TECHNISCHE UNIVERSITÄT DRESDEN

DATABASE THEORY

Lecture 5: Conjunctive Queries

Markus Krötzsch

TU Dresden, 28 April 2016

Overview

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

Review: FO Query Complexity

The evaluation of FO queries is

- $\textsc{PSpace}$-complete for combined complexity
- $\textsc{PSpace}$-complete for query complexity
- AC^0-complete for data complexity

$\leadsto$ $\textsc{PSpace}$ is rather high
$\leadsto$ 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

$$\exists y_1, \ldots, y_m . A_1 \wedge \ldots \wedge A_\ell$$

where each A_i is an atom of the form $\mathsf{R}(t_1, \ldots, t_k)$. 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)

$$\exists y_{\mathsf{SID}}, y_{\mathsf{Stop}}, y_{\mathsf{To}} . \mathsf{Stops}(y_{\mathsf{SID}}, y_{\mathsf{Stop}}, \texttt{"true"}) \wedge \mathsf{Connect}(y_{\mathsf{SID}}, y_{\mathsf{To}}, x_{\mathsf{Line}})$$

Conjunctive Queries in Relational Calculus

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 $\pi_{a_1,\ldots,a_n}$, join $\bowtie$, and renaming $\delta_{a_1,\ldots,a_n \to b_1,\ldots,b_n}$.

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
$\leadsto$ the reason for this is query complexity (as we shall see)

Boolean Conjunctive Queries

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?"

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

Stops:

SID	Stop	Accessible
17	Hauptbahnhof	true
42	Helmholtzstr.	true
57	Stadtgutstr.	true
123	Gustav-Freytag-Str.	false
...	...	...

Connect:

From	To	Line
57	42	85
17	789	3
...	...	...

Boolean Conjunctive Queries

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?”

$$\exists y_{\mathrm{SID}}, y_{\mathrm{Stop}}, y_{\mathrm{To}}, y_{\mathrm{Line}}.\mathrm{Stops}(y_{\mathrm{SID}}, y_{\mathrm{Stop}}, \texttt{"true"}) \wedge \mathrm{Connect}(y_{\mathrm{SID}}, y_{\mathrm{To}}, y_{\mathrm{Line}})$$

Stops:

SID	Stop	Accessible
17	Hauptbahnhof	true
42	Helmholtzstr.	true
57	Stadtgutstr.	true
123	Gustav-Freytag-Str.	false
...	...	...

Connect:

From	To	Line
57	42	85
17	789	3
...	...	...

Boolean Conjunctive Queries

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?"

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

Stops:

SID	Stop	Accessible
17	Hauptbahnhof	true
42	Helmholtzstr.	true
57	Stadtgutstr.	true
123	Gustav-Freytag-Str.	false
...	...	...

Connect:

From	To	Line
57	42	85
17	789	3
...	...	...

How Hard is it to Answer CQs?

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

- Checking if a single ground atom $\mathsf{R}(c_1, \ldots, c_k)$ 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 $\mathsf{R}(c_1, \ldots, c_k)$ 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

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

Definition

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

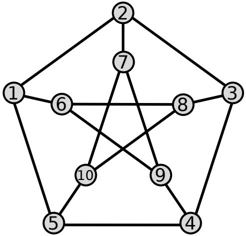
$$\text{if} \quad \langle d_1, \ldots, d_n \rangle \in R^{\mathcal{I}} \quad \text{then} \quad \langle h(d_1), \ldots, h(d_n) \rangle \in R^{\mathcal{J}}.$$

The homomorphism problem is the question if there is a homomorphism from $\mathcal{I}$ to $\mathcal{J}$.

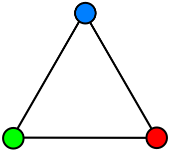
Example: Three-colouring as Homomorphism

$\mathcal{I}$:

1	2
1	5
1	6
2	3
2	7
3	4
3	8
. . .	. . .



$\mathcal{J}$:

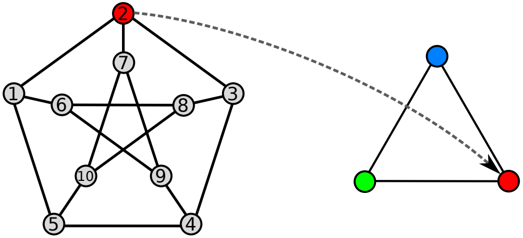


r	g
r	b
g	r
g	b
b	r
b	g

Example: Three-colouring as Homomorphism

$\mathcal{I}$:

1	2
1	5
1	6
2	3
2	7
3	4
3	8
...	...



