Overview

1. Introduction | Relational data model
2. First-order queries
3. Complexity of query answering
4. Complexity of FO query answering
5. Conjunctive queries
6. Tree-like conjunctive queries
7. Query optimisation
8. Conjunctive Query Optimisation / First-Order Expressiveness
9. First-Order Expressiveness / Introduction to Datalog
10. Expressive Power and Complexity of Datalog
11. Optimisation and Evaluation of Datalog
12. Evaluation of Datalog (2)
13. Graph Databases and Path Queries
14. Outlook: database theory in practice

See course homepage [⇒ link] for more information and materials
Review: Datalog

Datalog is a powerful recursive query language

Advantages:
- Natural extension of (U)CQs with recursion
- Can be extended with (EDB) negation
- Polynomial data complexity of query answering

Disadvantages:
- High query and combined complexity ($\text{ExpTime}$)
- Perfect optimisation is undecidable
- Somewhat complicated to write queries
Graph Databases

Our original motivation for going from FO queries to Datalog:
Reachability of nodes in a (directed) graph \( \leadsto \) let’s focus on graphs

**Graph database:** a DBMS that supports “graphs” as its datamodel

There are many kinds of graphs:

- Directed or undirected?
- Labelled or unlabelled edges/nodes?
- What kinds of labels? Datatypes?
- Parallel edges (multi-graphs)? With same label?
- One graph or several graphs per database?

Two types of graph database models dominate the market today:
*Resource Description Framework (RDF)* and *Property Graph*
RDF is a W3C standard for representing linked data on the Web

- Directed labelled graph; nodes are identified by their labels
- Labels are URIs or datatype literals
- Multiple parallel edges only when using different edge labels
- Supports multiple graphs in one database
- W3C standard; implementations for many programming languages
- Datatype support based on W3C XML Schema datatypes
- Graphs can be exchanged in many standard syntax formats
Property Graph

Property Graph is a popular data model of many graph databases

• Directed labelled multi-graph; labels do not identify nodes
• “Labels” can be lists of attribute-value pairs
• Multiple parallel edges with the exact same labels are possible
• No native multi-graph support (could be simulated with additional attributes)
• No standard definition of technical details; most common implementation: Tinkerpop/Blueprints API (Java)
• Datatype support varies by implementation
• No standard syntax for exchanging data
Representing Graphs

Graphs (of any type) are usually viewed as sets of edges

- RDF: triples of form subject-predicate-object
  - When managing multiple graphs, each triple is extended with a fourth component (graph ID) \(\sim\) quads
  - RDF databases are sometimes still called "triple stores", although most modern systems effectively store quads

- Property Graph: edge objects with attribute lists
  - represented by Java objects in Blueprints

Graphs can be stored in relational databases

- RDF: table Triple[Subject, Predicate, Object]
- Property Graph: tables Edge[Sourceld, Edgeld, Targetld] and Attributes[Id, Attribute, Value]
Representing Data in Graphs

Property Graphs can represent RDF:

- use attributes to store RDF node and edge labels (URIs)
- use key constraints to ensure that no two distinct nodes can have same label
Representing Data in Graphs

Property Graphs can represent RDF:

- use attributes to store RDF node and edge labels (URIs)
- use key constraints to ensure that no two distinct nodes can have same label

RDF can represent Property Graphs:

- use additional nodes to represent Property Graph edges
- use RDF triples with special predicates to represent attributes

Either model can also represent hypergraphs/RDBs (exercise)

\(\Rightarrow\) all models can represent all data in principle
\(\Rightarrow\) supported query features and performance will vary
Preferred query language depends on graph model

- RDF: W3C SPARQL query language
- Property Graph: no uniform approach to data access
  - many tools prefer API access over a query language
  - proprietary query languages, e.g., “Cypher” for Neo4j

However, there are some common basics in almost all cases:

- Conjunctive queries
- Regular path queries
Conjunctive Queries over Graphs

Basic descriptions of local patterns in a graph

Formally, it suffices to say:
CQs over RDF correspond to CQs over relational databases with a single table Triple[Subject,Predicate,Object]
(analogously for Property Graphs)

- All complexity results for query answering and optimisation carry over from RDBs (in particular, restricting to graphs does not make anything simpler)
- Details of representation of data in tables do not matter
- CQs are restricted to local patterns (no reachability . . . )
Regular Path Queries

Idea: use regular expressions to navigate over paths

Let’s consider a simplified graph model, where a graph is given by:

