Foundations of Graph Path Query Languages (Course Notes)

Diego Figueira

Foundations of Graph Path Query Languages
Course Notes for the Reasoning Web Summer School 2021

Diego Figueira

Univ. Bordeaux, CNRS, Bordeaux INP,
LaBRI, UMR 5800, F-33400, Talence, France

Abstract. We survey some foundational results on querying graph-structured data. We focus on general-purpose navigational query languages, such as regular path queries and its extensions with conjunctions, inverses, and path comparisons. We study complexity, expressive power, and static analysis. The course material should be useful to anyone with an interest in query languages for graph structured data, and more broadly in foundational aspects of database theory.

A graph database is an umbrella term for describing semi-structured data organized by means of *entities* (*i.e.*, nodes) and *relations* (*i.e.*, edges) between these entities. In other words, as a *finite graph*, which emphasizes the holistic, topological aspect of the model, where there is no order between nodes or edges. This is a flexible format, usually with no 'schemas', where adding or deleting data (or even integrating different data sources) does not imply rethinking the modeling of data. Data can be typically stored both in nodes and edges, but the shape of the graph itself is an essential part of the data. Querying mechanisms on this kind of data focus on the *topology* of the underlying graph as well as in the data contained inside the edges and nodes. This flexibility comes at a cost, since relations between entities have to be found in a possibly complex topology, most notably as *paths* or sets of paths in some specific configuration. Indeed a *path* in a graph database can be then seen as a first-order citizen. The most basic querying mechanism is then the problem of finding a "pattern" in the database, given as nodes and paths relating them with certain properties. This is, precisely, the kind of languages we will survey here, sometimes called "path query languages".

Example 1. Consider, for example, a very basic database of academic staff. This can be seen as a graph database, as shown in Figure 1. The kind of queries we're interested in are those which exploit the topology of graph, such as "are there two persons with the same supervisor at friend distance at most 5?" or "find all pairs of co-authors with a common ancestor in the supervisor-relation". ◁

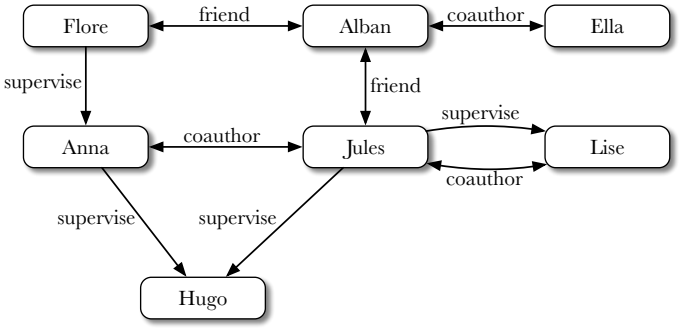

Fig. 1. A simple bibliometric graph database.

Graph databases are relevant to a growing number of applications in areas such as the Semantic Web, knowledge representation, analysis of Social Networks, biological and scientific databases, and others. This data model encompasses formats such as RDF [38], or property graphs. This is why in the last years there has been many theoretical and practical developments for querying graph databases (see [6,45,2,1] for surveys).

One of the most important research trends has hinged on the development of graph query languages that can reason about *topological* aspects of the graph. They are also known as *path query languages*, because topological information in the database typically amounts to querying the existence of paths satisfying certain constraints. The most basic form of navigation consists of querying whether there is a path with a certain property between two nodes. This type of queries have been introduced as Regular Path Queries, or RPQ [40], and it has laid the foundations of many more expressive query languages, including Conjunctive Regular Path Queries (CRPQ) [28] or Extended CRPQ (ECRPQ) [9].

Outline This brief survey concerns the computational task of querying graph databases via navigational query languages. We focus on the language of *regular path queries* and its standard extensions. We study the complexity of evaluation and static analysis tasks, and its expressive power.

1 Preliminaries

We will assume familiarity with some basic automata theory notions such as non-deterministic finite automata (NFA), regular languages, regular

expressions and its paradigmatic problems of containment, emptiness and equivalence. We use $\mathbb{A}, \mathbb{B}$ to denote finite alphabets. In our examples, we use the standard syntax for regular expressions over a finite alphabet $\mathbb{A}$

$$\mathit{regexp} ::= \emptyset \mid \varepsilon \mid a \mid \mathit{regexp}{\cdot}\mathit{regexp} \mid \mathit{regexp}{+}\mathit{regexp} \mid \mathit{regexp}^* \mid \mathit{regexp}^+ \qquad a \in \mathbb{A}$$

with the semantics $[\![\ ]\!] : \mathit{regexp} \to 2^{\mathbb{A}^*}$

$$\begin{aligned}
&[\![\emptyset]\!] = \emptyset, \quad [\![\varepsilon]\!] = \{\varepsilon\}, \quad [\![a]\!] = \{a\}, \quad [\![e_1 + e_2]\!] = [\![e_1]\!] \cup [\![e_2]\!],\\
&[\![e^+]\!] = \{u_1 \cdots u_n : n \geq 1 \text{ and } u_i \in [\![e]\!] \text{ for every } i\},\\
&[\![e^*]\!] = \{\varepsilon\} \cup [\![e^+]\!], \quad [\![e_1 \cdot e_2]\!] = \{u \cdot v : u \in [\![e_1]\!], v \in [\![e_2]\!]\}.
\end{aligned}$$

We use the word orderings of

- **prefix**: u is a prefix of v if $v = u \cdot w$ for some w;
- **suffix**: u is a suffix of v if $v = w \cdot u$ for some w;
- **factor** (*a.k.a.* infix, subword): u is a factor of v if $v = w \cdot u \cdot w'$ for some w, w';
- **subsequence** (*a.k.a.* scattered subword): u is a subsequence of v if u is the result of removing some (possibly none) positions from v.

We also use its "proper" versions: u is a **proper prefix** of v if it is a prefix of v and $u \neq v$; and similarly for the other orderings.

We also assume an elementary understanding of some fundamental complexity classes such as PTime, NL, PSpace, ExpSpace, the polynomial hierarchy, etc.

We often blur the distinction between an NFA $\mathcal{A}$ over $\mathbb{A}$ and the language $L(\mathcal{A}) \subseteq \mathbb{A}^*$ it recognizes; and we do similarly for regular expressions. In the sequel we may hence write $w \in c^* \cdot (a + b)^*$ or $w \in \mathcal{A}$. We denote by ε the empty word. We also assume some familiarity with the query language of Conjunctive Queries (CQ) and Unions of CQ (UCQ).

Graph databases We consider a **graph database** over a finite alphabet $\mathbb{A}$ to be a finite edge-labelled directed graph $G = (V, E)$ over a finite set of labels $\mathbb{A}$, where V is a finite set of vertices and $E \subseteq V \times \mathbb{A} \times V$ is the set of labelled edges. We write $u \xrightarrow{a} v$ to denote an edge $(u, a, v) \in E$. It should be stressed that this is often an *abstraction* for formats such as RDF [38] or property graphs (adopted, *e.g.*, by Neo4j). For example, the patterns used in SPARQL [32] (the W3C query language for RDF) are *triplets* rather than *edges*, but this can often be abstracted away by means of extra vertices and edges, without much loss of generality. Also, for most graph database formats, $\mathbb{A}$ may be from a complex infinite domain, and

further nodes may be labelled also with data. Graph databases, as defined here, are then a basic abstraction of these models which allows us to focus on querying the *topology of the graph*.

A (directed) **path** π of length $n \geq 0$ in G is a (possibly empty) sequence of edges of G of the form $(v_0, a_1, v_1), (v_1, a_2, v_2), \ldots, (v_{n-1}, a_n, v_n)$. There is always an empty path starting and ending at the same node. The **label** $\mathit{label}(\pi)$ of π is the word $a_1 \cdots a_n \in \mathbb{A}^*$. When $n = 0$ the label of π is the empty word ε.

