# Querying Graph Databases

Pablo Barceló
Dept. of Computer Science, University of Chile
pbarcelo@dcc.uchile.cl

## ABSTRACT

Graph databases have gained renewed interest in the last years, due to their applications in areas such as the Semantic Web and Social Networks Analysis. We study the problem of querying graph databases, and, in particular, the expressiveness and complexity of evaluation for several general-purpose navigational query languages, such as the regular path queries and its extensions with conjunctions and inverses. We distinguish between two semantics for these languages. The first one, based on simple paths, easily leads to intractability in data complexity, while the second one, based on arbitrary paths, allows tractable evaluation for an expressive family of languages.

We also study two recent extensions of these languages that have been motivated by modern applications of graph databases. The first one allows to treat paths as first-class citizens, while the second one permits to express queries that combine the topology of the graph with its underlying data.

## Categories and Subject Descriptors

H.2.3 [**Database Management**]: Languages—*Query Languages*

## Keywords

Graph databases, conjunctive regular path queries, query evaluation, expressiveness, containment.

## 1. INTRODUCTION

Graph databases are crucial for many applications in which the topology of the data is as important as the data itself. While early interest in graph databases could be explained by their applications in hypertext systems [33], or their connections with semistructured data [3, 24] and object databases [54], new application domains have taken the field by storm in the last decade, including the Semantic Web [10], social networks analysis [42], biological networks [63], data provenance [7], and several others.

In their simplest form, graph databases are finite, directed, edge-labeled graphs. We study the problem of querying those graph databases. An obvious question one faces regarding this problem is whether it could be tackled applying existing relational technology; that is, by first representing each graph database $\mathcal{G}$ as a relational database $\mathcal{D}(\mathcal{G})$ (in a standard way), and then querying $\mathcal{D}(\mathcal{G})$ (using a relational language) instead of $\mathcal{G}$. The drawback of this approach is that many graph database queries are navigational (e.g., *regular path queries* [36], that check the existence of a path between two nodes whose label satisfies a regular condition), and, thus, they cannot be easily expressed by relational languages, such as SQL, which allow limited recursion.

On the other hand, several navigational languages for graph databases can be embedded into the relational language Datalog [2], which is the recursive extension of the class of unions of conjunctive queries. Nevertheless, this translation is not really fruitful: While data complexity of Datalog is PTIME-complete (and, thus, not parallelizable under widely-held complexity assumptions [51]), the data complexity of many languages we study along the paper falls into the parallelizable class NLOGSPACE. Moreover, while some basic static analysis tasks for Datalog are undecidable (e.g. containment [2]), they are decidable for several expressive languages that we study in this article.

The needs of different graph database applications have led to a variety of navigational query languages. This includes languages for querying: (a) Graph-based object databases (e.g., GraphDB [53], GOOD [54] and G-Log [73]), (b) heterogeneous and unstructured data (e.g., Lorel [3], StruQL [43], and UnQL [26]), (c) social networks (e.g., SoQL [76], BiQL [40], and SNQL [78]), etc. Our study deals only with the most basic navigational query languages, and, in particular, those that express a common set of useful features for different graph database applications. This includes the regular path queries, as introduced above, and some languages that express the existence of patterns in a graph database satisfying a set of regular constraints (e.g., the *conjunctive* regular path queries [36, 34]). These two features, which form the core of most of the languages mentioned above, have been the topic of a vast amount of research over the last 25 years [36, 34, 69, 46, 27, 13, 16, 18].

**Main problems we study** The languages presented in the paper are studied with respect to expressiveness, complexity of query evaluation and the containment problem, as described below:

Expressiveness We study the limits of what can be expressed in a language. Obviously, there is a trade-off between the expressive power of a query language and the cost of query evaluation, as defined next.

Complexity of evaluation Given a query language $\mathcal{L}$, we study the cost of query evaluation in $\mathcal{L}$, i.e., the complexity of the problem $\mathcal{L}$-EVAL defined as follows: Given a graph database $\mathcal{G}$, a query $Q \in \mathcal{L}$, and a tuple $\bar{t}$ of objects (e.g., nodes, symbols, paths, etc) of the right type for $\mathcal{L}$, does $\bar{t}$ belong to the evaluation $[\![Q]\!]_{\mathcal{G}}$ of $Q$ on $\mathcal{G}$? The complexity is thus measured in terms of $|\mathcal{G}|$ and $|Q|$,


*PODS'13,* June 22–27, 2013, New York, New York, USA.

the lengths of reasonable encodings of $\mathcal{G}$ and $Q$, respectively. Note that the input to $\mathcal{L}$-EVAL consists of a graph database and a query, and, thus, in terms of Vardi's taxonomy, it measures the *combined complexity* of the evaluation problem [80].

In databases one is also interested in the *data complexity* of the evaluation problem [80], which is measured only in terms of the size of the graph database (i.e., the query $Q$ is assumed to be fixed). We denote by EVAL($Q$) the evaluation problem for a fixed query $Q$. If $\mathcal{C}$ is a complexity class, we say that $\mathcal{L}$-EVAL is in $\mathcal{C}$ in data complexity, if EVAL($Q$) is in $\mathcal{C}$ for each query $Q$ in $\mathcal{L}$. Moreover, $\mathcal{L}$-EVAL is $\mathcal{C}$-hard in data complexity, if there is a query $Q$ in $\mathcal{L}$ such that EVAL($Q$) is $\mathcal{C}$-hard. Finally, $\mathcal{L}$-EVAL is $\mathcal{C}$-complete in data complexity, if it is in $\mathcal{C}$ and it is $\mathcal{C}$-hard in data complexity.

Containment problem Containment is a crucial problem in query processing and optimization [2]. We study its complexity for several query languages. Formally, if $\mathcal{L}$ is a query language, the problem $\mathcal{L}$-CONT is defined as follows: Given queries $Q, Q' \in \mathcal{L}$, is $[\![Q]\!]_{\mathcal{G}} \subseteq [\![Q']\!]_{\mathcal{G}}$ for each graph database $\mathcal{G}$?

**Contents of the paper and organization** Section 2 presents the basics of graph databases. The study of query languages starts in Section 3, where we present a simple pattern language for graph databases, called $\mathbf{G}$, which was introduced in the late 80s by Cruz, Mendelzon and Wood [36]. The importance of this language is, of course, historical – it was one of the earliest graph database languages ever introduced – but also methodological – it identified for the first time a simple set of features that were common to many navigational graph query languages, and that required more detailed theoretical study. This language is also worth presentating due to its semantics based on simple paths, which has almost completely disappeared from modern query languages for graph databases due to its inherently high computational complexity.

In turn, modern navigational languages for graph databases implement a semantics based on arbitrary (as opposed to simple) paths. This semantics allows for tractable evaluation in data complexity for a family of languages that are based on $\mathbf{G}$. This includes the regular path queries (RPQs), and their extensions with conjunction (CRPQs) [36, 34], inverses (C2RPQs) [27], and unions (UC2RPQs). Some relevant restrictions of these classes, with tractable combined complexity, can also be identified. This includes the class of acyclic C2RPQs [13, 18] and the nested regular expressions [17]. We study these languages in detail in Section 4.

In Section 5 we study some recently proposed languages for graph databases – namely, the extended CRPQs, or ECRPQs [13, 12] – that treat paths as first-class citizens. In particular, ECRPQs extend CRPQs with the ability to output and compare paths. These features are motivated by some modern applications of graph databases, such as the Semantic Web, biological networks and provenance, for which the ability to verify and output complex semantic associations between nodes is crucial [8, 62, 52, 58]. Notably, the importance of including paths in the output was already foreseen by Abiteboul et al. in the 90s, and included as a feature in the language Lorel [3].

As is to be expected, the data complexity of ECRPQ evaluation depends on the language that we allow for expressing path comparisons. We study two such languages; the *regular* and the *rational* relations [41, 49, 20]. Regular relations are a simple formalism that extends the class of regular languages to relations of arbitrary arity over words, while rational relations are a powerful formalism that properly extends the regular relations. While the evaluation problem for ECRPQs with regular relations is tractable in data complexity, it becomes undecidable, or highly intractable, for queries that combine regular relations with some practically relevant rational relations such as the subword or subsequence relations. We identify, however, an important syntactic restriction of the class of ECRPQs with rational relations that is tractable in data complexity.

Most of the existing studies of navigational languages for graph databases have centered around queries that exploit the topology of the graph. On the other hand, the study of languages that combine the topology with the underlying data has been almost completely overlooked. We present some recent developments that go in the direction of overcoming this deficiency. In particular, we study the problem of querying graph databases in which each node contains a piece of data. We concentrate on *data path* queries, which check for the existence of a path whose label together with its underlying data values satisfy a given condition. It is shown that allowing queries that freely combine these two features easily leads to intractability in data complexity, but that there is a relevant class of queries – namely, the regular expressions with memory [65] – whose evaluation problem for fixed queries is tractable.

Finally, in Section 7 we present the concluding remarks and a list of big challenges for graph databases in the future.

**What is not included** The sheer volume of graph database literature, plus some space constraints, have forced us not to include several query languages we would have liked to. Notable omissions are the extensions of the query languages we study in Section 4 with aggregation [35], a family of query languages based on relation algebra that has been recently studied with respect to relative expressiveness [45], the powerfuk Walk Logic, which has been proposed as an alternative to ECRPQs for expressing path properties of graph databases [57], and the deep study of view-based query answering for views specified as (C)2RPQs [28, 29]. We expect to present each one of them in full detail in a full version of the paper.

**Proviso** We assume familiarity with regular expressions and automata. Throughout the paper we do not distinguish between a regular expression and the regular language it defines (e.g., we write $w \in R$ to express that the word $w$ belongs to the regular language defined by regular expression $R$). We also assume familiarity with usual query languages for relational databases, such as conjunctive queries (CQs), first-order logic (FO) and Datalog, and with usual complexity classes, such as NLOGSPACE, NP, PSPACE, and others.

## 2. GRAPH DATABASES

Different applications impose different constraints on its underlying graph data model, which has given rise to a myriad of different models for graph databases (see [5] for a survey). We study here the simplest possible such model: that of finite, directed and edge-labeled graphs. The reason is twofold; first, this model is flexible enough to express many interesting graph database scenarios, and, second, the most fundamental theoretical issues related to graph databases already appear in full force for it. We formalize this model below.

Let $\mathcal{V}$ be a countably infinite set of node ids. Given a finite alphabet $\Sigma$, a *graph database* $\mathcal{G}$ over $\Sigma$ is a pair $(V, E)$, where $V$ is a finite set of node ids (i.e., $V \subseteq \mathcal{V}$) and $E \subseteq V \times \Sigma \times V$. Thus, each edge in $\mathcal{G}$ is a triple $(v, a, v') \in V \times \Sigma \times V$, whose interpretation is an $a$-labeled edge from $v$ to $v'$ in $\mathcal{G}$. When $\Sigma$ is clear from the context, we shall speak simply of a graph database.

Notice that each graph database $\mathcal{G} = (V, E)$ over $\Sigma$ can be naturally seen as a non-deterministic finite automaton (NFA) over the alphabet $\Sigma$, without initial and final states. Its states are the nodes in $V$ and its transitions are the edges in $E$.

A *path* in a graph database $\mathcal{G} = (V, E)$ is a sequence

$$\rho \;=\; v_0 a_0 v_1 a_1 v_2 \cdots v_{k-1} a_{k-1} v_k, \quad (k \geq 0),$$

such that $(v_{i-1}, a_{i-1}, v_i) \in E$, for each $i$ with $1 \leq i \leq k$. The *label* of $\rho$, denoted $\lambda(\rho)$, is the string $a_0 a_1 \cdots a_{k-1}$ in $\Sigma^*$. Notice that the definition of a path includes the *empty* path $v$, for each $v \in V$. The label of such path is the empty string $\epsilon$. The path $\rho$ is *simple* if it does not go through the same node twice, i.e., $v_i \neq v_j$ for each $i, j$ with $0 \leq i < j \leq k$.

# 3. THE LANGUAGE G AND THE SIMPLE PATH SEMANTICS

Since the early days, graph databases and query languages were not thought as competitors of their relational counterparts, but as suitable complements for data that could be more naturally represented in the form of a graph (in particular, semistructured data). In such context, Cruz, Mendelzon and Wood designed in 1987 one of the earliest (and, also, most influential) navigational languages for edge-labeled graph-structured data, that was simply called **G** [36].

We present the syntax and semantics of **G** in the context of the graph data model studied here. We leave out some syntactic sugar and intricacies in the definition of the semantics that complicate the presentation without being essential to the query language.

The language **G** uses a semantics based on simple paths, motivated by early applications of graph databases for which simple paths were more meaningful than arbitrary ones. As we see in the present section, this causes intractability of query evaluation for the language even in data complexity.

**Syntax** Queries in **G** are unions of *graph patterns*, which are graph databases extended with three new features: (1) *node variables*, (2) *label variables*, and (3) regular expressions as labels for edges. Thus, a graph pattern over $\Sigma$ is a graph database in which each node is either a node id in $\mathcal{V}$ or a node variable, and each edge is labeled with a regular expression over the alphabet that extends $\Sigma$ with all label variables. We formalize this below.

