Nesting Depth of Operators in Graph Database Queries: Expressiveness vs. Evaluation Complexity*†

M. Praveen[1] and B. Srivathsan[2]

1 Chennai Mathematical Institute, Chennai, India
2 Chennai Mathematical Institute, Chennai, India

Abstract

Designing query languages for graph structured data is an active field of research, where expressiveness and efficient algorithms for query evaluation are conflicting goals. To better handle dynamically changing data, recent work has been done on designing query languages that can compare values stored in the graph database, without hard coding the values in the query. The main idea is to allow variables in the query and bind the variables to values when evaluating the query. For query languages that bind variables only once, query evaluation is usually NP-complete. There are query languages that allow binding inside the scope of Kleene star operators, which can themselves be in the scope of bindings and so on. Uncontrolled nesting of binding and iteration within one another results in query evaluation being PSPACE-complete.

We define a way to syntactically control the nesting depth of iterated bindings, and study how this affects expressiveness and efficiency of query evaluation. The result is an infinite, syntactically defined hierarchy of expressions. We prove that the corresponding language hierarchy is strict. Given an expression in the hierarchy, we prove that it is undecidable to check if there is a language equivalent expression at lower levels. We prove that evaluating a query based on an expression at level i can be done in level i of the polynomial time hierarchy. Satisfiability of quantified Boolean formulas can be reduced to query evaluation; we study the relationship between alternations in Boolean quantifiers and the depth of nesting of iterated bindings.

1998 ACM Subject Classification F.1.3 Complexity measures and classes: Complexity hierarchies, F.4.3 Formal languages: Classes defined by grammars or automata, H.2.3 Languages: Query languages

Keywords and phrases graphs with data, regular data path queries, expressiveness, query evaluation, complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2016.117

1 Introduction

Graph structures representing data have found many applications like semantic web [11], social networks [19] and biological networks [13]. Theoretical models of such data typically have a graph with nodes representing entities and edges representing relations among them. One reason for the popularity of these models is their flexibility in handling semi-structured data. While traditional relational databases impose rigid structures on the relations between

* A full version of this paper is available at `http://arxiv.org/abs/1603.00658`.
† Both the authors are partially funded by a grant from Infosys Foundation.

43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016).
Editors: Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi;
Article No. 117; pp. 117:1–117:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

data elements, graph databases are better equipped to handle data in which relations are not precisely known and/or developing dynamically.

A fundamental query language for such models is Regular Path Queries (RPQs), which is now part of the W3C recommendation [18]. An RPQ consists of a regular expression. Suppose a communication network is modeled by a graph, where nodes represent servers and edges labeled ℓ represent links between them. Evaluating the RPQ ℓ^* on this graph results in the set of pairs of nodes between which there exists a route. Suppose each link has a priority and we need pairs of connected nodes where all intermediate links have the same priority. We can hard code the set of priorities in the query. If the set of priorities is not static, a querying mechanism which avoids hard coding is better. Every edge can be labeled by a supplementary data value (priority of the link, in this example) and we want query languages that can compare data values without hard coding them in the syntax. Nodes can also carry data values. In generic frameworks, there is no a priori bound on the number of possible data values and they are considered to be elements of an infinite domain. Graph databases with data values are often called data graphs in theory and property graphs in practice.

One way to design querying languages for data graphs is to extend RPQs using frameworks that handle words on infinite alphabets [16, 15, 12, 23]. Expressiveness and efficient algorithms for query evaluation are conflicting goals for designing such languages. We study a feature common to many of these languages, and quantify how it affects the trade-off between expressiveness and complexity of query evaluation. Variable finite automata [10] and parameterized regular expressions [2] are conservative extensions of classical automata and regular expressions. They have variables, which can be bound to letters of the alphabet at the beginning of query evaluation. The query evaluation problem is NP-complete for these languages. Regular expressions with binding (REWBs) [15] is an extended formalism where binding of variables to values can happen inside a Kleene star, which can itself be in the scope of another binding operator and so on. Allowing binding and iteration to occur inside each other's scope freely results in the query evaluation problem being PSPACE-complete. Here we study how the expressiveness and complexity of query evaluation vary when we syntactically control the depth of nesting of iterated bindings.

Contributions

1. We syntactically classify REWBs according to the depth of nesting of iterated bindings.
2. The resulting hierarchy of data languages is strict, and so is the expressiveness of queries.
3. It is undecidable to check if a given REWB has a language equivalent one at lower levels.
4. An REWB query in level i can be evaluated in Σ_i in the polynomial time hierarchy.
5. For lower bounds, we consider quantified Boolean formulas with some restrictions on quantifications and reduce their satisfiability to query evaluation, with some restrictions on the queries.

For proving strictness of the language hierarchy, we build upon ideas from the classic star height hierarchy [9]. Universality of REWBs is known to be undecidable [17, 12]. We combine techniques from this proof with tools developed for the language hierarchy to prove the third result above. The Σ_i upper bound for query evaluation involves complexity theoretic arguments based on the same tools. In the reductions from satisfiability of quantified Boolean formulas to the query evaluation problem, the relation between the number of alternations (in the Boolean quantifiers) and the depth of nesting (of iterated bindings in REWBs) is not straight forward. We examine this relation closely in the framework of parameterized complexity theory, which is suitable for studying the effect of varying the structure of input instances on the complexity.

Related work: The quest for efficient evaluation algorithms and expressive languages to query graph databases, including those with data values, is an active area of research; [1] is a recent comprehensive survey. Numerous formalisms based on logics and automata exist for handling languages over infinite alphabets [20]. In [16], the suitability of these formalisms as query languages has been studied, zeroing in on register automata mainly for reasons of efficient evaluation. The same paper introduced regular expressions with memory and proved that they are equivalent to register automata. REWBs [15] have slightly less expressive power but have better scoping structure for the binding operator. Properties of these expressions have been further studied in [12]. In [14], XPath has been adapted to query data graphs. Pebble automata have been adapted to work with infinite alphabets in [17]. A strict language hierarchy based on the number of pebbles allowed in pebble automata has been developed in [22]. Many questions about comparative expressiveness of register and pebble automata are open [17]. Fixed-point logics can be used to define languages over infinite alphabets [4]. These logics can use the class successor relation, which relates two positions with the same data value if no intermediate position carries the same value. Expressiveness of these logics increase [6, 5], when the number of alternations between standard successor relation and class successor relation increase.

