International Journal of Foundations of Computer Science
© World Scientific Publishing Company

Regular expressions for querying data graphs

Tony Tan
Databases and theoretical computer science group, Universiteit Hasselt
Agoralaan Gebouw D 250A, BE 3590, Diepenbeek, Belgium
ptony.tan@gmail.com

Domagoj Vrgoč
LFCS, School of Informatics, University of Edinburgh
Informatics Forum, Crichton street, EH8 9AB, United Kingdom
domagoj.vrgoc@ed.ac.uk

The standard regular expressions over finite alphabets have been widely accepted as the most basic formalism to query graph databases. However, the major drawback of this approach is that it ignores the presence of data. In this paper we study the so called *regular expressions with binding* (REWB), that is, regular expressions equipped with variables to store data within a well defined scope. In particular, we study the complexity of the query evaluation of REWB queries over graph databases.

Keywords: Graph databases; Data words; Query languages.

1. Introduction

Graph data model has received much attention lately due to a high demand from services that find the traditional relational model too restrictive. It appears naturally in a variety of applications, most notably social networks and the Semantic Web. Its other applications include biology, network traffic, crime detection, and modeling object-oriented data [8, 15, 16]. Such databases are represented as graphs in which nodes are objects and the edge labels specify the relationships between them. See, for example, [1, 2] for surveys.

The most basic formalism to query the graph data model is that of *regular path queries (RPQ)*, which select nodes connected by a path described by a regular language over the labeling alphabet [6]. There are multiple extensions such as backward navigation, regular relations over paths, and non-regular features [3, 4, 5]. However, the major drawback of this approach is that it ignores the presence of data in the graphs. In real applications we have to deal with both navigational information and the data. So a query for graph databases should not only describe the paths among vertices, but also how the data values change along them.

To this end, we use the model of graph databases in which each edge contains not only a symbol from a finite alphabet, but also a data value. Hence, paths in a graph database can be viewed as a data words – words in which each position carries a letter from a finite alphabet as well as a data value from an infinite domain. This is a well known concept adopted from the XML research [17], where a path between two vertices in an XML tree is also modeled as a data word.

One of the most commonly used formalisms for describing the notion of regularity for data words is that of *register automata* [9]. These extend the standard NFAs with registers that can store data values; transitions can compare the currently read data value with values stored in registers. However, register automata are not convenient for specifying properties – ideally, we want to use regular expressions to define languages. These have been looked at in the context of data words (or words over infinite alphabets), and are based on the idea of using *variables* for binding data values. An initial attempt to define such expressions was made in [10], but it was a very limited formalism. Another formalism, called *regular expressions with memory*, was shown to be equivalent to register automata [11, 12]. At the first glance, they appear to be a good formalism: these are expressions like $a \downarrow_x (a[x^=])^*$ saying: read letter a, bind data value to x, and read the rest of the data word checking that all letters are a and the data values are the same as x. This will define data words $\binom{a}{d} \cdots \binom{a}{d}$ for some data value d. This is reminiscent of freeze quantifiers used in connection with the study of data word languages [7].

The serious problem with these expressions, however, is the *binding* of variables. The expression above is fine, but now consider the following expression: $a \downarrow_x (a[x^=]a \downarrow_x)^* a[x^=]$. This expression re-binds variable x inside the scope of another binding, and then crucially, when this happens, the original binding of x is *lost*! Such expressions really mimic the behavior of register automata, which makes them more procedural than declarative. (The above expression defines data words of the form $\binom{a}{d_1}\binom{a}{d_1} \cdots \binom{a}{d_n}\binom{a}{d_n}$.)

Losing the original binding of a variable when reusing it inside its scope goes completely against the usual practice of writing logical expressions, programs, etc., that have bound variables. Nevertheless, this feature was essential for capturing register automata [11]. A natural question then arises about expressions in which data are bound to variables within a well defined scope similar to the one in first-order logic. These expressions, called *regular expressions with binding* (REWB) were already examined from a language theoretic point of view in [14, 13].

Here we investigate REWB as a query language for graph databases. In particular, we show that the complexity of query evaluation for REWB is PSPACE-complete (Theorem 4), the same complexity as register automata. This is rather surprising (at least to us) because REWB is shown to considerably weaker than register automata [14, 13]. In fact, we also show that the minimal witnessing paths for REWB queries can be exponentially long (Proposition 3). Despite the "negative" results, we believe our results can be interesting and useful to the community to weed out some of the potentially unfruitful approaches.

This paper is an extended version of [13], where space limitations prohibited us from presenting the proofs. Here we present the detailed proofs of the results presented in [13].

2. Data words and data graphs

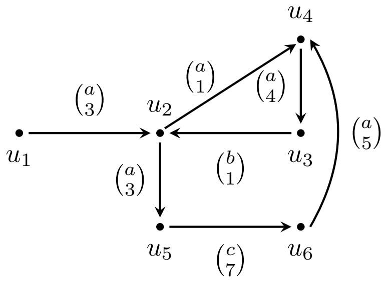
Let Σ be a finite alphabet and $\mathcal{D}$ a countable infinite set of data values. A **data word** is simply a finite string over the alphabet $\Sigma \times \mathcal{D}$. That is, in each position a data word carries a letter from Σ and a data value from $\mathcal{D}$. We will denote data words by $\binom{a_1}{d_1} \dots \binom{a_n}{d_n}$, where $a_i \in \Sigma$ and $d_i \in \mathcal{D}$.