Let $\mathcal{V}_{\text{node}}$ and $\mathcal{V}_{\text{label}}$ be two disjoint and countably infinite sets of node and label variables, respectively, which are pairwise disjoint from the set $\mathcal{V}$ of node ids. Given a (not necessarily finite) alphabet $\Gamma$, we denote by $\text{REG}(\Gamma)$ the set of regular expressions over $\Gamma$. A graph pattern $\pi$ over the finite alphabet $\Sigma$ is a pair $(N, A)$, where $N$ is a finite set of nodes ids and node variables (that is, $N \subseteq \mathcal{V} \cup \mathcal{V}_{\text{node}}$), and $A \subseteq V \times \text{REG}(\Sigma \cup \mathcal{V}_{\text{label}}) \times V$ is the set of edges labeled with regular expressions over the alphabet that extends $\Sigma$ with the label variables in $\mathcal{V}_{\text{label}}$.

In order to view a graph pattern as a query, it is necessary to specify which variables in the graph pattern are projected in the output. As such, we assume that each graph pattern $\pi = (N, A)$ has an associated pair $(\bar{x}, \bar{X})$ of *free* variables, where $\bar{x}$ is an ordered tuple of node variables in $N$ and $\bar{X}$ is an ordered tuple of label variables occuring in $A$. We write $\pi(\bar{x}, \bar{X})$ to specify that $(\bar{x}, \bar{X})$ are the free variables of $\pi$. If both $\bar{x}$ and $\bar{X}$ are the empty tuple, we follow usual terminology and say that $\pi$ is Boolean.

DEFINITION 1 (**G** LANGUAGE [36]). *A query $\Pi$ of the language **G** over $\Sigma$ is an expression of the form $\bigcup_{1 \leq i \leq k} \pi_i$ ($k \geq 1$), where each $\pi_i$ ($1 \leq i \leq k$) is a graph pattern over $\Sigma$.*

**Semantics** The semantics of **G** queries depends on the semantics of graph patterns, which is, in turn, based on a refined notion of *homeomorphism*. These are mappings that match node variables into node ids in $V$, label variables into elements of $\Sigma$, and each edge labeled by regular expression $R$ into a *simple* path in the graph database that is labeled with a word in $R$.

Since variables appear both in nodes and edges of patterns, it will be convenient to define homeomorphisms as pairs of mappings. Formally, given a graph pattern $\pi = (N, A)$ and a graph database $\mathcal{G} = (V, E)$, both over $\Sigma$, a *homeomorphism* from $\pi$ to $\mathcal{G}$ is a pair of mappings $(\nu_1, \nu_2)$, such that:

- $\nu_1$ maps nodes in $\pi$ to node ids in $V$, and $\nu_2$ maps label variables used in $A$ to symbols in $\Sigma$; that is, $\nu_1 : N \to V$, and if $\mathcal{X}$ is the set of label variables occuring in $A$, then $\nu_2 : \mathcal{X} \to \Sigma$;
- $\nu_1$ is the identitity on node ids; that is, $\nu_1(v) = v$, for each node id $v$ in $N$ (i.e., for each $v \in N \cap \mathcal{V}$);[1]
- each edge labeled by a regular expression $R$ is mapped to a simple path labeled by a word in $R$, after label variables are replaced according to $\nu_2$.

  Formally, for each $(p, L, q) \in A$, where $p, q \in N$ and $L \in \text{REG}(\Sigma \cup \mathcal{V}_{\text{label}})$, there is a simple path $\rho$ from $\nu_1(p)$ to $\nu_1(q)$ in $\mathcal{G}$ such that $\lambda(\rho) \in \nu_2(L)$, where $\nu_2(L)$ is the regular language over $\Sigma$ that is obtained from $L$ by replacing each occurence of a symbol $X \in \mathcal{X}$ with $\nu_2(X)$.

The evaluation $[\![\pi]\!]^{\mathbf{s}}_{\mathcal{G}}$ of a graph pattern $\pi(\bar{x}, \bar{X})$ on $\mathcal{G}$ is the set of ordered tuples of the form $(\nu_1(\bar{x}), \nu_2(\bar{X}))$, for each homeomorphism $(\nu_1, \nu_2)$ from $\pi$ to $G$. We use the superscript $\mathbf{s}$ in $[\![\pi]\!]^{\mathbf{s}}_{\mathcal{G}}$ to stress the fact that we are using a semantics based on simple paths (which is not the case for the rest of the languages we study in the paper). If $\pi$ is Boolean, i.e., $\bar{x}$ and $\bar{X}$ are the empty tuple, then the answer *true* is, as usual, modeled by the set containing the empty tuple, and the answer *false* by the empty set.

Finally, if $\Pi = \bigcup_{1 \leq i \leq k} \pi_i$ is a **G** query, then $[\![\Pi]\!]^{\mathbf{s}}_{\mathcal{G}} = \bigcup_{1 \leq i \leq k} [\![\pi_i]\!]^{\mathbf{s}}_{\mathcal{G}}$, for each graph database $\mathcal{G} = (V, E)$.

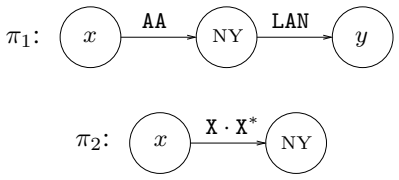


**Figure 1: Patterns $\pi_1$ and $\pi_2$ used in Example 1.**

EXAMPLE 1. We present a toy example based on ideas from [36]. Let $\Sigma$ be the set of airlines and $\mathcal{G} = (V, E)$ be the graph database over $\Sigma$ such that the node ids in $V$ represent cities and there is an edge in $E$ from city $c$ to city $c'$ labeled with airline `A` iff `A` has a direct flight from $c$ to $c'$.

Figure 1 shows the graphical depiction of a **G** query $\Pi$ composed by the union of two patterns $\pi_1(x, y)$ and $\pi_2(x, X)$. We assume all node and label variables to be free. Then $[\![\pi_1]\!]^{\mathbf{s}}_{\mathcal{G}}$ is the set of pairs $(c, c')$ of cities such that there is a flight from $c$ to NY with `AA` and a flight from NY to $c'$ with `LAN`. On the other hand, $[\![\pi_2]\!]^{\mathbf{s}}_{\mathcal{G}}$ is the set of tuples of the form $(c, \mathtt{A})$, where $c$ is a city and `A` is an airline, such that there is a (nonempty) sequence of flights from $c$ to NY with the same airline `A`. Finally, $[\![\Pi]\!]^{\mathbf{s}}_{\mathcal{G}} = [\![\pi_1]\!]^{\mathbf{s}}_{\mathcal{G}} \cup [\![\pi_2]\!]^{\mathbf{s}}_{\mathcal{G}}$. □

The previous example illustrates why the simple path semantics might be useful in some cases. In fact, if we are looking for flights connecting two cities under certain regular conditions, we are only interested in those that do not stop twice in the same city. As we will see next, this choice has an important cost in the complexity of query evaluation.

[1] In the original definition of the semantics of **G**, the mapping $\nu_1$ is also forced to be a 1-1 mapping; i.e., different node variables have to be mapped to different node ids. We have relaxed this condition to allow a uniform treatment with other query languages presented in the article. This modification is inessential to the complexity of query evaluation for the language.

## 3.1 Complexity of evaluation of G queries

The evaluation problem $\mathbf{G}$-EVAL is as follows: Given a query $\Pi$ in $\mathbf{G}$, a graph database $\mathcal{G} = (V, E)$, a tuple $\bar{v}$ of node ids in $V$ and a tuple $\bar{A}$ of symbols in $\Sigma$, determine whether $(\bar{v}, \bar{A}) \in [\![\Pi]\!]^{\mathbf{s}}_{\mathcal{G}}$. It is not hard to see that this problem can be solved in NP.

PROPOSITION 1. $\mathbf{G}$-EVAL *is in* NP.

The intuition behind this fact is as follows. In order to check whether $(\bar{v}, \bar{A}) \in [\![\Pi]\!]^{\mathbf{s}}_{\mathcal{G}}$, we only have to guess a graph pattern $\pi(\bar{x}, \bar{X})$ in $\Pi$ and a homeomorphism $(\nu_1, \nu_2)$ from $\pi$ to $\mathcal{G}$ such that $\nu_1(\bar{x}) = \bar{v}$ and $\nu_2(\bar{X}) = \bar{A}$. To verify that $(\nu_1, \nu_2)$ is indeed a homeomorphism, we have to check that for each edge $(p, L, q)$ there is a simple path in $\mathcal{G}$ from $\nu_1(p)$ to $\nu_1(q)$ whose label belongs to $\nu_2(L)$. But the length of each simple path in $\mathcal{G}$ is bounded by $|V|$, and hence this fact admits a polynomial size witness.

We show next that the query evaluation problem for $\mathbf{G}$ is NP-complete. Surprisingly, this holds already in data complexity for a very simple class of queries.

**Finding regular simple paths** In order to understand the precise complexity of query evaluation for $\mathbf{G}$, and, in particular, the impact of the simple path semantics, Mendelzon and Wood decided to study the evaluation problem for a simple class of $\mathbf{G}$ queries, often known as *regular path queries* (RPQs) [69]. These are graph patterns that consist of a single edge of the form $(x, L, y)$, where $x$ and $y$ are differerent node variables and $L$ is a regular expression over $\Sigma$ (i.e., $L$ contains no label variables). Thus, the query evaluation problem for RPQs boils down to the problem of finding regular simple paths in a graph database, as follows:

| | |
|---|---|
| PROBLEM: | REGULARSIMPLEPATH |
| INPUT: | A graph database $\mathcal{G}$, nodes $v, v'$ in $\mathcal{G}$, a regular expression $L$. |
| QUESTION: | Is there a simple path $\rho$ from $v$ to $v'$ in $\mathcal{G}$ such that $\lambda(\rho) \in L$? |

When the input consists of $\mathcal{G}$ and $v, v'$ only (i.e., when $L$ is fixed) we refer to this problem as FIXEDREGULARSIMPLEPATH($L$). We state below that for a big class of regular expressions this problem is NP-complete, which immediately implies that $\mathbf{G}$-EVAL is NP-complete in data complexity.

THEOREM 1 ([69]). *Let $\Sigma$ be a finite alphabet and $w$ be a string in $\Sigma^*$ of length at least 2. Then* FIXEDREGULARSIMPLEPATH($w^*$) *is* NP*-complete.*

Consider, for instance, the simplest possible case $\Sigma = \{0\}$ and $w = 00$. Then FIXEDREGULARSIMPLEPATH($(00)^*$) checks whether there is a simple path of even length from node $v$ to $v'$ in a directed graph $\mathcal{G}$, which is an NP-complete problem [64].

The problem FIXEDREGULARSIMPLEPATH($L$) is also NP-complete for some simple languages that do not follow the pattern in Theorem 1, such as $L = 0^*10^*$ [69]. On the other hand, FIXEDREGULARSIMPLEPATH($L$) can be solved in PTIME for each $L$ that defines a finite language (e.g., each $L$ that does not make use of the Kleene-star $*$). However, this restriction does not lead to tractability in combined complexity:

PROPOSITION 2. REGULARSIMPLEPATH *is* NP*-complete, even if restricted to the class of regular expressions that do not use the Kleene-star.*

Tractable cases of the problem REGULARSIMPLEPATH can be found by restricting the shape of graph databases (e.g., DAGs) or the class of regular languages allowed in RPQs (e.g., closed under removal of symbols from a word) [69].

**Final remarks** The intractability of $\mathbf{G}$ in data complexity is bad news, as it implies that the language is impractical. In order to cope with this problem, the graph database community has opted for a different semantics, based on arbitrary paths, that allows to evaluate full-fledged recursive queries with low data complexity.

In the last few years we have witnessed a revival of the simple path semantics, due to its application in early versions of SPARQL 1.1 [56], the standard for navigational querying of the Semantic Web data model RDF. It has been shown in [11, 67] that this easily leads to intractability in data complexity.

# 4. BASIC LANGUAGES UNDER THE ARBITRARY PATH SEMANTICS

In this section, we study the basics of modern navigational languages for graph databases. Its building blocks are clearly rooted on $\mathbf{G}$, but there are three important differences:

1. First, the semantics of current query languages for graph databases is based on arbitrary (instead of simple) paths. The choice of the new semantics is justified by two facts: (a) It leads to tractable combined complexity for RPQs and tractable data complexity for a family of expressive languages. (b) Several graph-based applications only care about connectivity of the data under regular constraints, and, thus, simple paths do not seem to be essential.

2. Second, label variables have mostly disappeared from modern graph query languages. We have not found an explanation for this, but provide here a plausible one: Although label variables do not increase data complexity for usual query languages, its inclusion in even the most basic language (RPQs) leads to intractability in combined complexity. As we argue in Section 4.3, this complicates the evaluation of the most basic queries with variables over modern applications of graph databases that store massive amounts of data.

3. Third, current query languages allow to traverse edges in the graph database in both directions. This allows an important increase in expressive power without having an impact on complexity.

For simplicity of presentation, we leave out constants (node ids) from queries, but they can be easily incorporated at no computational cost.

## 4.1 Regular path queries

RPQs (i.e., queries of the form $(x, L, y)$, for $L$ a regular language) are the basic navigational mechanism for graph databases. From now on, we use the shorthand $L$ for the RPQ $(x, L, y)$. Under the semantics we study in this section, the evaluation $[\![L]\!]_{\mathcal{G}}$ of the RPQ $L$ on a graph database $\mathcal{G} = (V, E)$ consists of the set of pairs $(v, v')$ of node ids in $V$ such that there is a (not necessarily simple) path $\rho$ in $\mathcal{G}$ from $v$ to $v'$ whose label $\lambda(\rho)$ belongs to $L$. Notice that we have removed the superscript $\mathbf{s}$ from $[\![L]\!]_{\mathcal{G}}$, as we are no longer using a simple paths semantics.