2 Preliminaries

2.1 Data Languages and Querying Data Graphs

We follow the notation of [15]. Let Σ be a finite alphabet and $\mathcal{D}$ a countably infinite set. The elements of $\mathcal{D}$ are called *data values*. A *data word* is a finite string over the alphabet $\Sigma \times \mathcal{D}$. We will write a data word as $\binom{a_1}{d_1}\binom{a_2}{d_2}\ldots\binom{a_n}{d_n}$, where each $a_i \in \Sigma$ and $d_i \in \mathcal{D}$. A set of data words is called a *data language*.

An extension of standard regular expressions, called *regular expressions with binding (REWB)*, has been defined in [15]. Here, data values are compared using variables. For a set $\{x_1, x_2, \ldots, x_k\}$ of variables, the set of conditions $\mathcal{C}_k$ is the set of Boolean combinations of $x_i^{=}$ and $x_i^{\neq}$ for $i \in \{1, \ldots, k\}$. A data value $d \in \mathcal{D}$ and a partial valuation $\nu : \{x_1, \ldots, x_k\} \to \mathcal{D}$ satisfies the condition $x_i^{=}$ (written as $d, \nu \models x_i^{=}$) if $\nu(x_i) = d$. The satisfaction for other Boolean operators are standard.

▶ **Definition 2.1** (Regular expressions with binding (REWB) [15]). Let Σ be a finite alphabet and $\{x_1, \ldots, x_k\}$ a set of variables. Regular expressions with binding over $\Sigma[x_1, \ldots, x_k]$ are defined inductively as: $r ::= \varepsilon \mid a \mid a[c] \mid r + r \mid r \cdot r \mid r^* \mid a \downarrow_x (r)$ where $a \in \Sigma$ is a letter in the alphabet, $c \in \mathcal{C}_k$ is a condition on the variables and $x \in \{x_1, \ldots, x_k\}$ is a variable.

We call $\downarrow_x$ the *binding operator*. In the expression $a \downarrow_x (r)$, the expression r is said to be the *scope* of the binding $\downarrow_x$. A variable x in an expression is *bound* if it occurs in the scope of a binding $\downarrow_x$. Otherwise it is *free*. We write *fv*(r) to denote the set of free variables in r and $r(\bar{x})$ to denote that $\bar{x}$ is the sequence of all free variables. The semantics of an REWB $r(\bar{x})$ over the variables $\{x_1, \ldots, x_k\}$ is defined with respect to a partial valuation $\nu : \{x_1, \ldots, x_k\} \to \mathcal{D}$ of the variables. A valuation ν is *compatible* with $r(\bar{x})$ if $\nu(\bar{x})$ is defined.

▶ **Definition 2.2** (Semantics of REWB). Let $r(\bar{x})$ be an REWB over $\Sigma[x_1, \ldots, x_k]$ and let $\nu : \{x_1, \ldots, x_k\} \to \mathcal{D}$ be a valuation of variables compatible with $r(\bar{x})$. The language of data words $L(r, \nu)$ defined by $r(\bar{x})$ with respect to ν is given as follows:

r	$L(r,\nu)$	r	$L(r,\nu)$	r	$L(r.\nu)$
ε	$\{\varepsilon\}$	a	$\{\binom{a}{d} \mid d \in \mathcal{D}\}$	$a[c]$	$\{\binom{a}{d} \mid d, \nu \models c\}$
$r_1 + r_2$	$L(r_1,\nu) \cup L(r_2,\nu)$	$r_1 \cdot r_2$	$L(r_1,\nu) \cdot L(r_2,\nu)$	r_1^*	$(L(r_1,\nu))^*$
$a \downarrow_{x_i} (r_1)$	$\bigcup_{d \in \mathcal{D}} \{\binom{a}{d}\} \cdot L(r_1, \nu[x_i \to d])$				

where $\nu[x_i \to d]$ denotes the valuation which is the same as ν except for x_i which is mapped to d. An REWB r defines the data language $L(r) = \bigcup_{\nu \text{ compatible with } r} L(r,\nu)$.

For example, the REWB $a \downarrow_x (b[x^=]^*)$ defines the set of data words of the form ab^* with all positions having the same data value. The REWB $(a \downarrow_x (b[x^=]))^*$ defines the set of data words of the form $\binom{a}{d_1}\binom{b}{d_1}\binom{a}{d_2}\binom{b}{d_2}\cdots\binom{a}{d_n}\binom{b}{d_n}$.

▶ **Definition 2.3** (Data graphs). A *data graph* G over a finite alphabet Σ and an infinite set of data values $\mathcal{D}$ is a pair (V, E) where V is a finite set of vertices, and $E \subseteq V \times \Sigma \times \mathcal{D} \times V$ is a set of edges which carry labels from $\Sigma \times \mathcal{D}$.

We do not have data values on vertices, but they can be introduced without affecting the results. A *regular data path query* is of the form $Q = x \xrightarrow{r} y$ where r is an REWB. Evaluating Q on a data graph G results in the set $Q(G)$ of pairs of nodes $\langle u, v\rangle$ such that there exists a data path from u to v and the sequence of labels along the data path forms a data word in $L(r)$. Evaluating a regular data path query on a data graph is known to be PSPACE-complete in general and NLOGSPACE-complete when the query is of constant size [15]. We sometimes identify the query Q with the expression r and write $r(G)$ for $Q(G)$. A query r_1 is said to be contained in another query r_2 if for every data graph G, $r_1(G) \subseteq r_2(G)$. It is known from [12, Proposition 3.5] that a query r_1 is contained in the query r_2 iff $L(r_1) \subseteq L(r_2)$. Hence, if a class E_2 of REWBs is more expressive than the class E_1 in terms of defining data languages, E_2 can also express more queries than E_1.

2.2 Parameterized Complexity

The size of queries are typically small compared to the size of databases. To analyze the efficiency of query evaluation algorithms, the size of the input can be naturally split into the size of the query and the size of the database. Parameterized complexity theory is a formal framework for dealing with such problems. An instance of a parameterized problem is a pair (x, k), where x is an encoding of the input structure on which the problem has to be solved (e.g., a data graph and a query), and k is a parameter associated with the input (e.g., the size of the query). A parameterized problem is said to be in the parameterized complexity class Fixed Parameter Tractable (FPT) if there is a computable function $f : \mathbb{N} \to \mathbb{N}$, a constant $c \in \mathbb{N}$ and an algorithm to solve the problem in time $f(k)|x|^c$.

