Graph Database Querying vs String Constraints

Pablo Barceló

*Millennium Institute for Foundational Research on Data &
DCC, University of Chile*

INTRODUCTION

Graph DBs and applications

- Graph DBs are crucial when topology is as important as data itself

Graph DBs and applications

- Graph DBs are crucial when topology is as important as data itself
- They gained renewed interest in last years due to trendy applications:
  - Web (semantic)
  - Social networks
  - Chemical and biological networks
  - Software bug localization
  - ...

Graph DBs and applications

- Graph DBs are crucial when topology is as important as data itself
- They gained renewed interest in last years due to trendy applications:
  - Web (semantic)
  - Social networks
  - Chemical and biological networks
  - Software bug localization
  - . . .
- They are an active area of research and industrial application:
  - Amazon Neptune, Neo4J, Facebook GraphQL, Google Knowledge Graph, Oracle Graph DBMS, RDF Virtuoso, Apache Jena, ...

Features of the query languages we study

Languages we study express essential features for querying graph DBs

- Navigation: Recursively traverse the edges of the graph
- Pattern matching: Check if a pattern appears in the graph DB
- Path comparisons: Based on relations over words

Features of the query languages we study

Languages we study express essential features for querying graph DBs

- Navigation: Recursively traverse the edges of the graph
- Pattern matching: Check if a pattern appears in the graph DB
- Path comparisons: Based on relations over words

Some of these features form the basis of recently formalized graph DB query languages:

- LDBC Proposal: G-CORE: A Core for Future Graph Query Languages (SIGMOD'18)
- Neo4J Proposal: Cypher: An Evolving Query Language for Property Graphs (SIGMOD'18)
- Survey: Foundations of Modern Query Languages for Graph Databases (ACM Comput. Surv.'17)

Problems we study:

Expressiveness: What can be said in a query language $\mathcal{L}$?

Problems we study:

Expressiveness: What can be said in a query language $\mathcal{L}$?

Complexity of evaluation: We study the problem:

PROBLEM:	EVAL($\mathcal{L}$)
INPUT:	A graph DB $\mathcal{G}$, a tuple $\bar{t}$ of objects, an $\mathcal{L}$-query Q.
QUESTION:	Is $\bar{t} \in Q(\mathcal{G})$?

- Combined complexity: Both $\mathcal{G}$ and Q are part of the input.
- Data complexity: Only $\mathcal{G}$ is part of the input and Q is fixed.

THE GRAPH DATA MODEL

Graph data model

Different apps have given rise to a myriad of different graph DB models

- (see (Angles, Gutiérrez (2008)))

Graph data model

Different apps have given rise to a myriad of different graph DB models

- (see (Angles, Gutiérrez (2008)))

We work with a simple graph data model:

Finite, directed, edge labeled graphs

Graph data model

Different apps have given rise to a myriad of different graph DB models

- (see (Angles, Gutiérrez (2008)))

We work with a simple graph data model:

Finite, directed, edge labeled graphs

Despite the simplicity of the model:

- It is flexible enough to accomodate many other more complex models and express interesting practical scenarios
- The most fundamental theoretical issues related to querying graph DBs appear in full force for it

Graph databases

Definition

A graph DB $\mathcal{G}$ over finite alphabet Σ is a pair:

$$(V, E)$$

finite set of node ids

set of edges of the form $v_1 \xrightarrow{a} v_2$
$(v_1, v_2 \in V,\ a \in \Sigma)$

Graph databases

Definition

A graph DB $\mathcal{G}$ over finite alphabet Σ is a pair:

$$(V, E)$$

finite set of node ids

set of edges of the form $v_1 \xrightarrow{a} v_2$
$(v_1, v_2 \in V,\ a \in \Sigma)$

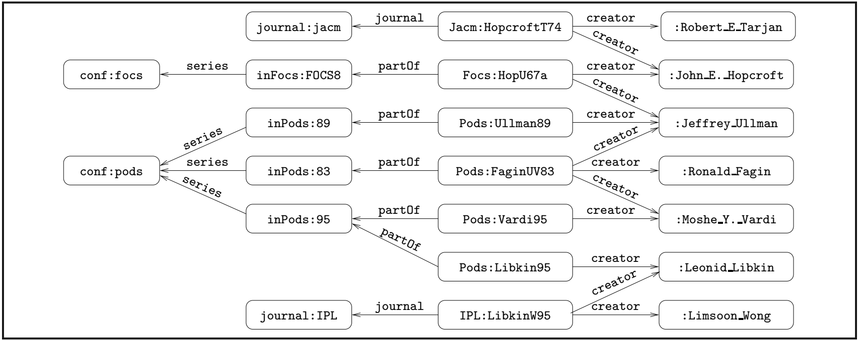
- A path in $\mathcal{G}$ is a sequence of the form:

$$\rho \;=\; v_1 \xrightarrow{a_1} v_2 \xrightarrow{a_2} v_3 \cdots v_k \xrightarrow{a_k} v_{k+1}$$

- The label of ρ, denoted $\lambda(\rho)$, is the string $a_1 a_2 \cdots a_{k-1} \in \Sigma^*$

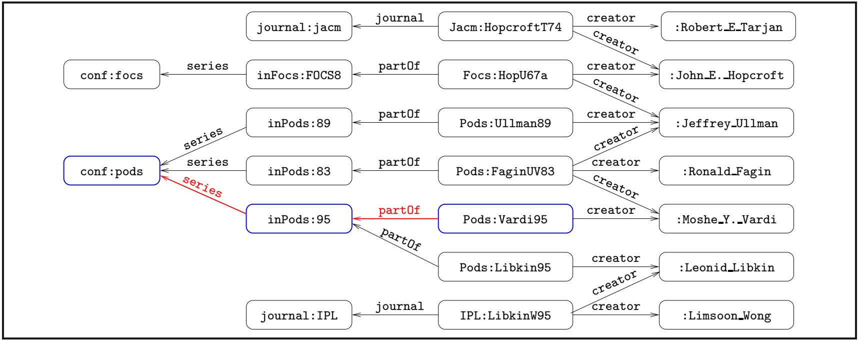
Graph DBs: Example

A graph DB representation of a fragment of DBLP



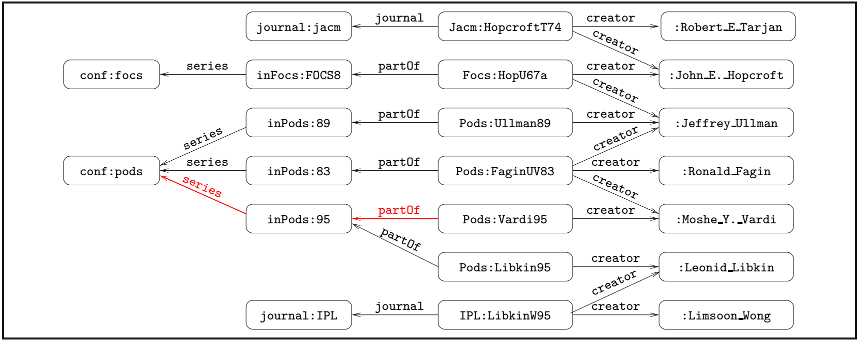
Graph DBs: Example

A path in this graph DB



Graph DBs: Example

The label of such path



Graph DBs vs NFAs

Important: Graph DBs can be naturally seen as NFAs.

