Graph Data Management

Why graphs?

The world is a graph – everything is connected

- people, places, events
- companies, markets
- countries, history, politics
- life sciences, bioinformatics, clinical data
- art, teaching
- technology, networks, machines, applications, users
- software, code, dependencies, architecture, deployments
- criminals, fraudsters and their behavior

The topology of the data is at least as important as the data itself

Humans think in graphs

- We understand and learn by
  - how something new is similar to what we already know
  - how it differs
  - i.e. by relating things
    - in a graph!

What are people using graph databases for?

Use Cases

Internal Applications

Master Data Management

Network and
IT Operations

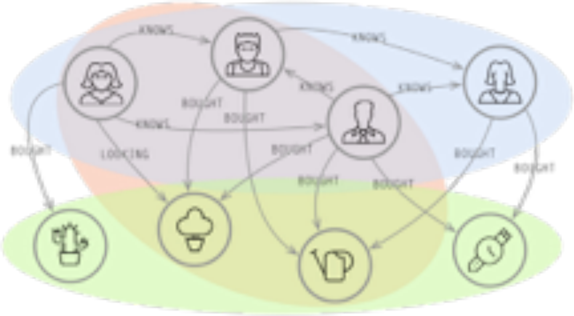
Fraud Detection

Customer-Facing Applications

Real-Time Recommendations

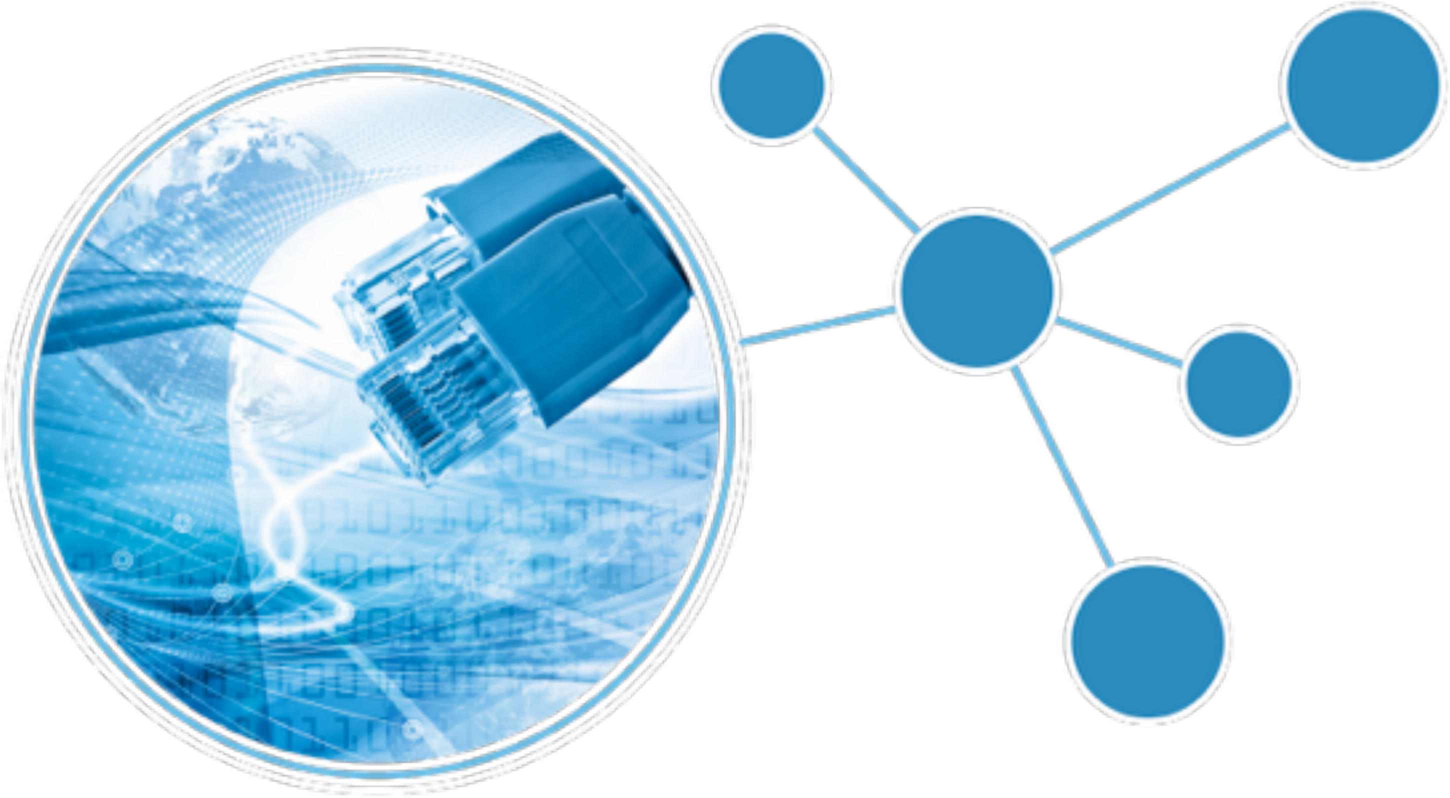
Graph-Based Search

Identity and
Access Management



Social Network

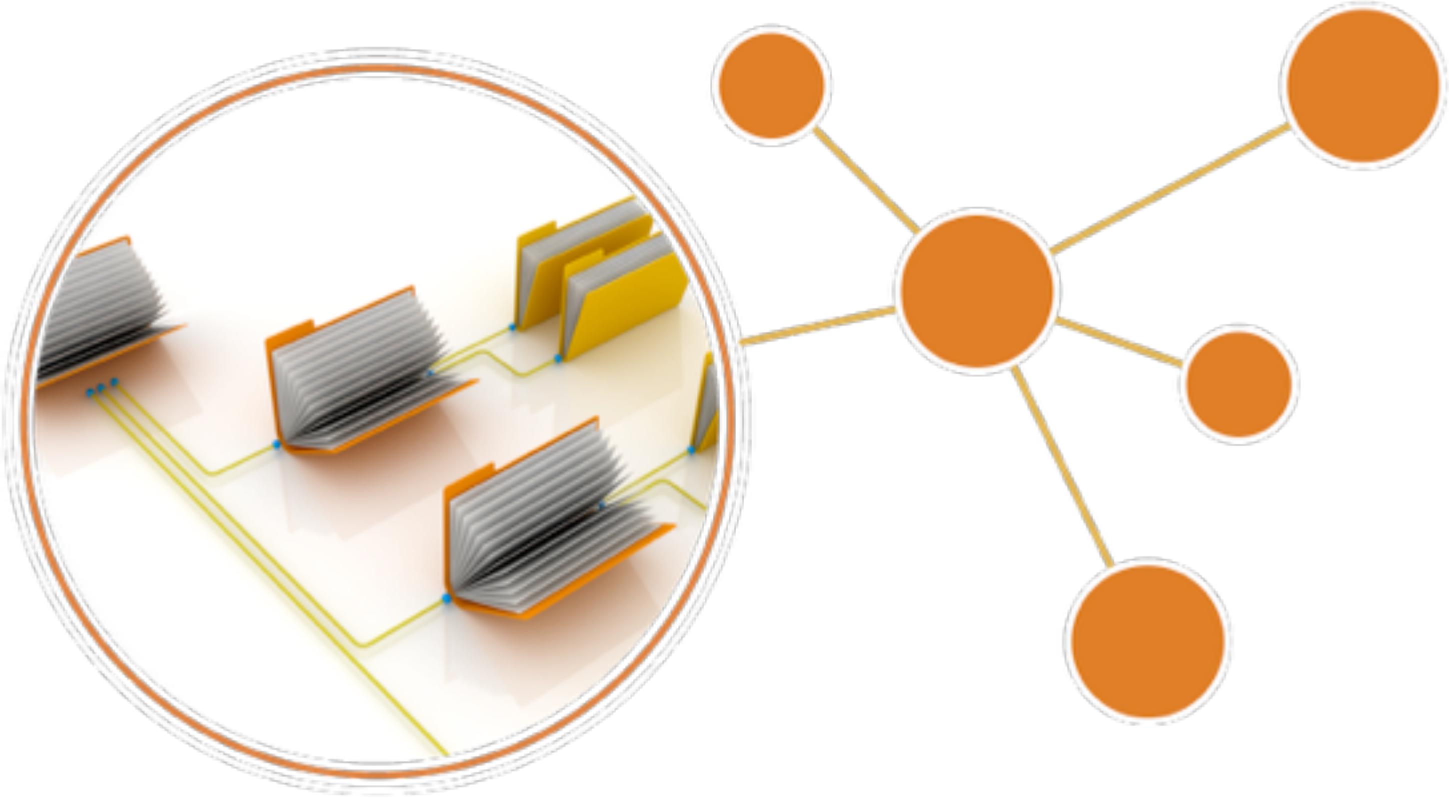
Impact Analysis



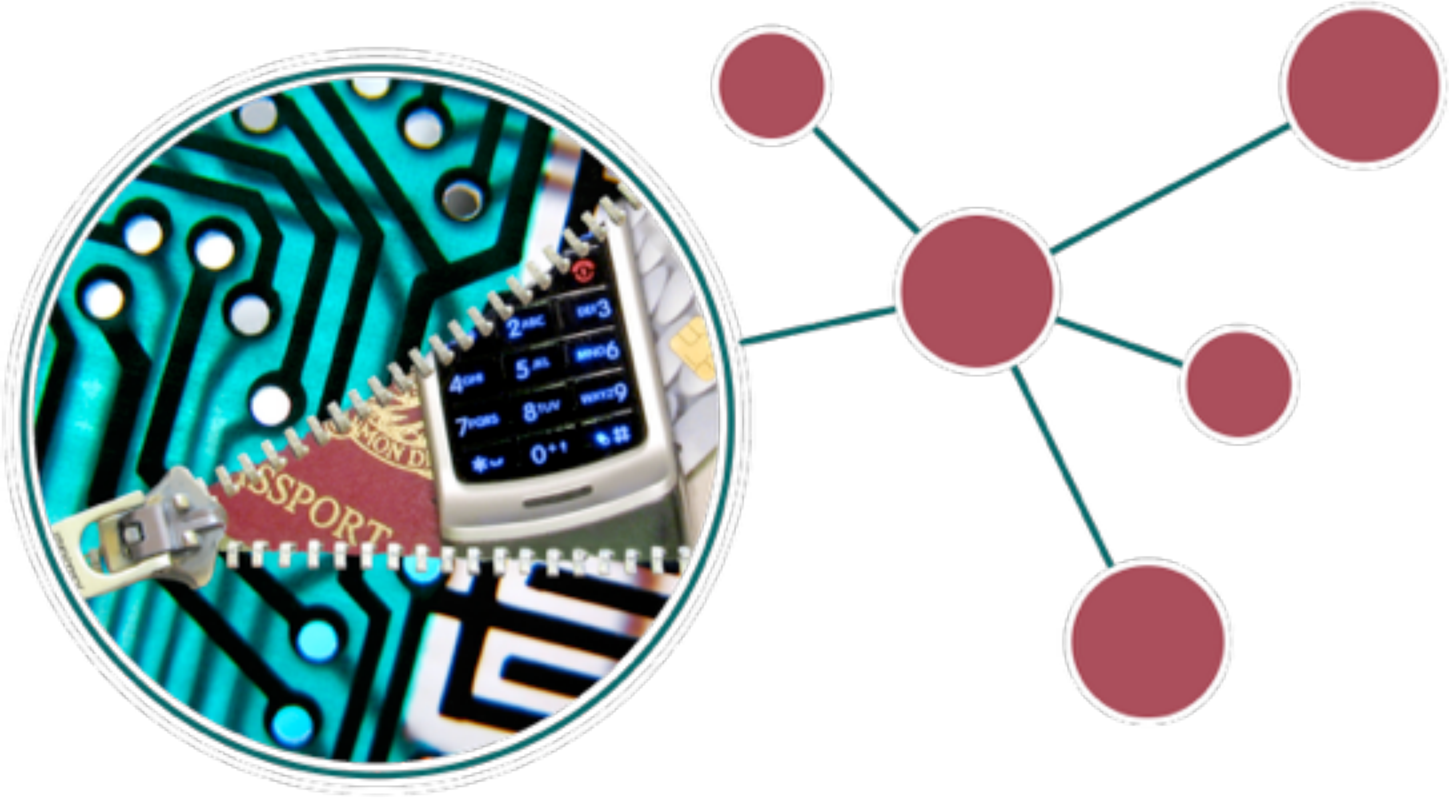
Logistics & Routing

Recommendations

Access Control



Fraud Analysis



Querying Graph Databases

Graph DBs and applications

- Graph DBs are crucial when topology is as important as data itself.
- Renewed interest due to new applications:
  - Semantic Web and RDF.
  - Social networks.
  - Security and crime detection.
  - Knowledge representation.
  - etc etc
  - ...