We show in this section that RPQs (and even its extension with inverses) can be evaluated efficiently under the arbitrary path semantics. However, this is no longer the case if RPQs are extended with label variables.

**RPQs with inverse** RPQs are often extended to traverse edges in both directions (a feature that was absent in $\mathbf{G}$). This defines the

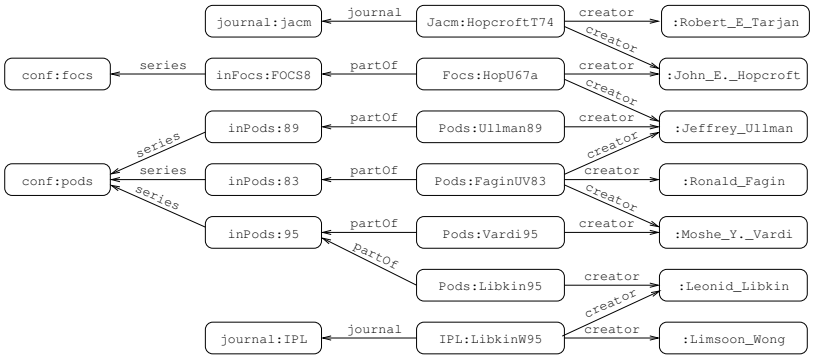


**Figure 2: An abstraction of a fragment of the RDF representation of DBLP available at `http://dblp.l3s.de/d2r/`**

notion of RPQs with *inverse*, or 2RPQs [27, 28]. A 2RPQ over $\Sigma$ is an RPQ over the alphabet $\Sigma^{\pm}$ which extends $\Sigma$ with the symbol $a^-$, for each $a \in \Sigma$. While evaluation of 2RPQs can be easily reduced to evaluation of RPQs by extending the underlying graph database with inverses of edges (something that we properly define in the next paragraph), the presence of inverses complicates the static analysis of queries, e.g., with respect to containment [30, 18].

To define the semantics of 2RPQs we use the notion of the *completion* of a graph database $\mathcal{G}$, denoted by $\mathcal{G}^{\pm}$. This is the graph database over $\Sigma^{\pm}$ that is obtained from $\mathcal{G} = (V, E)$ by adding the edge $(u, a^-, v)$, for each $(v, a, u) \in E$. We define the evaluation $[\![L]\!]_{\mathcal{G}}$ of the 2RPQ $L$ over $\mathcal{G}$ to be $[\![L]\!]_{\mathcal{G}^{\pm}}$ (the latter is well-defined since $L$ is an RPQ over $\Sigma^{\pm}$). Notice that the 2RPQ $a^-$ defines on a graph database $\mathcal{G} = (V, E)$ precisely the inverse of the RPQ $a$; that is, $[\![a^-]\!]_{\mathcal{G}} = \{(u, v) \mid (v, a, u) \in E\}$.

EXAMPLE 2. Let $\mathcal{G}$ be the graph database over $\Sigma = \{\texttt{creator}, \texttt{partOf}, \texttt{series}\}$ in Figure 2. This graph contains an abstraction of a fragment of the *RDF Linked Data* representation of DBLP [37] (and it is based on an example by Arenas and Pérez in an earlier PODS tutorial [10]). The following is a simple 2RPQ that matches all pairs $(x, y)$ such that $x$ is an author that published a paper in conference $y$:

$$L = \texttt{creator}^- \cdot \texttt{partOf} \cdot \texttt{series}$$

For example, the pairs $(\texttt{:Jeffrey\_D.\_Ullman}, \texttt{conf:focs})$ and $(\texttt{:Ronald\_Fagin}, \texttt{conf:pods})$ are in $[\![L]\!]_{\mathcal{G}}$. We can conclude that 2RPQs are more expressive than RPQs: For no RPQ $L'$ over $\Sigma$ it is the case that $[\![L]\!]_{\mathcal{G}} = [\![L']\!]_{\mathcal{G}}$. □

2RPQs have particularly good properties: They can be evaluated linearly in both the size of the data and the expression.

THEOREM 2 (SEE E.G., [69]). 2RPQ-EVAL *can be solved in time* $O(|\mathcal{G}| \cdot |L|)$*, for* $\mathcal{G}$ *a graph database and* $L$ *a* 2RPQ*.*

*Proof (Sketch):* We check whether $(u, v) \in [\![L]\!]_{\mathcal{G}}$, for a pair $(u, v)$ of node ids in $\mathcal{G}$, as follows. First, we compute $\mathcal{G}^{\pm}$ from $\mathcal{G}$ in time $O(|\mathcal{G}|)$, and then an NFA $\mathcal{A}_L$ that defines the same language than $L$ in time $O(|L|)$. Let $\mathcal{G}^{\pm}(u, v)$ be the NFA that is obtained from $\mathcal{G}^{\pm}$ by setting its initial and final states to be $u$ and $v$, respectively. Clearly, $(u, v) \in [\![L]\!]_{\mathcal{G}}$ iff $(u, v) \in [\![L]\!]_{\mathcal{G}^{\pm}}$ iff there is a word accepted by both $\mathcal{G}^{\pm}(u, v)$ and $\mathcal{A}_L$. The latter is equivalent to checking the product of $\mathcal{G}^{\pm}(u, v)$ and $\mathcal{A}_L$ for nonemptiness, which can be done in time $O(|\mathcal{G}^{\pm}| \cdot |\mathcal{A}_L|)$. The whole process takes time $O(|\mathcal{G}| + |L| + |\mathcal{G}^{\pm}| \cdot |\mathcal{A}_L|)$, that is, $O(|\mathcal{G}| \cdot |L|)$. □

It is worth contrasting Theorem 2 with Theorem 1, that states that RPQ-EVAL under the simple path semantics is intractable in data complexity. The difference is that checking the existence of an arbitrary path satisfying a regular condition can be efficiently reduced to a nonemptiness automata problem (see the proof of Theorem 2), but this is unlikely to be the case for simple paths.

In terms of data complexity, 2RPQs are in NLOGSPACE. Not only that, the whole set $[\![L]\!]_{\mathcal{G}}$ can be computed in NLOGSPACE for each fixed 2RPQ $L$:

PROPOSITION 3 (SEE E.G., [34]). *Let* $L$ *be a fixed* 2RPQ*. There is an* NLOGSPACE *procedure that computes* $[\![L]\!]_{\mathcal{G}}$ *for each graph database* $\mathcal{G}$*.*

*Proof (Sketch):* For each pair $(u, v)$ of node ids in $V$, we check whether it belongs to $[\![L]\!]_{\mathcal{G}}$ by following the proof of Proposition 2. Clearly, $\mathcal{G}^{\pm}$ can be constructed in LOGSPACE from $\mathcal{G}$, and $\mathcal{A}_L$ in constant time from $L$ (since $L$ is fixed). Nonemptiness of the product of $\mathcal{G}^{\pm}(u, v)$ and $\mathcal{A}_L$ can be checked in NLOGSPACE in $|\mathcal{G}|$ using a standard "on-the-fly" algorithm. The whole process can be carried out in NLOGSPACE (because NLOGSPACE computable functions are closed under composition). We conclude that $[\![L]\!]_{\mathcal{G}}$ can be computed in NLOGSPACE for each fixed 2RPQ $L$. □

It is easy to see, on the other hand, that RPQs are NLOGSPACE-complete in data complexity (under LOGSPACE reductions): If $\Sigma = \{0\}$ the RPQ $0^*$ checks, for each pair $(u, v)$ of node ids in $\mathcal{G}$, if there is a directed path from $u$ to $v$ in $\mathcal{G}$, which is a well-known NLOGSPACE-complete problem.

**RPQs with label variables** As we mentioned earlier, label variables have almost disappeared from modern graph query languages. We provide a partial explanation to this fact by stating that, as opposed to the case of RPQs, evaluation of RPQVs is intractable.

RPQs *with label variables* (RPQVs) are of the form $R$, for $R \in \mathrm{REG}(\Sigma \cup \mathcal{V}_{\mathrm{label}})$. Assume $\mathcal{X}$ is the set of label variables mentioned in $R$. The evaluation $[\![R]\!]_{\mathcal{G}}$ of $R$ on $\mathcal{G} = (V, E)$ is the set of pairs $(u, v)$ of node ids in $V$ such that there is a mapping $\nu : \mathcal{X} \to \Sigma$ and a path $\rho$ in $\mathcal{G}$ from $u$ to $v$ that satisfies that $\lambda(\rho) \in \nu(R)$.

THEOREM 3 ([15]). RPQV-EVAL *is* NP*-complete.*

## 4.2 Conjunctive 2RPQs

Recall that patterns in $\mathbf{G}$ are graph databases such that each one of its edges is an RPQ and some variables are allowed to be projected in the output. In other words, patterns in $\mathbf{G}$ represent the closure of RPQs under joins and existential quantification. In modern

graph query languages, this idea gives rise to the class of *conjunctive* 2RPQs, or C2RPQs [34, 1, 27]. We state in this section that C2RPQs (and even its extension with unions) preserve tractability in data complexity, but their combined complexity is NP-complete.

Let $\bar{x} = \{x_1, \ldots, x_n\}$ and $\bar{y} = \{y_1, \ldots, y_m\}$ be (possibly empty) disjoint sets of node variables. A C2RPQ, with free variables $\bar{x}$, over $\Sigma$ is a rule $\varphi(\bar{x})$ of the form

$$Ans(\bar{x}) \leftarrow \bigwedge_{1 \leq i \leq k} (z_i, L_i, z'_i) \quad (1)$$

where (a) $L_i$ is a 2RPQ over $\Sigma$, for every $i$ with $1 \leq i \leq k$, (b) $z_1, z'_1, \ldots z_k, z'_k$ are (not necessarily distinct) variables, and (c) $\{z_1, z'_1, \ldots, z_k, z'_k\} = \bar{x} \cup \bar{y}$. This C2RPQ is Boolean if $\bar{x} = \emptyset$. A CRPQ is a C2RPQ of the form (1), in which each $L_i$ is an RPQ.

The semantics of C2RPQs is defined in terms of *homomorphisms*. Given a graph database $\mathcal{G} = (V, E)$, a homomorphism from a C2RPQ $\varphi(\bar{x})$ of the form (1) to $\mathcal{G}$ is a mapping $h : \bar{x} \cup \bar{y} \to V$ such that $(h(z_i), h(z'_i)) \in [\![L_i]\!]_{\mathcal{G}}$, for each $1 \leq i \leq k$. Homomorphisms are not to be confused with the homeomorphisms introduced in Section 3, which define a simple path semantics.

The evaluation $[\![\varphi(\bar{x})]\!]_{\mathcal{G}}$ of $\varphi(\bar{x})$ on $\mathcal{G}$ is defined as the set of tuples of the form $h(\bar{x})$, for each homomorphism $h$ from $\varphi(\bar{x})$ to $\mathcal{G}$. Boolean C2RPQs evaluate to *true* and *false*, as defined for $\mathbf{G}$.

EXAMPLE 3. Consider the C2RPQ $\varphi(x, y)$:

$$Ans(x, y) \leftarrow (x, \texttt{creator}^-, u) \wedge (u, \texttt{partOf}, v) \wedge (v, \texttt{series}, w) \wedge (u, \texttt{creator}, y).$$

Its evaluation $[\![\varphi(x, y)]\!]_{\mathcal{G}}$ over the graph database $\mathcal{G}$ shown in Figure 2 consists of the pairs $(x, y)$ of (not necessarily distinct) authors that have a joint conference paper, e.g., $(\texttt{:Jeffrey\_D.\_Ullman}, \texttt{:Ronald\_Fagin})$ and $(\texttt{:Ronald\_Fagin}, \texttt{:Moshe\_Y.\_Vardi})$ are in $[\![\varphi(x, y)]\!]_{\mathcal{G}}$. It is easy to see that this query cannot be expressed as a 2RPQ over $\mathcal{G}$, that is, for every 2RPQ $L$ over $\Sigma$ it is the case that $[\![L]\!]_{\mathcal{G}} \neq [\![\varphi(x, y)]\!]_{\mathcal{G}}$. □

**Unions of C2RPQs** The language $\mathbf{G}$ is closed under unions. Closing C2RPQs under unions gives rise to the class of UC2RPQs, which are formulas $\psi(\bar{x})$ of the form $\bigcup_{1 \leq i \leq k} \varphi_i(\bar{x})$, where $\varphi_i(\bar{x})$ is a C2RPQ for each $i$ with $1 \leq i \leq k$. We have that $[\![\psi(\bar{x})]\!]_{\mathcal{G}} = \bigcup_{1 \leq i \leq k} [\![\varphi_i(\bar{x})]\!]_{\mathcal{G}}$, for each graph database $\mathcal{G}$.

Next proposition shows that the increase in expressiveness from 2RPQs to UC2RPQs has an important cost in the complexity of evaluation (even for CRPQs). Still, evaluating UC2RPQs is not more costly than evaluating CQs over relational databases; namely, NP-complete [32].