- Nodes are states
- Edges $u \xrightarrow{a} v$ are transitions
- There are no initial and final states

BASIC LANGUAGES FOR GRAPH DBs:

Tractability for a big class of languages

Regular path queries

Basic building block for graph queries: Regular path queries (RPQs)

- First studied by Mendelzon and Wood (1989)
- RPQs = Regular expressions over Σ
- Evaluation $L(\mathcal{G})$ of RPQ L on graph DB $\mathcal{G} = (V, E)$:
  - Pairs of nodes $(v, v') \in V$ linked by path labeled in L

RPQs with inverse

More often studied its extension with inverses, or 2RPQs

- First studied by Calvanese, de Giacomo, Lenzerini, Vardi (2000)
- 2RPQs = RPQs over $\Sigma^{\pm}$, where:
  - $\Sigma^{\pm} = \Sigma$ extended with the inverse a^{-} of each $a \in \Sigma$

RPQs with inverse

More often studied its extension with inverses, or 2RPQs

- First studied by Calvanese, de Giacomo, Lenzerini, Vardi (2000)
- 2RPQs = RPQs over $\Sigma^{\pm}$, where:
  - $\Sigma^{\pm} = \Sigma$ extended with the inverse a^{-} of each $a \in \Sigma$

Evaluation $L(\mathcal{G})$ of 2RPQ L over graph DB $\mathcal{G} = (V, E)$.

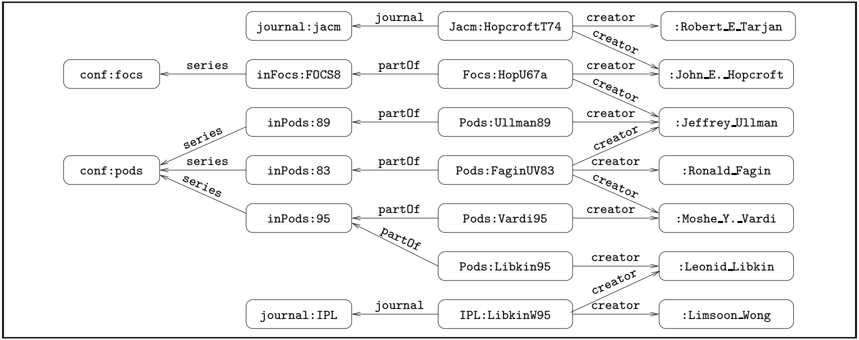
- Pairs of nodes in $\mathcal{G}$ that satisfy RPQ $L(\mathcal{G}^{\pm})$, where
  - $\mathcal{G}^{\pm}$ obtained from $\mathcal{G}$ by adding $u \xrightarrow{a^{-}} v$ for each $v \xrightarrow{a} u \in E$

Example of 2RPQ

The 2RPQ

$$\Big(\texttt{creator}^{-} \cdot \big((\texttt{partOf} \cdot \texttt{series}) \cup \texttt{journal}\big)\Big)$$

computes (a, v) s.t. author a published in conference or journal v



journal:jacm ← journal — Jacm:HopcroftT74 — creator → :Robert_E_Tarjan

Jacm:HopcroftT74 — creator → :John_E._Hopcroft

conf:focs ← series — inFocs:FOCS8 ← partOf — Focs:HopU67a — creator → :John_E._Hopcroft

Focs:HopU67a — creator → :Jeffrey_Ullman

inPods:89 ← partOf — Pods:Ullman89 — creator → :Jeffrey_Ullman

inPods:89 — series → conf:pods

Pods:FaginUV83 — creator → :Jeffrey_Ullman

conf:pods ← series — inPods:83 ← partOf — Pods:FaginUV83 — creator → :Ronald_Fagin

Pods:FaginUV83 — creator → :Moshe_Y._Vardi

inPods:95 — series → conf:pods

inPods:95 ← partOf — Pods:Vardi95 — creator → :Moshe_Y._Vardi

inPods:95 ← partOf — Pods:Libkin95 — creator → :Leonid_Libkin

IPL:LibkinW95 — creator → :Leonid_Libkin

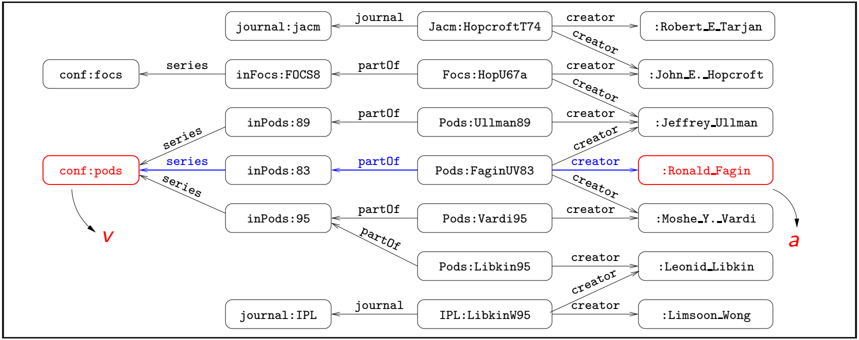
journal:IPL ← journal — IPL:LibkinW95 — creator → :Limsoon_Wong

Example of 2RPQ

The 2RPQ

$$\left(\texttt{creator}^{-} \cdot \left((\texttt{partOf} \cdot \texttt{series}) \cup \texttt{journal}\right)\right)$$

computes (a, v) s.t. author a published in conference or journal v

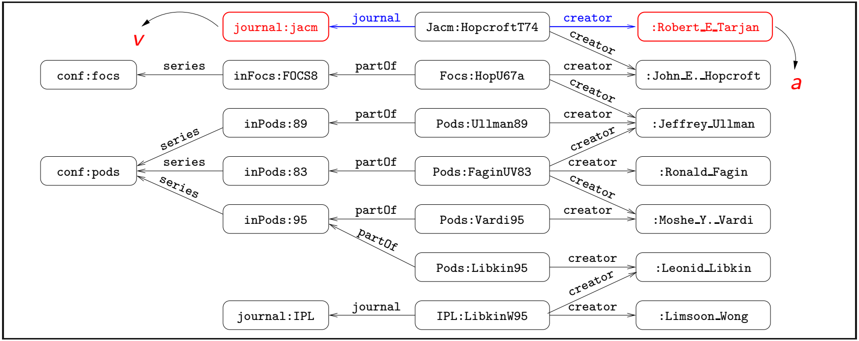


Example of 2RPQ

Example: The 2RPQ

$$\Big(\texttt{creator}^{-} \cdot \big((\texttt{partOf} \cdot \texttt{series}) \cup \texttt{journal}\big)\Big)$$

computes (a, v) s.t. author a published in conference or journal v



2RPQ evaluation

PROBLEM:	EVAL(2RPQ)
INPUT:	A graph DB $\mathcal{G}$, nodes v, v' in $\mathcal{G}$, a 2RPQ L
QUESTION:	Is $(v, v') \in L(G)$?

