Database Theory and Knowledge Representation

2nd Lecture

David Carral

University of Montpellier

October 28, 2021

Summary and Outlook

- The relational data model
- The relational calculus: RA and FO queries are equivalent
- Complexity of FO query entailment: ExpTime

Outline:

- Complexity of FO query entailment: PSpace
- Complexity of CQ answering: NP
- Complexity of tree-like queries: P

Remark

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSpace} \subseteq \mathrm{ExpTime}$$

7. Evaluating FO Queries

An Algorithm for Evaluating FO Queries

```
function Eval(φ, I)

01  switch (φ) {
02      case p(c_1, ..., c_n) : return p(c_1, ..., c_n) ∈ I
03      case ¬ψ : return ¬Eval(ψ, I)
04      case ψ_1 ∧ ψ_2 : return Eval(ψ_1, I) ∧ Eval(ψ_2, I)
05      case ∃x.ψ :
06          for c ∈ Δ^I {
07              if Eval(ψ[x ↦ c], I) then return true
08          }
09          return false
10  }
```

FO Algorithm Worst-Case Runtime

Let m be the size of φ, and let $n = |\mathcal{I}|$ (total table sizes)

- How many recursive calls of Eval are there?
  $\rightsquigarrow$ one per subexpression: at most m
- Maximum depth of recursion?
  $\rightsquigarrow$ bounded by total number of calls: at most m
- Maximum number of iterations of **`for`** loop?
  $\rightsquigarrow |\Delta^{\mathcal{I}}| \leq n$ per recursion level
  $\rightsquigarrow$ at most n^m iterations
- Checking $P(c_1, \ldots, c_n) \in \mathcal{I}$ can be done in linear time w.r.t. n

Runtime in $m \cdot n^m \cdot n = m \cdot n^{m+1}$

Time Complexity of FO Algorithm

Let m be the size of φ, and let $n = |\mathcal{I}|$ (total table sizes)

Runtime in $m \cdot n^{m+1}$

Theorem

Time complexity of FO query evaluation

- Combined complexity: in $\mathrm{ExpTime}$
- Data complexity (m is constant): in P
- Query complexity (n is constant): in $\mathrm{ExpTime}$

FO Algorithm Worst-Case Memory Usage

We can get better complexity bounds by looking at memory!
Let m be the size of φ, and let $n = |\mathcal{I}|$ (total table sizes)

$\rightsquigarrow$ on the whiteboard

Theorem

The evaluation of FO queries is PSpace-complete with respect to combined complexity.

Remark

One can show that FO query entailment is PSpace-hard via reduction to True QBF.

6. Conjunctive Queries

Conjunctive Queries

- Problem: answering FO queries is hard.
- Idea: restrict FO queries to conjunctive, positive features

Definition: Conjunctive Queries

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 $R(t_1, \ldots, t_k)$. In other words, a CQ is an FO query that only uses conjunctions of atoms and (outer) existential quantifiers.

Example: "Find all lines with an accessible stop":

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

Discuss

Can I express all FO queries as CQs? What is the complexity of BCQ entailment?

Exercises: CQ Examples

Films

Title	**Director**	**Actor**
The Imitation Game	Tyldum	Cumberbatch
The Imitation Game	Tyldum	Knightley
…	…	…
Internet's Own Boy	Knappenberger	Swartz
Internet's Own Boy	Knappenberger	Lessig
Internet's Own Boy	Knappenberger	Berners-Lee
…	…	…
Dogma	Smith	Damon
Dogma	Smith	Affleck

Venues

Cinema	**Address**	**Phone**
UFA	St. Petersburger Str. 24	4825825
Schauburg	Königsbrücker Str. 55	8032185
…	…	…

Program

Cinema	**Title**	**Time**
Schauburg	The Imitation Game	19:30
Schauburg	Dogma	20:45
UFA	The Imitation Game	22:45

5. List the pairs of persons such that the first directed the second in a film, and vice versa.

$$\exists y_T, z_T.\ \mathrm{Films}(y_T, x_D, x_A) \wedge \mathrm{Films}(z_T, x_A, x_D)[x_D, x_A]$$