PROPOSITION 4 (SEE E.G., [13]). UC2RPQ-EVAL *is* NP*-complete, even if restricted to Boolean* CRPQs*.*

On the other hand, moving from RPQs to UC2RPQs is free in data complexity.

PROPOSITION 5 (SEE E.G., [34]). *The problem* UC2RPQ-EVAL *is in* NLOGSPACE *in data complexity.*

*Proof (Sketch):* It is sufficient to show that each fixed C2RPQ $\varphi$ of the form (1) can be evaluated in NLOGSPACE. Let $\mathcal{G} = (V, E)$ be a graph database and $\bar{v}$ a tuple of node ids in $V$. To check whether $\bar{v} \in [\![\varphi]\!]_{\mathcal{G}}$, we compute the relational database $\mathcal{D}$ that contains all binary relations $R_i := [\![L_i]\!]_{\mathcal{G}}$, for $1 \leq i \leq k$. This can be done in NLOGSPACE using Proposition 3 and the fact that $\varphi$ is fixed. Clearly, $\bar{v} \in [\![\varphi]\!]_{\mathcal{G}}$ iff $\bar{v}$ belongs to the evaluation of the CQ $\exists \bar{y} \bigwedge_{1 \leq i \leq k} R_i(z_i, z'_i)$ over $\mathcal{D}$. But since evaluation of CQs is in LOGSPACE in data complexity [2], the whole process can be carried out in NLOGSPACE. □

Again, it is worth contrasting this result with the intractability of $\mathbf{G}$ in data complexity.

**Expressive power** UC2RPQs can be evaluated in NLOGSPACE in data complexity, but not every NLOGSPACE property can be expressed as a UC2RPQ. This is easy to see: While UC2RPQs are *monotone* (that is, $[\![Q]\!]_{\mathcal{G}} \subseteq [\![Q]\!]_{\mathcal{G}'}$ for every UC2RPQ $Q$ and graph databases $\mathcal{G}, \mathcal{G}'$ such that $\mathcal{G}'$ extends $\mathcal{G}$ with new nodes and edges), there are NLOGSPACE properties that are not monotone (e.g., the property that the number of nodes in the graph database is even).

More interestingly, there are some simple monotone NLOGSPACE computable queries that are not expressible as UC2RPQs. Consider again the C2RPQ $\varphi(x, y)$ in Example 3. Imagine that we want to express the transitive closure $\varphi(x, y)^+$ of $\varphi(x, y)$; in particular, over the graph database $\mathcal{G}$ in Figure 2 the query $\varphi(x, y)^+$ defines the set of pairs $(x, y)$ of authors that are linked by a *conference coauthorship sequence*. The syntax of UC2RPQs does not allow to express $\varphi(x, y)^+$ directly, and, even more, it can be proved (e.g., using techniques in [17]) that this query cannot be expressed as a UC2RPQ. This means that there is no UC2RPQ $\varphi'(x, y)$ such that $[\![\varphi'(x, y)]\!]_{\mathcal{G}} = [\![\varphi(x, y)^+]\!]_{\mathcal{G}}$, for each graph database $\mathcal{G}$ over $\Sigma = \{\texttt{creator}, \texttt{partOf}, \texttt{series}\}$. Notice that $\varphi(x, y)^+$ is a *monotone* query and EVAL$(\varphi(x, y)^+)$ can be computed in NLOGSPACE. We thus obtain the following:

PROPOSITION 6 ([17]). *There exists a monotone,* NLOGSPACE *computable query that is not expressible as a* UC2RPQ*.*

To overcome this lack of expressiveness, Consens and Mendelzon proposed a language, called GraphLog [34], that can be seen as a natural extension of the class of UC2RPQs that expresses all NLOGSPACE properties. Intuitively, GraphLog queries are recursive sets of UC2RPQs rules with distinguished outputs – in the style of a Datalog program – in which the body of a rule is allowed to use the head (output) $V(\bar{x})$ of another rule, its negation $\neg V(\bar{x})$, its transitive closure $V(\bar{x})^+$, and the negation of $V(\bar{x})^+$, with the expected semantics. For instance, the following simple GraphLog query defines the conference coauthorship sequence $\varphi(x, y)^+$ presented earlier: $\{V(x) \leftarrow \phi(x, y),\ Ans(x, y) \leftarrow V(x)^+\}$.

## 4.3 Low Complexity Queries

UC2RPQs are tractable in data complexity. However, this does not seem the right measure of complexity for several modern data-centric applications that store massive amounts of information. For instance, evaluation of a CRPQ $Q$ over a graph database $\mathcal{G}$ is of the order $|\mathcal{G}|^{O(|Q|)}$ [81]. Although this is polynomial in data complexity, it is clearly infeasible for big $\mathcal{G}$ even if $Q$ is small. In this scenario, we thus require query languages that are tractable in combined complexity, or, at least, *fixed-parameter tractable* [72]. Recall that the latter means that there exists a computable function $f : \mathbb{N} \to \mathbb{N}$, and a constant $c \geq 0$, such that the evaluation of each query $Q$ in the language over a graph database $\mathcal{G}$ can be solved in time $O(|\mathcal{G}|^c \cdot f(|Q|))$.

The only query language we have seen with tractable combined complexity is the class of 2RPQs (in fact, Theorem 2 states that each 2RPQ $L$ can be evaluated in linear time $O(|\mathcal{G}| \cdot |L|)$ over a graph database $\mathcal{G}$). The mildest extension of RPQs we know, the CRPQs, does not preserve this good property: Proposition 4

states that CRPQ-EVAL is NP-complete, and, even worst, under widely-held complexity theoretical assumptions, CRPQs are not fixed-parameter tractable [72]. We study below two ways in which the class of 2RPQs can be extended by preserving tractable combined complexity: By restricting the syntactic shape of C2RPQs, and by extending the navigational features of 2RPQs.

**Acyclic C2RPQs** In the case of CQs over relational databases, that are also NP-complete in combined complexity, several syntactic restrictions have been identified that lead to tractable evaluation. One of the oldest and most common such restriction is *acyclicity*. It was proved by Yannakakis that an acyclic CQ $Q$ can be evaluated over a relational database $\mathcal{D}$ in linear time $O(|\mathcal{D}| \cdot |Q|)$ [83].

The acyclicity condition can also be applied to find tractable cases of C2RPQ-EVAL [13]. Acyclicity of a C2RPQ $\phi = Ans(\bar{x}) \leftarrow \bigwedge_{1 \leq i \leq n}(z_i, L_i, z_i')$ is often defined in terms of the acyclicity of its *underlying CQ* [13, 18], but a simpler and equivalent definition can be provided in terms of the *underlying graph of* $\phi$: This is the graph $U_\phi$ whose nodes are the variables of $\phi$ and its set of edges is $\{\{z_i, z_i'\} \mid 1 \leq i \leq n,\ z_i \neq z_i'\}$. Notice that $U_\phi$ is simple (it contains neither loops nor multiedges) and undirected. The C2RPQ $\phi$ is acyclic if $U_\phi$ is acyclic. We denote by AC2RPQ the class of acyclic C2RPQs. It is worth noticing that acyclicity allows for cycles of length at most 2 in C2RPQs; e.g., the C2RPQ $\phi = Ans() \leftarrow (x, a, x) \wedge (x, a, y) \wedge (y, b, x)$ is acyclic (since $U_\phi$ consists of a unique edge linking the variables $x$ and $y$).

THEOREM 4. AC2RPQ-EVAL *can be solved in time* $O(|\mathcal{G}|^2 \cdot |\phi|^2)$*, for each graph database* $\mathcal{G}$ *and acyclic* C2RPQ $\phi$*.*

*Proof (Sketch):* Let $\mathcal{G} = (V, E)$ be a graph database and $\phi(\bar{x}) = Ans(\bar{x}) \leftarrow \bigwedge_{1 \leq i \leq n}(z_i, L_i, z_i')$ an acyclic C2RPQ. To check whether $\bar{v} \in [\![\phi]\!]_\mathcal{G}$, for $\bar{v}$ a tuple of node ids in $V$, we first compute the relational database $\mathcal{D}$ that contains all binary relations $R_i := [\![L_i]\!]_\mathcal{G}$, for $1 \leq i \leq n$. Each $R_i$ can be computed in time $O(|\mathcal{G}|^2 \cdot |L_i|)$ using Proposition 3, and, thus, $\mathcal{D}$ can be computed in time $O(|\mathcal{G}|^2 \cdot |\phi|)$. Clearly, $\bar{v} \in [\![\phi]\!]_\mathcal{G}$ if and only if $\bar{v}$ belongs to the evaluation of the CQ $Q = \exists \bar{y} \bigwedge_{1 \leq i \leq n} R_i(z_i, z_i')$ over $\mathcal{D}$, where $\bar{y}$ is the tuple of variables in $\phi$ that are not mentioned in $\bar{x}$. It can be proved that $Q$ is an acyclic CQ (since $\phi$ is acyclic), and hence it can be evaluated over $\mathcal{D}$ in time $O(|\mathcal{D}| \cdot |Q|)$. The whole process takes time $O(|\mathcal{G}|^2 \cdot |\phi|) + O(|\mathcal{D}| \cdot |Q|)$, and, thus, $O(|\mathcal{G}|^2 \cdot |\phi|^2)$ because $|Q|$ is $O(|\phi|)$. □

In terms of expressive power, the class of AC2RPQs lies strictly in between 2RPQs and C2RPQs: The C2RPQ $Ans(x, y) \leftarrow (x, \texttt{creator}^-, u) \wedge (u, \texttt{partOf}, v) \wedge (v, \texttt{series}, w) \wedge (u, \texttt{creator}, y)$, presented in Example 3, is acyclic but it is not expressible as a 2RPQ. On the other hand, a simple argument proves that the CRPQ $Ans(x, y, z) \leftarrow (x, a, y), (y, b, z), (z, c, x)$, over the alphabet $\Sigma = \{a, b, c\}$, is not expressible as an AC2RPQ.

The evaluation of AC2RPQs is quadratic in the size of the data, which might not be ideal for very big graph databases. The question of whether acyclic C2RPQs (or even acyclic CRPQs) can be evaluated linearly in the size of the data is related to some important open problems in the area of algorithmic graph theory. For instance, assume that $\Sigma = \{a\}$, then the acyclic CRPQ $Ans() \leftarrow (x, aaa, x)$ checks the existence of a *triangle* (i.e., a cycle of length 3) in an undirected simple graph, a problem for which no linear algorithm is known to date [6].

It is possible to obtain a class of AC2RPQs with linear time evaluation by further restricting its syntax. The only thing that we need to do is disallow loops and multiedges. Formally, let $\phi(\bar{x}) = Ans(\bar{x}) \leftarrow \bigwedge_{1 \leq i \leq k}(z_i, L_i, z_i')$ be an AC2RPQ. Then $\phi$ is *strongly* acyclic if (a) $z_i \neq z_i'$, for each $i$ with $1 \leq i \leq k$ (no loops), and (b) for each $i, j$ with $1 \leq i < j \leq k$ it is the case that $\{z_i, z_i'\} \neq \{z_j, z_j'\}$ (no multiedges). The class of strongly acyclic C2RPQs is denoted by SAC2RPQ. Using techniques in [17] one can prove the following:

PROPOSITION 7. SAC2RPQ-EVAL *can be solved in time* $O(|\mathcal{G}| \cdot |\phi|)$*, for each graph database* $\mathcal{G}$ *and* SAC2RPQ $\phi$*.*

**Nested regular expressions** We can extend the expressive power of 2RPQs with an existential branching operator $\langle \cdot \rangle$ *á la* PDL [55] or XPath [38], while retaining good complexity of evaluation. This gives rise to the class of *nested regular expressions* (NREs) [17], that were originally proposed (with a slightly different syntax) for querying Semantic Web data with an RDFS vocabulary [74].

Formally, given a finite alphabet $\Sigma$, the class of NREs $n$ over $\Sigma$ is defined by the following grammar:

$$n \ := \ \varepsilon \mid a\ (a \in \Sigma) \mid a^-\ (a \in \Sigma) \mid n{+}n \mid n{\cdot}n \mid n^* \mid \langle n \rangle$$

We inductively formalize the semantics of a NRE $n$ over a graph database $\mathcal{G} = (V, E)$ as a binary relation $[\![n]\!]_\mathcal{G}$ defined as follows, where $a$ is a symbol in $\Sigma$ and $n, n_1, n_2$ are arbitrary NREs over $\Sigma$:

$$\begin{aligned}
[\![\varepsilon]\!]_\mathcal{G} &= \{(u, u) \mid u \in V\} \\
[\![a]\!]_\mathcal{G} &= \{(u, v) \mid (u, a, v) \in E\} \\
[\![a^-]\!]_\mathcal{G} &= \{(u, v) \mid (v, a, u) \in E\} \\
[\![n_1 + n_2]\!]_\mathcal{G} &= [\![n_1]\!]_\mathcal{G} \cup [\![n_2]\!]_\mathcal{G} \\
[\![n_1 \cdot n_2]\!]_\mathcal{G} &= [\![n_1]\!]_\mathcal{G} \circ [\![n_2]\!]_\mathcal{G} \\
[\![n^*]\!]_\mathcal{G} &= [\![\varepsilon]\!]_\mathcal{G} \cup [\![n]\!]_\mathcal{G} \cup [\![n \cdot n]\!]_\mathcal{G} \cup [\![n \cdot n \cdot n]\!]_\mathcal{G} \cup \cdots \\
[\![\langle n \rangle]\!]_\mathcal{G} &= \{(u, u) \mid \text{there exists } v \text{ such that } (u, v) \in [\![n]\!]_\mathcal{G}\}.
\end{aligned}$$

Here, the symbol $\circ$ denotes the usual composition of binary relations, that is, $[\![n_1]\!]_\mathcal{G} \circ [\![n_2]\!]_\mathcal{G} = \{(u, v) \mid$ there exists $w$ such that $(u, w) \in [\![n_1]\!]_\mathcal{G}$ and $(w, v) \in [\![n_2]\!]_\mathcal{G}\}$. As it is customary, we use $n^+$ as a shortcut for $n \cdot n^*$.

