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 $G \rightarrow$ relational database $D(G)$, and query $D(G)$.

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 $\text{PTIME}$-complete, but in $\text{NLOGSPACE}$ 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:

- \textbf{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:

<table>
<thead>
<tr>
<th>Problem:</th>
<th>\text{Eval}(\mathcal{L})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input:</td>
<td>A graph DB ( \mathcal{G} ), a tuple ( \bar{t} ) of objects, an ( \mathcal{L} )-query ( Q ).</td>
</tr>
<tr>
<td>Question:</td>
<td>Is ( \bar{t} \in Q(\mathcal{G}) )?</td>
</tr>
</tbody>
</table>

- 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 \( \text{Cont}(\mathcal{L}) \):

- 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 $\Sigma$ 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 $\Sigma$ 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 $\rho$ 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:
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 $\Sigma$.
- Evaluation $L(G)$ of RPQ $L$ on graph DB $G = (V, E)$:
  - 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

\[
(\text{creator}^- \cdot ((\text{partOf} \cdot \text{series}) \cup \text{journal}))
\]

computes \((a, v)\) s.t. author \(a\) published in conference or journal \(v\).
Example of 2RPQ

The 2RPQ

\[
\left( \text{creator}^- \cdot ((\text{partOf} \cdot \text{series}) \cup \text{journal}) \right)
\]

computes \((a, v)\) s.t. author \(a\) published in conference or journal \(v\).
Example of 2RPQ

**Example:** The 2RPQ

\[
\left( \text{creator}^- \cdot ((\text{partOf} \cdot \text{series}) \cup \text{journal}) \right)
\]

computes \((a, v)\) s.t. author \(a\) published in conference or journal \(v\).
**Problem:** \( \text{Eval}(2\text{RPQ}) \)

**Input:** A graph DB \( G \), nodes \( v, v' \) in \( G \), a 2RPQ \( L \).

**Question:** Is \( (v, v') \in L(G) \)?
2RPQ evaluation

**Problem:** Eval(2RPQ)

**Input:** A graph DB $G$, nodes $v$, $v'$ in $G$, a 2RPQ $L$.

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

It boils down to:

**Problem:** RegularPath

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

**Question:** Is there a path $\rho$ from $v$ to $v'$ in $G^\pm$ such that $\lambda(\rho) \in L$?
Complexity of finding regular paths

**Theorem**

The `RegularPath` can be solved in time $O(|G| \cdot |L|)$.

**Proof idea:**

- Compute in linear time from $L$ an equivalent NFA $A$.
- Compute in linear time $(G^\pm, v, v')$: NFA obtained from $G^\pm$ by setting $v$ and $v'$ as initial and final states, respectively.
- Then $(v, v') \in L(G)$ iff $L(G^\pm, v, v') \cap L(A) \neq \emptyset$.
- For this need to solve the nonemptiness problem for the NFA $(G^\pm, v, v') \times A$.
- This can be done time $O(|G^\pm| \cdot |A|) = O(|G| \cdot |L|)$. 
Complexity of 2RPQ evaluation

2RPQs can be evaluated in linear time:

**Corollary**

\[
\text{Eval}(2\text{RPQ}) \text{ can be solved in linear time } O(|G| \cdot |L|).
\]
Data complexity of 2RPQ evaluation

Data complexity of 2RPQs belongs to a parallelizable class:

**Proposition**

Let $L$ be a fixed 2RPQ. 
There is $\text{NLOGSPACE}$ procedure that computes $L(G)$ for each $G$.

**Proof idea:**

- Construct $(G^\pm, v, v')$ from $G$ in $\text{NLOGSPACE}$.
- Check nonemptiness of $(G^\pm, v, v') \times A$ in $\text{NLOGSPACE}$.
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

\[ \text{Ans}(x, u) \leftarrow (x, \text{creator}^-, y), (y, \text{partOf} \cdot \text{series}, z), (y, \text{creator}, u) \]

computes pairs \((a_1, a_2)\) that are coauthors of a conference paper.
Example of C2RPQ

The C2RPQ

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

computes pairs $$(a_1, a_2)$$ that are coauthors of a conference paper.
Example of C2RPQ

The C2RPQ

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

computes pairs \((a_1, a_2)\) that are coauthors of a conference paper.
C2RPQ: Formal definition

**C2RPQ over Σ:** Rule of the form:

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

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 $\Sigma$: 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 $\Sigma$,
- 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