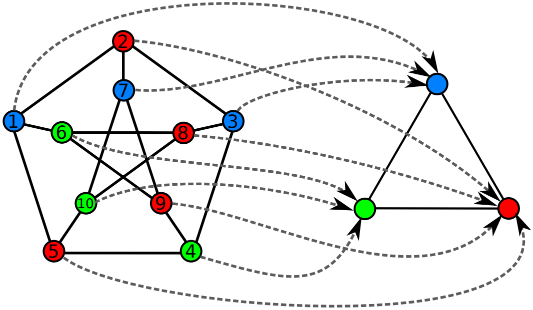
$\mathcal{J}$:

r	g
r	b
g	r
g	b
b	r
b	g

Example: Three-colouring as Homomorphism

$\mathcal{I}$:

1	2
1	5
1	6
2	3
2	7
3	4
3	8
...	...



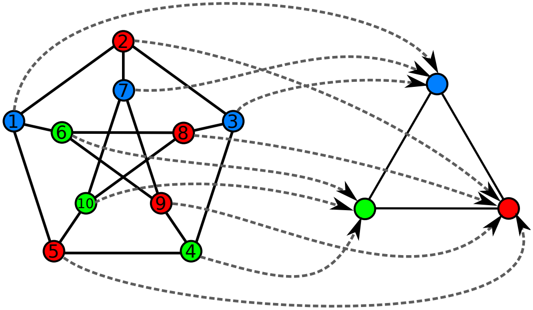
$\mathcal{J}$:

r	g
r	b
g	r
g	b
b	r
b	g

Example: Three-colouring as Homomorphism

$\mathcal{I}$:

1	2
1	5
1	6
2	3
2	7
3	4
3	8
...	...



$\mathcal{J}$:

r	g
r	b
g	r
g	b
b	r
b	g

3-colouring is NP-hard

$\leadsto$ the homomorphism problem is NP-hard

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 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 $R_a(x)$

$\leadsto$ both problems are equivalent

Complexity of Conjunctive Query Answering

We showed that BCQ answering is in NP and that the homomorphism problem is NP-hard, therefore:

Theorem

BCQ answering is

- NP-complete for combined complexity
- NP-complete for query complexity
- in AC^0 for data complexity (inherited from FO queries)

Constraint Satisfaction Problems

Another important problem equivalent to BCQ answering

Definition

A constraint satisfaction problem (CSP) over a domain Δ 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 Δ.

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

$\leadsto$ 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:

x_1	x_2	x_3	x_4	x_5		x_6
x_7				x_8	x_9	x_{10}
x_{11}	x_{12}	x_{13}		x_{14}		x_{15}
x_{16}		x_{17}		x_{18}		x_{19}
x_{20}	x_{21}	x_{22}	x_{23}	x_{24}	x_{25}	x_{26}

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

...

Equivalent Problems

Summing up, the following problems are equivalent:

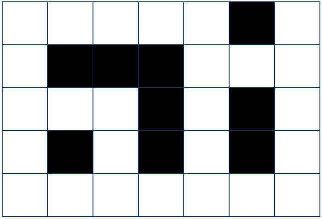
- Answering a conjunctive query over a database instance
- Finding a homomorphism from a relational structure to another
- Solving a constraint satisfaction problem

Each of these problems is NP-complete

Towards Better Complexities

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

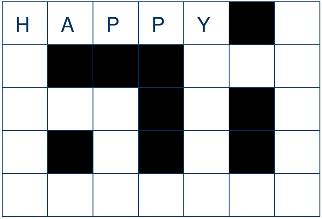
Problem: searching a match may require backtracking, eventually exploring all options



Towards Better Complexities

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

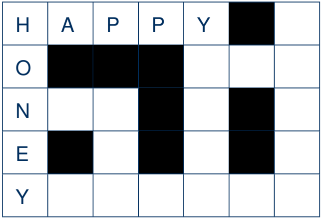
Problem: searching a match may require backtracking, eventually exploring all options



