DATABASE THEORY

Lecture 6: Conjunctive Queries

Markus Krötzsch
Knowledge-Based Systems

TU Dresden, 30th April 2019
The evaluation of FO queries is

- PSpace-complete for combined complexity
- PSpace-complete for query complexity
- $AC^0$-complete for data complexity

$\sim$ 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 6.1:** 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 6.2:** A conjunctive query (CQ) is a relational algebra expression that uses only the operations select $\sigma_{n=m}$, project $\pi_{a_1,\ldots,a_n}$, join $\bowtie$, and renaming $\delta_{a_1,\ldots,a_n \rightarrow b_1,\ldots,b_n}$.

Renaming is only relevant in named perspective.

$\leadsto$ 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
$\sim$ 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?”

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

### Stops:

<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>

### Connect:

<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?”

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

**Stops:**

<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>

**Connect:**

<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?”

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

<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>
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

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

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

**Theorem 6.3:** BCQ query answering is in NP for combined complexity (and also for query complexity).

$\leadsto$ 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 \( I \) and \( J \)

(= database instances, interpretations, hypergraphs)

**Definition 6.4:** A homomorphism \( h \) from \( I \) to \( J \) is a function \( h : \Delta^I \rightarrow \Delta^J \) such that, for all relation names \( R \):

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

The **homomorphism problem** is the question if there is a homomorphism from \( I \) to \( J \).
Example: Three-colouring as Homomorphism

\[ I : \]

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

\[ J : \]

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

3-colouring is NP-hard
Example: Three-colouring as Homomorphism

$I:$

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

$J:$

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

$3$-colouring is NP-hard

Markus Krötzsch, 30th April 2019
Example: Three-colouring as Homomorphism

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

\[ J : \begin{array}{c}
\end{array} \]

3-colouring is NP-hard.
Example: Three-colouring as Homomorphism

\[ I : \]

\[ J : \]

3-colouring is NP-hard
\[ \sim \] 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 $\mathcal{J} \models Q_{\mathcal{I}}$
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 $R_a(x)$

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

**Theorem 6.5:** BCQ answering is
- NP-complete for combined complexity
- NP-complete for query complexity
- in \( AC^0 \) for data complexity (inherited from FO queries)
Another important problem equivalent to BCQ answering:

**Definition 6.6:** 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).

$\leadsto$ alternative notation for BCQ answering/homomorphism problem
A combinatorial crossword puzzle:

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

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

Constraints:

1 vertically: H E A R T
H O N E Y
I R O N Y
L O G I C

1 horizontally: H A P P Y
I N F E R
L A B O R
L A T E R

5 vertically: R A D I O
R E T R O
Y A C H T
Y E R B A

...
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
Tractable CQ Answering
How to reduce complexities?

NP-complete query complexity is still intractable

Can we do better?

Idea:

- We have encoded 3-colourability to show NP-hardness
- Can we avoid hardness by restricting to certain cases?
The problem of 3-colourability was captured by the target structure of the homomorphism:

This part of the problem is called the template in CSP.
Excursion: CSP Complexity

The problem of 3-colourability was captured by the target structure of the homomorphism:

![Graph](image)

This part of the problem is called the template in CSP.

Another template:
Excursion: CSP Complexity

The problem of 3-colourability was captured by the target structure of the homomorphism:

This part of the problem is called the template in CSP.

Another template:

\[ \sim 2\text{-colourability}, \text{ a well-known problem in P} \]
Can we study CSP complexity based on a given template?

Yes, for a fixed template, there are still infinitely many instances, and we can ask for the complexity of the resulting language.

In 1993, Feder and Vardi famously conjecture the following Feder–Vardi Conjecture (simplified):

For any fixed template, the CSP-problem is either NP-complete or contained in P.

What's the big deal?

• According to Ladner's Theorem, if P \neq NP, then there are problems that are neither in P nor hard for NP—so-called NP-intermediate problems.

• According to Feder/Vardi, such problems do not exist in CSPs (which might contribute to explaining why they seem to be relatively rare).

For more details, see our lecture Complexity Theory.
Can we study CSP complexity based on a given template?

Yes, for a fixed template, there are still infinitely many instances, and we can ask for the complexity of the resulting language.
Can we study CSP complexity based on a given template?

Yes, for a fixed template, there are still infinitely many instances, and we can ask for the complexity of the resulting language.

In 1993, Feder and Vardi famously conjecture the following

**Feder–Vardi Conjecture (simplified):** For any fixed template, the CSP-problem is either NP-complete or contained in P.

What’s the big deal?
Excursion: CSP Complexity (2)

Can we study CSP complexity based on a given template?

Yes, for a fixed template, there are still infinitely many instances, and we can ask for the complexity of the resulting language.

In 1993, Feder and Vardi famously conjecture the following

**Feder–Vardi Conjecture (simplified):** For any fixed template, the CSP-problem is either NP-complete or contained in P.