2 Conjunctive Regular Path Queries

In graph databases, a fundamental querying mechanism is based on the existence of some paths in the database with certain properties. These properties include that the label of a path must belong to a certain language, or that the starting or terminal vertices of some paths must be equal. This gives rise to the much studied class of *Regular Path Queries* (RPQ) and *Conjunctive Regular Path Queries* (CRPQ) [21].

Example 2. An example of a CRPQ query is

$$Q_1(x) = x \xrightarrow{a^*b} y \wedge x \xrightarrow{(a+b)^*c} y.$$

It outputs all vertices v having one outgoing path with label in a^*b and one outgoing path with label in $(a + b)^*c$. Further these paths must end at the same vertex. ◁

Conjunctive Regular Path Queries (CRPQ) can be understood as the generalization of conjunctive queries with a very simple form of recursion. CRPQ are part of SPARQL, the W3C standard for querying RDF data [38], including well known knowledge bases such as DBpedia and Wikidata. In particular, RPQs are quite popular for querying Wikidata. They are used in over 24% of the queries (and over 38% of the unique queries), according to recent studies [37,16]. More generally, CRPQ constitute a basic building block for query languages on graph-structured data [6].

A **Regular Path Query (RPQ)** over the alphabet $\mathbb{A}$ is a query of the form

$$Q(x, y) = x \xrightarrow{L} y \tag{1}$$

where L is a regular language over $\mathbb{A}$, specified either as an NFA or a regular expression (we will not make a distinction here). Given a graph database G and a pair of node (v, v') therein, we say that the pair (v, v')

satisfies Q if there exists a path π from v to v' such that $label(\pi) \in L$. The result of evaluating Q on G is then the set of all pairs (v, v') of G satisfying Q.

Example 3. Consider the RPQ

$$Q(x, y) = x \xrightarrow{\text{coauthor}^*} y.$$

It retrieves all pairs persons related by a coauthorship. In particular on the graph database defined in Example 1 it retrieves (Anna, Lise), among other pairs. ◁

A conjunctive regular path query (CRPQ) is the closure under projection (*i.e.*, existential quantification) and conjunction of RPQ queries. That is, CRPQ is to RPQ what Conjunctive Queries is to first-order atoms. Concretely, a **Conjunctive Regular Path Query (CRPQ)** is a query of the form

$$Q(x_1, \ldots, x_n) = A_1 \wedge \cdots \wedge A_m$$

where the atoms $A_1, \ldots, A_m$ are RPQ. We call the variables $x_1, \ldots, x_n$ occurring on the left-hand side the **free variables**. Each free variable x_j has to occur also in some atom on the right-hand side, but not every variable on the right-hand side needs to be free.

A **homomorphism** from a CRPQ Q as above to a graph database $G = (V, E)$ is a mapping μ from the variables of Q (free and non-free) to V. Such a homomorphism **satisfies** an RPQ $A(x, y)$ if $(\mu(x), \mu(y))$ satisfies A; and it satisfies Q if it satisfies every RPQ atom of Q. The set of **answers** $Q(G)$ of a CRPQ $Q(x_1, \ldots, x_n)$ over a graph database G is the set of tuples $(v_1, \ldots, v_n)$ of nodes of G such that there exists a satisfying homomorphism for Q on G that maps x_i to v_i for every $1 \leq i \leq n$. We say that a CRPQ is **Boolean** if it has no free variables, in which case $Q(G) = \{()\}$ (where $()$ denotes the empty tuple) if there exists a satisfying homomorphism or $Q(G) = \{\}$ otherwise. We often write $G \models Q$ instead of $Q(G) = \{()\}$. Most of the results we will present hold also for expressive extensions of CRPQ with finite unions and two-way navigation, known as UC2RPQ [18]. However, for simplicity of presentation, we will focus on CRPQ.

Example 4. Consider the (Boolean) CRPQ

$$Q_1() = x \xrightarrow{\text{supervise}^+} x$$

It checks if the supervisor relation has cycles (*i.e.*, it is true whenever there are). Another CRPQ could be

$$Q_2(x) = x \xrightarrow{\text{supervise}^+} y \wedge x \xrightarrow{\text{friend}} y$$

retrieving all persons being friends with some descendant in the supervisor genealogy. ◁

It is worth observing that in the context of graph databases, a **Conjunctive Query (CQ)** is a CRPQ whose every regular expression denotes a language of the form $\{a\}$ for some $a \in \mathbb{A}$. Thus, CQ is included in CRPQ in terms of expressive power.

Alternative semantics For some applications such as transportation problems or DNA matching (see [5] for a more complete list of application scenarios) there is a need to require that the considered paths have no repeated nodes or no repeated edges. In this way, alternative semantics arise if we change the definition of "satisfaction" of an RPQ atom $x \xrightarrow{L} y$ for a given homomorphism μ. In the default (*a.k.a.* **arbitrary path**) semantics, we ask for the existence of any (directed) path from $\mu(x)$ to $\mu(y)$ with $\mathit{label}(\pi) \in L$. In the **trail** semantics, we demand that the path has also no repeated edges, and in the **simple path** semantics, we further enforce that the path must be simple (*i.e.*, no repeating vertices). It then follows that if $\bar{x} \in Q(G)$ under simple path semantics, then $\bar{x} \in Q(G)$ under trail semantics; and if $\bar{x} \in Q(G)$ under trail semantics then $\bar{x} \in Q(G)$ under arbitrary path semantics. But the converse directions do not hold in general. In the sequel we assume that we work with the default (*i.e.*, arbitrary path) semantics unless otherwise stated.

Structural fragments of CRPQ A standard way to define fragments of conjunctive queries is via their underlying graph (*a.k.a.* Gaifman graph). In a similar way, one can define fragments of CRPQ via its *underlying multi-graph*. Concretely, for any set X, let $\wp_2(X)$ denote the set of non-empty subsets of X of size at most 2. The **underlying multi-graph** of a CRPQ Q is the directed multi-graph (V, E, ν) where: V is the set of variables of Q, E is the set of atoms of Q, and $\nu : E \to \wp_2(V)$ is defined as $\nu(x \xrightarrow{L} y) = \{x, y\}$ for every RPQ atom $x \xrightarrow{L} y$ in Q. For a given class $\mathcal{C}$ of multi-graphs, let CRPQ($\mathcal{C}$) be the set of CRPQ whose underlying multi-graph is in $\mathcal{C}$.[1] In the sequel we will rather use the term *graph* to denote the underlying multi-graph of a CRPQ.

[1] Why multi-graphs and not just graphs? It turns out that, contrary to what happens to Conjunctive Queries, the multiplicity of edges makes a difference for some prob-

Example 5. Consider, for example, the CRPQ

$$Q(x,z) = x \xrightarrow{a^*} y \wedge y \xrightarrow{a+b^*} y \wedge x \xrightarrow{b^*} z \wedge z \xrightarrow{(b+c)^*} x$$

In Figure 2 there is its graphic representation and its underlying graph. ◁

graphic representation of Q underlying multi-graph of Q

Fig. 2. Underlying graph of a CRPQ.

3 Evaluation of CRPQ

The evaluation problem is the most fundamental decision problem on databases: the problem of whether a given data is retrieved by a query on a database.

PROBLEM Evaluation problem for a class $\mathcal{Q}$ of (graph database) queries (EVAL-$\mathcal{Q}$)
GIVEN $Q \in \mathcal{Q}$, a graph database G, a tuple $\bar{x}$ of nodes QUESTION Is $\bar{x} \in Q(G)$?

