DATABASE THEORY

Lecture 5: Conjunctive Queries

Markus Krötzsch

TU Dresden, 28 April 2016
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
The evaluation of FO queries is

- $\text{PSPACE}$-complete for combined complexity
- $\text{PSPACE}$-complete for query complexity
- $\text{AC}^0$-complete for data complexity

$\sim \text{PSPACE}$ is rather high

$\sim$ Are there relevant query languages that are simpler than that?
Conjunctive Queries

Idea: restrict FO queries to conjunctive, positive features

**Definition**

A conjunctive query (CQ) is an expression of the form

\[ \exists y_1, \ldots, y_m. A_1 \land \ldots \land A_\ell \]

where each \( A_i \) is an atom of the form \( R(t_1, \ldots, t_k) \). In other words, a conjunctive query is an FO query that only uses conjunctions of atoms and (outer) existential quantifiers.

Example: “Find all lines that depart from an accessible stop” (as seen in earlier lectures)

\[ \exists y_{\text{SID}}, y_{\text{Stop}}, y_{\text{To}}. \text{Stops}(y_{\text{SID}}, y_{\text{Stop}}, \text{"true"}) \land \text{Connect}(y_{\text{SID}}, y_{\text{To}}, x_{\text{Line}}) \]
The expressive power of CQs can also be captured in the relational calculus

**Definition**

A conjunctive query (CQ) is a relational algebra expression that uses only the operations select \( \sigma_{n=m} \), project \( \pi_{a_1,...,a_n} \), join \( \bowtie \), and renaming \( \delta_{a_1,...,a_n \rightarrow b_1,...,b_n} \).

Renaming is only relevant in named perspective

\( \rightarrow \) CQs are also known as SELECT-PROJECT-JOIN queries
Extensions of Conjunctive Queries

Two features are often added:

- **Equality**: CQs with equality can use atoms of the form $t_1 \approx t_2$ (in relational calculus: table constants)
- **Unions**: unions of conjunctive queries are called UCQs (in this case the union is only allowed as outermost operator)

Both extensions truly increase expressive power (as shown in exercise)

**Features omitted on purpose**: negation and universal quantifiers
$\Rightarrow$ the reason for this is query complexity (as we shall see)
A Boolean conjunctive query (BCQ) asks for a mapping from query variables to domain elements such that all atoms are true.

Example: “Is there an accessible stop where some line departs?”

\[ \exists y_{\text{SID}}, y_{\text{Stop}}, y_{\text{To}}, y_{\text{Line}}. \text{Stops}(y_{\text{SID}}, y_{\text{Stop}}, \text{"true"}) \land \text{Connect}(y_{\text{SID}}, y_{\text{To}}, y_{\text{Line}}) \]

<table>
<thead>
<tr>
<th>SID</th>
<th>Stop</th>
<th>Accessible</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>Hauptbahnhof</td>
<td>true</td>
</tr>
<tr>
<td>42</td>
<td>Helmholtzstr.</td>
<td>true</td>
</tr>
<tr>
<td>57</td>
<td>Stadtgutstr.</td>
<td>true</td>
</tr>
<tr>
<td>123</td>
<td>Gustav-Freytag-Str.</td>
<td>false</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>57</td>
<td>42</td>
<td>85</td>
</tr>
<tr>
<td>17</td>
<td>789</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
A **Boolean conjunctive query (BCQ)** asks for a mapping from query variables to domain elements such that all atoms are true.

**Example:** “Is there an accessible stop where some line departs?”

$$\exists y_{SID}, y_{Stop}, y_{To}, y_{Line}. \text{Stops}(y_{SID}, y_{Stop}, \text{"true"}) \land \text{Connect}(y_{SID}, y_{To}, y_{Line})$$

<table>
<thead>
<tr>
<th>SID</th>
<th>Stop</th>
<th>Accessible</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>Hauptbahnhof</td>
<td>true</td>
</tr>
<tr>
<td>42</td>
<td>Helmholtzstr.</td>
<td>true</td>
</tr>
<tr>
<td>57</td>
<td>Stadtgutstr.</td>
<td>true</td>
</tr>
<tr>
<td>123</td>
<td>Gustav-Freytag-Str.</td>
<td>false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>57</td>
<td>42</td>
<td>85</td>
</tr>
<tr>
<td>17</td>
<td>789</td>
<td>3</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>
A Boolean conjunctive query (BCQ) asks for a mapping from query variables to domain elements such that all atoms are true.

