Combined Tractability of Query Evaluation via Tree Automata and Cycluits

Antoine Amarilli, Pierre Bourhis, Mikaël Monet, Pierre Senellart

Combined Tractability of Query Evaluation via Tree Automata and Cycluits

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

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

Abstract

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

1998 ACM Subject Classification H.2 DATABASE MANAGEMENT

Keywords and phrases query evaluation; tree automata; provenance; treewidth; circuits

Digital Object Identifier 10.4230/LIPIcs...

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* [55] even for simple query languages.

To address this issue, two main directions have been investigated. The first is to restrict the class of *queries* to ensure tractability, for instance, to α-acyclic conjunctive queries [57], 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* [52] (we call them *treelike*). This has been notably studied by Courcelle [24], 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 [49].

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 [36] 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* [13] and *containment* [11]. To do this, one usually establishes a *treelike model property* to restrict the search to models of low treewidth (but dependent on the formula), and then compiles the formula to an automaton, so that the problems reduce to emptiness testing: expressive automata formalisms, such as *alternating two-way automata*, are typically used. One contribution of our work is to notice this connection, and show how query evaluation on treelike instances can benefit from these ideas: for instance, as we show, some queries can only be compiled efficiently to such concise automata, and not to the more common bottom-up tree automata.

From there, the first main contribution of this paper is to define the language of *intensional-clique-guarded Datalog* (ICG-Datalog), and show an efficient FPT-linear compilation procedure for this language, parameterized by the body size of rules: this implies FPT-linear combined complexity on treelike instances. While we present it as a Datalog fragment, our language shares some similarities with guarded logics; yet, its design incorporates several features (fixpoints, clique-guards, guarded negation, guarding positive subformulae) that are not usually found together in guarded fragments, but are important for query evaluation. We show how the tractability of this language captures the tractability of such query classes as two-way regular path queries [10] and α-acyclic conjunctive queries.

Already for conjunctive queries, we show that the treewidth of queries is not the right parameter to ensure efficient compilability. In fact, a second contribution of our work is a lower bound: we show that bounded treewidth queries cannot be efficiently compiled to automata at all, so we cannot hope to show combined tractability for them via automata methods. By contrast, ICG-Datalog implies the combined tractability of bounded-treewidth queries with an additional requirement (interfaces between bags must be clique-guarded), which is the notion of *simplicial decompositions* previously studied by Tarjan [53]. To our knowledge, our paper is the first to introduce this query class and to show its tractability on treelike instances. ICG-Datalog can be understood as an extension of this fragment to disjunction, clique-guarded negation, and inflationary fixpoints, that preserves tractability.

To derive our main FPT-linear combined complexity result, we define an operational semantics for our tree automata by introducing a notion of *cyclic provenance circuits*, that we call *cycluits*. These cycluits, the third contribution of our paper, are well-suited as a provenance representation for alternating two-way automata encoding ICG-Datalog programs, as they naturally deal with both recursion and two-way traversal of a treelike instance, which is less straightforward with provenance formulae [37] or circuits [26]. 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 [45, 51], where the semantics often features feedback loops that involve negation; we prohibit these by focusing on *stratified* circuits, which we show can be evaluated in linear time. We show that the provenance of alternating two-way automata can be represented as a stratified cycluit in FPT-linear time, generalizing results on bottom-up automata and circuits from [5].

Since cycluits directly give us a provenance representation of the query, we then investigate *probabilistic query evaluation*, which we showed in [5] to be linear-time in data complexity through the use of provenance circuits. We show how to remove cycles, so as to apply message-passing methods [42], yielding a 2EXPTIME upper bound for the combined complexity

of probabilistic query evaluation. While we do not obtain tractable probabilistic query evaluation in combined complexity, we give lower bounds showing that this is unlikely.