A **data graph** (over Σ) is pair $G = (V, E)$, where

- V is a finite set of nodes;
- $E \subseteq V \times \Sigma \times \mathcal{D} \times V$ is a set of edges where each edge contains a label from Σ and a data value from $\mathcal{D}$.

We write $V(G)$ and $E(G)$ to denote the set of nodes and edges of G, respectively. An edge e from a node u to a node u' is written in the form $(u, \binom{a}{d}, u')$, where $a \in \Sigma$ and $d \in \mathcal{D}$. We call a the label of the edge e and d the data value of the edge e. We write $\mathcal{D}(G)$ to denote the set of data values in G.

The following is an example of a data graph, with nodes $u_1, \dots, u_6$ and edges $(u_1, \binom{a}{3}, u_2)$, $(u_3, \binom{b}{1}, u_2)$, $(u_2, \binom{a}{3}, u_5)$, $(u_6, \binom{a}{5}, u_4)$, $(u_2, \binom{a}{1}, u_4)$, $(u_4, \binom{a}{4}, u_3)$ and $(u_5, \binom{c}{7}, u_6)$.



A path from a node v to a node v' in G is a sequence

$$\pi = v_1 \begin{pmatrix} a_1 \\ d_1 \end{pmatrix} v_2 \begin{pmatrix} a_2 \\ d_2 \end{pmatrix} v_3 \begin{pmatrix} a_3 \\ d_3 \end{pmatrix} \cdots v_n \begin{pmatrix} a_n \\ d_n \end{pmatrix} v_{n+1}$$

such that each $(v_i, \binom{a_i}{d_i}, v_{i+1})$ is an edge for each $i \le n$, and $v_1 = v$ and $v_{n+1} = v'$.

A path π defines a data word $w(\pi) = \binom{a_1}{d_1}\binom{a_2}{d_2}\binom{a_3}{d_3} \cdots \binom{a_n}{d_n}$.

Remark Note that we have chosen a model in which labels and data values appear in edges. Of course other variations are possible, for instance labels appearing in edges and data values in nodes. All of these easily simulate each other, very much in the same way as one can use either labeled transitions systems or Kripke structures as models of temporal or modal logic formulae. In fact both models – with labels in edges and labels in nodes – have been considered in the context of semistructured data and, at least from the point of view of their expressiveness, they are viewed as equivalent. Our choice is dictated by the ease of notation primarily, as it identifies paths with data words.

3. Regular expressions with binding

We now define regular expressions with binding for data words. As explained already, expressions with variables for data words were previously defined in [12] but those were really designed to mimic the transitions of register automata, and had very procedural, rather than declarative flavor. Here we define them using proper scoping rules.

Definition 1. *Let Σ be a finite alphabet and $\{x_1, \ldots, x_k\}$ a finite set of variables.* Regular expressions with binding *(REWB) over $\Sigma[x_1, \ldots, x_k]$ are defined inductively as follows:*

$$r \quad := \quad \varepsilon \quad | \quad a \quad | \quad a[x_i^=] \quad | \quad a[x_i^{\neq}] \quad | \quad r + r \quad | \quad r \cdot r \quad | \quad r^* \quad | \quad a \downarrow_{x_i} (r) \quad (1)$$

where $a \in \Sigma$.

A variable x_i is bound if it occurs in the scope of some $\downarrow_{x_i}$ operator and free otherwise. More precisely, free variables of an expression are defined inductively: ε and a have no free variables, in $a[x_i^=]$ and $a[x_i^{\neq}]$ occurrence of x_i is free, in $r_1 + r_2$ and $r_1 \cdot r_2$ the free variables are those of r_1 and r_2, the free variables of r^* are those of r, and the free variables of $a \downarrow_{x_i} (r)$ are those of r except x_i. We will write $r(x_1, \ldots, x_l)$ if $x_1, \ldots, x_l$ are the free variables in r.

A valuation on the variables $x_1, \ldots, x_k$ is a partial function $\nu : \{x_1, \ldots, x_k\} \mapsto \mathcal{D}$. We denote by $\mathcal{F}(x_1, \ldots, x_k)$ the set of all valuations on $x_1, \ldots, x_k$. For a valuation ν, we write $\nu[x_i \leftarrow d]$ to denote the valuation ν' obtained by fixing $\nu'(x_i) = d$ and $\nu'(x) = \nu(x)$ for all other $x \neq x_i$. Likewise, we write $\nu[\bar{x} \leftarrow \bar{d}]$ for a simultaneous substitution of values from $\bar{d} = (d_1, \ldots, d_l)$ for variables $\bar{x} = (x_1, \ldots, x_l)$. Also notation $\nu(\bar{x}) = \bar{d}$ means that $\nu(x_i) = d_i$ for all $i \leq l$.

Semantics Let $r(\bar{x})$ be an REWB over $\Sigma[x_1, \ldots, x_k]$. A valuation $\nu \in \mathcal{F}(x_1, \ldots, x_k)$ is compatible with r, if $\nu(\bar{x})$ is defined.

A regular expression $r(\bar{x})$ over $\Sigma[x_1, \ldots, x_k]$ and a valuation $\nu \in \mathcal{F}(x_1, \ldots, x_k)$ compatible with r define a language $L(r, \nu)$ of data words as follows.