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

simply evaluate the conjunctive query

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

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

Can also see it as

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

Will write $\varphi(G)$. 
C2RPQs vs 2RPQs

**Proposition**

The C2RPQ

\[
\text{Ans}(x, u) \leftarrow (x, \text{creator}^-, y), (y, \text{partOf} \cdot \text{series}, z), (y, \text{creator}, u)
\]

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**

$\text{Eval}(\text{C2RPQ})$ 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**

\( \text{Eval(C2RPQ)} \) *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 $\mathsf{NLogspace}$ in data complexity.
- For C2RPQs:
  - Retain good data complexity of 2RPQs.
  - Combined complexity is intractable.
- C2RPQs do not exhaust the $\mathsf{NLogspace}$ 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 $|G|^{O(|Q|)}$, which is impractical if $G$ is very big even for small $Q$.

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

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

The landscape so far:

- 2RPQs are tractable in combined complexity ($O(|G| \cdot |L|)$).
- 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 acyclicility.
  - An acyclic CQ $Q$ can be evaluated in linear time $O(|D| \cdot |Q|)$ over relational DB $D$ (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:

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

**Remark:** Acyclicity allows cycles of length $\leq 2$ in C2RPQs.

- The C2RPQ $\text{Ans()} \leftarrow (x, a, x), (x, b, y), (y, c, x)$ is acyclic.
Acyclic C2RPQs: Examples

- The following C2RPQ is acyclic:

  \[ \text{Ans}(x, u) \leftarrow (x, \text{creator}^-, y), (y, \text{partOf} \cdot \text{series}, z), (y, \text{creator}, u). \]

- The following C2RPQ is not acyclic:

  \[ \text{Ans}() \leftarrow (x, L_1, y), (y, L_2, z), (z, L_3, x). \]
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:

<table>
<thead>
<tr>
<th><strong>Problem:</strong></th>
<th><strong>RegularSimplePath</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong></td>
<td>A graph database $\mathcal{G}$, nodes $v, v'$ in $\mathcal{G}$, a regular expression $L$.</td>
</tr>
<tr>
<td><strong>Question:</strong></td>
<td>Is there a simple path $\rho$ from $v$ to $v'$ in $\mathcal{G}$ such that $\lambda(\rho) \in L$?</td>
</tr>
</tbody>
</table>
RPQs under simple paths semantics

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

<table>
<thead>
<tr>
<th><strong>Problem:</strong></th>
<th><strong>RegularSimplePath</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong></td>
<td>A graph database $G$, nodes $v$, $v'$ in $G$, a regular expression $L$.</td>
</tr>
<tr>
<td><strong>Question:</strong></td>
<td>Is there a simple path $\rho$ from $v$ to $v'$ in $G$ such that $\lambda(\rho) \in L$?</td>
</tr>
</tbody>
</table>

• $\text{RegularSimplePath}(L)$: For fixed $L$. 
Complexity of finding regular simple paths

**Theorem**

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

- **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

**CONT(\mathcal{L})**: Given \mathcal{L}-queries \( Q_1 \) and \( Q_2 \),
- is \( Q_1(G) \subseteq Q_2(G) \) for each graph DB \( G \)?
Static analysis: Containment for 2RPQs

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

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

Containment for 2RPQs is decidable:

**Theorem**

Cont(2RPQ) is PSPACE-complete. It is PSPACE-hard even for RPQs.

- For RPQs easy to prove:
  - $L_1(G) \subseteq L_2(G)$ for each $G$ \iff regular expression $L_1$ contained in regular expression $L_2$.
- For 2RPQs more work is required: Reason with two-way automata.
Containment for C2RPQs

Containment of C2RPQs still decidable with exponential blow-up:

**Theorem**

$\text{Cont}(\text{C2RPQ})$ is \text{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 $S$ of relations on words.

