



# Complexity of Evaluating GQL Queries



Diego Figueira  

University of Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, F-33400, Talence, France

Anthony W. Lin  

University of Kaiserslautern-Landau, Germany

MPI-SWS, Kaiserslautern, Germany

Liat Peterfreund  

Hebrew University of Jerusalem, Israel

---

## Abstract

GQL has recently emerged as the standard query language over graph databases, particularly, property graphs. Indeed, this is analogous to the role of SQL for relational databases. Unlike SQL, however, fundamental problems regarding GQL are still unsolved, most notably the complexity of query evaluation. In this paper we provide a complete solution to this problem for the core fragment of GQL and for its extension with path restrictors. In particular, we show that the data complexity of these fragments is  $P^{\text{NP}[\log]}$ -complete in general, and drops to NL-complete when restrictors are disallowed. Using techniques from embedded finite model theory, we show that this is true, even when the queries use data from infinite concrete domains such as real numbers with arithmetic. In proving these results, we establish and exploit tight connections between GQL and query languages over relational databases, especially extensions of relational calculus with transitive closure operators and fragments of second-order logic.

**2012 ACM Subject Classification** Theory of computation → Database theory

**Keywords and phrases** Graph query languages, GQL, complexity, database theory

**Digital Object Identifier** 10.4230/LIPIcs.ICDT.2026.13

**Related Version** *Full Version:* <https://arxiv.org/pdf/2407.06766>

**Funding** *Diego Figueira:* Partially supported by ANR grants INTENDED (ANR-19-CHIA-0014) and EXPAND (ANR-25-CE23-1215).

*Anthony W. Lin:* Supported by the European Research Council<sup>1</sup> under Grant No. 101089343 (LASD).

*Liat Peterfreund:* Supported by Israel Science Foundation 2355/24.

**Acknowledgements** We thank Michael Benedikt, Leonid Libkin and anonymous reviewers for valuable feedback.

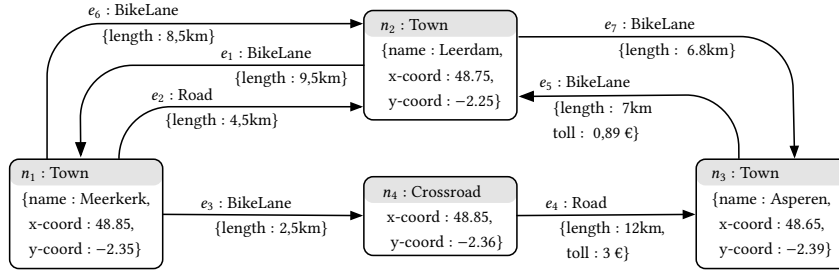
## 1 Introduction

Property-graph databases and related knowledge-graph applications gained significant popularity in the last decades, with numerous application domains, including social networks, semantic web, biological and ecological databases, and more. On the practical side, there has been rapid growth in the development of commercial graph database systems, including systems such as Neo4j, Oracle, IBM, TigerGraph, and JanusGraph. This wealth of available systems has also motivated the recent and still ongoing *standardization* effort of graph data models (in particular, *property graphs*) and query languages (in particular, GQL), which has been undertaken by several working groups consisting of research leaders from academia and industry (see, *e.g.*, [20, 11]).

---

<sup>1</sup> <https://doi.org/10.13039/100010663>





■ **Figure 1** A property graph  $G_{\text{bike}}$  of bike lanes and road connections between towns and crossroads.

GQL [10] was developed to fill in the role of SQL (i.e., as a yardstick query language) for property graphs. Unlike SQL, whose complexity landscape has been mapped for decades, the rigorous study of GQL is only now taking shape. A first informal description of the language appeared in [11], followed by a formal treatment of its pattern-matching layer in [14]. The relational-algebra layer was analyzed soon after in [15], and a study of expressiveness subsequently distilled a streamlined theoretical core [16]. Consequently, even the most basic issue – that of *data complexity* (query evaluation with only the database as input) – remains largely open. Hitherto, we know only that the complexity of enumerating query outputs is in PSpace [14], and we have an NP-hard lower bound easily derivable from [27]. In this paper, we identify the precise data complexity of the core fragment of GQL for the first time.

**GQL on property graphs in a nutshell.** A *property graph* [2] is a graph whose nodes and edges are labeled, often to represent their type or role in the graph, and are enriched with additional attributes called properties. These properties are stored in a key-value format, enabling a flexible and detailed representation of both entities and their relationships. Figure 1 depicts a property graph  $G_{\text{bike}}$  of bike lanes and roads between towns and crossroads. It has two node types indicated by their labels – Town and Crossroad, and two edge types, namely Bike-lane and Road. Towns have properties like names, and coordinates specifying their exact location, Bike-lanes and Roads have length and possibly toll.

GQL queries are essentially SQL queries over relations of nodes, edges, and properties, extracted using patterns that capture both data and graph topology (i.e., paths).

► **Example 1.1.** To find all pairs  $(x, y)$  of towns accessible by a bike lane we apply the pattern

$$\left( (x:\text{Town}) \xrightarrow{\text{:BikeLane}} (y:\text{Town}) \right)_{x,y}$$

on  $G_{\text{bike}}$ . The output is a binary relation with attributes  $x, y$ , containing, among others, the pair  $(n_2, n_1)$ . To find pairs of places accessible by one or more bike-lanes, we can incorporate *unbounded repetition* of edges:

$$R(x, y) := \left( (x:\text{Town}) \xrightarrow{\text{:BikeLane}}^{1..∞} (y:\text{Town}) \right)_{x,y} \quad (1)$$

Here, the output is the transitive closure of the previous output. We can refine matched patterns by appending a Boolean filter, e.g., returning every town  $y$  reachable *by bicycle* from Leerdam with one or more bike-lanes:

$$\left( (x:\text{Town}) \xrightarrow{\text{:BikeLane}}^{1..∞} (y) \langle x.\text{name} = \text{“Leerdam”} \rangle \right)_y$$

The condition  $\langle x.name = \text{“Leerdam”} \rangle$  filters the matches to paths starting at Leerdam. To prevent re-using the same bike-lane or revisiting the same place we prefix the pattern with an appropriate restrictor: we use `simple` to forbid repeating a node, or `trail` to forbid re-using an edge. For instance, suppose we want to organize a day trip from Meerkerk to Leerdam using at least two bike lanes without passing through the same town twice. In this case, we would use the pattern

$$\text{simple} \left( (x) \xrightarrow{\text{:BikeLane}^{2..∞}} (y) \langle x.name = \text{“Meerkerk”} \wedge y.name = \text{“Leerdam”} \rangle \right)$$

Here, the `simple` restrictor enforces the non-repetition of nodes.

In GQL, SQL-like-queries are applied to the relations produced by pattern matching over the graph. To query a round-trip, out by bicycle on bike lanes and back by car on roads, we combine the pattern in (1), namely  $R(x, y)$ , with the pattern

$$S(x', y') := \left( (x':\text{Town}) \xrightarrow{\text{:Road}^{1..∞}} (z':\text{Town}) \right)_{x', y'}$$

via  $\pi_{x,y}(\sigma_{y=x' \wedge x=z'}(R(x, y) \times S(x', z')))$  that joins the two relations on the conjunction  $y = x' \wedge x = z'$  of equalities.  $\lrcorner$

In this paper, we study the *query evaluation problem* of GQL: given a GQL query  $Q$ , a property graph  $G$ , and a candidate tuple  $\bar{t}$ , determine whether  $\bar{t}$  belongs to the result of evaluating  $Q$  over  $G$ . To study the complexity of the problem, we use *data complexity* [30], which is the standard complexity measure for query evaluation problems. That is, since we typically deal with queries of “small size” and “large” databases, we consider the query component  $Q$  as *fixed* (i.e., not part of the input) and only measure the computational complexity in terms of the data (i.e.,  $G$ ).

**Contributions.** Our analysis is centered on the core fragment of GQL as formalized in [16], which captures the essential pattern-matching constructs and their combination via relational algebra. In addition, we study an extension of this core with restrictors (e.g., `simple`, `trail`, `shortest`), which are part of practical GQL but are not included in the core fragment. Our complexity results distinguish explicitly between these two settings.

► **Theorem 1.2.** *The data complexity of GQL extended with path restrictors is  $P^{NP[\log]}$ -complete. The data complexity of GQL without restrictors improves to NL-complete.*

That is, GQL query evaluation is representative of problems that can be solved in polynomial time with logarithmically many calls to NP oracles. Note that  $P^{NP[\log]}$  lies in the second-level of polynomial hierarchy (which is in PSpace), and subsumes the entire boolean hierarchy (which is a boolean closure of NP-complete problems). The class  $P^{NP[\log]}$  includes some natural problems related to optimization (e.g., [31, 29]) and logical reasoning (e.g., [17, 18]). In the absence of path restrictors, the NP-hardness from [27] on querying simple paths or trails of even length no longer applies. Indeed, we show that the complexity in this case drops to NL-complete.