Querying graph DBs and relational technology

Why not to use relational technology?

- Translate graph DB $\mathcal{G} \rightarrow$ relational database $\mathcal{D}(\mathcal{G})$, and query $\mathcal{D}(\mathcal{G})$.

Problems:

1. Languages for graph DBs are navigational and require recursion.
2. They can be translated into Datalog, but there are problems:
   (a) Implementation:
   • SQL's recursion is hard to optimize, especially in complex queries, on large databases.
   (b) Complexity mismatch:
   • Datalog evaluation is $\textsc{Ptime}$-complete, but in $\textsc{NLogspace}$ for many graph languages.
   • Basic static analysis tasks undecidable for Datalog, but decidable for several graph languages.

Early graph query languages

Graph query languages flourished from the mid 80s to the late 90s:

- **G**, **G**$^+$, and GraphLog for hypertext and semistructured data, late 1980s
- GOOD for graph-based models of object DBs, 1990
- Hyperlog for hypergraphs, 1994
- Languages for heterogeneous and unstructured data, Lorel, StruQL, etc (late 1990s)

Features of graph query languages

- Navigation: Recursively traverse the edges of the graph.
- Pattern matching: Check if a pattern appears in the graph DB.

And more sophisticated features:

- Path comparisons.
- Comparisons of the underlying data.

Key problems theory studies:

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

Complexity of evaluation:

PROBLEM:	$\textsc{Eval}(\mathcal{L})$
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.

Containment: We study the problem $\textsc{Cont}(\mathcal{L})$:

- Given $\mathcal{L}$-queries Q_1, Q_2, is $Q_1(\mathcal{G}) \subseteq Q_2(\mathcal{G})$ for every graph DB $\mathcal{G}$?

Graph data model

Different applications have given rise to a many (slightly) different graph DB models. But the essence is the same:

Finite, directed, edge labeled graphs.

Despite the simplicity of the model:

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

Graph databases

Definition

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

$$(V, E)$$

finite set of node ids → V

E ← 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 $v_1 \xrightarrow{a} v_2$
$(v_1, v_2 \in V,\ a \in \Sigma)$

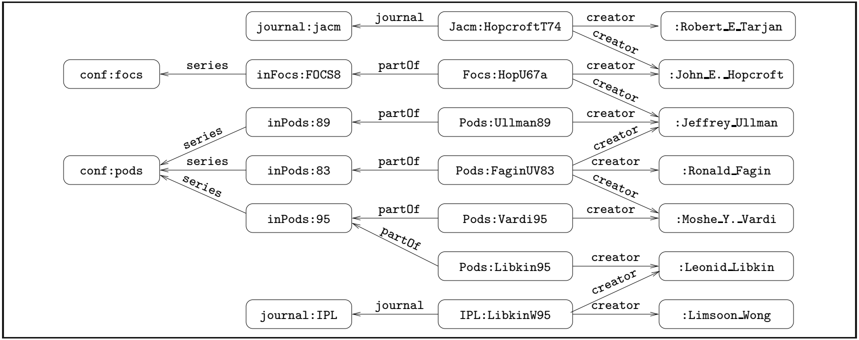
- A path in $\mathcal{G}$ is a sequence of the form:

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

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

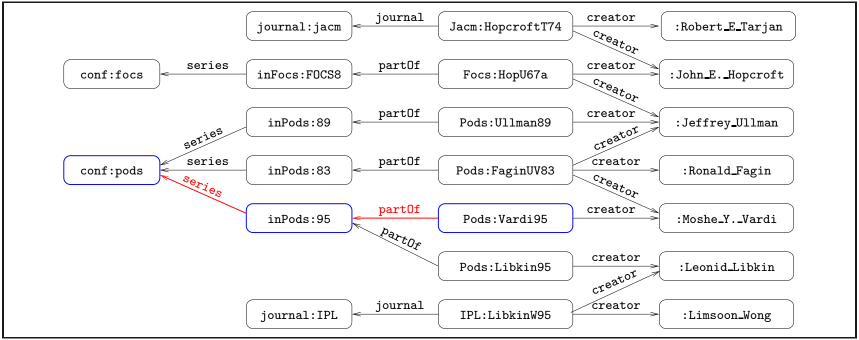
Graph DBs: Example

A graph DB representation of a fragment of DBLP:



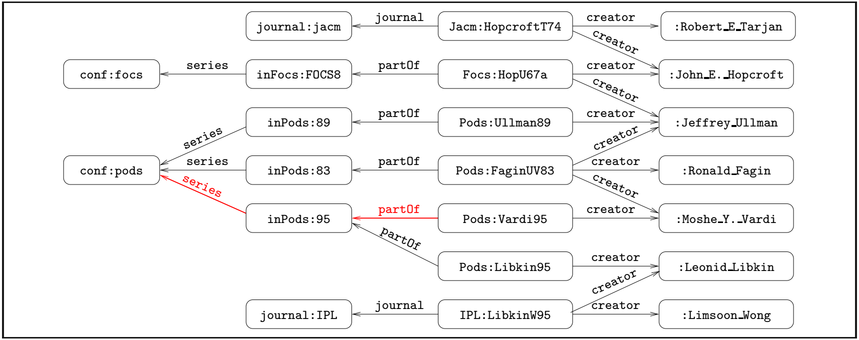
Graph DBs: Example

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.
Recall: NFA = Nondeterministic finite automaton.

