Querying Best Paths in Graph Databases*†

Jakub Michaliszyn[1], Jan Otop[2], and Piotr Wieczorek[3]

1 **Institute of Computer Science, University of Wrocław, Poland**
jmi@cs.uni.wroc.pl

2 **Institute of Computer Science, University of Wrocław, Poland**
jotop@cs.uni.wroc.pl

3 **Institute of Computer Science, University of Wrocław, Poland**
piotrek@cs.uni.wroc.pl

Abstract

Querying graph databases has recently received much attention. We propose a new approach to this problem, which balances competing goals of expressive power, language clarity and computational complexity. A distinctive feature of our approach is the ability to express properties of minimal (e.g. shortest) and maximal (e.g. most valuable) paths satisfying given criteria. To express complex properties in a modular way, we introduce labelling-generating ontologies. The resulting formalism is computationally attractive – queries can be answered in non-deterministic logarithmic space in the size of the database.

1998 ACM Subject Classification H.2.4 Query processing, H.2.3 Query Languages

Keywords and phrases graph databases, queries, aggregation

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.43

1 Introduction

Graphs are one of the most natural representations of data in a number of important applications such as modelling transport networks, social networks, technological networks (see surveys [2, 39, 5]). The main strength of graph representations is the possibility to naturally represent not only the data itself, but also the links among data. Effective search and analysis of graphs is an important factor in reasoning performed in various AI tasks. This motivates the study of query formalisms for graph databases, which are capable of expressing properties of paths.

One of the most challenging problems of recent years is to process big data, typically too large to be stored in the modern computers' memory. This stimulates a strong interest in algorithms working in logarithmic space w.r.t. the size of the database (data complexity) [11, 4, 7]. At the same time, even checking whether there is a path between two given nodes is already NL-complete, so NL is the best complexity for any expressive graph query language.

Our contribution. We propose a new approach to writing queries for graph databases, in which labelling-generating ontologies are first-class citizens. It can be integrated with many existing query formalisms. However, in order to make the presentation clear we introduce the concept by defining a new language OPRA. OPRA features NL-data complexity, good expressive power and a modular structure. The expressive power of OPRA strictly subsumes

* This work has been supported by Polish National Science Center grant UMO-2014/15/D/ST6/00719.
† The full version of this work is available as [33], https://arxiv.org/abs/1710.04419.

the expressive power of popular existing formalisms with same complexity (see Fig. 1). Distinctive properties expressible in OPRA are based on aggregation of data values along paths and computation of extremal values among aggregated data. One example of such a property is "p is a path from s to t that has both the minimal weight and the minimal length among all paths from s to t".

To ease the presentation, we define OPRA in two steps. First, we define the language PRA, whose main components are three types of constraints: P*ath*, R*egular* and A*rithmetical*. We use path constraints to specify endpoints of graph paths; the other constraints only specify properties of paths. Regular constraints specify paths using regular expressions, adapted to deal with multiple paths and infinite alphabets. Arithmetical constraints compare linear combinations of aggregated values, i.e., values of *labels* accumulated along whole paths.

The language PRA can only aggregate and compare the values of labelling functions already defined in the graph. The properties we are interested in often require performing some arithmetical operations on the labellings, either simple (taking a linear combination) or complicated (taking minimum, maximum, or even computing some subquery). Such operations are often nested inside regular expressions (as in LARE [23]) making queries unnecessarily complicated. Instead, similarly as in [3] we specify such operations in a modular way as *ontologies*. This leads to the language OPRA, which comprises Ontologies and PRA. In our approach all knowledge on graph nodes is encoded by labellings, and our ontologies also are defined as the auxiliary labellings. For example, having a labelling child(x, y) stating that x is a child of y, we can define a labelling descendant(x, y) stating that x is a descendant of y. Such labellings can be computed on-the-fly during the query evaluation.

Related work. *Regular Path Queries (RPQs)* [15, 12] are usually used as a basic construction for graph querying. RPQs are of the form $x \to^{\pi} y \wedge \pi \in L(e)$ where e is a (standard) regular expression. Such queries return pairs of nodes (v, v') connected by a path π such that the labelling of π forms a word from $L(e)$. *Conjunctive Regular Path Queries (CRPQs)* are the closure of RPQs under conjunction and existential quantification [14, 32]. Barcelo et al., [7] introduced *extended CRPQs (ECRPQs)* that can compare tuples of paths by *regular relations* [19, 22]. Examples of such relations are path equality, length comparisons, prefix (i.e., a path is a prefix of another path) and fixed edit distance. Regular relations on tuples of paths can be defined by the standard regular expressions over alphabet of tuples of edge symbols.

Graph nodes often store *data values* from an infinite alphabet. In such graphs, paths are interleaved sequences of data values and edge labels. This is closely related to *data words* studied in XML context [34, 18, 37, 10]. Data complexity of query answering for most of the formalisms for data words is NP-hard [30]. This is not, however, the case for *register automata* [26], which inspired Libkin and Vrgoč to define *Regular Queries with Memory (RQMs)* [30]. RQMs are again of the form $x \to^{\pi} y \wedge \pi \in L(e)$. However, e is now *Regular Expression with Memory (REM)*. REMs can store in a register the data value at the current position and test its equality with other values already stored in registers. Register Logic [6] is, essentially, the language of REMs closed under Boolean combinations, node, path and register-assignment quantification. It allows for comparing data values in different paths. The positive fragment of Register Logic, RL^+, has data complexity in NL, even when REMs can be nested using a branching operator. Walk Logic [25] extends FO with path quantification and equality tests of data values on paths. Query answering for WL is decidable but its data complexity is not elementary [6]. LARE [23] is a query language that can existentially quantify nodes and paths, and check relationship between many paths. Path relationships are defined by regular expressions with registers that allow for various arithmetic operations on registers.

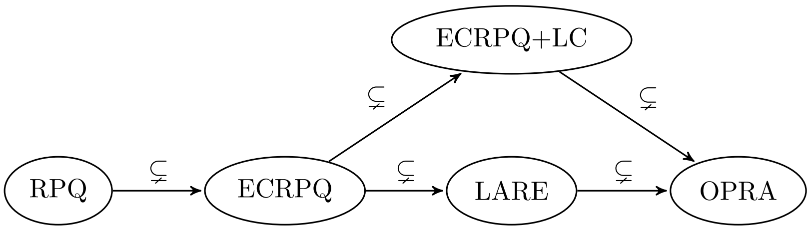

Figure 1 Comparison between different query languages.

Aggregation. Ability to use aggregate functions such as sum, average or count is a fundamental mechanism in database systems. Klug [28] extended the relational algebra and calculus with aggregate functions and proved their equivalence. Early graph query languages G^+ [16] or GraphLog [14, 13] can aggregate data values. Consens and Mendelzon [13] studied *path summarization*, i.e., summarizing information along paths in graphs. They assumed natural numbers in their data model and allowed to aggregate summarization results. In order to achieve good complexity (in the class NC) they allowed aggregate and summing operators that form a closed semiring. Other examples of aggregation can be found in [39].