2RPQ evaluation

Problem: Eval(2RPQ)

Input: A graph DB $\mathcal{G}$, nodes v, v' in $\mathcal{G}$,
a 2RPQ L

Question: Is $(v, v') \in L(G)$?

It boils down to:

Problem: RegularPath

Input: A graph DB $\mathcal{G}$, nodes v, v' in $\mathcal{G}$,
a regular expression L over $\Sigma^{\pm}$

Question: Is there a path ρ from v to v' in $\mathcal{G}^{\pm}$
such that $\lambda(\rho) \in L$?

Complexity of finding regular paths

Theorem (Folklore)

RegularPath *can be solved in time* $O(|\mathcal{G}| \cdot |L|)$

Complexity of finding regular paths

Theorem (Folklore)

RegularPath *can be solved in time* $O(|\mathcal{G}| \cdot |L|)$

Proof idea:

- Compute in linear time from L an equivalent NFA $\mathcal{A}$
- Compute in linear time $(\mathcal{G}^{\pm}, v, v')$: NFA obtained from $\mathcal{G}^{\pm}$ by setting v and v' as initial and final states, respectively
- Then $(v, v') \in L(\mathcal{G})$ iff NFA $(\mathcal{G}^{\pm}, v, v') \times \mathcal{A}$ is nonempty
- The latter can be checked in time $O(|\mathcal{G}^{\pm}| \cdot |\mathcal{A}|) = O(|\mathcal{G}| \cdot |L|)$

Complexity of finding regular paths

Theorem (Folklore)

RegularPath *can be solved in time* $O(|\mathcal{G}| \cdot |L|)$

Proof idea:

- Compute in linear time from L an equivalent NFA $\mathcal{A}$
- Compute in linear time $(\mathcal{G}^{\pm}, v, v')$: NFA obtained from $\mathcal{G}^{\pm}$ by setting v and v' as initial and final states, respectively
- Then $(v, v') \in L(\mathcal{G})$ iff NFA $(\mathcal{G}^{\pm}, v, v') \times \mathcal{A}$ is nonempty
- The latter can be checked in time $O(|\mathcal{G}^{\pm}| \cdot |\mathcal{A}|) = O(|\mathcal{G}| \cdot |L|)$

Complexity of finding regular paths

Theorem (Folklore)

RegularPath *can be solved in time* $O(|\mathcal{G}| \cdot |L|)$

Proof idea:

- Compute in linear time from L an equivalent NFA $\mathcal{A}$
- Compute in linear time $(\mathcal{G}^{\pm}, v, v')$: NFA obtained from $\mathcal{G}^{\pm}$ by setting v and v' as initial and final states, respectively
- Then $(v, v') \in L(\mathcal{G})$ iff NFA $(\mathcal{G}^{\pm}, v, v') \times \mathcal{A}$ is nonempty
- The latter can be checked in time $O(|\mathcal{G}^{\pm}| \cdot |\mathcal{A}|) = O(|\mathcal{G}| \cdot |L|)$

Complexity of finding regular paths

Theorem (Folklore)

REGULARPATH *can be solved in time* $O(|\mathcal{G}| \cdot |L|)$

Proof idea:

- Compute in linear time from L an equivalent NFA $\mathcal{A}$
- Compute in linear time $(\mathcal{G}^{\pm}, v, v')$: NFA obtained from $\mathcal{G}^{\pm}$ by setting v and v' as initial and final states, respectively
- Then $(v, v') \in L(\mathcal{G})$ iff NFA $(\mathcal{G}^{\pm}, v, v') \times \mathcal{A}$ is nonempty
- The latter can be checked in time $O(|\mathcal{G}^{\pm}| \cdot |\mathcal{A}|) = O(|\mathcal{G}| \cdot |L|)$

Complexity of finding regular paths

Theorem (Folklore)

$\textsc{RegularPath}$ *can be solved in time* $O(|\mathcal{G}| \cdot |L|)$

Proof idea:

- Compute in linear time from L an equivalent NFA $\mathcal{A}$
- Compute in linear time $(\mathcal{G}^{\pm}, v, v')$: NFA obtained from $\mathcal{G}^{\pm}$ by setting v and v' as initial and final states, respectively
- Then $(v, v') \in L(\mathcal{G})$ iff NFA $(\mathcal{G}^{\pm}, v, v') \times \mathcal{A}$ is nonempty
- The latter can be checked in time $O(|\mathcal{G}^{\pm}| \cdot |\mathcal{A}|) = O(|\mathcal{G}| \cdot |L|)$

Complexity of 2RPQ evaluation

Corollary

$\textsc{Eval}$(2RPQ) *can be solved in linear time* $O(|\mathcal{G}| \cdot |L|)$

Data complexity of 2RPQ evaluation

Data complexity of 2RPQs belongs to a parallelizable class:

Proposition

Let L be a fixed 2RPQ.
There is $\textsc{NLogspace}$ *procedure that computes* $L(\mathcal{G})$ *for each* $\mathcal{G}$

Proof idea:

- Construct $(\mathcal{G}^{\pm}, v, v')$ from $\mathcal{G}$ in $\textsc{Logspace}$
- Check nonemptiness for $(\mathcal{G}^{\pm}, v, v') \times \mathcal{A}$ in $\textsc{NLogspace}$

Conjunctive regular path queries (CRPQs)

RPQs still do not express arbitrary patterns over graph DBs.

- To do this we need to close RPQs under joins and projection

Conjunctive regular path queries (CRPQs)

RPQs still do not express arbitrary patterns over graph DBs.

- To do this we need to close RPQs under joins and projection

This is the class of conjunctive regular path queries (CRPQs).

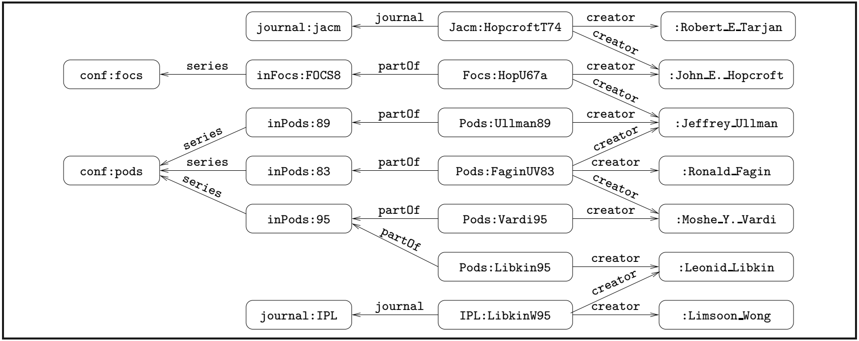
- Extended with inverses as C2RPQs in [Calvanese et al. (2000)]