- If $r = a$ and $a \in \Sigma$, then $L(r, \nu) = \{\binom{a}{d} \mid d \in \mathbb{N}\}$.
- If $r = a[x^=]$, then $L(r, \nu) = \{\binom{a}{\nu(x)}\}$.
- If $r = a[x^{\neq}]$, then $L(r, \nu) = \{\binom{a}{d} \mid d \neq \nu(x)\}$.
- If $r = r_1 + r_2$, then $L(r, \nu) = L(r_1, \nu) \cup L(r_2, \nu)$.
- If $r = r_1 \cdot r_2$, then $L(r, \nu) = L(r_1, \nu) \cdot L(r_2, \nu)$.
- If $r = r_1^*$, then $L(r, \nu) = L(r_1, \nu)^*$.
- If $r = a \downarrow_{x_i} (r_1)$, then $L(r, \nu) = \bigcup_{d \in \mathcal{D}} \left\{ \begin{pmatrix} a \\ d \end{pmatrix} \right\} \cdot L(r_1, \nu[x_i \leftarrow d])$.

A REWB r defines a language of data words as follows.

$$L(r) = \bigcup_{\nu \text{ compatible with } r} L(r, \nu).$$

In particular, if r is without free variables, then $L(r) = L(r, \emptyset)$. We will call such REWBs *closed*.

Relation with register automata As mentioned earlier, one of the first data words formalisms proposed for querying graphs were register automata [11]. Together with register automata, the notion of regular expressions with memory was introduced and showed equivalent to them [12]. REWBs restrict expressions with memory, by forcing proper scoping rules, thus making them more declarative since the use of variables now adheres to standard binding policies in logics or programs. It was however proved that this comes at a price. In particular, REWBs were shown strictly weaker in terms of expressive power than register automata or expressions with memory [13, 14].

Example 2. *We list several examples of languages expressible with our expressions. In all cases below we have a singleton alphabet $\Sigma = \{a\}$.*

- *The language that consists of data words where the data value in the first position is different from the others is given by: $a \downarrow_x ((a[x^{\neq}])^*)$.*
- *The language that consists of data words where the data values in the first and the last position are the same is given by: $a \downarrow_x (a^* \cdot a[x^=])$.*
- *The language that consists of data words where there are two positions with the same data value: $a^* \cdot a \downarrow_x (a^* \cdot a[x^=]) \cdot a^*$.*

4. REWBs as a query language for data graphs

Standard mechanisms for querying graph databases are based on *regular path queries*, or RPQs: those select nodes connected by a path belonging to a given regular language [2, 6, 4, 5]. For data graphs, we follow the same idea, but now the paths are described by REWBs. In this section we study the complexity of the REWB query evaluation on data graphs.

We need a few notations. For a data graph $G = (V, E)$, and a pair of nodes s, t, we define $\mathcal{L}(G, s, t) = \{w(\pi) \mid \pi \text{ is a path from } s \text{ to } t \text{ in } G\}$.

Let $r(\bar{x})$ be a REWB over $\Sigma[x_1, \ldots, x_k]$. and ν be compatible with r. We denote by $\mathcal{L}(G, s, t, r, \nu)$ the language $\mathcal{L}(G, s, t) \cap \mathcal{L}(r, \nu)$. If r is a closed REWB, then we drop ν and write $\mathcal{L}(G, s, t, r)$.

The answer of an REWB query r over a graph $G = (V, E)$ is the set $\mathcal{Q}(r, G)$ of triples $(s, t, \bar{d}) \in V \times V \times \mathcal{D}^k$ such that $\mathcal{L}(G, s, t, r, \nu[\bar{x} \leftarrow \bar{d}])) \neq \emptyset$. In other words, $(s, t, \bar{d}) \in \mathcal{Q}(r, G)$ if and only if there is a path π in G from s to t such that $w(\pi) \in L(r, \nu[\bar{x} \leftarrow \bar{d}])$. For simplicity we will work with closed REWBs only. It can be extended to REWB with free variables in a straightforward manner.

The following proposition shows that on some graphs the shortest "witnessing" path of an REWB query can be exponentially long, even if the query uses only one variable. Its proof can be found in Subsection 4.1.

Proposition 3. *Let $\Sigma = \{\$, ¢, a, b\}$ be a finite alphabet. There exists a family of* *data graphs $\{G_n(s,t)\}_{n>1}$ with two distinguished nodes s and t, and a family of closed REWBs $\{r_n\}_{n>1}$ such that*

- *each $G_n(s,t)$ is of size $O(n)$;*
- *each r_n is a closed REWB over $\Sigma[x]$ of length $O(n)$; and*
- *every data word in $\mathcal{L}(G_n, s, t, r_n)$ is of length $\Omega(2^{\lfloor n/2 \rfloor})$.*

The REWB query evaluation problem is defined as follows.

	Query Evaluation for REWB
Input:	A data graph G, two nodes $s,t \in V(G)$ and a REWB r.
Task:	Decide whether $(s,t) \in \mathcal{Q}(r,G)$.

Note that in this problem, both the data graph and the query, given by r, are inputs; this is referred to as the *combined complexity* of query evaluation. If the expression r is fixed, we are talking about *data complexity*.

Recall that for the usual graphs (without data), the combined complexity of evaluating RPQs is polynomial, but if conjunctions of RPQs are taken, it goes up to NP (and could be NP-complete, in fact [6, 5]). When we look at data graphs and specify paths with register automata, the complexity jumps to Pspace-complete [11].