6. List the names of directors who have acted in a film they directed.

$$\exists y_T.\ \mathrm{Films}(y_T, x_D, x_D)[x_D]$$

Exercises: CQ Examples

Films

Title	**Director**	**Actor**
The Imitation Game	Tyldum	Cumberbatch
The Imitation Game	Tyldum	Knightley
…	…	…
Internet's Own Boy	Knappenberger	Swartz
Internet's Own Boy	Knappenberger	Lessig
Internet's Own Boy	Knappenberger	Berners-Lee
…	…	…
Dogma	Smith	Damon
Dogma	Smith	Affleck

Venues

Cinema	**Address**	**Phone**
UFA	St. Petersburger Str. 24	4825825
Schauburg	Königsbrücker Str. 55	8032185
…	…	…

Program

Cinema	**Title**	**Time**
Schauburg	The Imitation Game	19:30
Schauburg	Dogma	20:45
UFA	The Imitation Game	22:45

9. Find the actors that are NOT cast in a movie by "Smith."

$$\exists y_T, y_D.\mathsf{Films}(y_T, y_D, x_A) \wedge$$
$$\forall x_T, x_D.(\mathsf{Films}(x_T, x_D, x_A) \rightarrow x_D \not\approx \texttt{"Smith"}[x_A]$$

10. Find all pairs of actors who act together in at least one film.

$$\exists y_T, y_D, y'_D.\ \mathsf{Films}(y_T, y_D, x_A) \wedge \mathsf{Films}(y_T, y'_D, x_{A'}) \wedge x_A \not\approx x_{A'}[x_A, x_{A'}]$$

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
$\rightsquigarrow$ 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

Features omitted on purpose: negation and universal quantifiers
$\rightsquigarrow$ the reason for this is query complexity

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
…	…	…

Exercise

Sudoku is a one-player puzzle game where one has to fill a grid with numbers. An example 4×4-Sudoku is as follows:

			3
			4
2			
3			

The grid has to be filled with numbers $\{1, 2, 3, 4\}$ such that every number occurs exactly once in each row, each column, and each 2×2-subgrid bordered in bold. For an arbitrary 4×4-Sudoku, specify a BCQ q and a database instance $\mathcal{J}$ such that $\mathcal{J} \models q$ if and only if the given Sudoku has a solution.

Exercise (cont'd)

For an arbitrary 4×4-Sudoku, specify a BCQ q and a database instance $\mathcal{J}$ such that $\mathcal{J} \models q$ iff the given Sudoku has a solution.

Solution.

- Let $\mathcal{J} = \{S(c_1^1, \ldots, c_1^4, \ldots, c_4^1, \ldots, c_4^4)\}$ such that, for all $1 \leq i \leq 4$ and all $1 \leq j \leq 4$, $c_i^j = k$ if position $\langle i, j \rangle$ contains the number k and c_i^j is the some constant e in all other cases.
- We set $\mathbf{y} = \langle y_1^1, \ldots, y_1^4, \ldots, y_4^1, \ldots, y_4^4 \rangle$, and let q be the query

$$\exists \mathbf{y}.\Big(\bigwedge_{i=1}^{4}\bigwedge_{j=1}^{4}(c_i^j \approx y_i^j \vee c_i^j \approx e) \wedge \bigwedge_{i\in\{1,\ldots,4\}}\bigwedge_{j\in\{1,\ldots,4\}\setminus\{i\}}(y_i^j \not\approx y_i^j \wedge y_j^i \not\approx y_j^i)$$

  The above query only considers columns and rows; extend to check quadrants!
- Then $\mathcal{J} \models q$ iff the Sudoku has a solution.

Complexity Review: Verifiers

Definition: Verifier

A TM $\mathcal{M}$ which halts on all inputs is called a verifier for a language **L** if

$$\mathbf{L} = \{w \mid \mathcal{M} \text{ accepts } (w\#c) \text{ for some string } c\}$$

The string c is called a certificate (or witness) for w.

Notation: $\#$ is a new separator symbol not used in words or certificates.