Our proof technique for the upper bound complexities is via a relational embedding of GQL to extensions of first-order logic over relational structures, namely, either with transitive closure operator, or with existential second-order definable relational views. This relational viewpoint of property graphs yields several benefits, including applicability of techniques from *embedded finite model theory* (see [24, Chapter 13] and [5]) to derive the same data complexity

of GQL in the presence of numeric data and arithmetic constraints (e.g. constraints from Tarski’s theory of real field). In particular, the framework essentially allows *unrestricted quantification* over an infinite domain such as the set of real numbers thereby allowing us to define interesting queries over spatial databases, e.g., there is a path from  $x$  to  $y$  that uses edges that lie on a straight-line (see [24, Chapter13]). We summarize this in the following corollary:

► **Corollary 1.3.** *Let  $\mathbf{S}$  be a structure satisfying the property of Restricted Quantifier Collapse (RQC), in particular Linear Real Arithmetic (LRA), Linear Integer Arithmetic (LIA), or Real-ordered Fields (ROF). Then, the data complexity of GQL with restrictors extended with unrestricted quantification over  $\mathbf{S}$  is  $P^{NP[\log]}$ -complete and, without restrictors, NL-complete.*

**Related Work.** The study of graph databases in database theory can be traced back to the work of Mendelzon and Wood [28] on Regular Path Queries (RPQ), where they popularized the data model of *edge-labeled graphs*. This has been the standard graph data model for a long time in database theory [22, 9, 28, 4, 13, 21]. Since then the area has undergone several paradigm shifts on the notion of a “graph data model”, including *data graphs* [25] and *property graphs* [2, 11, 14]. Data graphs were introduced in [26, 25] owing to the lack of a treatment of “data” (properties associated with nodes and edges, like “id”, “name” and “age” of a person) in the data model of edge-labeled graphs. Many classical query languages (like RPQ) have been extended to also incorporate such data [2, 25, 3, 12], e.g., in the query language RDPQ (Regular Data Path Queries). Data graphs still do not capture the graph model that has been adopted by practitioners, e.g., the “schemaless” nature of a graph. The property graph data model [2, 11] has been proposed to address such deficiencies.

Unlike the case of GQL, the precise data complexity of most query languages for edge-labeled graphs (e.g., RPQs and extensions) and data graphs (e.g., RDPQs) is known to be NL-complete. For example, see [28, 4, 13, 21, 22, 25]. These results were also recently extended to the case of numeric and string data domains and operations [12]. The reason for the low computational complexity in these settings is analogous to our NL upper bound for GQL without restrictors. By contrast, adding path restrictors is what enables NP-hardness (cf. [27]). Incidentally, our technique of relational embedding can be shown to provide a simpler proof of the NL upper bound in [12]. There are relatively few results on the expressivity landscape of GQL. Initial investigations have been conducted in [16, 8].

**Organization.** Section 2 recalls preliminaries on property graphs, GQL, and general logic. Section 3 presents our main complexity results for GQL. In Section 4, we extend the analysis to incorporate data values and domain-specific operators. Section 5 highlights the benefits of analyzing graph query languages from a logical perspective. We conclude in Section 6. Some details are deferred to the full version.

## 2 Preliminaries: Property Graphs, GQL and Logic

We follow the standard definition of property graph data model, and the formal syntax and semantics of GQL [16, 15, 14]. We also introduce some classical definitions from logic.

**Property Graphs.** We assume pairwise disjoint countable infinite sets of *nodes*  $\mathcal{N}$ , (directed) *edges*  $\mathcal{E}$ , *labels*  $\mathcal{L}$ , *property values*  $\mathcal{P}$  and *keys*  $\mathcal{K}$ . A *property graph*  $G$  is a tuple  $(N^G, E^G, \text{src}^G, \text{tgt}^G, \text{prop}^G, \text{lbl}^G)$  where  $N^G \subset \mathcal{N}$ ,  $E^G \subset \mathcal{E}$ , are finite sets of node and edge identifiers, respectively, and  $\text{src}^G, \text{tgt}^G: E^G \rightarrow N^G$  are functions that specify the source

and target, respectively, of an edge,  $\text{prop}^G: (E^G \cup N^G) \times \mathcal{K} \rightarrow \mathcal{P}$  is a partial function that maps a given edge or node and a key to the corresponding property value, if defined, and  $\text{lbl}^G \subset (E^G \cup N^G) \times \mathcal{L}$  specifies the labels attached to nodes and edges. We omit the superscript  $G$  when it is clear from the context.

## 2.1 GQL Syntax

We follow the core GQL definition [16]. Core GQL deliberately omits several features present in the full GQL language. In particular, patterns are designed to return relations over an explicitly specified set of variables, ensuring that all query results are in first normal form: relations contain no null values, and all attribute values are atomic (in particular, paths or lists are not first-class values).

Let  $\text{Vars}$  be an infinite set of variables. Path patterns  $\pi$  are defined via mutual recursion along with their *free variables*  $\text{fv}(\pi)$ . Pattern matching is the key component of GQL. *Path patterns* are defined recursively as follows:

$\pi$	$:=$	$(x)$	node pattern
		$\xrightarrow{x}$	edge pattern
		$\xleftarrow{x}$	edge pattern
		$\pi_1 \pi_2$	concatenation
		$\pi_1 + \pi_2$ if $\text{fv}(\pi_1) = \text{fv}(\pi_2)$	disjunction
		$\pi^{n..m}$	repetition
		$\pi\langle\theta\rangle$	filtering

where

- $x \in \text{Vars}$  and  $0 \leq n \leq m \leq \infty$ ;
- variables  $x$  in node patterns  $(x)$ , and edge patterns  $\xrightarrow{x}$  and  $\xleftarrow{x}$  are optional,
- $\pi\langle\theta\rangle$  is a conditional pattern, where conditions  $\theta$  are given by  $\theta := x.k = x'.k' \mid \ell(x)^2 \mid \theta \vee \theta \mid \theta \wedge \theta \mid \neg\theta$  where  $x, x' \in \text{Vars}$ ,  $\ell \in \mathcal{L}$ , and  $k, k' \in \mathcal{K}$ ;

The free variables of path patterns are defined as follows:

$$\begin{aligned}
 \text{fv}((x)) &= \text{fv}(\xrightarrow{x}) = \text{fv}(\xleftarrow{x}) &:= \{x\} \\
 \text{fv}(\pi_1 + \pi_2) &:= \text{fv}(\pi_1) \\
 \text{fv}(\pi_1 \pi_2) &:= \text{fv}(\pi_1) \cup \text{fv}(\pi_2) \\
 \text{fv}(\pi^{n..m}) &:= \emptyset^3 \\
 \text{fv}(\pi\langle\theta\rangle) &:= \text{fv}(\pi)
 \end{aligned}$$

We then define *output patterns* as  $\bar{\rho}\pi\Omega$  where

- $\bar{\rho} := [\text{shortest}][\text{simple}|\text{trail}]$  is sequence of *restrictors* such that parts within squared brackets are optional, and
- $\Omega$  is a (possibly empty) finite sequence of elements of the form  $x$  and  $x.k$  where  $x \in \text{Vars}$  and  $k \in \mathcal{K}$ .

Intuitively,  $\bar{\rho}$  imposes constraints on the matched paths, and  $\Omega$  specifies which variables and properties are returned (*i.e.*, projected) in the output relation.

<sup>2</sup> This condition checks whether  $x$  is labeled with  $\ell$ . For convenience, we often use the shorthand  $(x:\ell)$ , which incorporates the label directly into the node pattern.

Notice that while [16] omitted restrictors from the language, we include them and give them a precise semantics that is consistent with the standard [15], extending the core fragment and aligning the formal model more closely with the practical language.