- **Example:** $S$ 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 $S$ contains all regular languages.
Extended CRPQs

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

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

- by annotating each pair $(x_i, y_i)$ with a path variable $\pi_i$,
- comparing labels of paths in $\bar{\pi}_j$ wrt $S_j \in S$.
  - for $\bar{\pi}_j$ a tuple of path variables among the $\pi_i$’s,
- projecting some of $\pi_i$’s as a tuple $\bar{\chi}$ in the output.
Extended CRPQs

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

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

- by annotating each pair $(x_i, y_i)$ with a path variable $\pi_i$,
- comparing labels of paths in $\bar{\pi}_j$ wrt $S_j \in S$
  - for $\bar{\pi}_j$ a tuple of path variables among the $\pi_i$’s,
- projecting some of $\pi_i$’s as a tuple $\bar{\chi}$ in the output.
Extended CRPQs

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

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

- by annotating each pair $(x_i, y_i)$ with a path variable $\pi_i$,
- comparing labels of paths in $\bar{\pi}_j$ wrt $S_j \in S$
  - for $\bar{\pi}_j$ a tuple of path variables among the $\pi_i$’s,
- projecting some of $\pi_i$’s as a tuple $\chi$ in the output.
Extended CRPQs

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

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

- by annotating each pair $(x_i, y_i)$ with a path variable $\pi_i$,
- comparing labels of paths in $\bar{\pi}_j$ wrt $S_j \in S$
  - for $\bar{\pi}_j$ a tuple of path variables among the $\pi_i$’s,
- projecting some of $\pi_i$’s as a tuple $\bar{\chi}$ in the output.
Extended CRPQs and our requirements

ECRPQs meet our requirements:

\[
\text{Ans}(\bar{z}, \bar{\chi}) \leftarrow (x_1, \pi_1, y_1), \ldots, (x_m, \pi_m, y_m), \bigwedge_{1 \leq j \leq t} S_j(\bar{\pi}_j)
\]
Extended CRPQs and our requirements

ECRPQs meet our requirements:

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

- They allow paths in the output.
- They allow to compare labels of paths with relations \( S_j \in S \).
Extended CRPQs and our requirements

ECRPQs meet our requirements:

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

- They allow paths in the output.
- They allow to compare labels of paths with relations \( S_j \in S \).
Evaluation of ECRPQs

Evaluation of the ECRPQ($S$)

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

Same than for CRPQs but:

- Each $\pi_i$ is mapped to a path $\rho_i$ in the graph DB.
- For each $j$, if $\bar{\pi}_j = (\pi_{j_1}, \ldots, \pi_{j_k})$ then: $\left(\lambda(\rho_{j_1}), \ldots, \lambda(\rho_{j_k})\right) \in S_j$.
Evaluation of ECRPQs

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

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

Same than for CRPQs but:

- Each $\pi_i$ is mapped to a path $\rho_i$ in the graph DB.
- For each $j$, if $\bar{\pi}_j = (\pi_{j_1}, \ldots, \pi_{j_k})$ then: $\langle \lambda(\rho_{j_1}), \ldots, \lambda(\rho_{j_k}) \rangle \in S_j$.  

the labels of $(\rho_{j_1}, \ldots, \rho_{j_k})$
Evaluation of ECRPQs

Evaluation of the ECRPQ(\(S\))

\[ \theta(\vec{z}, \vec{\chi}) : \text{Ans}(\vec{z}, \vec{\chi}) \leftarrow (x_1, \pi_1, y_1), \ldots, (x_m, \pi_m, y_m), \land_j S_j(\vec{\pi}_j) \]

Same than for CRPQs but:

- Each \(\pi_i\) is mapped to a path \(\rho_i\) in the graph DB.
- For each \(j\), if \(\vec{\pi}_j = (\pi_{j_1}, \ldots, \pi_{j_k})\) then: \(\langle \lambda(\rho_{j_1}), \ldots, \lambda(\rho_{j_k}) \rangle \in S_j \cdot (\lambda(\rho_{j_1}), \ldots, \lambda(\rho_{j_k})) \in S_j\).

the labels of \((\rho_{j_1}, \ldots, \rho_{j_k})\)
Considerations about ECRPQ(\(S\))

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

- Expressiveness and complexity of ECRPQ(\(S\)):
  - Depends on the class \(S\).

