Evaluating Graph Queries Using Semantic Treewidth

Cristina Feier
University of Warsaw, Poland

Tomasz Gogacz
University of Warsaw, Poland

Filip Murlak
University of Warsaw, Poland

Abstract

Unions of conjunctive two-way regular path queries (UC2RPQs) are a common abstraction of query languages for graph databases, much like unions of conjunctive queries (UCQs) in the relational case. As in the case of UCQs, their evaluation is NP-complete in combined complexity. Semantic tree-width, i.e. the minimal treewidth of equivalent queries, has been proposed as a candidate criterion to characterize fixed-parameter tractability of UC2RPQs. It was recently shown how to decide the semantic tree-width of a UC2RPQ, by constructing the best under-approximation of a given treewidth, in the form of a UC2RPQ of size doubly exponential in the size of the original query. This leads to an fpt algorithm for evaluating UC2RPQs of semantic TW k which runs in time doubly exponential in the size of the parameter, i.e. in the UC2RPQ. Here we describe a more efficient fpt algorithm for evaluating UC2RPQs of semantic treewidth k which runs in time singly exponential in the size of the parameter. We do this by a careful construction of a witness query which, while still being doubly exponential, can be represented as a Datalog program of bounded width and singly exponential size.

2012 ACM Subject Classification Theory of computation → Regular languages; Information systems → Query languages; Theory of computation → Semantics and reasoning; Theory of computation → Automated reasoning; Theory of computation → Complexity theory and logic

Keywords and phrases conjunctive two-way regular path queries, fixed-parameter tractable evaluation, semantic treewidth, Datalog encoding, optimization

Digital Object Identifier 10.4230/LIPIcs.ICDT.2024.22

Funding This work was supported by Poland's National Science Centre grant 2018/30/E/ST6/00042.

1 Introduction

"The future is big graphs" [21]. Already today graph databases are mainstream in a wide range of application domains, including social networks, fraud detection, biological networks, bioinformatics, cheminformatics, medical data, and knowledge management [3]. By 2025, as Gartner predicts, graph technologies will be used in 80% of data and analytics innovations. The landscape of graph technologies, however, is currently very fragmented, with multiple vendors offering their own query languages (e.g. [13, 23]). This might change with the upcoming Graph Query Language Standard [11], but for now theoretical research naturally focuses on abstract formalisms, capturing the core of the multiple query languages.

While for relational databases conjunctive queries (CQs) and their unions (UCQs) are a premier abstract query language, graph databases are typically queried using *conjunctive regular path queries* (CRPQs) and unions thereof (UCRPQs) which generalize UCQs by replacing atoms with *regular path queries* (RPQs) [19]. RPQs specify connections between graph nodes using regular expressions over edge labels. If edges can be traversed both ways, one speaks of *two-way regular path queries* (2RPQs) and *(unions of) conjunctive two-way*

regular path queries, abbreviated as (U)C2RPQs. While the original semantics for such queries was based on the existence of simple paths, the most commonly adopted semantics nowadays relies on unrestricted paths. Under the latter semantics, 2RPQs can be evaluated in polynomial time, but the complexity of evaluating C2RPQs is the same as for CQs: NP-complete in combined complexity.

In the classical setting, the intractability of CQs lead to a long line of research, where different structural measures on queries were proposed in an attempt to identify cases where evaluation can be done efficiently. This spans from the well-known polynomial time algorithm of Yannakakis for evaluating acyclic CQs [24] to tractability results for queries of bounded treewidth [9], bounded (fractional) hypertreewidth [14, 16], and culminates with the result of Grohe [15] that establishes a dichotomy for fixed-parameter tractable evaluation of CQs in the bounded arity setting.

Under an assumption from parameterized complexity theory, that W[1] $\neq$ FPT, Grohe's result establishes that exactly those classes of UCQs which have bounded treewidth modulo equivalence (also known as *bounded semantic treewidth*) can be evaluated in FPT. Furthermore, bounded semantic treewidth also guarantees tractability in a classical sense, i.e. polynomial-time combined complexity. In the unbounded arity setting, a similar dichotomy (again subject to an assumption, the Exponential Time Hypothesis [17]) has been established, this time using another measure called submodular width [18, 10].

In the same vein, Barcelo et al. [4, 5] initiated the investigation of efficient evaluation of UC2RPQs. They showed that acyclic UC2RPQs can be evaluated in polynomial time (even linear in particular cases). Semantically acyclic UC2RPQs [6] enjoy the same good properties, although deciding whether a UC2RPQ is semantically acyclic is ExpSpace-complete (unlike for UCQs, for which it is NP-complete). This is not surprising given that the containment problem for UC2RPQs is ExpSpace-complete [8].

As concerns semantic treewidth of (U)C2RPQs, Romero et al. [20] introduced two notions of C2RPQ equivalence: one based on homomorphisms, and another based on logical equivalence. They show that there exists a notion of homomorphism for C2RPQs under which bounded treewidth modulo equivalence guarantees polynomial evaluation. For logical equivalence, however, tractability is no longer achievable, so the focus shifts to fixed parameter tractability. In this case, it follows from existing results on UCQ evaluation and RPQ evaluation that UC2RPQs of bounded treewidth are fixed-parameter tractable: one can simply materialize all the RPQs in the database and then regard the C2RPQ as a CQ over the materialized database. The authors lift this to UC2RPQs of bounded semantic treewidth, i.e. UC2RPQs that are logically equivalent to one of bounded treewidth. They achieve this by computing a so-called witness for bounded semantic treewidth, whose actual treewidth might be up to $2k + 1$, when the semantic treewidth of the query is k. Again, according to results on UCQ evaluation, together with the complexity of evaluating RPQs, such a query can be evaluated in time $O(f(\Phi)|\mathcal{G}|^{2k+1})$, where $|\Phi|$ is the size of the query, $|\mathcal{G}|$ is the size of the graph database and f is a singly exponential function.

It remained open how to decide semantic TW of UC2RPQs, and, in particular, how to construct a witness of optimal TW. This has been settled recently by Figueira and Morvan [12], who construct for a given k the best (in the sense of tightest) under-approximations of treewidth k of the original query. Such under-approximations have in the worst case size doubly exponential in the size of the original query. As such an under-approximation is equivalent to the original query when the semantic treewidth of the query is k, this leads to an algorithm for evaluating UC2RPQs of semantic TW k which runs in time $O(f(\Phi)|\mathcal{G}|^{k+1})$, with f being a doubly exponential function. While this is a significant improvement in data

complexity, as it lowers the exponent of the size of data from $2k+1$ to $k+1$ compared to existing algorithms, the algorithm is impractical due to its doubly exponential combined complexity.

In this paper, we zoom into the problem of evaluating a UC2RPQ Φ of semantic TW k and show that it is possible to evaluate Φ in time $O(g(\Phi)|\mathcal{G}|^{k+1})$, with g a singly exponential function. We do this by a careful construction of optimal approximations of a given treewidth (and implicitly of semantic treewidth witness queries). We construct several such approximations, starting with an infinitary one which we refine to ensure some good structural properties, and ending with a finite witness of bounded size. As the finite witness in the worst case is still doubly exponential in the size of Φ (as in [12]) we cannot use it to get better complexity bounds. However, we can exploit the infinitary witness with good structural properties. This query can be compiled into a union of singly exponentially many queries, called *skeleton* queries.

Skeleton queries group together C2RPQs which share some common structure, the *skeleton*. They can be seen as tree-shaped conjunctions of so-called *type reachability queries*. Such type reachability queries encode all ways in which a top $k+1$ tuple can be reached from a bottom $k+1$ tuple via a path query of width k. While there might be doubly exponentially many such ways to reach a tuple, the type reachability query can be encoded via an exponential sized Datalog program of width $k+1$ by employing suitable notions of types and compatibility between such types. As concerns skeleton queries, their tree-based structure makes them amenable to evaluation via a dynamic programming approach. The approach is reminiscent of Yannakakis' algorithm for evaluating acyclic CQs [24], if one regards type reachability queries as oracles that compute a set of target $(k+1)$-tuples when seeded with a set of source $(k+1)$-tuples. Again, this step can be encoded via a Datalog program of width $k+1$ and polynomial size. By executing such a Datalog program, we obtain an evaluation procedure which runs within the desired time bounds.

The paper is organized as follows. We start with some preliminaries in Section 2. Section 3 is dedicated to constructing TW k approximations. Then, in Section 4 we encode the evaluation of TW k approximations into a Datalog program, to achieve higher efficiency. Finally, in Section 5 we conclude and discuss future work.

2 Preliminaries

Relational structures, graph databases

A *schema* $\mathbf{S}$ is a finite set of relational symbols with associated arities. An $\mathbf{S}$-*fact* has the form $r(\mathbf{a})$, where $r \in \mathbf{S}$, and $\mathbf{a}$ is a tuple of constants of size the arity of r. An $\mathbf{S}$-*structure* A is a set of $\mathbf{S}$-facts. The domain of a structure A, $\mathsf{dom}(A)$, is the set of constants which occur in facts in A. A structure A *maps into* a structure B, written $A \to B$ if there exists a function $h : \mathsf{dom}(A) \to \mathsf{dom}(B)$ such that for every fact $r(\mathbf{a})$ in A, there exists a fact $r(h(\mathbf{a}))$ in B, where $h(\mathbf{a})$ is the tuple obtained from $\mathbf{a}$ by pointwise application of h. In this case, h is said to be a *homomorphism from A to B*. The Gaifman graph of a structure A is an undirected graph (V, E) with $V = \mathsf{dom}(A)$ and $\{a_1, a_2\} \in E$ iff there exists some fact $r(\mathbf{a})$ in A such that both a_1 and a_2 occur in $\mathbf{a}$.

In the following, let Σ be some countable alphabet. A *graph database* $\mathcal{G}$ over Σ is a finite directed graph with edges labeled with symbols from Σ. It can be seen as a relational structure over the schema $\{R_a \mid a \in \Sigma\}$, where each relational symbol is binary. A path p in a graph database is a sequence $(u_1 \xrightarrow{a_1} u_2 \dots u_l \xrightarrow{a_l} u_{l+1})$ denoting that for every $1 \le i \le l$, $R_{a_i}(u_i, u_{i+1})$ is in $\mathcal{G}$. The label of p as above, denoted $\lambda(p)$, is the word $a_1 \dots a_l$ from Σ^*.