**Example:** “Is there an accessible stop where some line departs?”

$$\exists y_{SID}, y_{Stop}, y_{To}, y_{Line}. \text{Stops}(y_{SID}, y_{Stop}, \text{"true"}) \land \text{Connect}(y_{SID}, y_{To}, y_{Line})$$

<table>
<thead>
<tr>
<th>SID</th>
<th>Stop</th>
<th>Accessible</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>Hauptbahnhof</td>
<td>true</td>
</tr>
<tr>
<td>42</td>
<td>Helmholtzstr.</td>
<td>true</td>
</tr>
<tr>
<td>57</td>
<td>Stadtgutstr.</td>
<td>true</td>
</tr>
<tr>
<td>123</td>
<td>Gustav-Freytag-Str.</td>
<td>false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>57</td>
<td>42</td>
<td>85</td>
</tr>
<tr>
<td>17</td>
<td>789</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
How Hard is it to Answer CQs?

If we know the variable mappings, it is easy to check:

- Checking if a single ground atom $R(c_1, \ldots, c_k)$ holds can be done in linear time
- Checking if a conjunction of ground atoms holds can be done in quadratic time
How Hard is it to Answer CQs?

If we know the variable mappings, it is easy to check:

- Checking if a single ground atom $R(c_1, \ldots, c_k)$ holds can be done in linear time
- Checking if a conjunction of ground atoms holds can be done in quadratic time

$\sim$ A candidate BCQ match can be verified in $P$

(The there are $n^m$ candidates: $n$ size of domain; $m$ number of query variables)

**Theorem**

BCQ query answering is in $NP$ for combined complexity (and also for query complexity).

$\sim$ Better than $PSPACE$ (presumably)
Can we do any better?

Not really. To see this, let’s look at some other problems.

Consider two relational structures $\mathcal{I}$ and $\mathcal{J}$
(= database instances, interpretations, hypergraphs)

**Definition**

A homomorphism $h$ from $\mathcal{I}$ to $\mathcal{J}$ is a function $h : \Delta^\mathcal{I} \rightarrow \Delta^\mathcal{J}$ such that, for all relation names $R$:

$$
\text{if } \langle d_1, \ldots, d_n \rangle \in R^\mathcal{I} \text{ then } \langle h(d_1), \ldots, h(d_n) \rangle \in R^\mathcal{J}.
$$

The homomorphism problem is the question if there is a homomorphism from $\mathcal{I}$ to $\mathcal{J}$.
Example: Three-colouring as Homomorphism

$I:$

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

$J:$

The 3-colouring problem is $\text{NP}$-hard.
Example: Three-colouring as Homomorphism

$I : 

\begin{array}{c|c}
1 & 2 \\
1 & 5 \\
1 & 6 \\
2 & 3 \\
2 & 7 \\
3 & 4 \\
3 & 8 \\
\ldots & \ldots \\
\end{array}

$J : 

\begin{array}{c|c|c}
\hline
& r & g \\
\hline
r & b \\
g & r \\
g & b \\
\hline
b & r \\
b & g \\
\hline
\end{array}

3-colouring is $\text{NP}$-hard.
Example: Three-colouring as Homomorphism

\( I : \) 

\begin{align*}
1 & \quad 2 \\
1 & \quad 5 \\
1 & \quad 6 \\
2 & \quad 3 \\
2 & \quad 7 \\
3 & \quad 4 \\
3 & \quad 8 \\
\ldots & \quad \ldots \\
\end{align*}

\( J : \)

\begin{align*}
1 & \quad 2 \\
1 & \quad 5 \\
1 & \quad 6 \\
2 & \quad 3 \\
2 & \quad 7 \\
3 & \quad 4 \\
3 & \quad 8 \\
\ldots & \quad \ldots \\
\end{align*}