Theorem 4.

- *The complexity of query evaluation for REWB is* Pspace-*complete.*
- *For each fixed r, the complexity of query evaluation for REWB is in* NLogspace.

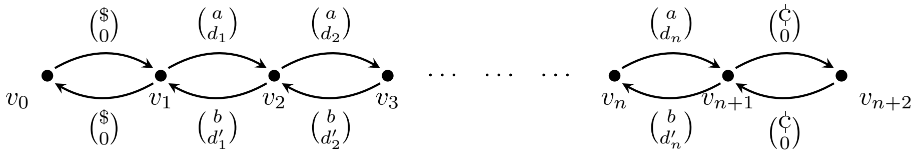
The proof of Theorem 4 can be found in Subsection 4.2. Note that the combined complexity is acceptable (it matches, for example, the combined complexity of standard relational query languages such as relational calculus and algebra), and that data complexity is the best possible for a language subsuming RPQs.

4.1. *Proof of Proposition* 3

To make the proof more precise we will introduce some additional notation. For the sake of readability, we will first prove it for REWB using multiple variables.

We write $Path(G, r, \nu)$ for the set of all paths π in G such that $w(\pi) \in L(r, \nu)$ and ν is compatible with r; and $Path(G, r, s, t, \nu)$ for the set of all paths π in G from s to t such that $w(\pi) \in L(r, \nu)$ and ν is compatible with r. Similarly, for a closed expression r, we can write $Path(G, r)$ to denote the set of all paths π in G such that $w(\pi) \in L(r)$; and $Path(G, r, s, t)$ the set of all paths π in G from s to t such that $w(\pi) \in L(r)$.

For $n \geq 2$, the graph G is defined as follows, where $0, d_1, \ldots, d_n, d'_1, \ldots, d'_n$ are pairwise different.



Obviously, G has $n+3$ vertices and $2n+4$ edges.

We define the following auxiliary REWBs $r_{k,a}$ and $r_{k,b}$ for each $k = 0, 1, \ldots, n$ as follows.

$$\begin{aligned}
r_{0,a} &:= a \ , \ r_{0,b} := b \\
r_{k,a} &:= a \downarrow_{x_k} \Big((r_{k-1,b})^* \cdot \$ \cdot \$ \cdot a^* \cdot a[x_k^=] \Big) \\
r_{k,b} &:= b \downarrow_{x_k} \Big((r_{k-1,a})^* \cdot \text{¢} \cdot \text{¢} \cdot b^* \cdot b[x_k^=] \Big)
\end{aligned}$$

The REWB r is defined as $\$ \cdot r_{n,a}^* \cdot \text{¢}$. The two nodes s and t are v_0 and v_{n+1}, respectively. Note that the length of $r_{k,a}$ and $r_{k,b}$ is $\mathcal{O}(n)$.

We claim that every path $\pi \in \mathit{Path}(G, r, s, t)$ is of length $\Omega(2^{\lfloor n/2 \rfloor})$. For this, we need a few auxiliary claims.

Claim 1. *For every $k \in \{0, 1, \ldots, n\}$ and for every node $v_i, v_j \in \{v_1, \ldots, v_n\}$ and for every valuation ν, the following holds.*

(1) There exists a path π from v_i to v_j such that $w(\pi) \in L(r_{k,a}, \nu)$ if and only if $j = i + 1$.

(2) There exists a path π from v_i to v_j such that $w(\pi) \in L(r_{k,b}, \nu)$ if and only if $i = j + 1$.

Proof. The proof is by induction on k. The basis $k = 0$ is trivial, due to the definition that $r_{0,a} = a$ and $r_{0,b} = b$, Thus,

- there exists a path π from v_i to v_j such that $w(\pi) \in L(a, \nu)$ if and only if the path is of consists of one edge labeled with a, which means $j = i + 1$; and
- there exists a path π from v_i to v_j such that $w(\pi) \in L(b, \nu)$ if and only if the path is of consists of one edge labeled with b, which means $i = j + 1$.

For the induction hypothesis we assume that our claim holds for the case of k.

For the induction step, we prove item (1) for the case of $k + 1$. Item (2) can be proved in a similar manner. The "only if" direction is as follows. Suppose there exists a path π from v_i to v_j such that $w(\pi) \in L(r_{k+1,a}, \nu)$. By definition of $r_{k+1,a}$, we have

$$r_{k+1,a} = a \downarrow_{x_{k+1}} \Big((r_{k,b})^* \cdot \$ \cdot \$ \cdot a^* \cdot a[x_{k+1}^=] \Big)$$

This means that the variable x_{k+1} is assigned with the data value d_i. Since the last step of the expression $\varphi_{k+1,a}$ is $a[x_{k+1}^=]$, and all the data values $0, d_1, \ldots, d_n$,

$d'_1, \ldots, d'_n$ are all different, the last edge in the path π must be $v_i\binom{a}{d_i}v_{i+1}$, which means $j = i+1$.

The "if" direction is as follows. We want to show that there exists a path π from v_i to v_{i+1} such that $w(\pi) \in L(r_{k+1,a}, \nu)$. We claim that there are the following paths for any valuation ν.