Observe that the evaluation problem has two kind of inputs of very different nature: the query and the database. In terms of size, one should expect the query to be several orders of magnitude smaller than the database, which raises the question of, for example, whether different algorithms running in time $O(2^{|Q|} \cdot |D|)$, $O(|Q| \cdot 2^{|D|})$ or $O(|D|^{|Q|})$ should be justly placed in the same "complexity class". This is the reason why several complexity variants are often considered, useful to understanding the various aspects of the complexity for the evaluation problem. The default one is the **combined complexity**, where one considers both the

lems, such as the containment problem. We need, hence, to have a more fine-grained notion than a simple graph.

query and database as being part of the input. The complexity when one considers the input query Q to be of *constant size* it is the **data complexity** [44]. Hence, an algorithm running in $O(2^{|Q|} \cdot |D|)$ would have exponential combined complexity but linear data complexity. If, on the other hand, one considers the database D to be of constant size we obtain the **query complexity**. There is, on the other hand, the **parameterized complexity** version of this problem in which the 'parameter' is the query, we will not give details here about parameterized complexity, and we refer the interested reader to [29]. On the parameterized complexity, the classes 'FPT' (for Fixed Parameter Tractable) and 'W[1]' are often considered as the PTime and NP analog of classical non-parameterized complexity classes, respectively. The idea is that an algorithm is FPT if it runs in time $O(f(|Q|) \cdot |D|^c)$ for any computable f and constant c. Thus, an algorithm running in time $O(|D|^{|Q|})$ is not (in principle) FPT, but an algorithm running in time $O(2^{|Q|} \cdot |D|)$ is FPT.

Theorem 1 (Folklore). Eval-*CRPQ is*

- NP-*complete in combined complexity,*
- NL-*complete is data complexity,*
- NL-*complete in query complexity,*
- W[1]-*complete in parameterized complexity.*

Eval-*RPQ is*

- NL-*complete in combined complexity,*
- NL-*complete in data complexity,*
- NL-*complete in query complexity,*
- FPT *in parameterized complexity.*

That is, the combined complexity follows the same behavior as that of Conjunctive Queries, with the exception that evaluating the 'atoms' is an NL-complete task —essentially, the classical graph problem of existence of a source to target path. In other words, the lower bounds for combined and parameterized complexities follow from the following classical result for CQ.

Theorem 2. *[19]* Eval-*CQ is*

- NP-*complete in combined complexity,*
- *in* LogSpace *(and in* AC^0*),*
- *in* LogSpace *(and in* AC^0*),*
- W[1]-*complete in parameterized complexity.*

As discussed before, a standard way to define subclasses of CRPQ is by means of its underlying graphs. In the light of the results of Theorem 1 above, one natural concern is whether the combined and parameterized complexities can be improved by considering queries of some 'simple' structure. The question is then: given a class $\mathcal{C}$ of graphs, is Eval-CRPQ($\mathcal{C}$) tractable? Or rather: For which $\mathcal{C}$ is Eval-CRPQ($\mathcal{C}$) tractable?

As it turns out, by straightforward reductions to and from the Conjunctive Query case, we obtain that the RPQ complexity extends to any class of CRPQ defined by a bounded treewidth class.[2] This notion, in fact, characterizes the tractable complexity classes.

Theorem 3 (consequence of [31]). *Assuming* W[1] $\neq$ FPT*, for any class $\mathcal{C}$ of graphs the following are equivalent:*

- Eval-*CRPQ($\mathcal{C}$) is in polynomial time in combined complexity,*
- Eval-*CRPQ($\mathcal{C}$) is* FPT *in parameterized complexity,*
- *$\mathcal{C}$ has bounded treewidth.*

The tractable cases of evaluation has been also extended to larger classes, in which either queries need to be equivalent to queries of bounded treewidth (obtaining FPT tractability) or they have to be homomorphically equivalent[3] to queries of bounded treewidth (obtaining polynomial time tractability) [42].

Alternative semantics Under alternative semantics, things are more complex, since Eval-RPQ is already an NP-complete problem.

Theorem 4. Eval-*RPQ is* NP*-complete both under trail and simple path semantics. Both in data and in combined complexity.*

In fact, NP-completeness under simple path or trail semantics already holds if we fix the query to be $x \xrightarrow{(aa)^*} y$ or $x \xrightarrow{a^*ba^*} y$ [40]. Interestingly, both these semantics enjoy a trichotomy characterization in terms of data complexity: for any fixed query $Q = x \xrightarrow{L} y$ the evaluation problem for Q is either NP-complete, NL-complete, or in LogSpace (even in AC^0, the data complexity of evaluating first-order formulas). What is more, given a query Q, one can effectively decide in which of these three cases falls (for each semantics).

[2] Intuitively, a graph with small treewidth resembles a tree (*e.g.*, trees have treewidth 1 and cacti have treewidth 2). Many results for trees can be generalized to bounded treewidth classes.

[3] For some suitable notion of homomorphism between queries.

Theorem 5 ([5,39]). *For each fixed regular language $L \subseteq \mathbb{A}^*$ and for each $\star \in \{\text{simple-path}, \text{trail}\}$, the data complexity of* Eval-RPQ *for $x \xrightarrow{L} x$ under $\star$-semantics is either* NP*-complete,* NL*-complete or in* AC^0*. Further, these characterizations are effective (and different for each semantics).*

4 Containment for CRPQ

As databases become larger, reasoning about queries (*e.g.*, for optimization) becomes increasingly important. One of the most basic static analysis problems on monotone query languages is that of query containment: *is every result returned by query Q_1 also returned by query Q_2, for every database?* This can be a means for query optimization, as it may avoid evaluating parts of a query, or reduce and simplify the query with an equivalent one. It falls in what is commonly known as *query reasoning* or *static analysis*, since it involves reasoning only about the query, and it may give rise to optimization tasks that can be carried out at compile time (rather than at running time). Furthermore, query containment has proven useful in knowledge base verification, information integration, integrity checking, and cooperative answering [18].

Concretely, given two CRPQ Q_1, Q_2, we say that Q_1 is **contained** in Q_2, denoted by $Q_1 \subseteq Q_2$, if $Q_1(G) \subseteq Q_2(G)$ for every graph database G, which raises the following decision problem for any fragment $\mathcal{Q}$ of CRPQ.

Problem	Containment problem for a class $\mathcal{Q}$ of (graph database) queries (Cont-$\mathcal{Q}$)
Given	$Q_1, Q_2 \in \mathcal{Q}$
Question	Is $Q_1(G) \subseteq Q_2(G)$ for every graph database G?

We say Q_1 is **equivalent** to Q_2, denoted by $Q_1 \equiv Q_2$, if $Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_1$.

The containment problem for RPQ and CRPQ are decidable in PSpace and ExpSpace respectively.

Theorem 6 (Folklore). Cont-*RPQ is* PSpace*-complete.*

In fact, for any two RPQ $Q_1 = x \xrightarrow{L_1} y$ and $Q_2 = x \xrightarrow{L_2} y$ it is easy to see that $Q_1 \subseteq Q_2$ if, and only if, $L_1 \subseteq L_2$. Hence RPQ containment is reducible from and to language containment. Since regular language containment is a PSpace-complete problem, it follows that Cont-RPQ is PSpace-complete. On the other hand, the bounds for CRPQ are somewhat more involved.

Theorem 7 ([18,28]). Cont-*CRPQ is* ExpSpace-*complete.*

It is interesting to remark that the above hardness result holds even for containment of CRPQ $Q_1 \subseteq Q_2$ where Q_1 is of the form $Q_1 = x \xrightarrow{L} y$ and Q_2 is of the form $Q_2 = \bigwedge_i x \xrightarrow{L_1} y$. In other words, Cont-CRPQ($\mathcal{C}$) is already ExpSpace-hard for the class $\mathcal{C}$ of multigraphs having exactly two nodes, and even for Boolean queries.

However, in certain circumstances, the ExpSpace-hardness of the containment problem can be avoided. That is, there are fragments $\mathcal{F}$ of CRPQ whose containment problem is in PSpace or even in lower classes. Which are these fragments? There are two natural systematic ways to define fragments of CRPQ, namely

1. as discussed before, by restricting the "shape" of the query, as in the underlying multigraph when regular expressions are abstracted away; or
2. by restricting the class of regular expressions that may occur in the queries RPQ atoms.