Summing vectors of numbers along graph paths have been already studied in the context of various formalisms based on automata or regular expressions and lead to a number of proposals that have combined complexity in PSPACE and data complexity in NL. Kopczyński and To [29] have shown that *Parikh images* (i.e., vectors of letter counts) for the usual finite automata can be expressed using a union of linear sets that is polynomial in the size of the automaton and exponential in the alphabet size (the alphabet size, in our context, corresponds to the dimension of vectors). Barcelo et al. [7] extended ECRPQs with linear constraints on the numbers of edge labels counts along paths. They expressed the constraints using reversal-bounded counter machines, translated further to Presburger arithmetic formulas of a polynomial size and evaluate them using techniques from [29, 36].

Figueira and Libkin [20] studied *Parikh automata* introduced in [27]. These are finite automata that additionally store a vector of *counters* in $\mathbb{N}^k$. Each transition specifies also a vector of natural numbers. While moving along graph paths according to a transition the automaton adds this transition's vector to the vector of counters. The automaton accepts if the computed vector of counters is in a given semilinear set in $\mathbb{N}^k$. Also a variant of regular expressions capturing the power of these automata is shown. This model has been used to define a family of variants of CRPQs that can compare tuples of paths using *synchronization languages* [21]. This is a relaxation of regularity condition for relations on paths of ECRPQs and leads to more expressive formalisms with data complexity still in NL. These formalisms are incomparable to ours since they can express non-regular relations on paths like suffix but cannot express properties of data values, nodes' degrees or extrema.

Cypher [38] is a practical query language implemented in the graph database Neo4j. It uses *property graphs* as its data model. These are graphs with labelled nodes and edges, but edges and nodes can also store attribute values for a set of *properties*. `MATCH` clause of Cypher queries allows for specifying graph patterns that depend on nodes' and edges' labels as well as on their properties values. Cypher does not allow full regular expressions however graph patterns can contain transitive closure over a single label. More on Cypher can be found in a survey [2].

RDF [17] is a W3C standard that allows encoding of the content on the Web in a form of a set of *triples* representing an edge-labelled graph. Each triple consists of the subject s, the predicate p, and the object o that are resource identifiers (URI's), and represents an edge from s to o labelled by p. Interestingly, the middle element, p, may play the role of the first or the third element of another triple. Our formalism OPRA allows to operate directly on RDF without any complex graph encoding, by using a ternary labelling representing RDF triples. This allows for convenient navigation by regular expressions in which also the middle element of a triple can serve as the source or the target of a single navigation step (cf. [31]). The standard query formalism for RDF is SPARQL [35, 24]. It implements *property paths* which are RPQs extended with inverses and limited form of negation (see survey [2]).

2 Language OPRA

Various kinds of data graphs are possible and presented in the literature. The differences typically lie in the way the elements of graphs are labelled – both nodes and edges may be labelled by finite or infinite alphabets, which may have some inner structure. Here, we choose a general approach in which a *labelled graph*, or simply a graph, is a tuple consisting of a finite number of *nodes* V and a number of labelling functions $\lambda : V^l \to \mathbb{Z} \cup \{-\infty, \infty\}$ assigning integers to vectors of nodes of some fixed size. While edges are not explicitly mentioned, if needed, one can consider an *edge* labelling λ_E such that $\lambda_E(v, v')$ is 1 if there is an edge from v to v' and it is 0 otherwise. More sophisticated edges, e.g., with integer labels, may be handled by means of standard embedding, defined in Section 4. For convenience, we assume that the set of nodes always contains a distinguished node $\Box$ – we use it as a "sink node", to avoid problems with paths of different lengths.

A *path* is a sequence of nodes. For a path $p = v_1 \dots v_k$, by $p[i]$ we denote its i-th element, v_i, if $i \leq k$, and $\Box$ otherwise.

2.1 Basic constructs

We first define the language PRA, which is the core of the language OPRA. The queries of PRA are of the form

MATCH NODES $\vec{x}$, PATHS $\vec{\pi}$
SUCH THAT Path_constraints
WHERE Regular_constraints
HAVING Arithmetical_constraints

where $\vec{x}$ are free node variables, $\vec{\pi}$ are free path variables, `Path_constraints` is a conjunction of path constraints, `Regular_constraints` is a conjunction of *regular constraints* and, finally, `Arithmetical_constraints` is a conjunction of *arithmetical constraints*, as defined below. The constraints may contain variables not listed in the `MATCH` clause (which are then existentially quantified).

Path constraints. Path constraints are expressions of the form $x_s \to^{\pi} x_t$, where x_s, x_t are node variables and π is a path variable, satisfied if π is a sequence of nodes starting from x_s and ending in x_t.

Regular constraints. The main building blocks of regular constraints are *node constraints*. Syntactically, a k-node constraint is an expression containing free node variables $@_1, @'_1, \dots, @_k, @'_k$ and of the form $X \sim X'$, where $\sim \in \{\leq, <, =\}$ and each of X, X' is an integer constant or a labelling function λ_i applied to some of the free variables.

A k-node constraint for a regular constraint over k paths may be seen as a function that takes a vector containing two nodes of each path: a *current node* (represented by $@_i$) and a *next node* (represented by $@'_i$), and returns a Boolean value. The semantics is given by applying the appropriate labelling functions to the nodes given as an input and comparing the value according to the $\sim$ symbol.

A regular constraint $R(\pi_1, \dots, \pi_k)$ is syntactically a regular expression over an infinite alphabet consisting of all the possible k-node constraints. Assume $p_1, \dots, p_k$ are paths and let $w_1 \dots w_s$ be the word such that $s = \max(|p_1|, \dots, |p_k|)$ and each $w_i \in V^{2k}$ is defined as $w_i = (p_1[i], p_1[i+1], \dots, p_k[i], p_k[i+1])$, i.e., it is a vector consisting of i-th and $i+1$-th node of each path (or can be substituted by $\Box$ if not present). We say that $p_1, \dots, p_k$ *satisfy* R, denoted as $R(p_1, \dots, p_k)$, if the $w_1 \dots w_s$ belongs to the language $L^G(R)$, defined inductively in the usual manner:

- $L^G(R)$, where R is a node constraint, is defined as a set of vectors of length $2k$ for which the constraint R returns true.
- $L^G(R \cdot R') = \{a \cdot b \mid a \in L^G(R) \wedge b \in L^G(R')\}$.
- $L^G(R + R')$ is the union of $L^G(R)$ and $L^G(R')$.
- $L^G(R^*)$ is the Kleene-star closure of $L^G(R)$.

