Theory of Computing Systems manuscript No.
(will be inserted by the editor)

Evaluating Datalog via Tree Automata and Cycluits

Antoine Amarilli · **Pierre Bourhis** ·
Mikaël Monet · **Pierre Senellart**

the date of receipt and acceptance should be inserted later

Abstract We investigate parameterizations of both database instances and queries that make query evaluation fixed-parameter tractable in combined complexity. We show that clique-frontier-guarded Datalog with stratified negation (CFG-Datalog) enjoys 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.

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* [59] even for simple query languages such as conjunctive queries [1].

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

A. Amarilli · M. Monet and P. Senellart
LTCI, Télécom ParisTech, Université Paris-Saclay, Paris, France
E-mail: first.last@telecom-paristech.fr

P. Bourhis
CRIStAL, CNRS & Université de Lille, Lille, France
E-mail: pierre.bourhis@lifl.fr

M. Monet · P. Senellart
Inria Paris, Paris, France
E-mail: first.last@inria.fr

P. Senellart
DI ENS, ENS, CNRS, PSL Research University, Paris, France
E-mail: pierre.senellart@ens.fr

queries [61], 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* [55] (we call them *treelike*). This has been notably studied by Courcelle [28], 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 [50].

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 [39] 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 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* [15] and *containment* [12]. 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. Exploiting this connection, we 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.

Contributions. From there, the first main contribution of this paper is to consider the language of *clique-frontier-guarded Datalog* (CFG-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 it is a Datalog fragment, CFG-Datalog shares some similarities with guarded logics; yet, its design incorporates several features (fixpoints, clique-guards, 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 [11] 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, CFG-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 [56]. To our knowledge, our paper is the first to introduce this query class and to show its tractability on treelike instances. CFG-Datalog can be understood as an extension of this fragment to disjunction, clique-guardedness, stratified 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 CFG-Datalog programs, as they naturally deal with both recursion and two-way traversal of a treelike instance, which is less straightforward with provenance formulae [41] or circuits [29]. 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 [46,53], 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 [7].

The current article is a significant extension of the conference version [5,6], which in particular includes all proofs. We improved the definition of our language to a more natural and more expressive one, allowing us to step away from the world of guarded negation logics and thus answering a question we left open in the conclusion of [5]. We show that, in contrast with guarded negation logics and the ICG-Datalog language of [5], satisfiability of CFG-Datalog is undecidable. To allow space for the new material, this paper does not include any of the applications to probabilistic query evaluation that can be found in [5,6] (see also [8] for a more in-depth study of the combined complexity of probabilistic query evaluation).

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 CFG-Datalog in Section 5. We then define the automata variants we use and compile CFG-Datalog to them in Section 6, before introducing cycluits and showing our provenance computation result in Section 7. We last present the proof of our compilation result in Section 8.

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* (with a single exception, in Proposition 20): hence, its *arity* $\mathrm{arity}(\sigma)$ (the maximal arity of relations in σ) is assumed to be 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 $\mathscr{Q}$ and instance class $\mathscr{I}$: given an instance $I \in \mathscr{I}$ and query $Q \in \mathscr{Q}$, check if $I \models Q$. Its *combined complexity* for $\mathscr{I}$ and $\mathscr{Q}$ is a function of I and Q, whereas *data complexity* assumes Q to be fixed. We also study cases where $\mathscr{I}$ and $\mathscr{Q}$ are *parameterized*: given infinite sequences $\mathscr{I}_1, \mathscr{I}_2, \ldots$ and $\mathscr{Q}_1, \mathscr{Q}_2, \ldots$, the *query evaluation problem parameterized by* k_I, k_Q applies to $\mathscr{I}_{k_\mathrm{I}}$ and $\mathscr{Q}_{k_\mathrm{Q}}$. The parameterized problem is *fixed-parameter tractable* (FPT), for $(\mathscr{I}_n)$ and $(\mathscr{Q}_n)$, if there is a constant $c \in \mathbb{N}$ and computable function f such that the problem can be solved with combined complexity $O(f(k_\mathrm{I}, k_\mathrm{Q}) \cdot (|I| \cdot |Q|)^c)$. 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 [33].

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 $\max(\mathrm{arity}(\sigma), \mathrm{arity}(\sigma_\mathrm{int}))$, and 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 [54] 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 model, 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* [31], 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* [37]. With Yannakakis's algorithm [61], 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 [36], moving from linear time to PTIME, has investigated more general CQ classes, in particular CQs of bounded treewidth [32], *hypertreewidth* [37], and *fractional hypertreewidth* [42]. 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 [21], 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 [48]). Further, even for treewidth 2 (e.g., triangles), it is not known if we can achieve linear data complexity [3].

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 $O(|I| \cdot |Q|)$ [45]. Tractability is independently known for FO^k, the fragment of FO where subformulae use at most k variables, with a simple evaluation algorithm in $O(|I|^k \cdot |Q|)$ [60].

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 [60], query evaluation is tractable for μGF [19, Theorem 3], 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 [32, Corollary 5.26] for rules with restricted bodies (i.e., strictly acyclic, or bounded strict treewidth). This result was generalized in [38] 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 [35]. 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 C2RPQs but with PTIME complexity [11].

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, using tree automata techniques.

Tree encodings. 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*. Formally, we define the domain $\mathscr{D}_k = \{a_1, \ldots, a_{2k+2}\}$ and the finite alphabet Γ_σ^k whose elements are pairs (d, s), with d being a subset of up to $k+1$ elements of $\mathscr{D}_k$, 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 $\langle E, \lambda \rangle$ (λ being the labeling function); the fact that $\langle E, \lambda \rangle$ is rooted and 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 $\mathscr{D}_k$ being decoded to the domain elements: each occurrence of an element $a_i \in \mathscr{D}_k$ 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 $\langle E, \lambda \rangle$ means 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 2 [32] (see [4] 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.*

Bottom-up tree automata. 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) $F \subseteq Q$ 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 [28] on graphs (extended to higher-arity in [32]), 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 3 [28,32]. *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 [28,32], i.e., is in $O(f(|Q|, k) \cdot |I|)$ for some computable function f. However, this tells little about the combined complexity, as f is generally nonelementary in Q [50]. 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 [12].

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 [39] have proposed a *quasi-guarded* Datalog fragment on *relational structures and their tree decompositions*, for which 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 [19] 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 [7] (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 the compilation result of Theorem 3 does not extend directly:

Proposition 4. *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 coding of j at length $(i+1)$ 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 $\mathscr{I}$ of the instances of treewidth $\leqslant 1$, and a specific set $\mathscr{E}$ of tree encodings of instances of $\mathscr{I}$, and show the claim on $\mathscr{E}$, which suffices to conclude.

To define $\mathscr{I}$, let $\mathscr{S}_i$ be the set of subsets of $\{0,\dots,2^i-1\}$ of cardinality 2^{i-1}, so that $|\mathscr{S}_i|$ is $\binom{2^i}{2^{i-1}}$. We will first define a family $\mathscr{I}'$ of instances indexed by $\mathscr{S}_i$ as follows. Given $S \in \mathscr{S}_i$, the instance I'_S of $\mathscr{I}'$ 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 $\mathscr{I}$ to consist of a root element with two C-children, each of which are the root element of an instance of $\mathscr{I}'$ (we call the two the *child instances*). It is clear that instances of $\mathscr{I}$ have treewidth 1, and we can check quite easily that an instance of $\mathscr{I}$ satisfies Q_i iff the child instances I'_{S_1} and I'_{S_2} are such that $S_1 \cup S_2 = \{1,\dots,2^i\}$.

We now define $\mathscr{E}$ to be tree encodings of instances of $\mathscr{I}$. First, define $\mathscr{E}'$ to consist of tree encodings of instances of $\mathscr{I}'$, which we will also index with $\mathscr{S}_i$, i.e., E_S is a tree encoding of I'_S. We now define $\mathscr{E}$ as the tree encodings E constructed as follows: given an instance $I \in \mathscr{I}$, 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 $\mathscr{E}$ of one child instance of I. We define n_2 analogously with the other child instance.

For each $S \in \mathscr{S}_i$, letting $\bar{S}$ be the complement of S relative to $\{0,\dots,2^i-1\}$, we call $I_S \in \mathscr{I}$ the instance where the first child instance is I'_S and the second child instance is $I'_{\bar{S}}$, and we call $E_S \in \mathscr{E}$ the tree encoding of I_S according to the definition above. We then call $\mathscr{Q}_S$ 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 $\mathscr{Q}_S$ is non-empty.

Further, we show that the $\mathscr{Q}_S$ are pairwise disjoint: for any $S_1 \neq S_2$ of $\mathscr{S}_i$, we show that $\mathscr{Q}_{S_1} \cap \mathscr{Q}_{S_2} = \emptyset$. 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 $q \in \mathscr{Q}_{S_1}$ and $q \in \mathscr{Q}_{S_2}$. Now, consider the instance $I \in \mathscr{I}$ where the first child instance is I_1, and the second child instance is $\bar{I}_2$, and let $E \in \mathscr{E}$ 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 $E \in \mathscr{E}$ be its tree encoding, A rejects E. We have reached a contradiction, so indeed the $\mathscr{Q}_S$ are pairwise disjoint.

As the $\mathscr{Q}_S$ are non-empty, we can construct a mapping from $\mathscr{S}_i$ to the state set of A by mapping each $S \in \mathscr{S}_i$ to some state of $\mathscr{Q}_S$: as the $\mathscr{Q}_S$ are pairwise disjoint, this

mapping is injective. We deduce that the state set of A has size at least $|\mathscr{S}_i|$, which concludes from the bound on the size of $\mathscr{S}_i$ that we showed previously. □

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 [20] and on trees [26]: 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 [20]. In this work, we combine both extensions and use *alternating two-way tree automata* [26, 23], formally introduced in Section 6, which leads to tractable combined complexity for evaluation. Our general results in the next section will then imply:

Proposition 5. *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 [32], attributed to [57]). 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 13 to compile in linear time in $|Q|$ an CFG-Datalog program P with body size bounded by a constant k_{P}.