Clearly, the class of 2RPQs is contained in the class of NREs, but the opposite does not hold: We know that the acyclic C2RPQ $Ans(x, y) \leftarrow (x, \texttt{creator}^-, u) \wedge (u, \texttt{partOf}, v) \wedge (v, \texttt{series}, w) \wedge (u, \texttt{creator}, y)$, presented in Example 3, cannot be expressed as a 2RPQ, but it is equivalent to the NRE

$$n \ = \ \texttt{creator}^- \cdot \langle \texttt{partOf} \cdot \texttt{series} \rangle \cdot \texttt{creator}.$$

Although strictly more expressive, NREs retain the good properties of 2RPQs for query evaluation:

THEOREM 5 (SEE E.G., [74]). NRE-EVAL *can be solved in time* $O(|\mathcal{G}| \cdot |n|)$*, for each graph database* $\mathcal{G}$ *and* NRE $n$*.*

The expressive power of C2RPQs and NREs can only be compared in terms of binary C2RPQs (since NREs define binary relations). Interestingly enough, binary C2RPQs and NREs are incomparable in terms of their expressive power [17]:

PROPOSITION 8.
- *There exists a* C2RPQ $\varphi(x, y)$ *over a finite alphabet* $\Sigma$*, such that for no* NRE $n$ *over* $\Sigma$ *it is the case that* $[\![\varphi(x, y)]\!]_\mathcal{G} = [\![n]\!]_\mathcal{G}$*, for each graph database* $\mathcal{G}$*.*
- *There exists a* NRE $n$ *over a finite alphabet* $\Sigma$*, such that for no* C2RPQ $\varphi(x, y)$ *over* $\Sigma$ *it is the case that* $[\![\varphi(x, y)]\!]_\mathcal{G} = [\![n]\!]_\mathcal{G}$*, for each graph database* $\mathcal{G}$*.*

The proof of the first part is very simple: NREs are acyclic, and, thus, they cannot define arbitrary non-acyclic CRPQs such as $Ans(x,y) \leftarrow (x,a,y),(y,b,z),(z,c,x)$. The second part is more interesting. The proof is based on the fact that NREs allow the use of the Kleene-star $*$ over the branching operator $\langle\cdot\rangle$, a feature that cannot be codified in C2RPQs. For instance, the NRE

$$n \;=\; \big(\texttt{creator}^{-} \cdot \langle \texttt{partOf} \cdot \texttt{series}\rangle \cdot \texttt{creator}\big)^{+}$$

expresses the transitive closure of the conference coauthorship query in Example 3. This query is not expressible as a C2RPQ.

## 4.4 Containment

We study the containment problem for some of the languages presented in this section: 2RPQs, NREs and UC2RPQs. Recall that the containment problem $\mathcal{L}$-CONT, for query language $\mathcal{L}$, is defined as follows: Given queries $Q$ and $Q'$ in $\mathcal{L}$, is it the case that $[\![Q]\!]_{\mathcal{G}} \subseteq [\![Q']\!]_{\mathcal{G}}$ for every graph database $\mathcal{G}$?

We start with 2RPQs:

THEOREM 6 ([30]). 2RPQ-CONT *is* PSPACE*-complete.*

Proving PSPACE-completeness for RPQs is easy. Let $L$ and $L'$ be RPQs, i.e., regular expressions over $\Sigma$. It can be proved that $[\![L]\!]_{\mathcal{G}} \subseteq [\![L']\!]_{\mathcal{G}}$, for each graph database $\mathcal{G}$, iff the regular language defined by $L$ is contained in the regular language defined by $L'$, which is a well-known PSPACE-complete problem [79]. Extending the PSPACE upper bound to 2RPQs requires more work, as one has to reason about paths that traverse edges in both directions with two-way automata, and then check containment for them with only one exponential blow up [30].

Notably, NREs not only preserve the complexity of evaluation of 2RPQs, but also the complexity of containment (at the cost of a more involved proof):

THEOREM 7 ([75]). NRE-CONT *is* PSPACE*-complete.*

But moving towards UC2RPQs causes a jump in complexity:

THEOREM 8 ([27]). UC2RPQ-CONT *is* EXPSPACE*-complete. The problem remains hard even if restricted to Boolean acyclic* CRPQs*.*

The upper bound in [27] is only proved for C2RPQs, but it is easy to see that the same techniques extend to UC2RPQs. Those techniques are based on a clever codification of the problem as a containment problem for two-way automata of exponential size. An EXPSPACE upper bound for the containment of CRPQs (i.e., no inverses) had been previously obtained in [46] using different techniques. The lower bound is proved via a reduction from an EXPSPACE version of the tiling problem [27]. An inspection of the proof shows that this lower bound holds even for checking containment of Boolean acyclic CRPQs.

# 5. PATH QUERIES

The class of UC2RPQs falls short of expressive power for several modern applications of graph databases. In many of these applications a minimal requirement for sufficiently expressive queries are: (a) the ability to define complex semantic relationships among paths, and (b) the ability to include paths in the output of a query. None of these functionalities is provided by UC2RPQs.

There are multiple examples of queries that require these new capabilities. For instance, several important Semantic Web queries (as we will see later) can only be expressed by comparing paths [8, 9], biological sequences are often compared with respect to edit distance and path similarity [52], route-finding applications need to compare paths based on length or number of occurences of labels [19], etc. In addition, including paths in the output has applications in the Semantic Web [9, 62], provenance of data [58], semantic search over the Web [82], and others.

For the sake of simplicity, we only consider extensions with the new capabilities for CRPQs. Fix a countable set of *node* variables (typically denoted by $x, y, z, \ldots$), and a countable set of *path* variables (denoted by $\pi, \omega, \chi, \ldots$). Let $\mathcal{S}$ be a set of relations on finite words, such that each $S \in \mathcal{S}$ is a relation over some finite alphabet $\Sigma$, and assume that $\mathcal{S}$ includes all regular languages. (Examples of a set $\mathcal{S}$ of this kind are the *regular* and *rational* relations, as we define afterwards). The class of $\mathcal{S}$*-extended* CRPQs (from now on, ECRPQ($\mathcal{S}$)) over $\Sigma$ consists of all rules $\theta(\bar{x},\bar{\chi})$ of the form:

$$Ans(\bar{x},\bar{\chi}) \leftarrow \bigwedge_{1\leq i\leq k} (z_i,\pi_i,z'_i),\ \bigwedge_{1\leq j\leq t} S_j(\bar{\omega}_j) \qquad (2)$$

such that the following holds:

- each one of the elements in $Z = \{z_1, z'_1, \ldots, z_k, z'_k\}$ is a node variable, and each one of the elements in $P = \{\pi_1, \ldots, \pi_t\}$ is a path variable;
- $\bar{x}$ is a tuple of node variables among those in $Z$, and $\bar{\chi}$ is a tuple of path variables among those in $P$;
- each $\bar{\omega}_j$ ($1 \leq j \leq t$) is a tuple of elements in $P$; and
- each $S_j$ ($1 \leq j \leq t$) is a relation in $\mathcal{S}$ over $\Sigma$, of the same arity than $\bar{\omega}_j$.

Notice that this definition meets our requirements: ECRPQ($\mathcal{S}$) queries allow for paths to be compared with respect to the relations in $\mathcal{S}$, and, in addition, path variables are admitted in the output.

The semantics of a rule $\theta(\bar{x},\bar{\chi})$ of the form (2) is defined in terms of homomorphisms. A homomorphism from $\theta$ to a graph database $\mathcal{G} = (V,E)$ consists of a pair $(\sigma,\nu)$ of mappings such that $\sigma : \{z_1, z'_1, \ldots, z_k, z'_k\} \to V$, the mapping $\nu$ assigns a path in $\mathcal{G}$ to each path variable $\pi_i$, $1 \leq i \leq k$, and the following holds: (1) $\nu(\pi_i)$ is a path from $\sigma(z_i)$ to $\sigma(z'_i)$, for each $i$ with $1 \leq i \leq k$, and (2) the tuple $\lambda(\nu(\bar{\omega}_j))$ – defined by the labels of the paths in the tuple $\nu(\bar{\omega}_j)$ – belongs to $S_j$, for each $j$ with $1 \leq j \leq t$. We define $[\![\theta(\bar{x},\bar{\chi})]\!]_{\mathcal{G}}$ as the set of tuples of the form $(\sigma(\bar{x}),\nu(\bar{\chi}))$, for each homomorphism $(\sigma,\nu)$ from $\theta(\bar{x},\bar{\chi})$ to $\mathcal{G}$.

We can now justify why the languages are called *extended* CRPQs. Indeed, the class of CRPQs is contained in the class of ECRPQ($\mathcal{S}$) queries: the CRPQ $Ans(\bar{x}) \leftarrow \bigwedge_i (z_i, L_i, z'_i)$ can be expressed as the ECRPQ($\mathcal{S}$) $Ans(\bar{x}) \leftarrow \bigwedge_i (z_i,\pi_i,z'_i), L_i(\pi_i)$ (since we assume that $\mathcal{S}$ contains all regular languages).

The expressiveness and complexity of ECRPQs depends, of course, on the class of relations on words we allow on $\mathcal{S}$. We study next two important such classes: regular and rational relations.

## 5.1 Regular relations

Following the idea behind CRPQs, which allow regular conditions on paths, we use *regular relations* [41, 49, 21] for path comparisons in ECRPQs [13]. As we will see next, regular relations permit a good trade-off between expressiveness and complexity for ECRPQs based on them.

Regular relations are recognized by *synchronous $n$-ary automata*, that have $n$ input tapes onto which the input strings are written, followed by an infinite sequence of $\perp$ symbols. At each step the automaton simultaneously reads the next symbol on each tape, terminating when it reads $\perp$ on each tape.

Formally, let $\perp$ be a symbol not in $\Sigma$. We denote the extended alphabet $(\Sigma \cup \{\perp\})$ by $\Sigma_\perp$. Let $\bar{s} = (s_1, \ldots, s_n)$ be an $n$-tuple of strings over alphabet $\Sigma$. We construct a string $[\bar{s}]$ over alphabet $(\Sigma_\perp)^n$, whose length is the maximum of the $s_j$'s, and whose $i$-th symbol is a tuple $(c_1, \ldots, c_n)$, where each $c_k$ is the $i$-th symbol of $s_k$, if the length of $s_k$ is at least $i$, or $\perp$ otherwise. In other words, we pad shorter strings with the symbol $\perp$, and then view the $n$ strings as one string over the alphabet of $n$-tuples of letters.

An $n$-ary relation $S$ on $\Sigma^*$ is *regular*, if the set $\{[\bar{s}] \mid \bar{s} \in S\}$ of strings over alphabet $(\Sigma_\perp)^n$ is regular (i.e., accepted by an automaton over $(\Sigma_\perp)^n$, or given by a regular expression over $(\Sigma_\perp)^n$). We denote by Reg the set of all regular relations.

Clearly, the regular relations of arity 1 are exactly the regular languages. Examples of binary regular relations on words $w_1$ and $w_2$ are: (a) path equality: $w_1 = w_2$; (b) length comparisons: $|w_1| = |w_2|$ (and likewise for $<$ and $\leq$); (c) prefix: $w_1$ is a prefix of $w_2$; (d) small edit distance: edit distance between $w_1$ and $w_2$ is at most $k$, for a fixed $k$. On the other hand, several interesting relations on words are not regular; e.g., the binary relation $\preceq_{\text{ss}}$, that consists of all pairs $(w_1, w_2)$ such that $w_1$ is a subsequence of $w_2$ (i.e., $w_1$ can be obtained by deleting some letters, perhaps none, from $w_2$), and the binary subword relation $\preceq_{\text{sw}}$ that contains all pairs $(w_1, w_2)$ such that $w_3 w_1 w_4 = w_2$, for words $w_3, w_4$.

Thus, an ECRPQ(Reg) query over $\Sigma$ is an expression of the form $Ans(\bar{x}, \bar{\chi}) \leftarrow \bigwedge_i (z_i, \pi_i, z'_i), \bigwedge_j R_j(\bar{\omega}_j)$, where each $R_j$ is a regular relation over $\Sigma$. While ECRPQ(Reg) contains all CRPQs, it can be proved, on the contrary, that the containment is proper: Assume that $el$ is the binary regular relation that checks whether two words have the same length. The ECRPQ(Reg) query

$$Ans(x, y) \leftarrow (x, \pi_1, z) \wedge (z, \pi_2, y) \wedge a^*(\pi_1) \wedge b^*(\pi_2) \wedge el(\pi_1, \pi_2)$$

computes all nodes in a graph database over $\Sigma = \{a, b\}$ that are linked by a path labeled in $\{a^n b^n \mid n \geq 0\}$. A pumping argument shows that this query is not expressible as a CRPQ [13, 48].