- There is a path π_1 from v_i to v_{i+1} such that $w(\pi_1) \in L(a, \nu[x_{k+1} \leftarrow d_i])$.
- There is a path π_2 from v_{i+1} to v_1 such that $w(\pi_2) \in L(r^*_{k,b}, \nu[x_{k+1} \leftarrow d_i])$.
- There is a path π_3 from v_1 to v_0 such that $w(\pi_3) \in L(\$, \nu[x_{k+1} \leftarrow d_i])$.
- There is a path π_4 from v_0 to v_1 such that $w(\pi_4) \in L(\$, \nu[x_{k+1} \leftarrow d_i])$.
- There is a path π_5 from v_1 to v_i such that $w(\pi_5) \in L(a^*, \nu[x_{k+1} \leftarrow d_i])$.
- There is a path π_6 from v_i to v_{i+1} such that $w(\pi_6) \in L(a[x^=_{k+1}], \nu[x_{k+1} \leftarrow d_i])$.

The existence of all the paths, except π_2, are trivially established. The existence of the path π_2 follows from the induction hypothesis that there exists a path $\pi_{(l+1,l)}$ from v_l to v_{l+1} such that $w(\pi_{(l+1,l)}) \in L(r_{k,b}, \nu[x_{k+1} \leftarrow d_i])$, for every $l = i+1, \ldots, 2$. Thus, we establish the existence of a path π from v_i to v_{i+1} such that $w(\pi) \in L(r_{k+1,a}, \nu)$. This completes the proof of the "if" direction, hence the proof of our claim. □

Now Claim 1 immediately implies the following claim.

Claim 2. *For every $k \in \{0, 1, \ldots, n\}$ and for every node $v_i, v_j \in \{v_1, \ldots, v_n\}$ and for every valuation ν, the following holds.*

(1) *There exists a path π from v_i to v_j such that $w(\pi) \in L(r^*_{k,a}, \nu)$ if and only if $j \geq i$.*
(2) *There exists a path π from v_i to v_j such that $w(\pi) \in L(r^*_{k,b}, \nu)$ if and only if $i \geq j$.*

We are going to need the following inequality. For any integer $k \geq 1$, for any integer $m \geq 1$,

$$\sum_{1 \leq i \leq m} i^k \geq \frac{m^{k+1}}{k+1}. \tag{2}$$

It can be proved by induction on m. The base case when $m = 1$ is trivial. Assume now that the claim holds for $m \geq 1$. For $m+1$ we have

$$\sum_{1 \leq i \leq m+1} i^k = \sum_{1 \leq i \leq m} i^k + (m+1)^k \geq \frac{m^{k+1}}{k+1} + (m+1)^k = \frac{m^{k+1} + (k+1)(m+1)^k}{k+1} \geq \frac{(m+1)^{k+1}}{k+1}$$

The first inequality follows from the induction hypothesis. The second inequality is obtained from the binomial expansion of $(m+1)^k$ and from the fact that $(k+1)\binom{k}{l} \geq \binom{k+1}{l}$.

For every $k \in \{0, 1, \ldots, n\}$, for every node $v_i, v_j \in \{v_1, \ldots, v_n\}$ where $i \leq j$ and for every valuation ν, the following holds.

1. *Every path π in G from v_i to v_j such that $w(\pi) \in L(r^*_{a,k}, \nu)$ has length $\geq \frac{(j-i)^{k+1}}{(k+1)!}$.*
2. *Every path π in G from v_j to v_i such that $w(\pi) \in L(r^*_{b,k}, \nu)$ has length $\geq \frac{(j-i)^{k+1}}{(k+1)!}$.*

Proof. The proof is by induction on k. The basis $k = 0$ is trivial. For the induction hypothesis, we assume that our claim holds for the case of k.

For the induction step, we prove item (1) for the case of $k + 1$. Item (2) can be proved in exactly the same manner. Let π be a path from v_i to v_j such that $w(\pi) \in L(r^*_{a,k+1}, \nu)$. By Claim 1, the path π consists of the path $\pi_{l,l+1}$ from the v_l to v_{l+1} such that $w(\pi_{l,l+1}) \in L(r_{a,k+1}, \nu)$ for every $l = i, \ldots, j-1$.

Now for every $l = i, \ldots, j-1$, the path $\pi_{l,l+1}$ consists of the following paths.

- A path π_1 from v_l to v_{l+1} such that $w(\pi_1) \in L(a, \nu[x_{k+1} \leftarrow d_l])$. The length of this path is 1.
- A path π_2 from v_{l+1} to v_1 such that $w(\pi_2) \in L(r^*_{k,b}, \nu[x_{k+1} \leftarrow d_l])$. By induction hypothesis, the length of this path is $\geq \frac{l^{k+1}}{(k+1)!}$.
- A path π_3 from v_1 to v_0 such that $w(\pi_3) \in L(\$, \nu[x_{k+1} \leftarrow d_l])$. The length of this path is 1.
- A path π_4 from v_0 to v_1 such that $w(\pi_4) \in L(\$, \nu[x_{k+1} \leftarrow d_l])$. The length of this path is 1.
- A path π_5 from v_1 to v_l such that $w(\pi_5) \in L(a^*, \nu[x_{k+1} \leftarrow d_l])$. The length of this path is $l - 1$.
- A path π_6 from v_l to v_{l+1} such that $w(\pi_6) \in L(a[x^=_{k+1}], \nu[x_{k+1} \leftarrow d_l])$. The length of this path is 1.