We now use Theorem 28 to construct, in FPT-linear time in $|P|$ (hence, in $|Q|$), parameterized by k_{I} and the constant k_{P}, an automaton A testing P for treewidth k_{I}; specifically, a stratified isotropic alternating two-way automata or SATWA (to be introduced in Definition 25).

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 13 does not produce any negated atom, and the compilation in the proof of Theorem 28 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 12 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. □

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 [48]. 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:

Theorem 6. *There is an arity-two signature σ for which there is no algorithm $\mathscr{A}$ 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 [14]. As this result heavily relies on [13], an unpublished extension of [14] whose relevant results are reproduced in [6], we deport its proof to Appendix A. Briefly, 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 7 [30]. *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 8. *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 $\text{arity}(\sigma) - 1$, 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 6. Hence:

Theorem 9. *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 13 to compile the CQ Q to an CFG-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 28 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} (see Definition 25). For the same reasons as in the proof of Proposition 5, it is actually a 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 12 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. □

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* [44, 18].

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 [34] that it occurs in some bag of 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 fixpoint, 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 CFG-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 10. *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.*

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:

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

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 [26]. 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 3 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 a 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 $n \geqslant 0$: $(\{a_1\}, \text{Begin}(a_1))$, $(\{a_1,a_2\}, X(a_1,a_2))$, $(\{a_2,a_3\}, X(a_2,a_3))$, $(\{a_3,a_1\}, X(a_3,a_1))$, …, $(\{a_n,a_{n+1}\}, X(a_n,a_{n+1}))$, $(\{a_{n+1}\}, \text{End}(a_{n+1}))$, 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. □

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 the rules must be *clique-frontier-guarded*, i.e., the variables in the head must co-occur in *positive* 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 11. *Let P be a stratified Datalog program. A rule r of P is* clique-frontier-guarded *if for any two variables $x_i \neq x_j$ in the head of r, x_i and x_j co-occur in some positive (extensional of intensional) predicate of the body of r. P is* clique-frontier-guarded *(CFG) if all its rules are clique-frontier-guarded. 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 CFG-Datalog is *FPT-linear* in *combined complexity*, when parameterized by the body size of the program and the instance treewidth.

Theorem 12. *Given a CFG-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 CFG-Datalog programs in FPT-linear time to a special kind of tree automata (Theorem 28), 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 9), including α-acyclic CQs, is shown by rewriting to CFG-Datalog of bounded body size:

Proposition 13. *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 CFG-Datalog program with body size at most $f_\sigma(k)$.*

To prove Proposition 13, we first prove the following lemma about simplicial tree decompositions:

Lemma 14. *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 $b_{S,1},\dots,b_{S,n_S}$ (so we have $S = \text{dom}(b) \cap \text{dom}(b_{S,j})$ for all $1 \leqslant j \leqslant n_S$), and we write $b'_{S,1},\dots,b'_{S,n_S}$ 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 13. In fact, as will be easy to notice from the proof, our construction further ensures that the equivalent CFG-Datalog program is positive, nonrecursive, and conjunctive. Recall that 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* [17] if each intensional relation R occurs in the head of at most one rule.

Proof of Proposition 13. Using Lemma 14, 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 [32] to annotate in linear time each node b of T by the set of atoms $\mathscr{A}_b$ of Q whose free variables are in the domain of b and such that for each atom A of $\mathscr{A}_b$, b is the topmost bag of T which contains all the variables of A. As the signature σ is fixed, note that we have $|\mathscr{A}_b| \leqslant g_\sigma(k)$ for some function g_σ depending only on σ.

We now perform a process similar to Lemma 3.1 of [32]. 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 $\mathscr{A}_b^{\text{guard}}$ 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 $\mathscr{A}_b^{\text{guard}}$, 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, $\mathscr{A}_b^{\text{guard}}$ 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 $\mathscr{A}_b^{\text{guard}}$ 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:

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

We now build our CFG-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 $\text{Goal}()$ 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 $P_b \leftarrow \bigwedge \mathscr{A}_b^{\text{guard}} \wedge \bigwedge \mathscr{A}_b$.
- If b is an internal node with children $b_1, \ldots, b_m$ (remember that $m \leqslant 2^{k+1}$), then ρ_b is $P_b \leftarrow \bigwedge \mathscr{A}_b^{\text{guard}} \wedge \bigwedge \mathscr{A}_b \wedge \bigwedge_{1 \leqslant i \leqslant m} P_{b_i}$.

We first check that P is clique-frontier-guarded, but this is the case because by construction the conjunction of atoms $\bigwedge \mathscr{A}_b^{\text{guard}}$ is a suitable guard for $\mathbf{x}$: for each $\{x,y\} \in \mathbf{x}$, it contains an atom where both x and y occur.

Second, we check that the body size of P is indeed $f_\sigma(k)$. It is clear that $\text{arity}(P) = \text{arity}(\sigma_{\text{int}} \cup \sigma) \leqslant k+1$. Further, the maximal number of atoms in the body of a rule is $g_\sigma(k) + 2^{k+1} + k(k+1)/2$, 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 $\mathscr{A}_b$. For the backward direction, noting that the only atoms of Q' that are not in Q are those added in the sets $\mathscr{A}_b^{\text{guard}}$, we observe that there is a homomorphism from Q' to Q defined by mapping each atom $A(\mathbf{w})$ occurring in some $\mathscr{A}_b^{\text{guard}}$ 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. □

This implies that CFG-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 CFG-Datalog when disallowing negation, recursion, and disjunction.

Proposition 15. *For any positive, conjunctive, nonrecursive CFG-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 15, we will use the notion of *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.

Proof of Proposition 15. 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 CFG-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 in 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 clique-frontier-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 CFG-Datalog program is an upper bound on the maximal number of variables in a rule body. □

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

Proposition 16. *There is a signature σ and a family $(P_n)_{n\in\mathbb{N}}$ of CFG-Datalog programs with body size at most 6 which are positive, conjunctive, and nonrecursive, such that $|P_n| = O(n)$ and any conjunctive query Q_n equivalent to P_n has size $\Omega(2^n)$.*

To prove Proposition 16, we recall the following classical notion:

Definition 17. *A* match *of a conjunctive query 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 18. *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 16:

Proof of Proposition 16. Fix σ to contain a binary relation R and a binary relation G. Consider the rule $\rho_0 : R_0(x,y) \leftarrow R(x,y)$ and define the following rules, for all $i > 0$:

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

For each $i > 0$, we let P_i consist of the rules ρ_j for $0 \leqslant j \leqslant i$, as well as the rule $\mathrm{Goal}() \leftarrow R_i(x,y)$. It is clear that each P_i is positive, conjunctive, and nonrecursive; further, the predicate G ensures that it is a CFG-Datalog program. The arity is 2 and the maximum number of atoms is the body is 3, so the body size is indeed 6.

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 18, 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. □

Guarded negation fragments. Having explained the connections between CFG-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 [10]. Indeed, when putting GNF formulae in *GN-normal form* [10] or even *weak GN-normal form* [16], we can translate them to CFG-Datalog, and we can use the *CQ-rank* parameter [16] (that measures the maximal number of atoms in conjunctions) to control the body size parameter.

Proposition 19. *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 CFG-Datalog program P of body size $f_\sigma(r)$.*

Proof. We recall from [16], 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 CFG-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. □

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 [15]), to give efficient procedures for GNF *satisfiability*. The *satisfiability* problem for a logic formally asks, given a sentence in this logic, whether it is satisfiable (i.e., there is an instance that satisfies it), and two variants of the problem exist: *finite satisfiability*, where we ask for the existence of a *finite* instance (as we defined them in this work), and *unrestricted satisfiability*, where we also allow the satisfying instance to be infinite. The decidability of both finite and unrestricted satisfiability for GNF is shown by compiling GNF to automata (for a treewidth which is not fixed, unlike in our context, but depends on the formula). CFG-Datalog further allows clique guards (similar to CGNFO [10]), can reuse subformulae (similar to the idea of DAG-representations in [16]), and supports recursion (similar to GNFP [10], or GN-Datalog [9] but whose combined complexity is intractable — P^{NP}-complete). CFG-Datalog also resembles μCGF [19], but μCGF is not a guarded *negation* logic, so, e.g., μCGF cannot express all CQs.

Hence, the design of CFG-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 CFG-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). Moreover CFG-Datalog is intrinsically more expressive than guarded negation logics as its satisfiability is undecidable, in contrast with GNF [10], CGNFO [10], GNFP [10], GN-Datalog [9], μCGF [40], the satisfiability of all of which is decidable.

Proposition 20. *Given a signature σ and a CFG-Datalog P over σ, determining if P is satisfiable is undecidable, in both the finite and unrestricted cases.*

Proof. We reduce from the implication problem for functional dependencies and inclusion dependencies, a problem known to be undecidable [51,25] over both finite and unrestricted instances. See also [1] for a general presentation of the problem and formal definitions and notation for functional dependencies and inclusion dependencies.

Let σ be a relational signature, let d be a functional dependency or an inclusion dependency over σ, and let Δ be a set of functional dependencies and inclusion dependencies over σ. The problem is to determine if Δ implies d.

We construct a CFG-Datalog program P over σ which is satisfiable over finite (resp., unrestricted) instances iff Δ implies d over finite (resp., unrestricted) instances, which establishes that CFG-Datalog satisfiability is undecidable.

The intensional signature of the program P is made of:

- a binary relation Eq;
- a nullary relation $P_{\neg\delta}$ for every dependency $\delta \in \Delta \cup \{d\}$;
- a relation $P_{\Pi_Z(S)}$ whose arity is $|Z|$ whenever there is at least one inclusion dependency $R[Y] \subseteq S[Z] \in \Delta \cup \{d\}$;
- the nullary relation Goal.

For every extensional relation R and for every $1 \leqslant i \leqslant \text{arity}(R)$, we add rules of the form:

$$\text{Eq}(x_i, x_i) \leftarrow R(\mathbf{x}).$$

Consequently, for every instance I over σ, the Eq-facts of $P(I)$ will be exactly $\{\text{Eq}(v,v) \mid v \in \text{dom}(I)\}$.