Note that acccording to the definitions above, the variables $@_1, @'_1, \dots, @_k, @'_k$ in a regular constraint $R(\pi_1, \dots, \pi_k)$ always refer to the nodes of the paths $\pi_1, \dots, \pi_k$, e.g., $@_4$ refers to the current node of the path π_4 and $@'_2$ refers to the next node of the path π_2. In order not to mix the variables with ordinary ones we disallow to use them in any other context.

Arithmetical constraints. An arithmetical constraint is an inequality $c_1\Lambda_1 + \ldots + c_j\Lambda_j \leq c_0$, where $c_0, \dots, c_j$ are integer constants and each Λ_ℓ is an expression of the form $\lambda_i[\pi_{i_1}, \dots, \pi_{i_k}]$. The value of $\lambda_i[\pi_{i_1}, \dots, \pi_{i_k}]$ over paths $p_1, \dots, p_n$ is defined as the sum $\sum_{i=1}^{s} \lambda_i(p_{i_1}[i], \dots, p_{i_k}[i])$, where $s = \max\{|p_1|, \dots, |p_k|\}$, i.e., the sum of the labelling for vectors of nodes on corresponding positions of all paths. Paths $\vec{p}$ satisfy the arithmetical expression $c_1\Lambda_1 + \ldots + c_j\Lambda_j \leq c_0$ if the value of the left hand side, with $\vec{\pi}$ instantiated to $\vec{p}$, is less than or equal to c_0.

Query semantics. Let $Q(\vec{x}, \vec{p})$ be a PRA query, and $\vec{x}'$ and $\vec{\pi}'$ be node and path variables in Q that are not listed as free. We say that nodes $\vec{v}$ and paths $\vec{p}$ (of some graph G) satisfy Q, denoted as $Q(\vec{v}, \vec{p})$, if and only if there exist nodes $\vec{v}'$ and paths $\vec{p}'$ such that the instantiation $\vec{x} = \vec{v}$, $\vec{x}' = \vec{v}'$, $\vec{\pi} = \vec{p}$ and $\vec{\pi}' = \vec{p}'$ satisfies all constraints in Q.

2.2 Auxiliary labelling

We introduce a way of defining auxiliary labellings of graphs, which are defined based on existing graph labellings and its structure. The ability to define auxiliary labellings significantly extends the expressive power of the language. The essential property of auxiliary labellings is that their values do not need to be stored in the database, which would require polynomial memory, but can be computed *on demand*. An auxiliary labelling may be seen as an *ontology* or a *view*.

We assume a set $\mathcal{F}$ of *fundamental functions* $f : (\mathbb{Z} \cup \{-\infty, \infty\})^* \to \mathbb{Z} \cup \{-\infty, \infty\}$ consisting of *aggregate functions* maximum Max, minimum Min, counting Count, summation Sum, and *binary* functions $+$, $-$, $\cdot$ and $\leq$ (assuming 0 for false and 1 for true, and that these functions return 0 if the number of inputs is not two). $\mathcal{F}$ can be extended, if needed, by any functions computable by a non-deterministic Turing machine whose size of all tapes while computing $f(\vec{x})$ is logarithmic in length of $\vec{x}$ and values in $\vec{x}$, assuming binary representation, provided that additional aggregate functions in $\mathcal{F}$ are invariant under permutation of arguments.

Terms. In order to specify values for auxiliary labellings we use *terms*. A term $t(\vec{x})$ is defined by the following BNF:

$$t(\vec{x}) ::= c \mid \lambda(\vec{y}) \mid [Q(\vec{y})] \mid \min_{\lambda,\pi} Q(\vec{y},\pi) \mid \max_{\lambda,\pi} Q(\vec{y},\pi)$$
$$\mid y = y \mid f(t(\vec{y}),\ldots,t(\vec{y})) \mid f'(\{t(x)\colon t(x,\vec{y})\})$$

where $\vec{x}$ is a vector of node variables, x is a fresh node variable, c is a constant, λ is a labelling, Q is a PRA query in G, $f \in \mathcal{F}$, $f' \in \mathcal{F}$ is aggregate, $\vec{y}$ ranges over vectors of variables among $\vec{x}$ and y ranges over variables among $\vec{x}$.

Let G be a graph. A *variable instantiation* $\eta^G : \vec{x} \to V$ in G is a function that maps variables in $\vec{x}$ to nodes of G. Such a function extends canonically to subvectors of $\vec{x}$. Below we inductively extend variable instantiations to terms. If G is clear from the context, we write $t(\vec{v})$ as a shorthand of $\eta^G(t(\vec{x}))$, where $\eta^G(\vec{x}[i]) = \vec{v}[i]$ for all i.

1. $\eta^G(c) = c$, where c is a constant,
2. $\eta^G(\lambda(\vec{y})) = \lambda(\eta^G(\vec{y}))$, where λ is a labelling of G
3. $\eta^G([Q(\vec{y})])$ is 1 if $Q(\eta^G(\vec{y}))$ holds in G and 0 otherwise,
4. $\eta^G(\min_{\lambda,\pi} Q(\vec{y},\pi))$ is the minimum of values of $\lambda[p]$, defined as in the arithmetical constraints, over all paths p such that $Q(\eta^G(\vec{y}),p)$ holds in G,
5. $\eta^G(\max_{\lambda,\pi} Q(\vec{y},\pi)) = \max(\{\lambda[p] \mid Q(\eta^G(\vec{y}),p)\})$,
6. $\eta^G(y = y')$ is 1 if $\eta^G(y) = \eta^G(y')$ and 0 otherwise,
7. $\eta^G(f(t_1(\vec{y}_1),\ldots,t_k(\vec{y}_k)) {=} f(\eta^G(t_1(\vec{y}_1),\ldots,\eta^G(t_k(\vec{y}_k))),$
8. $\eta^G(f(\{t(x){:}t'(x,\vec{y})\})) = f(t(v_1),\ldots,t(v_n))$, where $v_1,\ldots,v_n$ are all nodes v of G satisfying $t'(v,\eta^G(\vec{y})) = 1$.

Auxiliary labellings. Consider a term $t(\vec{x})$ and a graph G, which does not have a labelling λ. We define the graph $G[\lambda{:=}t]$ as the graph G extended with the labelling λ such that $\lambda(\vec{v}) = t(\vec{v})$ for all $\vec{v} \in V^k$. We call λ an *auxiliary labelling* of G. We write $G[\lambda_1{:=}t_1, \ldots, \lambda_n{:=}t_n]$ to denote the results of successively adding labellings $\lambda_1,\ldots,\lambda_n$ to the graph G, i.e., $G[\lambda_1{:=}t_1][\lambda_2{:=}t_2]\ldots[\lambda_n{:=}t_n]$.