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

Regular path queries

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

- First studied in 1989.
- An RPQ is a 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 in 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 in 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 $\mathcal{G} = (V, E)$:

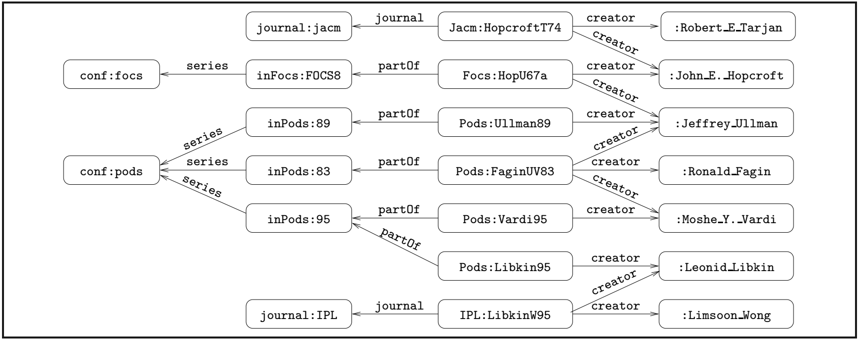
- Pairs of nodes in $\mathcal{G}$ that satisfy RPQ $L(\mathcal{G}^{\pm})$, where:
  - $\mathcal{G}^{\pm}$ obtained from $\mathcal{G}$ by adding $u \xrightarrow{a^{-}} v$ for each $v \xrightarrow{a} u \in E$.

Example of 2RPQ

The 2RPQ

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

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

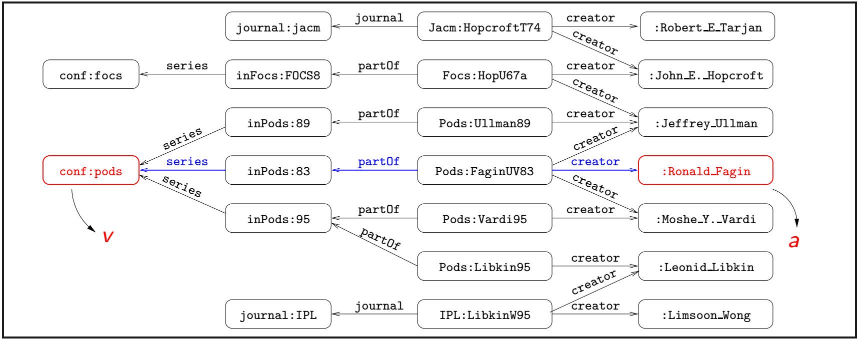


Example of 2RPQ

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.

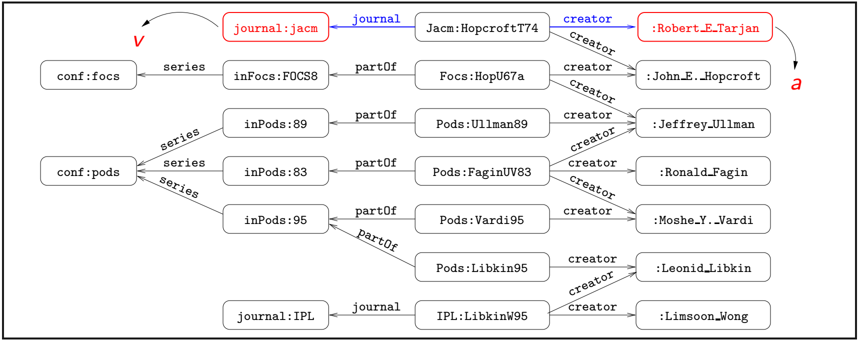


Example of 2RPQ

Example: The 2RPQ

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

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



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

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 $\mathcal{L}(\mathcal{G}^{\pm}, v, v') \cap \mathcal{L}(\mathcal{A}) \neq \emptyset$.
- For this need to solve the nonemptiness problem for the NFA $(\mathcal{G}^{\pm}, v, v') \times \mathcal{A}$.
- This can be done time $O(|\mathcal{G}^{\pm}| \cdot |\mathcal{A}|) = O(|\mathcal{G}| \cdot |L|)$.

Complexity of 2RPQ evaluation

2RPQs can be evaluated in linear time:

Corollary

$\textsc{Eval}(\text{2RPQ})$ *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 $\textsc{NLogspace}$.
- Check nonemptiness of $(\mathcal{G}^{\pm}, v, v') \times \mathcal{A}$ 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).

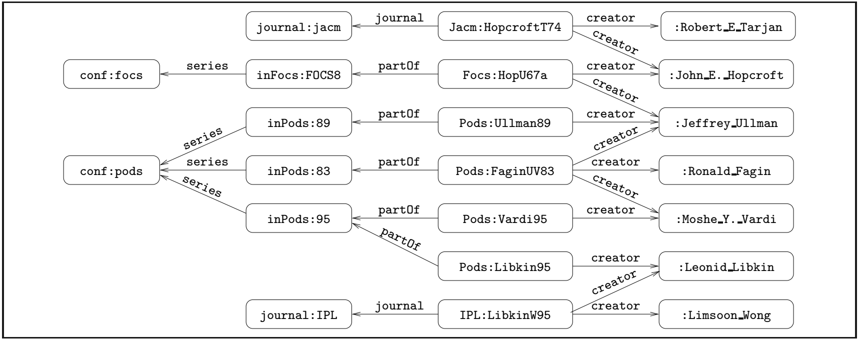
- Extended with inverses they are known as C2RPQs.

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.

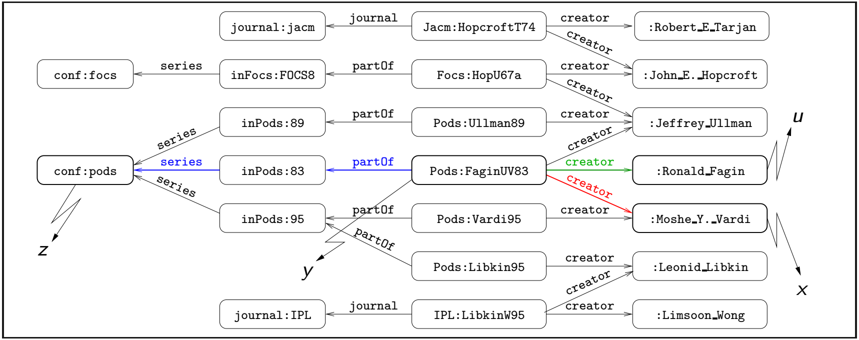


Example of C2RPQ

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.