For every functional dependency δ in $\Delta \cup \{d\}$ with $\delta = R[Y] \to R[Z]$, we add the following rules, for $1 \leqslant j \leqslant |Z|$:

$$P_{\neg\delta}() \leftarrow R(\mathbf{x}), R(\mathbf{x}'), \text{Eq}(y_1, y'_1), \ldots, \text{Eq}(y_{|Y|}, y'_{|Y|}), \neg\text{Eq}(z_j, z'_j)$$

where for each $1 \leqslant i \leqslant |Y|$, the variables y_i and y'_i are those at the Y_i-th position in $R(\mathbf{x})$ and $R(\mathbf{x}')$, respectively; and where the variables z_j and z'_j are those at the Z_j-th position in $R(\mathbf{x})$ and $R(\mathbf{x}')$, respectively.

For every inclusion dependency $\delta \in \Delta \cup \{d\}$, with $\delta = R[Y] \subseteq S[Z]$ we add two rules:

$$P_{\Pi_Z(S)}(\mathbf{z}) \leftarrow S(\mathbf{x}) \qquad P_{\neg\delta}() \leftarrow R(\mathbf{x}), \neg P_{\Pi_Z(S)}(\mathbf{y})$$

where $\mathbf{z}$ are the variables at positions Z within $S(\mathbf{x})$ and $\mathbf{y}$ are the variables at positions Y within $R(\mathbf{x})$.

Finally, we add one rule for the goal predicate:

$$\text{Goal}() \leftarrow P_{\neg d}(), \neg P_{\neg\delta_1}(), \cdots, \neg P_{\neg\delta_k}()$$

where $\Delta = \{\delta_1, \ldots, \delta_k\}$.

Note that all the rules that we have written are clearly in CFG-Datalog. Now, let I be some instance. It is clear that for each functional dependency δ, $P_{\neg\delta}()$ is in $P(I)$ iff I does not satisfy δ. Similarly, for each inclusion dependency δ, $P_{\neg\delta}()$ is in $P(I)$ iff I does not satisfy δ. Therefore, for each instance I, Goal() is in $P(I)$ iff I satisfies Δ and I does not satisfy d. Thus P is satisfiable over finite instances (resp., unrestricted

instances) iff there exists a finite instance (resp., a finite or infinite instance) that satisfies Δ and does not satisfy d, i.e., iff Δ does imply d over finite instances (resp., over unrestricted instances). □

We point out that the extensional signature is not fixed in this proof, unlike in the rest of the article. This is simply to establish the expressiveness of CFG-Datalog, it has no impact on our study of the combined complexity of query evaluation.

Recursive languages. The use of fixpoints in CFG-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 [39]:

Proposition 21. *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 11) of the monadic Datalog program.*

Proof. This is simply by observing that any monadic Datalog program is a CFG-Datalog program with the same body size, so we can simply apply Theorem 12. □

Second, CFG-Datalog can capture *two-way regular path queries* (2RPQs) [24, 11], a well-known query language in the context of graph databases and knowledge bases:

Definition 22. *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 23 [49,11]. *2RPQ query evaluation (on arbitrary instances) has linear time combined complexity.*

CFG-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 [12].

Proposition 24. *Given a Boolean SAC2RPQ Q, we can compute in time $O(|Q|)$ an equivalent CFG-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 [2] 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 CFG-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 a CFG-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 [11] 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 2RPQs. 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 CFG-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 $\text{Goal}_i()$ 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. □

The rest of the article presents the tools needed for our results and their technical proofs.

6 Compilation to Automata

In this section, we study how we can compile CFG-Datalog queries on treelike instances to tree automata, to be able to evaluate them efficiently. As we showed with Proposition 4, we need more expressive automata than bNTAs. Hence, we use instead the formalism of *alternating two-way automata* [26], 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 25. *A* stratified isotropic alternating two-way automata *on Γ-trees (Γ-SATWA) is a tuple $A = (\mathscr{Q}, q_{\mathrm{I}}, \Delta, \zeta)$ with $\mathscr{Q}$ a finite set of* states*, q_{I} the* initial state*, Δ the* transition function *from $\mathscr{Q} \times \Gamma$ to $\mathscr{B}(\mathscr{Q})$, and ζ a* stratification function*, i.e., a function from $\mathscr{Q}$ onto $\{0, \ldots, m\}$ for some $m \in \mathbb{N}$, such that for any $q, q' \in \mathscr{Q}$ 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 $q \in \mathscr{Q}$ with $\zeta(q) \leqslant i$, by the following (nested) inductive definition on T_r:*

- *For $q \in \mathscr{Q}$ 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 $q \in \mathscr{Q}$ such that $\zeta(q) = i$, if $\Delta(q, \lambda(w))$ has a satisfying assignment (P, N), if we have an i-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 $q \in \mathscr{Q}$ 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 [23], 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 CFG-Datalog is performed, but we first note that evaluation of Γ-SATWAs is in linear time. 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.

Proposition 26. *For any alphabet Γ, given a Γ-tree $\langle T, \lambda \rangle$ and a Γ-SATWA A, we can determine whether $\langle T, \lambda \rangle \models A$ in time $O(|T| \cdot |A|)$.*

Proof. We use Theorem 39 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 37.

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. □

We now give our main compilation result: we can efficiently compile any CFG-Datalog program of bounded body size into a stratified alternating two-way automaton that *tests* it (in the same sense as for bNTAs). For pedagogical purposes, we present the compilation for a subclass of CFG-Datalog, namely, *CFG-Datalog with guarded negations* (CFG$^{\text{GN}}$-Datalog), in which invocations of negative intensional predicates are guarded in rule bodies:

Definition 27. *Let P be a stratified Datalog program. A negative intensional literal $\neg A(\mathbf{x})$ 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 positive atom of ψ. A CFG^{GN}-Datalog program is a CFG-Datalog program such that for any rule $R(\mathbf{x}) \leftarrow \psi(\mathbf{x}, \mathbf{y})$, every negative intensional literal in ψ is clique-guarded in ψ.*

We will then prove in Section 8 the following compilation result, and explain at the end of Section 8 how it can be extended to full CFG-Datalog:

Theorem 28. *Given a CFG^{GN}-Datalog program P of body size k_{P} and $k_{\text{I}} \in \mathbb{N}$, we can build in FPT-linear time in $|P|$ (parameterized by $k_{\text{P}}, k_{\text{I}}$) 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 $q_r^{\nu', \mathscr{A}}$ where r is a rule with head relation R and $\mathscr{A}$ 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 $\mathscr{A}$ 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 CFG-Datalog programs could be compiled efficiently to tree automata that test them on treelike instances. To show that SATWAs can be evaluated in linear time (stated earlier as Proposition 26), 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 29. *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 [43,41], or (more recently) as Boolean circuits [29,7]. 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 28, this claim implies our main provenance result:

Theorem 30. *Given a CFG-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.*

Of course, this result implies the analogous claims for query languages that are captured by CFG-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 12. The rest of this section formally introduces cycluits and proves Theorem 30.

Cycluits. We define *cycluits* as Boolean circuits without the acyclicity requirement, as in [53]. To avoid the problem of feedback loops, however, we first study *monotone cycluits*, and then cycluits with stratified negation.

Definition 31. *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: 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. Formally, the semantics of monotone cycluits is 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

The Knaster–Tarski theorem [58] gives an equivalent characterization:

Proposition 32. *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 32. □

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.

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: $\{g \in C \mid \nu(g) = 1\}$

```
1  /* Precompute the in-degree of ∧ gates                              */
2  for g ∈ C s.t. μ(g) = ∧ do
3      M[g] := |{g' ∈ C | g' → g}|
4  Q := {g ∈ C_inp | ν(g) = 1} ∪ {g ∈ C | (μ(C) = ∧) ∧ M[g] = 0}     /* as a stack */
5  S := ∅                                                          /* as a bit array */
6  while Q ≠ ∅ do
7      pop g from Q
8      if g ∉ S then
9          add g to S
10         for g' ∈ C | g → g' do
11             if μ(g') = ∨ then
12                 push g' into Q
13             if μ(g') = ∧ then
14                 M[g'] := M[g'] − 1
15                 if M[g'] = 0 then
16                     push g' into Q
17 return S
```

Proposition 33. *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 32. 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 32. 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 32. This concludes the proof. □

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 34. *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 onto $\{0, \ldots, m\}$ 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, we can show that C is stratified if and only if it contains no cycle of gates involving a $\neg$-gate. Moreover if C is stratified we can compute a stratification function in linear time by a topological sort.

Proposition 35. *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 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. □

We can then use any stratification function to define the evaluation of C (which will clearly be independent of the choice of stratification function):

Definition 36. *Let C be a stratified cycluit with stratification function $\zeta : C \to \{0,\dots,m\}$, 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 37. *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 35 , and compute the evaluation following Definition 36. 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, 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. □

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 $\mathscr{T} := \langle T, \lambda \rangle$ for some alphabet Γ, as in Section 6. We define a (Boolean) *valuation* ν of $\mathscr{T}$ 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 $\nu(\mathscr{T}) := \langle T, (\lambda \times \nu) \rangle$, obtained by annotating each node of $\mathscr{T}$ by its ν-image. As

in [7], we define the provenance of a $\overline{\Gamma}$-SATWA A on $\mathscr{T}$, which intuitively captures all possible results of evaluating A on possible valuations of $\mathscr{T}$:

Definition 38. *The* provenance *of a $\overline{\Gamma}$-SATWA A on a Γ-tree $\mathscr{T}$ is the Boolean function φ defined on the nodes of T such that, for any valuation ν of $\mathscr{T}$, $\nu(\varphi) = 1$ iff A accepts $\nu(\mathscr{T})$.*

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

Theorem 39. *For any fixed alphabet Γ, given a $\overline{\Gamma}$-SATWA A and a Γ-tree $\mathscr{T} = \langle T, \lambda \rangle$, we can build a stratified cycluit capturing the provenance of A on $\mathscr{T}$ in time $O(|A| \cdot |\mathscr{T}|)$.*

The construction generalizes Proposition 3.1 of [7] from bNTAs and circuits to SATWAs and cycluits. The reason why we need cycluits rather than circuits is because two-way automata may loop back on previously visited nodes. We construct a cycluit $C^A_{\mathscr{T}}$ as follows. For each node w of T, we create an input node g^{i}_w, a $\neg$-gate $g^{\neg \mathrm{i}}_w$ defined as $\mathrm{NOT}(g^{\mathrm{i}}_w)$, and an OR-gate g^q_w for each state $q \in Q$. Now for each g^q_w, 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 $g^{q,b}_w$ be the output gate of that circuit, we replace each variable q' occurring positively by an OR-gate $\bigvee_{w' \in \mathrm{Nbh}(w)} g^{q'}_{w'}$, and we replace each variable q' occurring negatively by the gate $g^{q'}_w$. We then define g^q_w as $\mathrm{OR}(\mathrm{AND}(g^{\mathrm{i}}_w, g^{q,0}_w), \mathrm{AND}(g^{\neg \mathrm{i}}_w, g^{q,1}_w))$. Finally, we let the output gate of C be $g^{q_{\mathrm{I}}}_r$, where r is the root of T, and q_{I} is the initial state of A.