3-colouring is \( \text{NP}-\text{hard} \)
Example: Three-colouring as Homomorphism

$I$:

$\begin{array}{c|c}
1 & 2 \\
1 & 5 \\
1 & 6 \\
2 & 3 \\
2 & 7 \\
3 & 4 \\
3 & 8 \\
\cdots & \cdots \\
\end{array}$

$J$:

$\begin{array}{c|c}
r & g \\
r & b \\
g & r \\
g & b \\
b & r \\
b & g \\
\end{array}$

3-colouring is NP-hard
$\Rightarrow$ the homomorphism problem is NP-hard
The homomorphism problem can be reduced to BCQ answering:

- A relational structure $\mathcal{I}$ gives rise to a CQ $Q_{\mathcal{I}}$: replace domain elements by variables (one-to-one); add one query atom per relational tuple; existentially quantify all variables
- $\mathcal{I}$ has a homomorphism to $\mathcal{J}$ if and only if $\mathcal{J} \models Q_{\mathcal{I}}$
BCQ Answering as Homomorphism Problem

The homomorphism problem can be reduced to BCQ answering:

- A relational structure $\mathcal{I}$ gives rise to a CQ $Q_\mathcal{I}$:
  replace domain elements by variables (one-to-one); add one query atom per relational tuple; existentially quantify all variables
- $\mathcal{I}$ has a homomorphism to $\mathcal{J}$ if and only if $\mathcal{J} \models Q_\mathcal{I}$

BCQ answering can be reduced to the homomorphism problem:

- Clear for BCQs that don’t contain constants
- Eliminate query constants $a$: create new relation $R_a = \{\langle a \rangle\}$;
  replace $a$ by a fresh variable $x$ and add a query atom $R_a(x)$

$\iff$ both problems are equivalent
We showed that BCQ answering is in $\text{NP}$ and that the homomorphism problem is $\text{NP}$-hard, therefore:

**Theorem**

BCQ answering is
- $\text{NP}$-complete for combined complexity
- $\text{NP}$-complete for query complexity
- in $\text{AC}^0$ for data complexity (inherited from FO queries)
Another important problem equivalent to BCQ answering

**Definition**

A constraint satisfaction problem (CSP) over a domain \( \Delta \) is given by a set of variables \( \{x_1, \ldots, x_n\} \) and a set of constraints \( \{C_1, \ldots, C_m\} \), where each constraint \( C_i \) has the form \( \langle X_i, R_i \rangle \) with

- \( X_i \) a list of variables from \( \{x_1, \ldots, x_n\} \),
- \( R_i \) a \(|X_i|\)-ary relation over \( \Delta \).

A solution to the CSP is an assignment of variables to values from \( \Delta \) such that all constraints are satisfied (=all tuples occur in the respective relations).

\( \sim \) alternative notation for BCQ answering/homomorphism problem
CSP Example

A combinatorial crossword puzzle:

Domain: $\Delta = \{A, \ldots, Z\}$

Variables: $x_1, \ldots, x_{26}$

Constraints:

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>$x_2$</td>
<td>$x_3$</td>
<td>$x_4$</td>
<td>$x_5$</td>
<td>$x_6$</td>
</tr>
<tr>
<td>$x_7$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_{11}$</td>
<td>$x_{12}$</td>
<td>$x_{13}$</td>
<td>$x_{14}$</td>
<td>$x_{15}$</td>
<td></td>
</tr>
<tr>
<td>$x_{16}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_{20}$</td>
<td>$x_{21}$</td>
<td>$x_{22}$</td>
<td>$x_{23}$</td>
<td>$x_{24}$</td>
<td>$x_{25}$</td>
</tr>
</tbody>
</table>

1 vertically:
- HEART
- HONEY
- IRONY
- LOGIC

1 horizontally:
- HAPPY
- INFER
- LABOR
- LATER

5 vertically:
- RADIO
- RETRO
- YACHT
- YERBA

...
Equivalent Problems

Summing up, the following problems are equivalent:

- Answering a conjunctive query over a database instance
- Finding a homomorphism from a relational structure to another
- Solving a constraint satisfaction problem

Each of these problems is NP-complete
Towards Better Complexities

NP-complete problems are still intractable
\( \Rightarrow \) can we do better?