Thus, the length of the path $\pi_{l,l+1}$ is $\geq \frac{l^{k+1}}{(k+1)!} + l + 3$. Hence,

$$\text{the length of the path } \pi \geq \sum_{i \leq l \leq j-1} \frac{l^{k+1}}{(k+1)!} + l + 3$$
$$\geq \sum_{i \leq l \leq j-1} \frac{l^{k+1}}{(k+1)!} \geq \sum_{1 \leq l \leq j-i} \frac{l^{k+1}}{(k+1)!} \geq \frac{(j-i)^{k+2}}{(k+2)!}$$

The last inequality is obtained by applying Formula (2). □

Recall that the REWB r is defined as $\$ \cdot r^*_{n,a} \cdot ¢$ and that the two nodes s and t are v_0 and v_{n+1}, respectively. The following claim establishes that every path π from s to t such that $w(\pi) \in L(e)$ have exponential length.

Claim 4. *Every path $\pi \in Path(G, r, s, t)$ is of length $\Omega(2^{\lfloor n/2 \rfloor})$.*

Proof. It is immediate from Claim 3 that every path $\pi \in \mathit{Path}(G, r, s, t)$ is of length $\Omega(\frac{n^n}{n!})$. Since $\frac{n^n}{n!} \geq 2^{\lfloor n/2 \rfloor}$ for $n \geq 2$, our claim follows immediately. □

This completes our proof of Proposition 3 for the case of multiple variables.

It is now straightforward to check that the proof above can easily be modified when the expressions $r_{k,a}$ and $r_{k,b}$ are defined as before, but with x_i replaced by a single variable x. We omit the details since we essentially run through the same arguments.

4.2. *Proof of Theorem* 4

Note that the upper bound follows from the connection with register automata [13, 11]. Here we give an alternative proof, relying on the fact that data values reside in the edges. Let r be a REWB over $\Sigma[x_1, \dots, x_k]$ and G be a data graph. We write $\mathcal{D}(G)$ to denote the set of data values appearing in G.

For a valuation $\nu : \{x_1, \dots, x_k\} \mapsto \mathcal{D}(G)$, we define a standard regular expression $N_{G,\nu}(r)$ over the alphabet $\Sigma \times \mathcal{D}(G)$, called the normalization of r with respect to G and D, as follows.

- If $r = a$, then $N_{G,\nu}(r) = \displaystyle\bigcup_{d \in \mathcal{D}(G)} \binom{a}{d}$.
- If $r = a[x^{=}]$, then $N_{G,\nu}(r) = \binom{a}{\nu(x)}$.
- If $r = a[x^{\neq}]$, then $N_{G,\nu}(r) = \displaystyle\bigcup_{d \in \mathcal{D}(G) \text{ and } d \neq f(x)} \binom{a}{d}$.
- If $r = a \downarrow_x (r)$, then $N_{G,\nu}(r) = \displaystyle\bigcup_{d \in \mathcal{D}(G)} (a, d) \cdot N_{G,\nu[x \leftarrow d]}(r)$.
- If $r = r_1 \cdot r_2$, then $N_{G,\nu}(r) = N_{G,\nu}(r_1) \cdot N_{G,\nu}(r_2)$.
- If $r = r_1 + r_2$, then $N_{G,\nu}(r) = N_{G,\nu}(r_1) + N_{G,\nu}(r_2)$.
- If $r = r_1^*$, then $N_{G,\nu}(r) = (N_{G,\nu}(r_1))^*$.

First we show that $N_{G,\nu}(r)$ captures the desired semantics of the REWB r.

Claim 5. *For every valuation ν and for every path π, $w(\pi) \in L(r, \nu)$ if and only if $w(\pi) \in L(N_{G,\nu}(r))$.*

Proof. The proof is a straightforward induction on the structure of r. □

Observation 1. *It is readily checked that $|N_{G,\emptyset}(r)|$ is bounded by $O(|D|^{|r|})$, which is exponential in the length of the input for combined complexity (any polynomial for data complexity).*

The PSPACE upper bound now follows from simple reachability argument for regular expressions (that is answering RPQs in graph databases). Assume we are given G, s, t, a tuple $\bar{d} = (d_1, \dots, d_l)$ and a REWB r with free variables $x_1, \dots, x_l$. Let ν be a valuation such that $\nu(x_1) = d_1, \dots, \nu(x_l) = d_l$.

To find a path connecting s and t we check reachability in the product automaton of $N_{G,\nu}(r)$ and G, where we view G and an automaton over $\Sigma \times D$ with the initial state s and the final state t. From Observation 1 and standard on-the fly argument for reachability we get the desired upper bound. This also proves the NLOGSPACE bound in case of data complexity, since the length of normalization is now polynomial in the size of the input.

Now we prove the PSPACE-hardness of our theorem. The reduction is form QBF. Let

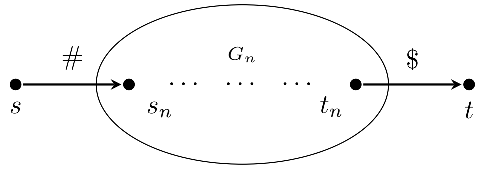
$$\begin{aligned}\Psi &= \forall x_n \exists y_n \dots \forall x_1 \exists y_1 \ \varphi \\ \varphi &= (\ell_{1,1} \vee \ell_{1,2} \vee \ell_{1,3}) \wedge (\ell_{2,1} \vee \ell_{2,2} \vee \ell_{2,3}) \wedge \cdots \wedge (\ell_{m,1} \vee \ell_{m,2} \vee \ell_{m,3})\end{aligned}$$

where each $\ell_{i,j}$ is a literal. We call a literal $\ell_{i,j}$ a *negative* literal, if it is a negation of a variable. Otherwise, we call it a *positive* literal.