Language OPRA. An OPRA query is an expression of the form `LET` O `IN` Q', where Q' is a PRA query, O is of the form $\lambda_1{:=}t_1,\ldots,\lambda_n{:=}t_n$ and $t_1,\ldots,t_n$ are terms. The query Q holds over graph G, nodes $\vec{v}$ and paths $\vec{p}$, denoted as $Q(\vec{v},\vec{p})$, if and only if $Q'(\vec{v},\vec{p})$ holds over $G[O]$. Note that $Q'(\vec{x})$ can refer to auxiliary labellings $\lambda_1,\ldots,\lambda_n$. The size of Q is the sum of binary representations of terms $t_1,\ldots,t_n$ and the size of query Q'.

3 Examples

We focus on the following scenario: a graph database that corresponds to a map of some area. Each graph's node represents either a *place* or a *link* from one place to another. The graph has four unary labellings and one binary labelling. The labelling λ_{type} represents the type of a place for places (e.g., square, park, pharmacy) or the mode of transport for links (e.g., walk, tram, train); we assume each type is represented by a constant, e.g., c_{square}, c_{park}. The labelling λ_{attr} represents attractiveness (which may be negative, e.g., in unsafe areas), and λ_{time} represents time. The binary labelling λ_{E} represents edges: for nodes v_1, v_2, the value $\lambda_{\text{E}}(v_1,v_2)$ is 1 if there is an edge from v_1 to v_2 and 0 otherwise. For example, the graph on Fig. 2 represents a map with two places: S is a square and P is a park. There are three nodes representing links: node W represents moving from S to P by walking, T moving from S to P by tram and B moving from P to S by bus.

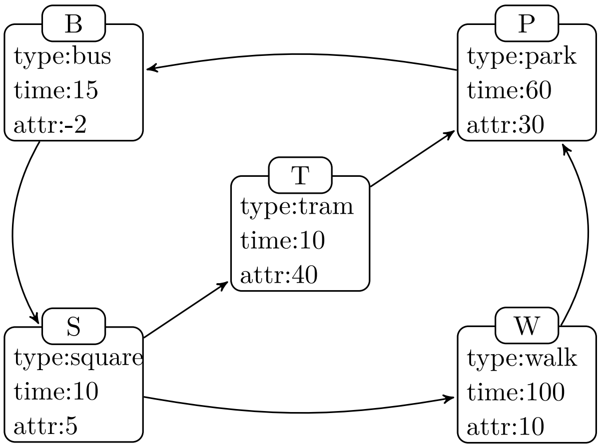

Figure 2 An example of a map-representing graph.

3.1 Language PRA

We begin with the query $Q_{\text{route}}(s, t, \pi)$ stating that there is a path π from a node s to a node t such that each pair of consecutive nodes on this path is connected by an edge given by the edge labelling λ_{E}. Recall that our path constraints of the form $s \rightarrow^{\pi} t$ require only that π is a sequence of nodes that starts at s and ends at t. It does not depend on any labelling, in particular λ_{E}. We introduce a regular constraint route(π) defined as $\langle \lambda_{\text{E}}(@_1, @'_1) = 1\rangle^* \langle \top \rangle(\pi)$ that states that any two consecutive nodes on π satisfy λ_{E}. As the last node has no successor (i.e., $@'_1 = \Box$ for the last node), the constraint ends with $\langle \top \rangle$ that is always satisfied. Then, we can express $Q_{\text{route}}(s, t, \pi)$ as

`MATCH NODES` (s, t) `SUCH THAT` $s \rightarrow^{\pi} t$ `WHERE` route(π)

Sums. The language PRA can express properties of paths' sums. For example, the query below holds iff there is a route from s to t that takes at most 6 hours and its attractiveness is over 100.

`MATCH NODES` (s, t) `SUCH THAT` $s \rightarrow^{\pi} t$ `WHERE` route(π)
`HAVING` $\lambda_{\text{time}}[\pi] \leq 360 \wedge \lambda_{\text{attr}}[\pi] > 100$

Furthermore, we can compute averages, to some extent. For example, the following arithmetical constraint says that for some path π the average attractiveness of π is at least 4 attractiveness points per minute: $\lambda_{\text{attr}}[\pi] \geq 4\lambda_{\text{time}}[\pi]$.

Multiple paths. We define a query that asks whether there is a route from s to t, such that from every place we can take a tram (e.g., if it starts to rain). We express that by stipulating a route π from s to t and a sequence ρ of tram links, such that every node of π representing a place is connected with the corresponding tram link in ρ. In a way, ρ works as an existential quantifier for nodes of π.

`MATCH NODES` (s, t) `SUCH THAT` $s \rightarrow^{\pi} t$
`WHERE` route$(\pi) \wedge \langle \lambda_{\text{type}}(@_1) = c_{\text{tram}} \rangle^*(\rho) \wedge \text{Link}(\pi, \rho)$

where $\text{Link} = (\langle \lambda_{\text{type}}(@_1) = c_{\text{bus}} \rangle + \langle \lambda_{\text{type}}(@_1) = c_{\text{walk}} \rangle + \langle \lambda_{\text{type}}(@_1) = c_{\text{tram}} \rangle + \langle \lambda_{\text{E}}(@_1, @_2) = 1 \rangle)^*$ states that every node of the first path either is not a place, i.e, it represents any of possible links (by a bus, a walk or a tram), or is connected with the corresponding node of the second path. Note also that in the regular constraint $\langle \lambda_{\text{type}}(@_1) = c_{\text{tram}} \rangle^*(\rho)$ the variable $@_1$ represents the current node of the path ρ, whereas, in $\text{Link}(\pi, \rho)$ the variable $@_1$ represents the current node of π, and $@_2$ represents the current node of ρ.

3.2 Language OPRA

We show how to employ auxiliary labellings in our queries. For readability, we introduce some syntactic sugar – constructions which do not change the expressive power of OPRA, but allow queries to be expressed more clearly. We use the function symbols $=, \neq$ and Boolean connectives, which can be derived from $\leq$ and arithmetical operations. Also, we use terms $t(x, y)$ in arithmetical constraints, which can be expressed by first defining the labelling $\lambda_{\mathrm{t}}(x, y){:=}t(x, y)$, defining additional paths $\rho_1 = x, \rho_2 = y$ of length 1, and using $\lambda_{\mathrm{t}}[\rho_1, \rho_2]$.