Problem: searching a match may require backtracking, eventually exploring all options
Towards Better Complexities

**NP**-complete problems are still intractable
\[\sim \text{can we do better?}\]

Problem: searching a match may require backtracking, eventually exploring all options

![Diagram](https://via.placeholder.com/150)
Towards Better Complexities

NP-complete problems are still intractable
\[\sim\] can we do better?

Problem: searching a match may require backtracking, eventually exploring all options

Intuition: life would be easier if we would not have to go back so much . . .

{the problem is with the cycles

Markus Krötzsch, 28 April 2016
Towards Better Complexities

NP-complete problems are still intractable
\( \sim \) can we do better?

Problem: searching a match may require backtracking, eventually exploring all options

```
HAPPY
O   A
N C
E H
Y T
```
Towards Better Complexities

NP-complete problems are still intractable
\( \sim \) can we do better?

Problem: searching a match may require backtracking, eventually exploring all options

```
H A P P Y
O A
N E W C
E H
Y T
```
Towards Better Complexities

**NP-complete problems are still intractable**

~~ can we do better?

**Problem:** searching a match may require backtracking, eventually exploring all options
Towards Better Complexities

NP-complete problems are still intractable
\implies can we do better?

Problem: searching a match may require backtracking, eventually exploring all options
Towards Better Complexities

NP-complete problems are still intractable
~ can we do better?

Problem: searching a match may require backtracking, eventually exploring all options

Intuition: life would be easier if we would not have to go back so much ...
~ the problem is with the cycles
Example: Cyclic CQs

“Is there a child whose parents are married with each other?”

$$\exists y_c, y_m, y_f. \text{mother}(y_c, y_m) \land \text{father}(y_c, y_f) \land \text{married}(y_m, y_f)$$

$\sim$ cyclic query
Example: Acyclic CQs

“Is there a child whose parents are married with someone?”

\[ \exists y_c, y_m, y_f, y_{mm}, y_{mf} . \text{mother}(y_c, y_m) \land \text{father}(y_c, y_f) \land \text{married}(y_m, y_{mm}) \land \text{married}(y_{mf}, y_f) \]

\[ \leadsto \text{acyclic query} \]
Defining Acyclic Queries

Queries in general are hypergraphs

What does “acyclic” mean?
Defining Acyclic Queries

Queries in general are hypergraphs

What does “acyclic” mean?

View hypergraphs as graphs to check acyclicity?

- **Primal graph**: same vertices; edges between each pair of vertices that occur together in a hyperedge
- **Incidence graph**: vertices and hyperedges as vertices, with edges to mark incidence (bipartite graph)
Defining Acyclic Queries

Queries in general are hypergraphs

What does “acyclic” mean?

View hypergraphs as graphs to check acyclicity?

- **Primal graph**: same vertices; edges between each pair of vertices that occur together in a hyperedge
- **Incidence graph**: vertices and hyperedges as vertices, with edges to mark incidence (bipartite graph)

However: both graphs have cycles in almost all cases
Acyclic Hypergraphs

GYO-reduction algorithm to check acyclicity:
(after Graham [1979] and Yu & Özsoyoğlu [1979])

Input: hypergraph $H = \langle V, E \rangle$ (we don’t need relation labels here)
Output: GYO-reduct of $H$

Apply the following simplification rules as long as possible:
(1) Delete all vertices that occur in at most one hyperedge
(2) Delete all hyperedges that are empty or that are contained in other hyperedges

**Definition**

A hypergraph is **acyclic** if its GYO-reduct is $\langle \emptyset, \emptyset \rangle$.
A CQ is **acyclic** if its associated hypergraph is.
Example 1: GYO-Reduction

Rule (1)

Rule (2)
Example 2: GYO-Reduction
Alternative Version of GYO-Reduction

An ear of a hypergraph \( \langle V, E \rangle \) is a hyperedge \( e \in E \) that satisfies one of the following:

(1) there is an edge \( e' \in E \) such that \( e \neq e' \) and every vertex of \( e \) is either only in \( e \) or also in \( e' \), or

(2) \( e \) has no intersection with any other hyperedge.