EXAMPLE 4. The ECRPQ(Reg) $Ans(x, y, z) \leftarrow (x, \pi_1, z) \wedge (y, \pi_2, z) \wedge el(\pi_1, \pi_2)$ defines the tuples $(x, y, z)$ of nodes such that $z$ can be reached from both $x$ and $y$ following paths of the same length. This query might be of interest, e.g., in route-finding applications. If we replace $el$ with the binary relation that checks whether the edit distance between the labels of $\pi_1$ and $\pi_2$ is at most $k$, we have a similarity query motivated by sequence alignment in biological networks.

In a query language for RDF introduced in [8], paths can be compared based on specific *semantic associations*. Edges correspond to RDF properties and paths to property sequences. A property $a$ can be declared to be a subproperty of property $b$, which we denote by $a \prec b$. Two property sequences $u$ and $v$ are called $\rho$-isomorphic iff $u = u_1 \cdots u_n$ and $v = v_1 \cdots v_n$, for some $n$, and $u_i \prec v_i$ or $v_i \prec u_i$ for every $i \leq n$. Nodes $x$ and $y$ are *$\rho$-isoAssociated* iff $x$ and $y$ are the origins of two $\rho$-isomorphic property sequences.

Finding nodes which are $\rho$-isoAssociated cannot be done in a query language supporting only conventional regular expressions, not least because doing so requires checking that two paths are of equal length. However, pairs of $\rho$-isomorphic sequences can be expressed using the regular relation $R$ given by the following regular expression: $(\bigcup_{a,b \in \Sigma:\ (a \prec b \vee b \prec a)} (a, b))^*$. Then an ECRPQ(Reg) returning pairs of nodes $x$ and $y$ that are $\rho$-isoAssociated, and the respective paths, can be written as follows: $Ans(x, y, \pi_1, \pi_2) \leftarrow (x, \pi_1, z_1) \wedge (y, \pi_2, z_2) \wedge R(\pi_1, \pi_2)$. □

**Complexity and containment** ECRPQ(Reg) extends the class of CRPQs, but is there an associated complexity cost? Recall that ECRPQ(Reg)-EVAL is the problem of, given a graph database $\mathcal{G} = (V, E)$, an ECRPQ(Reg) query $\theta(\bar{x}, \bar{\chi})$, a tuple $\bar{v}$ of nodes in $V$ and a tuple $\bar{\rho}$ of paths in $\mathcal{G}$, does $(\bar{v}, \bar{\rho})$ belong to $[\![\theta(\bar{x}, \bar{\chi})]\!]_{\mathcal{G}}$?

THEOREM 9 ([13]). ECRPQ(Reg)-EVAL *is* PSPACE*-complete, and* NLOGSPACE*-complete in data complexity.*

Thus, extending CRPQs with regular relations is free in data complexity, but combined complexity goes up from NP to PSPACE (which is, in any case, the same as the complexity of evaluation of FO over relational databases [2]).

Since ECRPQ(Reg) queries can return paths, the evaluation of a query might be infinite (for example, if there is a cycle in the graph database, then we have infinitely many paths). In such cases it is possible to return a compact representation of the answer. In fact, for each graph database $\mathcal{G}$, ECRPQ(Reg) query $\theta(\bar{x}, \bar{\chi})$, and tuple $\bar{v}$ of node ids, the set $\{\bar{\rho} \mid (\bar{v}, \bar{\rho}) \in [\![\theta]\!]_{\mathcal{G}}\}$ is a regular relation, and an automaton defining exactly this relation can be constructed in exponential time (and in polynomial time if $\theta$ is fixed) [13].

The precise complexity of evaluation for several extensions and restrictions of the class ECRPQ(Reg) is studied in [13]. The expressive power of the class is also by now well understood [48].

Let us consider finally the containment problem for ECRPQ(Reg). While this problem is decidable for CRPQs (Theorem 8), adding regular relations to compare paths dramatically changes the situation.

THEOREM 10 ([13]). ECRPQ(Reg)-CONT *is undecidable, even for Boolean queries over a fixed alphabet.*

The proof follows by a codification of the containment problem for *pattern languages* [77], which is undecidable [60, 47].

## 5.2 Rational relations

ECRPQ(Reg) queries are still short of the expressiveness needed in many applications. For instance, associations between paths used in RDF or biological networks often deal with subwords or subsequences, but these relations are not regular. They are rational [20]: they are still accepted by automata, but those whose heads move asynchronously.

Adding rational relations to a query language must be done with extreme care: simply replacing regular relations with rational in ECRPQs makes query evaluation undecidable or impractical. In fact, as we show below this happens even if the class ECRPQ(Reg) is extended with no more than the subword (or subsequence) relation to compare path labels. On the other hand, we can achieve tractable data complexity (and reasonable combined complexity) by restricting the syntactic shape of queries and disallowing rational relations of arity three or more.

Rational relations can be defined by means of *asynchronous* $n$-tape automata, that have $n$ heads for the tapes and one additional control; at every step, based on the state and the letters it is reading, the automaton can enter a new state and move some (but not necessarily all) tape heads. Alternatively, $n$-ary rational relations can be defined as regular expressions over the alphabet $(\Sigma \cup \{\epsilon\})^n$, where $\epsilon$ denotes the empty symbol [20]. We use the notation Rat to denote the class of all rational relations.

Clearly, Reg $\subseteq$ Rat. Furthermore, Reg $=$ Rat for relations of arity 1 (both define exactly the class of regular languages). On the other hand, Rat $\not\subseteq$ Reg for relations of arity bigger than 1: The binary subsequence and subword relations – $\preceq_{\text{ss}}$ and $\preceq_{\text{sw}}$, respectively, as defined in Section 5.1 – are not regular but they can easily be proved to be rational.

An ECRPQ(Rat) query over $\Sigma$ is thus an expression of the form $Ans(\bar{x}, \bar{\chi}) \leftarrow \bigwedge_i (z_i, \pi_i, z'_i), \bigwedge_j S_j(\bar{\omega}_j)$, where each $S_j$ is a rational relation over $\Sigma$. For instance, the ECRPQ(Rat) query

$$Ans(x, y) \leftarrow (x, \pi_1, z) \wedge (y, \pi_2, w) \wedge \preceq_{\text{ss}} (\pi_1, \pi_2)$$

defines the pairs $(x, y)$ of nodes such that $x$ is the starting point of the path $\pi_1$, $y$ is the starting point of the path $\pi_2$, and $\lambda(\pi_1)$ is a subsequence of $\lambda(\pi_2)$.

**Complexity** It follows easily from the undecidability of the intersection problem for rational relations [20] that ECRPQ(Rat)-EVAL is undecidable. However, we are not interested here in arbitrary rational relations but only on those that are useful in practice, e.g., $\preceq_{\text{sw}}$ and $\preceq_{\text{ss}}$. Unfortunately, none of these relations can be added to the class ECRPQ(Reg) without imposing further conditions:

THEOREM 11 ([12]). *1. There exists an* ECRPQ(Reg $\cup$ $\{\preceq_{\text{sw}}\}$) *query $Q$ such that* EVAL($Q$) *is undecidable.*

*2.* ECRPQ(Reg $\cup$ $\{\preceq_{\text{ss}}\}$)-EVAL *is decidable, but there exists an* ECRPQ(Reg $\cup$ $\{\preceq_{\text{ss}}\}$) *query $Q$ such that* EVAL($Q$) *is nonelementary.*

Theorem 11 rules out the applicability of query languages for graph databases that freely combine regular relations with some of the most common rational relations. In order to obtain acceptable complexity bounds, we thus need to further restrict the syntactic shape of queries. We study next a robust class of queries, with a simple syntactic definition, that yields tractable data complexity and reasonable combined complexity for queries defined by arbitrary rational relations of arity at most 2. The intuitive idea behind this restriction is forbidding features in ECRPQ(Rat) that allow to codify the intersection of rational relations, since it is known that this leads to undecidability of query evaluation.

Let $\mathsf{Rat}_{\leq 2}$ be the class of unary and binary rational relations. We can assume without loss of generality that an ECRPQ($\mathsf{Rat}_{\leq 2}$) query is an ECRPQ(Rat) expression of the form:

$$Ans(\bar{x}, \bar{\chi}) \leftarrow \bigwedge_{1 \leq i \leq k} (z_i, \pi_i, z'_i), S_i(\pi_i), \bigwedge_{1 \leq j \leq t} S'_j(\omega_j^1, \omega_j^2),$$

such that the $S_i$'s are regular languages, the $S'_j$'s are binary rational relations, and each $\omega_j^1$ and $\omega_j^2$ is a path variable among $\{\pi_1, \ldots, \pi_k\}$. This query is *intersection-free* if the following holds:

1. For each $1 \leq i \leq k$ it is the case that $\omega_i^1 \neq \omega_i^2$, and for each $1 \leq i < j \leq t$ it is the case that $\{\omega_i^1, \omega_i^2\} \neq \{\omega_j^1, \omega_j^2\}$. The first condition disallows the codification of the intersection of binary rational relations in an atom $S_i(\omega_i^1, \omega_i^2)$, and the second one over two different atoms $S'_i(\omega_i^1, \omega_i^2)$ and $S'_j(\omega_j^1, \omega_j^2)$.

2. Let $I$ be the subset of $\{1, \ldots, k\} \times \{1, \ldots, k\}$ such that $(i_1, i_2) \in I$ ($1 \leq i_1, i_2 \leq k$) if and only if for some $1 \leq j \leq t$ it is the case that $\omega_j^1 = \pi_{i_1}$ and $\omega_j^2 = \pi_{i_2}$. We require that the undirected graph defined by $I$ on $\{1, \ldots, k\}$ is acyclic. The reason is that intersection of binary rational relations can be codified in the query using cycles in $I$.

As an example, the following Boolean query is intersection-free (we have omitted the head predicate $Ans()$):

$$\bigwedge_{1 \leq i \leq 4} (x_i, \pi_i, y) \wedge \preceq_{\text{ss}} (\pi_1, \pi_2) \wedge \preceq_{\text{ss}} (\pi_2, \pi_3) \wedge \preceq_{\text{sw}} (\pi_2, \pi_4).$$

Intersection-free ECRPQ($\mathsf{Rat}_{\leq 2}$) queries have been studied in the literature under the name of *acyclic* queries [12]. We decided to change its name here since a different and important notion of acyclicity was already introduced in Section 4.3 for CRPQs.

Intersection-free ECRPQ($\mathsf{Rat}_{\leq 2}$) queries allow for tractable evaluation in data complexity, and its complexity coincides with that of ECRPQ(Reg) queries:

THEOREM 12 ([12]). *Let $\mathcal{F}$ be the class of intersection-free* ECRPQ($\mathsf{Rat}_{\leq 2}$) *queries. Then $\mathcal{F}$-*EVAL *is* PSPACE*-complete, and* NLOGSPACE*-complete in data complexity.*

Notably, the conditions imposed by intersection-free queries are, in a sense, optimal: Removing either condition (1) or (2) from its definition leads to undecidability [12]. The same happens if we allow relations of arity 3 or more in queries, even under strong extensions of conditions (1) and (2). On the other hand, the acyclicity condition (2) can be removed at the cost of restricting the rational relations allowed in queries. Results of this kind for several binary rational relations of interest (e.g., the subsequence or suffix relations) can be found in [12].

# 6. QUERIES ON GRAPHS WITH DATA

All query languages we have studied so far concentrate on the topological properties of the graph database, but do not talk much about the data itself. However, it is clear that graph databases contain data; e,g. in social networks nodes represent persons, and these persons have associated attributes such as name, age, location, etc. In addition, queries that combine topology and data are relevant in practice. Think, for instance, of the query that asks for pairs of persons of the same age that are connected via profesional links in a social network, or the query that asks for authors in DBLP that have papers in at least 3 different conferences.

We show in this section that languages that freely compare data values along a path easily become intractable in data complexity, but that there is a nice class of path queries, with tractable data complexity, that expresses relevant topological properties of the data.

**Data model** Let $\mathcal{D}$ be a countably infinite set of data values. A *data graph* is a graph database in which each node stores a data value from $\mathcal{D}$. Formally, given a finite alphabet $\Sigma$, a data graph $\mathfrak{G}$ over $\Sigma$ is a tuple $(V, E, \kappa)$, such that $(V, E)$ is a graph database over $\Sigma$ and $\kappa$ is a mapping that assigns a data value in $\mathcal{D}$ to each node $v \in V$. Notice that graph databases in which nodes are labeled with more than one data value can be represented in this model: We simply add extra edges to nodes with those data values.

Query languages over data graphs talk about *data paths*, which are obtained from paths by replacing each node by its data value. Formally, let $\mathfrak{G} = (V, E, \kappa)$ be a data graph. With each path $\rho = v_0 a_0 v_1 \cdots v_{k-1} a_{k-1} v_k$ in $(V, E)$ there is an associated data path $\rho_{\mathcal{D}} = \kappa(v_0) a_0 \kappa(v_1) \cdots \kappa(v_{k-1}) a_{k-1} \kappa(v_k)$ in $\mathfrak{G}$.