Processed labellings. Online route planners often allow to look for routes which do not require much walking. The following query asks whether there exists a route from s to t such that the total walking time is at most 10 minutes. To express it, we define a labelling $\lambda_{\mathrm{t\_walk}}(x)$, which is the time of x for x that are walking links, and 0 otherwise.
`LET` $\lambda_{\mathrm{t\_walk}}(x) := (\lambda_{\mathrm{type}}(x) = c_{\mathrm{walk}}) \cdot \lambda_{\mathrm{time}}(x)$ `IN`
`MATCH NODES` (s, t) `SUCH THAT` $s \rightarrow^{\pi} t$
`WHERE` $\mathrm{route}(\pi)$ `HAVING` $\lambda_{\mathrm{t\_walk}}[\pi] \leq 10$

Nested queries. It is often advisable to avoid crowded places, which are usually the most attractive places. We write a query that holds for routes that are always at least 10 minutes away from any node with attractiveness greater than 100. We define a labelling $\lambda_{\mathrm{crowded}}(x)$ as
`[MATCH NODES` (x) `SUCH THAT` $x \rightarrow^{\pi} y$ `WHERE`
$\mathrm{route}(\pi) \wedge \langle \top \rangle^{*} \langle \lambda_{\mathrm{attr}}(@_1) > 100 \rangle(\pi)$ `HAVING` $\lambda_{\mathrm{time}}[\pi] \leq 10$`]`
Notice that π and y are existentially quantified. We check whether the value of $\lambda_{\mathrm{crowded}}$ is 0 for each node of the path π.
`MATCH PATHS` (π) `WHERE` $\mathrm{route}(\pi) \wedge \langle \lambda_{\mathrm{crowded}}(@_1) = 0 \rangle^{*}(\pi)$

Nodes' neighbourhood. "Just follow the tourists" is an advice given quite often. With OPRA, we can verify whether it is a good advice in a given scenario. A route is called *greedy* if at every position, the following node on the path is the most attractive successor. We define a labelling $\lambda_{\mathrm{MAS}}(x, y)$ that is 1 if y is the most attractive successor of x, and 0 otherwise: $\lambda_{\mathrm{E}}(x, y) \wedge (\textsc{Count}(\{\lambda_{\mathrm{attr}}(z) \colon \lambda_{\mathrm{E}}(x, z) \wedge \lambda_{\mathrm{attr}}(z){\geq}\lambda_{\mathrm{attr}}(y)\}){=}1)$. We express that there is a greedy route from s and t.
`MATCH NODES` (s, t) `SUCH THAT` $s \rightarrow^{\pi} t$
`WHERE` $\langle \lambda_{\mathrm{MAS}}(@_1, @_1') = 1 \rangle^{*} \langle \top \rangle(\pi)$

Properties of paths' lengths. In route planning, we often have to balance time, money, attractions, etc. The following query asks whether is it possible to get from s to t in a shortest time possible, in the same time maximising the attractiveness of the route. Recall that Q_{route} is a PRA query defined in Subsection 3.1.
`MATCH NODES` (s, t) `SUCH THAT` $s \rightarrow^{\pi} t$ `WHERE` $\mathrm{route}(\pi)$
`HAVING` $(\lambda_{\mathrm{attr}}[\pi] = \max_{\lambda_{\mathrm{attr}}, \rho} Q_{\mathrm{route}}(s, t, \rho)) \wedge$
$(\lambda_{\mathrm{time}}[\pi] = \min_{\lambda_{\mathrm{time}}, \rho} Q_{\mathrm{route}}(s, t, \rho))$

Registers. Registers are an important concept often in graph query languages. For instance, to express that two paths have a non-empty intersection, we load a (non-deterministically picked) node from the first path to a register and check whether it occurs in the second path. The following query asks whether there exists a route from a club s to a club t on which the attractiveness of visited clubs never decreases. In the register-based approach, we achieve this by storing the most recently visited club in a separate register. Here, we express this register using an additional path ρ, storing the values of the register, and a labelling $\lambda_{\mathrm{r}}(x', y, y')$ which states that $y' = x'$ if x' is a club, and $y' = y$ otherwise, defined as $(\lambda_{\mathrm{type}}(x') = c_{\mathrm{club}} \Rightarrow y' = x') \wedge (\lambda_{\mathrm{type}}(x') \neq c_{\mathrm{club}} \Rightarrow y = y')$.

`MATCH NODES` (s, t) `SUCH THAT` $s \rightarrow^{\pi} t \wedge s \rightarrow^{\rho} t$

`WHERE` $\mathrm{route}(\pi) \wedge \mathrm{ends}(\pi) \wedge \mathrm{regs}(\pi, \rho) \wedge \mathrm{inc}(\rho)$

where $\mathrm{ends} = \langle\lambda_{\mathrm{type}}(@_1) = c_{\mathrm{club}}\rangle\langle\top\rangle^*\langle\lambda_{\mathrm{type}}(@_1) = c_{\mathrm{club}}\rangle$ states that the both ends of a path are clubs, $\mathrm{regs} = \langle\lambda_{\mathrm{r}}(@_1', @_2, @_2') = 1\rangle^*\langle\top\rangle$ ensures that at each position the second path contains the most recently visited club along the first path, and $\mathrm{inc} = \langle\lambda_{\mathrm{attr}}(@_1) \leq \lambda_{\mathrm{attr}}(@_1')\rangle^*\langle\top\rangle$ checks that the attractiveness never decreases.

4 Expressive power

We compare the expressive power of OPRA and other query languages for graph databases from the literature. We prove the results depicted in Figure 1: that OPRA subsumes ECRPQ, ECRPQ with linear constraints [7] and LARE [23] query languages.

These query languages assume a different notion of graphs from the one considered in this paper. We call graphs as defined in these papers *data graphs*. A *data graph* is a tuple $G = \langle V, E, \lambda\rangle$ where V is a finite set of nodes, $E \subseteq V \times \Sigma \times V$ is a set of edges labelled by a finite alphabet Σ, and $\lambda : V \rightarrow \mathbb{Z}^K$ is a labelling of nodes by vectors of K integers. A path in G is a sequence of interleaved nodes and edge labels $v_0 e_1 v_1 \ldots v_k$ such that for every $i < k$ we have $E(v_i, e_{i+1}, v_{i+1})$.

The difference between graphs and data graphs is mostly syntactical, yet it prevents us from comparing directly the languages of interest. To overcome this problem, we define the *standard embedding*, which is a natural transformation of data graphs to graphs. For a data graph $G = \langle V, E, \lambda\rangle$ with edges labelled by Σ and nodes labelled by $\mathbb{Z}^K$, we define the graph $G^E = (V^E, \lambda_1^E, \ldots, \lambda_K^E, \lambda_{K+1}^E)$, called the standard embedding of G, such that (1) $V^E = V \cup \Sigma$, (2) for every $i \in \{1, \ldots, K\}$ and every $v \in V$ we have $\lambda_i^E(v)$ equal to the i-th component of $\lambda(v)$, (3) for every $i \in \{1, \ldots, K\}$ and every $v \in \Sigma$ we have $\lambda_i^E(v) = 0$, and (4) for all $v_1, v_2, v_3 \in V^E$ we have $\lambda_{K+1}^E(v_1, v_2, v_3) = 1$ if $(v_1, v_2, v_3) \in E$ and $\lambda_{K+1}^E(v_1, v_2, v_3) = 0$ otherwise. Observe that every node v (resp., every path p) in G corresponds to the unique node v^E (resp., path p^E) in G^E.