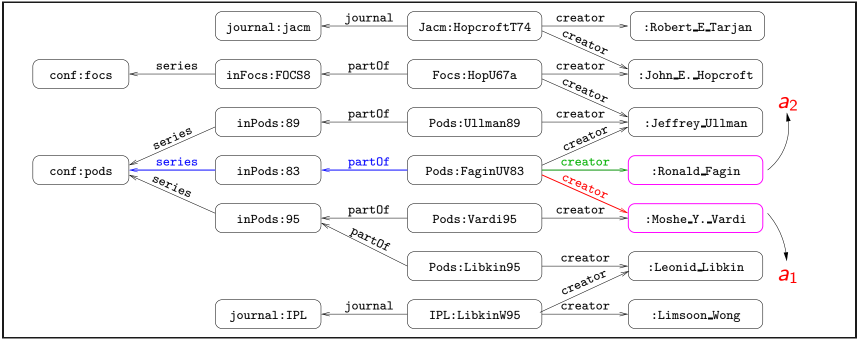


Example of C2RPQ

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.



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.

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.

CRPQ: C2RPQ without inverse.

Evaluation of C2RPQs

To evaluate C2RPQ $\varphi(\bar{z})$ of the form

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

simply evaluate the conjunctive query

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

where each $L_i(x_i, y_i)$ is the result of evaluating the 2RPQ L_i.

Can also see it as

$$\pi_{\bar{z}}(L_1 \bowtie \ldots \bowtie L_m)$$

Will write $\varphi(\mathcal{G})$.

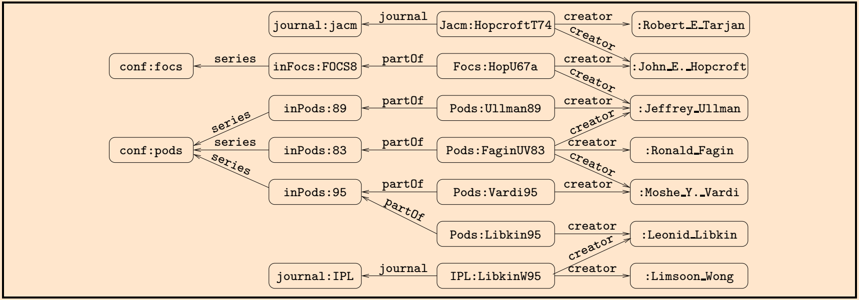
C2RPQs vs 2RPQs

Proposition

The C2RPQ

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

is not expressible as a 2RPQ L over the graph database:



Conclusion: Binary C2RPQs are strictly more expressive than 2RPQs.

Complexity of evaluation of C2RPQS

Increase in expressiveness has a cost in evaluation.

Proposition

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

- Upper bound by translation to evaluation of CQs.
- Lower bound holds since CRPQs contain CQs over graphs.

Data complexity of evaluation of (U)C2RPQS

But adding conjunctions is free in data complexity.

Proposition

EVAL(C2RPQ) *can be solved in* NLOGSPACE *in data complexity.*

Summary of basic query languages for graph DBs

- 2RPQs can be evaluated in linear time.
- 2RPQ evaluation is in NLOGSPACE in data complexity.
- For C2RPQs:
  - Retain good data complexity of 2RPQs.
  - Combined complexity is intractable.
- C2RPQs do not exhaust the NLOGSPACE properties.

Complexity of C2RPQs revisited

C2RPQs can be evaluated in polynomial time in data complexity, but is this a good measure for massive datasets?

CRPQ evaluation is of the order $|\mathcal{G}|^{O(|Q|)}$, which is impractical if $\mathcal{G}$ is very big even for small Q.

Idea: Look for languages that are tractable in combined complexity or, at least, fixed-parameter tractable (fpt).

- $\mathcal{L}$ is fpt if there is computable function $f : \mathbb{N} \to \mathbb{N}$ and constant $c \geq 0$ such that $\mathcal{L}$-queries can be evaluated in time $O(|\mathcal{G}|^c \cdot f(|\varphi|))$.

The landscape so far:

- 2RPQs are tractable in combined complexity ($O(|\mathcal{G}| \cdot |L|)$).
- CRPQs are intractable in combined complexity.
  CRPQs are not fpt (even CQs are not).

Structural restrictions of C2RPQs

Recall:

- Relational CQs are neither tractable in combined complexity nor fpt.
- Tractable cases of CQ evaluation can be obtained by restricting the syntactic shape of CQs.
- The most common such restriction is acyclicity.
  - An acyclic CQ Q can be evaluated in linear time $O(|\mathcal{D}| \cdot |Q|)$ over relational DB $\mathcal{D}$ (Yannakakis (1981)).
- Other restrictions include bounded (hyper-)treewidth.

Acyclic C2RPQs

A UC2RPQ is acyclic if its underlying CQ is acyclic.

A different way of stating this:

A C2RPQ Q is acyclic iff its underlying simple and undirected graph $\mathcal{U}(Q)$ is acyclic, where $\mathcal{U}(Q) = (V, E)$ for:

- $V = \{x_1, y_1, \ldots, x_m, y_m\}$;
- $E = \{\{x_i, y_i\} \mid 1 \leq i \leq m$ and $x_i \neq y_i\}$.

Remark: Acyclicity allows cycles of length ≤ 2 in C2RPQs.

- The C2RPQ $Ans() \leftarrow (x, a, x), (x, b, y), (y, c, x)$ is acyclic.

Acyclic C2RPQs: Examples

- The following C2RPQ is acyclic:

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

- The following C2RPQ is not acyclic:

$$Ans() \leftarrow (x, L_1, y), (y, L_2, z), (z, L_3, x).$$

Evaluation of acyclic C2RPQs

Evaluation of acyclic C2RPQs is tractable in combined complexity:

Proposition

Evaluation of an acyclic C2RPQ Q over a graph DB G takes time $O(|G|^2 \cdot |Q|^2)$.