- Set of nodes $N$ (without additional labels)
- Set of edges $E$, labelled by a function $\lambda : E \rightarrow L$, where $L$ is a finite set of labels

**Definition**

A *regular expression* over a set of labels $L$ is an expression of the following form:

$$E ::= L \mid (E \circ E) \mid (E + E) \mid E^*$$

A *regular path query* (RPQ) is an expression of the form $E(s, t)$, where $E$ is a regular expression and $s$ and $t$ are terms (constants or variables).
Semantics of Regular Path Queries

As usual, a regular expression $E$ matches a word $w = \ell_1 \cdots \ell_n$ if any of the following conditions is satisfied:

- $E \in L$ is a label and $w = E$.
- $E = (E_1 \circ E_2)$ and there is $i \in \{0, \ldots, n\}$ such that $E_1$ matches $\ell_1 \cdots \ell_i$ and $E_2$ matches $\ell_{i+1} \cdots \ell_n$ (the words matched by $E_1$ and $E_2$ can be empty if $i = 0$ or $i = n$, respectively).
- $E = (E_1 + E_2)$ and $w$ is matched by $E_1$ or by $E_2$
- $E = E_1^*$ and $w$ has the form $w_1 w_2 \cdots w_m$ for $n \geq 0$, where each word $w_i$ is matched by $E_1$

**Definition**

Let $a$ and $b$ be constants and $x$ and $y$ be variables. An RPQ $E(a, b)$ is entailed by a graph $G$ if there is a directed path from node $a$ to node $b$ that is labelled by a word matched by $E$. The answers to RPQs $E(x, y)$, $E(x, b)$, and $E(a, y)$ are defined in the obvious way.
Extending the Expressive Power of RPQs

Regular path queries can be used to express typical reachability queries, but are still quite limited. → extensions

2-Way Regular Path Queries (2RPQs)

- For every label \( \ell \in L \), also introduce a converse label \( \ell^- \)
- Allow converse labels in regular expressions
- Matched paths can follow edges forwards or backwards

Conjunctive Regular Path Queries (CRPQs)

- Extend conjunctive queries with RPQs
- RPQs can be used like binary query atoms
- Obvious semantics

Conjunctive 2-Way Regular Path Queries (C2RPQs) combine both extensions
C2RPQs: Examples

All ancestors of Alice:

\[((\text{father} + \text{mother}) \circ (\text{father} + \text{mother})^*)(\text{alice}, y)\]
C2RPQs: Examples

All ancestors of Alice:

\[((\text{father} + \text{mother}) \circ (\text{father} + \text{mother})^*) (\text{alice, } y)\]

People with finite Erdös number:

\[(\text{authorOf} \circ \text{authorOf}^{-1})^*(x, \text{paulErdös})\]
C2RPQs: Examples

All ancestors of Alice:

\[ (((\text{father} + \text{mother}) \circ (\text{father} + \text{mother})^*) (\text{alice}, y)) \]

People with finite Erdős number:

\[ ((\text{authorOf} \circ \text{authorOf}^{-1})^* (x, \text{paulErdös})) \]

Pairs of stops connected by tram lines 3 and 8:

\[ ((\text{nextStop3} \circ \text{nextStop3}^*) (x, y) \land (\text{nextStop8} \circ \text{nextStop8}^*) (x, y)) \]
Complexity of RPQs

A nondeterministic algorithm for Boolean RPQs:
- Transform regular expression into a finite automaton
- Starting from the first node, guess a matching path
- When moving along path, advance state of automaton
- Accept if the second node is reached in an accepting state
- Reject if path is longer than size of graph \( \times \) size of automaton

Space requirements when assuming query (and automaton) fixed:
- Pointer to current node in graph, pointer to current state of automaton, counter for length of path

Coversely, reachability in unlabelled graph is hard for \( \text{NL} \):
- RPQ matching is \( \text{NL} \)-complete (data complexity)

(Combined/query complexity is in \( \text{P} \), as we will see below)
Complexity of RPQs

A nondeterministic algorithm for Boolean RPQs:

- Transform regular expression into a finite automaton
- Starting from the first node, guess a matching path
- When moving along path, advance state of automaton
- Accept if the second node is reached in an accepting state
- Reject if path is longer than size of graph $\times$ size of automaton

Space requirements when assuming query (and automaton) fixed: pointer to current node in graph, pointer to current state of automaton, counter for length of path $\sim$ NL algorithm

Conversely, reachability in an unlabelled graph is hard for NL $\sim$ RPQ matching is NL-complete (data complexity)