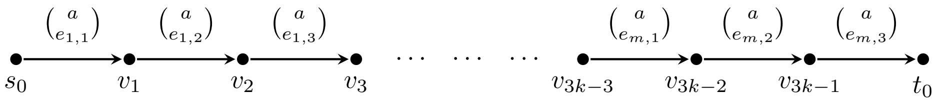
For each $i \in \{0, 1, \dots, n\}$, we will denote $\Psi_i = \forall x_i \exists y_i \dots \forall x_1 \exists y_1 \varphi$. Hence, $\Psi_0 = \varphi$ and $\Psi_n = \Psi$. We are going to construct (in polynomial time) a graph G, two nodes $s, t \in V(G)$ and an REWB r such that

$$\Psi \text{ is true if and only if } (s,t) \in \mathcal{Q}(r, G).$$

The construction of graph G and the two nodes $s, t \in V(G)$: The graph G is a data graph over $\Sigma = \{a, b, \#, \$\}$. Its construction is done inductively on $i \in \{0, 1, \dots, n\}$, where G_i, s_i, t_i are constructed from Ψ_i. The desired graph G and the two nodes $s, t \in V(G)$ is the following graph.



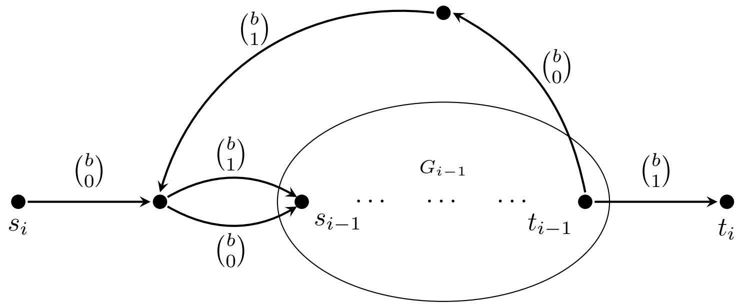
The construction of G_i, s_i, t_i is constructed inductively on i. The graph G_0 and the two vertices s_0, t_0 are as follows.



where

$$e_{i,j} = \begin{cases} 1 \text{ if the literal } \ell_{i,j} \text{ is positive} \\ 0 \text{ if the literal } \ell_{i,j} \text{ is negative} \end{cases}$$

Now we show the construction of G_i, s_i, t_i . Suppose we already constructed $G_{i-1}, s_{i-1}, t_{i-1}$. Then G_i, s_i, t_i is as follows.



<u>*The construction of the REWB r:*</u> In the following we are going to show the construction of the REWB r. We first show how to construct the auxiliary REWB r_i, for each $i = 0, 1, \ldots, m$, which is based on the formula Ψ_i. The desired REWB r is defined as $r = \# \cdot r_n \cdot \$$.

The REWB r_i is defined inductively on $k = i$. First we set

$$r_0 = \text{clause}_1 \cdot \text{clause}_2 \cdots \text{clause}_m,$$

where each clause$_i$ is defined as follows.

$$\text{clause}_i = a[x_{i,1}^{=}] \cdot a \cdot a + a \cdot a[x_{i,2}^{=}] \cdot a + a \cdot a \cdot a[x_{i,3}^{=}]$$

and $x_{i,1}, x_{i,2}, x_{i,3}$ are the variables in the literals $\ell_{i,1}, \ell_{i,2}, \ell_{i,3}$, respectively.

Now, assuming we have the REWB r_{i-1}, we define r_i as follows.

$$r_i = \Big(b \downarrow_{x_i} \big(b \downarrow_{y_i} (r_{i-1}) \cdot b[x_i^{=}] \big) \Big)^*.$$

Finally we set $r = \# \cdot r_n \cdot \$$.

It is straightforward to verify that the construction of both the data graph G and the REWB r runs in time polynomial in the length of the formula Ψ.

Remark 5. *For every $i = 0, 1, \ldots, n$,*

- *the formula Ψ_i has the free variables $x_{i+1}, y_{i+1}, \ldots, x_n, y_n$;*
- *the REWB r_i has the free variables $x_{i+1}, y_{i+1}, \ldots, x_n, y_n$.*

Moreover, for a tuple $\bar{d} \in \{0,1\}^{2(n-i)}$, we write $\Psi_i(\bar{d})$ to denote the formula Ψ_i in which the variables $x_{i+1}, y_{i+1}, \ldots, x_n, y_n$ are assigned with $\bar{d}$.

To prove that Ψ is true if and only if $(s,t) \in \mathcal{Q}(r,G)$, we prove the following claim.

Claim 6. *For each $i = 0, 1, \ldots, n$ and for every tuple $\bar{d} \in \{0,1\}^{2(n-i)}$, $\Psi_i(\bar{d})$ is true if and only if $((s_i,t_i),\bar{d}) \in \mathcal{Q}(r_i, G_i)$,*

Proof. The proof is by induction on i. The basis is $i = 0$. We have to prove that $\Psi_i(\bar{d})$ is true if and only if $((s_0,t_0),\bar{d}) \in \mathcal{Q}(r_0, G_0)$.