Towards Better Complexities

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

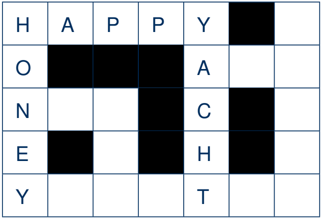
Problem: searching a match may require backtracking, eventually exploring all options



Towards Better Complexities

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

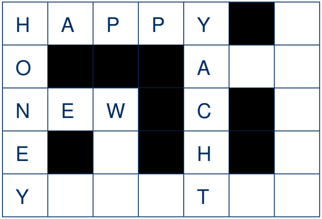
Problem: searching a match may require backtracking, eventually exploring all options



Towards Better Complexities

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

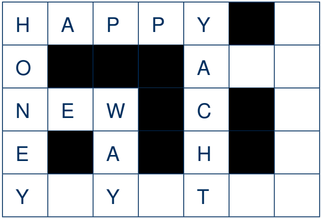
Problem: searching a match may require backtracking, eventually exploring all options



Towards Better Complexities

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

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



Towards Better Complexities

NP-complete problems are still intractable
$\leadsto$ 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		A		H		
Y		Y		T		

Towards Better Complexities

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

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

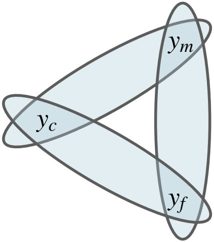
H	A	P	P	Y		
O				A		
N	E	W		C		
E		A		H		
Y		Y		T		

Intuition: life would be easier if we would not have to go back so much ...
$\leadsto$ the problem is with the cycles

Example: Cyclic CQs

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

$$\exists y_c, y_m, y_f.\mathsf{mother}(y_c, y_m) \wedge \mathsf{father}(y_c, y_f) \wedge \mathsf{married}(y_m, y_f)$$

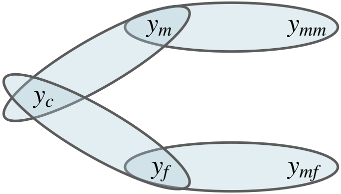


$\leadsto$ cyclic query

Example: Acyclic CQs

"Is there a child whose parents are married with someone?"

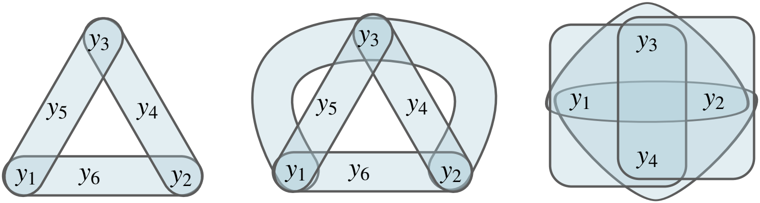
$$\exists y_c, y_m, y_f, y_{mm}, y_{mf}.\text{mother}(y_c, y_m) \wedge \text{father}(y_c, y_f) \wedge \text{married}(y_m, y_{mm}) \wedge \text{married}(y_{mf}, y_f)$$



$\rightsquigarrow$ acyclic query

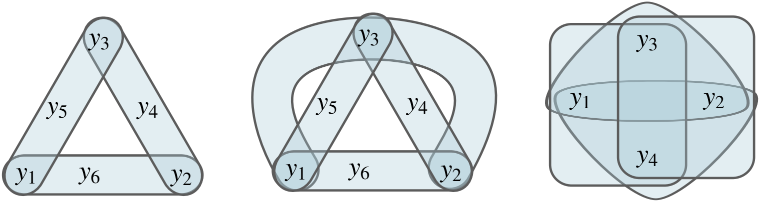
Defining Acyclic Queries

Queries in general are hypergraphs
⇝ What does “acyclic” mean?



Defining Acyclic Queries

Queries in general are hypergraphs
⇝ What does "acyclic" mean?

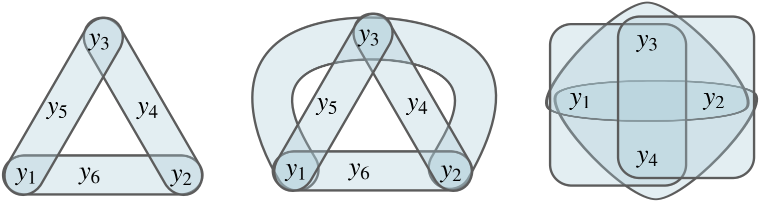


View hypergraphs as graphs to check acyclicity?