1. Restricting the shape Here we ask the same question as we did for the evaluation problem: given a class of multigraphs $\mathcal{C}$, is Cont-CRPQ($\mathcal{C}$) tractable? Of course here 'tractable' cannot be any better than PSpace, since it is the complexity of Cont-RPQ, corresponding to the graph having two vertices and one edge. It turns out that, just as in the case for Eval-CRPQ($\mathcal{C}$), one can characterize the classes of graphs $\mathcal{C}$ under which Cont-CRPQ($\mathcal{C}$) is in PSpace. However, the graph measure is not treewidth but *bridgewidth*, which we define next.

A **bridge** of a (multi)graph is a minimal set of edges (in the sense of inclusion) whose removal increases the number of connected components (see Figure 3 for some examples). The **bridge-width** of a graph is the maximum size of a bridge therein. Bridge-width is more restrictive than treewidth, in the sense that if a graph has bridge-width at most k then it also has treewidth at most k, but the converse does not necessarily hold. Let us define a class $\mathcal{C}$ of graphs to be **non-trivial** if it contains at least one graph with at least one edge, and let us call it **bridge-tame** if there is a polynomial time function $f : \mathbb{N} \to \mathcal{C}$ such that $f(n)$ has bridgewidth $\geq n$ for every n.

Theorem 8 ([25]). *For every non-trivial class $\mathcal{C}$ of graphs,*

– *if $\mathcal{C}$ has bounded bridge-width, then the containment problem for CRPQ($\mathcal{C}$) is* PSpace-*complete;*

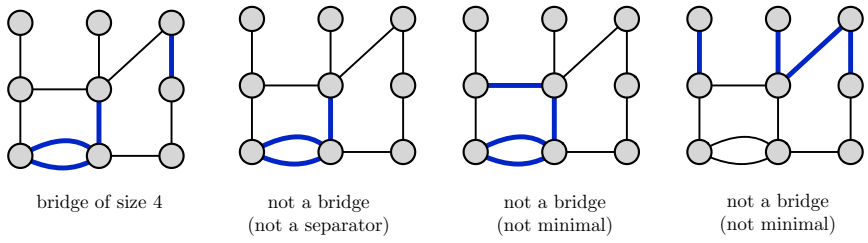

Fig. 3. Examples and non-examples of bridges.

– *if $\mathcal{C}$ is bridge-tame and has unbounded bridge-width, then the containment problem for CRPQ($\mathcal{C}$) is* ExpSpace*-complete.*

2. Restricting the regular expressions As we have remarked before, the lower bound construction of Theorems 7 and 8 make use of CRPQ which have a simple and regular shape (if seen as the underlying graph) but contain rather involved regular expressions, which do not correspond to CRPQ how they typically occur in practice. In fact, a large majority of regular expressions of queries used in practice are of a very simple form [16,17]. This motivates the study of CRPQ containment on fragments having commonly used kinds of regular expressions. The goal here is to identify restricted fragments of CRPQ that are both common in practice and have a reasonable complexity for query containment.

For a class of regular expressions $\mathcal{L}$, let CRPQ($\mathcal{L}$) be the set of CRPQ whose every RPQ atom uses an expression from $\mathcal{L}$.

Let $\mathcal{L}_s$ be the set of regular expressions of the form 's' for each symbol s of the finite alphabet. Let $\mathcal{L}_S$ be the set of expressions of the form '$a_1 + \cdots + a_n$' for $a_1, \ldots, a_n \in \mathbb{A}$ (*i.e.*, it corresponds to unions of $\mathcal{L}_s$). Finally, for $\alpha \in \{s, S\}$, let $\mathcal{L}_{\alpha^*}$ be the set of regular expressions of the form 'r^*' where $r \in \mathcal{L}_\alpha$. We next write $\mathcal{L}_{\alpha,\beta}$ as shorthand for $\mathcal{L}_\alpha \cup \mathcal{L}_\beta$.[4] Following this notation, observe that CRPQ($\mathcal{L}_s$) corresponds to the class of CQ (on graph databases), and that CRPQ($\mathcal{L}_S$) is contained in UCQ in terms of expressive power.

Theorem 9 ([26,23]).

1. *The containment problem for CRPQ($\mathcal{L}_{S,S^*}$) and CRPQ($\mathcal{L}_{s,S^*}$) are* ExpSpace*-complete.*

[4] The choice of the fragments $\mathcal{L}_s$, $\mathcal{L}_S$, $\mathcal{L}_{s^*}$, and $\mathcal{L}_{S^*}$ is based on recent studies on SPARQL queries on Wikidata and DBpedia [17,16,13].

2. *The containment problem for CRPQ($\mathcal{L}_{s,s^*}$), CRPQ($\mathcal{L}_{S,s^*}$), and for CRPQ($\mathcal{L}_S$) are all Π_2^p-complete.*

Observe that CRPQ($\mathcal{L}$) is closed under concatenation in the following sense: Let $\mathcal{L}^{conc}$ be the closure under concatenation[5] of $\mathcal{L}$, then CRPQ($\mathcal{L}$) and CRPQ($\mathcal{L}^{conc}$) are equi-expressive (and there is a linear time translation from one to the other). This means that, for example, the Π_2^p upper bound for CRPQ($\mathcal{L}_{S,s^*}$) also holds for CRPQ having concatenations of expressions of $\mathcal{L}_{S,s^*}$ in the RPQ atoms, like $x \xrightarrow{(a+b)\cdot b^*\cdot(b+c)} y$. Notice also that, in light of the previous Theorem 8, the ExpSpace lower bound of Theorem 9 uses —necessarily— queries of arbitrarily large bridge-width.

5 Boundedness of CRPQ

Boundedness is another important static analysis task of queries with a fixed-point feature. At an intuitive level, a query Q in any such logic is *bounded* if its fixed-point depth, *i.e.*, the number of iterations that are needed to evaluate Q on a database D, is bounded (and thus it is independent of the database D). In databases and knowledge representation, boundedness is regarded as an interesting theoretical phenomenon with relevant practical implications [35,14]. In fact, while several applications in these areas require the use of recursive features, actual real-world systems are either not designed or not optimized to cope with the computational demands that such features impose. Bounded formulas, in turn, can be reformulated in 'non-recursive' logics, such as first-order logic, or even as a union of conjunctive queries (UCQ) when Q itself is positive. Since UCQs form the core of most systems for data management and ontological query answering, it is relevant to understand when a query can be equivalently translated to a UCQ as an optimization task. It has also been experimentally verified in some contexts that recursive features encountered in practice are often used in a somewhat 'harmless' way, and that many of such queries are in fact bounded [33]. We say that a CRPQ query is **bounded** if it is equivalent to some UCQ.

[5] That is, $\mathcal{L}^{conc} = \{s_1 \cdots s_n : \text{ for } n \in \mathbb{N} \text{ and } s_1, \ldots, s_n \text{ expressions from } \mathcal{L}\}$.

Example 6. Consider the following three Boolean CRPQ Q_1, Q_2, Q_3 over the alphabet $\mathbb{A} = \{a, b, c, d\}$ such that

$$Q_1 = (x \xrightarrow{L_b} y \wedge x \xrightarrow{L_{b,d}} y),$$
$$Q_2 = (x \xrightarrow{L_d} y \wedge x \xrightarrow{L_{b,d}} y),$$
$$Q_3 = (x \xrightarrow{L_b + L_d} y \wedge x \xrightarrow{L_{b,d}} y),$$

where $L_b = a^+b^+c$, $L_d = ad^+c^+$, and $L_{b,d} = a^+(b+d)c^+$. As it turns out, Q_1 and Q_2 are unbounded. However, Q_3 is bounded, and in particular, it is equivalent to the UCQ $\varphi_1 \vee \varphi_2$, where

$$\varphi_1 = \exists x_0, x_1, x_2, x_3\, (x_0 \xrightarrow{a} x_1) \wedge (x_1 \xrightarrow{b} x_2) \wedge (x_2 \xrightarrow{c} x_3)$$
$$\varphi_2 = \exists x_0, x_1, x_2, x_3\, (x_0 \xrightarrow{a} x_1) \wedge (x_1 \xrightarrow{d} x_2) \wedge (x_2 \xrightarrow{c} x_3).$$

◁

PROBLEM Boundedness problem for a class $\mathcal{Q}$ of (graph database) queries (BOUND-$\mathcal{Q}$)
GIVEN $Q \in \mathcal{Q}$ QUESTION Is there a UCQ Q' such that $Q \equiv Q'$?