The simple path semantics

Simple paths: No node is repeated.

Simple paths semantics:

- Motivated by applications for which simple paths are more natural.
- Studied back in the late 1980s already.
- Revival due to application in early versions of SPARQL, a language for RDF.

RPQs under simple paths semantics

- RPQ evaluation in this context = Finding regular simple paths:

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

RPQs under simple paths semantics

- RPQ evaluation in this context = Finding regular simple paths:

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

- REGULARSIMPLEPATH(L): For fixed L.

Complexity of finding regular simple paths

Theorem

The problem $\textsc{RegularSimplePath}$ *is in* NP, *and for some L the problem* $\textsc{RegularSimplePath}$*(L) can be* NP*-complete.*

- $\textsc{RegularSimplePath}((00)^*)$:
- Is there simple directed path of even length? It is NP-complete.
- Query evaluation is NP-complete in data complexity – hence impractical.

Static analysis: Containment for 2RPQs

$\textsc{Cont}(\mathcal{L})$: Given $\mathcal{L}$-queries Q_1 and Q_2,

- is $Q_1(\mathcal{G}) \subseteq Q_2(\mathcal{G})$ for each graph DB $\mathcal{G}$?

Static analysis: Containment for 2RPQs

$\textsc{Cont}(\mathcal{L})$: Given $\mathcal{L}$-queries Q_1 and Q_2,

- is $Q_1(\mathcal{G}) \subseteq Q_2(\mathcal{G})$ for each graph DB $\mathcal{G}$?

Containment for 2RPQs is decidable:

Theorem

$\textsc{Cont}$(2RPQ) *is* $\textsc{Pspace}$*-complete. It is* $\textsc{Pspace}$*-hard even for RPQs.*

- For RPQs easy to prove:
  - $L_1(\mathcal{G}) \subseteq L_2(\mathcal{G})$ for each $\mathcal{G}$ $\Longleftrightarrow$ regular expression L_1 contained in regular expression L_2.
  - Containment of regular expressions: $\textsc{Pspace}$-complete (Stock+1)Meyer (1971)).
- For 2RPQs more work is required: Reason with two-way automata.

Containment for C2RPQs

Containment of C2RPQs still decidable with exponential blow-up:

Theorem

CONT(C2RPQ) *is* EXPSPACE*-complete, even for CRPQs.*

- Notice contrast with complexity of containment for CQs:
  - NP-complete (Chandra,Merlin (1977)).

Summary of containment

- Containment of C2RPQs is decidable in double exponential time.
- For 2RPQs containment can be checked in single exponential time.
- High lower bounds are due to the presence of regular expressions.

Path queries and comparisons

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.

Path queries and comparisons

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

Evaluation of ECRPQs

Evaluation of the ECRPQ($\mathcal{S}$)

$$\theta(\bar{z}, \bar{\chi}) : Ans(\bar{z}, \bar{\chi}) \leftarrow (x_1, \pi_1, y_1), \ldots, (x_m, \pi_m, y_m), \bigwedge_j S_j(\bar{\pi}_j)$$

Same than for CRPQs but:

- Each π_i is mapped to a path ρ_i in the graph DB.
- For each j, if $\bar{\pi}_j = (\pi_{j_1}, \ldots, \pi_{j_k})$ then: $\underbrace{(\lambda(\rho_{j_1}), \ldots, \lambda(\rho_{j_k}))}_{\text{the labels of } (\rho_{j_1}, \ldots, \rho_{j_k})} \in S_j$.

Evaluation of ECRPQs

Evaluation of the ECRPQ($\mathcal{S}$)

$$\theta(\bar{z}, \bar{\chi}) : Ans(\bar{z}, \bar{\chi}) \leftarrow (x_1, \pi_1, y_1), \ldots, (x_m, \pi_m, y_m), \bigwedge_j S_j(\bar{\pi}_j)$$

Same than for CRPQs but:

- Each π_i is mapped to a path ρ_i in the graph DB.
- For each j, if $\bar{\pi}_j = (\pi_{j_1}, \ldots, \pi_{j_k})$ then: $\underbrace{(\lambda(\rho_{j_1}), \ldots, \lambda(\rho_{j_k}))}_{\text{the labels of } (\rho_{j_1}, \ldots, \rho_{j_k})} \in S_j$.

Evaluation of ECRPQs

Evaluation of the ECRPQ($\mathcal{S}$)

$$\theta(\bar{z}, \bar{\chi}) : Ans(\bar{z}, \bar{\chi}) \leftarrow (x_1, \pi_1, y_1), \ldots, (x_m, \pi_m, y_m), \bigwedge_j S_j(\bar{\pi}_j)$$

Same than for CRPQs but:

- Each π_i is mapped to a path ρ_i in the graph DB.
- For each j, if $\bar{\pi}_j = (\pi_{j_1}, \ldots, \pi_{j_k})$
  then: $\underbrace{(\lambda(\rho_{j_1}), \ldots, \lambda(\rho_{j_k}))}_{\text{the labels of } (\rho_{j_1}, \ldots, \rho_{j_k})} \in S_j.(\lambda(\rho_{j_1}), \ldots, \lambda(\rho_{j_k})) \in S_j$.

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

- ECRPQ($\mathcal{S}$) extends the class of CRPQs.
  - $Ans(\bar{z}) \leftarrow \bigwedge_i (x_i, L_i, y_i)$ same as $Ans(\bar{z}) \leftarrow \bigwedge_i (x_i, \pi_i, y_i), L_i(\pi_i)$.
- 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

- 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

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

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

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

Synchronous automata

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

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

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

Synchronous automata

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

Synchronous automata

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

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 $k \geq 0$.

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 $k \geq 0$.

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

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

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

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 for data complexity.
• Combined complexity is that of relational calculus over relational databases.

Theorem

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

Containment for ECRPQ(REG)

Theorem

$\textsc{Cont}$(ECRPQ(REG)) *is undecidable.*

- Notice contrast with CRPQs for which containment is decidable.
- But this is like for full relational algebra/calculus.