Regular expressions, automata

A *non-deterministic finite automata (NFA)* A is a tuple $(Q, \Sigma, \delta, s_0, F)$, where Q is the *set of states*, $\delta\colon Q \times \Sigma \to 2^Q$ is the *transition function*, s_0 is the *initial state* and $F \subseteq Q$ is a *set of final states*. To access the set of states of any given automaton we use the function $\mathsf{states}(\cdot)$; that is, $\mathsf{states}(A) = Q$. A *run of* A on a word $w \in \Sigma^*$ is a sequence of states of the form $(s_0, \ldots, s_n)$, where $n = |w|$ and for every $1 \leq i \leq n$, $s_i \in \delta(w_i, s_{i-1})$. Automaton A accepts a word w if there exists a run $(s_0, \ldots, s_n)$ of A on w such that $s_n \in F$.

Every regular expression can be represented via an NFA of linear size. For such an expression L, we denote with $A(L)$ a fixed linear-size NFA for L. Conversely, each NFA A can be converted into a regular expression $E(A)$. For an NFA A and two states s, s' we denote with $A(s, s')$ the NFA having the same set of states and transition function as A, but having s as initial state and s' as a unique final state. When convenient, we abbreviate $E(A(L)(s, s'))$ as $L[s, s']$. While $L[s, s']$ may be exponentially larger than L as its computation involves the determinization of the NFA L[22], we actually always work with the NFA representation, whose size does not grow.

UC2RPQs

A *regular path query (RPQ)* is a query of the form $L(u, v)$ where L is a regular expression over Σ. For a graph database $\mathcal{G}$ and $u', v' \in \mathsf{dom}(\mathcal{G})$, it is the case that $\mathcal{G} \models L(u', v')$ if there exists a path p in $\mathcal{G}$ from u' to v' such that $\lambda(p) \in L$. When one is interested in two-way navigation along labeled edges in a graph database, the regular expression L can be over the alphabet $\Sigma^{\pm} = \Sigma \cup \{a^- \mid a \in \Sigma\}$. In this case, $L(u, v)$ is said to be a *two-way RPQ (2RPQ)* and it is evaluated over the *completion* $\mathcal{G}^{\pm}$ of a graph database with "reverse" edges: $\mathcal{G}^{\pm} = \mathcal{G} \cup \{R_{a^-}(v, u) \mid R_a(u, v) \in \mathcal{G}\}$. A *conjunctive two-way regular path query (C2RPQ)* $\phi(\mathbf{v}')$ is a query of the form $\exists \mathbf{u}' \bigwedge_{1 \leq i \leq n} L(u_i, v_i)$ where each $L(u_i, v_i)$ is a 2RPQ and $(\mathbf{v}', \mathbf{u}')$ is a disjoint partition of the set of variables $\mathbf{v} \cup \mathbf{u}$ occurring in the query. The variables $\mathbf{v}'$ are called the *free* variables of ϕ. When $\mathbf{v}' = \emptyset$, we say that ϕ is *Boolean*. If $\mathbf{v}' = \mathbf{v} \cup \mathbf{u}$, $\phi(\mathbf{v}')$ is said to be a *full* C2RPQ. If all atoms in a C2RPQ ϕ are of the form $L(u, v)$, with L a language over Σ, we say that ϕ is a *conjunctive query* (CQ). For $\mathcal{G}$ a graph database and $\mathbf{a}$ a tuple of constants from $\mathsf{dom}(\mathcal{G})$, it is the case that $\mathcal{G} \models \phi(\mathbf{a})$ if there exists some mapping $h\colon \mathbf{u} \cup \mathbf{v} \to \mathsf{dom}(\mathcal{G})$ such that $h(\mathbf{v}') = \mathbf{a}$ and $\mathcal{G} \models L(h(u_i), h(v_i))$, for every $1 \leq i \leq n$. When ϕ is Boolean, we say that h is a *homomorphism* from ϕ to $\mathcal{G}$.

A UC2RPQ $\Phi(\mathbf{v}')$ is a union of C2RPQs with free variables $\mathbf{v}'$. When $\mathbf{v}' = \emptyset$, Φ is said to be *Boolean*. For $\mathcal{G}$ a graph database and $\mathbf{a}$ a tuple of constants from $\mathsf{dom}(\mathcal{G})$, $\mathcal{G} \models \Phi(\mathbf{a})$, if $\mathcal{G} \models \phi(\mathbf{a})$ for some C2RPQ ϕ in Φ. We say that a UC2RPQ Φ_1 *is contained* in another UC2RPQ Φ_2, denoted $\Phi_1 \subseteq \Phi_2$, if for every graph database $\mathcal{G}$ and every tuple $\mathbf{a}$ of constants from $\mathsf{dom}(\mathcal{G})$, $\mathcal{G} \models \Phi_1(\mathbf{a})$ implies $\mathcal{G} \models \Phi_2(\mathbf{a})$. If $\Phi_1 \subseteq \Phi_2$ and $\Phi_2 \subseteq \Phi_1$, we say that Φ_1 and Φ_2 are *equivalent*, and we write $\Phi_1 \equiv \Phi_2$.

Trees, tree decompositions, semantic treewidth

A *tree* T is represented as an acyclic undirected graph (V, E), with a distinguished node $r \in V$, the *root*. Given a tree T as above and two nodes $v_1, v_2 \in V$, the unique simple path in T from v_1 to v_2 is denoted as $\mathsf{path}_T(v_1, v_2)$. We write $v \leq_T v'$ if $v \in \mathsf{path}_T(r, v')$ and $v <_T v'$ if $v \in \mathsf{path}_T(r, v')$ and $v \neq v'$. The *lowest common ancestor of* v_1 *and* v_2 *in* T, denoted as $\mathsf{lca}_T(v_1, v_2)$, is the node v such that $v \leq_T v_1$, $v \leq_T v_2$, and $v \in \mathsf{path}_T(v_1, v_2)$. We write $v_1 \not\leq_T v_2$ if v_1 and v_2 are $\leq_T$*-incomparable*; that is, neither $v_1 \leq_T v_2$ nor $v_1 \geq_T v_2$.

A *tree decomposition* of a graph $G = (V, E)$ is a pair $\delta = (T_\delta, \chi)$, with $T_\delta = (V_\delta, E_\delta)$ a tree with root r_δ, and χ a labeling function $V_\delta \to 2^V$ such that:

1. $\bigcup_{t \in V_\delta} \chi(t) = V$.
2. If $(v_1, v_2) \in E$, then $v_1, v_2 \subseteq \chi(t)$ for some $t \in V_\delta$.
3. For each $v \in V$, the set of nodes $\{t \in V_\delta \mid v \in \chi(t)\}$ induces a connected subtree of T_δ.

We will refer to elements of V_δ as *bags* of the tree decomposition and to $\chi(v)$ as the *content* of bag v, for $v \in V_\delta$. For each $u \in V$, we denote with u_δ the root of the subtree $\{v \in V_\delta \mid u \in \chi(v)\}$ of T_δ, i.e. the node v from V_δ for which there is no other node $v' \in V_\delta$ such that $u \in \chi(v')$, and $v' <_{T_\delta} v$. The width of δ is $\mathsf{max}_{v \in V_\delta}(|\chi(v)| - 1)$.

The *treewidth (TW)* of G, $\mathsf{TW}(G)$, is the smallest k such that there exists a tree decomposition δ of G of width k.

For a graph database D, the treewidth of D, $\mathsf{TW}(D)$, is the treewidth of its Gaifman graph G_D. For a Boolean C2RPQ ϕ with variables $\mathbf{v}$, we denote with G_ϕ, the graph (V, E) with $V = \mathbf{v}$ and $E = \{(u, v) \mid L(u, v) \in \phi\}$. The treewidth of ϕ, denoted $\mathsf{TW}(\phi)$, is $\mathsf{TW}(G_\phi)$. The treewidth of a Boolean UC2RPQ Φ is the maximum treewidth among its C2RPQs: $\mathsf{TW}(\Phi) = \mathsf{max}_{\phi \in \Phi} \mathsf{TW}(\phi)$.

The *semantic treewidth of a UC2RPQ* Φ is the minimum treewidth of a UC2RPQ which is equivalent to it.

Parameterized complexity

For a finite alphabet Σ, a *parameterized problem* is a tuple (P, κ), where $P \subseteq \Sigma^*$ is a problem, and $\kappa \colon \Sigma^* \to \mathbb{N}$ is a PTime computable function called the *parameterization* of P. Such a parameterized problem is *fixed-parameter tractable* if there exists an algorithm for deciding P for an input $x \in \Sigma^*$ in time $f(\kappa(x))poly(|x|)$, where f is a computable function and *poly* is a polynomial. The class of all fixed-parameter tractable problems is denoted as FPT. In this paper we are interested in the parameterized problem of evaluating Boolean UC2RPQs, where the parameter is the size of the query.

Datalog

A *term* is a constant or variable. An *atom* is of the form $p(t_1, \ldots, t_n)$, where p is a predicate symbol of arity n and $t_1, \ldots, t_n$ are terms. An atom is said to be *ground* if it contains no variables. A Datalog program Π is a set of rules of the form

$$(r)\colon \quad \beta(\mathbf{x}, \mathbf{y}) \to a(\mathbf{x}),$$

where $\beta(\mathbf{x}, \mathbf{y})$ is a set of atoms having as terms constants or variables from $\mathbf{x} \cup \mathbf{y}$ (called the *body* of r), and $a(\mathbf{x})$ is an atom with variables $\mathbf{x}$ (called the *head* of r). Predicate symbols which occur only in atoms in the body of rules are called *EDBs*, while all other predicates are called *IDBs*. The maximum number of variables occuring in some rule in Π is the *width* of Π, denoted $\mathsf{w}(\Pi)$.

A structure I satisfies a Datalog rule r as above, if for every function $h \colon \mathbf{x} \cup \mathbf{y} \to \mathsf{dom}(I)$ which is a homomorphism from $\beta(\mathbf{x}, \mathbf{y})$ to I, $a(h(\mathbf{x})) \in I$. Given a database D and a Datalog program Π, a structure I is a *model of* Π if it extends D (we adopt the *standard name assumption* – constants in $\mathsf{dom}(D)$ are interpreted as themselves) and satisfies every rule from Π. For a ground atom a, we say that $(\Pi, D) \models a$ if for every model I of Π and D, $I \models a$.

Datalog programs (in conjunction with DBs) have the property that they have a finite minimal model [1], i.e. a model $M_{\Pi,D}$ which is a sub-structure of every other model of Π and D. The minimal model $M_{\Pi,D}$ can be computed via a fix-point procedure (called *naive evaluation* or *chase*) which constructs a sequence of converging instances $(I_i)_{i\geq 0}$ as follows: $I_0 = D$; given I_i, I_{i+1} is obtained by considering for every rule in Π all homomorphisms h from $\beta(\mathbf{x},\mathbf{y})$ to I_i and adding $a(\mathbf{x})$ to I. Then, $M_{\Pi,D} = \bigcup I_i$. The procedure runs in time $O(f(|\Pi|)|D|^{\mathsf{w}(\Pi)})$, where f is a polynomial function.