Example of C2RPQ

The C2RPQ

$$Ans(x, u) \leftarrow (x, \texttt{creator}^{-}, y), (y, \texttt{partOf} \cdot \texttt{series}, z), (y, \texttt{creator}, u)$$

computes pairs (a_1, a_2) that are coauthors of a conference paper

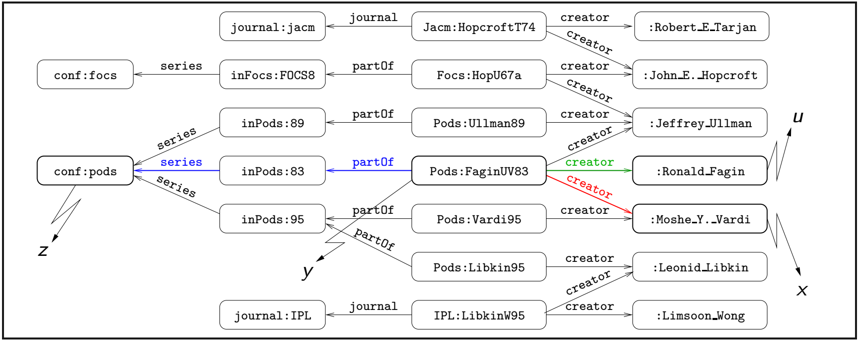


Example of C2RPQ

The C2RPQ

$$Ans(x,u) \leftarrow (x, \mathtt{creator}^-, y), (y, \mathtt{partOf}\cdot\mathtt{series}, z), (y, \mathtt{creator}, u)$$

computes pairs (a_1, a_2) that are coauthors of a conference paper

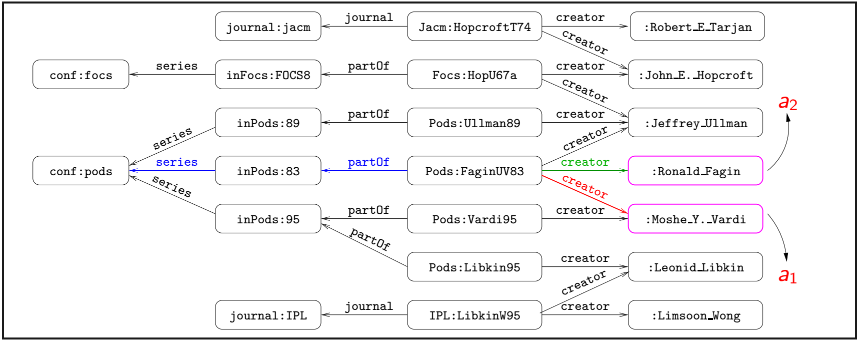


Example of C2RPQ

The C2RPQ

$$Ans(x,u) \leftarrow (x, \texttt{creator}^{-}, y), (y, \texttt{partOf} \cdot \texttt{series}, z), (y, \texttt{creator}, u)$$

computes pairs (a_1, a_2) that are coauthors of a conference paper



journal:jacm ← journal — Jacm:HopcroftT74 — creator → :Robert_E_Tarjan

Jacm:HopcroftT74 — creator → :John_E._Hopcroft

conf:focs ← series — inFocs:FOCS8 ← partOf — Focs:HopU67a — creator → :John_E._Hopcroft

Focs:HopU67a — creator → :Jeffrey_Ullman

conf:pods ← series — inPods:89 ← partOf — Pods:Ullman89 — creator → :Jeffrey_Ullman

conf:pods ← series — inPods:83 ← partOf — Pods:FaginUV83 — creator → :Ronald_Fagin

Pods:FaginUV83 — creator → :Jeffrey_Ullman

Pods:FaginUV83 — creator → :Moshe_Y._Vardi

conf:pods ← series — inPods:95 ← partOf — Pods:Vardi95 — creator → :Moshe_Y._Vardi

inPods:95 ← partOf — Pods:Libkin95 — creator → :Leonid_Libkin

journal:IPL ← journal — IPL:LibkinW95 — creator → :Limsoon_Wong

IPL:LibkinW95 — creator → :Leonid_Libkin

a_2

a_1

C2RPQ: Formal definition

C2RPQ over Σ: Rule of the form

$$Ans(\bar{z}) \leftarrow (x_1, L_1, y_1), \ldots, (x_m, L_m, y_m),$$

such that

- the x_i, y_i are variables,
- each L_i is a 2RPQ over Σ,
- the output $\bar{z}$ has some variables among the x_i, y_i's

C2RPQ: Formal definition

C2RPQ over Σ: Rule of the form

$$Ans(\bar{z}) \leftarrow (x_1, L_1, y_1), \ldots, (x_m, L_m, y_m),$$

such that

- the x_i, y_i are variables,
- each L_i is a 2RPQ over Σ,
- the output $\bar{z}$ has some variables among the x_i, y_i's

CRPQ: C2RPQ without inverse

Complexity of evaluation of C2RPQs

Increase in expressiveness from RPQs has a cost in evaluation

Proposition

$\textsc{Eval}(\text{C2RPQ})$ *is* NP*-complete, even if restricted to CRPQs*

Complexity of evaluation of C2RPQs

Increase in expressiveness from RPQs has a cost in evaluation

Proposition

$\textsc{Eval}$(C2RPQ) *is* NP*-complete, even if restricted to CRPQs*

But adding conjunctions is free in data complexity

Proposition

$\textsc{Eval}$(C2RPQ) *can be solved in* $\mathrm{N}\textsc{Logspace}$ *in data complexity*

PATH QUERIES:

The power of comparisons

CRPQs and path queries

CRPQs fall short of expressive power for applications that need:

- to include paths in the output of a query, and
- to define complex relationships among labels of paths

CRPQs and path queries

CRPQs fall short of expressive power for applications that need:

- to include paths in the output of a query, and
- to define complex relationships among labels of paths

Examples:

- Semantic Web queries:
  - establish semantic associations among paths
- Biological applications:
  - compare paths based on similarity
- Route-finding applications:
  - compare paths based on length or number of occurences of labels
- Data provenance and semantic search over the Web:
  - require returning paths to the user

Path comparisons

We use a set $\mathcal{S}$ of relations on words.