(Combined/query complexity is in P, as we will see below)
Complexity of C2RPQs

We already know:

- CQ matching is in $\text{AC}^0$ (data complexity) and $\text{NP}$-complete (query and combined complexity)
- RPQ matching is $\text{NL}$-complete (data) and in $\text{P}$ (query/combined)
- $\text{AC}^0 \subset \text{NL}$ and $\text{NL} \subset \text{NP}$

$\Rightarrow$ C2RPQs are $\text{NP}$-hard (combined/query) and $\text{NL}$-hard (data)
Complexity of C2RPQs

We already know:

- CQ matching is in $AC^0$ (data complexity) and $NP$-complete (query and combined complexity)
- RPQ matching is $NL$-complete (data) and in $P$ (query/combined)
- $AC^0 \subset NL$ and $NL \subseteq NP$

$\therefore$ C2RPQs are $NP$-hard (combined/query) and $NL$-hard (data)

It’s not hard to show that these bounds are tight:

**Theorem**

C2RPQ matching is $NP$-complete for combined and query complexity, and $NL$-complete for data complexity.
(C2)RPQs and Datalog

How do path queries relate to Datalog?

We already know:

- Datalog is \textsc{ExpTime}\text{-complete} (combined/query) and \textsc{P}\text{-complete} (data)
- C2RPQs are \textsc{NP}\text{-complete} (combined/query) and \textsc{NL}\text{-complete} (data)

$\implies$ maybe Datalog is more expressive than C2RPQs . . .
How do path queries relate to Datalog?

We already know:

- Datalog is $\mathsf{ExpTime}$-complete (combined/query) and $\mathsf{P}$-complete (data)
- C2RPQs are $\mathsf{NP}$-complete (combined/query) and $\mathsf{NL}$-complete (data)

$\Rightarrow$ maybe Datalog is more expressive than C2RPQs . . .

Indeed, we can express regular expressions in Datalog

For simplicity, assume that we have a binary EDB predicate $p_\ell$ for each label $\ell \in L$ (other encodings would work just as well)
We transform a regular expression $E$ to a Datalog query $\langle Q_E, P_E \rangle$:
2-Way Regular Expressions in Datalog

We transform a regular expression $E$ to a Datalog query $\langle Q_E, P_E \rangle$:

If $E = \ell \in L$ is a label, then $P_E = \{ Q_E(x, y) \leftarrow p_{\ell}(x, y) \}$
2-Way Regular Expressions in Datalog

We transform a regular expression $E$ to a Datalog query $\langle Q_E, P_E \rangle$:

If $E = \ell \in L$ is a label, then $P_E = \{ Q_E(x, y) \leftarrow p_\ell(x, y) \}$

If $E = \ell^-$ is the converse of a label $\ell \in L$, then

$$P_E = \{ Q_E(x, y) \leftarrow p_\ell(y, x) \}$$
2-Way Regular Expressions in Datalog

We transform a regular expression $E$ to a Datalog query $\langle Q_E, P_E \rangle$:

If $E = \ell \in L$ is a label, then $P_E = \{ Q_E(x, y) \leftarrow p_\ell(x, y) \}$

If $E = \ell^{-}$ is the converse of a label $\ell \in L$, then

$$P_E = \{ Q_E(x, y) \leftarrow p_\ell(y, x) \}$$

If $E = (E_1 \circ E_2)$ then

$$P_E = P_{E_1} \cup P_{E_2} \cup \{ Q_E(x, z) \leftarrow Q_{E_1}(x, y) \land Q_{E_2}(y, z) \}$$
2-Way Regular Expressions in Datalog

We transform a regular expression \( E \) to a Datalog query \( \langle Q_E, P_E \rangle \):

If \( E = \ell \in L \) is a label, then \( P_E = \{ Q_E(x, y) \leftarrow p_\ell(x, y) \} \)

If \( E = \ell^- \) is the converse of a label \( \ell \in L \), then

\[
P_E = \{ Q_E(x, y) \leftarrow p_\ell(y, x) \}
\]

If \( E = (E_1 \circ E_2) \) then

\[
P_E = P_{E_1} \cup P_{E_2} \cup \{ Q_E(x, z) \leftarrow Q_{E_1}(x, y) \land Q_{E_2}(y, z) \}
\]

If \( E = (E_1 + E_2) \) then

\[
P_E = P_{E_1} \cup P_{E_2} \cup \{ Q_E(x, y) \leftarrow Q_{E_1}(x, y), Q_E(x, y) \leftarrow Q_{E_2}(x, y) \}
\]
2-Way Regular Expressions in Datalog