3 Treewidth-k Approximations

In this section we investigate how one can approximate a given UC2RPQ with a UC2RPQ of a given treewidth.

▶ **Definition 1.** *A UC2RPQ Φ' is a* TW-k approximation *of a UC2RPQ Φ if*
(i) *Φ' has TW k;*
(ii) *$\Phi' \subseteq \Phi$; and*
(iii) *there is no UC2RPQ Φ'' of TW k such that $\Phi' \subset \Phi'' \subseteq \Phi$.*

The actual goal of this section is to develop a toolbox for handling TW-k approximations, suitable for our approach to query evaluation, described in Section 4. However, to focus our attention, we set a local goal of re-proving the following result, established in [12].

▶ **Theorem 2.** *For every UC2RPQ Φ and $k > 1$, it is possible to construct a TW-k approximation Φ_k of size doubly exponential in the size of Φ.*

Note that once we know how to construct TW-k approximations, we can compute the semantic treewidth of a given UC2RPQ by constructing TW-k approximations for increasing values of k and testing their equivalence to the original UC2RPQ. It is known that checking containment of UC2RPQs is decidable and can actually be performed in ExpSpace [8].

We construct the TW-k approximation in several steps, starting from an infinitary UC2RPQ and finally obtaining a UC2RPQ with the desired size bounds. For the initial step, we use the following sufficient condition.

▶ **Lemma 3.** *Let Φ be a UC2RPQ and Φ' be a UC2RPQ of TW k which is contained in Φ and is equivalent to Φ on graph databases of TW k. Then Φ' is a TW-k approximation of Φ.*

The proof of Lemma 3 uses the notion of *expansion*, which is a graph database of a certain shape which satisfies a C2RPQ.

▶ **Definition 4.** *Given a C2RPQ Φ and a graph database $\mathcal{G}$ such that $\mathcal{G} \models \Phi$, we say that $\mathcal{G}$ is* an expansion of Φ *if there exists an injective homomorphism h from Φ to $\mathcal{G}$ such that $\mathcal{G}$ can be seen as a set of paths of the form $h(u_i) \to u'_2 \to u'_3 \to \ldots \to h(v_i)$, one for each atom of the form $L(u_i, v_i)$ in Φ, that are pairwise disjoint except (possibly) for their endpoints.*

The following lemma summarises some properties of expansions:

▶ **Lemma 5** (Folklore)**.** *Let Φ be a C2RPQ. The following hold:*
1. *if Φ has TW k, with $k > 1$, so does every expansion $\mathcal{G}$ of Φ*
2. *for every graph database $\mathcal{G}$ such that $\mathcal{G} \models \Phi$, there exists an expansion $\mathcal{G}'$ of Φ such that $\mathcal{G}' \to \mathcal{G}$.*

We are ready to provide the proof of Lemma 3.

Proof. Towards contradiction, suppose that Φ' is not a TW-k approximation of Φ and let Φ'' be one. Then, there exists a graph database $\mathcal{G}$ such that $\mathcal{G} \models \Phi''$ but $\mathcal{G} \not\models \Phi'$. From Lemma 5, there must be an expansion $\mathcal{G}'$ of Φ'' of TW k such that $\mathcal{G}' \to \mathcal{G}$. Clearly, $\mathcal{G}' \models \Phi''$. Since $\Phi'' \subseteq \Phi$, $\mathcal{G}' \models \Phi$. As $\mathcal{G}'$ has TW k and $\Phi' \equiv_k \Phi$, $\mathcal{G}' \models \Phi'$. It follows immediately that $\mathcal{G} \models \Phi'$, which contradicts the initial assumption and concludes the proof. ◀

In the following, we will denote with $\equiv_k$ the notion of equivalence of UC2RPQs on TW-k databases. Similarly, for $\subseteq_k$.

For technical convenience, we shall assume that each C2RPQ has at most one atom of the form $L(x, y)$ for each regular expression L. This is without loss of generality, as we can always replace different occurrences of the same regular expression by equivalent but syntactically different regular expressions. By a *path* in a C2RPQ ψ we mean a sequence $A_0(x_0, x_1), A_1(x_1, x_2), \ldots, A_n(x_n, x_{n+1})$ of atoms from ψ. We say the path is *simple* if variables $x_0, x_1, \ldots, x_{n+1}$ are all different.

Following [20], we rely on *subdivisions* and *quotients* of C2RPQs, which we recall below. For a UC2RPQ Φ, and some $r \geq 1$, the set $\mathbb{SD}_r(\Phi)$ of *r-subdivisions* of Φ is the set of all C2RPQs that can be obtained from a C2RPQ ϕ in Φ as follows: for each atom $L(u, v)$ in ϕ, consider fresh variables $u_1, \ldots, u_l$, with $l < r$, and a sequence of states $s_0, s_1, \ldots, s_{l+1}$ of $A(L)$ such that s_0 and s_{l+1} are an initial and a final state of $A(L)$, and replace $L(u, v)$ in ϕ with a path $L[s_0, s_1](u, u_1), L[s_1, s_2](u_1, u_2), \ldots, L[s_l, s_{l+1}](u_l, v)$. We will refer to this path as the *L-path* and to the introduced atoms as the *L-atoms*. For a set S of C2RPQs, we write $\mathbb{Q}(S)$ for the set of *quotients* of C2RPQ in S; that is, C2RPQs that can be obtained from a C2RPQ in S by variable identification. We let $\mathbb{Q}_k(S)$ be the set of C2RPQs from $\mathbb{Q}(S)$ which have treewidth k.

With that, we can make our first step in constructing the desired approximation of Φ. Let $\Phi_{k,\infty}$ be the infinitary UC2RPQ consisting of all C2RPQs from $\bigcup_{r>1} \mathbb{Q}_k(\mathbb{SD}_r(\Phi))$.

▶ **Lemma 6.** *For each UC2RPQ Φ, $\Phi_{k,\infty}$ is a TW-k approximation of Φ.*

The next step is to simplify the structure of C2RPQs building up the TW-k approximation. This will help us ensure the desired size bounds, and will be crucial in the next section.

Given a C2RPQ ψ and a tree decomposition δ of ψ, we say that a path α in ψ is *unmeandering (wrt. δ)* if each bag of δ covers at most one atom of α, i.e. it contains at most one such atom with all arguments belonging to the bag.

Consider a C2RPQ $\psi \in \mathbb{Q}_k(\mathbb{SD}_r(\Phi))$ obtained by quotienting an r-subdivision of a C2RPQ ϕ from Φ. Identifying variables does not break paths, in the sense that L-paths are still present in ψ, except that they need not be simple any more. We call a tree decomposition δ of ψ *unmeandering* if each L-path in ψ is unmeandering wrt. δ.

Let $\Phi'_{k,\infty}$ be the UC2RPQ obtained from $\Phi_{k,\infty}$ by keeping only those C2RPQs that admit an unmeandering tree decomposition of width k, and dropping the remaining ones. By shortcutting meanders as shown in Figure 1, we can refine Lemma 6 as follows.

▶ **Lemma 7.** *For each UC2RPQ Φ, $\Phi'_{k,\infty}$ is a TW-k approximation of Φ.*

Proof. By construction, $\Phi'_{k,\infty} \subseteq \Phi_{k,\infty} \subseteq \Phi$. We show that $\Phi_{k,\infty} \subseteq \Phi'_{k,\infty}$ and thus, $\Phi \subseteq_k \Phi'_{k,\infty}$. Let ϕ' be a C2RPQ from $\Phi_{k,\infty}$. That is, ϕ' is obtained from a C2RPQ ϕ of Φ by an r-subdivision followed by variable identification, and ϕ' has TW k. Let $\delta = (T, \chi)$ be a TW-k decomposition of ϕ' and let $\mathcal{G}$ be a database such that $\mathcal{G} \models \phi'$. We construct a query ϕ'' from $\Phi'_{k,\infty}$ such that $\mathcal{G} \models \phi''$. For every atom $L(u, v)$ in ϕ, query ϕ' has atoms $L[s_i, s_{i+1}](v_i, v_{i+1})$ for $0 \leq i < l$. We obtain ϕ'' from ϕ' by merging these atoms exhaustively, as follows. As long

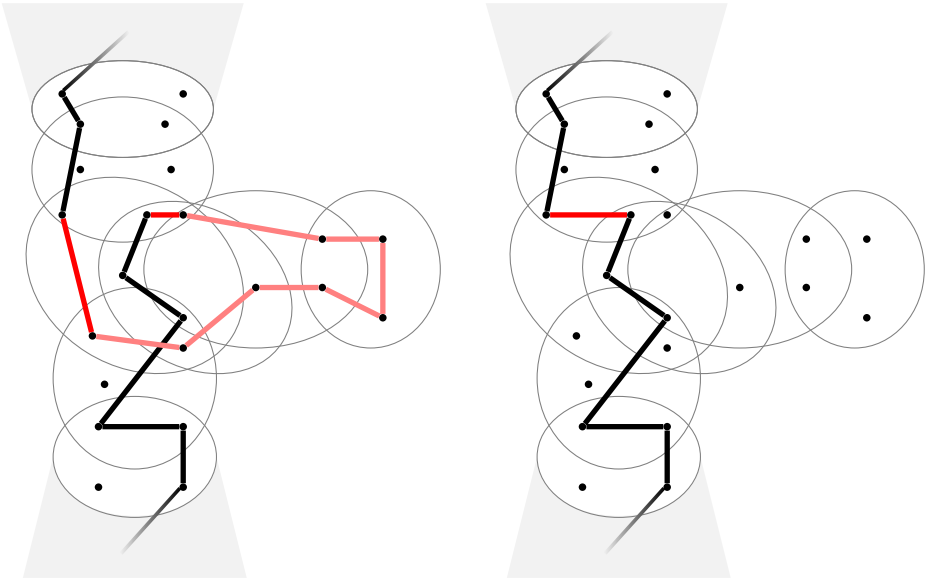

Figure 1 Shortcutting a meander. Whenever two atoms (dark red edges on the left) of some L-path are covered by the same bag, we replace the whole segment between them (dark and light red edges on the left) with a single summarizing atom (dark red on the right).

as some bag of δ covers two atoms of the forms $L[s_i, s_{i'}](v_i, v_{i'})$ and $L[s_{j'}, s_j](v_{j'}, v_j)$ with $i' \leq j'$, we replace them by $L[s_i, s_j](v_i, v_j)$. After each merging step, the current version of ϕ' satisfies the following invariant:

- it has TW k (a witnessing decompositon is obtained from δ by dropping unused variables);
- it is satisfied in $\mathcal{G}$ (via the same homomorphism restricted to currently used variables);
- it can be obtained from ϕ via an r-subdivision followed by variable identification; and
- it contains L-atoms $L[s_{i_0}, s_{i_1}](v_{i_0}, v_{i_1}), \ldots, L[s_{i_m}, s_{i_{m+1}}](v_{i_m}, v_{i_{m+1}})$ for some $m \leq l$ and $0 = i_0 < i_1 < \cdots < i_{m+1} = l + 1$.

