial assignment* of a monotone cycluit, which we can use to enforce that some gates are evaluated to 1, even when they are not input gates:

▶ **Definition 59.** For any monotone cycluit C and disjoint subsets $S^+ \sqcup S^-$ of gates that contain no input gate, letting $S := (S^+, S^-)$, we define the *partial assignment* $\rho_S(C)$ to be the monotone cycluit obtained from C by replacing each gate of S^+ by a 1-gate, and each gate of S^- by a 0-gate.

We will also rely on a variant of Algorithm 1 to evaluate parts of a cycluit in a tree decomposition based on the child parts in the decomposition. It will work using the following notion:

▶ **Definition 60.** The *constant gates* of a cycluit $C = (G, W, g_0, \mu)$ are the gates of $G \backslash C_{\mathsf{inp}}$ that have no input gates, i.e., $C_{\mathsf{cst}} := \{g \in G \mid \nexists g' \in G, g' \to g \in W\} \backslash C_{\mathsf{inp}}$: in other words,

Algorithm 3: Partition-based evaluation of monotone cycluits

Input: Monotone cycluit $C = (G, W, g_0, \mu)$, Boolean valuation $\nu : C_{\mathsf{inp}} \to \{0,1\}$, evaluation partition $G_1 \sqcup G_2$
Output: $\{g \in C \mid \nu(g) = 1\}$

$CST := \{g \in C_{\mathsf{inp}} \mid \nu(g) = 1\} \cup \{g \in C_{\mathsf{cst}} \mid \mu(g) = \wedge\}$
$T_0 := CST$
$n := |F(G_1 \sqcup G_2)| + 2$
for i *from* 1 *to* n **do**
  for j *from* 1 *to* 2 **do**
    $U_{i,j} := T_{i-1} \cap F_j(G_1 \sqcup G_2)$
    $V_{i,j} := \mathrm{SubEval}(C, CST \cup U_{i,j}, G_j)$
  $T_i := V_{i,1} \cup V_{i,2} \cup CST$
return T_n
Function $\mathrm{SubEval}(C, Z, G')$
  Input: Monotone cycluit $C = (G, W, g_0, \mu)$, subset $Z \subseteq G$ of true gates, subset $G' \subseteq G$ of iteration gates
  Output: Subset of gates that became true (observe the similarity to Algorithm 1)
  $Z_0 := Z$
  $l := 0$
  do
    $l := l + 1$
    $Z_l := Z_{l-1} \cup \left\{ g \in G' \mid (\mu(g) = \vee), \exists g' \in Z_{l-1}, g' \to g \in W \right\}$
    $\cup \left\{ g \in G' \mid (\mu(g) = \wedge), \{g' \mid g' \to g \in W\} \subseteq Z_l \right\}$
  While $Z_l \neq Z_{l-1}$
  return Z_l

these are precisely the 0- and 1-gates. The *internal gates* of G are the gates which are neither constant gates nor input gates.

An *evaluation partition* of a cycluit $C = (G, W, g_0, \mu)$ is a partition $G_1 \sqcup G_2$ of the internal gates of C. The *frontier* of $G_1 \sqcup G_2$ is $F(G_1 \sqcup G_2) := F_1(G_1 \sqcup G_2) \sqcup F_2(G_1 \sqcup G_2)$, where $F_j(G_1 \sqcup G_2) := \{g \in G_{3-j} \mid \exists g' \in G_j, g \to g' \in W\}$ for $j \in \{1, 2\}$.

This allows us to define Algorithm 3, a variant of Algorithm 1 which evaluates a cycluit by evaluating separately the parts of its evaluation partition, memorizing only the state of the frontier across evaluations. We claim the following:

▶ **Lemma 61.** *For any cycluit C, valuation ν, and evaluation partition $G_1 \sqcup G_2$ of C, the result of Algorithm 3 is indeed the set g of gates such that g evaluates to 1 in C under ν. In other words, the output is the same as that of Algorithm 1.*

Proof. Fix the cycluit C, valuation ν, and evaluation partition $G_1 \sqcup G_2$, let $F := F(G_1 \sqcup G_2)$, and write $n := |F| + 2$.

First, observe that the output on input gates and on constant gates is correct, as all such gates that evaluate to 1 are added to CST and are returned, and the ones that do not are never otherwise added to any T_i (remember that the G_j do not contain any such gates, so we only add internal gates to the Z_l). Hence, it suffices to restrict our attention to the internal gates.

The easy direction is the forward direction: let us show that, for any internal gate g in T_n, this gate is also in the output of Algorithm 1. We do so by induction on the step of the execution at which the gate was added to one of the sets Z_l. Initially, there are no internal gates in these sets. For the induction, observe that the gates added to such sets are either $\vee$-gates which have one input already previously in such a set (and so, by induction hypothesis, they are also in the output of Algorithm 1), or $\wedge$-gates where all inputs are already in such a set (and we can use the induction hypothesis again), so that indeed, when that same gate is considered in one iteration of Algorithm 1, it will also be added to the output of that algorithm.

We now prove the backward direction. To do this, first observe that, for any $j \in \{1, 2\}$, for any $1 \leqslant i < n$, we have $U_{i,j} \subseteq U_{i+1,j}$. Indeed, we have $U_{i,j} \subseteq V_{i,j} \subseteq T_i$, and all gates of $U_{i,j}$ are gates of $F_j(G_1 \sqcup G_2)$, so $U_{i+1,j}$ contains all gates of $U_{i,j}$. Now, as $U_{i,j}$ has size at most $|F(G_1 \sqcup G_2)|$, this means that there is $i' \leqslant |F(G_1 \sqcup G_2)| + 1$ such that $U_{i',j} = U_{i'+1,j}$. As the same argument applies to the other value of j, we deduce the existence of $1 \leqslant i' \leqslant |F(G_1 \sqcup G_2)| + 1$ such that this holds for all $j \in \{1, 2\}$. This clearly implies, then, that $T_{i'} = T_{i'+1}$, so that, as we chose n to be sufficiently large, we know that $T_n = T_{n-1}$.

We use this claim to conclude the proof of the backward direction. Assume by way of contradiction that there is a gate g which is added to the output of Algorithm 1 and which is not part of the output of Algorithm 3, and choose g to be the first gate that has this property (formally, as several such gates may be added in the same iteration of Algorithm 1, choose any gate among the first ones). By our initial remark about the correctness on constant gates and input gates, g must be an internal gate. Let Γ be the subset of the inputs of g which were evaluated to 1 at an iteration of Algorithm 1 which is strictly before the iteration where g was evaluated to 1. By definition of that algorithm, Γ is in the output of Algorithm 1, and hence, by minimality of g, it is in the output of Algorithm 3. Hence, $\Gamma \subseteq T_n$, hence $\Gamma \subseteq T_{n-1}$ by the claim that we showed. Let us use this to study what happened during the last iteration of Algorithm 3 and reach a contradiction.

As g is an internal gate, let $j \in \{1, 2\}$ such that $g \in G_j$, and let us show that $g \in V_{n,j}$, by considering the invocation $\mathrm{SubEval}(C, CST \cup U_{n,j}, G_j)$. It suffices to show that there is some $l \in \mathbb{N}$ such that $\Gamma \subseteq Z_l$. Indeed, if this is the case, then, taking a minimal such l, we can consider the $(l+1)$-th iteration of the loop of SubEval: this iteration takes place, because Z_l has just changed by addition of some gate in Γ (or, if $l = 0$, it takes place by definition of a **do... while** loop), and in this iteration we know that we have $g \in Z_{l+1}$, because g was set to 1 in Algorithm 1 on the basis of Γ, and the iterations in SubEval are defined in the same way on G_j. So let us show that $\Gamma \subseteq Z_l$ for some $l \in \mathbb{N}$. As the Z_l are monotone, we can show this gate-per-gate: so let $g' \in \Gamma$ and show that it is in Z_l for some $l \in \mathbb{N}$.

We do a case disjunction. First, if $g' \in CST$, then we have $g' \in Z_0$, so there is nothing to show. Second, assume that $g' \in G_j$. We use the fact that $g' \subseteq T_{n-1}$ and $g' \in G_j$ to deduce that, by definition of T_{n-1}, we must have had $g' \in \mathrm{SubEval}(C, CST \cup U_{n-1,j}, G_j)$. Now, as $U_{n-1,j} \subseteq U_{n,j}$, the monotonicity of SubEval ensures that we also have $g' \in \mathrm{SubEval}(C, CST \cup U_{n,j}, G_j)$. Hence, there is indeed $l \in \mathbb{N}$ during this evaluation such that $g' \in Z_l$. Third, assume that $g' \in G_{3-j}$. As g' is an input gate of g, and $g \in G_j$, this implies that $g' \in F_j(G_1 \sqcup G_2)$. Hence, as $\Gamma \subseteq T_{n-1}$, we have $g' \in U_{n,j}$, hence, $g' \in Z_0$.

This shows that, indeed, in the last iteration ($i = n$) of Algorithm 3, for j defined such that $g \in G_j$, there is some iteration of the corresponding invocation of SubEval where all gates of Γ are added to Z_i, so that we must add g to Z_{i+1}. Hence, g is in the output of Algorithm 3, which contradicts our initial assumption and concludes the proof. ◀

We are now ready to prove Theorem 58:

Proof of Theorem 58. We will not define α explicitly, but it will be apparent from the proof that a suitable α can be defined. Note that, up to adding a fresh gate to serve as the new distinguished output gate, we can clearly ensure that the distinguished output gate g_0 has exactly one input, and that it is not an input gate to any other gate of C, and we can modify T accordingly. We start by computing in time $O(f(k) \cdot |C|)$, for some suitable exponential f, a tree decomposition T of C [Bod96]. We use Lemmas 53, 55 and 57 to ensure that, up to increasing the degree of the polynomial in the exponent of f, we can assume that the monotone cycluit C is arity-two and that the tree decomposition T is in normal form. We choose to root the normal-form tree decomposition T at the bag $\varphi(g_0)$ where g_0 is the distinguished output gate. The resulting T is such that φ maps each gate to a leaf bag, except the distinguished output gate g_0 which is mapped to the root; however, from our initial modification of C and the fact that T is scrubbed, we know that g_0 only occurs at the root (specifically, the root bag contains g_0 and its one input gate). Hence, in the sequel, we will never apply φ to g_0, and so we see φ as a mapping to the leaf bags of T. To simplify the induction, we ensure that the root bag also has two children, by adding to it a child with empty domain; this does not break any of the conditions on T.