- 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:

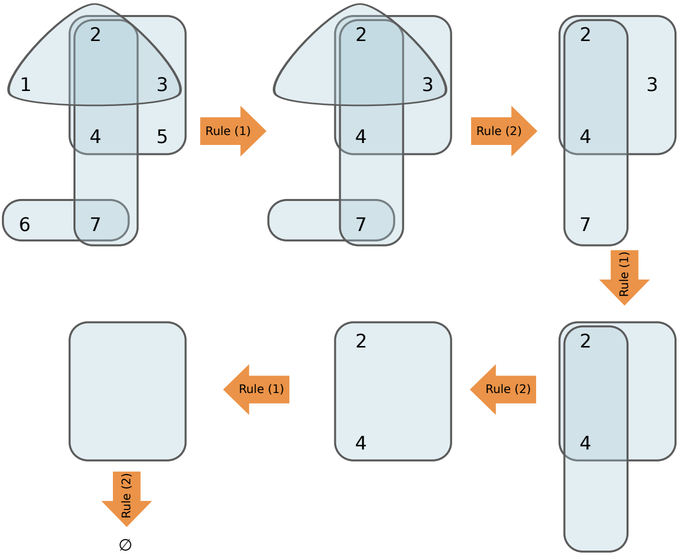
(1) Delete all vertices that occur in at most one hyperedge

(2) Delete all hyperedges that are empty or that are contained in other hyperedges

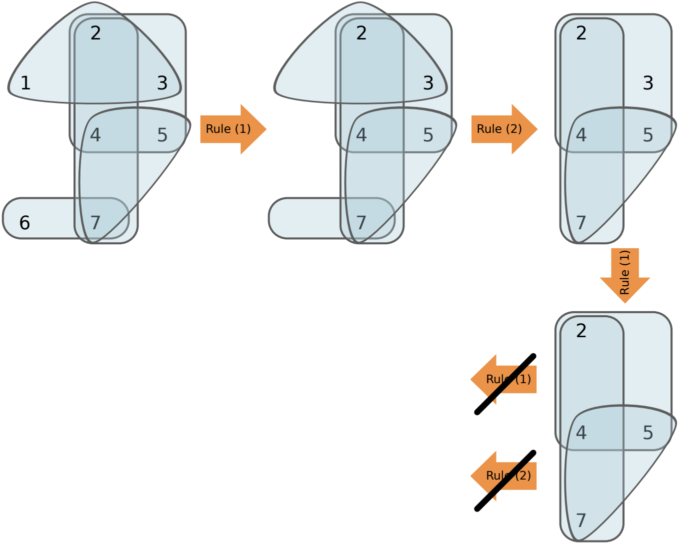
Definition

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



Example 2: GYO-Reduction



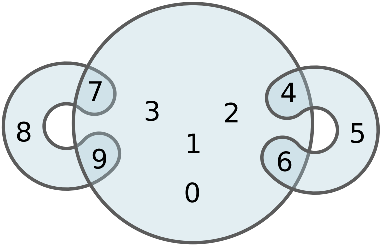
Alternative Version of GYO-Reduction

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.

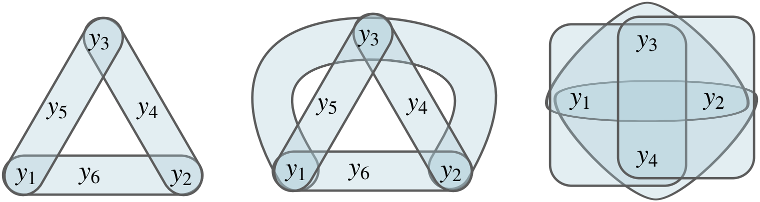
Example:



$\rightsquigarrow$ 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

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

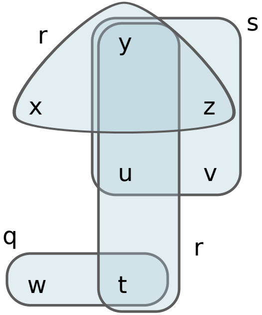
Definition

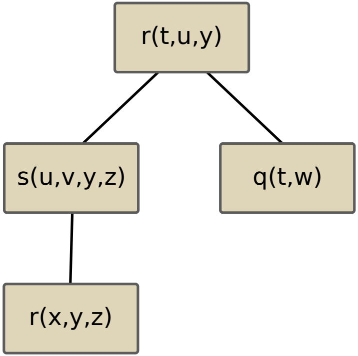
A join tree of a (B)CQ is an arrangement of its query atoms in a tree structure T, such that for each variable x, the atoms that refer to x are a connected subtree of T.