A query Q_1 on data graphs is equivalent w.r.t. the standard embedding, *se-equivalent* for short, to a query Q_2 on graphs if for all data graphs G, nodes $\vec{v}$ in G and paths $\vec{p}$ query $Q_1(\vec{v}, \vec{p})$ holds in G if and only if $Q_2(\vec{v}^E, \vec{p}^E)$ holds in G^E, where $G^E, \vec{v}^E, \vec{p}^E$ result from the standard embedding of respectively $G, \vec{v}, \vec{p}$. We say that OPRA *subsumes* a query language $\mathcal{L}$ if every query in $\mathcal{L}$ can be transformed in polynomial time to an se-equivalent OPRA query.

LARE. Queries in LARE are built from arithmetical regular expressions, which extend regular expressions with registers storing nodes and arithmetical operations on labels of the nodes stored in registers (which are natural numbers). We briefly discuss how to express the three main building blocks of LARE expressions: *edge constraints* specifying labels of edges, *register constraints* specifying values of registers, and *register assignments* specifying how registers change.

OPRA queries over the standard embedding can specify labels of edges in the original graph and hence can express edge constraints. Next, we arithmetize all logical operations assuming *true*:= 1 and *false*:= 0. With that, we can show by structural induction that OPRA labellings can express register constraints (e.g., construction $C \vee C'$ can be expressed by term $\max(t_C, t_{C'})$). Finally, we can express registers with additional paths as discussed in Section 3.2.

OPRA language is stronger than LARE. The set of (vectors of) paths satisfying a given LARE query is regular. Therefore, for a fixed LARE query $Q(\pi_1, \pi_2)$, we can decide whether $\forall \pi_1 \exists \pi_2 Q(\pi_1, \pi_2)$ holds in a given graph G in polynomial space in G. The set of paths satisfying an OPRA query Q is also related to automata, but due to linear constraints we can express properties of weighted automata. We can define an OPRA query $Q^U(\pi_1, \pi_2)$ which interprets the input graph G as a weighted automaton, path π_1 as an input word and π_2 as a run r on π_1; query Q^U holds only if the value of r is at most 0. Then, $\forall \pi_1 \exists \pi_2 Q(\pi_1, \pi_2)$ holds only if the value of every word (w.r.t. the weighted automaton corresponding to G) is at most 0. Such a problem for weighted automata is called (quantitative) universality problem and it is undecidable [1]. Therefore, checking whether a given graph G satisfies $\forall \pi_1 \exists \pi_2 Q(\pi_1, \pi_2)$ is undecidable. Thus, no LARE query is se-equivalent to Q^U.

▶ **Theorem 1.** *(1) OPRA subsumes LARE. (2) There is an OPRA query Q with no LARE query Q' se-equivalent to Q.*

ECRPQ with linear constraints. ECRPQ has been extended with linear constrains (ECRPQ+LC) [7], expressing that a given vectors paths $\vec{\pi}$ satisfying a given ECRPQ query satisfies linear inequalities, which specify the multiplicity of edge labels in various components of $\vec{\pi}$. Language OPRA subsumes LARE, which extends ECRPQ, and hence OPRA subsumes ECRPQ. Linear constraints can be expressed by arithmetical constraints of OPRA and hence OPRA subsumes ECRPQ+LC. Moreover, linear constrains are unaffected by nodes' labels and hence ECRPQ+LC cannot express a PRA query saying "the sum of integer labels of nodes along path p is positive". Thus, we have the following.

▶ **Theorem 2.** *(1) OPRA subsumes ECRPQ+LC. (2) There is an OPRA query Q with no ECRPQ+LC query Q' se-equivalent to Q.*

5 The query answering problem

The query-answering problem asks, given an OPRA $Q(\vec{x}, \vec{\pi})$, a graph G, nodes $\vec{v}$ and paths $\vec{p}$ of G, whether $Q(\vec{v}, \vec{p})$ holds in G. We are interested in the *data complexity* of the problem, where the size of a query is treated as constant, and *combined complexity*, where there is no such restriction.

To obtain the desired complexity results, we assume that the absolute values of the graph labels are polynomially bounded in the size of a graph. This allows us to compute arithmetical relations on these labels in logarithmic space. Without such a restriction, the data complexity of the query-answering problem we study is NP-hard by a straightforward reduction from the knapsack problem.

We state the complexity bounds as follows.

▶ **Theorem 3.** *The query answering problem for OPRA queries with bounded number of auxiliary labellings is* PSPACE*-complete and its data complexity is* NL*-complete.*

The emptiness problem (whether there exist nodes $\vec{v}$ and paths $\vec{p}$ such that a given OPRA query Q holds for $\vec{v}, \vec{p}$ in a given graph G) has the same complexity; this follows from the fact that a query $Q(\vec{x}, \vec{\pi})$ is non-empty in G iff $Q(\epsilon, \epsilon)$ (same query without free variables) holds in G.

The lower bounds in Theorem 3 follow from the PSPACE-hardness of ECRPQ [7], as discussed in Section 4, and for the NL-hardness of the reachability problem.

Recall that an OPRA query is of the form `LET` O `IN` Q', where Q' is a PRA query, O is of the form $\lambda_1{:=}t_1, \ldots, \lambda_n{:=}t_n$ and $t_1, \ldots, t_n$ are terms. Also, by $|O|$ we denote the number of labellings defined in O. The upper bound in Theorem 3 follows directly from the following lemma.

▶ **Lemma 4.** *For every fixed $s \geq 0$, we have:*

(1) *Given a graph G and an OPRA query Q := `LET` O `IN` Q' such that $|O| \leq s$, we can decide whether Q holds in G in non-deterministic polynomial space in Q and non-deterministic logarithmic space in G.*

(2) *Given a graph G and an OPRA query Q := `LET` O `IN` Q' such that $|O| \leq s$, we can compute $\min_{\lambda,\pi} Q(\vec{y}, \pi)$ (resp., $\max_{\lambda,\pi} Q(\vec{y}, \pi)$) non-deterministically in polynomial space in Q and logarithmic space in G. The computed value is either polynomial in G and exponential in Q, or $-\infty$ (resp., ∞).*

We first prove the upper bounds for PRA (i.e., for $s = 0$), and then extend the results to OPRA.

5.1 Language PRA

Assume a PRA query Q = `MATCH NODES` $\vec{x}$, `PATHS` $\vec{\pi}$ `SUCH THAT` P `WHERE` R `HAVING` A. We prove the results in two steps. First, we construct a Turing machine of a special kind (later on called QAM) that represents graphs, called *answer graphs*, with distinguished initial and final nodes, such that every path from an initial node to a final node in this graph is an encoding of paths that satisfy constraints P and R of Q in graph G (for some instantiation of variables $\vec{x}$). These graphs are augmented with the computed values of expressions that appear in arithmetical constraints A. Then, we prove that checking whether in an answer graph there is a path from an initial node to a final node that encodes a path in G satisfying A can be done within desired complexity bounds. Deriving values for $\vec{x}$ from computed paths is straightforward.