We must now create the circuit C' and its tree decomposition T'. In fact, we will create a *pre-tree-decomposition* T' of C', namely, a tree with same skeleton as T, where each bag contains at most 2^{k^α} gates of C', and which satisfies the following conditions: first, all wires in C' are either between nodes of a same bag, or from nodes of a bag to nodes of a parent bag (i.e., all wires go upward in T'); second, each gate only occurs in the bag where it is created, except the gates of $C'_{\mathsf{inp}} = C_{\mathsf{inp}}$, whose occurrences in T' will correspond to a subset of the bags of T where they occur. The pre-tree-decomposition T' can clearly be rewritten to a tree decomposition in linear time and with a blowup of a factor 3 in the width: first, by adding the input gates to the bags corresponding to all their actual occurrences in T, and second by adding each gate of T' to its parent bag, which ensures that all wires are indeed covered. Hence, we will only describe how to construct a pre-tree-decomposition T' of C'.

We will create C' and T' by a bottom-up traversal of T. At each bag b of T, we will denote by T_b the subtree of T rooted at b, we will write $\mathrm{dom}(T_b)$ to refer to all gates occurring in T_b, and will write C_b to mean the sub-cycluit of C obtained by restricting to the gates of $\mathrm{dom}(T_b)$ and a subset of the wires between them, namely, the wires whose target g is such that $\varphi(g) \in T_b$. In other words, when $g \in \mathrm{dom}(T_b)$ but $\varphi(g)$ is not in T_b, then g occurs in C_b but it has no inputs, even though it may be the case that both inputs also occur in $\mathrm{dom}(T_b)$. Further, the inputs of C_b are then precisely $C_{\mathsf{inp}} \cap \mathrm{dom}(T_b)$. Throughout the proof, for each bag b, we will partition the gates of $\mathrm{dom}(b)$ in two sets (clearly computable in linear time):

- The *upward gates*, written $\mathrm{dom}^{\uparrow}(b)$, which are the gates g such that $\varphi(g)$ is a descendant of b, as well as all the input gates g in $\mathrm{dom}(b)$ and all constant gates g in $\mathrm{dom}(b)$.
- The *downward gates*, written $\mathrm{dom}^{\downarrow}(b)$, which are all other gates.

Intuitively, when processing the bag b, the *upward gates* are those which can be evaluated from other gates of T_b (and from the valuation of the input gates) based on what happens in T_b. (However, this evaluation may depend on downward gates of $\mathrm{dom}(b)$, so that we may be unable to perform it fully.) By contrast, the *downward gates* are those for which we have no complete information yet, and whose evaluation will take place later

To rewrite each C_b to a circuit when processing T, what is relevant to us is the behavior of C_b for each valuation of its inputs, in terms of how the upward gates evaluate *depending on the value of the downward gates*. Intuitively, to rewrite C_b, we need to describe the "function" that gives us the value of the upward gates (which is propagated to parent bags

that we will rewrite later) depending on each possible valuation of the downward gates (for which we do not know the value). Specifically, for any subset $S^+ \subseteq \mathrm{dom}^\downarrow(b)$, letting $S := (S^+, \mathrm{dom}^\downarrow(b) \backslash S^+)$, for any gate $\gamma \in \mathrm{dom}^\uparrow(b)$, and for any valuation ν of the inputs of C_b, we wish to know whether γ evaluates to 1 in $\nu(\rho_S(C_b))$. Recalling the definition of partial assignments (Definition 59), note that this is well-defined because S^+ contains no input gates. For brevity we write $\Pi_b := 2^{\mathrm{dom}^\downarrow(b)} \times \mathrm{dom}^\uparrow(b)$

We are now ready to describe the inductive rewriting process. For each bag b of T, we will create a bag b' of T' containing gates of C'. Remember that the circuit C' must have the same inputs as C; in particular we will ensure that, for any $b \in T$, the cycluit C_b and the circuit $C'_{b'}$ have the same inputs; further, by definition of the pre-tree-decomposition T', all wires will go upward in T', meaning that, for any $b' \in T'$, the circuit $C'_{b'}$ can always be fully evaluated from a valuation of its inputs. From this observation, we will inductively guarantee that each bag b' in T', corresponding to bag b in T, will contain one gate $g_b^{S^+,\gamma}$ for each $(S^+, \gamma) \in \Pi_b$, with the following semantics: for any valuation ν of the inputs of C_b, the gate $g^{S^+,\gamma}$ evaluates to 1 in $\nu(C'_{b'})$ iff γ evaluates to 1 in $\nu(\rho_S(C_b))$ where $S := (S^+, \mathrm{dom}^\downarrow(b) \backslash S^+)$.

We first describe the base case of the construction. Remember that, as T is scrubbed, each leaf bag of T is either in the image of φ or has empty domain. For a leaf bag b of T with empty domain, there is nothing to do. Otherwise, let g be the preimage of b by φ, i.e., the unique g (because φ is injective, and it is surjective on leaf bags with non-empty domain) such that $b = \varphi(g)$. We then have $\mathrm{dom}^\uparrow(b) = \{g\} \cup (C_{\mathsf{inp}} \cap \mathrm{dom}(b))$ and $\mathrm{dom}^\downarrow(b) = \mathrm{ins}(g) \setminus C_{\mathsf{inp}}$; observe that, as C is arity-two, it has no self-loops, so that $\mathrm{ins}(g)$ and $\{g\}$ are disjoint. If g is an input gate, then $\mathrm{ins}(g) = \emptyset$ and the only gate to create in b' is $g_b^{\emptyset,g}$ which can simply be taken to be g itself and satisfy the required conditions. If g is not an input gate, the gates to define are $g_b^{S^+,g}$ for all $S^+ \subseteq \mathrm{ins}(g) \backslash C_{\mathsf{inp}}$ (which has size at most 2), and we can simply realize this following the truth table of the operation $\mu(g)$ of g, i.e., by setting $g_b^{S^+,g}$ to be a sub-circuit with input $\mathrm{dom}(b) \cap C_{\mathsf{inp}}$ whose value is the value of g in $\rho_S(C_b)$ with $S = (S^+, \mathrm{ins}(g) \setminus (C_{\mathsf{inp}} \cup S^+))$.

We now describe the inductive case of the construction. We consider an internal bag b which has two children b_1 and b_2 in T; we write b', b'_1, b'_2 the corresponding bags in T', with b'_1 and b'_2 being already inductively defined. In particular, we know that b'_1 and b'_2 contain gates (which for brevity we will write $g_1^{S^+,\gamma}$ and $g_2^{S^+,\gamma}$, rather than $g_{b'_1}^{S^+,\gamma}$ and $g_{b'_2}^{S^+,\gamma}$) with the prescribed properties; and we will omit for brevity the b subscript when defining gates of b' in C' standing for the current bag b of T. For i from 0 to $(k+1)+2$, we will now define gates $g^{S^+,\gamma,i}$ in b' as follows, for all $(S^+, \gamma) \in \Pi_b$:

- Case of constant gates: if γ is a 1-gate (resp., a 0-gate), then the gate $g^{S^+,\gamma,i}$ is a 1-gate (resp., a 0-gate) for all $0 \leqslant i \leqslant k+3$.
- Case of input gates: if γ is an input gate, then the gate $g^{S^+,\gamma,i}$ is the same input gate for all $0 \leqslant i \leqslant k+3$.
- Base case: the gate $g^{S^+,\gamma,0}$ is simply a 0-gate.
- Induction: letting j be such that $\varphi(\gamma)$ is a descendant of b_j (so that $\gamma \in \mathrm{dom}^\uparrow(b_j)$ and $\gamma \in \mathrm{dom}^\downarrow(b_{3-j})$ if $\gamma \in \mathrm{dom}(b_{3-j})$), the gate $g^{S^+,\gamma,i}$ for $i \geqslant 1$ is an OR-gate, for all $\Gamma' \subseteq \mathrm{dom}^\downarrow(b_j) \cap \mathrm{dom}^\uparrow(b_{3-j})$ (which is necessarily $\subseteq \mathrm{dom}(b)$), of the AND of $g^{S^+,\gamma',i-1}$ for all $\gamma' \in \Gamma'$ and of $g_j^{(S^+ \cap \mathrm{dom}^\downarrow(b_j)) \cup \Gamma',\gamma}$.

We simply let $g^{S^+,\gamma}$ to be $g^{S^+,\gamma,k+3}$. It is clear that we create only exponentially many gates at each bag of T', so that we can obey the prescribed treewidth bound (in addition to the transformations of C that we have performed at the beginning, namely, transformation to

arity-2, and up to the use of a normal-form tree decomposition which implies that the bag size may be multiplied by 3).

We now show the correctness of this construction at b, i.e., we check inductively whether the gates that we create satisfy their prescribed semantics. Fixing $(S^+, \gamma) \in \Pi_b$ and a valuation ν of the inputs of C_b, we consider the monotone cycluit $\rho_S(C_b)$ and its evaluation under ν, where $S := (S^+, \mathrm{dom}^\downarrow(b) \backslash S^+)$ (note that this does not include any input gates because $\mathrm{dom}^\downarrow(b)$ does not). What we must show is that $g^{S^+,\gamma}$ evaluates to 1 under ν in C' iff γ evaluates to 1 in $\nu(\rho_S(C_b))$. To show this claim, we must connect our construction of the $g^{S^+,\gamma,i}$ to the evaluation of the cycluit $\nu(\rho_S(C_b))$; we will do so using the notion of an evaluation partition, by defining one from the structure of T_b.