Finally, we associate every output pattern  $\bar{\rho}\pi_\Omega$  with a relation symbol  $R_{\bar{\rho}\pi_\Omega}$ . *GQL queries*  $\mathcal{Q}$  are then defined as the closure of these symbols under the standard operators of relational algebra.<sup>4</sup> Formally,

$$\mathcal{Q} := R_{\bar{\rho}\pi_\Omega} \mid \mathcal{Q} \cup \mathcal{Q} \mid \mathcal{Q} \times \mathcal{Q} \mid \mathcal{Q} \setminus \mathcal{Q} \mid \pi_{\bar{x}}\mathcal{Q} \mid \sigma_\theta\mathcal{Q}$$

for every output pattern  $\bar{\rho}\pi_\Omega$  where  $\bar{x}$  is a vector of  $\text{Vars}$  and  $\theta := x = y \mid \theta \vee \theta \mid \theta \wedge \theta \mid \neg\theta$  with  $x, y \in \text{Vars}$ . Note that the operators used here are the standard operators of relational algebra, namely union  $\cup$ , Cartesian product  $\times$ , difference  $\setminus$ , projection  $\pi$ , and selection  $\sigma$ , with the usual semantics over relations (see, e.g., [1]). The schema  $\text{sch}(\mathcal{Q})$  of a GQL query  $\mathcal{Q}$  is defined inductively. For the base case, we set  $\text{sch}(R_{\bar{\rho}\pi_\Omega}) := \Omega$ . For composite queries the schema is defined in the standard way (see, e.g., [1]).

## 2.2 GQL Semantics

The semantics of output patterns is defined in terms of the semantics of path patterns. We therefore begin by defining the latter.

A *path*  $p$  in a property graph  $G$  is an alternating sequence of nodes and edges that start and end with a node, that is,  $n_0 e_0 n_1 e_1 \cdots e_{k-1} n_k$  where  $n_0, \dots, n_k \in N$ , and  $e_0, \dots, e_{k-1} \in E$ . Each edge  $e_i$  must connect the adjacent nodes, meaning either (1)  $\text{src}(e_i) = n_i$  and  $\text{tgt}(e_i) = n_{i+1}$  or (2)  $\text{tgt}(e_i) = n_i$  and  $\text{src}(e_i) = n_{i+1}$ .

The semantics  $\llbracket \pi \rrbracket^G$  of a path pattern  $\pi$  over a property graph  $G$  is defined as a set of pairs  $(p, \mu)$ , where:

- $p$  is a path in  $G$ , and
- $\mu : \text{Vars} \rightarrow N \cup E$  is a partial assignment that is defined on  $\text{dom}(\mu) := \text{fv}(\pi)$ .

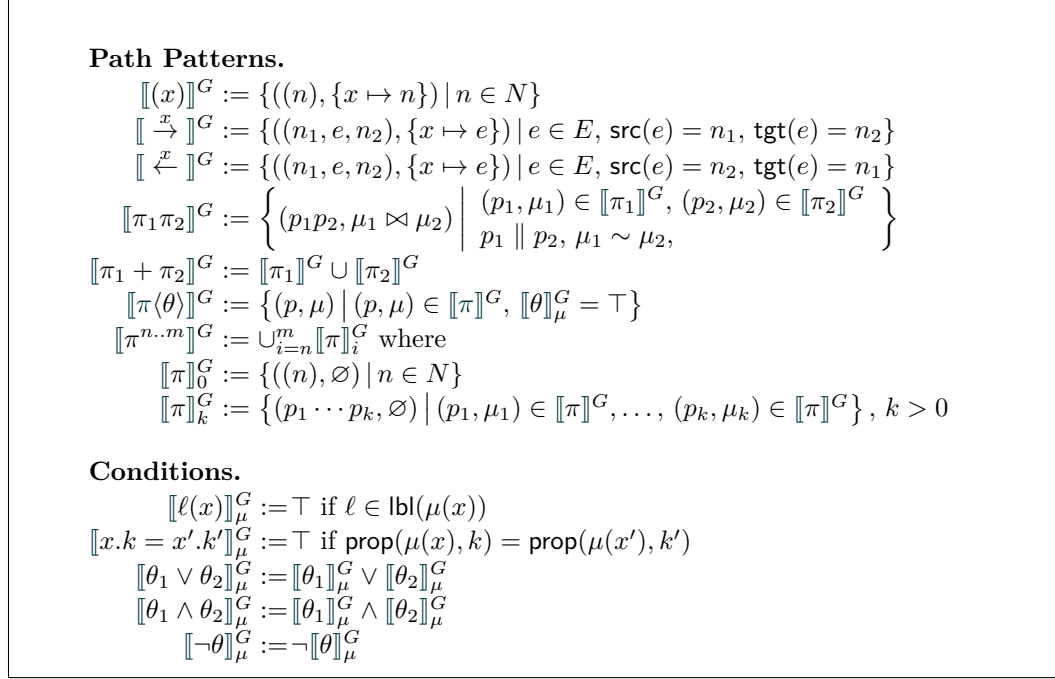
We define  $\text{src}(p) := n_0$ ,  $\text{tgt}(p) := n_k$ , and  $\text{len}(p) := k$ . For two paths  $p_1, p_2$ , we write  $p_1 \parallel p_2$  to indicate that  $\text{tgt}(p_1) = \text{src}(p_2)$ . In this case,  $p_1 p_2$  is the concatenation of the paths. For partial mappings  $\mu_1, \mu_2 : \text{Vars} \rightarrow N \cup E$ , we use  $\mu_1 \sim \mu_2$  to denote that for every  $x \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)$ , it holds that  $\mu_1(x) = \mu_2(x)$  and define  $(\mu_1 \bowtie \mu_2)(x) := \mu_1(x)$  if  $x \in \text{dom}(\mu_1)$ , and  $\mu_2(x)$  otherwise. We represent mappings  $\mu$  as sets of elements  $x \mapsto o$ , where  $\mu(x) = o$ . (If  $\text{dom}(\mu) = \emptyset$  then we represent  $\mu$  by  $\emptyset$ .)

The semantics  $\llbracket \pi \rrbracket^G$  of path patterns on a graph  $G$  is defined in Figure 2. We emphasize that the semantics of repetition deliberately ignores variable inside the repeated subpattern, following [16]. This restriction ensures that all query results are in first normal form, avoiding list-valued outputs that are permitted in the full GQL language but excluded from the core fragment.

For the semantics of output patterns  $\rho\pi_\Omega$ , we first define the intermediate semantics of path patterns preceded by restrictors, iterating through possible cases. For  $\rho \in \{\text{simple}, \text{trail}\}$ , we define  $\llbracket \rho\pi \rrbracket^G := \{(p, \mu) \mid (p, \mu) \in \llbracket \pi \rrbracket^G, \rho(p) = \top\}$  where  $\rho(p) = \top$  if and only if  $p$  is a simple path or a trail, respectively. Next, for  $\tilde{\pi}$  of the form  $[\text{simple}|\text{trail}]\pi$  (with the restrictors being optional) we define

$$\llbracket \text{shortest } \tilde{\pi} \rrbracket^G := \{(p, \mu) \mid (p, \mu) \in \llbracket \tilde{\pi} \rrbracket^G, \text{len}(p) = m_p\}$$

<sup>4</sup> Notice that while [16] uses a linear form of relational algebra, we use the more convenient standard relational algebra as the equivalence is already established in that work.



■ **Figure 2** Semantics of GQL's *Path Patterns* and *Conditions*.

where  $m_p$  is the minimal length among paths that match  $\tilde{\pi}$  with the same endpoints as  $p$ , or, more formally,

$$m_p = \min \{ \text{len}(p') \mid (p', \mu) \in \llbracket \tilde{\pi} \rrbracket^G, \text{src}(p') = \text{src}(p), \text{tgt}(p') = \text{tgt}(p) \}.$$

Let  $\Omega$  be a (possibly empty) finite sequence of elements of the form  $x$  and  $x.k$  where  $x \in \text{Vars}$  and  $k \in \mathcal{K}$ . We say that a mapping  $\mu$  is *compatible* with  $\Omega$  if for each  $x \in \Omega$ ,  $\mu$  is defined on  $x$ , and for each  $x.k \in \Omega$ ,  $\mu$  is defined on  $x$  and  $\text{prop}(\mu(x), k)$  is defined. In this case, we define  $\mu_\Omega: \Omega \rightarrow N \cup E \cup \mathcal{P}$  as the projection of  $\mu$  on  $\Omega$ :

$$\mu_\Omega(\omega) := \begin{cases} \mu(x) & \text{if } \omega = x \in \text{Vars} \\ \text{prop}(\mu(x), k) & \text{if } \omega = x.k \end{cases}$$

Finally, we define  $\llbracket \bar{\rho} \pi_\Omega \rrbracket^G := \{ \mu_\Omega \mid (p, \mu) \in \llbracket \bar{\rho} \pi \rrbracket^G \}$ . Note that the semantics of an output pattern can be interpreted as a relation as it is a set of partial mappings that share the same domain. Once this is fixed, the semantics of an entire GQL query is obtained exactly as in standard relational algebra (see, e.g., [1]).

### 2.3 First and Second-Order Logic

We assume basic familiarity with mathematical logic (e.g., [24]). In the sequel, we deal only with finite relational structures.