It is clear that this process runs in linear time in $|A| \cdot |\mathscr{T}|$. The proof of Theorem 39 then results from the following claim:

Lemma 40. *The cycluit $C^A_{\mathscr{T}}$ is a stratified cycluit capturing the provenance of A on $\mathscr{T}$.*

Proof. We first show that $C := C^A_{\mathscr{T}}$ 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^{i}_w, we set $\zeta'(g^{\mathrm{i}}_w) := 0$ and $\zeta'(g^{\neg \mathrm{i}}_w) := 1$.
- For an OR gate $g := \bigvee_{w' \in \mathrm{Nbh}(w)} g^{q'}_{w'}$, we set $\zeta'(g) := \zeta(q')$.
- For any state g^q_w, we set $\zeta'(g^n_w) := \zeta(q) + 1$, 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 $\bigvee_{w' \in \mathrm{Nbh}(w)} g^{q'}_{w'}$, which are covered by the previous point, or $g^{q'}_w$, 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:

- $g^{\mathrm{i}}_w \to g^{\neg \mathrm{i}}_w$: by construction, we have $\zeta(g^{\mathrm{i}}_w) < \zeta'(g^{\neg \mathrm{i}}_w)$.
- $g \to g'$ where g' is a gate of the form g^q_w and g is an intermediate AND-gate in the definition of a g^q_w: 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 $\rho = \langle T_r, \lambda_r \rangle$ 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 36). 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 36 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. □

Note that the proof can be easily modified to make it work for standard alternating two-way automata rather than our isotropic automata.

Proving Theorem 30. We are now ready to conclude the proof of Theorem 30 by explaining how this provenance construction for $\overline{\Gamma}$-SATWAs can be used to compute the provenance of a CFG-Datalog query on a treelike instance. This is again similar to [7].

Recall the definition of tree encodings from Section 3.2, 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 41. *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 [4]):

Lemma 42. *For any Γ_σ^k-SATWA A, we can compute in linear time a $\overline{\Gamma_\sigma^k}$-SATWA A' such that, for any $\overline{\Gamma_\sigma^k}$-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 [4]. □

We are now ready to conclude the proof of our main provenance result (Theorem 30):

Proof of Theorem 30. Given the program P and instance I, use Theorem 28 to compute in FPT-linear time in $|P|$ a Γ_σ^k-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_σ^k-tree E, using Lemma 2. Lift the Γ_σ^k-SATWA A in linear time using Lemma 42 to a $\overline{\Gamma_\sigma^k}$-SATWA A', and use Theorem 39 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. 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 , 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 29. 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 [4]). 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 30. □

This concludes the presentation of our provenance results.

8 Proof of Compilation

In this section, we prove our main technical theorem, Theorem 28, which we recall here:

Theorem 28. *Given a CFG^{GN}-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.*

We then explain at the end of the section how this can be extended to full CFG-Datalog (i.e., with negative intensional predicates not being necessarily guarded in rule bodies).

8.1 Guarded-Negation Case

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

Definition 43. *Given a tuple* $\mathbf{x}$ *of variables, a* partial valuation *of* $\mathbf{x}$ *is a function* ν *from* $\mathbf{x}$ *to* $\mathscr{D}_{k_\mathrm{I}} \sqcup \{?\}$. *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* $\mathbf{y} \subseteq \mathbf{x}$*, 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* $\nu(z) = \nu'(z)$ *for all* z *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* z *to* $\nu(z)$ *or* $\nu'(z)$ *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 44. *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* $\nu : \mathbf{x} \to \mathscr{D}_{k_\mathrm{I}}$ *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 28 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 formally 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 $a \in \mathscr{D}_{k_\mathrm{I}}$ 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 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 $\nu(x_j) \neq ?$ (i.e., x_j is defined by ν) 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 $\mathscr{A}$ of the literals in the body of r, and partial valuation ν of the variables in $\mathscr{A}$ that is total for the variables in the head of r, we introduce a state $q^{\nu,\mathscr{A}}_r$. This state is intended to prove the literals in $\mathscr{A}$ 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 partial valuation ν of $\mathbf{x}$, we have a state $q^{\nu}_{R(\mathbf{x})}$. This state is intended to prove $R(\mathbf{x})$ with a total extension of ν. Let $(d,s) \in \Gamma^{k_\mathrm{I}}_\sigma$ be a symbol; we have the following transitions:

- If there is a j such that x_j is defined by ν and $\nu(x_j) \notin d$, 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 if ν is not total, then $\Delta(q^{\nu}_{R(\mathbf{x})}, (d,s)) := q^{\nu}_{R(\mathbf{x})} \vee \bigvee_{a \in d, x_j \in U(\nu)} q^{\nu \cup \{x_j \mapsto a\}}_{R(\mathbf{x})}$. Again, either we continue navigating in the same state, or we guess a value for some undefined variable.
- Else, $\Delta(q^{\nu}_{R(\mathbf{x})}, (d,s))$ is defined as the disjunction of all the $q^{\nu',\mathscr{A}}_r$ 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 $\mathscr{A}$ is the set of all literals in the body of r. Notice that because ν is 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 $q^{\nu,\mathscr{A}}_r$. Let $(d,s) \in \Gamma^{k_\mathrm{I}}_\sigma$, then:

- If there is a variable z in $\mathscr{A}$ such that z is defined by ν and $\nu(z) \notin d$, then $\Delta(q^{\nu,\mathscr{A}}_r, (d,s)) := \bot$.
- Else, if $\mathscr{A}$ contains at least two literals, then $\Delta(q^{\nu,\mathscr{A}}_r, (d,s))$ is defined as a disjunction of $q^{\nu,\mathscr{A}}_r$ and of $\Big[$a disjunction over all the non-empty sets $\mathscr{A}_1, \mathscr{A}_2$ that partition $\mathscr{A}$ of $\Big[$a disjunction over all the total valuations ν' of $U(\nu) \cap \mathrm{vars}(\mathscr{A}_1) \cap \mathrm{vars}(\mathscr{A}_2)$ with values in d of $\big[q^{\nu \cup \nu', \mathscr{A}_1}_r \wedge q^{\nu \cup \nu', \mathscr{A}_2}_r\big]\Big]\Big]$. 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 $\mathscr{A} = \{T(\mathbf{y})\}$ where T is an extensional or an intensional relation, then $\Delta(q_r^{\nu,\mathscr{A}}, (d,s)) := q_{T(\mathbf{y})}^{\nu}$.
- Else, if $\mathscr{A} = \{\neg R'(\mathbf{y})\}$ 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 $\Delta(q_r^{\nu,\mathscr{A}}, (d,s)) := q_r^{\nu,\mathscr{A}} \vee \bigvee_{a \in d} q_r^{\nu \cup \{y \mapsto a\},\mathscr{A}}$.
- Else, if $\mathscr{A} = \{\neg R'(\mathbf{y})\}$ with R' intensional, then we will only define the transitions in the case where ν is total on $\mathbf{y}$, in which case we set $\Delta(q_r^{\nu,\mathscr{A}}, (d,s)) := \neg q_{R'(\mathbf{y})}^{\nu}$.

  It is sufficient to define the transitions in this case, because $q_r^{\nu,\{\neg R'(\mathbf{y})\}}$ 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 $q_r^{\nu,\{\neg R'(\mathbf{y})\}}$ is by a sequence of transitions involving $q_r^{\nu_0,\mathscr{A}_0}, \ldots, q_r^{\nu_m,\mathscr{A}_m}$, where $\mathscr{A}_0$ are all the literals in the body of r, $\mathscr{A}_m$ is $\mathscr{A}$ and ν_m is ν. We can then see that, during the partitioning process, $\neg R'(\mathbf{y})$ must have been separated from all the (positive) atoms that formed its guard (recall the definition of CFG$^{\text{GN}}$-Datalog), hence all its variables have been assigned a valuation.

  Finally, the initial state of A'_P is $q_{\text{Goal}}^{\emptyset}$.

We describe the stratification function ζ' of A'_P. Let ζ be that of P. For any state q of the form $q_{T(\mathbf{x})}^{\nu}$ or $q_r^{\nu,\mathscr{A}}$ 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 $\leqslant k_{\text{I}}$. 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 $\leqslant k_{\text{I}}$ and that we can define from $\langle T, \lambda_E \rangle$ a tree decomposition $\langle T, \text{dom} \rangle$ of I whose underlying tree is also T. For each node $n \in T$, let $\text{dec}_n : \mathscr{D}_{k_{\text{I}}} \to \text{dom}(n)$ 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 $\text{enc}_n : \text{dom}(n) \to \mathscr{D}_{k_{\text{I}}}$ be the inverse function that encodes back the elements, so that we have $\text{dec}_n \circ \text{enc}_n = \text{enc}_n \circ \text{dec}_n = \text{Id}$. We will denote elements of $\mathscr{D}_{k_{\text{I}}}$ by a and elements in the domain of I by c.