The first step is an adaptation of the technique commonly used in the field, e.g., in [7, 23]. We encode vectors $(p_1, \ldots, p_n)$ of paths of nodes from some V as a single path $p_1 \otimes \ldots \otimes p_n$ over the product alphabet V^n (shorter paths are padded with $\Box$).

Answer graphs. Consider a graph G with nodes V, its paths $\vec{p}$ and an OPRA query Q = `MATCH NODES` $\vec{x}$, `PATHS` $\vec{\pi}$ `SUCH THAT` P `WHERE` R `HAVING` $\bigwedge_{i=1}^{m} A_i \leq c_i$ with $|\vec{p}| = |\vec{\pi}|$ and existentially quantified path variables $\vec{\pi}'$. Let $k = |\vec{\pi}| + |\vec{\pi}'|$. The *answer graph* for Q on G, $\vec{p}$ is a triple (G', S, T), where $S, T \subseteq M_Q \times V^k$;

- G' is a graph with nodes $M_Q \times V^k$, where M_Q is a finite set computed from Q;
- for each $i \leq m$ and a node $(s, v_1, \ldots, v_k) \in M_Q \times V^k$, the labelling $\lambda_i((s, v_1, \ldots, v_k))$ is defined as the value of the arithmetic constraint A_i over single-node paths $v_1, \ldots, v_k$; and
- a path $q = q_1 \otimes \cdots \otimes q_k$ from (a node in) S to (a node in) T is such that $\lambda_E(v, v') = 1$ for all consecutive v, v' of q iff the paths $q_1, \ldots, q_k$ satisfy $P \wedge R$ and $(q_1, \ldots, q_{|\pi|}) = \vec{p}$.

Intuitively, the labelling λ_E can be defined in such a way that along paths of G' the V^k-components of the nodes correctly encode paths of Q satisfying the path constraints P and the M_Q components store valid runs of automata corresponding to the regular constraints R.

QAMs. Answer graphs can be represented (on-the-fly) in logarithmic space. A Query Applying Machine (QAM) is a non-deterministic Turing Machine which works in logarithmic space and only accepts inputs encoding tuples of the form (G, t, w), where G is a graph and t is a symbol among V, λ, S, T.

For a graph G and $k \geq 0$, a QAM M *gives* a graph $G_k^M = (V, \lambda_1, \ldots, \lambda_k)$ and sets of nodes S_G^M, T_G^M such that:

- V consists of all the *nodes* v s.t. M accepts on (G, V, v),
- λ_i is such that $\lambda_i(\vec{v}) = n$ iff M accepts on $(G, \lambda, (i, \vec{v}, n))$
- S_G^M (resp., T_G^M) consists of $v \in V$ such that M accepts on (G, S, v) (resp., (G, T, v)).

For soundness, we require that for each G, i and $\vec{v}$ there is exactly one n such that M accepts on $(G, \lambda, (i, \vec{v}, n))$.

▶ **Lemma 5.** *For a given query Q and paths $\vec{p}$, we can construct in polynomial time a QAM M^Q such that for every graph G, machine M^Q gives an answer graph for Q on $G, \vec{p}$.*

The second step amounts to the following lemma.

▶ **Lemma 6.** *Let G be a graph and M^Q a QAM, such that M^Q gives the graph $G_m^{M^Q}$ and the sets $S_G^{M^Q}$ and $T_G^{M^Q}$. Let Π be the set of paths from $S_G^{M^Q}$ to $T_G^{M^Q}$ satisfying $\bigwedge_{i=1}^m \lambda_i[\pi] \leq c_i$ in $G_m^{M^Q}$. Checking emptiness of Π can be done non-deterministically in polynomial space in Q and logarithmic space in G.*

To solve the emptiness problem from Lemma 6, we need to solve the weighted reachability problem for the corresponding answer graph, which is a graph labelled by vectors of integers. This problem is equivalent to the $\mathbb{Z}$-reachability problem for vector addition systems with states (VASS) [9]. The latter problem has polynomial size solutions. The answer graph is exponential in Q and polynomial in G. Therefore, if Π from Lemma 6 is non-empty, then it contains a path of exponential size in Q and polynomial size in G, and hence its existence can be verified non-deterministically in polynomial space in Q and logarithmic space in G.

5.2 Language OPRA

Assume $O = \lambda_1{:=}t_1, \ldots, \lambda_s{:=}t_s$. We show by induction on s that the values of the labellings of a graph $G[O]$ can be non-deterministically computed in space polynomial in O.

▶ **Lemma 7.** *Let s be fixed. For a graph G and $O = \lambda_1{:=}t_1, \ldots, \lambda_s{:=}t_s$, the value of each labelling of $G[O]$ can be non-deterministically computed in polynomial space in O and logarithmic space in G.*

The proof studies all the possible constructors of terms. In the case of subqueries, we apply the inductive assumption, i.e., Theorem 3 for a query with fewer auxiliary labellings. The case of minimum follow from a counterpart of Lemma 6 for the problem of computing $\min_{p\in\Pi} \lambda_i[p]$, which can be proved by deducing from [8] that $\min_{p\in\Pi} \lambda_i[p]$ is either $-\infty$, $+\infty$ or exponential in Q and polynomial in G. Therefore, it can be computed using Lemma 6 and the bisection method. The case for the maximum is symmetric. Finally, application of a function symbol to terms or ranges can be implemented in the expected complexity.

Now we give some intuition for the proof of Lemma 4. To solve the query answering problem for OPRA, for a given query `LET` O `IN` Q' and a graph G, we first build a QAM $M^{Q'}$, as in Lemma 5. $M^{Q'}$ may refer to labellings from O, not defined in G. We change it so that whenever it wants to access a value of one of labellings defined in O, it instead runs a procedure guaranteed by Lemma 7. Finally, we use Lemma 6 to determine the result.

6 Conclusions

We defined a new query language for graph databases, OPRA and demonstrated its expressive power in two ways. We presented examples of natural properties and OPRA queries expressing them in an organized, modular way. We showed that OPRA strictly subsumes query languages ECRPQ+LC and LARE. Despite additional expression power, the complexity of the query-answering problem for OPRA matches the complexity for ECRPQ+LC and LARE.