When no more merges of L atoms are possible, the resulting query still satisfies the invariant and additionally no bag of δ covers more than one of its L-atoms. We perform this exhaustive procedure for each atom $L(u, v)$ in ϕ. The resulting query ϕ'' belongs to $\Phi'_{k,\infty}$ and holds in $\mathcal{G}$, as required. ◀

We now prove a strong structural property for unmeandering paths, showing that they manifest in tree decompositions in one of three simple ways, illustrated in Figure 2.

▶ **Lemma 8.** *Consider a C2RPQ ψ, a tree decomposition $\delta = (T, \chi)$ of ψ, and an unmeandering path $A_0(x_0, x_1), A_1(x_1, x_2), \ldots, A_n(x_n, x_{n+1})$ in ψ.*

Then, there exists some $0 \leq i \leq n + 1$ such that:

$$\delta(x_0) >_T \delta(x_1) >_T \cdots >_T \delta(x_i) \leq_T \delta(x_{i+1}) <_T \delta(x_{i+2}) <_T \cdots <_T \delta(x_{n+1})$$

Moreover, if $1 \leq i \leq n$, then either:

1. $\delta(x_{i-1}) \not\leq_T \delta(x_{i+1})$ *(Figure 2A), or*
2. $\delta(x_{i-1}) >_T \delta(x_{i+1})$ *and* $\delta(x_{i-1}) \not\leq_T \delta(x_j)$ *for all $j > i + 1$ (Figure 2B), or*
3. $\delta(x_{i-1}) <_T \delta(x_{i+1})$ *and* $\delta(x_j) \not\leq_T \delta(x_{i+1})$ *for all $j < i - 1$ (Figure 2C).*

In particular, $\delta(x_0), \delta(x_1), \ldots, \delta(x_{n+1})$ are all different, except potentially $\delta(x_i)$ and $\delta(x_{i+1})$. Also, if x_j and $x_{j'}$ belong to the same bag, then $|j - j'| \leq 1$.

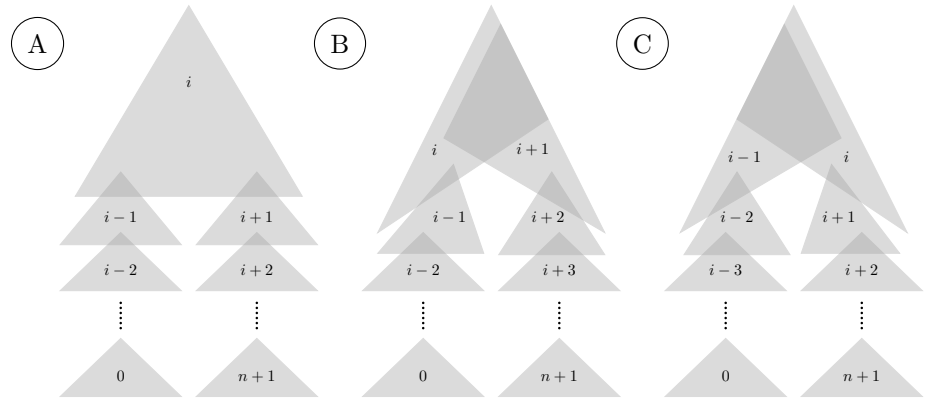

Figure 2 An unmeandering path in a tree decomposition. A triangle with label j represents the subtree of the decomposition consisting of bags containing the j-th variable of the path.

Note that Lemma 8 implies that each unmeandering path is simple, unless it consists of a single atom. Indeed, while $\delta(x_i)$ and $\delta(x_{i+1})$ might be equal, $x_i = x_{i+1}$ would mean that each bag covering $A_{i-1}(x_{i-1}, x_i)$ or $A_{i+1}(x_{i+1}, x_{i+2})$ also covers $A_i(x_i, x_{i+1})$.

As the last step before establishing the size bound, we use Lemma 8 to simplify tree decompositions of queries in $\Phi'_{k,\infty}$. Given a tree decomposition $\delta = (T, \chi)$ of a C2RPQ ψ, and an unmeandering path $\alpha = A_0(x_0, x_1), \ldots, A_n(x_n, x_{n+1})$, we define the *chevron* of α, written $\widehat{\alpha}$, as the union of two shortest paths in T: from $\delta(x_0)$ to $\delta(x_i)$ and from $\delta(x_i)$ to $\delta(x_{n+1})$. By the *main path* of the chevron we mean the shortest path from $\delta(x_0)$ to $\delta(x_{n+1})$. Note that the main path is contained in the chevron and the highest node on the main path is the lowest common ancestor of $\delta(x_0)$ to $\delta(x_{n+1})$. We call it the *key* of the chevron. We refer to $\delta(x_0)$ and $\delta(x_{n+1})$ as the *endpoints* of the chevron, and to $\delta(x_i)$ as the *tip*. We divide the main path into two parts: the *up-path* from $\delta(x_0)$ to the key, and the *down-path* from the key to $\delta(x_{n+1})$. Note that $\widehat{\alpha}$ contains all $\delta(x_0), \delta(x_1), \ldots, \delta(x_{n+1})$, and each atom $A_j(x_j, x_{j+1})$ of α is covered by a bag from the main path of $\widehat{\alpha}$. Given a C2RPQ $\psi \in \mathbb{Q}_k(\mathbb{SD}_r(\Phi))$, a *chevron tree decomposition* of ψ is any unmeandering tree decomposition in which each bag belongs to the chevron of some L-path in ψ.

▶ **Lemma 9.** *If a C2RPQ $\psi \in \mathbb{Q}_k(\mathbb{SD}_r(\Phi))$ has an unmeandering tree decomposition of width k, then it has a chevron tree decomposition of width k.*

Consider a chevron tree decomposition $\delta = (T, \chi)$ of a C2RPQ $\psi \in \mathbb{Q}_k(\mathbb{SD}_r(\Phi))$. By a *critical* node in T we mean a leaf, a root, a node that has at least two children, or an endpoint of the chevron of an L-path.A *segment* in δ is a path of the form $\mathsf{path}_T(u, v)$ for some critical nodes u and v such that $u <_T v$, and all remaining nodes in $\mathsf{path}_T(u, v)$ are non-critical. The number of leaves in δ is bounded by the number of variables in the original C2RPQ $\phi \in \Phi$ from which ψ was obtained, and there are at most twice as many critical nodes. There is no bound, however, on the length of the segments. Our last step in the proof of Theorem 2 essentially consists in providing such a bound. We begin with two preparatory observations, which will be also useful in Section 4.

▷ Claim 10. If a segment p in δ shares a non-critical node with the main path of a chevron $\widehat{\alpha}$ of an L-path α in ψ, then p is contained in the up-path or the down-path of $\widehat{\alpha}$ (but not both).

We write $\mathcal{L}(p)$ for the set of L-atoms such that segment p is contained in the main path of the chevron of the L-path in ψ. We also define $\mathsf{dir}_p \colon \mathcal{L}(p) \to \{\uparrow, \downarrow\}$ as $\mathsf{dir}_p(L) = \uparrow$ if p is contained in the up-path of the chevron of the L-path, and $\mathsf{dir}_p(L) = \downarrow$ if p is contained in the down-path of the chevron of the L-path.

▷ Claim 11. Let $p = \mathsf{path}_T(u, v)$ be a segment in δ and $\alpha = A_0(x_0, x_1), \ldots, A_n(x_n, x_{n+1})$ an L-path in ψ for some $L \in \mathcal{L}(p)$. Then, the atoms of α covered by bags from p are

$$A_\ell(x_\ell, x_{\ell+1}), A_{\ell+1}(x_{\ell+1}, x_{\ell+2}), \ldots, A_{m-1}(x_{m-1}, x_m)$$

for some $0 \leq \ell \leq m \leq n+1$ with $x_\ell \in \chi(u)$ and $x_m \in \chi(v)$, and there are nodes $u_\ell, u_{\ell+1}, \ldots, u_{m-1} \in p$ such that u_j covers $A_j(x_j, x_{j+1})$ for all $\ell \leq j < m$, and either $u_\ell <_T u_{\ell+1} <_T \cdots <_T u_{m-1}$ or $u_\ell >_T u_{\ell+1} >_T \cdots >_T u_{m-1}$.

We are now ready for the final step in the proof of Theorem 2. So far we have constructed an infinitary TW-k approximation based on arbitrary subdivisions of atoms in the original UC2RPQ. We next show that it suffices to consider bounded subdivisions. For a given r, let $\Phi'_{k,r}$ be the UC2RPQ obtained from $\Phi'_{k,\infty}$ by keeping only those C2RPQs that belong to $\mathbb{Q}_k(\mathbb{SD}_r(\Phi))$, and dropping the remaining ones.

▶ **Lemma 12.** *For each UC2RPQ Φ, there is a positive integer r, singly exponential in the size of Φ, such that $\Phi'_{k,r}$ is a TW-k approximation of Φ.*

Proof. We show that $\Phi'_{k,\infty} \subseteq \Phi'_{k,r}$ for some $r < \infty$ whose value will follow from the construction. Take ψ from $\Phi'_{k,\infty}$ and let $\delta = (T, \chi)$ be a chevron tree decomposition of ψ. We transform ψ into ψ' from $\Phi'_{k,r}$ for such that $\psi \subseteq \psi'$.

We have already seen that the number of critical nodes in δ is bounded linearly in the size of Φ, but the segments in δ can be arbitrarily long. We obtain ψ' by shrinking the segments.

Consider a segment p in δ and an internal node v in p. We annotate variables in v with multiple pairs of the form (L, s), where s is a state of the automaton for L. For each $L \in \mathcal{L}(p)$ with $\mathsf{dir}_p(L) = \downarrow$, we annotate x with (L, s) if there is an atom $L[s, s'](x, x')$ in ψ. For each $L \in \mathcal{L}(p)$ with $\mathsf{dir}_p(L) = \uparrow$, we annotate x with (L, s) if there is an atom $L[s', s](x', x)$ in ψ. Let $\zeta(v)$ be the set of variables in $\chi(v)$ that have non-empty annotation. We say that internal nodes v_1 and v_2 are *alike* if there is an isomorphism between ψ restricted to $\chi(v_1)$ and ψ restricted to $\chi(v_2)$ that preserves annotations and is identity over $\chi(v_1) \cap \chi(v_2)$. (Note that annotations of the same variable in different bags in p are the same.)