- Example: $\mathcal{S}$ may contain
  - Unary relations: Regular, context-free languages, etc.
  - Binary relations: prefix, equal length, subsequence, etc.
- Comparisons among labels of paths = Pertenence to some $S \in \mathcal{S}$
  - Example: w_1 is a substring of w_2
- We assume $\mathcal{S}$ contains all regular languages

Extended CRPQs

The $\mathcal{S}$-extended CRPQs (ECRPQ($\mathcal{S}$)) are rules obtained from a CRPQ:

$$Ans(\bar{z}, \quad) \leftarrow (x_1, L_1, y_1), \ldots, (x_m, L_m, y_m),$$

- by joining each pair (x_i, y_i) with a path variable π_i,
- comparing labels of paths in $\bar{\pi}_j$ wrt $S_j \in \mathcal{S}$
  - for $\bar{\pi}_j$ a tuple of path variables among the π_i's,
- projecting some of π_i's as a tuple $\bar{\chi}$ in the output

Extended CRPQs

The $\mathcal{S}$-extended CRPQs (ECRPQ($\mathcal{S}$)) are rules obtained from a CRPQ:

$$Ans(\bar{z}, \quad) \leftarrow (x_1, \pi_1, y_1), \ldots, (x_m, \pi_m, y_m),$$

- by joining each pair (x_i, y_i) with a path variable π_i,
- comparing labels of paths in $\bar{\pi}_j$ wrt $S_j \in \mathcal{S}$
  - for $\bar{\pi}_j$ a tuple of path variables among the π_i's,
- projecting some of π_i's as a tuple $\bar{\chi}$ in the output

Extended CRPQs

The $\mathcal{S}$-extended CRPQs (ECRPQ($\mathcal{S}$)) are rules obtained from a CRPQ:

$$Ans(\bar{z},\quad) \leftarrow (x_1, \pi_1, y_1), \ldots, (x_m, \pi_m, y_m), \bigwedge_{1 \leq j \leq t} S_j(\bar{\pi}_j)$$

- by joining each pair (x_i, y_i) with a path variable π_i,
- comparing labels of paths in $\bar{\pi}_j$ wrt $S_j \in \mathcal{S}$
  - for $\bar{\pi}_j$ a tuple of path variables among the π_i's,
- projecting some of π_i's as a tuple $\bar{\chi}$ in the output

Extended CRPQs

The $\mathcal{S}$-extended CRPQs (ECRPQ($\mathcal{S}$)) are rules obtained from a CRPQ:

$$Ans(\bar{z}, \bar{\chi}) \ \leftarrow \ (x_1, \pi_1, y_1), \ldots, (x_m, \pi_m, y_m), \bigwedge_{1 \leq j \leq t} S_j(\bar{\pi}_j)$$

- by joining each pair (x_i, y_i) with a path variable π_i,
- comparing labels of paths in $\bar{\pi}_j$ wrt $S_j \in \mathcal{S}$
  • for $\bar{\pi}_j$ a tuple of path variables among the π_i's,
- projecting some of π_i's as a tuple $\bar{\chi}$ in the output

Extended CRPQs and our requirements

ECRPQs meet our requirements:

$$Ans(\bar{z}, \bar{\chi}) \ \leftarrow \ (x_1, \pi_1, y_1), \ldots, (x_m, \pi_m, y_m), \ \bigwedge_{1 \leq j \leq t} S_j(\bar{\pi}_j)$$

Extended CRPQs and our requirements

ECRPQs meet our requirements:

$$Ans(\bar{z}, \bar{\chi}) \;\leftarrow\; (x_1, \pi_1, y_1), \ldots, (x_m, \pi_m, y_m),\ \bigwedge_{1 \leq j \leq t} S_j(\bar{\pi}_j)$$

- They allow to export paths in the output
- They allow to compare labels of paths with relations $S_j \in \mathcal{S}$

Extended CRPQs and our requirements

ECRPQs meet our requirements:

$$Ans(\bar{z}, \bar{\chi}) \; \leftarrow \; (x_1, \pi_1, y_1), \ldots, (x_m, \pi_m, y_m), \bigwedge_{1 \leq j \leq t} S_j(\bar{\pi}_j)$$

- They allow to export paths in the output
- They allow to compare labels of paths with relations $S_j \in \mathcal{S}$

Considerations about ECRPQ($\mathcal{S}$)

- ECRPQ($\mathcal{S}$) extends the class of CRPQs
  - $Ans(\bar{z}) \leftarrow \bigwedge_i (x_i, L_i, y_i) = Ans(\bar{z}) \leftarrow \bigwedge_i (x_i, \pi_i, y_i), L_i(\pi_i)$
- Expressiveness and complexity of ECRPQ($\mathcal{S}$):
  - Depends on the class $\mathcal{S}$
- We study two such classes with roots in formal language theory:
  - Regular relations [Elgot, Mezei (1965)]
  - Rational relations [Nivat (1968)]

COMPARING PATHS WITH REGULAR RELATIONS:

Preserving tractable data complexity

Introduction

- Regular relations: Regular languages for relations of any arity
  - REG: Class of regular relations
- Bottomline:

  ECRPQ(REG): Reasonable expressiveness and complexity

Regular relations

n-ary regular relation:

Set of n-tuples $(w_1, \ldots, w_n)$ of strings
accepted by synchronous automaton over Σ^n

Regular relations

n-ary regular relation:

Set of n-tuples $(w_1, \ldots, w_n)$ of strings
accepted by synchronous automaton over Σ^n

- The input strings are written in the n-tapes
- Shorter strings are padded with symbol $\perp$
- At each step:
  The automaton simultaneously reads next symbol on each tape

Synchronous automata

$$\begin{array}{rclllllll}
w_1 & = & \mathrm{a} & \mathrm{a} & \mathrm{b} & \cdots & \mathrm{a} & \mathrm{b} & \mathrm{c} \\
w_2 & = & \mathrm{a} & \mathrm{b} & \mathrm{a} & \cdots & \mathrm{a} & & \\
w_3 & = & \mathrm{b} & \mathrm{b} & & \cdots & & & \\
\vdots & & & & & \vdots & & & \\
w_n & = & \mathrm{a} & \mathrm{b} & \mathrm{b} & \cdots & \mathrm{a} & \mathrm{c} &
\end{array}$$

Synchronous automata

$$
\begin{array}{ccccccccc}
w_1 & = & \mathsf{a} & \mathsf{a} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{b} & \mathsf{c} \\
w_2 & = & \mathsf{a} & \mathsf{b} & \mathsf{a} & \cdots & \mathsf{a} & \perp & \perp \\
w_3 & = & \mathsf{b} & \mathsf{b} & \perp & \cdots & \perp & \perp & \perp \\
\vdots & & & & & \vdots & & & \\
w_n & = & \mathsf{a} & \mathsf{b} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{c} & \perp
\end{array}
$$

Synchronous automata

$$\begin{array}{lcccccccc}
w_1 & = & \mathsf{a} & \mathsf{a} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{b} & \mathsf{c} \\
w_2 & = & \mathsf{a} & \mathsf{b} & \mathsf{a} & \cdots & \mathsf{a} & \perp & \perp \\
w_3 & = & \mathsf{b} & \mathsf{b} & \perp & \cdots & \perp & \perp & \perp \\
\vdots & & & & & \vdots & & & \\
w_n & = & \mathsf{a} & \mathsf{b} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{c} & \perp \\
 & & \Uparrow & & & & & &
\end{array}$$

Synchronous automata

$$\begin{array}{ccccccccc}
w_1 & = & a & a & b & \cdots & a & b & c \\
w_2 & = & a & b & a & \cdots & a & \perp & \perp \\
w_3 & = & b & b & \perp & \cdots & \perp & \perp & \perp \\
\vdots & & & & & \vdots & & & \\
w_n & = & a & b & b & \cdots & a & c & \perp \\
 & & & \Uparrow & & & & &
\end{array}$$