Comparing with rational relations

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:

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

- 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:

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

- 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

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

Evaluation of $(\mathrm{ECRPQ}(\mathrm{REG} \cup \{\preceq_{\mathrm{sw}}\}))$ *queries is undecidable. The same is true for suffix in place of subword.*

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

Evaluation of $(\mathrm{ECRPQ}(\mathrm{REG} \cup \{\preceq_{\mathrm{sw}}\}))$ *queries is undecidable. The same is true for suffix in place of subword.*

Adding subsequence preserves decidability, but at a very high cost:

Theorem

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

Primitive-recursive, informally: any function you can think of!

Acyclic ECRPQ(RAT) queries

Acyclic ECRPQ(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) queries

Acyclic ECRPQ(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

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

Summary of path queries

- Usual query languages do not allow:
  - to export paths and compare labels of paths.
- This has led to the introduction of ECRPQ($\mathcal{S}$) queries:
  - They output paths and compare labels of paths with relations in $\mathcal{S}$.
- Comparing paths with regular relations:
  - Preserves tractable data complexity of evaluation.
  - Leads to undecidability of containment.
- Comparing paths with practically motivated rational relations:
  - Leads to undecidability or high complexity of evaluation.
  - Tractable cases found restricting the syntactic shape of queries.

Querying graphs with data

So far queries only talk about the topology of the data.

Queries that combine topology and data are important in practice:

- Example:
  People of the same age connected by professional links.

We present a language that expresses topological properties of the data:

- It requires an extension of the data model (data graphs).
- It talks about data paths:
  Summarize the topology and the underlying data of a path.

Data graphs and data paths

We work with data graphs and paths over set of data values $\mathcal{D}$.

Definition

A data graph $\mathfrak{G}$ over Σ is a tuple (V, E, δ), where:

- (V, E) is a graph database over Σ, and
- δ is a mapping that assigns a value in $\mathcal{D}$ to each node $v \in V$.

Data graphs and data paths

We work with data graphs and paths over set of data values $\mathcal{D}$.

Definition

A data graph $\mathfrak{G}$ over Σ is a tuple (V, E, δ), where:

- (V, E) is a graph database over Σ, and
- δ is a mapping that assigns a value in $\mathcal{D}$ to each node $v \in V$.

With each path $\rho = v_1 \xrightarrow{a_1} v_2 \cdots v_k \xrightarrow{a_k} v_{k+1}$ in (V, E):
We associate a data path in $\mathfrak{G}$ of the form

$$\rho_{\mathcal{D}} \;=\; \delta(v_1) \xrightarrow{a_1} \delta(v_2) \cdot \cdot \cdot \delta(v_k) \xrightarrow{a_k} \delta(v_{k+1}),$$

that is obtained from ρ by replacing each node by its data value.

Data paths and data words

Data paths are very close to data words:

- Object studied in XML and verification (Bojanczyk et al. (2006)).
- Data words are strings over $\Sigma \times \mathcal{D}$.

Mechanisms that query data words can be used for data paths:

- FO, MSO, and some versions of XPath (Bojanczyk et al. (2006)).
- Pebble automata (Neven, Schwentick, Vianu (2004)).
- Register automata (Kaminski, Francez (1994)).

The choice of a formalism

Formalism for querying data paths has to be chosen with care:

Theorem

The problem $\textsc{DistinctValues}$ *is* NP*-complete:*

- $\textsc{DistinctValues}$*:*
  Is there a path ρ *from* v *to* v' *s.t. no data value in* $\rho_{\mathcal{D}}$ *is repeated?*

The choice of a formalism

Formalism for querying data paths has to be chosen with care:

Theorem

The problem $\textsc{DistinctValues}$ *is* NP*-complete:*

- $\textsc{DistinctValues}$*:*
  Is there a path ρ *from* v *to* v' *s.t. no data value in* $\rho_{\mathcal{D}}$ *is repeated?*

Conclusion:

- If a language expresses $\textsc{DistinctValues}$:
  - It is NP-hard in data complexity $\Rightarrow$ Impractical.
- Rules out all formalisms except for one:
  - Register automata.

Regular expressions for register automata

Regular expressions with memory (REMs):
Same as register automata

Regular expressions for register automata

Regular expressions with memory (REMs):
Same as register automata

- REMs permit to specify when data values are remembered and used.
- Data values are remembered in k registers $\{x_1, \ldots, x_k\}$.
- At any point we can compare a data value with one in the registers.

REM: Example

Consider the REM $\downarrow x.a^{+}[x^{=}]$.

Intuition:

- Store the current data value d in register x.
- After reading a word in a^{+} check that d is seen again.

Semantics: Pairs (v, v') of nodes:

- Linked by a path labeled in a^{+}.
- v and v' contain the same data value.

REM: Conditions

- Conditions: Compare a data value with the ones in the registers.
- Conditions over $\{x_1, \ldots, x_k\}$ are given by the grammar:

$$c \;:=\; x_i^= \mid \neg c \mid c \wedge c \qquad (1 \leq i \leq k)$$

- We define $(d, \tau) \models c$ for $d \in \mathcal{D}$ and $\tau = (d_1, \ldots, d_k) \in \mathcal{D}^k$:
  - $(d, \tau) \models x_i^=$ iff $d = d_i$.
  - Boolean combinations are standard.

REMs: Syntax and semantics (Intuition)

REMs over Σ and $\{x_1, \ldots, x_k\}$ are defined by grammar:

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

where $a \in \Sigma$, c condition, and $\bar{x}$ tuple in $\{x_1, \ldots, x_k\}$.

REMs: Syntax and semantics (Intuition)

REMs over Σ and $\{x_1, \ldots, x_k\}$ are defined by grammar:

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

where $a \in \Sigma$, c condition, and $\bar{x}$ tuple in $\{x_1, \ldots, x_k\}$.

Intuition: Evaluation of REM e on data graph $\mathfrak{G}$ is:

- pairs (v, v') of nodes linked by path ρ such that $\rho_{\mathcal{D}} \models e$, where:

REMs: Syntax and semantics (Intuition)

REMs over Σ and $\{x_1, \ldots, x_k\}$ are defined by grammar:

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

where $a \in \Sigma$, c condition, and $\bar{x}$ tuple in $\{x_1, \ldots, x_k\}$.