We recall some properties of tree decompositions and tree encodings:

Property 45. *Let n_1, n_2 be nodes of T and $a \in \mathscr{D}_{k_{\text{I}}}$ be an (encoded) element that appears in the λ_E-image of n_1 and n_2. Then the element a appears in the λ_E-image of every node in the path from n_1 to n_2 if and only if* $\text{dec}_{n_1}(a) = \text{dec}_{n_2}(a)$.

Property 46. *Let n_1, n_2 be nodes of T and c be an element of I that appears in* $\text{dom}(n_1) \cap \text{dom}(n_2)$. *Then for every node n' on the path from n_1 to n_2, c is also in* $\text{dom}(n')$, *and moreover* $\text{enc}_{n'}(c) = \text{enc}_{n_1}(c)$.

We start with the following lemma about extensional facts:

Lemma 47. *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 again $(q^{\nu'}_{S(\mathbf{y})}, m)$ 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 at least one child π', the second component of the label of π' is some $m' \in \mathrm{Nbh}(m)$, and we have two cases depending on the first component of its label (i.e., the state):

- 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 45. 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 (because this would mean that we have escaped the "allowed subtree").
- 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 45.

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 in $\langle T, \lambda_E \rangle$, 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 46, 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 all relations (extensional or intensional). 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 48. *For every relation R, node $n \in T$ and partial 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 there exists $\mathbf{c}$ such that $R(\mathbf{c}) \in P(I)$ and $\mathbf{c}$ is compatible with ν at node n (i.e., we have $\mathrm{dec}_n(\nu(x_j)) = c_j$ for every x_j defined by ν).*

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 47. 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 $\neg(q^{\nu'}_{R'(\mathbf{y})}, m)$, then $R'(\mathrm{dec}_m(\nu'(\mathbf{y}))) \notin P(I)$ (remembering that in this case ν' must be total, thanks to the fact that negations are guarded in rule bodies).
(iii) If π is labeled with $(q_r^{\nu',\mathscr{A}}, m)$, then there exists a mapping $\mu : \mathrm{vars}(\mathscr{A}) \to \mathrm{Dom}(I)$ that is compatible with $\nu'_{|\mathrm{vars}(\mathscr{A})}$ at node m and such that:
  – For every positive literal $S(\mathbf{z})$ in $\mathscr{A}$, then $S(\mu(\mathbf{z})) \in P(I)$.
  – For every negative literal $\neg S(\mathbf{z})$ in $\mathscr{A}$, 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 47.

- π may be labeled by $(q^{\nu'}_{R'(\mathbf{y})}, m)$, where R' is intensional and verifies $\zeta(R') < i$. 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 then (i) follows from induction hypothesis on the stratum (using the forward direction of the equivalence lemma).
- π may be labeled by $\neg(q^{\nu'}_{R'(\mathbf{y})}, m)$, where R' is intensional and verifies $\zeta(R') < i$. Observe that by construction of the automaton, ν' is total (because negations are guarded in rule bodies). We need to show (ii). By definition of the run ρ there exists no run of A'_P starting at m in state $q^{\nu'}_{R'(\mathbf{y})}$. Hence 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' intensional. We need to prove (i). We distinguish two subsubcases:
  - ν' is not total. In that case, given the definition of $\Delta(q^{\nu'}_{R'(\mathbf{y})}, (d,s))$ and of the run, there exists a child π' of π labeled by $(q^{\nu''}_{R'(\mathbf{y})}, m')$, where $m' \in \mathrm{Nbh}(m)$ and ν'' is either ν' or is $\nu' \cup \{x_j \mapsto a\}$ for some x_j undefined by ν' and $a \in d$. Hence by induction on the run there exists $\mathbf{c}'$ such that $R'(\mathbf{c}') \in P(I)$ and $\mathbf{c}'$ is compatible with ν'' at node m'. We then take $\mathbf{c}$ to be $\mathbf{c}'$, and one can check that the compatibility condition holds.
  - ν' is total. In that case, given the definition of $\Delta(q^{\nu'}_{R'(\mathbf{y})}, (d,s))$ and of the run, there exists a child π' of π labeled by $(q_r^{\nu'',\mathscr{A}}, m')$, 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 $\mathscr{A}$ is the set of literals of r. Then, by induction on the run, there exists a mapping $\mu : \mathrm{vars}(\mathscr{A}) \to \mathrm{Dom}(I)$ 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 45 to m and m'.
- π may be labeled by $(q_r^{\nu',\mathscr{A}}, m)$, where $\mathscr{A} = \{\neg R''(\mathbf{y})\}$, where $|\mathbf{y}| = 1$, where the head of r uses relation R', and where y (writing y the one element of $\mathbf{y}$) is undefined by ν'. We need to prove (iii). By construction we have $\Delta(q_r^{\nu',\mathscr{A}}, (d,s)) = q_r^{\nu',\mathscr{A}} \vee \bigvee_{a \in d} q_r^{\nu' \cup \{y \mapsto a\},\mathscr{A}}$. 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 $(q_r^{\nu',\mathscr{A}}, m')$ or by $(q_r^{\nu' \cup \{y \mapsto a\},\mathscr{A}}, m')$ for some $a \in d$. In both cases it is easily seen that we can define an appropriate μ from the mapping μ' that we obtain by induction on the run (more details are given in the next bullet point).
- π may be labeled by $(q_r^{\nu',\mathscr{A}}, m)$, with $\mathscr{A} = \{R''(\mathbf{y})\}$, the head of r using relation R'. We need to prove (iii). By construction we have $\Delta(q_r^{\nu',\mathscr{A}}, (d,s)) = q^{\nu'}_{R''(\mathbf{y})}$, 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 45, $\mathbf{c}$ 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 $\mathrm{vars}(\mathscr{A}) = \mathbf{y}$, and this choice satisfies the required properties.

- π may be labeled by $(q_r^{\nu',\mathscr{A}}, m)$, with $\mathscr{A} = \{\neg R''(\mathbf{y})\}$ and ν' total on $\mathbf{y}$ and the head of r has relation R'. We again need to prove (iii). By construction we have $\Delta(q_r^{\nu',\mathscr{A}}, (d,s)) = \neg q^{\nu'}_{R''(\mathbf{y})}$ 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 $\mathrm{vars}(\mathscr{A}) = \mathbf{y}$, and the compatibility conditions are satisfied.
- π may be labeled by $(q_r^{\nu',\mathscr{A}}, m)$, with $|\mathscr{A}| \geqslant 2$. We need to prove (iii). Given the definition of $\Delta(q_r^{\nu',\mathscr{A}}, (d,s))$ 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 $(q_r^{\nu',\mathscr{A}}, m')$. By induction there exists $\mu' : \mathrm{vars}(\mathscr{A}) \to \mathrm{Dom}(I)$ satisfying (iii) for node m'. We can take μ to be μ', which satisfies the required properties.
  - There exist $m_1, m_2 \in \mathrm{Nbh}(m) \times \mathrm{Nbh}(m)$ and π_1, π_2 children of π and non-empty sets $\mathscr{A}_1, \mathscr{A}_2$ that partition $\mathscr{A}$ and a total valuation ν'' of $\mathrm{vars}(\mathscr{A}_1) \cap \mathrm{vars}(\mathscr{A}_2)$ with values in d such that π_1 is labeled by $(q_r^{\nu' \cup \nu'', \mathscr{A}_1}, m_1)$ and π_2 is labeled by $(q_r^{\nu' \cup \nu'', \mathscr{A}_2}, m_2)$. By induction there exists $\mu_1 : \mathrm{vars}(\mathscr{A}_1) \to \mathrm{Dom}(I)$ and similarly μ_2 that satisfy (iii). Thanks to the compatibility conditions for μ_1 and μ_2 and to Property 45 applied to m_1 and m_2 via m, we can define $\mu : \mathrm{vars}(\mathscr{A}) \to \mathrm{Dom}(I)$ 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 48). 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 partial 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 there exists $\mathbf{c}$ such that $R(\mathbf{c}) \in P(I)$ and $\mathbf{c}$ is compatible with ν at node n. Let R be a relation with $\zeta(R) = i$, $n \in T$ be a node and ν be a partial valuation of $\mathbf{x}$ such that there exists $\mathbf{c}$ such that $R(\mathbf{c}) \in P(I)$ and $\mathbf{c}$ is compatible with ν at node n. 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 $R(\mathbf{c}) \in \Xi^j_P(P_{i-1}(I))$, where Ξ^j_P is the j-th application of the immediate consequence operator for the program P (see [1]) 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 $R(\mathbf{c}) \in \Xi^0_P(P_{i-1}(I)) = P_{i-1}(I)$ implies that $\zeta(R) \leqslant i-1$, whereas we assumed $\zeta(R) = i$. For the inductive case ($j \geqslant 1$), we

have $R(\mathbf{c}) \in \Xi_P^j(P_{i-1}(I))$ 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 $\mu(\mathbf{z}) = \mathbf{c}$ 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 $R_l(\mu(\mathbf{y}_l)) \in \Xi_P^{j-1}(P_{i-1}(I))$
- If $L_l(\mathbf{y}_l) = \neg R_l(\mathbf{y}_l)$ is a negative literal, then $R_l(\mu(\mathbf{y}_l)) \notin P_{i-1}(I)$

Hence because $\mathbf{c}$ is clique-guarded and by a well-known property of tree decompositions (see Lemma 1 of [34], Lemma 2 of [22]), there exists a node n' such that $\mathbf{c} \subseteq \mathrm{dom}(n')$. Let $n = n_1, n_2, \dots, n_p = n'$ be the nodes on the path from n to n', and (d_i, s_i) be $\lambda_E(n_i)$ for $1 \leqslant i \leqslant p$. By compatibility, for every x_j defined by ν we have $\mathrm{dec}_n(\nu(x_j)) = c_j$. But $\mathrm{dec}_n(\nu(x_j)) \in \mathrm{dom}(n)$ and $c_j \in \mathrm{dom}(m)$ so by Property 46, for every $1 \leqslant i \leqslant p$ we have $c_j \in \mathrm{dom}(m_i)$ and $\mathrm{enc}_{m_i}(c_j) = \mathrm{enc}_n(c_j) = \mathrm{enc}_n(\mathrm{dec}_n(\nu(x_j))) = \nu(x_j)$, so that $\nu(x_j) \in d_i$. We can then start to construct the run ρ starting at node n in state $q^{\nu}_{R(\mathbf{x})}$ as follows. The root π_1 is labeled by $(q^{\nu}_{R(\mathbf{x})}, n)$, and for every $2 \leqslant i \leqslant p$, π_i is the unique child of π_{i-1} and is labeled by $(q^{\nu}_{R(\mathbf{x})}, m_i)$. 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 $\pi_{n'}$, we continue the run by staying at node n' and building up the valuation, until we reach a total valuation ν' such that $\nu'(\mathbf{x}) = \mathrm{enc}_{n'}(\mathbf{c})$. Hence we now only need to build a run ρ' starting at node n' in state $q^{\nu'}_{R(\mathbf{x})}$.