Outline. We give preliminaries in Section 2, and then position our approach relative to existing work in Section 3. We then present our tractable fragment, first for bounded-simplicial-width conjunctive queries in Section 4, then for our ICG-Datalog language in Section 5. We then define our automata and compile ICG-Datalog to them in Section 6, before introducing cycluits and showing our provenance computation result in Section 7. We last study the conversion of cycluits to circuits, and probability evaluation, in Section 8. Full proofs are provided in the extended version [4].

2 Preliminaries

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

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

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

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

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

use it to derive more facts. We have $I \models P$ iff we derive the fact Goal(). The *arity* of P is $\max(\text{arity}(\sigma), \text{arity}(\sigma_{\text{int}}))$. P is *monadic* if σ_{int} has arity 1.

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

(i) Each $R \in \sigma_{\text{int}}$ has a *stratum* $\zeta(R) \in \{1, \ldots, n\}$ such that all rules with R in the head are in $P_{\zeta(R)}$;
(ii) For any $1 \leqslant i \leqslant n$ and σ_{int}-atom $R(\mathbf{z})$ in a body of a rule of P_i, we have $\zeta(R) \leqslant i$;
(iii) For any $1 \leqslant i \leqslant n$ and negated σ_{int}-atom $R(\mathbf{z})$ in a body of P_i, we have $\zeta(R) < i$.

The stratification ensures that we can define the semantics of a stratified Datalog program by computing its interpretation for strata $P_1, \ldots, P_n$ in order: atoms in bodies always depend on a lower stratum, and negated atoms depend on strictly lower strata, whose interpretation was already fixed. Hence, there is a unique least fixpoint and $I \models P$ is well-defined.

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

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

Treewidth. Treewidth is a measure quantifying how far a graph is to being a tree, which we use to restrict instances and conjunctive queries. The *treewidth* of a CQ is that of its canonical instance, and the *treewidth* of an instance I is the smallest k such that I has a *tree decomposition* of *width* k, i.e., a finite, rooted, unranked tree T, whose nodes b (called *bags*) are labeled by a subset $\text{dom}(b)$ of $\text{dom}(I)$ with $|\text{dom}(b)| \leqslant k+1$, and which satisfies:

(i) for every fact $R(\mathbf{a}) \in I$, there is a bag $b \in T$ with $\mathbf{a} \subseteq \text{dom}(b)$;
(ii) for all $a \in \text{dom}(I)$, the set of bags $\{b \in T \mid a \in \text{dom}(b)\}$ is a connected subtree of T.

A family of instances is *treelike* if their treewidth is bounded by a constant.

3 Approaches for Tractability

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

3.1 Tractable Queries on All Instances

The best-known query language to ensure tractable query complexity is *α-acyclic queries* [28], 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* [34]. With Yannakakis's algorithm [57], 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 [33], moving from linear time to PTIME, has investigated more general CQ classes, in particular CQs of bounded treewidth [30], *hypertreewidth* [34], and *fractional hypertreewidth* [38]. 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 [19], 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 [47]). Further, even for treewidth 2 (e.g., triangles), it is not known if we can achieve linear data complexity [2].

In another direction, α-acyclicity has been generalized to queries with more expressive operators, e.g., disjunction or negation. The result on α-acyclic CQs thus extends to the *guarded fragment* (GF) of first-order logic, which can be evaluated on arbitrary instances in time $O(|I| \cdot |Q|)$ [44]. Tractability is independently known for FO^k, the fragment of FO where subformulae use at most k variables, with a simple evaluation algorithm in $O(|I|^k \cdot |Q|)$ [56].

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 [56], query evaluation is tractable [17, Theorem 3] for μGF, i.e., GF with some restricted least and greatest fixpoint operators, when *alternation depth* is bounded; without alternation, the combined complexity is in $O(|I| \cdot |Q|)$. We could alternatively express fixpoints in Datalog, but, sadly, most known tractable fragments are nonrecursive: nonrecursive stratified Datalog is tractable [30, Corollary 5.26] for rules with restricted bodies (i.e., strictly acyclic, or bounded strict treewidth). This result was generalized in [35] 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 [32]. Fixpoints were independently studied for graph query languages such as reachability queries and *regular path queries* (RPQ), which enjoy linear combined complexity on arbitrary input instances: this extends to two-way RPQs (2RPQs) and even strongly acyclic conjunctions of 2RPQs (SAC2RPQs), which are expressible in alternation-free μGF. Tractability also extends to acyclic RPQs but with PTIME complexity [10].