We will see later that the query evaluation problem is unlikely to be in FPT, when parameterized by the size of the regular data path query. There are many parameterized complexity classes that are unlikely to be in FPT, like W[SAT], W[P], AW[SAT] and AW[P]. To place parameterized problems in these classes, we use FPT-reductions.

▶ **Definition 2.4** (FPT reductions). A FPT reduction from a parameterized problem Q to another parameterized problem Q' is a mapping R such that:

1. For all instances (x, k) of parameterized problems, $(x, k) \in Q$ iff $R(x, k) \in Q'$.
2. There exists a computable function $g : \mathbb{N} \to \mathbb{N}$ such that for all (x, k), say with $R(x, k) = (x', k')$, we have $k' \leqslant g(k)$.
3. There exist a computable function $f : \mathbb{N} \to \mathbb{N}$ and a constant $c \in \mathbb{N}$ such that R is computable in time $f(k)|x|^c$.

3 Nesting Depth of Iterated Bindings and Expressive Power

A binding $\downarrow_x$ along with a condition $[x^=]$ or $[x^{\neq}]$ is used to constrain the possible data values that can occur at certain positions in a data word. A binding inside a star — an *iterated binding* — imposes the constraint arbitrarily many times. For instance, the expression $r_1 := (a_1 \downarrow_{x_1} (b_1[x_1^=]))^*$ defines data words in $(a_1b_1)^*$ where every a_1 has the same data value as the next b_1. We now define a syntactic mechanism for controlling the nesting depth of iterated bindings. The restrictions result in an infinite hierarchy of expressions. The expressions at level i are generated by F_i in the grammar below, defined by induction on i.

$$F_0 ::= \varepsilon \mid a \mid a[c] \mid F_0 + F_0 \mid F_0 \cdot F_0 \mid F_0^*$$
$$E_i ::= F_{i-1} \mid E_i + E_i \mid E_i \cdot E_i \mid a \downarrow_{x_j} (E_i)$$
$$F_i ::= E_i \mid F_i + F_i \mid F_i \cdot F_i \mid F_i^*$$

where $i \geqslant 1$, $a \in \Sigma$, c is a condition in $\mathcal{C}_k$ and $x_j \in \{x_1, \dots, x_k\}$. Intuitively, E_i can add bindings over iterations (occurring in F_{i-1}) and F_i can add iterations over bindings (occurring in E_i). The nesting depth of iterated bindings in an expression in F_i is therefore i. The union of all expressions at all levels equals the set of REWBs. In this paper, we use subscripts to denote the levels of expressions and superscripts to denote different expressions in a level: so e_5^1 is some expression in E_5, f_3^2 is some expression in F_3.

We now give a sequence of expressions $\{r_i\}_{i \geqslant 1}$ such that each r_i is in F_i but no language equivalent expression exists in F_{i-1}. For technical convenience, we use an unbounded number of letters from the finite alphabet and an unbounded set of variables. The results can be obtained with a constant number of letters and variables.

▶ **Definition 3.1.** Let $\{a_1, b_1, a_2, b_2, \dots\}$ be an alphabet and $\{x_1, x_2, \dots\}$ a set of variables. We define r_1 to be $(a_1 \downarrow_{x_1} (b_1[x_1^=]))^*$. For $i \geqslant 2$, define $r_i := (a_i \downarrow_{x_i} (r_{i-1}b_i[x_i^=]))^*$.

From the syntax, it can be seen that each r_i is in F_i. To show that $L(r_i)$ cannot be defined by any expression in F_{i-1}, we will use an "automaton view" of the expression, as this makes pigeon-hole arguments simpler. No automata characterizations are known for REWBs in general; the restrictions on the binding and star operators in the expressions of a given level help us build specific automata in stages.

Standard finite state automata can be converted to regular expressions by considering generalized non-deterministic finite automata, where transitions are labeled with regular expressions instead of a single letter (see e.g., [21, Lemma 1.32]). The language of an expression f_i^1 can be accepted by such an automaton, where transitions are labeled with expressions in E_i. We will denote this automaton by $\mathcal{A}(f_i^1)$. Similarly, the language of an expression e_i^1 can be accepted by an automaton whose transitions are labeled with expressions in F_{i-1} or with $a \downarrow_x$. We will denote this automaton by $\mathcal{A}(e_i^1)$. There are no cycles in $\mathcal{A}(e_i^1)$, since e_i^1 can not use the Kleene $*$ operator except inside expressions in F_{i-1}. The runs of $\mathcal{A}(e_i^1)$ are sequences of pairs of a state and a valuation for variables. The valuations are updated after every transition with a label of the form $a \downarrow_x$. Formal semantics are given in Appendix A of the full version of this paper, which also contains all the proofs in detail.

We will prove that $L(r_i)$ cannot be defined by any expression in E_i (and hence not by any expression in F_{i-1}). We first define the following sequence of words, which will be used in the proof. Let $\{d[j_1, j_2] \in \mathcal{D} \mid j_1, j_2 \in \mathbb{N}\}$ be a set of data values such that $d[j_1, j_2] \neq d[j_1', j_2']$

if $\langle j_1, j_2\rangle \neq \langle j_1', j_2'\rangle$. For every $n \geqslant 1$, define the words:

$$u_{1,n} := \binom{a_1}{d[1,1]}\binom{b_1}{d[1,1]}\binom{a_1}{d[1,2]}\binom{b_1}{d[1,2]} \cdots \binom{a_1}{d[1,n^2]}\binom{b_1}{d[1,n^2]}$$

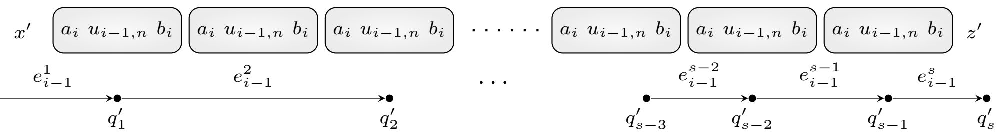
$$u_{i,n} := \binom{a_i}{d[i,1]} u_{i-1,n} \binom{b_i}{d[i,1]}\binom{a_i}{d[i,2]} u_{i-1,n} \binom{b_i}{d[i,2]} \cdots \binom{a_i}{d[i,n^2]} u_{i-1,n} \binom{b_i}{d[i,n^2]}$$

for all $i \geqslant 2$

In order to prove that $L(r_i)$ cannot be defined by any expression in E_i, we will show the following property: if $u_{i,n}$ occurs as a sub-word of a word w in the language of a "sufficiently small" expression e_i^1, then the same expression accepts a word where some a_j and a matching b_j have different data values. Let $\mathit{Mismatch}_{i,n}$ be the set of all data words obtained from $u_{i,n}$ by modifying the data values so that there exist two positions p, p' with $p < p'$ and a $j \leqslant i$ such that: p contains $\binom{a_j}{d}$ and p' contains $\binom{b_j}{d'}$ with $d \neq d'$; moreover between positions p and p', b_j does not occur in the word. We consider expressions in which no two occurrences of the binding operator use the same variable. For an expression e, let $|\mathcal{A}(e)|$ denote the number of states in the automaton $\mathcal{A}(e)$ and $|\mathit{var}(e)|$ denote the number of variables in e.