Synchronous automata

$$\begin{array}{ccccccccc}
w_1 & = & \mathsf{a} & \mathsf{a} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{b} & \mathsf{c} \\
w_2 & = & \mathsf{a} & \mathsf{b} & \mathsf{a} & \cdots & \mathsf{a} & \perp & \perp \\
w_3 & = & \mathsf{b} & \mathsf{b} & \perp & \cdots & \perp & \perp & \perp \\
\vdots & & & & & \vdots & & & \\
w_n & = & \mathsf{a} & \mathsf{b} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{c} & \perp \\
 & & & & \Uparrow & & & &
\end{array}$$

Synchronous automata

$$\begin{array}{ccccccccc}
w_1 & = & \mathsf{a} & \mathsf{a} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{b} & \mathsf{c} \\
w_2 & = & \mathsf{a} & \mathsf{b} & \mathsf{a} & \cdots & \mathsf{a} & \bot & \bot \\
w_3 & = & \mathsf{b} & \mathsf{b} & \bot & \cdots & \bot & \bot & \bot \\
\vdots & & & & & \vdots & & & \\
w_n & = & \mathsf{a} & \mathsf{b} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{c} & \bot \\
 & & & & & & \Uparrow & &
\end{array}$$

Synchronous automata

$$\begin{array}{ccccccccc}
w_1 & = & \mathsf{a} & \mathsf{a} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{b} & \mathsf{c} \\
w_2 & = & \mathsf{a} & \mathsf{b} & \mathsf{a} & \cdots & \mathsf{a} & \perp & \perp \\
w_3 & = & \mathsf{b} & \mathsf{b} & \perp & \cdots & \perp & \perp & \perp \\
\vdots & & & & & \vdots & & & \\
w_n & = & \mathsf{a} & \mathsf{b} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{c} & \perp \\
 & & & & & & & \Uparrow &
\end{array}$$

Synchronous automata

$$\begin{array}{ccccccccc}
w_1 & = & \mathrm{a} & \mathrm{a} & \mathrm{b} & \cdots & \mathrm{a} & \mathrm{b} & \mathrm{c} \\
w_2 & = & \mathrm{a} & \mathrm{b} & \mathrm{a} & \cdots & \mathrm{a} & \perp & \perp \\
w_3 & = & \mathrm{b} & \mathrm{b} & \perp & \cdots & \perp & \perp & \perp \\
\vdots & & & & & \vdots & & & \\
w_n & = & \mathrm{a} & \mathrm{b} & \mathrm{b} & \cdots & \mathrm{a} & \mathrm{c} & \perp \\
 & & & & & & & & \Uparrow
\end{array}$$

Examples of regular relations

- All regular languages
- The prefix relation defined by:

$$\big(\bigcup_{a\in\Sigma}(a,a)\big)^* \cdot \big(\bigcup_{a\in\Sigma}(a,\bot)\big)^*$$

- The equal length relation defined by:

$$\big(\bigcup_{a,b\in\Sigma}(a,b)\big)^*$$

- Pairs of strings at edit distance at most k, for fixed $k \geq 0$

Examples of regular relations

- All regular languages
- The prefix relation defined by:

$$\Big(\bigcup_{a\in\Sigma}(a,a)\Big)^* \cdot \Big(\bigcup_{a\in\Sigma}(a,\perp)\Big)^*$$

- The equal length relation defined by:

$$\Big(\bigcup_{a,b\in\Sigma}(a,b)\Big)^*$$

- Pairs of strings at edit distance at most k, for fixed $k \geq 0$

Proposition

The subsequence, subword and suffix relations are not regular

ECRPQ(REG)

ECRPQ(REG): Class of queries of the form

$$Ans(\bar{z}, \bar{\chi}) \leftarrow \bigwedge_i (x_i, \pi_i, y_i), \bigwedge_j S_j(\bar{\pi}_j),$$

where each S_j is a regular relation [B., Libkin, Lin, Wood (2012)]

ECRPQ(REG)

ECRPQ(REG): Class of queries of the form

$$Ans(\bar{z}, \bar{\chi}) \leftarrow \bigwedge_i (x_i, \pi_i, y_i), \bigwedge_j S_j(\bar{\pi}_j),$$

where each S_j is a regular relation [B., Libkin, Lin, Wood (2012)]

Example: The ECRPQ(REG) query

$$Ans(x, y) \leftarrow (x, \pi_1, z), (z, \pi_2, y), a^*(\pi_1), b^*(\pi_2), \mathrm{equal\_length}(\pi_1, \pi_2)$$

computes pairs of nodes linked by a path labeled in $\{a^n b^n \mid n \geq 0\}$

ECRPQ(REG)

ECRPQ(REG): Class of queries of the form

$$Ans(\bar{z}, \bar{\chi}) \leftarrow \bigwedge_i (x_i, \pi_i, y_i), \bigwedge_j S_j(\bar{\pi}_j),$$

where each S_j is a regular relation [B., Libkin, Lin, Wood (2012)]

Example: The ECRPQ(REG) query

$$Ans(x, y) \leftarrow (x, \pi_1, z), (z, \pi_2, y), a^*(\pi_1), b^*(\pi_2), \mathrm{equal\_length}(\pi_1, \pi_2)$$

computes pairs of nodes linked by a path labeled in $\{a^n b^n \mid n \geq 0\}$

Corollary

ECRPQ(REG) *properly extends the class of* CRPQs

Complexity of evaluation of ECRPQ(REG)

- Extending CRPQs with regular relations is free in data complexity
- Combined complexity is that of FO over relational databases

Theorem (B., Libkin, Lin, Wood (2012))

- EVAL(ECPRQ(REG)) *is* PSPACE*-complete*
- EVAL(ECPRQ(REG)) *is in* NLOGSPACE *in data complexity*

Complexity of evaluation of ECRPQ(REG)

- Extending CRPQs with regular relations is free in data complexity
- Combined complexity is that of FO over relational databases

Theorem (B., Libkin, Lin, Wood (2012))

- EVAL(ECPRQ(REG)) *is* PSPACE*-complete*
- EVAL(ECPRQ(REG)) *is in* NLOGSPACE *in data complexity*

Proof idea:

- Convert into RPQ evaluation over $\mathcal{G}^m$, for $m =$ size of ECRPQ
- For data complexity m is fixed

Expressiveness of ECRPQ(REG)

Understanding the expressive power of ECRPQ(REG) is difficult.

Proposition

Let L be a language of words. TFAE:

- *L is expressible by a binary ECRPQ(REG) formula*
- *L is definable by a word equation with constraints in REG*

COMPARING PATHS WITH RATIONAL RELATIONS:

The struggle for decidability and efficiency

Introduction

ECRPQ(REG) queries are still short of expressive power.

- RDF or biological networks:
  - Compare strings based on subsequence and subword relations
- These relations are rational: Accepted by asynchronous automata
  - RAT: Class of rational relations

Bottomline:

- ECRPQ(RAT) evaluation:
  - Undecidable or very high complexity
- Restricting the syntactic shape of queries yields tractability

Rational relations

n-ary rational relation:
Set of n-tuples $(w_1, \ldots, w_n)$ of strings
accepted by asynchronous automaton with n heads.