**Relational structures.** We fix disjoint countably infinite sets of *relation names*  $\text{Rel}$ , *constant names*  $\text{Const}$ , and *variables*  $\text{Vars}$ . Every relation name  $R \in \text{Rel}$  is associated with a positive integer, namely its *arity*  $\text{ar}(R)$ . We write  $R^{\text{ar}(R)}$  to make the arity explicit. A *vocabulary* is a finite set  $\sigma \subseteq \text{Rel}$ . A *relation* is a finite set of *tuples* in  $\text{Const}^{\text{ar}(R)}$ . A  $\sigma$ -*structure*  $\mathbf{S}$

## 13:8 Complexity of Evaluating GQL Queries

maps every  $R^{(k)} \in \sigma$  to a finite relation  $[R]^{\mathbf{S}} \subseteq \text{Const}^k$ . Its *active domain* is  $\text{adom}(\mathbf{S}) = \{c \in \text{Const} : c \text{ occurs in } [R]^{\mathbf{S}} \text{ for some } R \in \sigma\}$ . Property graphs can naturally be viewed as  $\sigma$ -structures over the vocabulary consisting of the relation names  $N, E, \text{src}, \text{tgt}, \text{prop}, \text{lbl}$  and interpreted accordingly.

**First-order logic.** FO is defined in the usual way over  $\sigma$ -structures. That is, terms  $t$  are defined by  $t := x \mid c$  where  $x \in \text{Vars}$ ,  $c \in \text{Const}$  and formulas  $\varphi := t = t' \mid R(t_1, \dots, t_k) \mid \neg \varphi \mid \varphi \wedge \varphi \mid \exists x \varphi$ . We use the standard  $\forall, \rightarrow$  as syntactic sugar.

For a formula  $\varphi$ , we write  $\varphi(\bar{x})$  to explicitly indicate that  $\bar{x}$  are its free variables. An  *$\bar{x}$ -valuation* is a function  $\nu$  mapping variables of  $\bar{x}$  to elements of the active domain of a relational structure  $\mathbf{S}$ . We write  $\llbracket \varphi(\bar{x}) \rrbracket_{\nu}^{\mathbf{S}}$  for the truth value  $\top$  or  $\perp$  of  $\varphi$  under the valuation  $\nu : \bar{x} \rightarrow \text{adom}(\mathbf{S})$ . We denote by  $\llbracket \varphi(\bar{x}) \rrbracket^{\mathbf{S}}$  the set of valuations  $\nu$  that satisfy  $\varphi$  in  $\mathbf{S}$ , that is, valuations for which  $\llbracket \varphi(\bar{x}) \rrbracket_{\nu}^{\mathbf{S}} = \top$ .

**Transitive closure.** First-order logic extended with the transitive-closure operator is denoted FO[TC]. Transitive-closure formulas are of the form  $\text{TC}_{\bar{u}, \bar{v}}(\varphi(\bar{u}, \bar{v}, \bar{p}))(\bar{x}, \bar{y})$ , where  $\bar{p}\bar{x}\bar{y}$  are the free variables. Such a formula is true in a structure  $\mathbf{S}$  under a valuation  $\nu$  iff there exists a finite sequence of tuples  $\bar{d}_0, \bar{d}_1, \dots, \bar{d}_n \in \text{adom}(\mathbf{S})^{|\bar{u}|}$  with  $\bar{d}_0 = \nu(\bar{x})$  and  $\bar{d}_n = \nu(\bar{y})$ , such that for every  $i < n$ ,  $\llbracket \varphi \rrbracket_{\mu_i}^{\mathbf{S}} = \top$ , where  $\mu_i$  is the valuation mapping  $\bar{u}, \bar{v}$  and  $\bar{p}$  to  $\bar{d}_i, \bar{d}_{i+1}$  and  $\nu(\bar{p})$  respectively.

**Second-order logic.** SO (second-order logic) extends FO by allowing quantification not only over individual elements of the active domain, but also over relation symbols of arbitrary arity. Concretely, SO formulas may include subformulas of the form

$$\forall R (\psi(R)) \quad \text{or} \quad \exists R (\psi(R)),$$

where  $R$  is a relation variable and  $\psi(R)$  is an FO formula in which  $R$  may occur as  $R(x_1, \dots, x_k)$ .

The fragment ESO consists of those SO formulas in which all second-order quantifiers are existentially quantified. Note that negation in ESO is permitted only within the first-order component and cannot apply to second-order quantifiers.

### 3 Complexity Bounds for GQL Queries

We now use standard logics to capture core GQL with and without restrictors, turning syntactic translations into complexity bounds.

#### 3.1 Restrictor-Free GQL Queries and FO[TC]

We call a GQL query *restrictor-free* if no output pattern occurring in it is prefixed by a restrictor.

► **Lemma 3.1.** *For every restrictor-free GQL query  $\mathcal{Q}$  there is an FO[TC] formula  $\varphi_{\mathcal{Q}}(\bar{x})$  where  $\bar{x} = \text{sch}(\mathcal{Q})$  such that  $\llbracket \mathcal{Q} \rrbracket^G = \llbracket \varphi_{\mathcal{Q}}(\bar{x}) \rrbracket^G$  for every graph  $G$ .*

**Proof Sketch.** The proof translates a *restrictor-free* GQL query into an FO[TC] formula whose answer relation coincides with the query result. The construction involves three main steps:

**(1) Path patterns.** For every path pattern  $\pi$  we build an FO[TC] formula  $\varphi_\pi(s, t, \bar{x})$  whose free variables are the start node  $s$  and end node  $t$  of the matched path, and the pattern variables  $\bar{x} = \text{fv}(\pi)$ . The translation is recursive:

- *Atomic patterns* use predicates such as  $N(x)$  for node patterns, and combination of  $E(e)$ ,  $\text{src}(e, s)$ ,  $\text{tgt}(e, t)$  for edge patterns, depending on their direction.
- *Union and concatenation.*  $\pi_1 + \pi_2$  translates to disjunction, and for  $\pi_1\pi_2$  we use an existentially quantified concatenation node.
- *Filters.* A condition  $\pi(\theta)$  becomes  $\varphi_\pi \wedge \varphi_\theta$ , where each atomic condition ( $x:l$ ,  $x.a = y.b$ , etc.) is rendered, in turn, into FO.
- *Kleene bounds.* Bounded repetition  $\pi^{n..m}$  with  $m \neq \infty$  unrolls into a finite disjunction of FO formula; unbounded repetition  $\pi^{0..\infty}$  is captured with a transitive closure operator applied to  $\varphi_\pi$ .

**(2) Output patterns.** An output pattern  $\pi_\Omega$  returns a tuple of node/edge IDs and property values. To construct the corresponding output pattern formula, we start from  $\varphi_\pi$  and, for each element in the return list  $\Omega$ , add a first-order constraint that links the output variable to the witness variables introduced by  $\varphi_\pi$ .

**(3) Full queries.** A restrictor-free GQL query is just a relational-algebra expression over the auxiliary relations defined in step (2). Because every RA operator is FO-definable, an easy induction pushes all definitions inside and yields a single FO[TC] sentence  $\varphi_\Omega(\bar{x})$  whose free variables  $\bar{x} = \text{sch}(\mathcal{Q})$ . ◀

► **Example 3.2.** Query (1) from Section 1 can be expressed as  $\varphi_1(x, y) := [\text{TC}(\varphi(u, v))](x, y)$  where  $\varphi(u, v) := \exists e \left( E(e) \wedge \text{src}(e, u) \wedge \text{tgt}(e, v) \wedge \text{lbl}(e, \text{“BikeLane”}) \right)$ . Here, the relation names  $E$ ,  $\text{src}$ ,  $\text{tgt}$ , and  $\text{lbl}$  constitute the vocabulary of the graph. ▽

Combining Lemma 3.1 with the known fact that the data complexity of the evaluation problem for FO[TC] is NL-complete [24] gives an NL upper bound. A matching lower bound via a reduction from graph reachability yields the following tight bound.

► **Theorem 3.3.** *The data complexity of the evaluation problem for restrictor-free GQL queries is NL-complete.*

Having settled the complexity of restrictor-free queries, we now turn to queries with restrictors.

## 3.2 GQL Queries and FO[ESO]

We define FO[ESO] as first-order logic extended with ESO-based first-order views. Specifically, we call an ESO formula whose free variables are first-order variables an *ESO FO-view*, and define FO[ESO] as the set of first-order formulas that may include such views as sub-formulas.