For an RPQ $Q(x, y) = x \xrightarrow{L} y$ it is easy to see that the boundedness problem is really the finiteness problem of L: Q is bounded if, and only if, L is finite (*i.e.*, an NL-complete problem). However, if some of the variables are existentially quantified the problem is not as trivial. For example, a CRPQ of the form $\exists y\ x \xrightarrow{L} y$ is bounded if, and only if, the language

$$L_{prefix} = \{w \in L : \text{ there is no proper prefix of } w \text{ in } L\}$$

is finite [8]. Likewise a CRPQ of the form $\exists x, y\ x \xrightarrow{L} y$ is bounded iff

$$L_{factor} = \{w \in L : \text{ there is no proper factor of } w \text{ in } L\}$$

is finite. Both these problems are already PSPACE-complete [8]. For general CRPQ it turns out that the problem is related to the boundedness problem for an extension of finite automata which associate to each word in the language a natural number or 'cost', called *Distance Automata* [34] (*a.k.a.* weighted automata over the $(\min, +)$-semiring [24], min-automata [15], or $\{\varepsilon, ic\}$-B-automata [20]). The resulting complexity for the boundedness problem for CRPQ is, just as for the containment problem, EXPSPACE-complete.

Theorem 10 ([8]). BOUND-*CRPQ is* EXPSPACE-*complete. If a CRPQ is bounded, then it is equivalent to a UCQ of triple exponential size; and this bound is optimal.*

Contrary to the containment problem, very little is known when restricting either the shape or the languages of the CRPQ with regards to the boundedness problem.

6 Semantic membership for CRPQ

As we have seen before, natural classes of CRPQ with tractable evaluation arise from considering bounded treewidth classes $\mathcal{C}$. However, this is a syntactic property, which begs the following question for any fixed class $\mathcal{C}$ of bounded treewidth: given a CRPQ Q, is it equivalent to some query Q' from CRPQ($\mathcal{C}$)? If so, we can replace the costly query Q with Q', or adapt the strategy for the (polynomial) evaluation of Q' to Q. The idea behind this optimization task is —as for boundedness— that the time needed to compute Q' may be comparatively small to the gain of having a polynomial time algorithm for the evaluation problem. Let $\mathcal{T}_k$ be the set of all multigraphs of treewidth at most k. We can then consider the following family of decision problems.

PROBLEM Treewidth-k semantic membership (MEM-tw_k)
GIVEN a CRPQ Q
QUESTION Is there a query Q' in CRPQ($\mathcal{T}_k$) such that $Q \equiv Q'$?

For the case where the input query turns out to be a Conjunctive Query his is a studied problem which is decidable, NP-complete [22] (basically, it reduces to testing treewidth of the *core* of a graph). However, for CRPQ this problem turns out to be more challenging. It has been shown to be decidable only for $k = 1$, that is, for 'trees', where trees should be understood as the class of multigraphs whose every simple cycle is of length 1 (*i.e.*, a self-loop) or 2 (*i.e.*, a cycle between a parent and a child).

Theorem 11 ([11]). MEM-tw_1 *is decidable,* EXPSPACE-*complete.*

It is however unknown if MEM-tw_k is decidable for any other k.

7 Extending CRPQ with union and two-wayness

Two-wayness. Observe that semantics of RPQ and CRPQ are based on the notion of directed path. This means that the query cannot "freely"

move around the graph edges, but it has to comply with the direction of edges. Remark that not even the reachability query "x and y belong to the same connected component in the underlying undirected graph" is expressible as a CRPQ. A standard extension of CRPQ and RPQ that palliates this lack of expressive power, consists in adding the ability to navigate the graph database with inverse relations, and it is known as **C2RPQ** and **2RPQ**, respectively (2 for "two-way navigation").

For any alphabet $\mathbb{A}$, let us define the alphabet $\mathbb{A}^{\pm} := \mathbb{A} \dot\cup \mathbb{A}^{-1}$ that extends $\mathbb{A}$ with the set $\mathbb{A}^{-1} := \{a^{-1} \mid a \in \mathbb{A}\}$ of "inverse" symbols. For a graph database G over $\mathbb{A}$, let $G^{\pm}$ be the result of adding an edge (v, a^{-1}, v') for every $(v', a, v) \in E$.

Now the regular expressions of a (C)2RPQ are defined over the extended alphabet $\mathbb{A}^{\pm}$. The semantics of a 2RPQ is extended as expected: a pair (v, v') of vertices of G satisfies a 2RPQ $x \xrightarrow{L} y$ if there is a path π in $G^{\pm}$ from v to v' such that $\mathit{label}(\pi) \in L$ (remember, now L is a regular subset of $(\mathbb{A}^{\pm})^*$). The semantics of C2RPQ follows the same definition based on 2RPQ.

Example 7. Consider the following 2RPQ

$$Q(x, y) = x \xrightarrow{(\text{supervise} \,\cdot\, \text{supervise}^{-1})^*} y.$$

On a graph database as the one of Example 1, Q returns pairs of people related by a "co-supervision" chain. ◁

Union. A CRPQ, contrary to a CQ, has some restricted built-in union by the simple fact that regular languages are closed under union. However, the general structure of the query is fixed.

Example 8. Consider the following two Boolean CQs

$$Q_1 = x \xrightarrow{a} y$$
$$Q_2 = x \xrightarrow{b} x$$

It can be shown that there is no CRPQ expressing $Q_1 \vee Q_2$. ◁

As for Conjunctive Query, it is a rather standard extension to add the possibility to have finite unions of queries, and it is known as **UCRPQ**. A UCRPQ is thus a query of the form $Q = Q_1 \vee \cdots \vee Q_n$, where every Q_i has the same set of free variables. We then define $\bar{x} \in Q(G)$ for a graph database G if $\bar{x} \in Q_i(G)$ for some i.

Finally, the extension including both possibilities of having two-way navigation as well as unions is denoted by **UC2RPQ**, and its semantics is as expected.

As it turns out, most known results extend to UC2RPQ in a seamless way: it just turns out that upper bound techniques (involving invariably some classes of automata) can extend to handle UC2RPQ (involving classes of *two-way* automata). In particular, the results on the containment and evaluation problems of Theorems 1, 3, 6, 7, 8, 10, and 11 extend to UC2RPQ while preserving the stated upper bounds.

8 Extending CRPQ with path comparison

Another studied extension of CRPQ conveys the ability to compare paths for certain relations on the labels. In comparison, one can think of CRPQ as testing for unary relations.