▶ **Lemma 3.2.** *Let e_i^1 be an expression and let $n \in \mathbb{N}$ be greater than $(|\mathcal{A}(e)| + 1)$ and $(|\mathit{var}(e)| + 1)$ for every sub-expression e of e_i^1. Let ν be a valuation of $\mathit{fv}(e_i^1)$ and let x, z be data words. Then: $xu_{i,n}z \in L(e_i^1, \nu) \implies x\bar{u}_{i,n}z \in L(e_i^1, \nu)$ for some $\bar{u}_{i,n} \in \mathit{Mismatch}_{i,n}$.*

Proof idea. By induction on i. Suppose $xu_{i,n}z \in L(e_i^1, \nu)$. The run of $\mathcal{A}(e_i^1)$ on $xu_{i,n}z$ consists of at most n transitions, since the automaton is acyclic and has at most n states. Each of the (at most) n transitions reads some sub-word in the language of some sub-expression f_{i-1}^1, while the whole word consists of n^2 occurrences of $a_i u_{i-1,n} b_i$. Hence, at least one sub word consists of n occurrences of $a_i u_{i-1,n} b_i$. A run of $\mathcal{A}(f_{i-1}^1)$ on such a sub-word is shown below.



Every transition of this run reads sub-words in the language of some sub-expression e_{i-1}^j. If some transition of this run reads an entire sub-word $u_{i-1,n}$ (as in transition $q_1' \longrightarrow q_2'$), then we can create a mismatch inside this $u_{i-1,n}$ by induction hypothesis. Otherwise, none of the transitions read an a_i and the corresponding b_i together (as in $q_{s-2}' \longrightarrow q_{s-1}'$ in the figure). None of the b_is is compared with the corresponding a_i, so the data value of one of the b_is can be changed to create a mismatch. The resulting data word will be accepted provided the change does not result in a violation of some condition. Since the range of the valuation has at most $(n-1)$ distinct values, one of the n b_is is safe for changing the data value. ◀

▶ **Theorem 3.3.** *For any i, the language $L(r_i)$ cannot be defined by any expression in E_i.*

Proof. Suppose r_i is equivalent to an expression e_i^1. Pick an n bigger than $|\mathcal{A}(e)|$ and $|\mathit{fv}(e)|$ for every sub-expression e of e_i^1. The word $u_{i,n}$ belongs to $L(r_i)$ and hence $L(e_i^1)$. By Lemma 3.2, we know that if this is the case, then $\bar{u}_{i,n} \in L(r_i)$ for some word $\bar{u}_{i,n} \in \mathit{Mismatch}_{i,n}$. But $L(r_i)$ cannot contain words with a mismatch. A contradiction. ◀

Given an expression at some level, it is possible that its language is defined by an expression at lower levels. Next we show that it is undecidable to check this.

▶ **Theorem 3.4.** *Given an expression in F_{i+1}, checking if there exists a language equivalent expression in F_i is undecidable.*

Proof idea. By reduction from Post's Correspondence Problem (PCP). The basic idea is from the proof of undecidability of universality of REWBs and related formalisms [17, 15]. For an instance $\{(u_1, v_1), \ldots, (u_n, v_n)\}$ of PCP, a solution (if it exists) can be encoded by a data word of the form $w_1\#r_i\#w_2$, where w_1 is made up of u_i's, w_2 is made up of v_i's and r_i is from Definition 3.1. To ensure that such a data word indeed represents a solution, we need to match up the u_i's in w_1 with the v_i's in w_2, which can be done through matching data values. Consider the language of data words of the form $w_1'\#r_i\#w_2'$ that are *not* solutions of the given PCP instance. This language can be defined by an expression Δ in E_{i+1}, which compares data values in the left of $\#r_i\#$ with those on the right side, to catch mismatches. We can prove that no equivalent expression exists in lower levels, using techniques used in Lemma 3.2. On the other hand, if the given PCP instance does not have a solution, no data word encodes a solution, so the given language is defined by $\Sigma^* r_i \Sigma^*$, which is in F_i. ◀

4 Complexity of Query Evaluation

In this section, we will study how the depth of nesting of iterated bindings affects the complexity of evaluating queries. An instance of the query evaluation problem consists of a data graph G, an REWB e, a valuation ν for $fv(e)$ and a pair $\langle u, v\rangle$ of nodes in G. The goal is to check if u is connected to v by a data path in $L(e, \nu)$.

4.1 Upper Bounds

An expression in F_i can be thought of as a standard regular expression (without data values) over the alphabet of its sub-expressions. This is the main idea behind our upper bound results. The main result proves that evaluating queries in E_i can be done in Σ_i in the polynomial time hierarchy.

▶ **Lemma 4.1.** *With an oracle for evaluating E_i queries, F_i queries can be evaluated in polynomial time.*

Proof idea. Suppose the query f_i^1 is to be evaluated on the data graph G and f_i^1 consists of the sub-expressions $e_i^1, \ldots, e_i^m$ in E_i. For every j, add an edge labeled e_i^j between those pairs $\langle v_1, v_2\rangle$ of nodes of G for which $\langle v_1, v_2\rangle$ is in the evaluation of e_i^j on G. Evaluating the sub-expressions can be done with the oracle. Now f_i^1 can be treated as a standard regular expression over the finite alphabet $\{e_i^1, \ldots, e_i^m\}$, and can be evaluated in polynomial time using standard automata theoretic techniques. ◀

▶ **Theorem 4.2.** *For queries in E_i, the evaluation problem belongs to Σ_i.*

Proof idea. Since bindings in E_i are not iterated, each binding is performed at most once. The data value for each variable is guessed non-deterministically. The expression can be treated as a standard regular expression over its sub-expressions and the guessed data values. The sub-expressions are in F_{i-1}, which can be evaluated in polynomial time (Lemma 4.1) with an oracle for evaluating queries in E_{i-1}. This argument will not work in general for arbitrary REWBs — bindings that are nested deeply inside iterations and other bindings may occur more than polynomially many times in a single path. ◀

Next we consider the query evaluation problem with the size of the query as the parameter. An instance of the *parameterized weighted circuit satisfiability* problem consists of a Boolean circuit and the parameter $k \in \mathbb{N}$. The goal is to check if the circuit can be satisfied by a truth assignment of weight k (i.e., one that sets exactly k propositional atoms to true). The class W[P] is the set of all parameterized problems which are FPT-reducible to the weighted circuit satisfiability problem.