To achieve our goal of building a run starting at node n' in state $q^{\nu'}_{R(\mathbf{x})}$, it suffices 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 total valuation ν_l of $\mathbf{y}_l$ with $\mathrm{dec}_{m_l}(\nu_l(\mathbf{y}_l)) = \mu(\mathbf{y}_l)$ such that there exists a run ρ_l starting at node m_l in state $q^{\nu_l}_{R_l(\mathbf{y}_l)}$.

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

We first prove (i). We have $R_l(\mu(\mathbf{y}_l)) \in \Xi_P^{j-1}(P_{i-1}(I))$, and because P is clique-frontier-guarded, there exists a node m such that $\mu(\mathbf{y}_l) \subseteq m$. We take m_l to be m and ν_l to be such that $\nu_l(\mathbf{y}_l) = \mathrm{enc}_{m_l}(\mu(\mathbf{y}_l))$. We then directly obtain (i) by induction hypothesis (on j). We then prove (ii). Because the negative literals are guarded in rule bodies, there exists a node m such that $\mu(\mathbf{y}_l) \subseteq m$. We take m_l to be m and ν_l to be such that $\nu_l(\mathbf{y}_l) = \mathrm{enc}_{m_l}(\mu(\mathbf{y}_l))$. We straightforwardly get (ii) by using the induction on the strata of our equivalence lemma.

The second step is, from the runs ρ_l that we just constructed, to construct a run starting at node n' in state $q_r^{\nu'', \{L_1, \dots, L_t\}}$. 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 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 $\mathrm{dec}_n(\mu)$.
- 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 (namely, $\mu(x)$), which implies that it must occur in the node 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 $q_r^{\nu',\mathscr{A}}$ 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 $\mathbf{a} \in \mathscr{D}_{k_\mathrm{I}}^{\mathrm{arity}(R)}$. 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 $q_r^{\nu',\mathscr{A}}$ for each rule r such that the head of r is $R(\mathbf{y})$, $\nu'(\mathbf{y}) = \mathbf{a}$ and $\mathscr{A}$ is the set of all literals in the body of r.

The key point is that a given $q_r^{\nu',\mathscr{A}}$ 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 $\mathscr{A} = \{R'(\mathbf{y})\}$ with R' intensional, then $\Delta(q_r^{\nu,\mathscr{A}}, (d,s)) = q_{R'(\nu(\mathbf{y}))}$.
- Else, if $\mathscr{A} = \{\neg R'(\mathbf{y})\}$ with R' intensional, then $\Delta(q_r^{\nu,\mathscr{A}}, (d,s)) = \neg q_{R'(\nu(\mathbf{y}))}$.

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.

8.2 Managing Unguarded Negations

We now explain how compilation can be extended to the full CFG-Datalog fragment. We recall that the difference with CFG$^{\text{GN}}$-Datalog is that negative literals in rule bodies no longer need to be clique-guarded. Remember that clique-frontier-guardedness was used in the compilation of CFG$^{\text{GN}}$-Datalog to ensure the following property: when the automaton is trying to prove a rule $r := R(\mathbf{z}) \leftarrow L_1(\mathbf{y}_1) \dots L_t(\mathbf{y}_t)$ at some node n, i.e., when it is in a state $q_r^{\nu,\mathscr{A}}$ at node n for some subset $\mathscr{A}$ of literals of the body of r and partial valuation ν of the variables of $\mathscr{A}$, then, for each literal $L_l(\mathbf{y}_l)$ for $1 \leqslant l \leqslant t$, the images of $\mathbf{y}_l$ all appear together in a bag. More formally, let $\mu : \text{vars}(r) \to \text{dom}(I)$ be a mapping with $\mu(\mathbf{z}) = \text{dec}_n(\nu(\mathbf{z}))$ that witnesses that $R(\mu(\mathbf{z})) \in P(i)$: that is, if $L_l(\mathbf{y}_l)$ is a positive literal $S_l(\mathbf{y}_l)$ then we have $S_l(\mu(\mathbf{y_l})) \in P(i)$ and if $L(\mathbf{y}_l)$ is a negative literal $\neg S_l(\mathbf{y}_l)$ then we have $S_l(\mu(\mathbf{y}_l)) \notin P(I)$. In this case, we know that each $\mu(\mathbf{y}_l)$ must be contained in a bag of the tree decomposition.

This property is still true in CFG-Datalog when $L(\mathbf{y}_l)$ is a positive literal. However, when $L(\mathbf{y}_l)$ is a negative literal $\neg S_l(\mathbf{y}_l)$, it is now possible that $\mu(\mathbf{y}_l)$ is not contained in any bag of the tree decomposition. This can be equivalently rephrased as follows: there are $y_i, y_j \in \mathbf{y}_l$ with $y_i \neq y_j$ such that the occurrence subtrees of $\mu(y_i)$ and that of $\mu(y_j)$ are disjoint. If this happens, the automaton that we construct in the previous proof no longer works: it cannot assign the correct values to y_i and y_j, because once a value is assigned to y_i, the automaton cannot leave the occurrence subtree of $\mu(y_i)$ until a value is also assigned to y_j, which is not possible if the occurrence subtrees are disjoint.

To circumvent this problem, we will first rewrite the CFG Datalog program P into another program which intuitively distinguishes between two kinds of negations: the negative atoms that will hold as in the case of clique-guarded negations in CFG$^{\text{GN}}$-Datalog, and the ones that will hold because two variables have disjoint occurrence subtrees. First, we create a vacuous unary fact Adom, we modify the input instance and tree encoding in linear time to add the fact Adom for every element a in the active domain, and we modify P in linear time: for each rule r, for each variable x in the body of r, we add the fact $\text{Adom}(x)$. This ensures that each variable of rule bodies occurs in at least one positive fact.

Second, we rewrite P to a different program P'. For each rule r, we consider the set $\mathscr{N}$ of negative atoms in the body of r, and we consider every partition of $\mathscr{N}$ in $\mathscr{N}_{\text{G}} \cup \mathscr{N}_{\neg\text{G}}$, intuitively distinguishing the guarded and unguarded negations. For each such partition, for every atom of $\mathscr{N}_{\neg\text{G}}$, we choose a pair $y_i \neq y_l$ of variables

of this atom, and consider the possible graphs $\mathscr{G}$ formed by these edges (we may choose the same edge for two different atoms). The graph $\mathscr{G}$ intuitively describes the variables that must be mapped to elements having disjoint occurrence subtrees in the tree encoding. For each choice of $\mathscr{N}_{\mathrm{G}} \cup \mathscr{N}_{\neg \mathrm{G}}$ and $\mathscr{G}$, we create a rule $r_{\mathscr{N}_{\mathrm{G}},\mathscr{N}_{\neg \mathrm{G}},\mathscr{G}}$ defined as follows: it has the same head as r, and its body contains the positive atoms of the body of r (including the Adom-facts) and the negative atoms of $\mathscr{N}_{\mathrm{G}}$. We call $\mathscr{G}$ the *unguardedness graph of* $r_{\mathscr{N}_{\mathrm{G}},\mathscr{N}_{\neg \mathrm{G}},\mathscr{G}}$. Note that the semantics of P' is defined relative to the instance and also relative to the tree encoding of the instance that we consider: specifically, a rule can fire if there is a valuation that satisfies it in the sense of CFG$^{\mathrm{GN}}$-Datalog (i.e., for all atoms, including negative atoms, all variables must be mapped to elements that occur together in some node), and which further respects the unguardedness graph, i.e., for any two variables $x \neq y$ with an edge in the graph, the elements to which they are mapped must have disjoint occurrence subtrees in the tree encoding. Note that we can compute P' from P in FPT-linear time parameterized by the body size, because the number of rules created in P' for each rule of P can be bounded by the body size; further, the bound on the body size of P' only depends on that of P, specifically it only increases by the addition of the atoms $\mathrm{Adom}(x)$.

The compilation of P' can now be done as in the case of CFG$^{\mathrm{GN}}$-Datalog that we presented before; the only thing to explain is how the automaton can ensure that the unguardedness graph is satisfied. To this end, we will first make two general changes to the way that our automaton is defined, and then present the specific tweaks to handle the unguardedness graph. The two general changes can already be applied to the original automaton construction that we presented, without changing its semantics.