- 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

\textit{n-ary regular relation:}

Set of \(n\)-tuples \((w_1, \ldots, w_n)\) of strings accepted by \textit{synchronous} automaton over \(\Sigma^n\).
Regular relations

\textit{n-ary regular relation:}

Set of \( n \)-tuples \((w_1, \ldots, w_n)\) of strings accepted by \textbf{synchronous} automaton over \( \Sigma^n \).

\begin{itemize}
\item The input strings are written in the \( n \)-tapes.
\item Shorter strings are padded with symbol \( \bot \).
\item At each step:
  \begin{itemize}
  \item The automaton simultaneously reads next symbol on each tape.
  \end{itemize}
\end{itemize}
Synchronous automata

\[ w_1 = a \ a \ b \ \ldots \ a \ b \ c \]
\[ w_2 = a \ b \ a \ \ldots \ a \]
\[ w_3 = b \ b \ \ldots \]
\[ \vdots \]
\[ w_n = a \ b \ b \ \ldots \ a \ c \]
Synchronous automata

\[ w_1 = a \ a \ b \ \ldots \ a \ b \ c \]
\[ w_2 = a \ b \ a \ \ldots \ a \ \bot \ \bot \]
\[ w_3 = b \ b \ \bot \ \ldots \ \bot \ \bot \ \bot \]
\[ \vdots \]
\[ w_n = a \ b \ b \ \ldots \ a \ c \ \bot \]
Synchronous automata

\[ w_1 = a \ a \ b \ \cdots \ a \ b \ c \]
\[ w_2 = a \ b \ a \ \cdots \ a \ \perp \ \perp \]
\[ w_3 = b \ b \ \perp \ \cdots \ \perp \ \perp \ \perp \]
\[ \vdots \]
\[ w_n = a \ b \ b \ \cdots \ a \ c \ \perp \]

\[ \uparrow \]
Synchronous automata

\[ w_1 = a\ a\ b\ \cdots\ a\ b\ c \]
\[ w_2 = a\ b\ a\ \cdots\ a\ \perp\ \perp \]
\[ w_3 = b\ b\ \perp\ \cdots\ \perp\ \perp\ \perp \]
\[ \vdots \]
\[ w_n = a\ b\ b\ \cdots\ a\ c\ \perp \]
\[ \uparrow \]
Synchronous automata

\[ w_1 = a \ a \ b \ \ldots \ a \ b \ c \]
\[ w_2 = a \ b \ a \ \ldots \ a \ \bot \ \bot \]
\[ w_3 = b \ b \ \bot \ \ldots \ \bot \ \bot \ \bot \ \bot \]
\[ \vdots \quad \vdots \]
\[ w_n = a \ b \ b \ \ldots \ a \ c \ \bot \]

\[ \uparrow \]
Synchronous automata

\[ w_1 = a \ a \ b \ \cdots \ a \ b \ c \]
\[ w_2 = a \ b \ a \ \cdots \ a \ \perp \ \perp \]
\[ w_3 = b \ b \ \perp \ \cdots \ \perp \ \perp \ \perp \]
\[ \vdots \]
\[ w_n = a \ b \ b \ \cdots \ a \ c \ \perp \]
\[ \uparrow \]
Synchronous automata

\[ w_1 = \textcolor{blue}{a} \textcolor{red}{a} \textcolor{blue}{b} \cdots \textcolor{blue}{a} \textcolor{red}{b} \textcolor{blue}{c} \]
\[ w_2 = \textcolor{blue}{a} \textcolor{red}{b} \textcolor{blue}{a} \cdots \textcolor{red}{a} \downarrow \downarrow \]
\[ w_3 = \textcolor{blue}{b} \textcolor{blue}{b} \downarrow \cdots \downarrow \downarrow \downarrow \downarrow \]
\[ \vdots \quad \vdots \quad \vdots \]
\[ w_n = \textcolor{blue}{a} \textcolor{red}{b} \textcolor{blue}{b} \cdots \textcolor{blue}{a} \textcolor{red}{c} \downarrow \uparrow \]
Synchronous automata

\[ w_1 = a \ a \ b \ \cdots \ a \ b \ c \]
\[ w_2 = a \ b \ a \ \cdots \ a \ \bot \ \bot \]
\[ w_3 = b \ b \ \bot \ \cdots \ \bot \ \bot \ \bot \]
\[ \vdots \]
\[ w_n = a \ b \ b \ \cdots \ a \ c \ \bot \]

\[ \uparrow \]
Examples of regular relations

- All regular languages.
- The **prefix** relation defined by:
  \[(\bigcup_{a \in \Sigma} (a, a))^* \cdot (\bigcup_{a \in \Sigma} (a, \bot))^*.\]
- The **equal length** relation defined by:
  \[(\bigcup_{a, b \in \Sigma} (a, b))^*.\]
- 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:

\[
( \bigcup_{a \in \Sigma} (a, a))^* \cdot ( \bigcup_{a \in \Sigma} (a, \bot))^*.
\]