**Query languages** Data paths are very close to an object that has received considerable attention in the XML community: *Data words* [23], which are words over the infinite alphabet $\Sigma \times \mathcal{D}$. In fact, one can represent each data path as a data word with an extra data value in the end. The importance of this is that one can easily adapt the multiple formalisms that have been developed in the literature to query data words to express queries about data paths. These formalisms include FO extended with a binary relation $\sim$ that stores pairs of nodes with the same data value [22], pebble automata [71], register automata [61] and some versions of XPath.

Nevertheless, choosing the right formalism for querying data graphs has to be done with care. This is because checking even some simple data path properties is an intractable problem:

THEOREM 13 ([65]). *The following problem is* NP*-complete: Given a data graph $\mathfrak{G} = (V, E, \kappa)$ and two nodes $v, v' \in V$, determine if there is a data path $\rho_{\mathcal{D}}$ in $\mathfrak{G}$ from $v$ to $v'$ such that all data values in $\rho_{\mathcal{D}}$ are different.*

Thus, any query language for data graphs that expresses this simple property will be NP-complete in data complexity, and, thus, impractical. This rules out all formalisms mentioned before, except register automata. Libkin and Vrgoč studied the class of register automata as a querying formalism for data paths, and established some of its good properties [65]. But, as they argue, expressing properties of data paths with register automata is not simple. This motivated them to introduce the class of *regular expressions with memory* (REMs), that is based on register automata, but allows a more natural specification of queries over data graphs. We present this formalism below.

**REMs** REMs allow us to specify when data values are remembered and used. Data values are stored in $k$ registers, represented by variables $x_1, \ldots, x_k$. At any point we can compare a data value with one previously stored in the registers. As an example, consider the REM $\downarrow x.a^+[x^=]$. It can be read as follows: Store the current data value in $x$, and then check that after reading a word in $a^+$ we see the same data value again (condition $[x^=]$). We formally define the class of REMs below.

Let $x_1, \ldots, x_k$ be variables. The set of *conditions* over $\{x_1, \ldots, x_k\}$ is recursively defined as: $c := x_i^= \mid c \wedge c \mid \neg c$, for $1 \leq i \leq k$. Satisfaction of conditions is defined with respect to a data value $d \in D$ and a tuple $\tau = (d_1, \ldots, d_k) \in \mathcal{D}^k$ as follows (we omit Boolean conditions): $(d, \tau) \models x_i^=$ iff $d = d_i$.

The class of REMs over $\Sigma$ and $\{x_1, \ldots, x_k\}$ is given by the grammar:

$$e := \varepsilon \mid a \mid e \cup e \mid e \cdot e \mid e^+ \mid e[c] \mid \downarrow \bar{x}.e$$

where $a$ ranges over symbols in $\Sigma$, $c$ over conditions over $\{x_1, \ldots, x_k\}$, and $\bar{x}$ over tuples of elements in $\{x_1, \ldots, x_k\}$. We need to rule out some pathological cases if we want to have a clear semantics for REMs. In order to do that, we assume the following: (1) Subexpressions of the form $e^+$, $e[c]$ and $\downarrow \bar{x}.e$ are not allowed, for each $e$ that *reduces* to $\varepsilon$. The expression $e$ reduces to $\varepsilon$ if $e = \epsilon$, or $e$ is of the form $e_1 \cup e_2$ or $e_1 \cdot e_2$ or $e_1^+$ or $e_1[c]$ or $\downarrow \bar{x}.e_1$, where $e_1$ (and $e_2$) reduce to $\varepsilon$. (2) No variable appears in a condition before it has been mentioned in $\downarrow \bar{x}$.

REMs are used as analogs of RPQs for data graphs. As such, they define pairs of nodes linked by data paths satisfying the conditions expressed in the REM. To define the evaluation $[\![e]\!]_{\mathfrak{G}}$ of a REM $e$ over a data graph $\mathfrak{G} = (V, E, \kappa)$, we use a relation $(e, \rho_{\mathcal{D}}, \lambda) \vdash \lambda'$, for $e$ a REM, $\rho_{\mathcal{D}}$ a data path, and $\lambda, \lambda'$ two $k$-tuples over $\mathcal{D} \cup \{\perp\}$ (the symbol $\perp$ represents that the variable has not been assigned a data value yet). The intuition behind the relation $(e, \rho_{\mathcal{D}}, \lambda) \vdash \lambda'$ is the following: The data path $\rho_{\mathcal{D}}$ can be parsed according to $e$, with $\lambda$ being the initial assignment of the variables, in such a way that the final assignment is $\lambda'$. We then define $[\![e]\!]_{\mathfrak{G}}$ as the pairs $(u, v)$ of node ids in $V$, such that there is a data path $\rho_{\mathcal{D}}$ in $\mathfrak{G}$ from $u$ to $v$ for which it is the case that $(e, \rho_{\mathcal{D}}, \perp^k) \vdash \lambda$, for some $k$-tuple $\lambda$ over $\mathcal{D} \cup \{\perp\}$.

We inductively define the relation $(e, \rho_{\mathcal{D}}, \lambda) \vdash \lambda'$ below. But before, we need to introduce some extra terminology. We assume that $\lambda_{\bar{x}=d}$ is the tuple obtained from $\lambda$ by setting all variables in $\bar{x}$ to be $d$. Also, the *concatenation* of two data paths of the form $d_0 a_0 d_1 \cdots d_{k-1} a_{k-1} d_k$ and $d_k a_k d_{k+1} \cdots d_{n-1} a_{n-1} d_n$ is defined as $d_0 a_0 d_1 \cdots d_{k-1} a_{k-1} d_k a_k d_{k+1} \cdots d_{n-1} a_{n-1} d_n$. If $\rho_{\mathcal{D}}$ is obtained by concatenating the data paths $\rho_{\mathcal{D}}^1, \rho_{\mathcal{D}}^2, \ldots, \rho_{\mathcal{D}}^\ell$, i.e., $\rho_{\mathcal{D}} = \rho_{\mathcal{D}}^1 \rho_{\mathcal{D}}^2 \cdots \rho_{\mathcal{D}}^\ell$, then we say that there is a *splitting* of $\rho_{\mathcal{D}}$ into $\rho_{\mathcal{D}}^1 \rho_{\mathcal{D}}^2 \cdots \rho_{\mathcal{D}}^\ell$. Then:

- $(\varepsilon, \rho_{\mathcal{D}}, \lambda) \vdash \lambda'$ iff $\rho_{\mathcal{D}} = d$, for some $d \in \mathcal{D}$, and $\lambda = \lambda'$.
- $(a, \rho_{\mathcal{D}}, \lambda) \vdash \lambda'$ iff $\rho_{\mathcal{D}} = d_1 a d_2$ and $\lambda = \lambda'$.
- $(e_1 \cup e_2, \rho_{\mathcal{D}}, \lambda) \vdash \lambda'$ iff $(e_1, \rho_{\mathcal{D}}, \lambda) \vdash \lambda'$ or $(e_2, \rho_{\mathcal{D}}, \lambda) \vdash \lambda'$.
- $(e_1 \cdot e_2, \rho_{\mathcal{D}}, \lambda) \vdash \lambda'$ iff there is a splitting $\rho_{\mathcal{D}}^1 \rho_{\mathcal{D}}^2$ of $\rho_{\mathcal{D}}$ and a $k$-tuple $\lambda''$ over $\mathcal{D} \cup \{\perp\}$, such that $(e_1, \rho_{\mathcal{D}}^1, \lambda) \vdash \lambda''$ and $(e_2, \rho_{\mathcal{D}}^2, \lambda'') \vdash \lambda'$.
- $(e^+, \rho_{\mathcal{D}}, \lambda) \vdash \lambda')$ iff there is a splitting $\rho_{\mathcal{D}}^1 \rho_{\mathcal{D}}^2 \cdots \rho_{\mathcal{D}}^\ell$ of $\rho_{\mathcal{D}}$ and $k$-tuples $\lambda = \lambda_0, \lambda_1, \ldots, \lambda_m = \lambda'$ over $\mathcal{D} \cup \{\perp\}$, such that $(e, \rho_{\mathcal{D}}^i, \lambda_{i-1}) \vdash \lambda_i$, for all $i$ with $1 \leq i \leq \ell$.
- $(e[c], \rho_{\mathcal{D}}, \lambda) \vdash \lambda'$ iff $(e, \rho_{\mathcal{D}}, \lambda) \vdash \lambda'$ and $(d, \lambda') \models c$, where $d$ is the last data value of $\rho_{\mathcal{D}}$.
- $(\downarrow \bar{x}.e, \rho_{\mathcal{D}}, \lambda) \vdash \lambda'$ iff $(e, \rho_{\mathcal{D}}, \lambda_{\bar{x}=d}) \vdash \lambda'$, where $d$ is the first data value of $\rho_{\mathcal{D}}$.

EXAMPLE 5. The REM $\Sigma^* \cdot (\downarrow x.\Sigma^+[x^=]) \cdot \Sigma^*$ defines the pairs of nodes that are linked by a data path in which two data values are the same. Notice that the complement of this query defines precisely the NP-complete property in Theorem 13, which is not expressible as a REM [65] (in particular, REMs are not closed under complementation). The REM $\downarrow x.(a[\neg x^=])^+$ defines the pairs of nodes that are linked by a data path labeled in $a$, such that its first data value is different from all other data values. □

REMs are tractable in data complexity and have no worst combined complexity than FO over relational databases:

THEOREM 14 ([65]). *REM-*EVAL *is* PSPACE*-complete, and* NLOGSPACE*-complete in data complexity.*

It is easy to see that this behavior extends to the class of *conjunctive* REMs, that can be defined analogously to CRPQs but assuming that basic atoms are REMs. Tractable cases of REM evaluation in combined complexity can be obtained by restricting the data comparison power of the expressions [65]. Different query languages for data graphs with tractable combined complexity have been designed using XPath features [66].

Finally, containment of REMs is undecidable. This follows from undecidability of containment for register automata [71].

THEOREM 15. *REM-*CONT *is undecidable.*

# 7. CONCLUSIONS AND FUTURE CHALLENGES

Figure 3 summarizes the complexity of evaluation and containment for most of the query languages studied in the paper. This includes traditional query languages for graph databases, such as **G**, 2RPQs, and C2RPQs. For the first one, even data complexity is intractable due to the simple path semantics. Allowing a semantics based on arbitrary paths yields tractable data complexity for C2RPQs. Combined complexity of C2RPQs is intractable, which has motivated the search for expressive fragments with good evaluation properties. This includes the classes of acyclic C2RPQs and NREs, that can be evaluated in quadratic and linear time in the size of the data, respectively. The containment problem for C2RPQs is in EXPSPACE, but for 2RPQs and NREs it is in PSPACE.

We also studied expressive languages for path queries. The class ECRPQ(Reg) preserves good data complexity, but fails to retain

| | Combined complexity | Data complexity | Containment |
|---|---|---|---|
| **G** | NP-complete | NP-complete | ? |
| 2RPQs | $O(\|\mathcal{G}\| \cdot \|L\|)$ | NLOGSPACE-complete | PSPACE-complete |
| C2RPQs | NP-complete | NLOGSPACE-complete | EXPSPACE-complete |
| AC2RPQs | $O(\|\mathcal{G}\|^2 \cdot \|\phi\|^2)$ | NLOGSPACE-complete | EXPSPACE-complete |
| NREs | $O(\|\mathcal{G}\| \cdot \|n\|)$ | NLOGSPACE-complete | PSPACE-complete |
| ECRPQ(Reg) | PSPACE-complete | NLOGSPACE-complete | undecidable |
| ECRPQ(Reg $\cup \{\preceq_{\text{sw}}\}$) | undecidable | undecidable | undecidable |
| ECRPQ(Reg $\cup \{\preceq_{\text{ss}}\}$) | nonelementary | nonelementary | undecidable |
| Intersection-free ECRPQ($\mathsf{Rat}_{\leq 2}$) | PSPACE-complete | NLOGSPACE-complete | undecidable |
| REMs | PSPACE-complete | NLOGSPACE-complete | undecidable |

**Figure 3: Combined complexity, data complexity, and complexity of containment for several languages studied in the paper.**

decidability for containment. Adding some natural non-rational relations (such as subword or subsequence) to ECRPQ(Reg) leads to either undecidability or high complexity of evaluation (even for a fixed query). However, intersection-free ECRPQ($\mathsf{Rat}_{\leq 2}$) queries, that allow arbitrary rational relations of arity at most 2 but restrict the syntactic shape of queries, preserve the evaluation properties of ECRPQ(Reg). Finally, we studied languages for graph databases with data values. Even some simple data path properties are intractable in data complexity, but an expressive language, that is also tractable, was identified; namely, REMs.

The study of graph databases is at an early stage of development, and many crucial questions about them remain unanswered:

*Optimization:* Most of the work on optimization has dealt with the containment and equivalence problem for 2RPQs and C2RPQs. Almost nothing is known about decidable cases of these problems for path or data queries. Furthermore, the study of heuristics based on optimization rules that help determining the most efficient way to execute a query has received not much attention.

Recently, a new notion of *approximate* optimization has been introduced in the context of CQ evaluation over relational databases [16], and subsequently studied for UC2RPQs over graph databases [18]. This notion is motivated by the scenarios of big graph databases, in which tractable combined complexity of evaluation is crucial. The basic idea is, given a UC2RPQ $Q$ and a tractable class $\mathcal{C}$ of UC2RPQs (e.g. unions of AC2RPQs), find a query $Q' \in \mathcal{C}$ (the approximation) that is "as close as possible" to $\mathcal{C}$, and then run the fast query $Q'$ over the underlying data. Approximations studied so far have several good properties, but are a bit coarse in some cases. The introduction of more informative notions of approximations in graph databases is a challenging open problem.