We transform a regular expression $E$ to a Datalog query $\langle Q_E, P_E \rangle$:

If $E = \ell \in L$ is a label, then $P_E = \{ Q_E(x, y) \leftarrow p_\ell(x, y) \}$

If $E = \ell^-$ is the converse of a label $\ell \in L$, then

$$P_E = \{ Q_E(x, y) \leftarrow p_\ell(y, x) \}$$

If $E = (E_1 \circ E_2)$ then

$$P_E = P_{E_1} \cup P_{E_2} \cup \{ Q_E(x, z) \leftarrow Q_{E_1}(x, y) \land Q_{E_2}(y, z) \}$$

If $E = (E_1 + E_2)$ then

$$P_E = P_{E_1} \cup P_{E_2} \cup \{ Q_E(x, y) \leftarrow Q_{E_1}(x, y), Q_E(x, y) \leftarrow Q_{E_2}(x, y) \}$$

If $E = E_1^*$ then

$$P_E = P_{E_1} \cup \{ Q_E(x, x) \leftarrow, Q_E(x, z) \leftarrow Q_E(x, y) \land Q_{E_1}(y, z) \}$$
Reprise: Combined Complexity of 2RPQs

As a side effect, the previous translation shows that 2RPQs can be evaluated in $P$ combined complexity:

- Each (2-way) regular expression $E$ leads to a Datalog query $⟨Q_E, P_E⟩$ of polynomial size
- Each rule in $P_E$ has at most three variables
  $\leadsto$ the grounding of $P_E$ for a graph with nodes $N$ is of size $|P_E| \times |N|^3$
- propositional logic rules can be evaluated in polynomial time
  $\leadsto$ polynomial time decision procedure
Expressing C2RPQs in Datalog

It is now easy to express C2RPQs in Datalog:

- Use the encoding of CQs in Datalog as shown in the exercise
- Express 2RPQ atoms in Datalog as just shown

Can every Datalog query over binary “labelled-edge” EDB predicates be expressed with (C2)RPQs?

This would imply $P = NL$ (but not that $NP = \text{ExpTime}$!): unlikely but not known to be false

However, there are stronger direct arguments that show the limits of C2RPQs (exercise)
It is now easy to express C2RPQs in Datalog:

- Use the encoding of CQs in Datalog as shown in the exercise
- Express 2RPQ atoms in Datalog as just shown

Can every Datalog query over binary “labelled-edge” EDB predicates be expressed with (C2)RPQs?

- This would imply \( P = NL \) (but not that \( NP = \text{ExpTime} \! \)): unlikely but not known to be false
- However, there are stronger direct arguments that show the limits of C2RPQs (exercise)
Linear Datalog and Binary Datalog

Expressing 2RPQs in Datalog requires only restricted forms of Datalog:

**Definition**

A Datalog program is **linear** if each of its rules has at most one IDB atom in its body. A Datalog program is **binary** if all of its IDB predicates have arity at most two.

The following complexity results are known:

**Theorem**

Query answering in linear Datalog is \( \text{NL} \)-complete for data complexity, and \( \text{PSPACE} \)-complete for combined and query complexity.

Combined complexity further drops to \( \text{NP} \) for binary Datalog.

\[\sim\] complexity results that are more similar to \((\text{C2})\text{RPQs}\) …
2RPQs and Linear Datalog

The Datalog translation of 2RPQs does not lead to linear Datalog, but we can fix this.

We transform a regular expression $E$ to a linear Datalog query $\langle Q_E, P_E^{\text{lin}} \rangle$:

- Construct a non-deterministic automaton $A_E$ for $E$.
- For every state $q$ of $A_E$, we use a binary IDB predicate $S_q$.
- For the starting state $q_0$ of $A_E$, we add a rule $S_{q_0}(x, x) \leftarrow$.
- For every transition $q \xrightarrow{\ell} q'$ of $A_E$, we add a rule
  \[ S_{q'}(x, z) \leftarrow S_q(x, y) \land p_\ell(y, z) \]
- For every final state $q_f$ of $A_E$, we add a rule
  \[ Q_E(x, y) \leftarrow S_{q_f}(x, y) \]

Two-way queries can be captured by allowing two-way transitions.
Linear Datalog vs. 2RPQs

So all 2RPQs can be expressed in linear Datalog
Is the converse also true?