Specifically, consider the evaluation partition $G_1 \sqcup G_2$ of the internal gates of $\rho_S(C_b)$ defined by putting in G_1 the internal gates whose image by φ is a descendant of b_1, and in G_2 those whose image is a descendant of b_2. We show the following claims:

- *This partition indeed covers the internal gates of $\rho_S(C_b)$.* Indeed, all internal gates of C_b are either gates of $\mathrm{dom}(T_1) \backslash \mathrm{dom}(b)$ (in which case their φ-image must be in T_1), or gates of $\mathrm{dom}(T_2) \backslash \mathrm{dom}(b)$ (same reasoning), or gates of $\mathrm{dom}(b)$ in which case they must be in $\mathrm{dom}^\uparrow(b)$ (as the gates of $\mathrm{dom}^\downarrow(b)$ are constant in $\rho_S(C_b)$) and then their φ-image is a descendant of b by definition. Now, the φ-image cannot be b itself as φ maps only to leaves, so it must be either a descendant of b_1 or of b_2.
- *For each $j \in \{1, 2\}$, we have $F_j(G_1 \sqcup G_2) = \mathrm{dom}^\uparrow(b_{3-j}) \cap \mathrm{dom}^\downarrow(b_j)$.* Indeed, for the forward inclusion, any gate g of $F_j(G_1 \sqcup G_2)$ is a gate of G_{3-j}, that is, $\varphi(g)$ is a descendant of b_{3-j}. Further, there is a wire (g, g') with $g' \in G_j$, that is, $\varphi(g')$ is a descendant of b_j, and g occurs in $\varphi(g')$. Now, as $g \in \mathrm{dom}(T_j) \cap \mathrm{dom}(T_{3-j})$, as T is a tree decomposition, we have $g \in \mathrm{dom}(b_j)$ and $g \in \mathrm{dom}(b_{3-j})$, and more specifically we have $g \in \mathrm{dom}^\uparrow(b_{3-j})$ and $g \in \mathrm{dom}^\downarrow(b_j)$ by what precedes. Conversely, any gate g of $\mathrm{dom}^\uparrow(b_{3-j})$ is such that $\varphi(g)$ is a descendant of b_{3-j}, so that $g \in G_{3-j}$. Further, as $g \in \mathrm{dom}(b_j)$, as the tree decomposition T is scrubbed, there is a descendant bag b of b_j where g appears in a φ-image, and as $\varphi(g)$ is not a descendant of b_j it must be the case that g appears in the φ-image of some g' (so that $g' \in G_j$) such that $g \to g'$ is a wire. Hence, $g \in G_{3-j}$ and g is an input of a gate in G_j, so indeed $g \in F_j(G_1 \sqcup G_2)$.
- *We have $|F(G_1 \sqcup G_2)| \leqslant k + 1$.* This follows from the previous point. Indeed, we have $|F(G_1 \sqcup G_2)| = |F_1(G_1 \sqcup G_2)| + |F_2(G_1 \sqcup G_2)| \leqslant \left|\mathrm{dom}^\uparrow(b_1)\right| + \left|\mathrm{dom}^\downarrow(b_1)\right| \leqslant |\mathrm{dom}(b)| \leqslant k + 1$ because T is a tree decomposition of width $\leqslant k$.

Now, consider the evaluation of $\rho_S(C_b)$ under ν using the evaluation partition $G_1 \sqcup G_2$, implemented with Algorithm 3: write $T_0(S), T_1(S), \ldots, T_{k+3}(S)$ the sequence of sets defined in the execution of the algorithm. (As this sequence may be shorter, if necessary we pad it until $T_{k+3}(S)$ by copying the last value.) Nested in our induction over T, we now show by induction on $0 \leqslant i \leqslant k + 1$ that, in b' as we have defined it, for any $S^+ \subseteq \mathrm{dom}^\downarrow(b)$ and $\gamma \in \mathrm{dom}^\uparrow(b)$, for any valuation ν of the inputs of C_b, the gate $g^{S^+,\gamma,i}$ evaluates to 1 under ν iff $\gamma \in T_i(S) \cap \mathrm{dom}^\uparrow(b)$. For the base case of $i = 0$, indeed $g^{S^+,\gamma,0}$ evaluates to true iff $\gamma \in T_0(S) \cap \mathrm{dom}^\uparrow(b)$ by definition of $T_0(S)$. For the induction step, observe that, by definition, $g^{S^+,\gamma,i}$ evaluates to true iff, letting j be such that $\varphi(\gamma)$ is a descendant of b_j, there is some $\Gamma' \subseteq \mathrm{dom}^\downarrow(b_j) \cap \mathrm{dom}^\uparrow(b_{3-j})$ (equivalently, by the second bullet point above, $\Gamma' \subseteq F_j(G_1 \sqcup G_2)$) such that $g^{S^+,\gamma',i-1}$ evaluates to 1 for all $\gamma' \in \Gamma'$ and $g_j^{(S^+ \cap \mathrm{dom}^\downarrow(b_j)) \cup \Gamma', \gamma}$ evaluates to 1.

For the forward direction of this induction claim (for the nested induction), assume the existence of such a Γ', and show that $\gamma \in T_i(S)$ (this suffices as we have $\gamma \in \mathrm{dom}^\uparrow(b)$ by

definition). By the induction hypothesis for the inner induction, as $g^{S^+,\gamma',i-1}$ evaluates to 1 for all $\gamma' \in \Gamma'$, we know that $\Gamma' \subseteq T_{i-1}(S)$. By the induction hypothesis for the outer induction, as $g_j^{(S^+ \cap \mathrm{dom}^\downarrow(b_j)) \cup \Gamma', \gamma}$ evaluates to 1, we know that γ evaluates to 1 in $\nu_j(\rho_{S_j}(C_j))$, where $S_j := ((S^+ \cap \mathrm{dom}^\downarrow(b_j)) \cup \Gamma', \mathrm{dom}^\downarrow(b_j) \backslash (S^+ \cup \Gamma'))$ and ν_j is the restriction of ν to the inputs of C_j. Now, using Lemma 61, consider the execution of Algorithm 3 (on $\rho_S(C_b)$ under ν), and consider the invocation of SubEval for this i and j. We know that CST contains all input gates of $\rho_{S_j}(C_j)$ set to 1 and all gates of S^+ (these are constant gates of $\rho_S(C_b)$), so they are in Z_0; and as $\Gamma' \subseteq F_j(G_1 \sqcup G_2)$ and $\Gamma' \subseteq T_{i-1}(S)$, we know that $\Gamma' \subseteq U_{i,j}$, so $\Gamma' \subseteq Z_0$. Now, as we know that γ evaluates to 1 under ν_j in $\rho_{S_j}(C_j)$, consider the evaluation of Algorithm 1 to witness this, and observe that, by what precedes, the invocation of SubEval that we are considering is such that Z_0 contains all gates which are initially true in $\nu_j(\rho_{S_j}(C_j))$, namely, the input gates that evaluate to 1, the constant 1-gates, and the gates of the first component of S_j, specifically, those of Γ' and a subset of those of S^+. Hence, each time Algorithm 1 sets a gate to 1, SubEval also does. Hence, γ is also returned by SubEval, so indeed $\gamma \in T_i(S)$, which concludes the forward direction.

For the backward direction of the nested induction claim, assume that $\gamma \in T_i(S) \cap \mathrm{dom}^\uparrow(b)$, and show the existence of a suitable Γ'. Let $j \in \{1,2\}$ be such that $\gamma \in \mathrm{dom}^\uparrow(b_j)$. We choose Γ' to be $T_{i-1}(S) \cap (\mathrm{dom}^\downarrow(b_j) \cap \mathrm{dom}^\uparrow(b_{3-j}))$, or, in other words, $U_{i,j}$ in Algorithm 3. Now, for any $\gamma' \in \Gamma'$, observe that $\gamma' \in \mathrm{dom}^\uparrow(b)$ (as $\gamma' \in \mathrm{dom}^\uparrow(b_{3-j})$, we know that $\varphi(\gamma')$ is a descendant of b_{3-j}, hence of b), so that $\gamma' \in T_{i-1}(S) \cap \mathrm{dom}^\uparrow(b)$, and by induction hypothesis for the inner induction we know that $g^{S^+,\gamma',i-1}$ evaluates to 1. Now we will use the induction hypothesis for the outer induction to argue that $g_j^{(S^+ \cap \mathrm{dom}^\downarrow(b_j)) \cup \Gamma', \gamma}$ evaluates to 1, by showing that γ evaluates to true under ν_j in $\rho_{S_j}(C_j)$, with the same notations for ν_j and S_j as for the forward direction above. To do this, as in the forward direction, we consider the evaluation of SubEval at iteration i and for the current j. In this evaluation, the gates that are initially true are the constant 1-gates (in particular those of S^+), the input gates that are true under ν, and the gates of $T_{i-1}(S) \cap F_j(G_1 \sqcup G_2)$, that is, of Γ'. In $\rho_{S_j}(C_j)$, all gates of Γ' are also true, and the other gates are also true whenever they appear in C_j: in other words, all gates that are in Z_0 at the beginning of this evaluation of SubEval are also initially true in $\nu_j(\rho_{S_j}(C_j))$ except if they are gates that do not occur in $\rho_{S_j}(C_j)$ (and have no inputs or outputs in $\rho_S(C_b)$). This property ensures that all gates that evaluate to true in SubEval, which are gates of G_j, can also evaluate to true in Algorithm 1 thanks to Lemma 61. Hence, indeed γ evaluates to true under ν_j in $\rho_{S_j}(C_j)$, so we conclude that indeed $g_j^{(S^+ \cap \mathrm{dom}^\downarrow(b_j)) \cup \Gamma', \gamma}$ evaluates to 1. This finishes the proof of the inner induction, and hence concludes the correctness proof of the induction step of the outer induction.

To conclude the proof of the Theorem, it suffices to observe that the root bag b'_{r} of the rewriting C' contains a gate $g := g_{\mathrm{r}}^{\emptyset, \{g_0\}}$ where g_0 is the output gate of C, whose semantics (by the outer induction claim) is guaranteed to be the following: for any valuation ν of C (hence, of C'), the gate g evaluates to true under ν in C' iff g_0 evaluates to true in $\nu(C)$. Hence, we set g as the output gate of C', and we have shown that C' is indeed equivalent to C. This concludes the proof. ◀

E.1.5 Isotropic rewritings of monotone cycluits

To prove Theorem 34 for non-monotone cycluits, we will of course perform an induction on the strata. The challenge when doing so is that, when rewriting a stratum, we may need to access the value of *all* gates of the lower strata, not just a single output gate. So, in the induction step on strata, it will not be sufficient to rewrite a stratum to a circuit with a

single output gate: we need to rewrite in a way where all gates of the circuit can be reused by lower strata. We define this formally:

▶ **Definition 62.** A circuit (or cycluit) C' *emulates* a cycluit C iff C and C' have the same input gates, all gates of C are gates of C', and for any valuation ν of the inputs, for any gate g of C, g evaluates to true in C under ν iff it does in C' under ν.