Suppose that segment p contains two internal nodes v_1 and v_2 such that $v_1 <_T v_2$ and v_1 and v_2 are alike via an isomorphism ι. Let u_1 be the parent of v_1, and u_2 the parent of v_2. Note that u_1 and u_2 belong to p, and $v_1 \leq_T u_2$. We can then shrink p as follows. We replace the subtree of T rooted at v_1 with the subtree of T rooted at v_2, and replace each variable $x \in \chi(u_1) \cap \chi(v_1)$ with $\iota(x)$, both in the tree decomposition and in the query. Note that this removes all bags from $\mathsf{path}_T(v_1, u_2)$. We modify the query accordingly: we remove all atoms that use a variable that does not occur in the modified tree decomposition.

Recalling the characterization of Claim 11, we can see that the effect this has on the query is that for each $L \in \mathcal{L}(p)$, a subpath

$$L_j[s_j, s_{j+1}](x_j, x_{j+1}), L_{j+1}[s_{j+1}, s_{j+2}](x_{j+1}, x_{j+2}), \ldots L_{k-1}[s_{k-1}, s_k](x_{k-1}, x_k)$$

with $s_{j+1} = s_k$ gets replaced with $L_j[s_j, s_k](x_j, x_k)$, as shown in Figure 3. Hence, after a single shrinking, the query still belongs to $\mathbb{Q}(\mathbb{SD}_r(\Phi))$. Also, the modified tree decomposition has still width at most k and is unmeandering. Hence, the modified query belongs to $\Phi'_{k,\infty}$. Finally, if the original query holds in some database, so does the modified one, with the same witnessing mapping of variables to nodes.

We obtain ψ' by repeatedly shrinking segments until it is no longer possible. By the above invariants, $\psi' \in \Phi'_{k,\infty}$ and $\psi \subseteq \psi'$. It remains to see that $\psi' \in \mathbb{Q}_k(\mathbb{SD}_r(\Phi))$ for some r exponential in the size of Φ. Because we the tree decomposition corresponding to ψ'

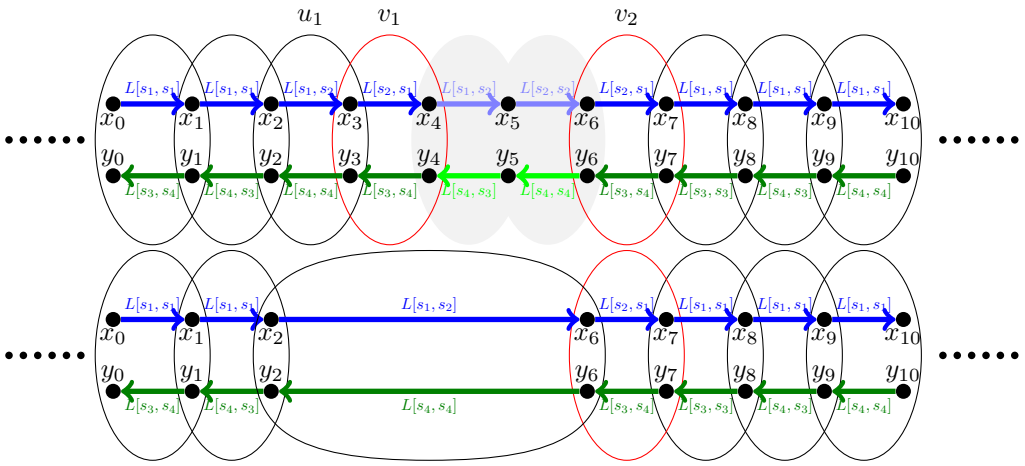

Figure 3 Shrinking a segment.

is unmeandering, the number of atoms in each L-path is bounded by the size of the tree decomposition. We already know that the number of critical nodes is linear in the size of Φ. Hence, so is the number of segments. After shrinking, no two internal bags in a segment are alike, so the length of each segment is bounded by a function single exponential in the size of Φ. Hence, the whole tree decomposition is of size single exponential in the size of Φ, and we are done. ◀

Lemma 12 yields a TW-k approximation of a UC2RPQ Φ in which each C2RPQ has size at most singly exponential in the size of Φ. As there are at most doubly exponentially many such C2RPQs, Theorem 2 follows directly from the lemma.

Related Work

As explained in the beginning of this section, TW-k approximations for UC2RPQs have already been constructed in [12]. Our construction has some common points, but also differences, to the construction from [12]. In both approaches, an infinitary approximation is constructed, which is first normalized, and then is shrunk by bounding the height of tree decompositions. We will address these below.

Normalization of the approximation. In our case the normalization step consists in constructing approximations which admit unmeandering tree decompositions, while in [12] original paths are contracted such that a bag of the tree decomposition is not revisited once it is left. However, inside bags, atoms belonging to original paths might form cyclic paths and they are not contracted. This is achieved by so-called *tagged tree decompositions*. As we will see in the next section, the notion of unmeandering tree decomposition will be crucial for our algorithm to efficiently evaluate UC2RPQs. As part of the algorithm, we will define a notion of types which capture the content of bags that occur in such decompositions. While we do not preclude the possibility to define a similar notion based on decompositions with arbitrary bags, it would be much more cumbersome.

Shrinking of the approximation. Both approaches shrink paths of unbounded length in tree decompositions of approximations. In our approach we annotate bags of tree decompositions and completely remove paths between nodes which are alike w.r.t. annotations. Figueira

and Morvan [12] annotate bags as well, using a less-constrained notion for identifying similar bags. However, due to this less-constrained notion, it is is not possible for them to fully remove the path between such nodes: instead, by using *nice tree decompositions*, they shrink the path between two such similar nodes to a path of constant length.

4 Exploiting Semantic Treewidth

As discussed in the introduction, it is already known from [6] that UC2RPQs of bounded TW are fpt. There, an algorithm based on computing and evaluating a witness of bounded TW is presented. However, the treewidth of the witness might not be optimal: for a UC2RPQ Φ of semantic TW-k, the witness query Φ_w from [6] might have treewidth $2k+1$. Using standard results concerning CQ evaluation [15] and RPQ evaluation [7], it is possible to evaluate Φ_w in time $O(p(|\Phi_w|)|\mathcal{G}|^{2k+2}))$, for some polynomial p: one can first "materialize" all the RPQs which occur in C2RPQs in Φ_w in the database, i.e. create a new database $\mathcal{G}'$ which contains all facts $L(a_1, a_2)$ such that $\mathcal{G} \models L(a_1, a_2)$, for every atom of the form $L(u, v)$ in Φ_w. This can be done in polynomial time in the size of Φ_w and $\mathsf{dom}(\mathcal{G}') = \mathsf{dom}(\mathcal{G})$. In a second step, Φ_w can be seen as a CQ of TW $2k+1$ over $\mathcal{G}'$, which can then be evaluated in time $O(p(|\Phi_w|)|\mathcal{G}'|^{2k+2})$, where p is some polynomial. As $|\Phi_w|$ is bounded by a singly exponential function in $|\Phi|$, one obtains an algorithm for evaluating Φ which runs in time $O(g(|\Phi|)|\mathcal{G}'|^{2k+2})$, for g some singly exponential function.

On the other hand, similarly to [12], in Section 3, for every $k > 1$, we constructed semantic treewidth witnesses with optimal TW in the form of TW k-approximations of size doubly exponential in the size of Φ. Let Φ_k be such an approximation of a UC2RPQ Φ. Using similar arguments as for Φ_w, we obtain that there must be some function g such that Φ_k can be evaluated in time $O(g(|\Phi|)|\mathcal{G}|^{k+1})$, with g a doubly exponential function. While in our parameterized setting, where we expect the size of the data to be much larger than that of the query, this is an important improvement compared to the approach based on the witness of non-optimal TW $2k+1$, the doubly exponential combined complexity makes such an approach prohibitive.

However, as we will show in the following, there is an algorithm for evaluating Φ in time $O(g(\Phi)|\mathcal{G}|^{k+1})$, with g a singly exponential function. The evaluation procedure uses $\Phi'_{k,\infty}$, the infinitary UC2RPQ constructed in Section 3, Lemma 7, which has the nice structural property that all its C2RPQs admit unmeandering tree decompositions. We first observe that $\Phi'_{k,\infty}$ can be regarded as a union of singly exponentially many queries, where each of these queries is an infinitary UC2RPQ which contains all C2RPQs from $\Phi'_{k,\infty}$ which share some common structure, called *skeleton*. We achieve a compact representation of each such UC2RPQ by means of so-called *reachability queries* which are queries which impose reachability conditions from one $k+1$-tuple of variables to another $k+1$-tuple. In this view a *skeleton query* is a tree of reachability queries, one query associated to each edge of the tree. This makes it possible to encode the evaluation of such a query in a Datalog program which simulates a bottom-up evaluation of the skeleton tree. As the program has width $k+1$ and singly exponential size, we obtain the above-mentioned complexity bounds.

Abstracting C2RPQs via Skeletons

Recall that each C2RPQ ϕ' from $\Phi'_{k,\infty}$ is from $\mathbb{Q}_k(\mathbb{SD}_l(\phi))$, for some $\phi \in \Phi$, and $l > 0$ and, thus there exists a natural mapping h from $\mathsf{var}(\phi)$ to $\mathsf{var}(\phi')$. Given a tree decomposition δ' for ϕ' one can define the set of critical nodes and segments of δ', denoted here as $\mathsf{crit}(\delta')$, and $\mathsf{seg}(\delta')$, as in Section 3. A *segment signature* σ is a tuple of the form $(\mathcal{L}_\sigma, \mathsf{dir}_\sigma)$, where $\mathcal{L}_\sigma$ is

a set of regular expressions and $\mathsf{dir}_\sigma \colon \mathcal{L}_\sigma \to \{\uparrow, \downarrow\}$. For every segment $p \in \mathsf{seg}(\delta')$, we denote with $\mathsf{sig}(p)$ its segment signature defined as follows: $\mathcal{L}_{\mathsf{sig}(p)} = \mathcal{L}(p)$ and $\mathsf{dir}_{\mathsf{sig}(p)}(L) = \mathsf{dir}_p(L)$, for every $L \in \mathcal{L}_{\mathsf{sig}(p)}$. We denote with $\mathsf{sig}(\delta')$ the set of all segment signatures $\mathsf{sig}(p)$, where $p \in \mathsf{seg}(\delta')$ and with $\mathsf{sig}(\Phi'_{k,\infty})$ the union of all sets $\mathsf{sig}(\delta')$, where δ' is an unmeandering tree decomposition of some C2RPQ ϕ' from $\Phi'_{k,\infty}$.