3.2 Tractability on Treelike Instances

We now study another approach for tractable query evaluation: this time, we restrict the shape of the *instances*, using treewidth. This ensures that we can translate them to a tree for efficient query evaluation. Informally, having fixed the signature σ, for a fixed treewidth $k \in \mathbb{N}$, there is a finite tree alphabet Γ_σ^k such that σ-instances of treewidth $\leqslant k$ can be translated in FPT-linear time (parameterized by k), following the structure of a tree decomposition, to a Γ_σ^k*-tree*, i.e., a rooted full ordered binary tree with nodes labeled by Γ_σ^k, which we call a *tree encoding*. We omit the formal construction: see the extended version [4] for more details.

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

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

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

We say that a bNTA A *tests* a query Q for treewidth k if, for any Γ_σ^k-encoding $\langle E, \lambda \rangle$ coding an instance I (of treewidth $\leqslant k$), A accepts $\langle E, \lambda \rangle$ iff $I \models Q$. By a well-known result of Courcelle [24] on graphs (extended to higher-arity in [30]), we can use bNTAs to evaluate all queries in *monadic second-order logic* (MSO), i.e., first-order logic with second-order variables of arity 1. MSO subsumes in particular CQs and monadic Datalog (but not general

Datalog). Courcelle showed that MSO queries can be compiled to a bNTA that tests them:

▶ **Theorem 2** ([24, 30]). *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 [24, 30], i.e., it is in $O\left(f(|Q|, k) \cdot |I|\right)$ for some computable function f. However, this tells little about the combined complexity, as f is generally nonelementary in Q [49]. 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 [11].

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 [36] have proposed a *quasiguarded* Datalog fragment on *relational structures and their tree decompositions*, with query evaluation is in $O(|I| \cdot |Q|)$. However, this formalism requires queries to be expressed in terms of the tree decomposition, and not just in terms of the relational signature. Second, Berwanger and Grädel [17] remark (after Theorem 4) that, when alternation depth and *width* are bounded, μCGF (the *clique-guarded* fragment of FO with fixpoints) enjoys FPT-linear query evaluation when parameterized by instance treewidth. Their approach does not rely on automata methods, and subsumes the tractability of α-acyclic CQs and alternation-free μGF (and hence SAC2RPQs), on treelike instances. However, μCGF is a restricted query language (the only CQs that it can express are those with a chordal primal graph), whereas we want a richer language, with a parameterized definition.

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

4 Conjunctive Queries on Treelike Instances

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

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

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

The intuition of the proof is that bNTAs can only make one traversal of the encoding of the input instance. Faced by this, we propose to use different tree automata formalisms, which are generally more concise than bNTAs. There are two classical generalizations of

nondeterministic automata, on words [18] and on trees [23]: 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 [18]. In this work, we combine both extensions and use *alternating two-way tree automata* [23, 20], formally introduced in Section 6, which leads to tractable combined complexity for evaluation. Our general results in the next section will then imply:

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

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

Bounded-treewidth queries. Having re-proven the combined tractability of α-acyclic queries (on bounded-treewidth instances), we naturally try to extend to *bounded-treewidth* CQs. Recall from Section 3.1 that these queries have PTIME combined complexity on all instances, but are unlikely to be FPT when parameterized by the query treewidth [47]. 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 [23]:

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

This result is obtained from a variant of the 2EXPTIME-hardness of monadic Datalog containment [12]. We show that efficient compilation of bounded-treewidth CQs to automata would yield an EXPTIME containment test, and conclude by the time hierarchy theorem.