• The equal length relation defined by:

\[
( \bigcup_{a, b \in \Sigma} (a, b))^*.
\]

• 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

\[ \text{Ans}(\bar{z}, \bar{x}) \leftarrow \bigwedge_i (x_i, \pi_i, y_i), \bigwedge_j S_j(\bar{\pi}_j), \]

where each \( S_j \) is a regular relation
ECRPQ(REG)

**ECRPQ(REG):** Class of queries of the form

\[ \text{Ans}(\vec{z}, \vec{\chi}) \leftarrow \bigwedge_i (x_i, \pi_i, y_i), \bigwedge_j S_j(\vec{\pi}_j), \]

where each \( S_j \) is a regular relation

**Example:** The ECRPQ(REG) query

\[ \text{Ans}(x, y) \leftarrow (x, \pi_1, z), (z, \pi_2, y), a^*(\pi_1), b^*(\pi_2), \text{equal\_length}(\pi_1, \pi_2) \]

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

\[
\text{Ans}(\overline{z}, \overline{\chi}) \leftarrow \bigwedge_i (x_i, \pi_i, y_i), \bigwedge_j S_j(\overline{\pi}_j),
\]

where each \( S_j \) is a regular relation.

**Example:** The ECRPQ(REG) query

\[
\text{Ans}(x, y) \leftarrow (x, \pi_1, z), (z, \pi_2, y), a^*(\pi_1), b^*(\pi_2), \text{equal\_length}(\pi_1, \pi_2)
\]

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**

- $\text{Eval(ECPRQ(REG))}$ is \text{PSPACE-complete}.
- $\text{Eval(ECPRQ(REG))}$ is in \text{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

\textit{n-ary rational relation:}
Set of $n$-tuples $(w_1, \ldots, w_n)$ of strings accepted by \textit{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 $\leq_{ss}$ defined by:

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

• The subword relation $\leq_{sw}$ defined by:

$$
\bigcup_{a \in \Sigma} (a, \epsilon)^* \cdot \bigcup_{b \in \Sigma} (b, b)^* \cdot \bigcup_{a \in \Sigma} (a, \epsilon)^*.
$$
Examples of rational relations

- All regular relations.
- The subsequence relation $\preceq_{ss}$ defined by:
  \[
  \left( \bigcup_{a \in \Sigma} (a, \varepsilon) \right)^* \bigcup_{b \in \Sigma} (b, b) \bigcup_{a \in \Sigma} (a, \varepsilon)^*.
  \]
- The subword relation $\preceq_{sw}$ defined by:
  \[
  \left( \bigcup_{a \in \Sigma} (a, \varepsilon) \right)^* \cdot \left( \bigcup_{b \in \Sigma} (b, b) \right)^* \cdot \left( \bigcup_{a \in \Sigma} (a, \varepsilon) \right)^*.
  \]

**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

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

where each \( S_j \) is a rational relation.

Example: The ECRPQ(RAT) query

\[ \text{Ans}(x, y) \leftarrow (x, \pi_1, z), (y, \pi_2, w), \pi_1 \preceq_{ss} \pi_2 \]

computes \( x, y \) that are origins of paths \( \rho_1 \) and \( \rho_2 \) such that:

\[ \lambda(\rho_1) \text{ is a subsequence of } \lambda(\rho_2). \]
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, $\preceq_{ss}$ and $\preceq_{sw}$. 
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, $\leq_{ss}$ and $\leq_{sw}$.

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

**Theorem**

*Evaluation of \((ECRPQ(REG \cup \{\leq_{sw}\}))\) 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, $\leq_{ss}$ and $\leq_{sw}$.

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

**Theorem**

*Evaluation of* $\text{ECRPQ(REG} \cup \{\leq_{sw}\})$ *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* $\text{ECRPQ(REG} \cup \{\leq_{ss}\})$ *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:

\[ \text{Ans}(\bar{z}) \leftarrow \bigwedge_{i \leq k} (x_i, \pi_i, y_i), L_i(\pi_i), \bigwedge_j S_j(\pi_{j1}, \pi_{j2}), \]

where the graph on \( \{1, \ldots, k\} \) defined by edges \((\pi_{j1}, \pi_{j2})\) is acyclic.
Acyclic ECRPQ(RAT) queries

Acyclic ECRPQ(RAT) queries yield tractable data complexity.