Let for each $i = 1, \ldots, m$ and $j = 1, 2, 3$, we write $d_{i,j}$ to denote the 0-1 value assigned to the variable in the literal $\ell_{i,j}$. Let ν denote the valuation where $\nu(x_1), \nu(y_1), \ldots, \nu(x_n), \nu(y_n)$ are assigned with $\bar{d}$, respectively. Then, we have

$$
\begin{gathered}
\Psi_0(\bar{d}) \text{ is true} \\
\Updownarrow \\
\text{every clause } (\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}) \text{ is true under the assignment } \nu \\
\Updownarrow \\
\text{for each } i = 1, \ldots, m, \text{ there exists } j \in \{1, 2, 3\} \text{ such that} \\
d_{i,j} = \begin{cases} 1 \text{ if } \ell_{i,j} \text{ is positive} \\ 0 \text{ if } \ell_{i,j} \text{ is negative} \end{cases} \\
\Updownarrow \\
\text{for each } i = 1, \ldots, m, w(\pi_i) \in L(\text{clause}_i, \nu) \text{ where} \\
\pi_i = v_{3i+0} \binom{a}{d_{i,1}} v_{3i+1} \binom{a}{d_{i,2}} v_{3i+2} \binom{a}{d_{i,3}} v_{3i+3} \\
\Updownarrow \\
((s_0, t_0), \bar{d}) \in \mathcal{Q}(r_0, G_0)
\end{gathered}
$$

For the induction hypothesis, we assume that $\Psi_i(\bar{d})$ is true if and only if $((s_i, t_i), \bar{d}) \in \mathcal{Q}(r_i, G_i)$. For the induction step, we prove the claim for $i + 1$, which follows from the following equality.

$$
\begin{gathered}
\Psi_{i+1}(\bar{d}) \text{ is true} \\
\Updownarrow \\
\text{there exist } e_0, e_1 \in \{0, 1\} \text{ such that } \Psi_i(\bar{d}0e_0) \text{ and } \ \Psi_i(\bar{d}1e_1) \text{ are true} \\
\Updownarrow \\
\text{there exist } e_0, e_1 \in \{0, 1\} \text{ such that } ((s_i, t_i), \bar{d}0e_0), ((s_i, t_i), \bar{d}1e_1) \in \mathcal{Q}(r_i, G_i). \\
\Updownarrow \\
\text{there exists a path } \pi \text{ from } s_{i+1} \text{ to } t_{i+1} \text{ such that } w(\pi) \in L(r_{i+1}, \bar{d})
\end{gathered}
$$

The last inequality follows from the definition of r_{i+1}, where

$$
r_{i+1} = \Big(b \downarrow_{x_{i+1}} \big(b \downarrow_{y_{i+1}} (r_i) \cdot b[x_{i+1}^{=}] \big) \Big)^*
$$

and to go from the vertex s_{i+1} to t_{i+1}, the path π has to go thorough G_i at least twice: once when the variable x_{i+1} is assigned with 0 and at least once when the variable x_{i+1} is assigned with 1. Thus, we have $\Psi_{i+1}(\bar{d})$ is true if and only if $((s_{i+1}, t_{i+1}), \bar{d}) \in \mathcal{Q}(r_{i+1}, G_{i+1})$.

This completes the proof of our claim. □

This concludes the proof of the hardness part, hence, our theorem.

5. Conclusions

While register automata and their expression equivalent were shown to have acceptable query evaluation bounds [11], the somewhat cumbersome syntax of automata and the relatively unnatural one of these expressions led us to search for a more suitable graph querying mechanism. Here we present and study REWB, where the scope of binding variables with data is properly defined and show that the complexity of query evaluation remains the same, despite the fact that REWB is weaker than register automata. Despite the "negative" results, we believe our results can be interesting and useful to the community to weed out some of the potentially unfruitful approaches.

Acknowledgement

We would like to thank Leonid Libkin for inspiring discussion.

References

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

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

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

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

[5] M. P. Consens, A. O. Mendelzon. GraphLog: a visual formalism for real life recursion. In *PODS'90*, pages 404–416.

[6] I. Cruz, A. Mendelzon, P. Wood. A graphical query language supporting recursion. In *SIGMOD'87*, pages 323–330.

[7] S. Demri, R. Lazić. LTL with the freeze quantifier and register automata. *ACM TOCL* 10(3): (2009).

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

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

[10] M. Kaminski and T. Tan. Regular expressions for languages over infinite alphabets. *Fundamenta Informaticae*, 69(3):301–318 (2006).

[11] L. Libkin, D. Vrgoč. Regular path queries on graphs with data. In *ICDT'12*, pages 74–85.

[12] L. Libkin, D. Vrgoč. Regular expressions for data words. *LPAR'12*, pages 274–288.

[13] L. Libkin, T. Tan, D. Vrgoč. Regular Expressions with Binding over Data Words for Querying Graph Databases. In *DLT'13*.

[14] L. Libkin, T. Tan, D. Vrgoč. Regular expressions for data words. In *Manuscript, 2013.*

[15] F. Olken. Graph data management for molecular biology. *OMICS* 7: 75–78 (2003).

[16] J. Pérez, M. Arenas, C. Gutierrez. Semantics and complexity of SPARQL. *ACM TODS* 34(3): 1–45 (2009).

[17] L. Segoufin. Automata and logics for words and trees over an infinite alphabet. In *CSL'06*, pages 41-57.