Similarly to how we modified the circuit at the beginning of the proof of Theorem 58, it will be convenient to rewrite the circuit in linear time to ensure that each gate of interest has exactly one input and is not itself the input to another gate. We call such gates the *potential outputs*:

▶ **Definition 63.** A *potential output* of a circuit (or cycluit) C is a gate g of C which has exactly one input gate and is not itself the input to any gate.

We can very simply rewrite our input cycluit so that all gates of interest are potential outputs:

▶ **Lemma 64.** *For any cycluit C of treewidth k, we can compute in linear time a cycluit C' of treewidth $\leqslant 2k$ that emulates C, such that, for each gate g of $C \setminus C_{\mathsf{inp}}$, the gate g in C' is a potential output.*

Proof. Simply build C' by renaming all gates of C that are not input gates to a different name. Now, for each gate g of C, add gate g to C' by setting it to be an AND-gate whose one input is the gate g' which is the renaming of g in C'. This clearly ensures that g evaluates to TRUE iff its renaming does, so that C' indeed emulates C. Further, the process is clearly in linear time and the treewidth bound is respected because we are simply duplicating each gate. ◀

This allows us to assume that we are working with a cycluit where all "relevant" gates are not the input gates to other gates. Thanks to this, the process that we really need for an induction on strata is the following variant of Theorem 58:

▶ **Theorem 65.** *There is an $\alpha \in \mathbb{N}^*$ s.t., for any monotone cycluit C of treewidth k, we can compute in time $O(2^{k^\alpha} \cdot |C|)$ a circuit C' that emulates C and has treewidth $\leqslant 2^{k^\alpha}$.*

Proof. . First note that, thanks to Lemma 64, we can process the input monotone cycluit C in linear time, keeping the parametrization in the treewidth, such that all gates of the original input cycluit are now potential output gates. Hence, it now suffices to construct a cycluit C' that *quasi-emulates* the cycluit C that we are working with, i.e., only emulates the potential output gates of C: indeed, C' will then emulate all gates of the original cycluit. As in Theorem 58, we will assume that C is arity-two, and that it has a normal-form tree decomposition T. Further, as in the proof of Theorem 58, we will construct a pre-tree decomposition T' of the result C', i.e., all wires are either within one bag or between a gate of a bag and a gate of a neighboring bag: T' will have same skeleton as T when forgetting about edge orientation.

We first notice that we can obtain the desired circuit C' in a naive way, which runs in quadratic time and does not satisfy the treewidth bound. To do so, we can simply apply Theorem 58, for each potential output g of C, on the tree decomposition T^b obtained by rooting T at $b := \varphi(g)$. We could then build the rewriting of C as a circuit C' that quasi-emulates C, by taking the union of the rewritings thus produced: these rewritings are disjoint except for the input gates. Further, we can construct a pre-tree decomposition T'

of C' by taking all the pre-tree decompositions produced in the applications of Theorem 58, forgetting about their orientation (so they have the same undirected skeleton as T^b, namely, that of T), taking their bag-per-bag union, and rerooting the result at an arbitrary bag to obtain a rooted tree decomposition.

Clearly the result T' of this process is still a pre-tree decomposition. First, all facts are still contained within bags or hold between one bag and its neighboring bag (as it did in the original pre-tree decompositions). Second, the occurrences of gates still form a connected subtree. Indeed, for input gates, this connected subtree is the same as in T, as is readily observed from the construction of Theorem 58. For the other gates, the connected subtree is the same as the original pre-tree decomposition from which they come. Further, Theorem 58 ensures that the resulting circuit C' quasi-emulates the cycluit C: note that C' is still a cycluit, because it is a union of acyclic circuits which is disjoint except for the input gates. The two problems are that C' (and hence the overall computation process) is of quadratic size, and that the treewidth of C' is no longer constant (each bag contains a linear number of groups of gates, each group having been produced by the application of Theorem 58).

To explain how these problems can be addressed, we recall the specifics of the construction used to prove Theorem 58. For each bag b of the rooted tree decomposition T'' to which it is applied, the construction creates a set of gates $g_b^{S^+,\gamma}$ for some subsets S^+ and for some γ (along with some intermediate gates, e.g., $g_b^{S^+,\gamma,i}$), whose inputs are gates $g_{b_1}^{\cdots}$ and $g_{b_2}^{\cdots}$ for the two children of b in T'', and these gates describe the behavior of the subtree of T'' rooted at b. In the process that we explained, for each internal bag b of the original normal-form tree decomposition T, we have considered b across all the rootings $T^{b'}$ of T for all b' being the φ-image of a potential output gate. For each such choice of root bag b', we have applied Theorem 58, so that in the bag b'' obtained as a final result of the process, we have one group of gates $g_b^{S^+,\gamma}$ (with their intermediate gates) for each choice of b'. However, the crucial observation is that there are only three possible groups, up to equivalence. Indeed, for each internal non-root bag b, we can classify the possible root bags b' depending on their orientation relative to b: as b has degree 3, there are only three possibilities. Specifically, for any two choices of root b'_1 and b'_2 that have the same orientation relative to b, the gates $g_b^{S^+,\gamma}$ were created for the same directed subtree of T rooted at b, so they are equivalent: the same is in fact true of all the intermediate gates at b, and indeed all gates for all descendants of b according to this orientation. In other words, the bottom-up construction of Theorem 58 at node b does the exact same thing for b'_1 and for b'_2.

From this observation, we can rewrite the result of the naive process to have linear size and to satisfy the treewidth bound, by taking each bag b of T, considering the $O(C)$ groups of gates created in the analogue b'' of b in the pre-tree decomposition T'' obtained as a result of the naive process, and merge these groups depending on the orientation of the root used for each group relative to b, so that only three groups of gates remain. This ensures that the result has linear size, and that the width bound is respected (i.e., the width is only multiplied by a factor of 3 relative to the width in the output of Theorem 58). By what precedes, the gates that we merge are equivalent, and, as they occur in the same bags of T'', the result is still a pre-tree decomposition. Further, the result of this process is still an acyclic circuit: indeed, if we perform the merges bottom-up following some rooting of T'', it is clear that whenever we merge two gates then they have the same input gates, so no cycles can be introduced. Hence, the result of this process is a linear-sized circuit obeying the treewidth bound, and it still quasi-emulates C, so it satisfies all of the required conditions.

The last thing to justify is that, instead of performing the quadratic-time construction and merging the equivalent gates, we can directly construct the final output with the merged

gates in linear time. We now explain how to do so. For each bag b of T and each neighbor b' of b in T, we call $S_b(b')$, the gates to create at bag b for the rootings of T at a φ-image whose unique simple path to b goes via b'. We then start by performing the entire bottom-up construction of Theorem 58 for some rooting T^{b_r} of T, where b_r is the φ-image of some arbitrary gate: for each bag b, this creates the gates $S_b(b')$ for b' the parent of b in T^{b_r}. Second, we perform an analogous process in a top-down pass on the tree T to reach the leaves, i.e., the φ-images of the other output gates, so as to cover all the other possible rootings. We explain what the top-down process does on a bag b with parent b' and children b_1 and b_2 in T^{b_r}. Recalling that $S_b(b')$ has already been created, the top-down process must create $S_b(b_1)$ and $S_b(b_2)$. Recall that the gates of $S_b(b_1)$ will stand for the subtree rooted at b with children b' and b_2: their inputs are gates of $S_{b_2}(b)$ and gates of $S_{b'}(b)$. The first kind of gates have already been created by the bottom-up process at b_2, and the second kind of gates have been created previously in the top-down process, because b' was visited before b. So the top-down process simply creates the gates of $S_b(b_1)$ as in the inductive step of the construction of Theorem 58 from these inputs. Likewise, the inputs to the gates of $S_b(b_2)$ are gates of $S_{b_1}(b)$ and gates of $S_{b'}(b)$, which have already been created. We can see that this process creates the same gates in each bag as those created by the naive process, except that we do not create the multiple equivalent sets of gates, so its output satisfies the required conditions. Further, it is easy to see that this process is in linear time, at it consists of one bottom-up application of Theorem 58, and one top-down traversal of the tree while performing two times at each node the inductive step of the construction of Theorem 58. This concludes the proof. ◀

E.1.6 Rewriting stratified cycluits

We now use Theorem 65 to conclude the proof of Theorem 34:

Proof of Theorem 34. Given a stratified cycluit C of treewidth k, assuming (up to adding an additional gate) that its output gate is a potential output (recall Definition 63), we transform C in linear time to an arity-two cycluit by the same construction as in the proof of Lemma 53. Indeed, this construction still applies to non-monotone cycluits, because it does not rely on the semantics of the gates except the associativity of OR- and AND-gates (NOT-gates have arity 1 so they do not need to be rewritten); and the construction clearly does not affect the fact that the cycluit is stratified. We now compute a stratification function for C in linear time by Proposition 48. We write $C_1, \ldots, C_m$ the strata of C, and write $C_{\leqslant i} := \bigcup_{p \leqslant i} C_p$ for brevity. Analogously to the proof of Lemma 64, we now rewrite C in linear time by re-wiring it in the following way: whenever a wire connects a gate g of stratum i to a gate g' of a stratum $j > i$, we introduce an intermediate AND-gate on this wire assigned to the stratum i, so that g is a potential output gate of $C_{\leqslant i}$. It is clear that this only doubles the size of bags in the tree decomposition. This process is intuitively designed to ensure that, whenever we have rewritten a prefix of the strata to a circuit C' that emulates it (using Theorem 65), the rewriting of the next stratum can proceed using only the gates of C'.

We now compute in linear time a normal-form tree decomposition T of C as in Lemma 57: clearly this still applies to non-monotone cycluits, as it does not depend on the semantics of gates. Clearly T is still a tree decomposition of any union $\bigcup_{p \leqslant i}$ of lower strata, and, if we restrict the bags to the gates of this union, it is still a normal-form tree decomposition.