The first change is that, instead of isotropic automata, we will use automata that take the directions of the tree into account, as in [23] for example (with stratified negation as we do for SATWAs). Specifically, we change the definition of the transition function. Remember that a SATWA has a transition function $\Delta : \mathscr{Q} \times \Gamma \to \mathscr{B}(\mathscr{Q})$ that maps each pair of a state and a label to a propositional formula on states of $\mathscr{Q}$. To handle directions, Δ will instead map to a propositional formula on pairs of states of $\mathscr{Q}$ and of directions in the tree, in $\{\bullet, \uparrow, \leftarrow, \rightarrow\}$. The intuition is that the corresponding state is only evaluated on the tree node in the specified direction (rather than on any arbitrary neighbor). We will use these directions to ensure that, while the automaton considers a rule application and navigates to find the atoms used in the rule body, then it never visits the same node twice. Specifically, consider two variables y_i and y_j that are connected by an edge in the unguardedness graph, and imagine that we first assign a value a to y_i in some node n. To assign a value to y_j, we must leave the occurrence subtree of the current a in the tree encoding, and must choose a value outside of this occurrence subtree. Thus, the automaton must "remember" when it has left the subtree of occurrences of a, so that it can choose a value for y_j. However, an isotropic automaton cannot "remember" that it has left the subtree of occurrences of a, because it can always come back on a previously visited node, by going back in the direction from which it came. However, using SATWAs with directions, and changing the automata states to store the last direction that was followed, we can ensure that the automaton cannot come back to a previously visited node (while locating the facts that correspond to the body of a rule application). This ensures that, once the automaton has left the subtree of occurrences of an element, then it cannot come back in this

subtree (while it is considering the same rule application). Hence, the first general change is that we use SATWAs with directions, and we use the directions to ensure that the automaton does not go back to a previously visited node while considering the same rule application.

The second general change that we perform on the automaton is that, when guessing a value for an undefined variable, then we only allow the guess to happen as early as possible. In other words, suppose the automaton is at a node n in the tree encoding while it was previously at node n'. Then it can assign a value $a \in n$ to some variable y only if a was not in n', i.e., a has just been introduced in n. Obviously an automaton can remember which elements have been introduced in this sense, and which elements have not. This change can be performed on our existing construction without changing the semantics of the automaton, by only considering runs where the automaton assigns values to variables at the moment when it enters the occurrence subtree of this element.

Having done these general changes, we will simply reuse the previous automaton construction (not taking the unguardedness graph $\mathscr{G}$ into account) on the program P', and make two tweaks to ensure that the unguardedness graph is respected. The first tweak is that, in states of the form $q_r^{\nu,\mathscr{A}}$, the automaton will also remember, for each undefined variable x (i.e., x is in the domain of ν but $\nu(x)$ is still undefined), a set $\beta(x)$ of *blocking elements* for x, which are elements of the tree encoding. While $\beta(x)$ is non-empty, then the automaton is not allowed to guess a value for x, intuitively because we know that it is still in the occurrence subtree of some previously mapped variable $y \in \beta(x)$ which is adjacent to x in $\mathscr{G}$. Here is how these sets of blocking elements are computed and updated:

- When the automaton starts to consider the application of a rule r, then $\beta(x) := \emptyset$ for each variable x of the body of r.
- When the automaton guesses a value a for a variable x, then for every undefined variable y, if x and y are adjacent in $\mathscr{G}$, then we set $\beta(y) := \beta(y) \cup \{a\}$, intuitively adding a to the set of blocking elements for y. This intuitively means that the automaton is not allowed to guess a value for y until it has exited the subtree of occurrences of a. Note that, if the automaton wishes to guess values for multiple variables while visiting one node (in particular when partitioning the literals of $\mathscr{A}$), then the blocking sets are updated between each guess: this implies in particular that, if there is an edge in $\mathscr{G}$ between two variables x and y, then the automaton can never guess the value for x and for y at the same node.
- When the automaton navigates to a new node n' of the tree encoding, then for every variable x in the domain of ν which does not have an image yet, we set $\beta(x) := \beta(x) \cap n'$. Intuitively, when an element a was blocking for x but disappears from the current node, then a is no longer blocking. Note that $\beta(x)$ may then become empty, meaning that the automaton is now free to guess a value for x.

The blocking sets ensure that, when the automaton guesses a value for x, then it is guaranteed not to occur in the occurrence subtree of variables that are adjacent to x in $\mathscr{G}$ and have been guessed before. This also relies on the second general change above: we can only guess values for variables as early as possible, i.e., we can only use elements in guesses when we have just entered their occurrence subtree, so when

$\beta(x)$ becomes empty then the possible guesses for x do not include any element whose occurrence subtree intersects that of $\nu(y)$ for any variable y adjacent to x in $\mathscr{G}$.

The second tweak is that, when we partition the set of literals to be proven, then we use the directionality of the automaton to ensure that the remaining literals are split across the various directions. For instance, considering the rule body $\{\mathrm{Adom}(x), \mathrm{Adom}(y)\}$ and the unguardedness graph $\mathscr{G}$ having an edge between x and y, the automaton may decide at one node to partition $\mathscr{A} = \{\mathrm{Adom}(x), \mathrm{Adom}(y)\}$ into $\{\mathrm{Adom}(x)\}$ and $\{\mathrm{Adom}(y)\}$, and these two subsets of facts will be proven by two independent runs: these two runs are required to go in different directions of the tree. This will ensure that, even though the edge $\{x,y\}$ of $\mathscr{G}$ will not be considered explicitly by either of these runs (because the domain of their valuations will be $\{x\}$ and $\{y\}$ respectively), it will still be the case that x and y will be mapped to elements whose occurrence subtrees do not intersect: this is again using the fact that we map elements as early as possible.

We now summarize how the modified construction works:

1. Assume that the automaton A is at some node n in state $q^{\nu''}_{R(\mathbf{x})}$, with ν'' being total in $\mathbf{x}$.
2. At node n, the automaton chooses a rule $r' : R(\mathbf{z}) \leftarrow L_1(\mathbf{y}_1) \dots L_t(\mathbf{y}_t)$ of P' and goes to state $q^{\nu,\mathscr{A}}_{r'}$ where $\nu := \mathrm{Hom}_{\mathbf{z},\mathbf{x}}(\nu'')$ and $\mathscr{A}$ is the let of literals in the body of r'. That is, it simply chooses a rule to prove $R(\nu''(\mathbf{x}))$. This amounts to choosing a rule of the original program P, and choosing which negative atoms will be guarded (i.e., mapped to variables that occur together in some node of the tree encoding), and choosing the unguardedness graph $\mathscr{G}$ to ensure that the unguarded negations hold. The blocking sets $\beta(x)$ of each variable x in the rule body is initialized to the empty set, and whenever the automaton will move to a different node n' then each element a that is no longer present in n' is removed from $\beta(x)$ for each variable x, formally, $\beta(x) := \beta(x) \cap n'$.
3. From now on, assume the automaton A always remembers (stores in its state) which elements N have just been introduced in the current node of the tree encoding. That is, N is initialised with the elements in n, and when A goes from some node n' to node n'', N becomes $n'' \setminus n'$. When guessing values for variables, the automaton will only use values in N, so as to respect the condition that we guess the value of variables as early as possible.
4. While staying at node n, the automaton chooses some undefined variables x (i.e., variables in the domain of ν that do not have a value yet), and guesses some values in N for them. For each such variable x, we first verify that $\beta(x) = \emptyset$ (otherwise we fail), we set $\nu(x) := a$ where a is the guessed value, and then, for every edge $\{x,y\}$ in $\mathscr{G}$ such that y is an undefined variable (i.e., it is in the domain of ν but does not have an image by ν yet), we set $\beta(y) := \beta(y) \cup \{a\}$, ensuring that no value will be guessed for y until the automaton has left the subtree of occurrences for a. We call ν' the resulting new valuation.
5. While staying at node n, the automaton guesses a partition of $\mathscr{A}$ as $P_{\text{directions}} = (\mathscr{A}^\bullet, \mathscr{A}^\uparrow, \mathscr{A}^\leftarrow, \mathscr{A}^\rightarrow)$ to decide in which direction each one of the remaining facts is sent. Of course, if there is a direction for which n has no neighbor (e.g., $\leftarrow$ and

$\rightarrow$ if n is a leaf, or $\uparrow$ if n is the root), then $\mathscr{A}^d$ in the corresponding direction d must be empty.
6. If $\mathscr{A}^\bullet$ is the only class that is not empty, meaning that all remaining facts will be witnessed at the current node, then go to step 10.
7. While staying at node n, the automaton checks that each variable x that appears in two different classes of $P_{\text{directions}}$ have been assigned a value, i.e., $\nu(x)$ is defined; otherwise, the automaton fails. This is to ensure that the partitioning is consistent. The automaton also checks that $\nu(x)$ is defined for each variable occurring in $\mathscr{A}^\bullet$, that is, we assume without loss of generality that atoms that will be proven at the current node have all their variables already mapped. This ensures that the undefined variables are partitioned between directions in $\{\uparrow, \leftarrow, \rightarrow\}$.
8. The automaton then launches a run $q_{r'}^{\nu', \mathscr{A}^d}$ for each direction $d \in \{\bullet, \uparrow, \leftarrow, \rightarrow\}$ at the corresponding node (n for $\bullet$, the parent of n for $\uparrow$, the left of right child of n for $\leftarrow$ or $\rightarrow$).
9. For each of these runs, we go back to step 4, except that now A remembers the direction from which it comes, and does not go back to the previously visited node. For example if the automaton goes from some node n to the parent n' of n such that n is the left child of n', then in the partition that will be guessed at step 5 we will have $\mathscr{A}^{\leftarrow} = \emptyset$. Further, in each of these runs, of course, the automaton remembers the values of the blocking sets $\beta(x)$ for each undefined variable x.
10. Check that all the variables have been assigned. Launch positive states for each positive intensional literal and negative states for each negative literal, i.e., start from step 1: in this case, when the automaton proves a different rule application, then of course it forgets the values of the blocking sets, and forgets the previous direction (i.e., it can again visit the entire tree from the node where it starts). For each positive extensional literal, simply check that the atom is indeed encoded in the current node of the tree encoding.

All these modifications can be implemented in FPT-linear time provided that the arity of P is bounded, which is the case because the body size of P is bounded. Moreover, as we pointed out after the proof of Theorem 39, the construction of provenance cycluits can easily be modified to work for stratified alternating two way automata with directions, so that all our results about CFG-Datalog (evaluation and provenance cycluit computation in FPT linear time) still hold on this modified automaton.

9 Conclusion

We introduced CFG-Datalog, a 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.