We introduce some further notations: an *extended*[1] *C2RPQ* is a C2RPQ in which we allow also atoms of the form $L[s](v)$, where L is a regular expression and $s \in \mathsf{states}(A(L))$. For a C2RPQ θ, we denote with $\mathsf{ext}(\theta)$ the extended C2RPQ obtained from θ by adding atoms $L[s](v)$ and $L[s'](v')$, for every atom of the form $L[s, s'](v, v')$ in θ. Conversely, for an extended C2RPQ γ, we denote with $\mathsf{bin}(\gamma)$, the C2RPQ obtained from γ by maintaining only binary atoms.

The notation var is lifted to extended C2RPQ as expected. Also, given an extended C2RPQ θ and a set of variables $\mathcal{V} \subseteq \mathsf{var}(\theta)$, we denote with $\theta_\mathcal{V}$ the extended C2RPQ obtained from θ by retaining every atom from θ with all arguments from $\mathcal{V}$ (be it binary or unary). Given a set of regular expressions $\mathcal{L}$, and a set of variables $\mathcal{V}$, $\mathbb{C}^{\mathcal{L}}_{k+1}(\mathcal{V})$ is the set of all extended C2RPQs using regular expressions from $\mathcal{L}$ with at most $k+1$ variables from $\mathcal{V}$.

We are now ready to define the data structure which will serve as a common abstraction for different C2RPQs from $\Phi_{k,\infty}$:

▶ **Definition 13.** *Given $\phi' \in \Phi_{k,\infty}$ and $\delta' = (T', \chi')$ a tree decomposition of ϕ', the* skeleton of ϕ' w.r.t. δ', $\mathsf{sk}(\phi', \delta')$, *is a triple (T, q, μ), where:*

1. *$T = (\mathsf{crit}(\delta'), E)$ is the tree induced by $<'_T$ on $\mathsf{crit}(\delta')$;*
2. *$q \colon \mathsf{crit}(\delta') \to \mathbb{C}^{\mathcal{L}_{\phi'}}_{k+1}(\mathsf{var}(\phi'))$ is the function: $q(v) = \mathsf{ext}(\phi')_{\chi'(v)}$, for $v \in \mathsf{crit}(\delta')$; and*
3. *$\mu \colon E \to \mathsf{sig}(\delta')$ is the function: $\mu(v_1, v_2) = \mathsf{sig}(v_2, v_1)$, for $(v_1, v_2) \in E$.*

Intuitively, $\mathsf{sk}(\phi', \delta')$ abstracts ϕ' w.r.t. δ' by keeping only atoms covered by critical bags (endpoints of segments) together with new unary atoms which are pointers to the intersection(s) of L-paths with such a bag; the inner parts of segments are replaced with their signature (captured by edges from E labelled by μ).

Let $\mathsf{sk}(\Phi'_{k,\infty})$ be the set of all $\mathsf{sk}(\phi', \delta')$, where ϕ' is a C2RPQ from $\Phi'_{k,\infty}$ and δ' is an unmeandering tree decomposition of ϕ'. For $sk \in \mathsf{sk}(\Phi'_{k,\infty})$, and ϕ' from $\Phi'_{k,\infty}$, we say that ϕ' *abstracts to* sk if there exists an unmeandering tree decomposition δ' such that $\mathsf{sk}(\phi', \delta') = sk$. In the following, for every $sk \in \mathsf{sk}(\Phi'_{k,\infty})$, we let Φ_{sk} be the UC2RPQ which is the union of all C2RPQs from $\Phi'_{k,\infty}$ which abstract to sk. Thus, Φ_{sk} contains all C2RPQs which share common structure in the form of sk.

As all disjuncts of $\Phi'_{k,\infty}$ are covered by some UC2RPQ of the form Φ_{sk}, with $sk \in \mathsf{sk}(\Phi'_{k,\infty})$, we have that:

$$\Phi'_{k,\infty} = \bigvee_{sk \in \mathsf{sk}(\Phi'_{k,\infty})} \Phi_{sk}.$$

▶ **Proposition 14.** *The set $\mathsf{sk}(\Phi'_{k,\infty})$ is of size singly exponential in $|\Phi|$ and can be computed in singly exponential time in $|\Phi|$.*

In the following, we show how each query Φ_{sk} can be reformulated into a query which mirrors the structure of a skeleton, in the sense that it can be seen as a conjunction of queries, one corresponding to each edge in the tree underlying the skeleton. Each such query can be seen as a type reachability query, i.e. a query which specifies how to reach the top type induced by the segment signature attached to an edge from the corresponding bottom type.

[1] We will not define the semantics of such a C2RPQ as its purpose is to serve mainly as a syntactical object for later constructions.

Types and Reachability Queries

We start by defining σ-types over at most $k+1$ variables, where σ is a segment signature. Intuitively, such a type captures for every $L \in \mathcal{L}_\sigma$ the intersection of the L-path with a bag in some unmeandering tree decomposition belonging to a σ-segment, be it either in the form of an L-atom or a single variable which is visited by the path in a certain state. We use extended C2RPQs to capture such information. In the following, we reserve a set of fresh $2k+1$ variables $\mathbb{V} = \{v_1, \dots v_{2k+1}\}$.

▶ **Definition 15.** *For a segment signature σ, a σ-type τ is an extended full C2RPQ from $\mathbf{C}^{\mathcal{L}_\sigma}_{k+1}(\mathbb{V})$ such that for every $L \in \mathcal{L}_\sigma$, one of the following holds:*
1. *τ contains exactly one atom of the form $L[s, s'](v, v')$;*
2. *τ contains exactly one atom of the form $L[s](v)$.*

Given such a σ-type τ, based on the direction of paths in σ, we can compute enter and exit points in the form of pairs (variable, state) for each path in σ w.r.t. τ. To this purpose, we define functions $\mathsf{in}_\tau \colon \mathcal{L}_\sigma \to \mathsf{var}(\tau) \times \bigcup_{L \in \mathcal{L}_\sigma} \mathsf{states}(A(L))$ and $\mathsf{out}_\tau \colon \mathcal{L}_\sigma \to \mathsf{var}(\tau) \times \bigcup_{L \in \mathcal{L}_\sigma} \mathsf{states}(A(L))$ as follows:
1. if τ contains an L-atom of the form $L[s, s'](v, v')$ and $\mathsf{dir}_s(L) = \uparrow$, then $\mathsf{in}_\tau(L) = (v, s)$ and $\mathsf{out}_\tau(L) = (v', s')$;
2. if τ contains an L-atom of the form $L[s, s'](v, v')$ and $\mathsf{dir}_s(L) = \downarrow$, then $\mathsf{in}_\tau(L) = (v', s')$ and $\mathsf{out}_\tau(L) = (v, s)$;
3. if τ contains no L-atom of the form $L[s, s'](v, v')$, then $\mathsf{in}_\tau(L) = \mathsf{out}_\tau(L) = (v, s)$, where $L[s](v)$ is the unique L-atom from τ.

Conditions (1) and (2) above can be explained as follows. To build segments, types have to be matched. With "out" we single out the node and state from which the path should be continued in the next type and with "in" the node and state which continues the path from a previous type; this is dependent on the direction of the path relative to the segment it traverses. Because segments are built bottom-up, upward paths are built forward and downward paths are built backwards. That is, for downward paths, the first argument of an atom will have to be matched when we stack up another type; for upward paths, the second. Constraint (3) simply ensures that paths are carried over in σ-types; it might be the case that nothing new is added by a type to a certain path, in which case both "in" and "out" refer to the same variable and the same state.

Building on these functions, we introduce a notion of type compatibility which captures the idea that *a σ-type τ_1 is compatible with a σ-type τ_2* if τ_1 can be continued by τ_2.

▶ **Definition 16.** *Given a segment signature σ, and two σ-types τ_1 and τ_2, τ_1* is compatible with *τ_2 if for every $L \in \mathcal{L}_\sigma$ it is the case that either:*
1. *there is some atom of the form $L[s, s'](v_1, v_2) \in \tau_2 \cap \tau_1$, or*
2. *$\mathsf{in}_{\tau_2}(L) = \mathsf{out}_{\tau_1}(L)$;*

Furthermore, for every $v \in \mathsf{var}(\tau_1) \cap \mathsf{var}(\tau_2)$, it must be the case that there exists some $L \in \mathcal{L}_\sigma$ such that either $v = v_1$ or $v = v_2$ as in condition (1) above, or $\mathsf{in}_{\tau_2}(L) = \mathsf{out}_{\tau_1}(L) = v$.

Thus, for a type τ_1 to be continuable by a type τ_2 it must be the case that they either share a common part of a path in the form of an L-atom or the ingoing L-variable of the second type τ_2 coincides with the outgoing L-variable of the first type τ_1. Also, only variables which occur in such a shared atom or are used to continue L-paths from one type to the next, occur in both types.

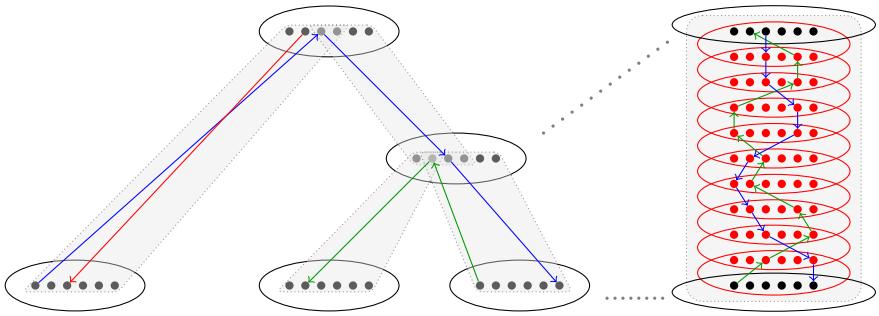

Figure 4 Constructing a skeleton query.

We will next introduce the notion of reachability query between so-called σ-types instantiations. Given a type τ and a tuple of variables $\vec{u}$ such that $|\vec{u}| = |\mathsf{var}(\tau)|$ (we assume that all variables in $\vec{u}$ are distinct), the *instantiation of* τ *with* $\vec{u}$, $\tau(\vec{u})$, is the extended C2RPQ obtained from τ by replacing $\mathsf{var}(\tau)$ with $\vec{u}$.