We now describe the final rewriting process by induction on the strata of C. The invariant is that, once we have processed the strata $C_1, \ldots, C_i$, we have obtained a circuit C'_i which

emulates $C_{\leqslant i}$, with a tree decomposition $T'_{\leqslant i}$ with same skeleton as T where, for each bag b', letting b be the corresponding bag in b, $|\mathrm{dom}(b')|$ is within some factor (depending only on k, and in $O(2^{k^\alpha})$ for some constant α) of $|\mathrm{dom}(b) \cap G_i|$, where G_i are the gates of $C_{\leqslant i}$, and where each potential output gate g of $C_{\leqslant i}$ occurs at least in the same bags as it does in T. To achieve the overall linear running time, it suffices to ensure that each stratum C_i is processed in time linear in C_i and in the union of connected subtrees of T (which we can precompute for each i) that contains gates of stratum C_i. The reason why this ensures linear running time overall is that each bag of T is visited a linear number of times for each stratum that includes a gate in it, the number of which is a constant depending only on k. Note that, because of the fact that we only visit for each stratum the union of connected subtrees of T containing gates of this stratum, the tree decomposition $T'_{\leqslant i}$ that we compute at each stratum is technically not fully materialized (we do not materialize some parts where all bags have empty domain), but by a slight abuse we will nevertheless see it as having same skeleton as T.

The base case is that of the empty circuit, and of a tree decomposition with same skeleton as T and empty bags (which of course we do not need to materialize).

For the induction step, we consider stratum C_i, and let T_i be the normal-form tree decomposition of C_i with same skeleton as T obtained by restricting the bags of T to the gates of C_i. By definition of a tree decomposition, the bags with non-empty domains form connected subtrees of T_i, which we can process independently (as in this case the corresponding gates have no wires connecting them in C_i), and to which we can restrict our attention to ensure the linear time requirement for stratum i. To represent in C_i the missing gates from lower strata (remembering that they are guaranteed to be potential output gates in $C_{\leqslant i-1}$), we add these gates as *input gates*, putting them in C_i, and putting each of them in T_i only at the leaf bags which are φ-images of a gate that uses one of them as input.

Now, we can apply Theorem 65 to each disjoint part, the result of which is a circuit C'_i that emulates C_i, with a (not fully materialized) tree decomposition T'_i having same skeleton as T; and clearly the bags that were empty in T_i remain empty in T'_i, and the size of the others depends on the size of the corresponding bags in T_i by a constant factor depending only on k and satisfying the prescribed bound, as can be observed from the construction of Theorems 58 and 65. (Note that, in this accounting, we ignore the input gates that we added, as they will be accounted for by their occurrence in the lower strata.) We now ensure that all potential output gates of C_i, which for now only appear in the bag of T'_i corresponding to their φ-image in T_i (i.e., in T), occur in all bags where they appear in T_i; this does not affect the width requirement, as each gate thus added to a bag of T'_i is added for a gate (namely, the same gate) that was in the corresponding bag in T_i, and for which no gate had been added in T_i yet.

What remains to be done is to connect this rewriting C'_i of C_i to the rewriting $C'_{\leqslant i-1}$ obtained by induction for the lower strata, and produce the circuit $C'_{\leqslant i}$ and tree decomposition $T'_{\leqslant i}$ satisfying the conditions. We simply do this by unioning C'_i with $C'_{\leqslant i-1}$, and substituting to the input gates added to C'_i the actual gates of $C'_{\leqslant i-1}$: as $C'_{\leqslant i-1}$ emulates $C_{\leqslant i}$, it emulates these gates, as they must be potential output gates of $C_{\leqslant i}$ thanks to the rewriting that we perform analogously to Lemma 64. It is clear by composition that $C'_{\leqslant i}$ has the correct semantics, and of course this kind of substitution does not affect the acyclicity of $C'_{\leqslant i}$. The tree decomposition $T'_{\leqslant i}$ is constructed by unioning T'_i with $T'_{\leqslant i}$ as computed in the induction (this only needs to traverse the bags where T'_i is materialized), and substituting the input gates; the fact that the result is still a tree decomposition of the result is clear for the occurrences of the gates of stratum i (which are the same as in T'_i), and of the occurrences

of the gates of lower strata, because each time a fresh input gate is substituted to a gate g' of a lower stratum, the construction ensures that this is within a bag which is the φ-image of a gate g of stratum i with g' as input, and the occurrence of g' in the corresponding bag in T ensures that g' must occur in that bag in $T'_{\leqslant i-1}$. Likewise, $T'_{\leqslant i}$ covers all wires, because this needs only be checked for the wires across strata, but it is the case for the same reason as we just explained. This concludes the proof of the induction case.

We now check that the claimed treewidth bound of $O(2^{k^\alpha})$ for some α is respected, by accounting for the transformations that we have performed:

- We have added one additional gate to the input cycluit at the very beginning.
- We have transformed it to arity-2 as in Lemma 53, so squaring the bag size.
- We have applied a process analogous to that of Lemma 64, which doubles the bag size.
- We have chosen the tree decomposition to be regrouped as in Lemma 55, which multiplies the bag size by 3, then normal-form as in Lemma 55, which does not change the bag size.
- Now, we have performed the induction on strata, multiplying the bag size by at most $2^{k^{\alpha'}}$ for some α', using Theorem 65.

Hence, we can clearly respect the prescribed bound of $O(2^{k^\alpha})$ for some α.

The result of the induction is indeed a circuit C' which emulates $C_{\leqslant m} = C$ (in particular, it emulates the output gate of C, as it was a potential output gate), with a tree decomposition T' where each bag size is within the requested bounds, and computed in overall linear time.

This concludes the proof of Theorem 34. ◀

E.2 Hardness of PQE

▶ **Proposition 36.** *There is a fixed arity-two signature on which PQE is #P-hard even when imposing that the input instances have treewidth* 1 *and the input queries are α-acyclic CQs.*

Proof. We reduce from the #P-hard problem #MONOTONE-2-SAT [Val79], that asks, given a conjunction Φ of disjunctions of positive literals over variables $x_1, \ldots, x_n$, the number of assignments that satisfy Φ.

We consider a signature σ formed of a unary relation R, and of three binary relations V, K, and C.

Given Φ, we encode it in PTIME to a TID instance (I, π) which comprises the following facts:

- One fact $R(r)$.
- For each variable v_i of Φ, the fact $V(r, v_i)$ with probability $1/2$ in π, which intuitively codes the valuation of variable v_i.
- For each clause C_j of Φ that contains variable v_i, the following gadget:

$$\text{a length-}j\text{ path } K(v_i, c_{i,j,1}), K(c_{i,j,1}, c_{i,j,2}), \ldots, K(c_{i,j,j-1}, c_{i,j,j}) \quad \text{and the fact } C(c_{i,j,j}, c'_{i,j,j}),$$

  each of these facts having probability 1 in π. Intuitively, to remain on a fixed signature, we write the clause number in unary as the path length.

It is immediate that I is a tree, so it has the prescribed treewidth. We now construct in PTIME the query Q comprising the following atoms:

- One atom $R(x)$.

- For each clause C_j, one atom $V(x, y_j)$, one length-j K-path from y_j to a variable $y_{j,j}$ and the fact $C(y_{j,j}, y'_{j,j})$.

Again, it is immediate that Q is acyclic.

We now claim that the probability of Q on (I, π) is exactly the number of satisfying assignments of Φ divided by 2^n, so that the computation of one reduces in PTIME to the computation of the other, concluding the proof. To see why, we define a bijection between the valuations of $v_1, \ldots, v_n$ to the possible worlds J of (I, π) in the expected way: retain fact $V(r, v_i)$ iff v_i is assigned to true in the valuation. Now, observe that J satisfies Q iff the corresponding valuation makes Φ true. Indeed, from the satisfaction of Q by J, for any j, observing the element to which $y_{j,j}$ is matched, and observing its ancestor v_{i_j} in I, we deduce that v_{i_j} must be made true by the valuation, and, by construction of I, the variable v_{i_j} appears in C_j, hence C_j is true in Φ. Hence, Φ is true because every clause is true. Conversely, assuming that the valuation satisfies Φ, we construct a match of Q on I by mapping each branch of Q via the witnessing variable for that clause in the valuation of Φ. This concludes the proof. ◀

▶ **Proposition 37.** *There is a fixed arity-two signature on which PQE is #P-hard even when imposing that the input instances have treewidth* 1 *and the input queries are path queries.*

Proof. We adapt the previous proof. We extend the signature to include one binary relation S_- for each binary relation S. We modify the definition of the instance I to add, whenever we created a fact $S(a, b)$, the fact $S_-(b, a)$, each of these inverse facts being given probability 1 in π. It is clear that I still has treewidth 1, as we can just use the same tree decomposition as before. We now define the path query Q' as follows, following a traversal of the query Q of the previous proof:

- $V(x_1, y_1), K(y_1, y_{1,1}), C(y_{1,1}, y'_{1,1}), C_-(y'_{1,1}, y''_{1,1}), K_-(y''_{1,1}, y''_1), V_-(y''_1, x_2)$;
- $V(x_2, y_2), K(y_2, y_{2,1}), K(y_{2,1}, y_{2,2}), C(y_{2,2}, y'_{2,2}), C_-(y'_{2,2}, y''_{2,2}), K_-(y''_{2,2}, y''_{2,1})$, $K_-(y''_{2,1}, y''_2), V_-(y''_2, x_3)$;
- etc.

It is straightforward to observe that, when inverse facts are added like we did, Q has a match M on I iff Q' has this same match: constructing the match of Q' from that of Q is trivial, and any match of Q' is a match of Q thanks to the fact that, I being a tree, each element of $\mathrm{dom}(I)$ has at most one ingoing inverse fact, so that each inverse fact must in fact be mapped to the same element as the corresponding fact that was traversed earlier. This concludes. ◀

F Results from [BBGS16]

This appendix contains some results from [BBGS16], an extended version of [BBS12] that deals with the containment of monadic Datalog programs, but which is currently unpublished. Relevant parts of this work are reproduced here for completeness.

F.1 Treelike canonical set of instances of a Datalog program

▶ **Definition 66.** A *canonical set of instances* for a Datalog program P is a (generally infinite) family $\mathcal{I}$ of instances that all satisfy P and such that, for any CQ Q, if there is an instance I satisfying $P \wedge \neg Q$, then there is an instance in $\mathcal{I}$ with the same property.

▶ **Definition 67.** The *var-size* of a Datalog program P is the maximal number of variables used in a rule of P.