► **Lemma 3.4.** *For every GQL query  $\mathcal{Q}$  with restrictors there is an FO[ESO] formula  $\varphi_\mathcal{Q}(\bar{x})$  with  $\bar{x} = \text{sch}(\mathcal{Q})$  such that  $[\mathcal{Q}]^G = \emptyset$  if and only if  $[\varphi_\mathcal{Q}(\bar{x})]^G = \emptyset$  for every graph  $G$ .*

**Proof Sketch.** The proof translates a GQL query with restrictors into a FO[ESO] formula whose emptiness coincides with the query’s emptiness.

## 13:10 Complexity of Evaluating GQL Queries

**(1) Encoding a restricted path.** We encode a path  $(n_1, e_1, \dots, n_k, e_k, n_{k+1})$  with a ternary relation  $R(x, e, y)$  containing the  $k$  triples  $(n_j, e_j, n_{j+1})$ . For the **simple** restrictor we use the FO test  $\psi^{\text{simple}}(R)$ . To define it we use the following auxiliary shortcuts

$$\begin{aligned} \text{edge}(e) &:= \exists x, y R(x, e, y), \\ \text{node}(x) &:= \exists e, y (R(x, e, y) \vee R(y, e, x)), \\ \text{source}(x) &:= \text{node}(x) \wedge \neg \exists e, y R(y, e, x), \\ \text{target}(x) &:= \text{node}(x) \wedge \neg \exists e, y R(x, e, y). \end{aligned}$$

The conjunction of the following requirements gives us  $\psi^{\text{simple}}(R)$ :

(i) There is exactly one source

$$\exists s (\text{source}(s) \wedge \forall s' (\text{source}(s') \rightarrow s' = s)).$$

(ii) There is exactly one target

$$\exists t (\text{target}(t) \wedge \forall t' (\text{target}(t') \rightarrow t' = t)).$$

(iii) Every internal node (*i.e.*, neither the source nor the target) has precisely one incoming edge and one outgoing edge  $\forall n ((\text{node}(n) \wedge \neg \text{source}(n) \wedge \neg \text{target}(n)) \rightarrow (\text{one\_in}(n) \wedge \text{one\_out}(n)))$  where  $\text{one\_in}(n) := \exists y, e (R(y, e, n) \wedge \forall y', e' (R(y', e', n) \rightarrow (e' = e \wedge y' = y)))$  and  $\text{one\_out}(n)$  is defined symmetrically.

For the **trail** restrictor, we drop condition (iii) and instead require that each edge has at most one successor edge and one predecessor edge. To express this, we define a successor formula over pairs of edges.

$$\text{succ}(e, e') := \exists x, y, z (R(x, e, y) \wedge R(y, e', z))$$

and with that at hand, we define

$$\begin{aligned} \forall e, e_1, e_2 ((\text{succ}(e, e_1) \wedge \text{succ}(e, e_2)) \rightarrow e_1 = e_2) \wedge \\ \forall e, e_1, e_2 ((\text{succ}(e_1, e) \wedge \text{succ}(e_2, e)) \rightarrow e_1 = e_2). \end{aligned}$$

For the **shortest** restrictor, no special handling is needed, as the emptiness of a pattern is preserved when this restrictor is applied.

**(2) Changing the atomic patterns.** For every path pattern  $\pi$  we build, by structural induction, an ESO formula  $\varphi_\pi^R(\bar{x})$  with  $\bar{x} = \text{fv}(\pi)$  that is true iff  $R$  encodes a restricted path matching  $\pi$ . Only the *base cases* differ from the FO[TC] construction of Lemma 3.1, for example:

$$\begin{aligned} \varphi_{(x)}(s, t, x) &:= \text{node}(x) \wedge s = x \wedge t = x, \\ \varphi_{\bar{x}}(s, t, x) &:= \text{edge}(x) \wedge \text{src}(x, s) \wedge \text{tgt}(x, t), \end{aligned}$$

with  $\text{node}(x)$  and  $\text{edge}(x)$  as reconstructed above from  $R$ . A restricted path pattern is therefore represented by  $\exists R (\psi^p(R) \wedge \varphi_\pi^R(\bar{x}))$ , an ESO sentence with free variables  $\bar{x}$ .

**(3) From output patterns to queries.** Every output pattern  $\bar{\rho} \pi_\Omega$  introduces a relation symbol  $R_{\bar{\rho} \pi_\Omega}$  and the ESO definition obtained in step 2. A complete GQL query is an RA expression over these symbols. Because each RA operator (projection, join, union, difference) is FO-definable, substituting the ESO definitions and pushing the FO connectives outward yields an FO[ESO] formula  $\varphi_\Omega(\bar{x})$  satisfying  $\llbracket \Omega \rrbracket^G = \emptyset \iff \llbracket \varphi_\Omega(\bar{x}) \rrbracket^G = \emptyset$  for every graph  $G$ . ◀

► **Example 3.5.** Continuing Example 3.2, if the pattern is preceded by the restrictor `simple` then the translation is  $\varphi_2(x, y) := \exists R : \psi^{\text{simple}}(R) \wedge \tilde{\varphi}_1(x, y)$  where  $\tilde{\varphi}_1(x, y)$  is obtained from  $\varphi_1(x, y)$  by replacing  $E(e)$  with  $\text{edge}(e)$  derived from  $R$  (as detailed in the proof sketch of Lemma 3.4).  $\lrcorner$

The complexity class  $P^{\text{NP}[\log]}$  consists of decision problems solvable in deterministic polynomial time using logarithmically many adaptive queries to an NP oracle [31, 29]. Equivalently, it can be defined as the class of problems solvable by first generating polynomially many NP oracle queries, performing all of them in parallel (non-adaptively), and then computing the final answer from the oracle's yes/no responses.

► **Corollary 3.6.** *The data complexity of the evaluation problem for GQL queries is in  $P^{\text{NP}[\log]}$ .*

**Proof Sketch.** Let  $\mathcal{Q}$  be a GQL query and let  $\varphi_{\mathcal{Q}}$  be the FO[ESO] formula guaranteed in Lemma 3.4. Let us denote by  $\varphi_1, \dots, \varphi_k$  the first-order ESO views. Assume that each  $\varphi_i$  has arity  $k_i$ , and let us denote  $n := |\text{adom}(G)|$ .

1. For every  $i$  and every tuple  $\bar{a} \in \text{adom}(G)^{k_i}$  ask the NP oracles whether  $\llbracket \varphi_i \rrbracket_{\bar{x} \rightarrow \bar{a}}^G = \top$ . The total number of queries is  $\sum_i n^{k_i} = n^{O(1)}$ .
2. Issue this batch of polynomially many queries *in parallel* to obtain the complete satisfying assignments for all of the  $\varphi_i$ s.
3. Evaluate the remaining first-order part of  $\varphi_{\mathcal{Q}}$  deterministically in  $\text{poly}(n)$  time.

Notice that the algorithm makes polynomially many non-adaptive NP calls.  $\blacktriangleleft$

**Is the bound tight?** To investigate this, we introduce the *Odd-Index problem*. In its original form [31, Theorem 5.2], the problem asks: given a list of Boolean formulas  $\varphi_1, \dots, \varphi_n$  in 3-CNF, is there an odd index  $i \in [n]$  such that:

- all formulas  $\varphi_1, \dots, \varphi_i$  are satisfiable, and
- all formulas  $\varphi_{i+1}, \dots, \varphi_n$  are unsatisfiable?

This problem can be generalized by replacing the 3-CNF satisfiability with any NP-hard decision problem. In our context we replace it with *simple-path (trail) even-length reachability* in graphs. Formally, we define the following problem:

- Given a list  $(G_1, s_1, t_1), \dots, (G_n, s_n, t_n)$ , where each  $G_i$  is a graph and  $s_i, t_i$  are designated nodes, determine whether there exists an *odd index*  $i \in [n]$  such that:
- for all  $j \leq i$ , there is a simple path (trail) of even length from  $s_j$  to  $t_j$  in  $G_j$ , and
  - for all  $j > i$ , there is no such simple path (trail) of even length.

We reduce this problem to the evaluation of a GQL query, thereby establishing the following lower bound.

► **Lemma 3.7.** *The evaluation problem for GQL queries with restrictors is  $P^{\text{NP}[\log]}$ -hard.*

Combining Corollary 3.6 with Lemma 3.7 yields the following tight bound:

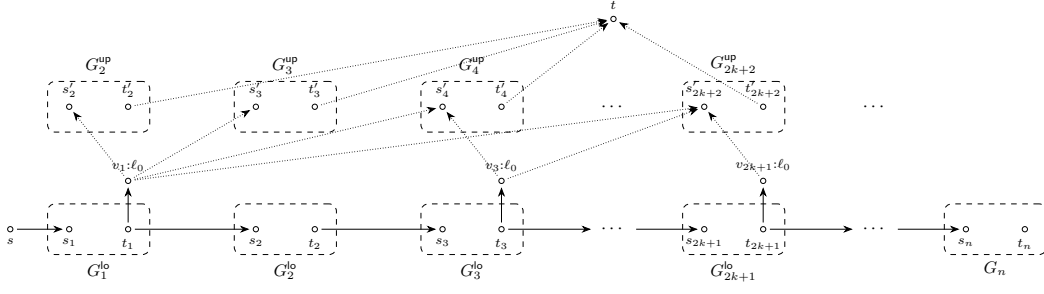
► **Theorem 3.8.** *The evaluation problem for GQL queries with restrictors is  $P^{\text{NP}[\log]}$ -complete.*

This completes the  $P^{\text{NP}[\log]}$ -completeness characterization for GQL queries with restrictors.