Example 9. Observe that the query Q_1 from Example 2 could be equivalently written as

$$\begin{aligned} Q_1(x) \quad = \quad & \exists y\, \exists \pi_1, \pi_2 \; x \xrightarrow{\pi_1} y \wedge x \xrightarrow{\pi_2} y \wedge \\ & \mathit{label}(\pi_1) \in L_{a^*b} \wedge \mathit{label}(\pi_2) \in L_{(a+b)^*c} \end{aligned}$$

Where $L_\star$ is the language given by the expression $\star$. With this notation in mind, consider the following query

$$\begin{aligned} Q_2(x) \quad = \quad & \exists y\, \exists \pi_1, \pi_2 \; x \xrightarrow{\pi_1} y \wedge y \xrightarrow{\pi_2} x \wedge \\ & (\mathit{label}(\pi_1), \mathit{label}(\pi_2)) \in R \end{aligned}$$

where $R \subseteq \mathbb{A}^* \times \mathbb{A}^*$ is now a *word relation* such as, for example, the equality relation $R = \{(u, v) \in \mathbb{A}^* \times \mathbb{A}^* : u = v\}$. Such query Q_2 would then output all vertices v having one cycling path with label $w\,w$ for some $w \in \mathbb{A}^*$. ◁

Indeed, some scenarios require the ability to "compare paths", *i.e.*, to relate the words given by the labels corresponding to the existentially quantified paths in CRPQ. For example, when handling biological sequences, querying paths constrained under some *similarity* measure between paths is of great importance. See [9] for a more detailed discussion on the applicability of these features. The extension of CRPQ with non-monadic word relations gives rise to several expressive extensions which have been studied lately [43,9,7,10,3,4]. For a class of finite word relations $\mathcal{K}$ one

can consider "CRPQ+$\mathcal{K}$", the result of extending CRPQ with testing of $\mathcal{K}$ relations on path labels.

Concretely, for any class $\mathcal{K}$ of finite word relations, the query language of **conjunctive regular path query with $\mathcal{K}$-relations (CRPQ+$\mathcal{K}$)** is a pair $(Q, \mathcal{R})$ where $\mathcal{R} \subseteq \mathcal{K}$ is a finite set of relations, each $R \in \mathcal{R}$ having arity $\mathit{arity}(R) \geq 1$. A CRPQ+$\mathcal{K}$ query Q, possibly having some free variables $\bar{x}$, is a query of the form

$$Q(\bar{x}) = \exists \bar{y}\, \exists \bar{\pi}\ \gamma(\bar{x}\bar{y}\bar{\pi}) \wedge \rho(\bar{\pi}). \tag{2}$$

It is a query over two sorts of (infinite, disjoint) sets of variables: **node variables** (denoting nodes of the graph database) and **path variables** (denoting paths). In (2), $\bar{x}$, $\bar{y}$ span over node variables and $\bar{\pi}$ over path variables. The idea is that γ tells how node variables are connected through path variables, while ρ describes the properties and relations between path variables in terms of the regular languages and relations of $\mathcal{R}$. Concretely, the subformula $\gamma(\bar{x}\bar{y}\bar{\pi})$, which we may call the **reachability subquery**, is a finite conjunction of **reachability atoms** of the form $z \xrightarrow{\pi} z'$, where z, z' are from $\bar{x}\bar{y}$ and π is from $\bar{\pi}$, with the restriction that every path variable π from $\bar{\pi}$ appears in exactly one reachability atom. That is, node variables may repeat in γ, but path variables may not. Let us call the subformula $\rho(\bar{\pi})$ the **relation subquery**, which is a finite conjunction of atoms of the form $R(\pi_1, \ldots, \pi_r)$, were $R \in \mathcal{R}$, $r = \mathit{arity}(R)$ and $\pi_1, \ldots, \pi_r$ are pairwise distinct path variables from $\bar{\pi}$.

For the classes of relations $\mathcal{K}$ we will consider here, we can think of each relation $R \in \mathcal{K}$ of arity $k \geq 1$ over an alphabet $\mathbb{A}$ as being described by an NFA over the alphabet of k-tuples $(\mathbb{A} \dot{\cup} \{\bot\})^k$. The underlying idea is that a word $w \in ((\mathbb{A} \dot{\cup} \{\bot\})^k)^*$ describes the k-tuple $(w_1, \ldots, w_k) \in (\mathbb{A}^*)^k$, where each w_i is obtained from w by (1) projecting onto the i-th component and (2) replacing each $\bot$ with the empty word ε. Thus, for example $(a, \bot, \bot)(b, b, \bot)(\bot, \bot, a)(c, \bot, \bot)$ describes (abc, b, a). In this way, any such NFA $\mathcal{A}$ denotes the k-ary relation R consisting of all tuples described by the words in the language of $\mathcal{A}$. Among the most basic classes of finite word relations are the classes of Recognizable, Synchronous (*a.k.a.* Automatic or Regular), and Rational relations [12]. The class of **Rational relations** is the set of all relations which are recognized by such automata, and it includes relations such as factor or subsequence. **Synchronous relations** are those that can be recognized by automata whose every word satisfies that, for every $i \leq k$, if a position has a $\bot$-symbol in its i-th component, then the next position (if it exists) must also have $\bot$ in its i-th component. For example, prefix or equal-length

are synchronous relations. Finally, **Recognizable relations** are equivalent to finite unions of products of regular languages, *i.e.*, relations of the form $R = \bigcup_{i \in I} L_{i,1} \times \cdots \times L_{i,k}$ for a finite I, where the $L_{i,j}$'s are all regular languages over $\mathbb{A}$. They can also be defined in terms of NFA over $(\mathbb{A} \dot\cup \{\bot\})^k$ restricted to only accepting words such that: (1) no position contains more than one symbol from $\mathbb{A}$ and (2) every component projects onto a word from $\bot^* \cdot \mathbb{A}^* \cdot \bot^*$. An instance of a recognizable relation is $\{(u, v) : u, v \text{ start with the same letter}\}$ or $\{(u, v) : |u| + |v| = 3 \bmod 7\}$. These three classes form a proper hierarchy: Recognizable $\subsetneq$ Synchronous $\subsetneq$ Rational. Observe that any of the three classes of word relations contains the class of regular languages as (unary) relations. We refer the reader to [12] for more details on these classes.

Given a graph database $D = (V, E)$ over an alphabet $\mathbb{A}$ and an assignment f_n of $\bar{x}\bar{y}$ to V and an assignment f_p of $\bar{\pi}$ to paths of D, we say that (f_n, f_p) is a **satisfying assignment** if

1. for every reachability atom $z \xrightarrow{\pi} z'$ of γ, $f_p(\pi)$ is a directed path from $f_n(z)$ to $f_n(z')$ in D; and
2. for every r-ary atom $R(\pi_1, \ldots, \pi_r)$ of ρ the tuple
$$(\mathit{label}(f_p(\pi_1)), \ldots, \mathit{label}(f_p(\pi_r))) \in (\mathbb{A}^*)^r$$
is in the relation $R \in \mathcal{R}$.

Assuming $\bar{x} = (x_1, \ldots, x_\ell)$, the answers $Q(D)$ of the CRPQ+$\mathcal{K}$ query Q to the database D is the set of all $(f_n(x_1), \ldots, f_n(x_\ell)) \in V^\ell$ for every satisfying assignment (f_n, f_p).

It is plain to see that CRPQ+Recognizable is not more expressive than UCRPQ. On the other hand, the evaluation of CRPQ+Rational queries is undecidable, even for very simple rational relations. On the contrary, CRPQ+Synchronous seems to enjoy a good tradeoff of complexity and expressive power. This is partly because Synchronous relations constitute a very robust class, closed under Boolean operations and enjoying most of the decidability and algorithmic properties inherited from regular languages. CRPQ+Synchronous, commonly known as **ECRPQ** ('Extended' CRPQ).

Theorem 12.