▶ **Theorem 4.3.** *Evaluating REWB queries in E_1, parameterized by the size of the query is in* W[P]*.*

Proof idea. It is proved in [3, Lemma 7, Theorem 8] that a parameterized problem is in W[P] iff there is a non-deterministic Turing machine that takes an instance (x, k) and decides the answer within $f(k)|x|^c$ steps, of which at most $f(k)\log|x|$ are non-deterministic (for some computable function f and a constant c). Such a Turing machine exists for evaluating REWB queries in E_1, using the steps outlined in the proof idea of Theorem 4.2. ◀

Thus, the number of non-deterministic steps needed to evaluate an E_1 query depends only logarithmically on the size of the data graph. It is also known that W[P] is contained in the class PARA-NP — the class of parameterized problems for which there are deterministic algorithms taking instances (x, k) and computing an equivalent instance of the Boolean satisfiability problem in time $f(k)|x|^c$. Hence, we can get an efficient reduction to the satisfiability problem, on which state of the art SAT solvers can be run. Many hard problems in planning fall into this category [7].

We next consider the parameterized complexity of evaluating queries at higher levels. The parameterized class *uniform*-XNL is the class of parameterized problems Q for which there exists a computable function $f : \mathbb{N} \to \mathbb{N}$ and a non-deterministic algorithm that, given a pair (x, k), decides if $(x, k) \in Q$ in space at most $f(k)\log|x|$ [3, Proposition 18].

▶ **Theorem 4.4.** *Evaluating REWB queries, with size of the query as parameter, is in uniform*-XNL*.*

Proof idea. Let k be the size of the query e_i^1 to be evaluated, on a data graph with n nodes. Suppose a pair of nodes is connected by a data path w in $L(e_i^1)$. Iterations in e_i^1 can only occur inside its F_{i-1} sub-expressions. Hence w consists of at most k sub-paths, each sub-path w_j in the language of some sub-expression f_{i-1}^j. When f_{i-1}^j is considered as a standard regular expression over its sub-expressions (in E_{i-1}), there are no bindings. By a standard pigeon hole principle argument, we can infer that w_j consists of at most kn sub-paths, each one in the language of some sub-expression e_{i-1}^1. This argument can be continued to prove that w is of length at most $(k^2n)^i$. The existence of such a path can be guessed and verified by a non-deterministic Turing machine in space $\mathcal{O}(ik^2\log n)$. ◀

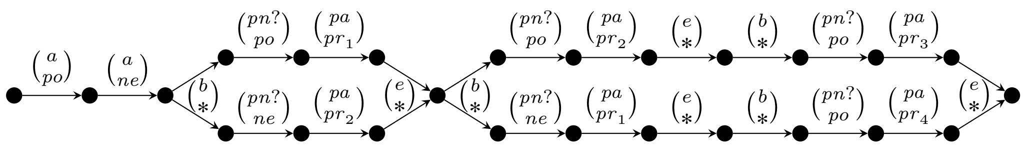
4.2 Lower Bounds

We obtain our lower bounds by reducing various versions of the Boolean formula satisfiability problem to query evaluation. We begin by describing a schema for reducing the problem of evaluating a Boolean formula on a given truth assignment to the problem of evaluating a query on a data graph. The basic ideas for the gadgets we construct below are from [15, proofs of Proposition 2, Theorem 5]. We will need to build on these ideas to address finer questions about the complexity of query evaluation.

Suppose the propositional atoms used in the Boolean formula are among $\{pr_1, \ldots, pr_n\}$. We use $pr_1, \ldots, pr_n$ also as data values. An edge labeled $\binom{pa}{pr_j}$ indicates the propositional atom pr_j occurring in a sub-formula. The data values po and ne appear on edges labeled with the letter $pn?$, to indicate if a propositional atom appears positively or negatively. The symbol $*$ denotes an arbitrary data value different from all others. We will assume that the Boolean formula is in negation normal form, i.e., negation only appears in front of propositional atoms. This restriction does not result in loss of generality, since any Boolean formula can be converted into an equi-satisfiable one in negation normal form with at most linear blowup in the size. The data graph is a series parallel digraph with a source and a sink, defined as follows by induction on the structure of the Boolean formula.

- Positively occurring propositional atom pr_j: $\cdot \xrightarrow{\binom{b}{*}} \cdot \xrightarrow{\binom{pn?}{po}} \cdot \xrightarrow{\binom{pa}{pr_j}} \cdot \xrightarrow{\binom{e}{*}} \cdot$.
- Negatively occurring propositional atom pr_j: $\cdot \xrightarrow{\binom{b}{*}} \cdot \xrightarrow{\binom{pn?}{ne}} \cdot \xrightarrow{\binom{pa}{pr_j}} \cdot \xrightarrow{\binom{e}{*}} \cdot$.
- $\phi_1 \wedge \cdots \wedge \phi_r$: inductively construct the data graphs for the conjuncts, then do a standard serial composition, by fusing the sink of one graph with the source of the next one.
- $\phi_1 \vee \cdots \vee \phi_r$: inductively construct the data graphs for the disjuncts, then do a standard parallel composition, by fusing all the sources into one node and all the sinks into another node.
- After the whole formula is handled, the source of the resulting graph is fused with the sink of the following graph: $\cdot \xrightarrow{\binom{a}{po}} \cdot \xrightarrow{\binom{a}{ne}} \cdot$.

Let G_ϕ denote the data graph constructed above for formula ϕ. The data graph G_ϕ is shown below for $\phi = (pr_1 \vee \neg pr_2) \wedge ((pr_2 \wedge pr_3) \vee (\neg pr_1 \wedge pr_4))$.