### 3.3 Proof of Lemma 3.7: $P^{\text{NP}[\log]}$ Lower Bound

We give a log-space reduction from the  $P^{\text{NP}[\log]}$ -complete Odd-Index problem to the evaluation of a GQL query that uses the `simple` restrictor. (A similar construction works with `trail`.)

## 13:12 Complexity of Evaluating GQL Queries



■ **Figure 3** The graph  $G^*$  used in the reduction. **Lower layer:** Copies  $G_i^{lo}$  of each input graph  $G_i$  are linked by edges  $s \rightarrow s_1$ ,  $t_i \rightarrow s_{i+1}$ , and odd-index nodes  $v_j$  (label  $\ell_0$ ) are appended via  $t_j \rightarrow v_j$ . **Upper layer:** Fresh copies  $G_j^{up}$  for  $j \geq 2$  have sources  $s'_j$  connected from every  $v_i$  with  $i < j$ , and all upper targets  $t'_j$  feed into the global sink  $t$ . Source nodes  $s, s_i, s'_j$  carry label  $s$ ; targets  $t, t_i, t'_j$  carry  $t$ ; and each  $v_j$  (odd  $j$ ) carries  $\ell_0$ .

**Query.** We define the query  $Q := Q_L \bowtie Q_R$  where

$$Q_L := \text{simple} \left( (:s) \rightarrow \pi \rightarrow (x:\ell_0) \right)_x \quad \text{where} \quad \pi := (:s)(\rightarrow \rightarrow)^*(:t) \left( \rightarrow (:s)(\rightarrow \rightarrow)^*(:t) \right)^*$$

$$Q_R := (x:\ell_0)_x \setminus \text{simple} \left( (x:\ell_0) \rightarrow (:s)(\rightarrow \rightarrow)^*(:t) \rightarrow (:t) \right)_x$$

**Graph construction.** Assume the input list  $(G_1, s_1, t_1), \dots, (G_n, s_n, t_n)$  is pairwise disjoint, and each  $s_i, t_i$  carries labels  $s, t$ , respectively. We build  $G^*$ , depicted in Figure 3, as follows:

1. *Lower layer.*
  - a. For every  $1 \leq i \leq n$  add a fresh copy  $G_i^{lo}$ .
  - b. Add a fresh node  $s$  labeled by  $s$ , and an edge  $s \rightarrow s_1$ .
  - c. For each  $i < n$  add an edge  $t_i \rightarrow s_{i+1}$ .
  - d. For each *odd*  $j$  add a node  $v_j$  labeled  $\ell_0$ , and an edge  $t_j \rightarrow v_j$ .
2. *Upper layer.*
  - a. For every  $2 \leq j \leq n$  add a fresh copy  $G_j^{up}$  with nodes  $s'_j, t'_j$  (replacing  $s_j, t_j$ ).
  - b. Add an edge  $v_j \rightarrow s'_i$  for every *odd*  $j$  and  $i \geq j+1$ .
  - c. Add a fresh node  $t$  labeled by  $t$ , and edges  $t'_i \rightarrow t$  for all  $i \geq 2$ .

Label summary: all sources  $s, s_1, \dots, s_n, s'_2, \dots, s'_n$  carry  $s$ ; all targets  $t, t_1, \dots, t_n, t'_2, \dots, t'_n$  carry  $t$ ; each  $v_j$  (*odd*  $j$ ) carries  $\ell_0$ .

Notice that  $G^*$  construction is first-order definable and uses only logarithmic space.

**Correctness.** To prove correctness, we investigate both parts  $Q_L$  and  $Q_R$  of the query in the following lemmas.

► **Lemma 3.9.**  $Q_L$  evaluated on  $G^*$  returns exactly those nodes  $v_j$  such that for every  $i \leq j$ , the graph  $G_i$  contains a simple path of even-length from  $s_i$  to  $t_i$ .

**Proof.** Assume first that  $Q_L$  evaluated on  $G^*$  returns a node  $n$ . Due to  $Q_L$ 's definition, there must exist in  $G^*$  a *simple* path of the form

$$s \rightarrow s_1 \xrightarrow{p_1} t_1 \rightarrow s_2 \xrightarrow{p_2} t_2 \rightarrow \dots \rightarrow s_i \xrightarrow{p_i} t_i \rightarrow n,$$

where (1)  $1 \leq i \leq n$ ; (2)  $s$  is the unique fresh node labeled  $s$  created in Step 1(b), (3)  $s_i$  is the source of  $G_i^{lo}$  in the lower layer; (4) each  $p_i$  consists of an even number of edges; and (5)  $n$  is labeled  $\ell_0$ .

Notice that the only  $\ell_0$ -labelled nodes are  $v_1, v_3, v_5, \dots$ . Let us denote  $n = v_j$  for the corresponding odd  $j$ . By construction  $v_j$  is attached to  $t_j$ , hence  $t' = t_j$ .

Due to the structure of  $G^*$  and to the labels specified in  $Q_L$ , the matched path is forced to traverse, in order,

$$G_1^{\text{lo}}, G_2^{\text{lo}}, \dots, G_j^{\text{lo}},$$

employing the connecting edges  $t_i \rightarrow s_{i+1}$  for  $i < j$ . As the overall walk is simple, in each lower copy  $G_i^{\text{lo}}$  the sub-path from  $s_i$  to  $t_i$  is itself simple. Consequently, for all  $i \leq j$ , it holds that  $G_i$  contains a simple path of even length  $s_i \rightsquigarrow t_i$  from  $s_i$  to  $t_i$ .

Conversely, suppose there is an odd index  $j$  such that every  $G_i$  ( $i \leq j$ ) has a simple even-length path  $s_i \rightsquigarrow t_i$ . Concatenate those paths with the connecting edges

$$s \rightarrow s_1, \quad t_i \rightarrow s_{i+1} \ (i < j), \quad t_j \rightarrow v_j;$$

The result is a simple path that matches  $Q_L$ , hence  $v_j$  is returned.  $\blacktriangleleft$

**► Lemma 3.10.**  $Q_R$  evaluated on  $G^*$  returns exactly those nodes  $v_j$  such that, for every  $i \geq j + 1$ , the graph  $G_i$  contains no simple path of even-length from  $s_i$  to  $t_i$ .

**Proof.** Recall  $Q_R := (x:\ell_0)_x \setminus \mathbf{simple} \left( (x:\ell_0) \rightarrow (:s)(\rightarrow\rightarrow)^*(:t) \rightarrow (:t) \right)_x$ . We refer to the patterns before and after  $\setminus$ , respectively, as *left* and *right* parts.

**Left part.** The projection  $(x:\ell_0)_x$  simply returns every  $\ell_0$ -labelled vertex, i.e. the odd-indexed nodes  $v_1, v_3, v_5, \dots$

**Right part.** A node  $v_j$  is matched to the right part if there is a path

$$v_j \rightarrow s'_i \rightsquigarrow t'_i \rightarrow t \quad \text{for some } i \geq j + 1 \quad (*)$$

exists in  $G^*$ . where  $s'_i \rightsquigarrow t'_i$  is of even length, The edge  $t'_i \rightarrow t$  is the *only* occurrence of two consecutive  $t$ -labelled nodes, so any matched path must enter the upper copy  $G_i^{\text{up}}$  at its source  $s'_i$  and reach  $t'_i$  without repeating nodes. Because each  $G_i^{\text{up}}$  is disjoint from the rest of the graph, such a sub-path exists *iff* the original graph  $G_i$  contains a simple path of even length  $s_i \rightsquigarrow t_i$ .