- *(Folklore) CRPQ+Recognizable is contained in UCRPQ in terms of expressive power.*
- *[9]* Eval-*CRPQ+Recognizable and* Eval-*CRPQ+Synchronous (a.k.a. ECRPQ) are both* PSpace-*complete [resp.* NL-*complete] in combined [resp. data] complexity.*

- *[9]* Eval-*CRPQ+Rational is undecidable.*
- *[7]* Eval-*CRPQ+*($Synchronous \cup \{R\}$) *is undecidable, for any* $R \in \{suffix, factor\}$*;* Eval-*CRPQ+*($Synchronous \cup \{subsequence\}$) *is decidable and non-multiply-recursive hard.*[6]
- *[10]* Eval-*CRPQ+*$\{factor\}$ *is* PSpace*-complete (both in data and combined complexity);* Eval-*CRPQ+*$\{subsequence\}$ *is* NExpTime*-complete [resp.* NP*-complete] in combined [resp. data] complexity.*

Observe that the data complexity for the evaluation of ECRPQ queries is the same as for CRPQ (*i.e.*, NL), but the combined complexity jumps from NP to PSpace. In the same spirit as done for CRPQ in Theorem 3, there exists a characterization of the underlying structures $\mathcal{C}$ for which Eval-ECRPQ($\mathcal{C}$) has better complexity. While the underlying structure of a CRPQ is the result of abstracting away its *languages*, the underlying structure of an ECRPQ is the result of abstracting away its *relations*. In contrast to the evaluation problem for CRPQ, the complexity of Eval-ECRPQ($\mathcal{C}$) can be, depending on $\mathcal{C}$, either PTime, NP, or PSpace in combined complexity, and either XNL, W[1], or FPT in parameterized complexity [27]. Further, the FPT and PTime cases do not coincide.

While the evaluation problem for ECRPQ remains decidable, the containment and equivalence problems turn out to be (roubstly) undecidable.

Theorem 13. Cont-*ECRPQ is undecidable [9]. Further, undecidability holds even for the fragment CRPQ+*($\mathcal{RL} \cup \{eq\text{-}len\}$)*, where eq-len is the equal length binary relation and* $\mathcal{RL}$ *is the class of regular languages (seen as unary relations). In fact, this undecidability results even holds when one of the two inputs is a plain CRPQ (which is the same as CRPQ +* $\mathcal{RL}$*) [30].*

Other extensions

A different extension to CRPQ with an expanded ability for path relational querying, consists in having "xregex" regular expressions with string variables (*a.k.a.* backreferences) [43], which is incomparable, in terms of expressive power, to ECRPQ.

As we remarked before, these graph query languages we have covered operate on an abstraction over a finite alphabet $\mathbb{A}$ of graph databases. However, graph databases carry so called "data values" in the nodes

[6] In particular, this means that the time or space required by any algorithm deciding Eval-CRPQ+(Synchronous $\cup$ $\{subsequence\}$) grows faster than the Ackermann function.

and/or edges, that is, values with concrete domains, such as strings or numbers, and practical query languages can of course make tests on these data values, not only for equality but also using some domain-specific functions and relations. There have been proposals for querying mechanisms combining the ability to *test for data values* and *query the topology* [36]. In particular, one can explore different forms of querying paths while constraining the way data in paths changes. However, the theory of querying "data graphs" (*i.e.*, graph databases carrying elements from infinite domains) remains insofar a largely unexplored terrain.

The graph pattern languages here are based on a very simple form of recursion, namely applying regular expressions on paths. Another possible extension is by allowing a more complex form of recursion, such as nested regular expressions or Datalog-like rules, which increases the expressive power while often preserving complexities of CRPQ (see, *e.g.*, [41] and references therein).

9 Conclusions

We have explored some fundamental ways of querying graph databases via the concept of 'paths' on a simple abstractions of graph databases, that of edge-labelled graphs. One can draw a parallel between formalisms of CRPQ and its extensions and Conjunctive Queries, in the sense that these corresponds to the most basic form of querying, using existentially quantified "patterns", by means of languages closed under homomorphisms. However, in this scenario, one deals with two-sorted languages dealing with *nodes* and *paths*, inheriting from the theory of words (or word relations) and of testing patterns (*i.e.*, Conjunctive Queries).

References

1. Angles, R., Arenas, M., Barceló, P., Hogan, A., Reutter, J.L., Vrgoc, D.: Foundations of modern query languages for graph databases. ACM Comput. Surv. **50**(5), 68:1–68:40 (2017). https://doi.org/10.1145/3104031, https://doi.org/10.1145/3104031
2. Angles, R., Gutiérrez, C.: Survey of graph database models. ACM Comput. Surv. **40**(1), 1:1–1:39 (2008). https://doi.org/10.1145/1322432.1322433, https://doi.org/10.1145/1322432.1322433
3. Anyanwu, K., Maduko, A., Sheth, A.P.: SPARQ2L: towards support for subgraph extraction queries in rdf databases. In: Williamson, C.L., Zurko, M.E., Patel-Schneider, P.F., Shenoy, P.J. (eds.) Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007. pp. 797–806. ACM (2007). https://doi.org/10.1145/1242572.1242680, https://doi.org/10.1145/1242572.1242680

4. Anyanwu, K., Sheth, A.P.: ρ-queries: enabling querying for semantic associations on the semantic web. In: Hencsey, G., White, B., Chen, Y.R., Kovács, L., Lawrence, S. (eds.) Proceedings of the Twelfth International World Wide Web Conference, WWW 2003, Budapest, Hungary, May 20-24, 2003. pp. 690–699. ACM (2003). https://doi.org/10.1145/775152.775249, https://doi.org/10.1145/775152.775249
5. Bagan, G., Bonifati, A., Groz, B.: A trichotomy for regular simple path queries on graphs. J. Comput. Syst. Sci. **108**, 29–48 (2020). https://doi.org/10.1016/j.jcss.2019.08.006, https://doi.org/10.1016/j.jcss.2019.08.006
6. Barceló, P.: Querying graph databases. In: ACM Symposium on Principles of Database Systems (PODS). pp. 175–188. ACM (2013)
7. Barceló, P., Figueira, D., Libkin, L.: Graph logics with rational relations. Logical Methods in Computer Science (LMCS) **9**(3:01) (2013). https://doi.org/10.2168/LMCS-9(3:1)2013
8. Barceló, P., Figueira, D., Romero, M.: Boundedness of conjunctive regular path queries. In: International Colloquium on Automata, Languages and Programming (ICALP). Leibniz International Proceedings in Informatics (LIPIcs), vol. 132, pp. 104:1–104:15. Leibniz-Zentrum für Informatik (2019). https://doi.org/10.4230/LIPIcs.ICALP.2019.104
9. Barceló, P., Libkin, L., Lin, A.W., Wood, P.T.: Expressive languages for path queries over graph-structured data. ACM Transactions on Database Systems (TODS) **37**(4), 31 (2012). https://doi.org/10.1145/2389241.2389250
10. Barceló, P., Muñoz, P.: Graph logics with rational relations: The role of word combinatorics. ACM Trans. Comput. Log. **18**(2), 10:1–10:41 (2017). https://doi.org/10.1145/3070822, https://doi.org/10.1145/3070822
11. Barceló, P., Romero, M., Vardi, M.Y.: Semantic acyclicity on graph databases. SIAM J. Comput. **45**(4), 1339–1376 (2016). https://doi.org/10.1137/15M1034714, https://doi.org/10.1137/15M1034714
12. Berstel, J.: Transductions and context-free languages, Teubner Studienbücher : Informatik, vol. 38. Teubner (1979), https://www.worldcat.org/oclc/06364613
13. Bielefeldt, A., Gonsior, J., Krötzsch, M.: Practical linked data access via SPARQL: the case of wikidata. In: Workshop on Linked Data on the Web (LDOW) (2018)
14. Bienvenu, M., Hansen, P., Lutz, C., Wolter, F.: First order-rewritability and containment of conjunctive queries in horn description logics. In: International Joint Conference on Artificial Intelligence (IJCAI). pp. 965–971 (2016)
15. Bojańczyk, M., Toruńczyk, S.: Deterministic automata and extensions of weak MSO. In: IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS). Leibniz International Proceedings in Informatics (LIPIcs), vol. 4, pp. 73–84. Leibniz-Zentrum für Informatik (2009). https://doi.org/10.4230/LIPIcs.FSTTCS.2009.2308
16. Bonifati, A., Martens, W., Timm, T.: Navigating the maze of Wikidata query logs. In: World Wide Web Conference (WWW). pp. 127–138 (2019)
17. Bonifati, A., Martens, W., Timm, T.: An analytical study of large SPARQL query logs. VLDB Journal (2020), to appear, https://doi.org/10.1007/s00778-019-00558-9
18. Calvanese, D., De Giacomo, G., Lenzerini, M., Vardi, M.Y.: Containment of conjunctive regular path queries with inverse. In: Principles of Knowledge Representation and Reasoning (KR). pp. 176–185 (2000)
19. Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational data bases. In: Hopcroft, J.E., Friedman, E.P., Harrison, M.A.