A careful inspection of the proofs shows that our results can be used to derive PTIME combined complexity results on arbitrary instances, e.g., XP membership when

parametrizing only by program size; this recaptures in particular the tractability of some query languages on arbitrary instances, such as α-acyclic queries or SAC2RPQs. We also intend to extend our cycluit framework to support more expressive provenance semirings than Boolean provenance (e.g., formal power series [41]).

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 [52]. 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 [47].

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

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of databases. Addison-Wesley (1995)
2. Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools, 2nd edn. Addison-Wesley (2006)
3. Alon, N., Yuster, R., Zwick, U.: Finding and counting given length cycles. Algorithmica **17**(3) (1997)
4. Amarilli, A.: Leveraging the structure of uncertain data. Ph.D. thesis, Télécom ParisTech (2016)
5. Amarilli, A., Bourhis, P., Monet, M., Senellart, P.: Combined tractability of query evaluation via tree automata and cycluits. In: ICDT (2017)
6. Amarilli, A., Bourhis, P., Monet, M., Senellart, P.: Combined tractability of query evaluation via tree automata and cycluits (extended version). ArXiv e-prints (2017). `https://arxiv.org/abs/1612.04203v1`. Extended version of [5]
7. Amarilli, A., Bourhis, P., Senellart, P.: Provenance circuits for trees and treelike instances. In: ICALP, *LNCS*, vol. 9135 (2015)
8. Amarilli, A., Monet, M., Senellart, P.: Conjunctive queries on probabilistic graphs: Combined complexity. In: Proc. PODS, pp. 217–232. Chicago, USA (2017)
9. Bárány, V., ten Cate, B., Otto, M.: Queries with guarded negation. PVLDB **5**(11) (2012)
10. Bárány, V., ten Cate, B., Segoufin, L.: Guarded negation. J. ACM **62**(3) (2015)
11. Barceló, P.: Querying graph databases. In: PODS (2013)
12. Barceló, P., Romero, M., Vardi, M.Y.: Does query evaluation tractability help query containment? In: PODS (2014)
13. Benedikt, M., Bourhis, P., Gottlob, G., Senellart, P.: Monadic datalog and limited access containment (2016). Unpublished
14. Benedikt, M., Bourhis, P., Senellart, P.: Monadic datalog containment. In: ICALP (2012)
15. Benedikt, M., Bourhis, P., Vanden Boom, M.: A step up in expressiveness of decidable fixpoint logics. In: LICS (2016)
16. Benedikt, M., ten Cate, B., Vanden Boom, M.: Effective interpolation and preservation in guarded logics. In: LICS (2014)
17. Benedikt, M., Gottlob, G.: The impact of virtual views on containment. PVLDB **3**(1-2) (2010)
18. Berry, A., Pogorelcnik, R., Simonet, G.: An introduction to clique minimal separator decomposition. Algorithms **3**(2) (2010)
19. Berwanger, D., Grädel, E.: Games and model checking for guarded logics. In: LPAR (2001)
20. Birget, J.C.: State-complexity of finite-state devices, state compressibility and incompressibility. Mathematical systems theory **26**(3) (1993)
21. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput. **25**(6) (1996)
22. Bodlaender, H.L., Koster, A.M.C.A.: Treewidth computations I. Upper bounds. Inf. Comput. **208**(3) (2010)

23. Cachat, T.: Two-way tree automata solving pushdown games. In: Automata logics, and infinite games, chap. 17. Springer (2002)
24. Calvanese, D., De Giacomo, G., Lenzeniri, M., Vardi, M.Y.: Containment of conjunctive regular path queries with inverse. In: KR (2000)
25. Chandra, A.K., Vardi, M.Y.: The implication problem for functional and inclusion dependencies is undecidable. SIAM Journal on Computing **14**(3) (1985)
26. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata: Techniques and applications (2007). Available from `http://tata.gforge.inria.fr/`
27. Cosmadakis, S., Gaifman, H., Kanellakis, P., Vardi, M.: Decidable optimization problems for database logic programs. In: STOC (1988)
28. Courcelle, B.: The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. Inf. Comput. **85**(1) (1990)
29. Deutch, D., Milo, T., Roy, S., Tannen, V.: Circuits for Datalog provenance. In: ICDT (2014)
30. Diestel, R.: Simplicial decompositions of graphs: A survey of applications. Discrete Math. **75**(1) (1989)
31. Fagin, R.: Degrees of acyclicity for hypergraphs and relational database schemes. J. ACM **30**(3) (1983)
32. Flum, J., Frick, M., Grohe, M.: Query evaluation via tree-decompositions. J. ACM **49**(6) (2002)
33. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer (2006)
34. Gavril, F.: The intersection graphs of subtrees in trees are exactly the chordal graphs. J. Combinatorial Theory **16**(1) (1974)
35. Gottlob, G., Grädel, E., Veith, H.: Datalog LITE: A deductive query language with linear time model checking. ACM Trans. Comput. Log. **3**(1) (2002)
36. Gottlob, G., Greco, G., Scarcello, F.: Treewidth and hypertree width. In: L. Bordeaux, Y. Hamadi, P. Kohli (eds.) Tractability: Practical Approaches to Hard Problems, chap. 1. Cambridge University Press (2014)
37. Gottlob, G., Leone, N., Scarcello, F.: Hypertree decompositions and tractable queries. JCSS **64**(3) (2002)
38. Gottlob, G., Leone, N., Scarcello, F.: Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. JCSS **66**(4) (2003)
39. Gottlob, G., Pichler, R., Wei, F.: Monadic Datalog over finite structures of bounded treewidth. TOCL **12**(1) (2010)
40. Grädel, E.: Guarded fixed point logics and the monadic theory of countable trees. Theor. Comput. Sci. **288**(1) (2002). DOI 10.1016/S0304-3975(01)00151-7. URL `http://dx.doi.org/10.1016/S0304-3975(01)00151-7`
41. Green, T.J., Karvounarakis, G., Tannen, V.: Provenance semirings. In: PODS (2007)
42. Grohe, M., Marx, D.: Constraint solving via fractional edge covers. TALG **11**(1) (2014)
43. Imielinski, T., Lipski Jr., W.: Incomplete information in relational databases. J. ACM **31**(4) (1984). DOI 10.1145/1634.1886. URL `http://doi.acm.org/10.1145/1634.1886`
44. Leimer, H.G.: Optimal decomposition by clique separators. Discrete Math. **113**(1-3) (1993)
45. Leinders, D., Marx, M., Tyszkiewicz, J., den Bussche, J.V.: The semijoin algebra and the guarded fragment. Journal of Logic, Language and Information **14**(3) (2005)
46. Malik, S.: Analysis of cyclic combinational circuits. In: ICCAD (1993)
47. Maniu, S., Cheng, R., Senellart, P.: An indexing framework for queries on probabilistic graphs. ACM Transactions on Database Systems **42**(2), 13:1–13:34 (2017)
48. Marx, D.: Can you beat treewidth? Theory of Computing **6**(1) (2010)
49. Mendelzon, A.O., Wood, P.T.: Finding regular simple paths in graph databases. In: VLDB (1989)
50. Meyer, A.R.: Weak monadic second order theory of succesor is not elementary-recursive. In: Logic Colloquium (1975)
51. Mitchell, J.C.: The implication problem for functional and inclusion dependencies. Information and Control **56**(3) (1983)
52. Monet, M.: Probabilistic evaluation of expressive queries on bounded-treewidth instances. In: SIGMOD/PODS PhD Symposium (2016)
53. Riedel, M.D., Bruck, J.: Cyclic Boolean circuits. Discrete Applied Mathematics **160**(13-14) (2012)
54. Robertson, N., Seymour, P.D.: Graph minors. III. Planar tree-width. J. Comb. Theory, Ser. B **36**(1) (1984)
55. Robertson, N., Seymour, P.D.: Graph minors. II. Algorithmic aspects of tree-width. J. Algorithms **7**(3) (1986)

56. Tarjan, R.E.: Decomposition by clique separators. Discrete Math. **55**(2) (1985)
57. Tarjan, R.E., Yannakakis, M.: 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)
58. Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. Pacific Journal of Mathematics **5** (1955)
59. Vardi, M.Y.: The complexity of relational query languages. In: STOC (1982)
60. Vardi, M.Y.: On the complexity of bounded-variable queries. In: PODS, pp. 266–276 (1995)
61. Yannakakis, M.: Algorithms for acyclic database schemes. In: VLDB (1981)

A Proof of Theorem 6

Theorem 6. *There is an arity-two signature σ for which there is no algorithm $\mathscr{A}$ 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 [13], an extended version of [14]. Since [13] is currently unpublished, relevant results are reproduced as Appendix F of [6], in particular Lemma 68, Theorem 69, and their proofs.

Proof of Theorem 6. Let σ be $\mathscr{S}^{\mathsf{Bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child},\mathsf{Child}^?}$ as in Theorem 69 of [6]. We pose $c = 3$, $k_\mathrm{I} = 2 \times 3 - 1 = 5$. Assume by way of contradiction that there exists an algorithm $\mathscr{A}$ satisfying the prescribed properties. We will describe an algorithm to solve any instance of the containment problem of Theorem 69 of [6] in singly exponential time. As Theorem 69 of [6] 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 69 of [6], 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 of [6], compute in singly exponential time the $\Gamma_\sigma^{k_\mathrm{I}}$-bNTA A_P. Using the putative algorithm $\mathscr{A}$ on Q, compute in singly exponential time an alternating two-way automaton A_Q of polynomial size. As A_P describes a family $\mathscr{I}$ of canonical instances for P, there is an instance satisfying $P \wedge \neg Q$ iff there is an instance in $\mathscr{I}$ satisfying $P \wedge \neg Q$. Now, as $\mathscr{I}$ is described as the decodings of the language of A_P, all instances in $\mathscr{I}$ have treewidth $\leqslant k_\mathrm{I}$. Furthermore, the instances in $\mathscr{I}$ satisfy P by definition of $\mathscr{I}$. 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 [27] 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 [26]. This solves the containment problem in EXPTIME. As we explained initially, we have reached a contradiction, because it is 2EXPTIME-hard. □