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

▶ **Definition 6** ([27])**.** A *simplicial decomposition of a graph G* is a tree decomposition T of G such that, for any bag b of T and child bag b' of b, letting S be the intersection of the domains of b and b', then the subgraph of G induced by S is a complete subgraph of G.

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

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

Bounding the simplicial width of CQs is of course more restrictive than bounding their treewidth, and this containment relation is strict: cycles have treewidth $\leqslant 2$ but have unbounded simplicial width. This being said, bounding the simplicial width is less restrictive than imposing α-acyclicity: the join tree of an α-acyclic CQ is in particular a simplicial

decomposition, so α-acyclic CQs have simplicial width at most $\text{arity}(\sigma) - 1$, 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 5. Hence:

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

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

Notice the technicality that the simplicial decomposition T must be provided as input to the procedure, because it is not known to be computable in FPT-linear time, unlike tree decompositions. While we are not aware of results on the complexity of this specific task, quadratic time algorithms are known for the related problem of computing the *clique-minimal separator decomposition* [43, 16].

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 [31] that it occurs in some bag in the instance tree decomposition and can be "seen" by a tree automaton. This intuition can be generalized, beyond conjunctive queries, to design an expressive query language featuring disjunction, negation, and fixpoints, with the same properties of efficient compilation to automata and FPT-linear combined complexity of evaluation on treelike instances. We introduce such a Datalog variant in the next section.

5 ICG-Datalog on Treelike Instances

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

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

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

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

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

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

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

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

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

This implies that ICG-Datalog can express any CQ up to increasing the body size parameter, unlike, e.g., μCGF. Conversely, we can show that bounded-simplicial-width CQs *characterize* the queries expressible in ICG-Datalog when disallowing negation, recursion and disjunction. Specifically, a Datalog program is *positive* if it contains no negated atoms. It is *nonrecursive* if there is no cycle in the directed graph on σ_{int} having an edge from R to S whenever a rule contains R in its head and S in its body. It is *conjunctive* [14] if each intensional relation R occurs in the head of at most one rule. We can then show:

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

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

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

Guarded negation fragments. Having explained the connections between ICG-Datalog and CQs, we now study its connections to the more expressive languages of guarded logics, specifically, the *guarded negation fragment* (GNF), a fragment of first-order logic [9]. Indeed, when putting GNF formulae in *GN-normal form* [9] or even *weak GN-normal form* [15], we can translate them to ICG-Datalog, and we can use the *CQ-rank* parameter [15] (that measures the maximal number of atoms in conjunctions) to control the body size parameter.

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

In fact, the efficient compilation of bounded-CQ-rank normal-form GNF programs (using the fact that subformulae are "answer-guarded", like our guardedness requirements) has been used recently (e.g., in [13]), to give efficient procedures for GNF *satisfiability*, compiling them to automata (to a treewidth which is not fixed, unlike in our context, but depends on the formula). ICG-Datalog further allows clique guards (similar to CGNFO [9]), can reuse subformulae (similar to the idea of DAG-representations in [15]), and supports recursion

(similar to GNFP [9], or GN-Datalog [8] but whose combined complexity is intractable — P^{NP}-complete). ICG-Datalog also resembles μCGF [17], but remember that it is not a guarded *negation* logic, so, e.g., μCGF cannot express all CQs.

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

Recursive languages. The use of fixpoints in ICG-Datalog, in particular, allows us to capture the combined tractability of interesting recursive languages. First, observe that our guardedness requirement becomes trivial when all intensional predicates are monadic (arity-one), so our main result implies that *monadic Datalog* of bounded body size is tractable in combined complexity on treelike instances. This is reminiscent of the results of [36]:

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

Second, ICG-Datalog can capture *two-way regular path queries* (2RPQs) [21, 10], a well-known query language in the context of graph databases and knowledge bases:

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

▶ **Proposition 18** ([48, 10])**.** *2RPQ query evaluation (on arbitrary instances) has linear time combined complexity.*

ICG-Datalog allows us to capture this result for Boolean 2RPQs on treelike instances. In fact, the above result extends to SAC2RPQs, which are trees of 2RPQs with no multi-edges or loops. We can prove the following result, for Boolean 2RPQs and SAC2RPQs, which further implies compilability to automata (and efficient compilation of provenance representations). We do not know whether this extends to the more general classes studied in [11].