(eds.) Proceedings of the 9th Annual ACM Symposium on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA. pp. 77–90. ACM (1977). https://doi.org/10.1145/800105.803397, https://doi.org/10.1145/800105.803397
20. Colcombet, T.: The theory of stabilisation monoids and regular cost functions. In: International Colloquium on Automata, Languages and Programming (ICALP). Lecture Notes in Computer Science, vol. 5556, pp. 139–150. Springer (2009). https://doi.org/10.1007/978-3-642-02930-1_12
21. Consens, M.P., Mendelzon, A.O.: Graphlog: a visual formalism for real life recursion. In: Rosenkrantz, D.J., Sagiv, Y. (eds.) Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, April 2-4, 1990, Nashville, Tennessee, USA. pp. 404–416. ACM Press (1990). https://doi.org/10.1145/298514.298591, https://doi.org/10.1145/298514.298591
22. Dalmau, V., Kolaitis, P.G., Vardi, M.Y.: Constraint satisfaction, bounded treewidth, and finite-variable logics. In: Hentenryck, P.V. (ed.) Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002, Ithaca, NY, USA, September 9-13, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2470, pp. 310–326. Springer (2002). https://doi.org/10.1007/3-540-46135-3_21, https://doi.org/10.1007/3-540-46135-3_21
23. Deutsch, A., Tannen, V.: Optimization properties for classes of conjunctive regular path queries. In: Ghelli, G., Grahne, G. (eds.) Database Programming Languages, 8th International Workshop, DBPL 2001, Frascati, Italy, September 8-10, 2001, Revised Papers. Lecture Notes in Computer Science, vol. 2397, pp. 21–39. Springer (2001). https://doi.org/10.1007/3-540-46093-4_2, https://doi.org/10.1007/3-540-46093-4_2
24. Droste, M., Kuich, W., Vogler, H.: Handbook of weighted automata. Springer Science & Business Media (2009)
25. Figueira, D.: Containment of UC2RPQ: the hard and easy cases. In: International Conference on Database Theory (ICDT). Leibniz International Proceedings in Informatics (LIPIcs), Leibniz-Zentrum für Informatik (2020)
26. Figueira, D., Godbole, A., Krishna, S., Martens, W., Niewerth, M., Trautner, T.: Containment of simple conjunctive regular path queries. In: Principles of Knowledge Representation and Reasoning (KR) (2020), https://hal.archives-ouvertes.fr/hal-02505244
27. Figueira, D., Ramanathan, V.: When is the evaluation of Extended CRPQ tractable? (2021), https://hal.archives-ouvertes.fr/hal-03353483, working paper or preprint
28. Florescu, D., Levy, A., Suciu, D.: Query containment for conjunctive queries with regular expressions. In: ACM Symposium on Principles of Database Systems (PODS). pp. 139–148. ACM Press (1998). https://doi.org/10.1145/275487.275503
29. Flum, J., Grohe, M.: Parameterized Complexity Theory. Texts in Theoretical Computer Science. An EATCS Series, Springer (2006). https://doi.org/10.1007/3-540-29953-X, https://doi.org/10.1007/3-540-29953-X
30. Freydenberger, D.D., Schweikardt, N.: Expressiveness and static analysis of extended conjunctive regular path queries. J. Comput. Syst. Sci. **79**(6), 892–909 (2013). https://doi.org/10.1016/j.jcss.2013.01.008, https://doi.org/10.1016/j.jcss.2013.01.008
31. Grohe, M., Schwentick, T., Segoufin, L.: When is the evaluation of conjunctive queries tractable? In: Vitter, J.S., Spirakis, P.G., Yannakakis, M. (eds.) Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece. pp. 657–666. ACM (2001). https://doi.org/10.1145/380752.380867, https://doi.org/10.1145/380752.380867

32. Gyssens, M., Paredaens, J., Gucht, D.V.: A graph-oriented object database model. In: Rosenkrantz, D.J., Sagiv, Y. (eds.) Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, April 2-4, 1990, Nashville, Tennessee, USA. pp. 417–424. ACM Press (1990). https://doi.org/10.1145/298514.298593, https://doi.org/10.1145/298514.298593
33. Hansen, P., Lutz, C., Seylan, I., Wolter, F.: Efficient query rewriting in the description logic EL and beyond. In: International Joint Conference on Artificial Intelligence (IJCAI). pp. 3034–3040 (2015)
34. Hashiguchi, K.: Limitedness theorem on finite automata with distance functions. Journal of Computer and System Sciences (JCSS) **24**(2), 233–244 (1982)
35. Hillebrand, G.G., Kanellakis, P.C., Mairson, H.G., Vardi, M.Y.: Tools for datalog boundedness. In: ACM Symposium on Principles of Database Systems (PODS). pp. 1–12 (1991)
36. Libkin, L., Martens, W., Vrgoc, D.: Querying graphs with data. J. ACM **63**(2), 14:1–14:53 (2016). https://doi.org/10.1145/2850413, https://doi.org/10.1145/2850413
37. Malyshev, S., Krötzsch, M., González, L., Gonsior, J., Bielefeldt, A.: Getting the most out of Wikidata: Semantic technology usage in Wikipedia's knowledge graph. In: International Semantic Web Conference (ISWC). pp. 376–394 (2018)
38. Manola, F., Miller, E., McBride, B., et al.: Rdf primer. W3C recommendation **10**(1-107), 6 (2004)
39. Martens, W., Niewerth, M., Trautner, T.: A trichotomy for regular trail queries. In: Paul, C., Bläser, M. (eds.) 37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France. LIPIcs, vol. 154, pp. 7:1–7:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020). https://doi.org/10.4230/LIPIcs.STACS.2020.7, https://doi.org/10.4230/LIPIcs.STACS.2020.7
40. Mendelzon, A.O., Wood, P.T.: Finding regular simple paths in graph databases. SIAM J. Comput. **24**(6), 1235–1258 (1995). https://doi.org/10.1137/S009753979122370X, https://doi.org/10.1137/S009753979122370X
41. Reutter, J.L., Romero, M., Vardi, M.Y.: Regular queries on graph databases. Theory Comput. Syst. **61**(1), 31–83 (2017). https://doi.org/10.1007/s00224-016-9676-2, https://doi.org/10.1007/s00224-016-9676-2
42. Romero, M., Barceló, P., Vardi, M.Y.: The homomorphism problem for regular graph patterns. In: Annual Symposium on Logic in Computer Science (LICS). pp. 1–12. IEEE Computer Society Press (2017). https://doi.org/10.1109/LICS.2017.8005106, https://doi.org/10.1109/LICS.2017.8005106
43. Schmid, M.L.: Conjunctive regular path queries with string variables. In: Suciu, D., Tao, Y., Wei, Z. (eds.) Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020. pp. 361–374. ACM (2020). https://doi.org/10.1145/3375395.3387663, https://doi.org/10.1145/3375395.3387663
44. Vardi, M.Y.: The complexity of relational query languages (extended abstract). In: Lewis, H.R., Simons, B.B., Burkhard, W.A., Landweber, L.H. (eds.) Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA. pp. 137–146. ACM (1982). https://doi.org/10.1145/800070.802186, https://doi.org/10.1145/800070.802186

45. Wood, P.T.: Query languages for graph databases. SIGMOD Rec. **41**(1), 50–60 (2012). https://doi.org/10.1145/2206869.2206879, https://doi.org/10.1145/2206869.2206879