**What’s the big deal?**

- According to Ladner’s Theorem, if P ≠ NP, then there are problems that are neither in P nor hard for NP—so-called NP-intermediate problems\(^1\)

- According to Feder/Vardi, such problems do not exist in CSPs (which might contribute to explaining why they seem to be relatively rare)

\(^1\) For more details, see our lecture Complexity Theory
The End of the Feder–Vardi Conjecture

In 2017, the Feder–Vardi Conjecture has been proven independently by two authors:

- Andrei A. Bulatov: A Dichotomy Theorem for Nonuniform CSPs (FOCS 2017)
- Dmitriy Zhuk: A Proof of CSP Dichotomy Conjecture (FOCS 2017)

A third attempt by Tomás Feder, Jeff Kinne, and Arash Rafiey did not (so far) work out.

These and further results have also significantly improved our understanding of tractable templates. For more information, look out for lectures by Manuel Bodirsky (TU Dresden).
The End of the Feder–Vardi Conjecture

In 2017, the Feder–Vardi Conjecture has been proven independently by two authors:

- Andrei A. Bulatov: **A Dichotomy Theorem for Nonuniform CSPs** (FOCS 2017)
- Dmitriy Zhuk: **A Proof of CSP Dichotomy Conjecture** (FOCS 2017)

A third attempt by Tomás Feder, Jeff Kinne, and Arash Rafiey did not (so far) work out.

These and further results have also significantly improved our understanding of tractable templates. For more information, look out for lectures by Manuel Bodirsky (TU Dresden).

Can we exploit this for better BCQ answering complexities?
The End of the Feder–Vardi Conjecture

In 2017, the Feder–Vardi Conjecture has been proven independently by two authors:

- Andrei A. Bulatov: **A Dichotomy Theorem for Nonuniform CSPs** (FOCS 2017)
- Dmitriy Zhuk: **A Proof of CSP Dichotomy Conjecture** (FOCS 2017)

A third attempt by Tomás Feder, Jeff Kinne, and Arash Rafiey did not (so far) work out.

These and further results have also significantly improved our understanding of tractable templates. For more information, look out for lectures by Manuel Bodirsky (TU Dresden).

Can we exploit this for better BCQ answering complexities?

Not really:

- The template corresponds to the database in CQ answering
- We do not want to answer many queries over fixed databases
- Assuming that only families of “easy” databases are considered is not realistic
Towards Better Complexities

**Idea 2:**
- Searching a match may require backtracking, eventually exploring all options
- Can we constrain the query to avoid this?
Towards Better Complexities

Idea 2:

- Searching a match may require backtracking, eventually exploring all options
- Can we constrain the query to avoid this?

![Diagram with black squares and words]

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

**Idea 2:**

- Searching a match may require backtracking, eventually exploring all options
- Can we constrain the query to avoid this?

<table>
<thead>
<tr>
<th>H</th>
<th>A</th>
<th>P</th>
<th>P</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Towards Better Complexities

Idea 2:

- Searching a match may require backtracking, eventually exploring all options
- Can we constrain the query to avoid this?

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

Idea 2:

• Searching a match may require backtracking, eventually exploring all options
• Can we constrain the query to avoid this?

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

{the problem is with the cycles

Markus Krötzsch, 30th April 2019
Towards Better Complexities

Idea 2:

• Searching a match may require backtracking, eventually exploring all options
• Can we constrain the query to avoid this?

\[
\begin{array}{cccc}
H & A & P & Y \\
O & & & A \\
N & E & W & C \\
E & A & H & \\
Y & Y & T & \\
\end{array}
\]
Towards Better Complexities

Idea 2:

• Searching a match may require backtracking, eventually exploring all options
• Can we constrain the query to avoid this?

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

{the problem is with the cycles

Markus Krötzsch, 30th April 2019

Database Theory
Towards Better Complexities

Idea 2:

- Searching a match may require backtracking, eventually exploring all options
- Can we constrain the query to avoid this?

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

\[ \rightsquigarrow \text{the problem is with the cycles} \]
Example: Cyclic CQs

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

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

\( \leadsto \) cyclic query
Example: Acyclic CQs

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

$$\exists y_c, y_m, y_f, y_{mm}, y_{mf} \cdot \text{mother}(y_c, y_m) \land \text{father}(y_c, y_f) \land \text{married}(y_m, y_{mm}) \land \text{married}(y_{mf}, y_f)$$

$$\sim$$ acyclic query
Defining Acyclic Queries

Queries in general are hypergraphs

What does “acyclic” mean?
Defining Acyclic Queries

Queries in general are hypergraphs

\(\sim\) 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 6.7: 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

Markus Krötzsch, 30th April 2019
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:

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

Any ears?
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 6.8:** The GYO-reduct is $\langle \emptyset, \emptyset \rangle$ if and only if the GYO’-reduct is $\langle \emptyset, \emptyset \rangle$