Rational relations

n-ary rational relation:
Set of *n*-tuples $(w_1, \ldots, w_n)$ of strings
accepted by asynchronous automaton with *n* heads.

- The input strings are written in the *n*-tapes
- At each step:
  The automaton enters a new state and move some tape heads

Rational relations

n-ary rational relation:
Set of *n*-tuples $(w_1, \ldots, w_n)$ of strings
accepted by asynchronous automaton with *n* heads.

- The input strings are written in the *n*-tapes
- At each step:
  The automaton enters a new state and move some tape heads

n-ary rational relation:
Described by regular expression over alphabet $(\Sigma \cup \{\epsilon\})^n$

Examples of rational relations

- All regular relations
- The subsequence relation $\preceq_{\mathrm{ss}}$ defined by

$$\Big(\big(\bigcup_{a \in \Sigma} (a, \epsilon) \big)^* \bigcup_{b \in \Sigma} (b, b) \Big)^* \cdot \big(\bigcup_{a \in \Sigma} (a, \epsilon) \big)^*$$

- The subword relation $\preceq_{\mathrm{sw}}$ defined by

$$\big(\bigcup_{a \in \Sigma} (a, \epsilon) \big)^* \cdot \big(\bigcup_{b \in \Sigma} (b, b) \big)^* \cdot \big(\bigcup_{a \in \Sigma} (a, \epsilon) \big)^*$$

Examples of rational relations

- All regular relations
- The subsequence relation $\preceq_{\mathrm{ss}}$ defined by

$$\Big(\big(\bigcup_{a \in \Sigma} (a, \epsilon) \big)^* \bigcup_{b \in \Sigma} (b, b) \Big)^* \cdot \big(\bigcup_{a \in \Sigma} (a, \epsilon) \big)^*$$

- The subword relation $\preceq_{\mathrm{sw}}$ defined by

$$\big(\bigcup_{a \in \Sigma} (a, \epsilon) \big)^* \cdot \big(\bigcup_{b \in \Sigma} (b, b) \big)^* \cdot \big(\bigcup_{a \in \Sigma} (a, \epsilon) \big)^*$$

Proposition

The set of pairs (w_1, w_2) such that w_1 is the reversal of w_2 is not rational.

ECRPQ(RAT)

ECRPQ(RAT): Class of queries of the form

$$Ans(\bar{z}, \bar{\chi}) \leftarrow \bigwedge_i (x_i, \pi_i, y_i), \bigwedge_j S_j(\bar{\pi}_j),$$

where each S_j is a rational relation [B., Figueira, Libkin (2012)]

Example: The ECRPQ(RAT) query

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

computes x, y that are origins of paths ρ_1 and ρ_2 such that:

- $\lambda(\rho_1)$ is a subsequence of $\lambda(\rho_2)$

Evaluation of ECRPQ(RAT) queries

Evaluation of queries in ECRPQ(RAT) is undecidable, but:

- True if we allow only practically motivated rational relations?
  - For example, $\preceq_{\mathrm{ss}}$ and $\preceq_{\mathrm{sw}}$

Evaluation of ECRPQ(RAT) queries

Evaluation of queries in ECRPQ(RAT) is undecidable, but:

- True if we allow only practically motivated rational relations?
  - For example, $\preceq_{\mathrm{ss}}$ and $\preceq_{\mathrm{sw}}$

Adding subword relation to ECRPQ(REG) leads to undecidability:

Theorem (B., Figueira, Libkin (2012))

$\textsc{Eval}$(ECRPQ(REG $\cup\{\preceq_{\mathrm{sw}}\}$)) *is undecidable (even in data complexity)*

Evaluation of ECRPQ(RAT) queries

Evaluation of queries in ECRPQ(RAT) is undecidable, but:

- True if we allow only practically motivated rational relations?
  - For example, $\preceq_{\text{ss}}$ and $\preceq_{\text{sw}}$

Adding subword relation to ECRPQ(REG) leads to undecidability:

Theorem (B., Figueira, Libkin (2012))

$\textsc{Eval}$(ECRPQ(REG $\cup\{\preceq_{\text{sw}}\}$)) *is undecidable (even in data complexity)*

Adding subword to CRPQ leads to intractability in data complexity:

Theorem (B., Muñoz (2014))

$\textsc{Eval}$(CRPQ($\preceq_{\text{sw}}$)) *is* PSPACE*-complete in data complexity*

- *But* $\textsc{Eval}$(CRPQ($\preceq_{\text{suff}}$)) *is in* $\textsc{NLogspace}$ *in data complexity*

Consequences for word equations

Observation 1: PSPACE upper bound for CRPQ($\preceq_{\text{sw}}$)

- Uses PSPACE procedure for word equations with regular expressions

Consequences for word equations

Observation 1: PSPACE upper bound for CRPQ($\preceq_{\text{sw}}$)

- Uses PSPACE procedure for word equations with regular expressions

Observation 2: There exists a fixed word equation e such that

- solving e under a single constraint in REG is undecidable
- solving e with regular language constraints is PSPACE-complete

Evaluation of ECRPQ(RAT) queries

Adding subsequence to ECRPQ preserves decidability at a very high cost:

Theorem (B., Figueira, Libkin (2012))

$\textsc{Eval}$(ECRPQ(REG $\cup\{\preceq_{\mathrm{ss}}\}$)) *is decidable, but non-primitive-recursive.*

- *This holds even in data complexity.*

Evaluation of ECRPQ(RAT) queries

Adding subsequence to ECRPQ preserves decidability at a very high cost:

Theorem (B., Figueira, Libkin (2012))

$\textsc{Eval}$(ECRPQ(REG $\cup\{\preceq_{\mathrm{ss}}\}$)) *is decidable, but non-primitive-recursive.*

- *This holds even in data complexity.*

Adding subsequence to CRPQ leads to intractability in data complexity:

Theorem (B., Muñoz (2014))

$\textsc{Eval}$(CRPQ($\preceq_{\mathrm{ss}}$)) *is* NP*-complete in data complexity*

Evaluation of ECRPQ(RAT) queries

Adding subsequence to ECRPQ preserves decidability at a very high cost:

Theorem (B., Figueira, Libkin (2012))

$\textsc{Eval}$(ECRPQ(REG $\cup\{\preceq_{\text{ss}}\}$)) *is decidable, but non-primitive-recursive.*

- *This holds even in data complexity.*

Adding subsequence to CRPQ leads to intractability in data complexity:

Theorem (B., Muñoz (2014))

$\textsc{Eval}$(CRPQ($\preceq_{\text{ss}}$)) *is* NP*-complete in data complexity*

Observation 3: Word equations $+ \preceq_{\text{ss}}$ undecidable [Halfon et al (2017)]

- Is this also the case for $\textsc{Eval}$(CRPQ($\preceq_{\text{ss}} \cup \preceq_{\text{sw}}$))?