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

6 Compilation to Automata

In this section, we study how we can compile ICG-Datalog queries on treelike instances to tree automata, to be able to evaluate them efficiently. As we showed with Proposition 3, we need more expressive automata than bNTAs. Hence, we use instead the formalism of *alternating two-way automata* [23], i.e., automata that can navigate in trees in any direction, and can express transitions using Boolean formulae on states. Specifically, we introduce for our purposes a variant of these automata, which are *stratified* (i.e., allow a form of stratified negation), and *isotropic* (i.e., no direction is privileged, in particular order is ignored).

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

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

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

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

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

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

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

Observe that the internal nodes of a run starting in some state q are labeled by states q' in the same stratum as q. The leaves of the run may be labeled by states of a strictly lower stratum or negations thereof, or by states of the same stratum whose transition function is tautological, i.e., by some (q', w) such that $\Delta(q', \lambda(w))$ has $\emptyset, \emptyset$ as a satisfying assignment. Intuitively, if we disallow negation in transitions, our automata amount to the alternating two-way automata used by [20], with the simplification that they do not need parity acceptance conditions (because we only work with finite trees), and that they are *isotropic*: the run for each positive child state of an internal node may start indifferently on *any* neighbor of w in the tree (its parent, a child, or w itself), no matter the direction. (Note, however, that the run for negated child states must start on w itself.)

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

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

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

Compilation. We now give our main compilation result: we can efficiently compile any ICG-Datalog program of bounded body size into a SATWA that *tests* it (in the same sense as for bNTAs). This is our main technical claim, which is proven in the extended version [4].

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

Proof sketch. The idea is to have, for every relational symbol R, states of the form $q^{\nu}_{R(\mathbf{x})}$, where ν is a partial valuation of $\mathbf{x}$. This will be the starting state of a run if it is possible to navigate the tree encoding from some starting node and build in this way a total valuation ν' that extends ν and such that $R(\nu'(\mathbf{x}))$ holds. When R is intensional, once ν' is total on $\mathbf{x}$, we go into a state of the form $q_r^{\nu',\mathcal{A}}$ where r is a rule with head relation R and $\mathcal{A}$ is the set of atoms in the body of r (whose size is bounded by the body size). This means that we choose a rule to prove $R(\nu'(\mathbf{x}))$. The automaton can then navigate the tree encoding, build ν' and coherently partition $\mathcal{A}$ so as to inductively prove the atoms of the body. The clique-guardedness condition ensures that, when there is a match of $R(\mathbf{x})$, the elements to which $\mathbf{x}$ is mapped can be found together in a bag. The fact that the automaton is isotropic relieves us from the syntactic burden of dealing with directions in the tree, as one usually has to do with alternating two-way automata. ◀

7 Provenance Cycluits

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

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

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

We can represent Boolean provenance as Boolean formulae [39, 37], or (more recently) as Boolean circuits [26, 5]. In this section, we first introduce *monotone cycluits* (monotone Boolean circuits with cycles), for which we define a semantics (in terms of the Boolean function that they express); we also show that cycluits can be evaluated in linear time, given a valuation. Second, we extend them to *stratified cycluits*, allowing a form of stratified negation. We conclude the section by showing how to construct the *provenance* of a SATWA as a cycluit, in FPT-linear time. Together with Theorem 22, this claim implies our main provenance result:

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

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

Cycluits. We define *cycluits* as Boolean circuits without the acyclicity requirement, as in [51]. To avoid the problem of feedback loops, however, we first study *monotone cycluits*, and then cycluits with stratified negation.