$\sim$ 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 6.9:** 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 6.10:** 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

s:

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

r:

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

q:

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

r(t,u,y)

s(u,v,y,z)

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)

r(t,u,y)

s(u,v,y,z)

q(t,w)

r(x,y,z)

Markus Krötzsch, 30th April 2019
Database Theory slide 33 of 35
Yannakakis’ Algorithm by Example

\begin{align*}
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} \\
s(u,v,y,z) & \quad r(t,u,y) & \quad q(t,w) \\
\end{align*}
Yannakakis’ Algorithm by Example

Let's consider the following relations:
- \( s(u,v,y,z) \)
- \( r(x,y,z) \)
- \( q(t,w) \)

The algorithm involves matching elements between these relations. Here's a visual representation of the algorithm:

1. Start with the relation \( r(t,u,y) \).
2. Find corresponding elements in \( s(u,v,y,z) \) and \( q(t,w) \) as per the matching criteria.
3. If a match is found, proceed with the next step.
4. Continue this process until all matching criteria are met.

The matching criteria are as follows:
- For \( r(t,u,y) \): Match \( t \) with \( s \) and \( u \) with \( q \).
- For \( s(u,v,y,z) \): Ensure \( v \) and \( z \) are also matched.
- For \( q(t,w) \): Ensure \( w \) is matched.

The algorithm ensures a correct matching of elements across the relations.
Yannakakis’ Algorithm by Example

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

\[ ⋉ \]

The algorithm proceeds as follows:

**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

Markus Krötzsch, 30th April 2019

Database Theory

slide 33 of 35
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, 30th April 2019
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  

Markus Krötzsch, 30th April 2019  
Database Theory  
slide 33 of 35
Yannakakis’ Algorithm by Example
Yannakakis’ Algorithm by Example

s:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>8</td>
<td>3</td>
<td>5</td>
</tr>
<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:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<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:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>7</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)
r(t,u,y)
r(t,w)

r(t,u,y)
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

q:

2 3
4 5
4 7
6 5
7 2

r(x,y,z)

r(t,u,y)

s(u,v,y,z)

q(t,w)
Yannakakis’ Algorithm by Example

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

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

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

Markus Krötzsch, 30th April 2019
Yannakakis’ Algorithm by Example

\[
\begin{align*}
\text{s:} & \quad \begin{array}{cccc}
2 & 8 & 3 & 5 \\
2 & 4 & 4 & 6 \\
3 & 4 & 2 & 3 \\
7 & 1 & 3 & 5 \\
8 & 5 & 6 & 4 \\
9 & 2 & 7 & 3
\end{array} \\
\text{r:} & \quad \begin{array}{cccc}
1 & 2 & 3 \\
3 & 3 & 5 \\
4 & 7 & 3 \\
7 & 9 & 7
\end{array} \\
\text{q:} & \quad \begin{array}{cc}
2 & 3 \\
4 & 5 \\
4 & 7 \\
6 & 5 \\
7 & 2
\end{array}
\end{align*}
\]
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>
```

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

Markus Krötzsch, 30th April 2019
Yannakakis’ Algorithm by Example

\[ r(x,y,z) \]
\[ s(u,v,y,z) \]
\[ r(t,u,y) \]
\[ q(t,w) \]
Yannakakis’ Algorithm by Example

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

\[ \triangleright \]

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:

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

Markus Krötzsch, 30th April 2019

Database Theory
Yannakakis’ Algorithm by Example

r(x,y,z)

s(u,v,y,z)

r(t,u,y)

q(t,w)

\[ s: \]

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

\[ r: \]

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

\[ q: \]

\[
\begin{array}{cccc}
2 & 3 \\
4 & 5 \\
4 & 7 \\
6 & 5 \\
7 & 2 \\
\end{array}
\]
Yannakakis’ Algorithm by Example

\[
\begin{align*}
\text{s:} & \quad 2 & 8 & 3 & 5 \\
& \quad 2 & 4 & 4 & 6 \\
& \quad 3 & 4 & 2 & 3 \\
& \quad 7 & 1 & 3 & 5 \\
& \quad 8 & 5 & 6 & 4 \\
& \quad 9 & 2 & 7 & 3 \\
\text{r:} & \quad 1 & 2 & 3 \\
& \quad 3 & 3 & 5 \\
& \quad 4 & 7 & 3 \\
& \quad 7 & 9 & 7 \\
\text{q:} & \quad 2 & 3 \\
& \quad 4 & 5 \\
& \quad 6 & 5 \\
& \quad 7 & 2 \\
\end{align*}
\]
Yannakakis’ Algorithm: Summary

Polynomial time procedure for answering BCQs

Does not immediately compute answers in the version given here
\[\rightarrow\text{modifications needed}\]

Even tree queries can have exponentially many results,
but each can be computed (not just checked) in P
\[\rightarrow\text{output-polynomial computation of results}\]
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 NP-complete

CQ answering is simpler, namely in 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?