Example:

\[
\begin{array}{cccccc}
4 & 5 & 6 & 7 & 8 & 9 \\
\end{array}
\]

\[
\begin{array}{cccccc}
3 & 2 & 4 & 5 \\
0 & 1 & 6 & \\
9 & 7 & \\
\end{array}
\]

\( \sim \) edges \( \langle 4, 5, 6 \rangle \) and \( \langle 7, 8, 9 \rangle \) are ears
Examples

Any ears?

- Any ears?
  - y1
  - y2
  - y3
  - y4
  - y5
  - y6
GYO’-Reduction

Input: hypergraph $H = \langle V, E \rangle$
Output: GYO’-reduct of $H$

Apply the following simplification rule as long as possible:
- Select an ear $e$ of $H$
- Delete $e$
- Delete all vertices that only occurred in $e$

Theorem
The GYO-reduct is $\langle \emptyset, \emptyset \rangle$ if and only if the GYO’-reduct is $\langle \emptyset, \emptyset \rangle$

$\leadsto$ alternative characterization of acyclic hypergraphs
Join Trees

Both GYO algorithms can be implemented in linear time

Open question: what benefit does BCQ acyclicity give us?
Join Trees

Both GYO algorithms can be implemented in linear time

Open question: what benefit does BCQ acyclicity give us?

Fact: if a BCQ is acyclic, then it has a join tree

**Definition**

A join tree of a (B)CQ is an arrangement of its query atoms in a tree structure $T$, such that for each variable $x$, the atoms that refer to $x$ are a connected subtree of $T$.

A (B)CQ that has a join tree is called a tree query.
Example: Join Tree

$$\exists x, y, z, t, u, v, w. (r(x, y, z) \land r(t, u, y) \land s(u, v, y, z) \land q(t, w))$$
Join trees can be processed in polynomial time

Key ingredient: the semijoin operation

**Definition**

Given two relations $R[U]$ and $S[V]$, the semijoin $R^I \bowtie S^I$ is defined as $\pi_U(R^I \bowtie S^I)$.

Join trees can now be processed by computing semijoins bottom-up

$\sim$ Yannakakis’ Algorithm
Yannakakis’ Algorithm by Example

r(x,y,z)
s(u,v,y,z)
r(t,u,y)
q(t,w)

r(t,u,y)

s(u,v,y,z)

q(t,w)

r(x,y,z)

Markus Krötzsch, 28 April 2016
Yannakakis’ Algorithm by Example

s:

\[
\begin{array}{cccc}
2 & 8 & 3 & 5 \\
2 & 4 & 4 & 6 \\
3 & 4 & 2 & 3 \\
7 & 1 & 3 & 5 \\
8 & 5 & 6 & 4 \\
9 & 2 & 7 & 3 \\
\end{array}
\]

r:

\[
\begin{array}{cccc}
1 & 2 & 3 \\
3 & 3 & 5 \\
4 & 7 & 3 \\
7 & 9 & 7 \\
\end{array}
\]

q:

\[
\begin{array}{cccc}
2 & 3 \\
4 & 5 \\
4 & 7 \\
6 & 5 \\
7 & 2 \\
\end{array}
\]

s(u,v,y,z)

r(t,u,y)

q(t,w)

r(x,y,z)
Yannakakis’ Algorithm by Example

- $r(x,y,z)$
- $s(u,v,y,z)$
- $r(t,u,y)$
- $q(t,w)$

- $s:$
  - 2 8 3 5
  - 2 4 4 6
  - 3 4 2 3
  - 7 1 3 5
  - 8 5 6 4
  - 9 2 7 3

- $r:$
  - 1 2 3
  - 3 3 5
  - 4 7 3
  - 7 9 7

- $q:$
  - 2 3
  - 4 5
  - 6 5
  - 7 2
Yannakakis’ Algorithm by Example

\[ r(x, y, z) \]

\[ s(u, v, y, z) \]