Intuition: Evaluation of REM e on data graph $\mathfrak{G}$ is:

- pairs (v, v') of nodes linked by path ρ such that $\rho_{\mathcal{D}} \models e$, where:

  - $\rho_{\mathcal{D}} \models e[c]$ if and only if

$\rho_{\mathcal{D}}$

$$\underbrace{\kappa(v_1) \xrightarrow{a_1} \kappa(v_2) \cdots \kappa(v_k) \xrightarrow{a_k} \kappa(v_{k+1})}_{\text{can be parsed wrt } e}$$

starting from empty registers

finishing in register value $\tau \in \mathcal{D}^k$ st $(\kappa(v_{k+1}), \tau) \models c$

REMs: Syntax and semantics (Intuition)

REMs over Σ and $\{x_1, \ldots, x_k\}$ are defined by grammar:

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

where $a \in \Sigma$, c condition, and $\bar{x}$ tuple in $\{x_1, \ldots, x_k\}$.

Intuition: Evaluation of REM e on data graph $\mathfrak{G}$ is:

- pairs (v, v') of nodes linked by path ρ such that $\rho_{\mathcal{D}} \models e$, where:
  - $\rho_{\mathcal{D}} \models \downarrow \bar{x}.e$ if and only if

$\rho_{\mathcal{D}}$

$$\underbrace{\kappa(v_1) \xrightarrow{a_1} \kappa(v_2) \cdots \kappa(v_k) \xrightarrow{a_k} \kappa(v_{k+1})}_{\text{can be parsed wrt } e}$$

starting from the register value
that assigns $\kappa(v_1)$ to each $x \in \bar{x}$

REM: Example

Consider the REM $\Sigma^* \cdot (\downarrow x.\Sigma^+[x^=]) \cdot \Sigma^*$:

- Defines pairs of nodes linked by path ρ such that:
  - $\rho_{\mathcal{D}}$ contains the same data value twice.
- The complement of this language is $\textsc{DistinctValues}$.

REM: Example

Consider the REM $\Sigma^* \cdot (\downarrow x.\Sigma^+[x^=]) \cdot \Sigma^*$:

- Defines pairs of nodes linked by path ρ such that:
  - $\rho_{\mathcal{D}}$ contains the same data value twice.
- The complement of this language is DistinctValues.

Corollary

REMs are not closed under complement.

Complexity of REM evaluation

- Data complexity of REM evaluation coincides with that of CRPQs.
- Combined complexity same than for FO over relational databases.

Theorem

- EVAL(REM) *is* PSPACE*-complete.*
- *It is in* NLOGSPACE *in data complexity.*

- Both bounds extend to the class of conjunctive REMs.

Summary of queries on graphs with data

- Most query languages for graph DBs:
  - talk about topology, but not about underlying data.
- Query languages that combine topology and data:
  - talk about data paths in data graphs.
- Choosing a formalism to query data paths must be done with care:
  - intractability can be reached easily.
- To query data paths:
  - Can use REMs, which are based on register automata.
  - REMs can be evaluated efficiently in data complexity.

Comments on papers

- Isabel F. Cruz, Alberto O. Mendelzon, Peter T. Wood: A Graphical Query Language Supporting Recursion. SIGMOD Conference 1987: 323-330
- Mariano P. Consens, Alberto O. Mendelzon: Low Complexity Aggregation in GraphLog and Datalog. Theor. Comput. Sci. 116(1): 95-116 (1993)

  Original papers introducing (C)RPQs
- Pablo Barcelo: Querying graph databases. PODS 2013: 175-188
- Renzo Angles, Claudio Gutiérrez: Survey of graph database models. ACM Comput. Surv. 40(1) (2008)
- Peter T. Wood: Query languages for graph databases. SIGMOD Record 41(1): 50-60 (2012)

  Three suveys of graph languages, two are more theoretical, one more practical.
- Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Moshe Y. Vardi: Rewriting of Regular Expressions and Regular Path Queries. J. Comput. Syst. Sci. 64(3): 443-465 (2002)

  Introducing two-way queries.

Comments on papers

- Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Moshe Y. Vardi: Reasoning on regular path queries. SIGMOD Record 32(4): 83-92 (2003)
- Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Moshe Y. Vardi: Containment of Conjunctive Regular Path Queries with Inverse. KR 2000: 176-185

  Static analysis of regular path queries.
- Leonid Libkin, Wim Martens, Domagoj Vrgoc: Querying graph databases with XPath. ICDT 2013: 129-140

  Adding data values to (C)RPQs
- Pablo Barcelo, Leonid Libkin, Anthony Widjaja Lin, Peter T. Wood: Expressive Languages for Path Queries over Graph-Structured Data. ACM Trans. Database Syst. 37(4): 31 (2012)

  Extending RPQs with regular relations; topics to concentrate on are those not covered in class.
- Pablo Barcelo, Diego Figueira, Leonid Libkin: Graph Logics with Rational Relations .Logical Methods in Computer Science 9(3) (2013)

  Likewise for rational relations.

Comments on papers

- Dominik D. Freydenberger, Nicole Schweikardt: Expressiveness and Static Analysis of Extended Conjunctive Regular Path Queries. AMW 2011

  Resolving some of the questions on the containment of path queries.
- Jelle Hellings, Bart Kuijpers, Jan Van den Bussche, Xiaowang Zhang: Walk logic as a framework for path query languages on graph databases. ICDT 2013: 117-128

  A different approach to expanding the power of path languages.
- Pablo Barcelo, Leonid Libkin, Juan L. Reutter: Querying Regular Graph Patterns. Journal of the ACM 61(1): 8:1-8:54 (2014)

  Incomplete information in graph databases and querying it.
- Wenfei Fan, Xin Wang, Yinghui Wu: Querying big graphs within bounded resources. SIGMOD Conference 2014: 301-312
- Wenfei Fan: Graph pattern matching revised for social network analysis. ICDT 2012: 8-21

  Two papers on making graph queries scalable.