- Queries of the form:

\[ \text{Ans}(\vec{z}) \leftarrow \bigwedge_{i \leq k} (x_i, \pi_i, y_i), L_i(\pi_i), \bigwedge_j S_j(\pi_{j_1}, \pi_{j_2}), \]

where the graph on \{1, \ldots, k\} defined by edges \((\pi_{j_1}, \pi_{j_2})\) 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\((S)\) queries:
  - They output paths and compare labels of paths with relations in \(S\).

- 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** $\mathcal{G}$ over $\Sigma$ is a tuple $(V, E, \delta)$, where:
- $(V, E)$ is a graph database over $\Sigma$, and
- $\delta$ 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** $\mathcal{G}$ over $\Sigma$ is a tuple $(V, E, \delta)$, where:
- $(V, E)$ is a graph database over $\Sigma$, and
- $\delta$ 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 $\mathcal{G}$ of the form

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

that is obtained from $\rho$ 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 $\Sigma \times D$.

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 **DistinctValues** is NP-complete:

- **DistinctValues**: Is there a path $\rho$ from $v$ to $v'$ s.t. no data value in $\rho_D$ is repeated?
The choice of a formalism

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

**Theorem**

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

- $\text{DistinctValues}$:
  
  Is there a path $\rho$ from $v$ to $v'$ s.t. no data value in $\rho_D$ is repeated?

**Conclusion:**

- If a language expresses $\text{DistinctValues}$:
  
  - 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 (REM$s$):
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.
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:

  \[
  c := x_i \equiv | \neg c | c \land c \quad (1 \leq i \leq k)
  \]

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

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

\[
e := \varepsilon \mid a \mid e \cup e \mid e \cdot e \mid e^+ \mid e[c] \mid \downarrow \vec{x}.e
\]

where $a \in \Sigma$, $c$ condition, and $\vec{x}$ tuple in $\{x_1, \ldots, x_k\}$. 
REMs: Syntax and semantics (Intuition)

REMs over \( \Sigma \) and \( \{x_1, \ldots, x_k\} \) are defined by grammar:

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

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

**Intuition:** Evaluation of REM \( e \) on data graph \( \mathcal{G} \) is:
- pairs \((v, v')\) of nodes linked by path \( \rho \) such that \( \rho_D \models e \), where:
REMs: Syntax and semantics (Intuition)

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

$$e ::= \varepsilon \mid a \mid e \cup e \mid e \cdot e \mid e^+ \mid e[c] \mid \downarrow x.e$$

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

Intuition: Evaluation of REM $e$ on data graph $\mathcal{G}$ is:
- pairs $(\nu, \nu')$ of nodes linked by path $\rho$ such that $\rho_D \models e$, where:
  - $\rho_D \models e[c]$ if and only if
    
    $\begin{align*}
    \rho_D :& \quad \kappa(v_1) \xrightarrow{a_1} \kappa(v_2) \cdots \kappa(v_k) \xrightarrow{a_k} \kappa(v_{k+1}) \\
    & \text{starting from empty registers} \\
    & \text{can be parsed wrt } e \\
    & \text{finishing in register value } \\
    & \tau \in D^k \text{ st } (\kappa(v_{k+1}), \tau) \models c
    \end{align*}$
REMs: Syntax and semantics (Intuition)

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

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

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

Intuition: Evaluation of REM $e$ on data graph $G$ is:

- pairs $(v, v')$ of nodes linked by path $\rho$ such that $\rho^D \models e$, where:

  \[
  \rho^D \models \downarrow \bar{x}.e \text{ if and only if } \rho^D \models \downarrow \bar{x}.e
  \]

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

can be parsed wrt $e$
Consider the REM $\Sigma^* \cdot (\downarrow x.\Sigma^+ [x=]) \cdot \Sigma^*$:

- Defines pairs of nodes linked by path $\rho$ such that:
  - $\rho_D$ contains the same data value twice.
- The complement of this language is `DISTINCTVALUES`. 
REM: Example

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

- Defines pairs of nodes linked by path $\rho$ such that:
  - $\rho_D$ 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**

- \( \text{Eval(REM)} \) is \( \text{PSPACE-complete} \).
- It is in \( \text{NLOGSPACE} \) 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

  Original papers introducing (C)RPQs
- Pablo Barcelo: Querying graph databases. PODS 2013: 175-188
- 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.
  Introducing two-way queries.
Comments on papers


  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


  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.

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.