\[ r(t, u, y) \]

\[ q(t, w) \]

\[ r(t, u, y) \]

\[ s(u, v, y, z) \]

\[ q(t, w) \]

\[ r(x, y, z) \]

\[ r(t, u, y) \]

\[ s(u, v, y, z) \]

\[ q(t, w) \]

\[ r(x, y, z) \]

\[ r(t, u, y) \]

\[ s(u, v, y, z) \]

\[ q(t, w) \]

\[ r(x, y, z) \]

\[ r(t, u, y) \]

\[ s(u, v, y, z) \]

\[ q(t, w) \]

\[ r(x, y, z) \]

\[ r(t, u, y) \]

\[ s(u, v, y, z) \]

\[ q(t, w) \]

\[ r(x, y, z) \]

\[ r(t, u, y) \]

\[ s(u, v, y, z) \]

\[ q(t, w) \]

\[ r(x, y, z) \]

\[ r(t, u, y) \]

\[ s(u, v, y, z) \]

\[ q(t, w) \]

\[ r(x, y, z) \]

\[ r(t, u, y) \]

\[ s(u, v, y, z) \]

\[ q(t, w) \]

\[ r(x, y, z) \]

\[ r(t, u, y) \]

\[ s(u, v, y, z) \]

\[ q(t, w) \]

\[ r(x, y, z) \]

\[ r(t, u, y) \]

\[ s(u, v, y, z) \]

\[ q(t, w) \]

\[ r(x, y, z) \]

\[ r(t, u, y) \]

\[ s(u, v, y, z) \]

\[ q(t, w) \]

\[ r(x, y, z) \]

\[ r(t, u, y) \]

\[ s(u, v, y, z) \]

\[ q(t, w) \]

\[ r(x, y, z) \]

\[ r(t, u, y) \]

\[ s(u, v, y, z) \]

\[ q(t, w) \]

\[ r(x, y, z) \]

\[ r(t, u, y) \]

\[ s(u, v, y, z) \]

\[ q(t, w) \]

\[ r(x, y, z) \]
Yannakakis’ Algorithm by Example

```
s:  2 8 3 5
   2 4 4 6
   3 4 2 3
   7 1 3 5
   8 5 6 4
   9 2 7 3
```

```
r:  1 2 3
    3 3 5
    4 7 3
    7 9 7
```

```
s(t,u,y,z)
```

```
r(t,u,y)
```

```
r(x,y,z)
```

```
q(t,w)
```

```:
...  1 2 3
...  3 3 5
...  4 7 3
...  7 9 7
```

```
s:
...  2 8 3 5
...  2 4 4 6
...  3 4 2 3
...  7 1 3 5
...  8 5 6 4
...  9 2 7 3
```
Yannakakis’ Algorithm by Example

```
s:  
2  8  3  5
2  4  4  6
3  4  2  3
7  1  3  5
8  5  6  4
9  2  7  3

r:  
1  2  3
3  3  5
4  7  3
7  9  7

s(u,v,y,z)  r(t,u,y)  q(t,w)

r(x,y,z)
X
```
Yannakakis’ Algorithm by Example

\[ r(x,y,z) \]
\[ s(u,v,y,z) \]
\[ r(t,u,y) \]
\[ q(t,w) \]

\[ r(t,u,y) \]
\[ s(u,v,y,z) \]
\[ q(t,w) \]

\[ s: \]
\[ 2 \ 8 \ 3 \ 5 \]
\[ 2 \ 4 \ 4 \ 6 \]
\[ 3 \ 4 \ 2 \ 3 \]
\[ 7 \ 1 \ 3 \ 5 \]
\[ 8 \ 5 \ 6 \ 4 \]
\[ 9 \ 2 \ 7 \ 3 \]

\[ r: \]
\[ 1 \ 2 \ 3 \]
\[ 3 \ 3 \ 5 \]
\[ 4 \ 7 \ 3 \]
\[ 7 \ 9 \ 7 \]

\[ q: \]
\[ 2 \ 3 \]
\[ 4 \ 5 \]
\[ 4 \ 7 \]
\[ 6 \ 5 \]
\[ 7 \ 2 \]
Yannakakis’ Algorithm by Example

The algorithm proceeds as follows:

1. **Start the Algorithm**: Choose a relation **r(x,y,z)**. In this example, we start with **r(x,y,z)**.