A (B)CQ that has a join tree is called a tree query.

Example: Join Tree

$$\exists x, y, z, t, u, v, w.(r(x, y, z) \land r(t, u, y) \land s(u, v, y, z) \land q(t, w))$$





Processing Join Trees Efficiently

Join trees can be processed in polynomial time

Key ingredient: the semijoin operation

Definition

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

Join trees can now be processed by computing semijoins bottom-up
$\leadsto$ Yannakakis' Algorithm

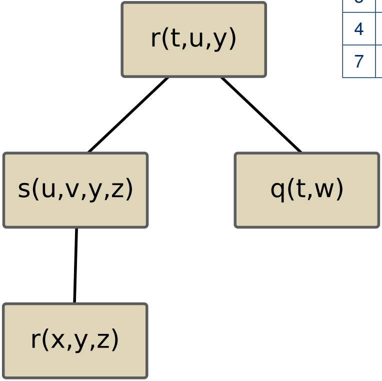
Yannakakis' Algorithm by Example

r:

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



q:

2	3
4	5
4	7
6	5
7	2

r:

1	2	3
3	3	5
4	7	3
7	9	7

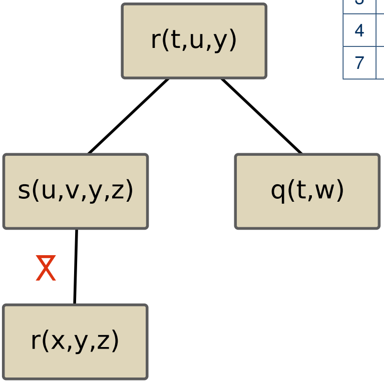
Yannakakis' Algorithm by Example

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



r:

1	2	3
3	3	5
4	7	3
7	9	7

q:

2	3
4	5
4	7
6	5
7	2

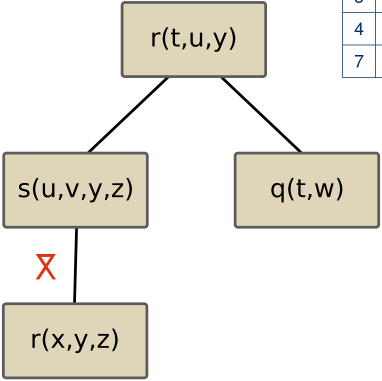
Yannakakis' Algorithm by Example

r:

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



q:

2	3
4	5
4	7
6	5
7	2

r:

1	2	3
3	3	5
4	7	3
7	9	7

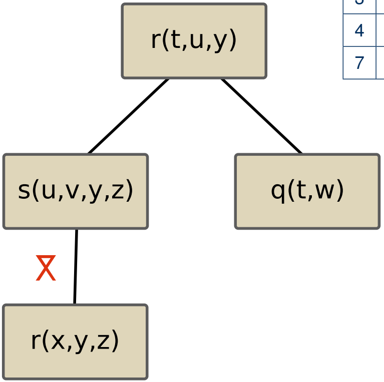
Yannakakis' Algorithm by Example

r:

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



q:

2	3
4	5
4	7
6	5
7	2

r:

1	2	3
3	3	5
4	7	3
7	9	7

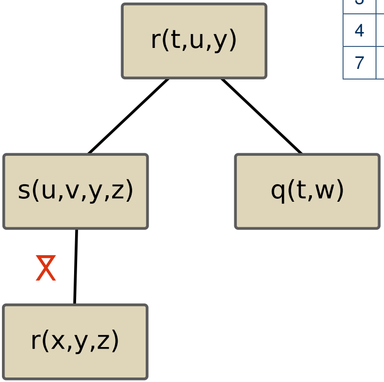
Yannakakis' Algorithm by Example

r:

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



q:

2	3
4	5
4	7
6	5
7	2

r:

1	2	3
3	3	5
4	7	3
7	9	7

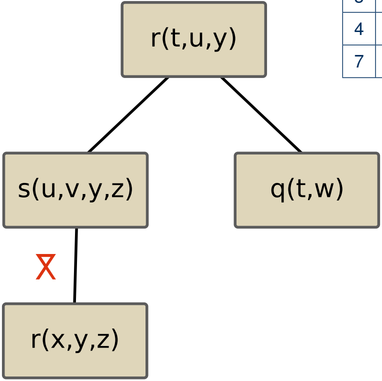
Yannakakis' Algorithm by Example

r:

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



q:

2	3
4	5
4	7
6	5
7	2

r:

1	2	3
3	3	5
4	7	3
7	9	7

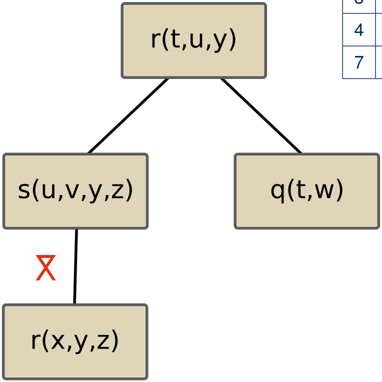
Yannakakis' Algorithm by Example

r:

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



q:

2	3
4	5
4	7
6	5
7	2

r:

1	2	3
3	3	5
4	7	3
▶ 7	9	7

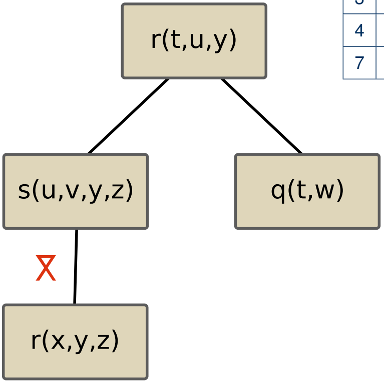
Yannakakis' Algorithm by Example

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



r:

1	2	3
3	3	5
4	7	3
7	9	7

q:

2	3
4	5
4	7
6	5
7	2

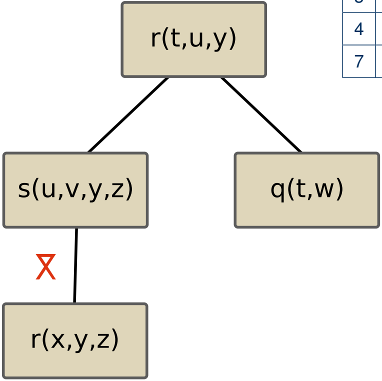
Yannakakis' Algorithm by Example

r:

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



q:

2	3
4	5
4	7
6	5
7	2

r:

1	2	3
3	3	5
4	7	3
7	9	7

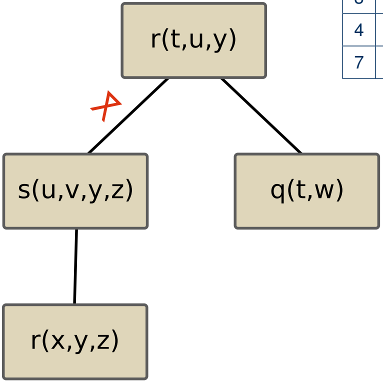
Yannakakis’ Algorithm by Example

r:

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



q:

2	3
4	5
4	7
6	5
7	2

r:

1	2	3
3	3	5
4	7	3
7	9	7

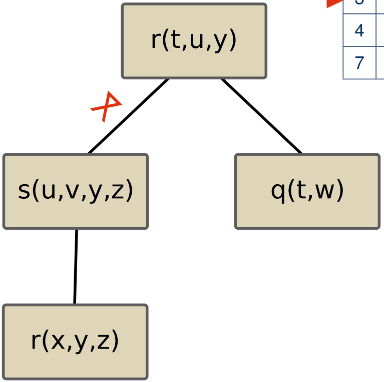
Yannakakis' Algorithm by Example

r:

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



q:

2	3
4	5
4	7
6	5
7	2

r:

1	2	3
3	3	5
4	7	3
7	9	7

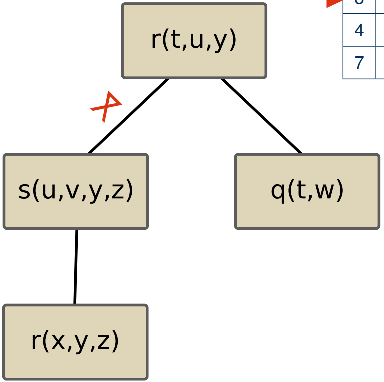
Yannakakis' Algorithm by Example

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:

1	2	3
3	3	5
4	7	3
7	9	7

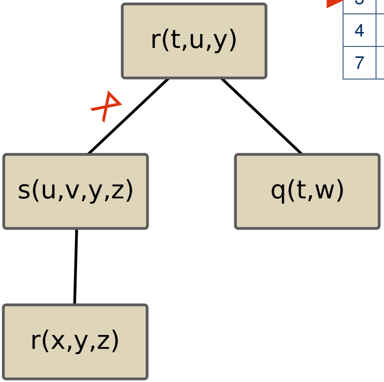
Yannakakis' Algorithm by Example

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:

1	2	3
3	3	5
4	7	3
7	9	7

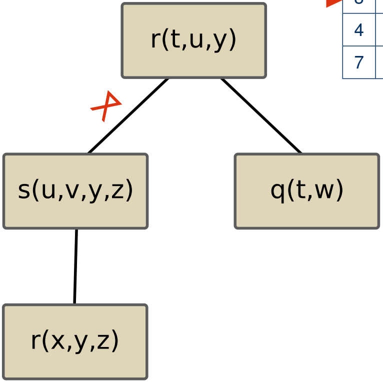
Yannakakis' Algorithm by Example

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:

1	2	3
3	3	5
4	7	3
7	9	7

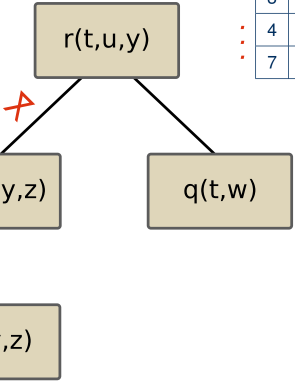
Yannakakis' Algorithm by Example

r:

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



q:

2	3
4	5
4	7
6	5
7	2

r:

1	2	3
3	3	5
4	7	3
7	9	7

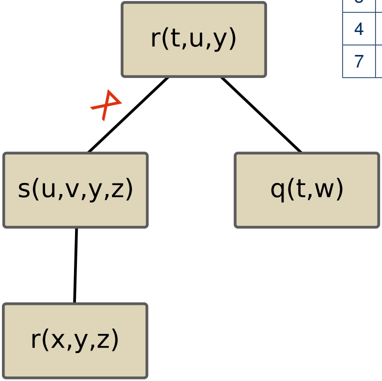
Yannakakis' Algorithm by Example

r:

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



q:

2	3
4	5
4	7
6	5
7	2

r:

1	2	3
3	3	5
4	7	3
7	9	7

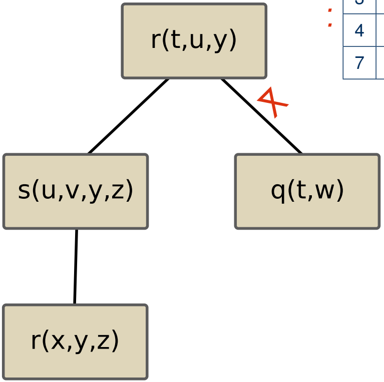
Yannakakis' Algorithm by Example

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:

1	2	3
3	3	5
4	7	3
7	9	7

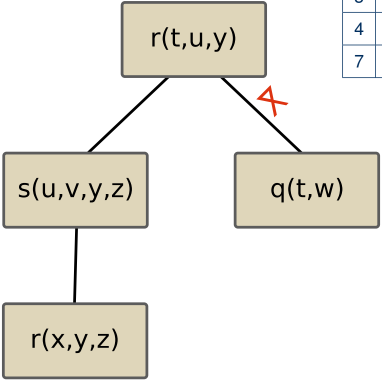
Yannakakis' Algorithm by Example

r:

~~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



q:

2	3
4	5
4	7
6	5
7	2

r:

1	2	3
3	3	5
4	7	3
7	9	7

Yannakakis' Algorithm: Summary

Polynomial time procedure for answering BCQs

Does not immediately compute answers in the version given here
$\leadsto$ modifications needed

Even tree queries can have exponentially many results,
but each can be computed (not just checked) in P
$\leadsto$ 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?