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

We now define the semantics of monotone cycluits. A (Boolean) *valuation* of C is a function $\nu : C_{\mathsf{inp}} \to \{0, 1\}$ indicating the value of the input gates. As for standard monotone circuits, a valuation yields an *evaluation* $\nu' : C \to \{0, 1\}$, that we will define shortly, indicating the value of each gate under the valuation ν: we abuse notation and write $\nu(C) \in \{0, 1\}$ for the *evaluation result*, i.e., $\nu'(g_0)$ where g_0 is the output gate of C. The Boolean function *captured* by a cycluit C is thus the Boolean function φ on C_{inp} defined by $\nu(\varphi) := \nu(C)$ for each valuation ν of C_{inp}. We define the evaluation ν' from ν by a least fixed-point computation (see the algorithm in the extended version [4]): we set all input gates to their value by ν, and other gates to 0. We then iterate until the evaluation no longer changes, by evaluating OR-gates to 1 whenever some input evaluates to 1, and AND-gates to 1 whenever all their inputs evaluate to 1. The Knaster–Tarski theorem [54] gives an equivalent characterization:

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

We show that this definition is computable in linear time (see the extended version [4]):

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

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

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

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

▶ **Definition 29.** Let C be a stratified cycluit with stratification function $\zeta : C \to \{0, \ldots, m\}$, and let ν be a Boolean valuation of C. We inductively define the *i-th stratum evaluation* ν_i, for i in the range of ζ, by setting $\nu_0 := \nu$, and letting ν_i extend the ν_j $(j < i)$ as follows:

1. For g such that $\zeta(g) = i$ with $\mu(g) = \neg$, set $\nu_i(g) := \neg\nu_{\zeta(g')}(g')$ for its one input g'.
2. Evaluate all other g with $\zeta(g) = i$ as for monotone cycluits, considering the $\neg$-gates of point **1.** and all gates of lower strata as input gates fixed to their value in ν_{i-1}.

Letting g_0 be the output gate of C, the Boolean function φ *captured* by C is then defined as $\nu(\varphi) := \nu_m(g_0)$ for each valuation ν of C_{inp}.

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

Building provenance cycluits. Having defined cycluits as our provenance representation, we compute the provenance of a query on an instance as the *provenance* of its SATWA on a tree encoding. To do so, we must give a general definition of the provenance of SATWAs. Consider a Γ-tree $\mathcal{T} := \langle T, \lambda\rangle$ for some alphabet Γ, as in Section 6. We define a (Boolean) *valuation* ν of $\mathcal{T}$ as a mapping from the nodes of T to $\{0, 1\}$. Writing $\overline{\Gamma} := \Gamma \times \{0, 1\}$, each valuation ν then defines a $\overline{\Gamma}$-tree $\nu(\mathcal{T}) := \langle T, (\lambda \times \nu)\rangle$, obtained by annotating each node of $\mathcal{T}$ by its ν-image. As in [5], we define the provenance of a $\overline{\Gamma}$-SATWA A on $\mathcal{T}$, which intuitively captures all possible results of evaluating A on possible valuations of $\mathcal{T}$:

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

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

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

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

8 From Cycluits to Circuits and Probability Bounds

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

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

▶ **Proposition 33** (([51], Theorem 2)). *For any stratified cycluit C, we can compute in time $O(|C|^2)$ a Boolean circuit C' which is equivalent to C.*

In addition to being quadratic rather than linear, another disadvantage of this approach is that bounds on the treewidth of the cycluit (which we will need later for probability computation) are generally not preserved on the output. Hence, we prove a second cycle removal result, that proceeds in FPT-linear time when parameterized by the treewidth of the input cycluit. When we use this result, we no longer preserve FPT combined complexity of the overall computation, because the stratified cycluits produced by Theorem 24 generally have treewidth $\Omega(|P|)$. On the other hand, we obtain an FPT-linear data complexity bound, and a bounded-treewidth circuit as a result.