References

1. Shaull Almagor, Udi Boker, and Orna Kupferman. What's decidable about weighted automata? In *Automated Technology for Verification and Analysis*, pages 482–491. Springer, 2011.
2. Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan Reutter, and Domagoj Vrgoč. Foundations of Modern Graph Query Languages. *CoRR, abs/1610.06264*, 2016. arXiv:1610.06264.
3. Marcelo Arenas, Georg Gottlob, and Andreas Pieris. Expressive languages for querying the semantic web. In *PODS 2014*, pages 14–26. ACM, 2014.
4. Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev. DL-lite in the light of first-order logic. In *Proceedings of the national conference on artificial intelligence*, volume 22, page 361. AAAI Press; MIT Press, 2007.
5. Pablo Barceló. Querying graph databases. In *Proceedings of the 32nd Symposium on Principles of Database Systems (PODS13)*, pages 175–188, 2013.
6. Pablo Barceló, Gaëlle Fontaine, and Anthony W. Lin. Expressive path queries on graph with data. *Logical Methods in Comp. Sci.*, 11(4), 2015.
7. Pablo Barceló, Leonid Libkin, Anthony W. Lin, and Peter T. Wood. Expressive languages for path queries over graph-structured data. *ACM Transactions on Database Systems*, 37:31:1–31:46, 2012.
8. Michael Blondin, Alain Finkel, Stefan Göller, Christoph Haase, and Pierre McKenzie. Reachability in two-dimensional vector addition systems with states is PSpace-complete. *CoRR*, abs/1412.4259, 2014.
9. Michael Blondin, Alain Finkel, Stefan Göller, Christoph Haase, and Pierre McKenzie. Reachability in two-dimensional vector addition systems with states is pspace-complete. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 32–43. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.14.
10. Mikołaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Log.*, 12(4):27, 2011.
11. Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Data complexity of query answering in description logics. In *Proceedings 10th International Conference on Principles of Knowledge Representation and Reasoning (KR06)*, volume 6, pages 260–270, 2006.
12. Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Containment of conjunctive regular path queries with inverse. In *Proceedings of the 7th International Conference Principles of Knowledge Representation and Reasoning (KR00)*, pages 176–185, 2000.
13. Mariano Consens and Alberto Mendelzon. Low complexity aggregation in GraphLog and Datalog. *Theor. Comp. Sci.*, 116(1):95–116, 1993.
14. Mariano P. Consens and Alberto O. Mendelzon. Graphlog: a visual formalism for real life recursion. In Daniel J. Rosenkrantz and Yehoshua Sagiv, editors, *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, April 2-4, 1990, Nashville, Tennessee, USA*, pages 404–416. ACM Press, 1990. doi:10.1145/298514.298591.

15 Isabel F. Cruz, Alberto O. Mendelzon, and Peter T. Wood. A graphical query language supporting recursion. In *Proc. of the ACM Special Interest Group on Management of Data (SIGMOD87)*, pages 323–330, 1987.

16 Isabel F. Cruz, Alberto O. Mendelzon, and Peter T. Wood. G+: recursive queries without recursion. In *Expert Database Conf.*, pages 645–666, 1988.

17 Richard Cyganiak, Markus Lanthaler, and David Wood. RDF 1.1 concepts and abstract syntax. W3C Recommendation, 2014. URL: `http://www.w3.org/TR/2014/REC-rdf11-concepts/`.

18 Stéphane Demri, Ranko Lazic, and David Nowak. On the freeze quantifier in Constraint LTL: Decidability and complexity. *Inf. Comput.*, 205(1):2–24, 2007.

19 C.C. Elgot and J.E. Mezei. On relations defined by generalized finite automata. *IBM J. Res. Dev.*, 9(1):47–68, 1965.

20 Diego Figueira and Leonid Libkin. Path logics for querying graphs: Combining expressiveness and efficiency. In *Proceedings of the 30th Annual Symposium on Logic in Computer Science (LICS15)*, pages 329–340, 2015.

21 Diego Figueira and Leonid Libkin. Synchronizing relations on words. *Theory Comput. Syst.*, 57(2):287–318, 2015.

22 Christiane Frougny and Jacques Sakarovitch. Synchronized rational relations of finite and infinite words. *Theor. Comput. Sci.*, 108(1):45–82, 1993. `doi:10.1016/0304-3975(93)90230-Q`.

23 Michał Graboń, Jakub Michaliszyn, Jan Otop, and Piotr Wieczorek. Querying data graphs with arithmetical regular expressions. In *Proceedings of the 25rd International Joint Conference on Artificial Intelligence (IJCAI16)*. AAAI Press, 2016. URL: `http://www.ii.uni.wroc.pl/~jmi/papers/ijcai16.pdf`.

24 Steve Harris and Andy Seaborne. SPARQL 1.1 Query Language. W3C Recommendation, 2013. URL: `https://www.w3.org/TR/sparql11-query/`.

25 Jelle Hellings, Bart Kuijpers, Jan Van den Bussche, and Xiaowang Zhang. Walk logic as a framework for path query languages on graph databases. In *Proceedings of the 16th International Conference on Database Theory (ICDT13)*, pages 117–128, 2013.

26 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.

27 Felix Klaedtke and Harald Rueß. Monadic second-order logics with cardinalities. In *Automata, Languages and Programming, 30th International Colloquium, ICALP 2003*, pages 681–696, 2003.

28 Anthony C. Klug. Equivalence of relational algebra and relational calculus query languages having aggregate functions. *J. ACM*, 29(3):699–717, 1982.

29 Eryk Kopczyński and Anthony Widjaja To. Parikh images of grammars: Complexity and applications. In *Proceedings of the 25th Symposium on Logic in Comp. Sci. (LICS10)*, pages 80–89, 2010.

30 Leonid Libkin, Wim Martens, and Domagoj Vrgoč. Querying graphs with data. *J. ACM*, 63(2):14:1–14:53, 2016.

31 Leonid Libkin, Juan L. Reutter, and Domagoj Vrgoc. Trial for RDF: adapting graph query languages for RDF data. In *PODS 2013*, pages 201–212, 2013.

32 Alberto Mendelzon and Peter Wood. Finding regular simple paths in graph databases. *SIAM Journal on Computation*, 24(6):1235–1258, 1995.

33 Jakub Michaliszyn, Jan Otop, and Piotr Wieczorek. Querying best paths in graph databases. *CoRR*, abs/1710.04419, 2017. URL: `http://arxiv.org/abs/1710.04419`.

34 Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004.

35 Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. W3C Recommendation, 2008. URL: http://www.w3.org/TR/rdf-sparql-query/.
36 Bruno Scarpellini. Complexity of subcases of Presburger arithmetic. *Transactions of the American Mathematical Society*, 284(1):203–218, 1984.
37 Luc Segoufin. Automata and logics for words and trees over an infinite alphabet. In *Proceedings of the 20th Int. Workshop on Computer Science Logic (CSL06)*, pages 41–57, 2006.
38 The Neo4j Team. The Neo4j Manual v3.1., 2017. URL: http://neo4j.com/docs/.
39 Peter T. Wood. Query languages for graph databases. *SIGMOD Record*, 41(1):50–60, 2012.