**The difference of the left part and the right part** Due to the above,  $v_j$  is outputted by  $Q_R$  whenever there is no even-length simple path in  $G_i^{\text{up}}$  from  $s_i$  to  $t_i$  for every  $i \geq j + 1$ .  $\blacktriangleleft$

**Combining Claims 1 and 2.** Claim 1 shows that  $Q_L$  returns precisely those markers  $v_j$  for which every graph  $G_i$  with  $i \leq j$  admits a simple even-length  $s_i \rightsquigarrow t_i$  path. Claim 2 shows that  $Q_R$  returns exactly those same markers  $v_j$  for which no graph  $G_i$  with  $i \geq j + 1$  admits such a path. Since our full query takes the intersection of these two results, it will be nonempty exactly when there exists some index  $j$  satisfying both “all earlier graphs succeed” and “all later graphs fail”. But this is exactly the odd-index condition required by the source problem, which completes the correctness argument.

Exactly the same reduction goes through if we replace the **simple** restrictor by **trail**, yielding identical bound.

## 4 Incorporating Domain-Specific Operators

Logics considered so far let us combine the topology of a graph with simple data tests restricted to equality. Yet, practical languages operate over concrete domains equipped with a rich set of comparisons and functions.

To add such data operations without worsening complexity, we rely on classical results from the study of constraint databases [23] and embedded finite model theory [19, 6]. These results allow us to extend GQL with typed data operations while preserving data-complexity bounds.

The underlying idea is to “embed” a finite database inside an infinite structure  $\mathbf{S}$  with a decidable first-order theory. Queries can then use formulas over  $\mathbf{S}$  to express operations on domain values. Three notable examples [19, Chapter 5]) include:

- The *Linear Integer Arithmetic (LIA)* structure  $\mathfrak{M}_{\mathbb{N},+,<} = \langle \mathbb{N}, 0, 1, +, \leq \rangle$  – linear arithmetic over integers.
- The *Linear Real Arithmetic (LRA)* structure  $\mathfrak{M}_{\mathbb{R},+,\leq} = \langle \mathbb{R}, 0, 1, +, \leq \rangle$  – linear arithmetic over reals.
- The *Real Ordered Field (ROF)* structure  $\mathfrak{M}_{\mathbb{R},+,\times,\leq} = \langle \mathbb{R}, 0, 1, +, \cdot, \leq \rangle$  – real closed field (Tarski’s structure).

We focus on these numeric structures, though similar approaches exist for other domains like strings (e.g., reducts of the universal automatic structure [7]).

### 4.1 Embedded Finite Model Framework

**Embedded finite structures.** An element  $u$  from the domain  $D$  of  $\mathfrak{M}$  is *definable* over  $\mathfrak{M}$  if there exists a  $\mathfrak{M}$ -formula  $\varphi(x)$  such that for each  $v \in D$  we have  $\llbracket \varphi(x) \rrbracket_{x \mapsto u}^{\mathfrak{M}} = \top$  iff  $u = v$ . For example, over  $\mathfrak{M}_{\mathbb{R},+,\leq}$ , the definable elements are exactly the rationals. An  *$\mathfrak{M}$ -embedded finite relational structure* over  $\sigma$  is a finite relational structure whose domain consists of such definable elements.

**Logics with active-domain quantifiers.** A hybrid transitive-closure operator

$$\text{TC}_{\bar{u},\bar{v}}(\varphi(\bar{u}\bar{v}\bar{p}))$$

is defined like the standard TC except that every intermediate tuple must lie in the active domain. We denote by **FO[HTC]** first-order logic extended with hybrid TC operators.

Active-domain restrictions also apply to second-order quantifiers. In **HSO** for hybrid second-order logic, relation variables range only over the active domain. We write **HESO** for its existential fragment.

For any  $L \in \{\text{FO[HTC]}, \text{HSO}, \text{HESO}\}$ , let  $L[\mathfrak{M}]$  denote the logic extended with predicates from an infinite structure  $\mathfrak{M}$ , enriched with *active-domain quantifiers*  $\exists^{\text{adom}}x$  and  $\exists^{\text{adom}}R$ . A formula is *active-domain* if all quantifiers are of this form.

**Restricted-quantifier collapses.** Over the arithmetic domains introduced above, quantifiers can be restricted to the active domain without losing expressive power:

► **Proposition 4.1** (Restricted-quantifier collapse [6]). *Let  $\mathfrak{M} \in \{\mathfrak{M}_{\mathbb{N},+,<}, \mathfrak{M}_{\mathbb{R},+,\leq}, \mathfrak{M}_{\mathbb{R},+,\times,\leq}\}$ . Over  $\mathfrak{M}$ -embedded structures:*

- (i) *Every FO[HTC][ $\mathfrak{M}$ ]-formula is equivalent to an active-domain FO[HTC][ $\mathfrak{M}$ ]-formula, and its data complexity lies in NL.*
- (ii) *Every HESO[ $\mathfrak{M}$ ]-formula is equivalent to an active-domain HESO[ $\mathfrak{M}$ ]-formula, and its data complexity lies in NP.*

These collapses allow us enriching GQL with operations from  $\mathfrak{M}$  while keeping the usual bounds.

## 4.2 Extending GQL with Data Types

We fix an infinite structure  $\mathfrak{M}$ . and define GQL queries with an  $\mathfrak{M}$ -data type. To this end, we first extend the definition of terms to  $\mathfrak{M}$ -terms defined as follows:

$$\chi := x.a \mid c \mid y \mid f^m(\chi_1, \dots, \chi_m)$$

where  $c$  is an  $\mathfrak{M}$ -definable constant,  $y$  is a variable ranging over the domain of  $\mathfrak{M}$ ,  $f^m$  is an  $m$ -ary function name in the vocabulary of  $\mathfrak{M}$ ,  $x \in \text{Vars}$ ,  $a \in \mathcal{K}$ , and every  $\chi_i$  is an  $\mathfrak{M}$ -term. We also extend the definition of conditions to  $\mathfrak{M}$ -conditions by adding  $\theta := R^m(\chi_1, \dots, \chi_m)$  where  $R^m$  is an  $m$ -ary relation name in the vocabulary of  $\mathfrak{M}$ , and every  $\chi_i$  is an  $\mathfrak{M}$ -term. The semantics of  $\theta$  extends by interpreting  $\chi$  and  $\theta$  “inside”  $\mathfrak{M}$ .

► **Example 4.2.** Continuing Example 3.2, suppose we also want to enforce that the Manhattan distance between the towns is below a fixed  $c$ . We can add the filter

$$|x.x\text{-coord} - y.x\text{-coord}| + |x.y\text{-coord} - y.y\text{-coord}| < c,$$

which requires access to built-in arithmetic relations for absolute value, addition, subtraction, and comparison  $<$ . ┘

Using Proposition 4.1, the upper bounds extend to GQL queries with data types.

► **Theorem 4.3.** *For every  $\mathfrak{M} \in \{\mathfrak{M}_{\mathbb{N},+,<}, \mathfrak{M}_{\mathbb{R},+,\leq}, \mathfrak{M}_{\mathbb{R},+,\times,\leq}\}$ , the data complexity of the evaluation problem of a Boolean GQL query  $\mathcal{Q}$  with an  $\mathfrak{M}$ -data type is*

- *NL-complete if  $\mathcal{Q}$  is restrictor-free;*
- *$P^{NP[\log]}$ -complete otherwise.*

GQL can support multiple data types (e.g., strings, numbers) using disjoint infinite structures with restricted quantifier collapse, while preserving data complexity as long as typed variables range over disjoint domains.

## 5 Logical Graph Querying Benefits

This section advocates viewing graph query languages through a logical lens. Embedding GQL into formal logic simplifies reasoning, clarifies expressiveness, and preserves complexity bounds. We showcase several examples unlocked by this embedding, including schema-level queries and extensions beyond plain GQL.

**Meta-Querying.** GQL does not support quantification over properties or labels, limiting its ability to express meta-level queries such as schema checks or property comparisons. For instance, to return pairs of nodes that hold exactly the same data, we have:

$$\varphi_{eq}(n, n') = \forall k, v \ (\text{prop}(n, k, v) \leftrightarrow \text{prop}(n', k, v))$$

where  $\leftrightarrow$  is used as a shorthand for bidirectional implication. Similarly, to ensure that all bike lanes in Example 3.2 are toll-free, we can replace  $\varphi(u, v)$  with

$$\varphi(u, v) := \exists e \left( E(e) \wedge \text{src}(e, u) \wedge \text{tgt}(e, v) \wedge \text{lbl}(e, \text{“BikeLane”}) \wedge \text{no\_toll}(e) \right)$$

where  $\text{no\_toll}(e) := \neg \exists x : \text{prop}(e, \text{‘toll’}, x)$  verifies the condition holds. While GQL cannot test for the absence of a toll property, this enriched query remains in NL.