2. **Join with **s(u,v,y,z)****: Join **r(x,y,z)** with **s(u,v,y,z)**. The join condition is based on the values of **y**.

3. **Subsequent Joins**: Continue joining with subsequent relations according to the algorithm's rules. This process is illustrated with **q(t,w)**.

**Tables**:

- **s**:
  - 2 8 3 5
  - 2 4 4 6
  - 3 4 2 3
  - 7 1 3 5
  - 8 5 6 4
  - 9 2 7 3

- **r**:
  - 1 2 3
  - 3 3 5
  - 4 7 3
  - 7 9 7

- **q**:
  - 2 3
  - 4 5
  - 4 7
  - 6 5
  - 7 2
Yannakakis’ Algorithm by Example

\[
\begin{array}{cccc}
2 & 8 & 3 & 5 \\
2 & 4 & 4 & 6 \\
3 & 4 & 2 & 3 \\
7 & 1 & 3 & 5 \\
8 & 5 & 6 & 4 \\
9 & 2 & 7 & 3 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 3 \\
3 & 3 & 5 \\
4 & 7 & 3 \\
7 & 9 & 7 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 3 \\
3 & 3 & 5 \\
4 & 7 & 3 \\
7 & 9 & 7 \\
\end{array}
\]

Markus Krötzsch, 28 April 2016

Database Theory

slide 57 of 65
Yannakakis’ Algorithm by Example
Yannakakis’ Algorithm by Example

Markus Krötzsch, 28 April 2016  Database Theory  slide 60 of 65
Yannakakis’ Algorithm by Example

r(x, y, z)
s(u, v, y, z)
r(t, u, y)
q(t, w)

s:
2 8 3 5
2 4 4 6
3 4 2 3
7 1 3 5
8 5 6 4
9 2 7 3

r:
1 2 3
3 3 5
4 7 3
7 9 7

q:
2 3
4 5
4 7
6 5
7 2
Yannakakis’ Algorithm by Example

\[
\begin{align*}
\text{s:} & \quad \begin{array}{cccc}
2 & 8 & 3 & 5 \\
2 & 4 & 4 & 6 \\
3 & 4 & 2 & 3 \\
7 & 1 & 3 & 5 \\
8 & 5 & 6 & 4 \\
9 & 2 & 7 & 3
\end{array} \\
\text{r:} & \quad \begin{array}{cccc}
1 & 2 & 3 \\
3 & 3 & 5 \\
4 & 7 & 3 \\
7 & 9 & 7
\end{array} \\
\text{r(t,u,y)} & \\
\text{s(u,v,y,z)} & \text{q(t,w)}
\end{align*}
\]
Yannakakis’ Algorithm by Example

s:

<table>
<thead>
<tr>
<th>2</th>
<th>8</th>
<th>3</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>7</td>
<td>3</td>
</tr>
</tbody>
</table>

r:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>7</td>
</tr>
</tbody>
</table>

q:

<table>
<thead>
<tr>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
</tr>
</tbody>
</table>

r(x, y, z)

s(u, v, y, z)

q(t, w)
Yannakakis’ Algorithm: Summary

Polynomial time procedure for answering BCQs

Does not immediately compute answers in the version given here
⇒ modifications needed

Even tree queries can have exponentially many results, but each can be computed (not just checked) in $P$
⇒ output-polynomial computation of results
Summary and Outlook

 Conjunctive queries (CQs) are an important special case of FO queries

 Boolean CQ answering, the homomorphism problem and constraint satisfaction problems are equivalent and \( \mathsf{NP} \)-complete

 CQ answering is simpler, namely in \( \mathsf{P} \), when CQs are tree queries
  
   - Check acyclicity with GYO algorithm
   - Evaluate query using Yannakakis’ Algorithm

 Open questions:
  
   - Tree queries are rather special. Are there more general conditions for good queries?
   - What about query optimisation?