The query uses $x_1, \dots, x_k$ to remember the propositional atoms that are set to true.

$$e_{eval}[k] := a \downarrow_{x_{po}} (a \downarrow_{x_{ne}} (\qquad (1)$$
$$(b(pn?[x^{=}_{po}] \cdot pa[x^{=}_1 \vee \cdots \vee x^{=}_k] + pn?[x^{=}_{ne}] \cdot pa[x^{\neq}_1 \wedge \cdots \wedge x^{\neq}_k])e)^* \ )) \ .$$

▶ **Lemma 4.5.** *Let ϕ be a Boolean formula over the propositional atoms $pr_1, \dots, pr_n$ and $\nu : \{x_1, \dots, x_k\} \to \{pr_1, \dots, pr_n, *\}$ be a valuation. The source of G_ϕ is connected to its sink by a data path in $L(e_{eval}[k], \nu)$ iff ϕ is satisfied by the truth assignment that sets exactly the propositions in $\{pr_1, \dots, pr_n\} \cap Range(\nu)$ to true.*

Proof idea. The two bindings in the beginning of $e_{eval}[k]$ forces x_{po}, x_{ne} to contain po, ne respectively. A positively occurring propositional atom generates a data path of the form $\cdot \xrightarrow{\binom{b}{*}} \cdot \xrightarrow{\binom{pn?}{po}} \cdot \xrightarrow{\binom{pa}{pr_j}} \cdot \xrightarrow{\binom{e}{*}} \cdot$, which can only be in the language of the expression $b \cdot pn?[x^{=}_{po}] \cdot pa[x^{=}_1 \vee \cdots \vee x^{=}_k]e$. This forces pr_j to be contained in one of $x_1, \dots, x_k$. Similar arguments works for negatively occurring atoms. Rest of the proof is by induction on the structure of the formula. ◀

▶ **Theorem 4.6.** *For queries in E_1, the evaluation problem is* NP*-hard.*

Proof idea. To check if a Boolean formula ϕ is satisfiable, evaluate the query $a \downarrow_{x_1} a \downarrow_{x_2} \cdots a \downarrow_{x_n} e_{eval}[n]$ on the data graph $\cdot \xrightarrow{\binom{a}{pr_1/*}} \cdot \xrightarrow{\binom{a}{pr_2/*}} \cdots \xrightarrow{\binom{a}{pr_n/*}} \cdot - G_\phi \to \cdot$. Here, $\xrightarrow{\binom{a}{pr_j/*}}$ denotes two edges in parallel, one labeled with $\binom{a}{pr_j}$ and another with $\binom{a}{*}$. ◀

Evaluating queries in E_1 is NP-complete, evaluating REWB queries in general is PSPACE-complete and evaluating queries in E_i is in Σ_i. To prove a corresponding Σ_i lower bound, one would need to simulate Σ_i computations using queries with bounded depth of nesting of iterated bindings. However, this does not seem to be possible. We take a closer look at this in the rest of the paper. Finding the exact complexity of evaluating queries in E_i remains open.

We now extend our satisfiability-to-query evaluation schema to handle Boolean quantifiers. Let $PR = \{pr_1, \ldots, pr_n\}$ be a set of propositional atoms. To handle existential Boolean quantifiers, we build a new graph and a query. These gadgets build on earlier ideas to bring out the difference in the role played by the data graph and the query while reducing satisfiability to query evaluation. The new graph $G[\exists k/PR] \circ G$, is as follows: $\cdot \xrightarrow{\binom{a_1}{pr_1}} \cdot \xrightarrow{\binom{a_1}{pr_2}} \cdots \xrightarrow{\binom{a_1}{pr_n}} \cdot - G \rightarrow \cdot$. We assume that the letter a_1 is not used inside G, which is equal to G_ϕ for some Boolean formula ϕ. The new query $e[\exists k] \circ e$ is defined as follows:

$$e[\exists k] \circ e := a_1^* a_1 \downarrow_{x_1} a_1^* a_1 \downarrow_{x_2} a_1^* \cdots a_1^* a_1 \downarrow_{x_k} a_1^* e \tag{2}$$

where $e = e_{eval}[k]$ for some $k \in \mathbb{N}$.

We now give a parameterized lower bound for evaluating E_1 queries. An instance of the *weighted satisfiability* problem consists of a Boolean formula (not necessarily in Conjunctive Normal Form) and a parameter $k \in \mathbb{N}$. The goal is to check if the formula is satisfied by a truth assignment of weight k. The class W[SAT] is the set of all parameterized problems that are FPT-reducible to the weighted satisfiability problem (see [8, Chapter 25]).

▶ **Lemma 4.7.** *Let ϕ be a Boolean formula over the set PR of propositions and $k \in \mathbb{N}$. We can construct in polynomial time a data graph G and an REWB e_1^1 satisfying the following conditions.*

1. *The source of G is connected to its sink by a data path in $L(e_1^1)$ iff ϕ has a satisfying assignment of weight k.*
2. *The size of e_1^1 depends only on k.*

Proof idea. The required data graph is $G[\exists k/PR] \circ G_\phi$ and e_1^1 is $e[\exists k] \circ e_{eval}[k]$. The data path $\cdot \xrightarrow{\binom{a_1}{pr_1}} \cdot \xrightarrow{\binom{a_1}{pr_2}} \cdots \xrightarrow{\binom{a_1}{pr_n}} \cdot$ in the graph $G[\exists k/PR] \circ G_\phi$ has to be in the language of $a_1^* a_1 \downarrow_{x_1} a_1^* a_1 \downarrow_{x_2} a_1^* \cdots a_1^* a_1 \downarrow_{x_k} a_1^*$. This induces a valuation ν' which maps $\{x_1, \ldots, x_k\}$ injectively into PR, denoting the k propositions that are set to true. With this the data path continues from the source of G_ϕ to its sink. Rest of the proof follows from Lemma 4.5. ◀

▶ **Theorem 4.8.** *Evaluating REWB queries in E_1, parameterized by the size of the query is hard for* W[SAT] *under* FPT *reductions.*

Proof. The reduction given in Lemma 4.7 is a FPT reduction from the weighted satisfiability problem to the problem of evaluating E_1 queries , parameterized by the size of the query. ◀

Finally we extend our gadgets to handle universal Boolean quantifiers. These gadgets build upon the previous ideas and bring out the role of nested iterated bindings when satisfiability is reduced to query evaluation. We would first like to check if the source of some graph G is connected to its sink by a data path in the language of some REWB e, for every possible injective valuation $\nu : \{x_1, \ldots, x_k\} \rightarrow PR$. We will now design some data graphs and expressions to achieve this. Let *skip* be a letter not used in G. The data graphs $G_0, \ldots, G_k$ are as shown in Figure 1.