▶ **Definition 17.** *Given two σ-types instantiations $\tau'_1(\mathbf{x}_1)$ and $\tau'_2(\mathbf{x}_2)$, a* reachability query from $\tau'_1(\mathbf{x}_1)$ to $\tau'_2(\mathbf{x}_2)$, *denoted* $\mathsf{reach}(\tau'_1(\mathbf{x}_1), \tau'_2(\mathbf{x}_2))$ *is the union of all C2RPQs of the form*

$$\exists \mathbf{y} \bigwedge_{0 \le i \le m} \mathsf{bin}(\tau_i(\mathbf{y}_i)),$$

where $m \in \mathbb{N}$, $m \geq 2$, and $(tau_i(\mathbf{y}_i))_{1 \le i \le m}$ is a sequence of type instantiations such that:

1. *$\tau_0(\mathbf{y}_0)$ is $\tau'_1(\mathbf{x}_1)$ and $\tau_m(\mathbf{y}_m)$ is $\tau'_2(\mathbf{x}_2)$;*
2. *for every $0 \leq i < m$: τ_i is compatible with τ_{i+1};*
3. *for every $i \neq j$, $1 \leq l \leq |\mathbf{y}_i|$, and $1 \leq n \leq |\mathbf{y}|_j$: $\mathbf{y}_{i,l} = \mathbf{y}_{j,n}$ iff $\mathbf{z}_{i,l} = \mathbf{z}_{j,n}$ where $\mathbf{z}_i = \mathsf{var}(\tau_i)$ and $\mathbf{z}_j = \mathsf{var}(\tau_j)$, and*
4. *$\mathbf{y} = \bigcup_{0<i<m} \mathbf{y}_i \setminus (\mathbf{x}_1 \cup \mathbf{x}_2)$.*

Thus, the reachability query is a UC2RPQ with free variables $\mathbf{x}_1 \cup \mathbf{x}_2$, in which each C2RPQ can be seen as the conjunction of C2RPQs occurring in type instantiations from a sequence of type instantiations. Condition (3) in Definition 17 asks for tuples $\mathbf{y}_i$ and $\mathbf{y}_j$ to have the same intersection profile as the tuples of variables pertaining to types τ_i and τ_j; for example, if $\mathsf{var}(\tau_i) = (v_1, v_2, v_3)$ and $\mathsf{var}(\tau_j) = (v_2, v_3, v_4)$, then $\mathbf{y}_i$ and $\mathbf{y}_j$ will be of the form $(\_, y_1, y_2)$ and $(y_1, y_2, \_)$, respectively, and $\mathbf{y}_i \cap \mathbf{y}_j = \{y_1, y_2\}$.

Skeleton Queries

We now return to $sk = (T, q, \mu)$ from $\mathsf{sk}(\Phi'_{k,\infty})$. We associate to every edge $e = (v_1, v_2) \in E$ a reachability query between the two $\mu(e)$-type instantiations induced by v_1 and v_2. Recall that for every $v \in V$, $q(v)$ is an extended C2RPQ which contains atoms of the form $L[s, s'](u, u')$ or $L[s](u)$. For $e = (v_1, v_2) \in E$ and $i \in \{1, 2\}$, we can read $\mu(e)$-type instantiations $\tau^i_e(\mathbf{x}^i_e)$ from $q(v_i)$ by simply maintaining all L-atoms with $L \in \mathcal{L}_{\mu(e)}$.

Let $\mathsf{reach}_e(\mathbf{x}^2_e, \mathbf{x}^1_e)$ be an abbreviation for $\mathsf{reach}_e(\tau^2_e(\mathbf{x}^2_e), \tau^1_e(\mathbf{x}^1_e))$ and let q_{sk} be the following Boolean query:

$$\exists_{e \in E, i \in \{1,2\}} \mathbf{x}^i_e \bigwedge_{e \in E} \mathsf{reach}_e(\mathbf{x}^2_e, \mathbf{x}^1_e).$$

Figure 4 describes the construction of query q_{sk}. On the left hand side, it depicts the underlying skeleton sk: circles denote nodes from V with their corresponding extended C2RPQs, while the grey quadrangles together with the colored directed paths denote segment signatures. On the right hand side, we show how a C2RPQ from a reachability query corresponding to an edge in the skeleton looks like.

An important observation is the following:

▶ **Observation 18.** *For every $sk \in \mathsf{sk}(\Phi'_{k,\infty})$, it holds that: $q_{sk} \equiv \Phi_{sk}$.*

Proof. The query q_{sk} is a conjunction of reachability queries, thus a conjunction of disjunctions of C2RPQs. By distributing the disjunction over conjunction one obtains an equivalent UC2RPQ Θ in which each C2RPQ is from $\Phi'_{k,\infty}$, and, in particular, from Φ_{sk} (as it abstracts to sk). Conversely, each C2RPQ which abstracts to sk can be seen as a conjunction of realizations of reachability queries over segments and thus is from Θ. ◀

As a corollary, we obtain the following:

▶ **Corollary 19.** *For every $k > 1$, it is the case that $\Phi'_{k,\infty} \equiv \bigvee_{sk \in \mathsf{sk}(\Phi'_{k,\infty})} q_{sk}$.*

Using this insight, it is possible to evaluate queries of the form Φ_{sk} by means of a dynamic programming approach based on traversing the data structure sk. The approach can be seen as a Yannakakis-style [24] bottom-up evaluation algorithm for q_{sk} in which we have oracles to compute the upper end of a segment/reachability query given its lower end. We encode both parts of the procedure, the meta-level, where segments are combined, and the segment-specific type reachability queries, as a joint $(k+1)$-Datalog program Π_{sk}.

Datalog Encoding

We are now ready to define a Datalog rewriting $\Pi_{\Phi,k}$ for $\Phi_{\infty,k}$. The program will have several components. First, we assume that we have a program $\Pi_{\mathcal{L}}$ which contains binary IDBs L (we slightly overload the notation, to avoid introducing new names), one for each atom $L(u,v)$ occurring in $\Phi_{\infty,k}$. The purpose of the program is to materialize such atoms w.r.t. the database. The encoding is fairly standard using rules with only two variables, so we do not provide it here [2].

Next we define for every segment signature σ, a program Π_σ which captures σ-types and the way they can be interlinked to build segments. We start by introducing IDBs corresponding to types. For every σ-type τ, we will have an IDB of the form $I[\tau]$ with arity $|\mathsf{var}(\tau)|$. Furthermore, for every such type we denote with exit_τ the set of all variables from $\mathsf{var}(\tau)$ which are exit variable for some path $L \in \mathcal{L}(p)$, i.e. there exist some state s such that $\mathsf{out}_\tau(L) = (v,s)$. Then, for such a type and every set of variables $\mathbf{v}'$ such that $\mathsf{exit}(\tau) \subseteq \mathbf{v}' \subset \mathsf{var}(\tau)$ we will have an IDB predicate $I[\tau,\mathbf{v}']$ of arity $|\mathbf{v}'|$. Intuitively, the predicate stands for projections of a type; it collects instances of the (sub-)type in the database. For every set of variables $\mathbf{v}'$ such that $\mathsf{exit}(\tau) \subseteq \mathbf{v}' \subset \mathsf{var}(\tau)$, program Π_σ contains a rule of the form:

$$I[\tau](\mathsf{var}(\tau)) \to I[\tau,\mathbf{v}'](\mathbf{v}') \tag{1}$$

We next provide a rule for combining σ-types. For every two σ-types τ_1 and τ_2 such that τ_1 is compatible with τ_2, let $\mathbf{v} = \mathsf{var}(\tau_1) \cap \mathsf{var}(\tau_2)$. Then we add the following rule to Π_σ:

$$I[\tau_1,\mathbf{v}](\mathbf{v}), \mathsf{bin}(\tau_2) \to I[\tau_2](\mathsf{var}(\tau_2)) \tag{2}$$

Intuitively, to keep the width of the program bounded by $k+1$, when combining compatible types, we first project the first type on the variables relevant for matching, i.e. those variables which occur also in the second type. We remark that although $\mathsf{var}(\mathsf{bin}(\tau_2))$ might not contain all variables from $\mathsf{var}(\tau_2)$, the conditions on type compatibility ensure that $\mathbf{v} \cup \mathsf{var}(\mathsf{bin}(\tau_2)) = \mathsf{var}(\tau_2)$. In particular, $\mathbf{v}$ contains all those variables from $\mathsf{var}(\tau_2)$ for which there is no L-atom in $\mathsf{bin}(\tau_2)$, but are used to carry over information about L-paths by means of unary L-atoms in τ_2.

We next provide for every $sk \in \mathsf{sk}(\Phi_{k,\infty})$, rules which, building on the axiomatizations of previously introduced IDBs, encode the query Φ_{sk}, by simulating a bottom-up evaluation of the equivalent query q_{sk}. We denote the set of such rules as Π_{sk}.

Let $sk = (T, q, \mu)$ with $T = (V, E)$ and r being the root of T. Recall that each edge $e = (v_1, v_2)$ in E stands for a type reachability query $\mathsf{reach}_e(\tau_e^2(\mathbf{x}_e^2), \tau_e^1(\mathbf{x}_e^1))$, where τ_e^i is the type instantiation induced by $\mu(e)$ and $q(v_i)$, for $i \in \{1, 2\}$. Each node $v \in V$ is in the scope of several such type reachability queries: some correspond to edges of the form (v', v), and at most one such a query corresponds to an edge of the form (v, v'). For a node v which is not a leaf, we denote with E_v the set of edges of the form (v', v) and with $\mathsf{low}(v)$ the set of type instantiations corresponding to the second argument of such an edge: $\{\tau_e^2(\mathbf{x}_e^2) \mid e \in E_v\}$. Also, for a node v which is not the root, let e_v be the unique edge (v, v') and $\tau_v(\mathbf{x}_v)$ be the type instantiation corresponding to the first argument of e_v (i.e. an abbreviation for $\tau_{e_v}^1(\mathbf{x}_{e_v}^1)$).