The following result from [BBGS16] shows that any Datalog program has a bounded-treewidth set of canonical instances, an encoding of which can be described by a bNTA constructible in exponential time. It heavily relies on notions introduced in [CV97].

▶ **Lemma 68.** *For all $c \in \mathbb{N}^*$, letting $k_{\mathrm{I}} := 2c - 1$, for any signature σ, given a Datalog program P of var-size bounded by c, we can compute in exponential time in P a $\Gamma_\sigma^{k_{\mathrm{I}}}$-bNTA A_P such that the set of the instances obtained by decoding the trees in the language of A_P is canonical for P.*

Proof. This proof is based on the notion of *unfolding expansion tree* of a Datalog program P (Definition 2.4 of [CV97]).

An *expansion tree* of a Datalog program is a ranked tree (non-binary in general) where each node is labeled by an *instantiated* rule r of P (i.e., an homomorphic image of the rule by some *one-to-one* mapping from the variables of the rule to some set of variables), and has a child for each intensional predicate atom A appearing in r, whose label is a rule r' with A for head, with the variables of A mapped to the same variables as in r. We further require that the root of the tree has a rule with the goal predicate in the head.

An *unfolding expansion tree* is an expansion tree with the additional condition that, for each node n of the tree, each variable occurring in the body of the rule labeling n but not in its head does not occur in the label of any ancestor of n. In other words, the same variable is never re-used across rules unless the variable is propagated through the head.

From an unfolding expansion tree t, it is possible to get an instance over the extensional signature satisfying P as follows. Let ν be a one-to-one mapping from the variables in the rules labeling the nodes of t to constants. We denote by $\nu(t)$ the tree obtained by replacing each variable appearing in the labels of t by its image by ν. We note that t and $\nu(t)$ are identical up to renaming the values in variables. Let $\Pi_{\mathrm{ext}}(\nu(t))$ be the tree obtained by keeping only the facts over the extensional signature in each label of a node of $\nu(t)$. We denote by $I(\Pi_{\mathrm{ext}}(\nu(t)))$ the instance composed of the facts appearing in $\Pi_{\mathrm{ext}}(\nu(t))$. It is clear that $I(\Pi_{\mathrm{ext}}(\nu(t)))$ satisfies P: indeed, a simple bottom-up induction on $\nu(t)$ shows that it only contains intensional facts that are derivable by P from $I(\Pi_{\mathrm{ext}}(\nu(t)))$.

First note that, for P a Datalog program, the set of instances of the form $I(\Pi_{\mathrm{ext}}(\nu(t)))$, where t is un unfolding expansion tree of P and ν is some fixed one-to-one mapping to constants, is a canonical set of instances for P. This comes from Proposition 2.6 of [CV97]: the Datalog program is equivalent to the infinite disjunction of the $\Pi_{\mathrm{ext}}(t)$, where t is an unfolding expansion tree of P, each $\Pi_{\mathrm{ext}}(t)$ being seen as a conjunctive query. So for any CQ Q, if $I \models P \wedge \neg Q$, then in particular $I \models \Pi_{\mathrm{ext}}(t)$ for some unfolding expansion tree t. But then $I(\Pi_{\mathrm{ext}}(\nu(t))) \not\models Q$, since $I(\Pi_{\mathrm{ext}}(\nu(t)))$ is a canonical model of $\Pi_{\mathrm{ext}}(t)$ and $\Pi_{\mathrm{ext}}(t)$ has a model that does not satisfy Q.

In Section 5.1 of [CV97], the notion of *proof tree* is introduced. A proof tree for a Datalog program P is defined as an expansion tree over a finite fixed set of variables $\{x_1, x_2, \ldots, x_{2s}\}$ where s is the var-size of P.

In the proof of Proposition 5.6 of [CV97], it is shown that for every unfolding expansion tree t, there is an associated proof tree t' obtained by a mapping μ such that $t' = \mu(t)$. Now, observe that $\Pi_{\mathrm{ext}}(t')$ can be seen as a form of a tree encoding (though not of the same form as our tree encodings: the domain is a subset of the x_i's, facts are extensional facts of the instance) of $I(\Pi_{\mathrm{ext}}(\nu(t)))$ for any one-to-one mapping ν from variables to constants. This witnesses $I(\Pi_{\mathrm{ext}}(\nu(t))$ is of treewidth $\leqslant 2s - 1$.

For any Datalog program of var-size c, Proposition 5.9 of [CV97] shows that there exists a tree automaton of size exponential in P recognizing the proof trees of P (and the proof of this result makes it clear that this automaton is computable in exponential time). From such a tree automaton (running on the ranked expansion trees), by projection, we can construct in polynomial time in the size of the automaton a bNTA A recognizing the $\Pi_{\text{ext}}(t')$ for any proof tree t'. We now apply the technique of the proof of Proposition B.1 in [ABS15a] to transform A, in time polynomial in A, into a $\Gamma_\sigma^{k_\text{I}}$-bNTA A_P that recognizes (σ, k_I)-tree encodings of the $I(\Pi_{\text{ext}}(\nu(t)))$ for an arbitrary unfolding expansion tree t and one-to-one mapping ν to constants. A technical detail imposed by the technique of [ABS15a] is that we must ensure that each rule of the Datalog program has either 0 or 2 intensional facts; we can always do that by an initial transformation of the Datalog program, introducing intermediate intensional predicates for rules with more than 2 intensional facts or adding a trivial intensional predicate atom for rules with 1 intensional fact. Note that the var-size of the Datalog program is not affected by this transformation.

We have thus shown that it is possible to compute A_P in exponential time, and that the set of instances in the decoded language of A_P is canonical for P. ◀

F.2 2EXPTIME-Hardness of Treelike CQ Validity over Valid Trees

We consider "universality" or "validity" problems for queries over trees: given a schema describing a set of trees and a Boolean query over trees, does every tree satisfy the query.

Let Sch be a finite set of labels. The *relational signature of ordered, labeled, binary trees*, denoted $\mathcal{S}^{\mathsf{Bin}}_{\mathsf{Ch1},\mathsf{Ch2}}$, is made out of the binary predicates $\mathsf{FirstChild}$, $\mathsf{SecondChild}$, unary Root, Leaf predicates, and Label_α predicates for all $\alpha \in \mathsf{Sch}$.

We denote as $\mathcal{S}^{\mathsf{Bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child},\mathsf{Child}^?}$ the relational signature containing all the relations of $\mathcal{S}^{\mathsf{Bin}}_{\mathsf{Ch1},\mathsf{Ch2}}$ together with binary Child and $\mathsf{Child}^?$ relations.

A tree T over $\mathcal{S}^{\mathsf{Bin}}_{\mathsf{Ch1},\mathsf{Ch2}}$ is a relational instance such that:

(i) the non-empty Label_α^T's for $\alpha \in \mathsf{Sch}$ form a partition of $\mathrm{dom}(T)$ (one can thus talk about the *label* of a node n, which is the $\alpha \in \mathsf{Sch}$ such that $n \in \mathsf{Label}_\alpha^T$);

(ii) $\mathsf{FirstChild}^T$ and $\mathsf{SecondChild}^T$ are one-to-one partial mappings with the same domain (the set of *internal nodes*), whose complement is Leaf^T (the set of *leaves*), and with disjoint ranges;

(iii) the inverses of $\mathsf{FirstChild}^T$ and $\mathsf{SecondChild}^T$ are one-to-one partial mappings;

(iv) $\exists x\, \mathsf{FirstChild}(x,x) \vee \mathsf{SecondChild}(x,x)$ does not hold;

(v) Root^T contains exactly one element (the *root* r of T), and the following formula does not hold for r: $\exists x\, \mathsf{FirstChild}(x,r) \vee \mathsf{SecondChild}(x,r)$.

A *tree* T over $\mathcal{S}^{\mathsf{Bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child},\mathsf{Child}^?}$ is a relational instance that verifies the same axioms as a tree over $\mathcal{S}^{\mathsf{Bin}}_{\mathsf{Ch1},\mathsf{Ch2}}$, where Child^T is the disjoint union of $\mathsf{FirstChild}^T$ and $\mathsf{SecondChild}^T$, and where the following formula holds: $\mathsf{Child}^?(x,y) \leftrightarrow (\mathsf{Child}(x,y) \vee x = y)$.

A Boolean query on one of the signatures above is *valid* over a bNTA if for all trees that satisfy the schema, the query returns true.

We can now prove the following result, which closely tracks Theorem 6 of [BMS08a].

▶ **Theorem 69.** *Given a CQ Q on $\mathcal{S}^{\mathsf{Bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child},\mathsf{Child}^?}$ of treewidth $\leqslant 2$ and a bNTA A, it is 2EXPTIME-hard to decide whether Q is valid over A.*

Proof. We adapt the proof of Theorem 6 of [BMS08a], given in Appendix C.2 of [BMS08b], which states that validity with respect to a bNTA of a CQ with *child and descendant* predicates over *unranked trees* is 2EXPTIME-hard. We adapt it by moving from unranked trees to binary

trees (with the changes that it implies in our definition of a bNTA), writing the reduction using $\mathsf{Child}^?$ instead of the descendant predicate, and proving that the resulting CQ has bounded treewidth. The final CQ will be defined through intermediate subformulae, and we will use the following immediate observation to bound its treewidth: the treewidth of a CQ of the form $Q = \exists \mathbf{xy}\, Q_1(\mathbf{x}) \wedge Q_2(\mathbf{x}, \mathbf{y})$ is the maximum of the width of a decomposition of $\exists \mathbf{x}\, Q_1(\mathbf{x})$ where all variables of $\mathbf{x}$ are in the same bag, and of the treewidth of $Q' = \exists \mathbf{xy}\, R(\mathbf{x}) \wedge Q_2(\mathbf{x}, \mathbf{y})$ where $R(\mathbf{x})$ is a single atom.

We give a self-contained presentation keeping the notation from [BMS08a] as much as possible, with notable departures highlighted in **bold font** throughout the proof.

As in [BMS08a], we reduce from the termination of an alternating EXPSPACE Turing Machine M, a 2EXPTIME-hard problem [CKS81]. The next three paragraphs are taken in part from [BMS08a], with some minor simplifications, as we need to introduce the same concepts.