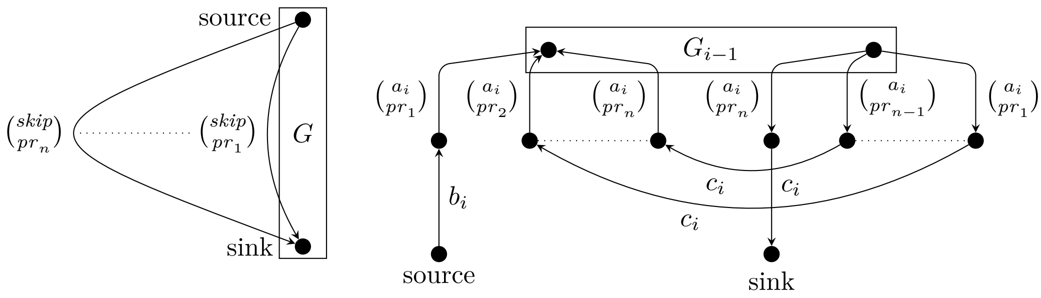


Figure 1 Data graphs G_0 (left) and G_i (right).

The expressions $e^0, \ldots, e^k$ are as follows.

$$e^0 := e + \bigplus_{1 \leqslant i < j \leqslant k} skip[x_i^= \wedge x_j^=] \qquad\qquad e^i := b_i(a_i \downarrow_{x_i} (e^{i-1} a_i[x_i^=])c_i)^* \tag{3}$$

The graph G_0 and the expression e^0 are designed to ensure that the source of G is connected to its sink by a path in $L(e, \nu)$, unless ν is not injective, in which case G can be bypassed by one of the edges labeled $\binom{skip}{pr_j}$ introduced in G_0. The graph G_i and the expression e_i are designed to ensure that any path from the source of G_i to its sink has to go through G_{i-1} multiple times, once for each pr_j stored in the variable x_i. The nesting depth of iterated bindings in the expression e^i is one more than that of e^{i-1}.

Suppose ν is a partial valuation of some variables, whose domain does not intersect with $\{x_1, \ldots, x_k\}$. We denote by $\nu[\{x_1, \ldots, x_k\} \to PR]$ the set of valuations ν' that extend ν such that domain$(\nu') = $ domain$(\nu) \cup \{x_1, \ldots, x_k\}$ and $\{\nu'(x_1), \ldots, \nu'(x_k)\} \subseteq PR$. We additionally require that ν' is injective on $\{x_1, \ldots, x_k\}$ when we write $\nu[\{x_1, \ldots, x_k\} \xrightarrow{1:1} PR]$.

▶ **Lemma 4.9.** *Let $i \in \{1, \ldots, k\}$ and ν_i be a valuation for $fv(e^i) \setminus \{x_1, \ldots, x_i\}$. The source of G_i is connected to its sink by a data path in $L(e^i, \nu_i)$ iff for every $\nu \in \nu_i[\{x_1, \ldots, x_i\} \to PR]$, there is a data path in $L(e^0, \nu)$ connecting the source of G_0 to its sink.*

Proof idea. The data path has to begin with $b_i\binom{a_i}{pr_1}$ in the language of $b_i a_i \downarrow_{x_i}$, forcing x_i to store pr_1. Then the path has to traverse G_{i-1} using e^{i-1}. At the sink of G_{i-1}, the path is forced to take $\binom{a_i}{pr_1}c_i$ to satisfy the condition in $a_i[x_i^=]c_i$. This forces the path to start again in $\binom{a_i}{pr_2}$ and so on. ◀

We write $G[\forall k/PR] \circ G$ and $e[\forall k] \circ e$ to denote the graph G_k and REWB e^k constructed above. We implicitly assume that the variables $x_1, \ldots, x_k$ are not bound inside e. We can always rename variables to ensure this. If e is in E_i, then $e[\forall k] \circ e$ is in F_{i+k-1}.

▶ **Lemma 4.10.** *Let ν be a valuation for $fv(e) \setminus \{x_1, \ldots, x_k\}$ for some REWB e. The source of $G[\forall k/PR] \circ G$ is connected to its sink by a data path in $L(e[\forall k] \circ e, \nu)$ iff for all $\nu' \in \nu[\{x_1, \ldots, x_k\} \xrightarrow{1:1} PR]$, the source of G is connected to its sink by a data path in $L(e, \nu')$.*

Proof idea. Lemma 4.9 ensures that there is a path $w_{\nu'}$ in $L(e^0, \nu')$ connecting the source of G_0 to its sink for every valuation $\nu' \in \nu[\{x_1, \ldots, x_k\} \to PR]$. From Figure 1, $w_{\nu'}$ can either be a *skip* edge, or a path through G. By definition, e^0 allows a skip edge to be taken only when two variables among $x_1, \ldots, x_k$ have the same data value. Hence for valuations ν' that are injective on $\{x_1, \ldots, x_k\}$, $w_{\nu'}$ is in $L(e, \nu')$. ◀

If ϕ is a partially quantified Boolean formula with the propositional atoms in PR occurring freely, we write $\exists^k PR\ \phi$ to denote that atoms in PR are existentially quantified with the constraint that exactly k of them should be set to true. We write $\forall^k PR\ \phi$ to denote that atoms in PR are universally quantified and that only those assignments that set exactly k of the atoms to true are to be considered. An instance of the *weighted quantified satisfiability* problem consists of a Boolean formula ϕ over the set PR of propositional atoms, a partition $PR_1, \ldots, PR_\ell$ of PR and numbers $k_1, \ldots, k_\ell$. The goal is to check if $(\exists^{k_1} PR_1 \forall^{k_2} PR_2 \cdots \phi)$ is true.