*Constraints:* Graph database constraints received a good amount of attention about 10 years ago, particularly in relationship with the containment problem [4, 50, 39, 25]. On the other hand, query optimization in the presence of constraints is almost unexplored (save for simple queries, such as RPQs [44]), and the topic of dependencies over data graphs has not been studied. Furthermore, research on general purpose languages that allow to express the kind of dependencies that modern graph database applications (e.g. social or biological networks) require is completely open.

*A practical query language:* Current systems for graph databases, such as Dex [68], Neo4j [70], InfiniteGraph [59], and others, lack a language with clear syntax and semantics. This difficults the accurate evaluation of the expressive power and the computational cost of the queries they permit. We believe that the time is ripe for the theoretical community to interact with the graph database vendors, and design a core query language for graph databases that allows to express a common set of queries for different domains, and that can be evaluated at a reasonable computational cost.

**Acknowledgements** I am very grateful to Marcelo Arenas, Gaelle Fontaine, Miguel Romero and Juan Reutter for carefully reading an earlier version of the article and providing me with comments. The author is funded by Fondecyt grant 1130104.

# 8. REFERENCES

[1] S. Abiteboul, P. Buneman, D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kauffman, 1999.

[2] S. Abiteboul, R. Hull, V. Vianu. *Foundations of databases*. Addison-Wesley, 1995.

[3] S. Abiteboul, D. Quass, J. McHugh, J. Widom, J. L. Wiener. The Lorel query language for semistructured data. *Int. J. on Digital Libraries* 1(1), pages 68-88, 1997.

[4] S. Abiteboul, V. Vianu. Regular path queries with constraints. *JCSS* 58(3), pages 428-452, 1999.

[5] R. Angles, C. Gutiérrez. Survey of graph database models. *ACM Comput. Surv.* 40(1), 2008.

[6] N. Alon, R. Yuster, U. Zwick. Finding and counting given length cycles (Extended abstract). In *ESA* 1994, pages 354-364.

[7] M. K. Anand, S. Bowers, B. Ludäscher. Techniques for efficiently querying scientific workflow provenance graphs. In *EDBT* 2010, pages 287-298.

[8] K. Anyanwu, A. P. Sheth. $\rho$-queries: enabling querying for semantic associations on the semantic web. In *WWW* 2003, pages 690-699.

[9] K. Anyanwu, A. Maduko, A. P. Sheth. SPARQ2L: towards support for subgraph extraction queries in RDF databases. In *WWW* 2007, pages 797-806.

[10] M. Arenas, J. Pérez. Querying semantic web data with SPARQL. In *PODS* 2011, pages 305-316.

[11] M. Arenas, S. Conca, J. Pérez. Counting beyond a Yottabyte, or how SPARQL 1.1 property paths will prevent adoption of the standard. In *WWW* 2012, pages 629-638.

[12] P. Barceló, D. Figueira, L. Libkin. Graph-logics with rational relations and the generalized intersection problem. In *LICS* 2012, pages 115-124.

[13] P. Barceló, L. Libkin, A. W. Lin, P. T. Wood. Expressive languages for path queries over graph-structured Data. *TODS* 37(4), 2012.

[14] P. Barceló, L. Libkin, J. Reutter. Querying graph patterns. In *PODS* 2011, pages 199-210.

[15] P. Barceló, L. Libkin, J. Reutter. Parameterized regular expressions and their languages. *TCS* 474, pages 21-45, 2013.

[16] P. Barceló, L. Libkin, M. Romero. Efficient approximations of conjunctive queries. In *PODS*, pages 249-260, 2012.

[17] P. Barceló, J. Reutter, J. Pérez. Relative expressiveness of nested regular expressions. In *AMW* 2012, pages 180-195.

[18] P. Barceló, M. Romero, M. Y. Vardi. Semantic acyclicity on graph databases. In *PODS* 2013.

[19] C.L. Barrett, R. Jacob, M.V. Marathe. Formal-language-constrained path problems. *SIAM J. on Comp.*, 30(3), pages 809–837, 2000.

[20] J.M. Berstel. *Transductions and Context-Free Languages*. B. G. Teubner, 1979.

[21] A. Blumensath, E. Grädel. Automatic structures. In *LICS* 2000, pages 51-62.

[22] M. Bojanczyk, A. Muscholl, Th. Schwentick, L. Segoufin. Two-variable logic on data trees and XML reasoning. *JACM* 56(3), 2009.

[23] M. Bojanczyk. Automata for data words and data trees. In *RTA*, 2010.

[24] P. Buneman. Semistructured data. In *PODS* 1997, pages 117-121.

[25] P. Buneman, W. Fan, S. Weinstein. Path constraints in semistructured databases. *JCSS* 61(2), pages 146-193, 2000.

[26] P. Buneman, M. F. Fernandez, D. Suciu. UnQL: A query language and algebra for semistructured data based on structural recursion. *VLDB J.* 9(1), pages 76-110, 2000.

[27] D. Calvanese, G. de Giacomo, M. Lenzerini, M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In *KR* 2000, pages 176-185.

[28] D. Calvanese, G. de Giacomo, M. Lenzerini, M. Y. Vardi. Rewriting of regular expressions and regular path queries. *JCSS*, 64(3), pages 443-465, 2002.

[29] D. Calvanese, G. de Giacomo, M. Lenzerini, M. Y. Vardi. View-based query containment. In *PODS* 2003, pages 56-67.

[30] D. Calvanese, G. de Giacomo, M. Lenzerini, M. Y. Vardi. Reasoning on regular path queries. *SIGMOD Record* 32(4), pages 83-92, 2003.

[31] P. Chambart, Ph. Schnoebelen. Post embedding problem is not primitive recursive, with applications to channel systems. In *FSTTCS* 2007, pages 265-276.

[32] A. Chandra and P. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC* 1977, pages 77–90.

[33] M. P. Consens, A. O. Mendelzon. Expressing structural hypertext queries in GraphLog. In *Hypertext* 1989, pages 269-292.

[34] M. P. Consens, A. O. Mendelzon. GraphLog: a visual formalism for real life recursion. In *PODS* 1990, pages 404-416.

[35] M. P. Consens, A. O. Mendelzon. Low complexity aggregation in graphLog and datalog. *TCS* 116(1 & 2), pages 95-116, 1993.

[36] I. Cruz, A. Mendelzon, P. Wood. A graphical query language supporting recursion. In *SIGMOD* 1987, pages 323-330.

[37] L3S dblp bibliography DB: `http://dblp.l3s.de/d2r/`.

[38] S. DeRose. J. Clark. Xml path language (xpath). W3C Recommendation, November 1999, http://www.w3.org/TR/xpath.

[39] A. Deutsch, V. Tannen. Optimization properties for classes of conjunctive regular path queries. In *DBPL* 2001, pages 21-39.

[40] A. Dries, S. Nijssen, L. De Raedt. A query language for analyzing networks. In *CIKM* 2009, pages 485-494.

[41] C. Elgot, J. Mezei. On relations defined by generalized finite automata. *IBM Journal of Research and Development*, 9(1), pages 47-68, 1965.

[42] W. Fan. Graph pattern matching revised for social network analysis. In *ICDT* 2012, pages 8-21.

[43] M. F. Fernández, D. Florescu, A. Y. Levy, D. Suciu. Declarative specification of web sites with Strudel. *VLDB J.* 9(1), pages 38-55, 2000.

[44] M. F. Fernandez, D. Suciu. Optimizing regular path expressions using graph schemas. In *ICDE* 1998, pages 14-23.

[45] G. H. L. Fletcher, M. Gyssens, D. Leinders, J. Van den Bussche, D. Van Gucht, S. Vansummeren, Y. Wu. Relative expressive power of navigational querying on graphs. In *ICDT* 2011, pages 197-207.

[46] D. Florescu, A. Y. Levy, D. Suciu. Query containment for conjunctive queries with regular expressions. In *PODS* 1998, pages 139-148.

[47] D. D. Freydenberger, D. Reidenbach. Bad news on decision problems for patterns. *Inf. Comput.* 208(1), pages 83-96, 2010.

[48] D. D. Freydenberger, N. Schweikardt. Expressiveness and static analysis of extended conjunctive regular path queries. In *AMW* 2011.

[49] Ch. Frougny, J. Sakarovitch. Rational relations with bounded delay. In *STACS* 1991, pages 50-63.

[50] G. Grahne, A. Thomo. Query containment and rewriting using views for regular path queries under constraints. In *PODS* 2003, pages 111-122.

[51] R. Greenlaw, J. Hoover, W. Ruzzo. *Limits to parallel computation: P-completeness theory*. Oxford University Press, 1995.

[52] D. Gusfield. *Algorithms on strings, trees and sequences: Computer science and computational biology*. Cambridge University Press, 1997.

[53] R. H. Güting. GraphDB: Modeling and querying graphs in databases. In *VLDB* 1994, pages 297-308.

[54] M. Gyssens, J. Paredaens, J. Van den Bussche, D. Van Gucht. A graph-oriented object database model. *IEEE Trans. Knowl. Data Eng.* 6(4), pages 572-586, 1994.

[55] D. Harel, D. Kozen, J. Tiuryn. *Dynamic Logic*. MIT Press, 2000.

[56] S. Harris, A. Seaborne. SPARQL 1.1 query language. W3C working draft. `http://www.w3.org/TR/sparql11-query/`, July 2012.

[57] J. Hellings, B. Kuijpers, J. Van den Bussche, X. Zhang. Walk logic as a framework for path query languages on graph databases. In *ICDT* 2013, pages 117-128.

[58] D.A. Holland, U. Braun, D. Maclean, K.K. Muniswamy-Reddy, M.I. Seltzer. Choosing a data model and query language for provenance. In *IPAW* 2008, pages 98-115.

[59] Infinite graph. `http://objectivity.com`

[60] T. Jiang, A. Salomaa, K. Salomaa, S. Yu. Decision problems for patterns. *JCSS* 50(1), pages 53-63, 1995.

[61] M. Kaminski, N. Francez. Finite memory automata. *TCS*, 134(2), pages 329-363, 1994.

[62] K. Kochut, M. Janik. SPARQLeR: Extended Sparql for semantic association discovery. In *ESWC* 2007, pages 145-159.

[63] Z. Lacroix, H. Murthy, F. Naumann, L. Raschid. Links and paths through life sciences data Sources. In *DILS* 2004, pages 203-211.

[64] A. LaPaugh, Ch. Papadimitriou. The even path problem for graphs and digraphs. *Networks* 14(4), pages 507-513, 1984.

[65] L. Libkin, D. Vrgoč. Regular path queries on graphs with data. In *ICDT* 2012, pages 74-85.

[66] L. Libkin, W. Martens, D. Vrgoč. Querying graph databases with XPath. In *ICDT* 2013.

[67] K. Losemann, W. Martens. The complexity of evaluating path expressions in SPARQL. In *PODS* 2012, pages 101-112.

[68] N. Martínez-Bazan, V. Muntés-Mulero, S. Gomez-Villamor, J. Nin, M. Sánchez-Martínez, J. L. Larriba-Pey. Dex: high-performance exploration on large graphs for information retrieval. In *CIKM* 2007, pages 573-582.

[69] A. Mendelzon, P. Wood. Finding regular simple paths in graph databases. *SIAM J. Comput.* 24(6), pages 1235-1258, 1995.

[70] Neo4j. `http://www.neo4j.org/`

[71] F. Neven, Th. Schwentick, V. Vianu. Finite state machines for strings over infinite alphabets. *ACM TOCL* 5(3), pages 403-435, 2004.

[72] Ch. Papadimitriou, M. Yannakakis. On the complexity of database queries. In *PODS* 1997, pages 12-19.

[73] J. Paredaens, P. Peelman, L. Tanca. G-Log: A graph-based query language. *IEEE Trans. Knowl. Data Eng.* 7(3), pages 436-453, 1995.

[74] J. Pérez, M. Arenas, C. Gutierrez. nSPARQL: A navigational language for RDF. *Journal of Web Semantics* 8(4), pages 255-270, 2010.

[75] J. Reutter. Containment of nested regular expressions. `http://arxiv.org/abs/1304.2637`

[76] R. Ronen, O. Shmueli. SoQL: A language for querying and creating data in social networks. In *ICDE* 2009, pages 1595-1602.

[77] A. Salomaa. Patterns. *Bulletin of the EATCS* 54, pages 194-206, 1994.

[78] M. San Martín, C. Gutierrez, P. T. Wood. SNQL: A social networks query and transformation language. In *AMW* 2011.

[79] L. J. Stockmeyer, A. R. Meyer. Word problems requiring exponential time: Preliminary report. In *STOC* 1973, pages 1-9.

[80] M. Y. Vardi. The complexity of relational query languages. In *STOC* 1982, pages 137-146.

[81] M.Y. Vardi. On the complexity of bounded variable queries. In *PODS* 1995, pages 266-276.

[82] G. Weikum, G. Kasneci, M. Ramanath, F. M. Suchanek. Database and information-retrieval methods for knowledge discovery. *CACM* 52(4), pages 56-64, 2009.

[83] M. Yannakakis. Algorithms for acyclic database schemes. In *VLDB* 1981, pages 82-94.