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

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

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

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

Earlier work [25] showed that PQE has #P-hard data complexity even for some CQs of a simple form, but [5, 6] shows that PQE is tractable in data complexity for any Boolean query in monadic second-order (MSO) if the input instances are required to be treelike.

We now explain how to use Theorem 34 for PQE. Let P be an ICG-Datalog program of body size k_{P}. Given a TID instance (I, π) of treewidth k_{I}, we compute a provenance cycluit for P on I of treewidth $O(|P|)$ in FPT-linear time in $|I| \cdot |P|$ by Theorem 24. By Theorem 34, we compute in $O(2^{|P|^\alpha}\,|I|\,|P|)$ an equivalent circuit of treewidth $O(2^{|P|^\alpha})$. Now, by Theorem D.2 of [6], we can solve PQE for P and (I, π) in $O(2^{2^{|P|^\alpha}}\,|I|\,|P| + |\pi|)$ up to PTIME arithmetic costs. Linear-time data complexity was known from [5], but 2EXPTIME combined complexity is novel, as [5] only gave non-elementary combined complexity bounds.

Acyclic queries on tree TIDs. A natural question is then to understand whether better bounds are possible. In particular, is PQE tractable in *combined complexity* on treelike instances? We show that, unfortunately, treewidth bounds are *not* sufficient to ensure this. The proof draws some inspiration from earlier work [40] on the topic of tree-pattern query evaluation in *probabilistic XML* [41].

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

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

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

Tractable cases. In which cases, then, could PQE be tractable in combined complexity? One example is in [22]: PQE is tractable in combined complexity over *probabilistic XML*, when queries are written as *deterministic* tree automata. In this setting, that the edges of the XML document are *directed* (preventing, e.g., the inverse construction used in the proof of Proposition 37). Further, as the result works on *unranked* trees, it is important that children of a node are *ordered* as well (see [3] for examples where this matters).

We leave open the question of whether there are some practical classes of instances and of queries for which such a deterministic tree automaton can be obtained from the query in polynomial time to test the query for a given treewidth. As we have shown, path queries and instances of treewidth 1, even though very restricted, do not suffice to ensure this. Note that, in terms of data complexity, we have shown in [7] that treelike instances are essentially the only instances for which first-order tractability is achievable.

9 Conclusion

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

We believe that ICG-Datalog can be further improved by removing the guardedness requirement on negated atoms, which would make it more expressive and step back from the world of guarded negation logics. In particular, we conjecture that our FPT-linear tractability result generalizes to *frontier-guarded Datalog*, and its extensions with clique-guards and stratified (but unguarded) negation, taking the rule body size and instance treewidth as the parameters. We further hope that our results could be used to derive PTIME combined complexity results on instances of arbitrary treewidth, e.g., XP membership when parametrizing by program size; this could in particular recapture the tractability of bounded-treewidth queries. Last, we intend to extend our cycluit framework to support more expressive provenance semirings than Boolean provenance (e.g., formal power series [37]).

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 [50]. 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 [46].

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

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*. Addison-Wesley, 1995.

2 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3), 1997.

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

4 Antoine Amarilli, Pierre Bourhis, Mikaël Monet, and Pierre Senellart. Combined tractability of query evaluation via tree automata and cycluits (extended version). *CoRR*, abs/1612.04203, 2017. `https://arxiv.org/abs/1612.04203`.

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

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

7 Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Tractable lineages on treelike instances: Limits and extensions. In *PODS*, 2016.

8 Vince Bárány, Balder ten Cate, and Martin Otto. Queries with guarded negation. *PVLDB*, 5(11), 2012.

9 Vince Bárány, Balder ten Cate, and Luc Segoufin. Guarded negation. *J. ACM*, 62(3), 2015.

10 Pablo Barceló. Querying graph databases. In *PODS*, 2013.

11 Pablo Barceló, Miguel Romero, and Moshe Y Vardi. Does query evaluation tractability help query containment? In *PODS*, 2014.