▶ **Lemma 4.11.** *Given an instance of the weighted quantified satisfiability problem, We can construct in polynomial time a data graph G and an REWB $e^1_{1+k_2+k_4+\cdots}$ satisfying the following conditions.*

1. *The source of G is connected to its sink by a data path in $L(e^1_{1+k_2+k_4+\cdots})$ iff the given instance of the weighted quantified satisfiability problem is a yes instance.*
2. *The size of $e^1_{1+k_2+k_4+\cdots}$ depends only on $k_1, \ldots, k_\ell$.*

Proof idea. The required data graph G is $G[\exists k_1/PR_1] \circ G[\forall k_2/PR_2] \circ \cdots \circ G_\phi$ and the required REWB $e^1_{1+k_2+k_4+\cdots}$ is $e[\exists k_1] \circ e[\forall k_2] \circ \cdots \circ e_{eval}[k_1 + \cdots + k_\ell]$. We assume that $\circ$ associates to the right, so $G_1 \circ G_2 \circ G_3$ is $G_1 \circ (G_2 \circ G_3)$ and $e^1 \circ e^2 \circ e^3$ is $e^1 \circ (e^2 \circ e^3)$. Correctness follows from Lemma 4.10 and Lemma 4.5. ◀

The weighted quantified satisfiability problem is parameterized by $\ell + k_1 + \cdots + k_\ell$. The class AW[SAT] is the set of parameterized problems that are FPT-reducible to the weighted quantified satisfiability problem (see [8, Chapter 26]).

▶ **Theorem 4.12.** *Evaluating REWB queries, parameterized by the size of the query is hard for* AW[SAT] *under* FPT *reductions.*

Proof. The reduction given in Lemma 4.11 is a FPT reduction from the weighted quantified satisfiability problem to the problem of evaluating REWB queries, with query size as the parameter. ◀

5 Summary and Open Problems

We have proved that increasing the depth of nesting of iterated bindings in REWBs increase expressiveness. Given an REWB, it is undecidable to check if its language can be defined with another REWB with smaller depth of nesting of iterated bindings. The complexity of query evaluation problems are summarized in the following table, followed by a list of technical challenges to be overcome for closing the gaps.

Query level	Evaluation	Parameterized complexity, query size is parameter
E_1	NP-complete	(?2)W[SAT] lower bound, W[P] upper bound
E_i, $i > 1$	(?1), Σ_i upper bound	(?3)
Unbounded	PSPACE-complete [15]	(?4)AW[SAT] lower bound, uniform-XNL upper bound

1. Suppose we want to check the satisfiability of a Σ_2 Boolean formula over $(n_e + n_u)$ propositional atoms of which the first n_e atoms are existentially quantified and the last n_u are universally quantified. With currently known techniques, reducing this to query evaluation results in an REWB in $E_{(n_u+1)}$. Hence, with bounded nesting depth, we cannot even prove a Σ_2 lower bound.

2. Weighted formula satisfiability, complete for W[SAT], can be simulated with series-parallel graphs. Queries in E_1 do not seem to be powerful enough for weighted circuits.
3. Without parameterization, the Σ_i upper bound is obtained by an oracle hierarchy of NP machines. With parameterization, an oracle hierarchy of W[P] machines does not correspond to any parameterized complexity class. See [3, Section 4] for discussions on subtle points which make classical complexity results fail in parameterized complexity.
4. As in point 2, here one might hope for a AW[P] lower bound, which is quantified weighted circuit satisfiability (stronger than AW[SAT], which is quantified weighted formula satisfiability). Even if this improvement can be made, there is another classical complexity result not having analogous result in parameterized complexity: not much is known about the relation between parameterized alternating time bounded class (AW[P]) and parameterized space bounded class (uniform-XNL).

Acknowledgements. The authors thank Partha Mukhopadhyay and Geevarghese Philip for helpful discussions about polynomial time hierarchy and parameterized complexity theory.

References

1 P. Barceló. Querying graph databases. In *Proceedings of PODS*, pages 175–188, New York, NY, USA, 2013. ACM.
2 P. Barceló, J. Reutter, and L. Libkin. Parameterized regular expressions and their languages. *Theoretical Computer Science*, 474:21–45, 2013.
3 Yijia Chen, J. Flum, and M. Grohe. Bounded nondeterminism and alternation in parameterized complexity theory. In *Computational Complexity, 2003*, pages 13–29, 2003.
4 T. Colcombet and A. Manuel. Generalized data automata and fixpoint logic. In *FSTTCS*, volume 29 of *LIPIcs*, pages 267–278, 2014.
5 T. Colcombet and A. Manuel. Combinatorial expressions and lower bounds. In *STACS*, volume 30 of *LIPIcs*, pages 249–261, 2015.
6 T. Colcombet and A. Manuel. Fragments of fixpoint logic on data words. In *FSTTCS*, volume 45 of *LIPIcs*, pages 98–111, 2015.
7 R. De Haan, M. Kronegger, and A. Pfandler. Fixed-parameter tractable reductions to sat for planning. In *Proceedings of IJCAI*, pages 2897–2903, 2015.
8 R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Thomson Brooks/Cole, 1997.
9 L. C. Eggan. Transition graphs and the star-height of regular events. *Michigan Math. J.*, 10(4):385–397, 1963.
10 O. Grumberg, O. Kupferman, and S. Sheinvald. Variable automata over infinite alphabets. In *LATA*, volume 6031 of *LNCS*, pages 561–572, 2010.
11 C. Gutierrez, C. Hurtado, and A. Mendelzon. Foundations of semantic web databases. *JCSS*, 77(3):520–541, 2011.
12 E.V. Kostylev, J.L. Reutter, and D. Vrgoč. Containment of data graph queries. In *ICDT*, pages 131–142, 2014.
13 U. Leser. A query language for biological networks. *Bioinformatics*, 21(suppl 2):ii33–ii39, 2005.
14 L. Libkin, W. Martens, and D. Vrgoč. Querying graph databases with xpath. In *ICDT*, pages 129–140, New York, NY, USA, 2013. ACM.
15 L. Libkin, T. Tan, and D. Vrgoč. Regular expressions with binding over data words for querying graph databases. In *DLT*, volume 7907 of *LNCS*, pages 325–337, 2013.
16 L. Libkin and D. Vrgoč. Regular path queries on graphs with data. In *ICDT*, pages 74–85, 2012.

17 F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Logic*, 5(3):403–435, 2004.

18 W3C Recommendation. Sparql 1.1 query language. 21 March 2013.

19 R. Ronen and O. Shmueli. Soql: A language for querying and creating data in social networks. In *ICDE*, pages 1595–1602, 2009.

20 L. Segoufin. Automata and logics for words and trees over an infinite alphabet. In *CSL*, volume 4207 of *LNCS*, pages 41–57, 2006.

21 M. Sipser. *Introduction to the Theory of Computation.* Springer, 2013.

22 T. Tan. Graph reachability and pebble automata over infinite alphabets. *ACM Trans. Comput. Logic*, 14(3):19:1–19:31, 2013.

23 D. Vrgoč. Using variable automata for querying data graphs. *Information Processing Letters*, 115(3):425–430, 2015.