**Beyond GQL’s Expressiveness.** GQL is limited in checking conditions on unbounded edge traversals, for instance, checking for decreasing values on edges along a path [16]. The next query expresses this: outputs towns accessible by bike lanes with decreasing lengths:

$$\varphi_{<}(e, e') := \exists e, e' (\varphi_{<}(e, e') \wedge \text{tgt}(e, n) \wedge \text{src}(e', n))$$

where

$$\varphi_{<}(e, e') := [\text{TC}_{u,v}(E(u) \wedge E(v) \wedge \varphi_{\text{incE}}(u, v))](e, e')$$

where  $\varphi_{\text{incE}}(u, v)$  checks if  $u$  and  $v$  are adjacent edges with decreasing lengths:

$$\begin{aligned} \varphi_{\text{incE}}(u, v) := \exists n (\text{src}(v, n) \wedge \text{tgt}(u, n)) \wedge \exists l, l' \\ (\text{prop}(u, \text{“length”}, l) \wedge \text{prop}(v, \text{“length”}, l') \wedge l < l') \end{aligned}$$

Here we compute the transitive closure over edges rather than nodes, as it is in GQL’s iteration.

## 6 Conclusion

In this paper, we have answered a fundamental questions regarding evaluating GQL queries, i.e., its data complexity. Although its complexity ( $\text{P}^{\text{NP}[\log]}$ ) is higher than that of its SQL counterpart (polynomial-time), we have shown that the restrictor-free fragment is in NL. We have shown that our results extend to settings with data.

**Future Work.** We believe that our logical perspectives on graph databases could further advance the study of graph databases. As we have shown, not only do they generalize GQL in expressivity (e.g., meta-querying, cf. Section 5), they also allow us to employ classical techniques from relational structures (e.g., embedded finite model theory).

One interesting future avenue concerns the extension of GQL with aggregation over paths (by, e.g., summing, counting, averaging, over a given path). While aggregation results are available for relational databases (e.g., in embedded finite model theory), most positive results (e.g., see Chapter 5 of [19]) do not extend to paths and remain within first-order logic.

---

## References

- 1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison–Wesley, Reading, MA, 1995.
- 2 Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan L. Reutter, and Domagoj Vrgoc. Foundations of modern query languages for graph databases. *ACM Comput. Surv.*, 50(5):68:1–68:40, 2017. doi:10.1145/3104031.
- 3 Pablo Barceló, Gaëlle Fontaine, and Anthony Widjaja Lin. Expressive path queries on graph with data. *Log. Methods Comput. Sci.*, 11(4), 2015. doi:10.2168/LMCS-11(4:1)2015.
- 4 Pablo Barceló, Leonid Libkin, Anthony Widjaja Lin, and Peter T. Wood. Expressive languages for path queries over graph-structured data. *ACM Trans. Database Syst.*, 37(4):31:1–31:46, 2012. doi:10.1145/2389241.2389250.
- 5 Michael Benedikt. Generalizing finite model theory. In *Logic Colloquium ’03*, pages 3–24. Cambridge University Press, 2006.
- 6 Michael Benedikt and Leonid Libkin. Relational queries over interpreted structures. *J. ACM*, 47(4):644–680, 2000. doi:10.1145/347476.347477.

- 7 Michael Benedikt, Leonid Libkin, Thomas Schwentick, and Luc Segoufin. Definable relations and first-order query languages over strings. *J. ACM*, 50(5):694–751, 2003. doi:10.1145/876638.876642.
- 8 Michael Benedikt, Anthony Widjaja Lin, and Di-De Yen. Revisiting the expressiveness landscape of data graph queries. *CoRR*, abs/2406.17871, 2024. doi:10.48550/arXiv.2406.17871.
- 9 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Containment of conjunctive regular path queries with inverse. In *KR*, 2000.
- 10 GQL Standards Committee. GQL standards website, 2024. Accessed: November 2024. URL: <https://www.gqlstandards.org/>.
- 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 *SIGMOD*, 2022. doi:10.1145/3514221.3526057.
- 12 Diego Figueira, Artur Jež, and Anthony W. Lin. Data path queries over embedded graph databases. In *ACM Symposium on Principles of Database Systems (PODS)*, 2022. doi:10.1145/3517804.3524159.
- 13 Daniela Florescu, Alon Y. Levy, and Dan Suciu. Query containment for conjunctive queries with regular expressions. In *ACM Symposium on Principles of Database Systems (PODS)*, 1998. doi:10.1145/275487.275503.
- 14 Nadime Francis, Amélie Gheerbrant, Paolo Guagliardo, Leonid Libkin, Victor Marsault, Wim Martens, Filip Murlak, Liat Peterfreund, Alexandra Rogova, and Domagoj Vrgoc. GPC: A pattern calculus for property graphs. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 241–250. ACM, 2023. doi:10.1145/3584372.3588662.
- 15 Nadime Francis, Amélie Gheerbrant, Paolo Guagliardo, Leonid Libkin, Victor Marsault, Wim Martens, Filip Murlak, Liat Peterfreund, Alexandra Rogova, and Domagoj Vrgoc. A researcher’s digest of GQL (invited talk). In *International Conference on Database Theory (ICDT)*, volume 255 of *LIPICs*, pages 1:1–1:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.ICDT.2023.1.
- 16 Amélie Gheerbrant, Leonid Libkin, Liat Peterfreund, and Alexandra Rogova. GQL and SQL/PGQ: Theoretical Models and Expressive Power. *Proceedings of the VLDB Endowment (PVLDB)*, 18(6):1798–1810, 2025. doi:10.14778/3725688.3725707.
- 17 Stefan Göller, Richard Mayr, and Anthony Widjaja To. On the computational complexity of verifying one-counter processes. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA*, pages 235–244. IEEE Computer Society, 2009. doi:10.1109/LICS.2009.37.
- 18 Georg Gottlob. Np trees and carnap’s modal logic. *J. ACM*, 42(2):421–457, 1995. doi:10.1145/201019.201031.
- 19 E. Grädel, P. G. Kolaitis, L. Libkin, M. Marx, J. Spencer, M. Y. Vardi, Y. Venema, and S. Weinstein. *Finite Model Theory and Its Applications*. Springer, 2007.
- 20 Alastair Green, Paolo Guagliardo, and Leonid Libkin. Property graphs and paths in gql: Mathematical definitions. Technical Reports TR-2021-01, Linked Data Benchmark Council (LDBC), October 2021. doi:10.54285/ldbc.TZJP7279.
- 21 Jelle Hellings, Bart Kuijpers, Jan Van den Bussche, and Xiaowang Zhang. Walk logic as a framework for path query languages on graph databases. In *International Conference on Database Theory (ICDT)*, 2013. doi:10.1145/2448496.2448512.
- 22 H. V. Jagadish, Angela Bonifati, George Fletcher, Hannes Voigt, and Nikolay Yakovets. *Querying Graphs*. Morgan and Claypool Publishers, 2018.
- 23 Gabriel Kuper, Leonid Libkin, and Jan Paredaens. *Constraint Databases*. Springer, 2000.
- 24 Leonid Libkin. *Elements of finite model theory*, volume 41. Springer, 2004. doi:10.1007/978-3-662-07003-1.

## 13:18 Complexity of Evaluating GQL Queries

- 25 Leonid Libkin, Wim Martens, and Domagoj Vrgoc. Querying graphs with data. *J. ACM*, 63(2):14:1–14:53, 2016. doi:10.1145/2850413.
- 26 Leonid Libkin and Domagoj Vrgoc. Regular path queries on graphs with data. In Alin Deutsch, editor, *International Conference on Database Theory (ICDT)*, pages 74–85. ACM, 2012. doi:10.1145/2274576.2274585.
- 27 Wim Martens, Matthias Niewerth, and Tina Popp. A trichotomy for regular trail queries. *Log. Methods Comput. Sci.*, 19(4), 2023. doi:10.46298/LMCS-19(4:20)2023.
- 28 Alberto O. Mendelzon and Peter T. Wood. Finding regular simple paths in graph databases. *SIAM J. Comput.*, 24(6):1235–1258, 1995. doi:10.1137/S009753979122370X.
- 29 Holger Spakowski and Jörg Vogel.  $\Theta_2^P$ -completeness: A classical approach for new results. In Sanjiv Kapoor and Sanjiva Prasad, editors, *Foundations of Software Technology and Theoretical Computer Science, 20th Conference, FST TCS 2000 New Delhi, India, December 13-15, 2000, Proceedings*, volume 1974 of *Lecture Notes in Computer Science*, pages 348–360. Springer, 2000. doi:10.1007/3-540-44450-5\_28.
- 30 Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 137–146, New York, NY, USA, 1982. Association for Computing Machinery. doi:10.1145/800070.802186.
- 31 Klaus W. Wagner. More complicated questions about maxima and minima, and some closures of NP. *Theor. Comput. Sci.*, 51:53–80, 1987. doi:10.1016/0304-3975(87)90049-1.