An alternating Turing machine (ATM) is a tuple $M = (\Omega, \Gamma, \Delta, q_0)$ where $\Omega = \Omega_\forall \uplus \Omega_\exists \uplus \{q_\mathrm{a}\} \uplus \{q_\mathrm{r}\}$ is a finite set of states partitioned into universal states from $\Omega_\forall$, existential states from $\Omega_\exists$, an accepting state q_a, and a rejecting state q_r. The finite tape alphabet is Γ. The initial state of M is $q_0 \in \Omega$. The transition relation Δ is a subset of $(\Omega \times \Gamma) \times (\Omega \times \Gamma \times \{\mathrm{L}, \mathrm{R}, \mathrm{S}\})$. The letters L, R, and S denote the directions left, right, and stay, according to which the tape head is moved.

An *accepting computation tree* for an ATM M is a finite *unranked* tree labeled by configurations (tape content, reading head position, and internal state) of M such that (1) if node v is labeled by an existential configuration, then v has one child, labeled by one of the possible successor configurations; (2) if v is labeled by a universal configuration, then v has one child for each possible successor configuration; (3) the root is labeled by the initial configuration (input word on the tape, head at the beginning of the word, initial state); and (4) all leaves are labeled by accepting configurations (and accepting configurations only appear as leaves). An ATM M accepts a word $w \in \Gamma^*$ if there exists an accepting computation tree for M with w as initial tape content.

The overall idea of the proof of [BMS08a], that we closely adapt, is as follows. Given an ATM M and a word w of length n, we construct, in polynomial time, (1) an ATM M_w which accepts the empty word if and only if M accepts w; and (2) a bNTA A that checks most important properties of (suitably encoded) computation trees of M_w, except their consistency w.r.t. the transition relation of M_w. The consistency is tested by a query Q that we define. To be precise, Q is satisfied by a tree T in $L(A)$ if and only if the transition relation of M_w is not respected by t. This means that Q is valid w.r.t. A, iff there does not exist a consistent, accepting computation tree for M_w. Since 2EXPTIME is closed under complementation, we conclude that validity of CQs on $\mathcal{S}^{\mathsf{Bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child},\mathsf{Child}^?}$ with respect to bNTAs is 2EXPTIME-hard.

Without loss of generality, we assume that universal states of M_w have exactly two successors, whatever the symbol read – if they have less, we can just add transition(s) to an accepting configuration, and if they have more, we can introduce intermediary states to encode the n-ary conjunction as a tree of binary conjunctions, with no change to the tape.

We do not give the non-deterministic tree automaton explicitly, but trees in its language will have **the shape represented by Figure 1.**[1] The bold nodes are the nodes added to

[1] **Since our definition of bNTA requires a binary tree to be full, we need to add dummy nodes where needed, with labels distinct from real nodes. This technicality has no impact,**

the tree of the proof of Theorem 19 of [BMS08a]. The labels of dashed edges indicate the number of nodes between a node and its ancestor. This bNTA encodes trees that represent the executions of M_w.

Each configuration in an execution is encoded by a subtree rooted by a node labeled CT; the CT-node for the initial configuration appears as the unique child of a chain of ℓ nodes with dummy labels from the root for some integer ℓ that we will define further (this chain of ℓ nodes is only needed for technical reasons). A CT-node has **two** children labeled r and **NextCT**. The subtree rooted by the r-node represents the tape of the configuration **and the subtree rooted by the NextCT-node has zero, one, or two CT-children representing dummy nodes where needed, that we will regard as zero, one, or two following configurations depending whether the current state is accepting, existential, or universal.**

A tape (under a node r) is written as a complete binary tree of depth n, with leaves of the tree containing information about the 2^n cells. This complete binary tree, that we will refer to as the *tape tree*, is itself encoded for querying purposes in the following fashion: for $1 \leqslant i \leqslant n$, a node with label s represents a node at depth i in the tape tree; each such node representing a node at depth i, with $1 \leqslant i \leqslant n-1$, has for first child a node of label p that serves as a *navigation widget* indicating the position of this node in the tape tree, and **as second child a node of label TTCh (for *tape tree children*) that has for children two nodes of labels *s***, encoding the two children of the current encoded node in the tape tree. The navigation widget is a p-labeled node with a single x-child that has itself a single y-child. If the current encoded node in the tape tree was a left child, $x = 0$ and $y = 1$; otherwise, $x = 1$ and $y = 0$ (this widget thus encodes the i-th bit of the address of a cell). The node r itself has two children, the two nodes labeled with s encoding the first level of the tape tree. Finally, nodes labeled with s representing a node at depth n in the tape tree also have a p-labeled navigation widget, but have as second child a c-node that encodes the content of the cell.

We encode the same information about configuration tapes as in [BMS08b]: symbols on basic cells, symbol and transition followed on the current cell, and current symbol, previous state, previous symbol on previous tape cells. This means each cell is virtually annotated with an element of $\Gamma \cup (\Gamma \times \Delta) \cup (\Gamma \times \Omega \times \Gamma)$: there are polynomially many such annotations, we refer to them in the following as $1, \ldots, k$ fixing an arbitrary order. As in [BMS08b], we want to impose a number of horizontal constraints (constraints on the annotations of neighboring cells in a given configuration) and of vertical constraints (constraints on the annotation of the same cell in successive configurations). These constraints can be written as two sets of pairs $H(M_w)$ and $V(M_w)$ of integers $1 \leqslant i, j \leqslant k$, respectively, indicating respectively whether j can appear to the right of i in a configuration, and whether j can appear in the same cell as i in a successive configuration. We refer to [BMS08b] for the full set of constraints required.

For each cell, the c-node has two children, labeled with m (for *me*) and f (for *forbidden*), each having as descendants a chain of k nodes that can have labels either 0 or 1. Only one node has label 1 under m: the one whose depth gives the current content of the cell. Under f, for a cell at position i in the tape, node at depth j has label 0 if and only if $(i, j) \in V(M_w)$.

As in [BMS08b], we can construct in polynomial time a bNTA that enforces that all trees have the described form, including respect of horizontal constraints, initial configuration at the root and accepting configuration at the leaves, but *excluding vertical constraints*.

and we will ignore these nodes.

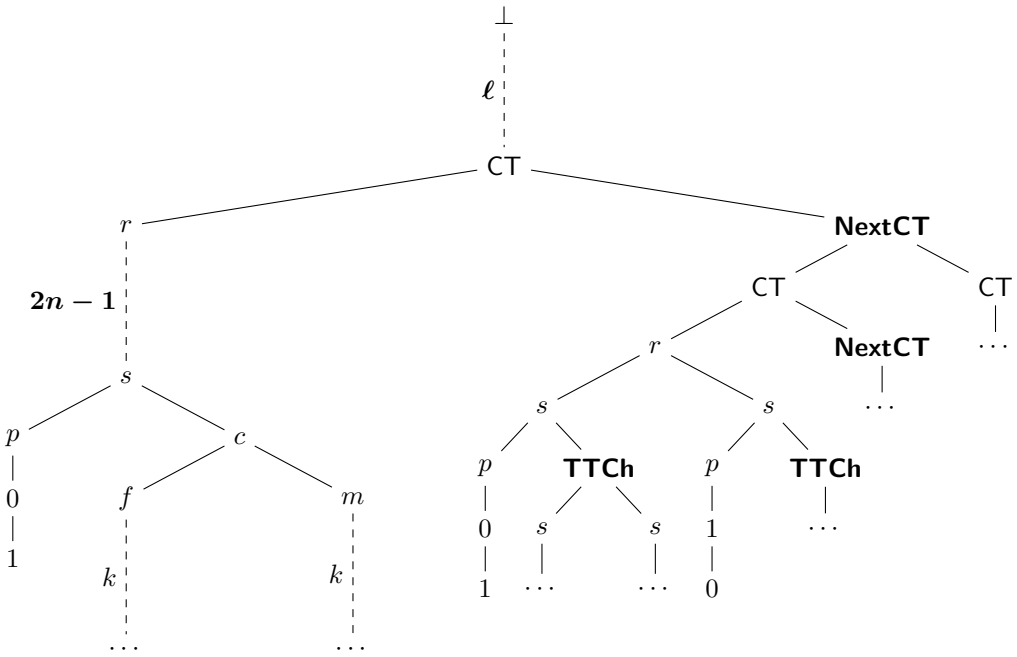

Figure 1 General structure of trees in proof of Theorem 69; bold labels and counters highlight changes from the proof of Theorem 6 of [BMS08a]

Indeed, vertical constraints cannot (at least straightforwardly) be imposed on the tree as they relate nodes of the tree that are very far apart – see [BMS08b] for how to encode horizontal constraints and the general structure. Modifications needed because of our binary setting are minor. The language of this bNTA is exactly the codes of accepting computation trees for M_w, except that vertical constraints may be violated.

We now construct a conjunctive query that holds if vertical constraints are violated. In what follows, we denote by $R^i(x, y)$ the chain $\exists x_1 \dots x_{i-1}\, R(x, x_1) \wedge \dots \wedge R(x_{i-1}, y)$ for R a binary relation and $i \geqslant 1$. The query is built up from the following subformulae:

- A formula $\mathsf{Succ}(r_1, r_2)$ that expresses that r_1 and r_2 are roots of a tree encoding tape, with the configuration of r_2 being the successor of that of r_1. Formally:

$$\mathsf{Succ}(r_1, r_2) := \exists s_1 s_2\, \mathsf{Label}_r(r_1) \wedge \mathsf{Label}_r(r_2) \\ \wedge \mathsf{Child}(s_1, r_1) \wedge \mathsf{Child}(s_2, r_2) \wedge \mathsf{Child}^{\mathbf{2}}(s_1, s_2).$$

  There is a tree decomposition of width 2 of this subquery where both exported variables r_1 and r_2 are in the same bag.
- A formula $\Phi_i(x, y)$ that expresses that x and y are s-nodes encoding a node at the i-th level of two tape trees, such that the configuration of y is a successor of the configuration of x:

$$\Phi_i(x, y) := \exists r_1 r_2\, \mathsf{Label}_s(x) \wedge \mathsf{Label}_s(y) \wedge \mathsf{Succ}(r_1, r_2) \\ \wedge \mathsf{Child}^{\mathbf{2i-1}}(r_1, x) \wedge \mathsf{Child}^{\mathbf{2i-1}}(r_2, y).$$

  There is a tree decomposition of width 2 of this subquery where both exported variables x and y are in the same bag.