12 Michael Benedikt, Pierre Bourhis, and Pierre Senellart. Monadic datalog containment. In *ICALP*, 2012.

13 Michael Benedikt, Pierre Bourhis, and Michael Vanden Boom. A step up in expressiveness of decidable fixpoint logics. In *LICS*, 2016.

14 Michael Benedikt and Georg Gottlob. The impact of virtual views on containment. *PVLDB*, 3(1-2), 2010.

15 Michael Benedikt, Balder ten Cate, and Michael Vanden Boom. Effective interpolation and preservation in guarded logics. In *LICS*, 2014.

16 Anne Berry, Romain Pogorelcnik, and Genevieve Simonet. An introduction to clique minimal separator decomposition. *Algorithms*, 3(2), 2010.

17 Dietmar Berwanger and Erich Grädel. Games and model checking for guarded logics. In *LPAR*, 2001.

18 Jean-Camille Birget. State-complexity of finite-state devices, state compressibility and incompressibility. *Mathematical systems theory*, 26(3), 1993.

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

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

21 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzeniri, and Moshe Y. Vardi. Containment of conjunctive regular path queries with inverse. In *KR*, 2000.

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

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

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

25 Nilesh Dalvi and Dan Suciu. Management of probabilistic data: foundations and challenges. In *PODS*, 2007.

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

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

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

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

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

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

32 Georg Gottlob, Erich Grädel, and Helmut Veith. Datalog LITE: A deductive query language with linear time model checking. *ACM Trans. Comput. Log.*, 3(1), 2002.

33 Georg Gottlob, Gianluigi Greco, and Francesco Scarcello. Treewidth and hypertree width. In Lucas Bordeaux, Youssef Hamadi, and Pushmeet Kohli, editors, *Tractability: Practical Approaches to Hard Problems*, chapter 1. Cambridge University Press, 2014.

34 Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *JCSS*, 64(3), 2002.

35 Georg Gottlob, Nicola Leone, and Francesco Scarcello. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. *JCSS*, 66(4), 2003.

36 Georg Gottlob, Reinhard Pichler, and Fang Wei. Monadic Datalog over finite structures of bounded treewidth. *TOCL*, 12(1), 2010.

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

38 Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *TALG*, 11(1), 2014.

39 Tomasz Imielinski and Witold Lipski, Jr. Incomplete information in relational databases. *J. ACM*, 31(4), 1984.

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

41 Benny Kimelfeld and Pierre Senellart. Probabilistic XML: Models and complexity. In Zongmin Ma and Li Yan, editors, *Advances in Probabilistic Databases for Uncertain Information Management*. Springer, 2013.

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

43 Hanns-Georg Leimer. Optimal decomposition by clique separators. *Discrete Math.*, 113(1-3), 1993.

44 Dirk Leinders, Maarten Marx, Jerzy Tyszkiewicz, and Jan Van den Bussche. The semijoin algebra and the guarded fragment. *Journal of Logic, Language and Information*, 14(3), 2005.

45 Sharad Malik. Analysis of cyclic combinational circuits. In *ICCAD*, 1993.

46 Silviu Maniu, Reynold Cheng, and Pierre Senellart. ProbTree: A query-efficient representation of probabilistic graphs. In *BUDA*, June 2014. Workshop without formal proceedings.

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

48 Alberto O. Mendelzon and Peter T. Wood. Finding regular simple paths in graph databases. In *VLDB*, 1989.

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

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

51 Marc D. Riedel and Jehoshua Bruck. Cyclic Boolean circuits. *Discrete Applied Mathematics*, 160(13-14), 2012.

52 Neil Robertson and Paul D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7(3), 1986.

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

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

55 Moshe Y Vardi. The complexity of relational query languages. In *STOC*, 1982.

56 Moshe Y. Vardi. On the complexity of bounded-variable queries. In *PODS*, pages 266–276, 1995.

57 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, 1981.