There are three types of rules in the definition of Π_{sk}: those pertaining to leaves of T, one rule pertaining to the root, and rules pertaining to the other nodes. We start with the first type. For every node $v \in V$ which is a leaf of T, we define a rule which seeds the type instantiation $\tau_v(\mathbf{x}_v)$. Note that there might be variables in $\mathbf{x}_v$ which do not occur in $\mathsf{bin}(\tau_v(\mathbf{x}_v))$. To initialize such variables we assume that we have a built-in unary predicate $\mathsf{dom}()$ which binds a variable to the domain of the given database. Assuming that $v_1, \ldots, v_l$ are those dangling variables, our rule is as follows:

$$\mathsf{bin}(\tau_v(\mathbf{x}_v)), \mathsf{dom}(v_1), \ldots, \mathsf{dom}(v_l) \;\to\; I[\tau_v](\mathbf{x}_v) \tag{3}$$

We move on to nodes $v \in V$ which are neither leaves, nor the root of T. For such nodes, both $\tau_v(\mathbf{x}_v)$ and $\mathsf{low}(v)$ are defined. Assuming $\mathsf{low}(v) = \{\tau_1(\mathbf{x}_1), \ldots, \tau_l(\mathbf{x}_l)\}$ we add the folowing rule to Π_{sk}:

$$I[\tau_1](\mathbf{x}_1), \ldots, I[\tau_l](\mathbf{x}_l), \mathsf{bin}(\tau_v(\mathbf{x}_v)) \;\to\; I[\tau_v](\mathbf{x}_v) \tag{4}$$

Rule 4 intersects the IDBs corresponding to the top ends of segments coming from below to populate the IDB corresponding to the bottom end of the segment going upwards. Finally, we move on to the root node of T, r. Assuming $\mathsf{low}(r) = \{\tau_1(\mathbf{x}_1), \ldots, \tau_l(\mathbf{x}_l)\}$ we add the folowing rule to Π_{sk}:

$$I[\tau_1](\mathbf{x}_1), \ldots, I[\tau_l](\mathbf{x}_l) \;\to\; \mathsf{true} \tag{5}$$

Note that rules of type (4) and (5) as above have width at most $k+1$ as all $\mathbf{x}_i$-s are from some $\mathsf{var}(q(v))$, with $v \in V$.

Finally, let

$$\Pi_{\Phi,k} = \Pi_{\mathcal{L}} \cup \bigcup_{\sigma \in \mathsf{sig}(\Phi'_{k,\infty})} \Pi_\sigma \cup \bigcup_{sk \in \mathsf{sk}(\Phi'_{k,\infty})} \Pi_{sk}$$

The following proposition follows from Proposition 14 and the fact that there are singly exponentially many segment signatures, and for each segment signature σ, singly exponentially many σ-types:

▶ **Proposition 20.** *For Φ a Boolean UC2RPQ, and for every $k > 1$, the Datalog program $\Pi_{\Phi,k}$ is of width $k+1$, has size singly exponential in $|\Phi|$ and can be constructed in singly exponential time in $|\Phi|$.*

Further on, we show that the program $\Pi_{\Phi,k}$ encodes TW k approximations:

▶ **Proposition 21.** *For Φ a Boolean UC2RPQ, $k > 1$, and $\mathcal{G}$ be a graph database, it is the case that: $\mathcal{G} \models \Phi'_{k,\infty}$ iff $(\mathcal{G}, \Pi_{\Phi,k}) \models \mathsf{true}$.*

Proposition 20 and Proposition 21 together with complexity results on evaluation of Datalog programs of width $k+1$, yield our main result:

▶ **Theorem 22.** *Let $\mathcal{G}$ be a graph database and Φ be a Boolean UC2RPQ of semantic TW k with $k > 1$. Then, Φ be evaluated in time $O(f(|\Phi|)|\mathcal{G}|^{k+1})$, where f is a singly exponential function.*

Related Work

The skeleton queries we constructed in this section have connections both to the witness queries Φ_w constructed in [20] discussed at the beginning of this section and to the so-called *summary queries* introduced in [12].

Figueira and Morvan [12] introduce so-called *summary queries* which are exponentially more succinct representations of their TW-k approximations. To this purpose, they introduce so-called *path-l approximations* which are queries whose semantics is defined in terms of infinitary C2RPQs which admit a path-l decomposition, where a path decomposition is defined to be any tree decomposition whose underlying tree is a path. Our notions of skeleton queries and type reachability queries are based on similar intuitions, with the difference that our queries are retricted to those which admit unmeandering tree decompositions, and thus we have a clear notion of types and how they can be chained in a type reachability query.

Romero et. al [20] essentially construct witness queries of TW $2k+1$ by first constructing infinitary TW-k approximations using sub-divisions and quotients. In a subsequent step, they identify a set of nodes which correspond to our set of critical nodes, and contract paths by considering only their intersection with this set of critical nodes. By virtue of this contraction operation, they obtain tree decompositions of TW $2k+1$: intuitively, each two critical nodes which are top and bottom end of some segment are merged giving rise to a bag in the new tree decomposition. Each of our skeleton queries can be seen as the TW-k approximation of such a C2RPQ of TW $2k+1$ belonging to Φ_w, as instead of contracting the paths between two critical nodes, we consider all their realizations via paths of width k.

5 Summary and Outlook

In this paper we looked at the problem of efficient evaluation of UC2RPQs of bounded semantic treewidth. Previous approaches based on computing a witness of semantic TW k of doubly ponential size were running in time $O(f(|\Phi|)|\mathcal{G}|^{k+1})$, with f a doubly exponential function, where Φ is the size of the UC2RPQ and $|\mathcal{G}|$ is the size of the graph database. We showed that it is possible to evaluate such UC2RPQs in time $O(g(|\Phi|)|\mathcal{G}|^{k+1})$, with g a singly exponential function. We did this by encoding the evaluation problem into a Datalog program of singly exponential size and width $k+1$.

Besides the improvement in worst-case running time, the Datalog encoding also opens the way for practical approaches to evaluating UC2RPQs levaraging the plethora of available Datalog engines.

As open questions, a major one is a full characterisation of fixed parameter tractability of UC2RPQs, in particular establishing lower bounds. Another question concerns the precise complexity of computing semantic treewidth: for now it is known [12] that it is ExpSpace-hard and in 2ExpSpace.

References

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

2 Foto Afrati and Francesca Toni. Chain queries expressible by linear datalog programs. *Databases and Logic Programming*, pages 49–58, dec 1997.

3 Renzo Angles and Claudio Gutiérrez. Survey of graph database models. *ACM Computing Surveys (CSUR)*, 40(1):1–39, feb 2008. doi:10.1145/1322432.1322433.

4 Pablo Barceló, Leonid Libkin, Anthony W. Lin, and Peter T. Wood. Expressive languages for path queries over graph-structured data. *ACM Trans. Database Syst.*, 37(4), 2012. doi:10.1145/2389241.2389250.

5 Pablo Barceló, Jorge Pérez, and Juan L. Reutter. Relative expressiveness of nested regular expressions. In *Proc. of the 6th Alberto Mendelzon International Workshop on Foundations of Data Management, June 27-30, 2012*, volume 866, pages 180–195. CEUR-WS.org, 2012. URL: https://ceur-ws.org/Vol-866/paper13.pdf.

6 Pablo Barceló, Miguel Romero, and Moshe Y. Vardi. Semantic acyclicity on graph databases. *SIAM Journal on Computing*, 45(4):1339–1376, 2016. doi:10.1137/15M1034714.

7 Pablo Barceló. Querying graph databases. In *Proc. of the 32nd Symposium on Principles of Database Systems*, pages 175–188. ACM, 2013. doi:10.1145/2463664.2465216.

8 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Containment of conjunctive regular path queries with inverse. In *Proc. of the Seventh International Conference on Principles of Knowledge Representation and Reasoning*, KR'00, pages 176–185, 2000.

9 Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. *Theor. Comput. Sci.*, 239(2):211–229, 2000. doi:10.1016/S0304-3975(99)00220-0.

10 Hubie Chen, Georg Gottlob, Matthias Lanzinger, and Reinhard Pichler. Semantic width and the fixed-parameter tractability of constraint satisfaction problems. In Christian Bessiere, editor, *IJCAI*, pages 1726–1733, 2020. doi:10.24963/ijcai.2020/239.

11 Alin Deutsch, Nadime Francis, Alastair Green, Keith Hare, Bei Li, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Wim Martens, Jan Michels, Filip Murlak, Stefan Plantikow, Petra Selmer, Oskar van Rest, Hannes Voigt, Domagoj Vrgoc, Mingxi Wu, and Fred Zemke. Graph pattern matching in GQL and SQL/PGQ. In Zachary Ives, Angela Bonifati, and Amr El Abbadi, editors, *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, pages 2246–2258. ACM, 2022. doi:10.1145/3514221.3526057.

12 Diego Figueira and Rémi Morvan. Approximation and Semantic Tree-width of Conjunctive Regular Path Queries. In *26th International Conference on Database Theory*, Ioannina, Greece, mar 2023. Secondary link: https://www.morvan.xyz/papers/main-crpq-tw-icdt-v1.pdf. doi:10.4230/LIPIcs.ICDT.2023.15.

13 Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. Cypher: An evolving query language for property graphs. In *Proc. of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 1433–1445, 2018. doi:10.1145/3183713.3190657.

14 Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences*, 64(3):579–627, 2002. doi:10.1006/jcss.2001.1809.

15 Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1):1:1–1:24, 2007. doi:10.1145/1206035.1206036.

16 Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Trans. Algorithms*, 11(1), aug 2014. doi:10.1145/2636918.

17 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

18 Dániel Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. In *STOC*, pages 735–744, 2010. doi:10.1145/1806689.1806790.

19 Alberto O. Mendelzon and Peter T. Wood. Finding regular simple paths in graph databases. *SIAM Journal on Computing*, 24(6):1235–1258, 1995. doi:10.1137/S009753979122370X.

20 Miguel Romero, Pablo Barceló, and Moshe Y. Vardi. The homomorphism problem for regular graph patterns. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005106.

21 Sherif Sakr, Angela Bonifati, Hannes Voigt, Alexandru Iosup, Khaled Ammar, Renzo Angles, Walid Aref, Marcelo Arenas, Maciej Besta, Peter A. Boncz, Khuzaima Daudjee, Emanuele Della Valle, Stefania Dumbrava, Olaf Hartig, Bernhard Haslhofer, Tim Hegeman, Jan Hidders, Katja Hose, Adriana Iamnitchi, Vasiliki Kalavri, Hugo Kapp, Wim Martens, M. Tamer Özsu, Eric Peukert, Stefan Plantikow, Mohamed Ragab, Matei R. Ripeanu, Semih Salihoglu, Christian Schulz, Petra Selmer, Juan F. Sequeda, Joshua Shinavier, Gábor Szárnyas, Riccardo Tommasini, Antonino Tumeo, Alexandru Uta, Ana Lucia Varbanescu, Hsiang-Yun Wu, Nikolay Yakovets, Da Yan, and Eiko Yoneki. The future is big graphs: A community view on graph processing systems. *Commun. ACM*, 64(9):62–71, 2021. doi:10.1145/3434642.

22 Michael Sipser. *Introduction to the Theory of Computation*. Course Technology Inc., third edition, 2013.

23 Oskar van Rest, Sungpack Hong, Jinha Kim, Xuming Meng, and Hassan Chafi. PGQL: A property graph query language. In *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems*, pages 1–6. ACM, 2016. doi:10.1145/2960414.2960421.

24 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, pages 82–94, 1981.