---

Counterexample:

Query \((x, z) \leftarrow p_a(x, y) \land p_b(y, z)\)

Query \((x, z) \leftarrow p_a(x, x') \land \text{Query}(x', z') \land p_b(z', z)\)

The linear Datalog program matches paths with labels from a context-free, non-regular language.

Intuition: linear Datalog generalises context-free languages.
Linear Datalog vs. 2RPQs

So all 2RPQs can be expressed in linear Datalog
Is the converse also true?

**No.** Counterexample:

\[
\text{Query}(x, z) \leftarrow p_a(x, y) \land p_b(y, z)
\]

\[
\text{Query}(x, z) \leftarrow p_a(x, x') \land \text{Query}(x', z') \land p_b(z', z)
\]

The linear Datalog program matches paths with labels from \(a^n b^n\)
\(\leadsto\) context-free, non-regular language
\(\leadsto\) not expressible in (C2)RPQs

Intuition: linear Datalog generalises context-free languages
Recall the basic static optimisation problems of database theory:

- Query containment
- Query equivalence
- Query emptiness

Which of these are decidable for (C2)RPQs?
Query Optimisation for C2RPQs

Recall the basic static optimisation problems of database theory:

- Query containment
- Query equivalence
- Query emptiness

Which of these are decidable for (C2)RPQs?

Observation: query emptiness is trivial
Containment for RPQs

Containment of Regular Path Queries corresponds to containment of regular expressions \( \sim \) known to be decidable in \( \text{PSPACE} \)

Proof sketch for checking \( E_1 \sqsubseteq E_2 \):

1. Construct non-deterministic automata (NFAs), \( A_1 \) and \( A_2 \) for the regular expressions \( E_1 \) and \( E_2 \), respectively
2. Construct an automaton \( \bar{A}_2 \) that accepts the complement of \( A_2 \).
3. Construct the intersection \( A_1 \cap \bar{A}_2 \) of \( A_1 \) and \( \bar{A}_2 \)
4. Check if \( A_1 \cap \bar{A}_2 \) accepts a word (if yes, then there is a counterexample that disproves \( E_1 \sqsubseteq E_2 \); if no, then the containment holds)

Complexity estimate:
\( A_1 \cap \bar{A}_2 \) is exponential (blow-up by powerset construction in step (2)) but step (4) is possible by checking reachability on the state graph
\( \sim \) \( \text{NL} \) algorithm on an exponential state graph
\( \sim \) \( \text{NPSPACE} \) algorithm (construct the state graph on the fly)
\( \sim \) \( \text{PSPACE} \) algorithm (Savitch’s Theorem)
Containment for (C)2RPQs

Things are more tricky when adding converses and conjunctions

Theorem

- Containment of 2RPQs is $\text{PSPACE}$-complete
- Containment of C2RPQs is $\text{EXPSPACE}$-complete

The proofs are more involved.

Automata-theoretic constructions are used, but with more complicated automata models and for somewhat different languages (there is no good “language of possible C2RPQ matches on a graph” $\sim$ consider language of possible proofs instead)
Query Optimisation for Path Queries

Decidable in $PSPACE$ (2RPQs) and $EXPSPACE$ (C2RPQs)

Should be compared to linear Datalog:

**Theorem**
Query containment for linear Datalog queries is undecidable.

Proof: see Lecture 11 (Post Correspondence Problem in Datalog – in fact, in linear Datalog)

Essentially no adoption in practice

$\sim$ maybe the complexities are too high \ldots

$\sim$ or maybe path query optimisers are just too primitive
We have seen that C2RPQs are $\mathbf{NL}$-complete for data.

$\sim$ can all $\mathbf{NL}$-complete queries be captured by a C2RPQ?
We have seen that C2RPQs are NL-complete for data
\( \Rightarrow \) can all NL-complete queries be captured by a C2RPQ?

No. For many reasons.

- C2RPQs have no disjunction (\( \Rightarrow \) Unions of C2RPQs)
- C2RPQs have no negation

FO-queries with a binary transitive closure operator capture NL

Several (regular) extensions of path queries:

- Nested unary 2RPQs in regular expressions (“test operators”)
- Nested binary C2RPQs in regular expressions
- Other more expressive fragments of “regular Datalog”, e.g., Monadically Defined Queries
Summary and Outlook

Graph databases as an important class of “noSQL” databases

Two main data models
  • Resource Description Framework (RDF)
  • Property Graph

Path queries as common foundation of all graph query languages
  • higher data complexities than CQs/FO queries
  • lower complexities than Datalog queries
  • decidable query optimisation

Next topics:
  • Applications
  • Summary