- A formula $\Psi_i(x, y)$ that expresses that $\Phi_i(x, y)$ holds and that, additionally, x and y are both first children or both second children of their parents; note that we could not use FirstChild and SecondChild here as it would require disjunction. We can, however, use the navigation widgets:

$$\begin{aligned}\Psi_i(x,y) := \exists p_x p_y t_x t_y \boldsymbol{t'_x t'_y} z\ \Phi_i(x,y) \wedge \mathsf{Label}_p(p_x) \wedge \mathsf{Label}_p(p_y) \wedge \mathsf{Label}_1(t_x) \wedge \mathsf{Label}_1(t_y)\\ \wedge \mathsf{Child}(x,p_x) \wedge \mathsf{Child}(y,p_y) \wedge \boldsymbol{\mathsf{Child}(p_x, t'_x)} \wedge \boldsymbol{\mathsf{Child}(p_y, t'_y)}\\ \wedge \boldsymbol{\mathsf{Child}^?(t'_x, t_x)} \wedge \boldsymbol{\mathsf{Child}^?(t'_y, t_y)}\\ \wedge \mathsf{Child}^{\boldsymbol{(2i-1)+4}}(z,t_x) \wedge \mathsf{Child}^{\boldsymbol{(2i-1)+6}}(z,t_y)\end{aligned}$$

  Observe that when x and y are both first children, the t_x and t_y are grandchildren of the p-node, and therefore **at distance $(2i-1)+3$ of the r-node**, so going up $(2i-1)+4$ times brings us to the CT-node of the current configuration, and going up $(2i-1)+6$ times brings us to the CT-node of the preceding configuration. Similarly, if x and y are both second children, the t_x and t_y are children of the p-node, so going up $(2i-1)+4$ times brings us to the parent of the CT-node of the current configuration, and going up $(2i-1)+6$ times brings us to the parent of the CT-node of the preceding configuration. **This is one of the two places we need the chain of ℓ nodes at the root: otherwise, since the initial configuration does not have a preceding configuration, we would not be able to go high enough up in the tree to find the z node. Taking $\ell \geqslant 1$ suffices.**
  There is a tree decomposition of width 2 of this subquery where both exported variables x and y are in the same bag.
- A formula $\mathsf{SameCell}(s_1, s_2)$ that expresses that two s-nodes encoding a node at depth n in the tape tree (i.e., at the bottom of the tape tree) correspond to the same cell of successive configuration tapes:

$$\begin{aligned}\mathsf{SameCell}(s_1,s_2) := \exists x_1 \cdots x_{n-1} y_1 \cdots y_{n-1} \bigwedge_{1 \leqslant i < n-1} \left(\mathsf{Child}^{\boldsymbol{2}}(x_i, x_{i+1}) \wedge \mathsf{Child}^{\boldsymbol{2}}(y_i, y_{i+1})\right)\\ \wedge \mathsf{Child}^{\boldsymbol{2}}(x_{n-1}, s_1) \wedge \mathsf{Child}^2(y_{n-1}, s_2)\\ \wedge \Psi_n(s_1,s_2) \wedge \bigwedge_{1 \leqslant i < n} \Psi_i(x_i, y_i).\end{aligned}$$

  There is a tree decomposition of width 2 of this subquery where both exported variables s_1 and s_2 are in the same bag.

We can now use these subformulae in the following sentence, that expresses the final conjunctive query Q. It checks whether the two same cells s_1 and s_2 of successive configurations violate vertical constraints. Remember that the value of a cell is encoded under the m-node, while vertical constraints are encoded under the f-node. A vertical constraint occurs when the (unique) position of a 1-node under the m-descendant of s_2 is equal to the position of a 1-node under the f-descendant of s_1.

$$\begin{aligned}Q := \exists s_1 s_2 t_1 t_2 f_1 m_2 p_1 p_2 z\ \mathsf{SameCell}(s_1,s_2) \wedge \mathsf{Child}(s_1,t_1) \wedge \mathsf{Child}(s_2,t_2)\\ \wedge \mathsf{Child}(t_1,f_1) \wedge \mathsf{Child}(t_2,m_2)\\ \wedge \mathsf{Label}_f(f_1) \wedge \mathsf{Label}_m(m_2) \wedge \mathsf{Label}_1(p_1) \wedge \mathsf{Label}_1(p_2)\\ \wedge \boldsymbol{(\mathsf{Child}^?)^k(f_1, p_1)} \wedge \boldsymbol{(\mathsf{Child}^?)^k(m_2, p_2)}\\ \wedge \mathsf{Child}^{\boldsymbol{(2n-1+3)}+k}(z,p_1) \wedge \mathsf{Child}^{\boldsymbol{(2n-1+5)}+k}(z,p_2).\end{aligned}$$

This is the other place we need the chain of ℓ nodes at the root: otherwise, again, since the initial configuration does not have a preceding configuration, we would not be able to go high enough up in the tree to find the z node. Taking $\ell \geqslant k-1$ suffices.

The query Q can be constructed in polynomial time, and Q is valid over the bNTA previously constructed if and only if the Turing machine M_w has no accepting (EXPSPACE) computation tree. ***Q* has treewidth 2.** ◀

F.3 2EXPTIME-Hardness of MDL containment in a Treelike CQ

We show a 2EXPTIME lower bound for the problem of checking the containment of a monadic Datalog program in a CQ of treewidth $\leqslant 2$. This matches the general upper bound for the containment of a Datalog query within a union of CQs.

▶ **Theorem 70.** *The following containment problem is 2EXPTIME-hard over the arity-two signature $\mathcal{S}^{\mathsf{Bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child},\mathsf{Child}^?}$: given a monadic Datalog program P with var-size $\leqslant 3$ and a conjunctive query Q of treewidth $\leqslant 2$, decide whether there exists some instance I satisfying $P \wedge \neg Q$.*

Proof. We reduce from the problem of validity of a CQ on $\mathcal{S}^{\mathsf{Bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child},\mathsf{Child}^?}$ of treewidth $\leqslant 2$ over a bNTA, which is 2EXPTIME-hard by Theorem 69.

Let $A = (\mathcal{Q}, F, \iota, \Delta)$ be a bNTA satisfying the two requirements in the previous paragraph, and Q a conjunctive query of treewidth $\leqslant 2$. We build a monadic Datalog program P as follows, which obeys the desired bound on var-size:

- For every $q \in \mathcal{Q}$, we have an intensional monadic predicate P_q.
- For every $q \in F$, we have a rule:

$$\mathrm{Goal}() \leftarrow \mathsf{Root}(r), P_q(r).$$

- For every symbol $\alpha \in \mathsf{Sch}$, for every $q \in \iota(\alpha)$, we have a rule:

$$P_q(l) \leftarrow \mathsf{Leaf}(l), \mathsf{Label}_\alpha(l), \mathsf{Child}^?(l, l).$$

- For every symbol $\alpha \in \mathsf{Sch}$, for every $q_1, q_2, q' \in \mathcal{Q}$ such that $q' \in \Delta(\alpha, q_1, q_2)$, we have a rule:

$$\begin{aligned} P_{q'}(n) \leftarrow\ & \mathsf{Label}_\alpha(n), P_{q_1}(n_1), P_{q_2}(n_2), \\ & \mathsf{FirstChild}(n, n_1), \mathsf{Child}(n, n_1), \mathsf{Child}^?(n, n_1), \\ & \mathsf{SecondChild}(n, n_2), \mathsf{Child}(n, n_2), \mathsf{Child}^?(n, n_2), \\ & \mathsf{Child}^?(n, n) \end{aligned}$$

Now, by construction, for every unfolding expansion tree t of P (see proof of Lemma 68), $I(\Pi_{\mathrm{ext}}(t))$ is a tree over $\mathcal{S}^{\mathsf{Bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child},\mathsf{Child}^?}$. Again by the proof of Lemma 68, the set of the instances of the form $I(\Pi_{\mathrm{ext}}(t))$ where t is an unfolding expansion tree of P is a canonical set of instances. In particular, there exists an instance satisfying P and not satisfying Q if and only if Q is valid over A. ◀

References for the Appendix

ABS15a A. Amarilli, P. Bourhis, and P. Senellart. Provenance circuits for trees and treelike instances (extended version). *CoRR*, abs/1511.08723, 2015. Extended version of [ABS15b].

ABS15b A. Amarilli, P. Bourhis, and P. Senellart. Provenance circuits for trees and treelike instances. In *ICALP*, volume 9135 of *LNCS*, 2015.

AHV95 S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases*. Addison-Wesley, 1995.

ALSU06 A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 2nd edition, 2006.

Ama16 A. Amarilli. *Leveraging the structure of uncertain data*. PhD thesis, Télécom ParisTech, 2016.

Bar13 P. Barceló. Querying graph databases. In *PODS*, 2013.

BBGS16 M. Benedikt, P. Bourhis, G. Gottlob, and P. Senellart. Monadic datalog and limited access containment. Unpublished, 2016.

BBS12 M. Benedikt, P. Bourhis, and P. Senellart. Monadic datalog containment. In *ICALP*, 2012.

BK10 H. L. Bodlaender and A. M. C. A. Koster. Treewidth computations I. Upper bounds. *Inf. Comput.*, 208(3), 2010.

BMS08a H. Björklund, W. Martens, and T. Schwentick. Optimizing conjunctive queries over trees using schema information. In *MFCS*, 2008.

BMS08b H. Björklund, W. Martens, and T. Schwentick. Optimizing conjunctive queries over trees using schema information. `http://www.theoinf.uni-bayreuth.de/download/Optimizing_Conjunctive_Queries_over_Trees.pdf`, 2008. Extended unpublished version of [BMS08a].

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

BtCV14 M. Benedikt, B. ten Cate, and M. Vanden Boom. Effective interpolation and preservation in guarded logics. In *LICS*, 2014.

CDG$^+$07 H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata: Techniques and applications, 2007. Available from `http://tata.gforge.inria.fr/`.

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

CKS81 A. K. Chandra, D. Kozen, and L. J. Stockmeyer. Alternation. *J. ACM*, 28(1), 1981.

CV97 S. Chaudhuri and M. Y. Vardi. On the equivalence of recursive and nonrecursive Datalog programs. *Journal of Computer and System Sciences*, 54(1), 1997.

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

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

TY84 R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13(3):566–579, 1984.

Val79 L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3), 1979.