Acyclic CRPQ(RAT) queries

Acyclic CRPQ(RAT) queries yield tractable data complexity.

- Queries of the form

$$Ans(\bar{z}) \leftarrow \bigwedge_{i \le k} (x_i, \pi_i, y_i), L_i(\pi_i), \bigwedge_{j} S_j(\pi_{j_1}, \pi_{j_2}),$$

  where the graph on $\{1, \ldots, k\}$ defined by edges (π_{j_1}, π_{j_2}) is acyclic

Acyclic CRPQ(RAT) queries

Acyclic CRPQ(RAT) queries yield tractable data complexity.

- Queries of the form

$$Ans(\bar{z}) \leftarrow \bigwedge_{i \leq k} (x_i, \pi_i, y_i), L_i(\pi_i), \bigwedge_j S_j(\pi_{j_1}, \pi_{j_2}),$$

where the graph on $\{1, \ldots, k\}$ defined by edges (π_{j_1}, π_{j_2}) is acyclic

Acyclic ECRPQ(RAT) is not more expensive than ECRPQ(REG):

Theorem (B., Figueira, Libkin (2012))

- *Evaluation of acyclic* ECRPQ(RAT) *queries is* PSPACE*-complete*
- *It is in* NLOGSPACE *in data complexity*

STRING SOLVING:

Applying previous ideas

The problem we study

We study satisfiability for conjunctions of:

- Atomic relational constraints:

$$y = x_1 \cdots x_n \mid R(x, y)$$

- Boolean combinations of regular expressions:

$$L(x) \mid \varphi \wedge \psi \mid \neg\varphi$$

The problem we study

We study satisfiability for conjunctions of:

- Atomic relational constraints:

$$y = x_1 \cdots x_n \mid R(x, y)$$

- Boolean combinations of regular expressions:

$$L(x) \mid \varphi \wedge \psi \mid \neg\varphi$$

Example: $x = w_1 y w_2 z w_3 \wedge R(y, z) \wedge \neg S(z)$

The problem we study

We study satisfiability for conjunctions of:

- Atomic relational constraints:

$$y = x_1 \cdots x_n \mid R(x, y)$$

- Boolean combinations of regular expressions:

$$L(x) \mid \varphi \wedge \psi \mid \neg \varphi$$

Example: $x = w_1 y w_2 z w_3 \wedge R(y, z) \wedge \neg S(z)$

This class is

- Useful: Encodes transductions often used in web security applications, e.g., replace_all
- Very expressive: Subsumes word equations with rational constraints

In full generality the problem is undecidable

Proposition

Satisfiability of expressions $R(x, x)$ is undecidable

In full generality the problem is undecidable

Proposition

Satisfiability of expressions $R(x, x)$ is undecidable

Idea: Use acyclicity restrictions as we did for ECRPQ(RAT)

In full generality the problem is undecidable

Proposition

Satisfiability of expressions $R(x, x)$ is undecidable

Idea: Use acyclicity restrictions as we did for ECRPQ(RAT)

But not just on the graph defined by rational relations ...

- $R(x, x)$ is equivalent to $x = y \wedge R(x, y)$
- Satisfiability of formulas of the form $x = yz \wedge R(x, z)$, for R a regular relation, is undecidable [B., Figueira, Libkin (2013)]

In full generality the problem is undecidable

Proposition

Satisfiability of expressions $R(x, x)$ is undecidable

Idea: Use acyclicity restrictions as we did for ECRPQ(RAT)

But not just on the graph defined by rational relations ...

- $R(x, x)$ is equivalent to $x = y \wedge R(x, y)$
- Satisfiability of formulas of the form $x = yz \wedge R(x, z)$, for R a regular relation, is undecidable [B., Figueira, Libkin (2013)]

Notion of acyclicity needs to consider expressions $y = x_1 \cdots x_n$

Acyclicity restriction

We write $R(x, y)$ as $y = R(x)$

The straight line (SL) fragment:

$$\bigwedge_{i=1}^{m} x_i = P(x_1, \ldots, x_{i-1}),$$

such that $P(x_1, \ldots, x_{i-1})$ is either

$$L(x_j) \quad \text{or} \quad x_{j_1} \cdots x_{j_n}, \quad \text{for } \{x_j, x_{j_1}, \ldots x_{j_n}\} \subseteq \{x_1, \ldots, x_{i-1}\}.$$

Acyclicity restriction

We write $R(x, y)$ as $y = R(x)$

The straight line (SL) fragment:

$$\bigwedge_{i=1}^{m} x_i = P(x_1, \ldots, x_{i-1}),$$

such that $P(x_1, \ldots, x_{i-1})$ is either

$$L(x_j) \quad \text{or} \quad x_{j_1} \cdots x_{j_n}, \quad \text{for } \{x_j, x_{j_1}, \ldots x_{j_n}\} \subseteq \{x_1, \ldots, x_{i-1}\}.$$

Example: The formula $x = yz \wedge R(x, y)$ is not in SL, while the formula $x = w_1 y w_2 z w_3 \ \wedge \ R(y, z)$ is in SL

The main result

Theorem (Lin, B. (2016))

Satisfiability of expressions in SL is Expspace*-complete*

The main result

Theorem (Lin, B. (2016))

Satisfiability of expressions in SL is $\textsc{Expspace}$*-complete*

Proof idea for upper bound:

- Replace concatenations in the expression φ with "exponentially big" DNF expressions consisting exclusively of regular expressions and regular relations $x = y$
- If $\varphi \in SL$, then the resulting expression φ' is acyclic in the sense studied for ECRPQ(RAT)
- Check satisfiability of φ' in $\textsc{Pspace}$, i.e., in $\textsc{Expsace}$ in terms of the size of the input φ

A better behaved fragment

SL_k: Restriction of SL to expressions of depth $k \geq 1$

- Depth of a variable x is number of variables on which x depends
- Depth of an expression is maximum depth of a variable

A better behaved fragment

SL_k: Restriction of SL to expressions of depth $k \geq 1$

- Depth of a variable x is number of variables on which x depends
- Depth of an expression is maximum depth of a variable

Theorem (Lin, B. (2016))

Satisfiability of expressions in SL_k is $\textsc{Pspace}$*-complete*

FINAL REMARKS

Graph DB query languages and string verification share:

- interest in expressing complex interactions among words
- understanding which restrictions on such problems can lead to practical tools in real-world applications

Graph DB query languages and string verification share:

- interest in expressing complex interactions among words
- understanding which restrictions on such problems can lead to practical tools in real-world applications

I presented somes interaction between graph DBs, string verification, and word equations, but others are also possible.

- Graph QLs with arithmetic expressions:
  - Require applying tools based on Presburguer atithmetic and bounded-reversal counter automata [B., Libkin, Lin, Wood (2012)]
- Monadic decomposability:
  - Can a regular relation be expressed as a Boolean combination of products of regular languages? [B., Hong, Le, Li, Niskanen (2019)]
  - Related to *boundedness* problems for recursive query languages

THANKS