Complexity Review: Polynomial-Time Verifier

A TM $\mathcal{M}$ is a polynomial-time verifier for **L** if $\mathcal{M}$ is polynomially time bounded and

$$\mathbf{L} = \{w \mid \mathcal{M} \text{ accepts } (w\#c) \text{ for some string } c \text{ with } |c| \leq p(|w|)\}$$

for some fixed polynomial p.

Complexity Review: The Class NP

Intuitive definition: The class of problems for which a possible solution can be verified in P.

Definition: NP

The class of languages that have polynomial-time verifiers is called NP.

In other words: NP is the class of all languages **L** such that:

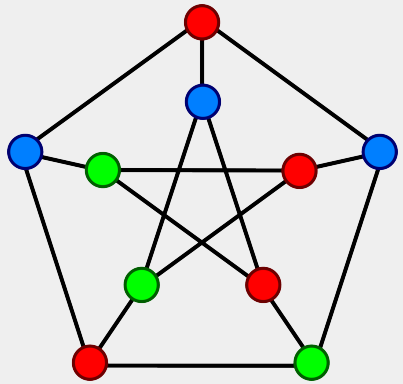
- for every $w \in \mathbf{L}$, there is a certificate $c_w \in \Sigma^*$, where
- the length of c_w is polynomial in the length of w, and
- the language $\{(w \# c_w) \mid w \in \mathbf{L}\}$ is in P

Complexity Review: NP Examples

Examples:

- Sudoku solvability (guess: filled-out grid)
- Composite (non-prime) number (guess: factorization)
- Propositional logic satisfiability (guess: satisfying assignment)
- Graph colourability (guess: coloured graph)

			3
			4
2			
3			



p	q	r	$p \rightarrow q$
f	f	f	w
f	w	f	w
w	f	f	f
w	w	f	w
f	f	w	w
f	w	w	w
w	f	w	f
w	w	w	w

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

$\rightsquigarrow$ 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.

$\rightsquigarrow$ Better than PSpace (presumably)

BCQ Entailment is in NP

Proof by showing that there is a poly-time verifier:

Theorem

BCQ query answering is in NP for combined complexity.

Consider the following BCQ and some database instance $\mathcal{I}$:

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

What is the certificate that we can use to verify that the query is entailed by the database $\mathcal{I}$?

7. Tractable Query Answering

How to reduce complexities?

NP-complete query complexity is still intractable!

Remark

Usually, we say that a problem is tractable if it can be solved in polynomial time.

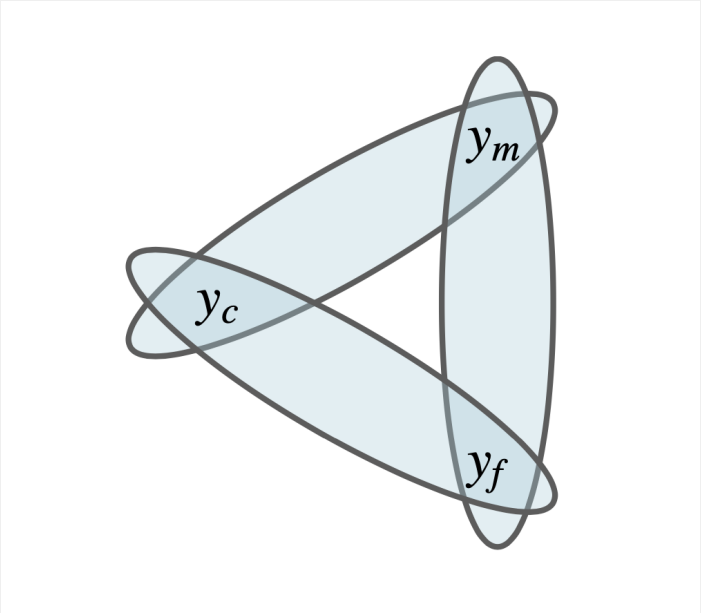
Can we do better? *Idea:*

- We have encoded 3-colourability to show NP-hardness
- Can we avoid hardness by restricting to certain cases?

Example: Cyclic CQs

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

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

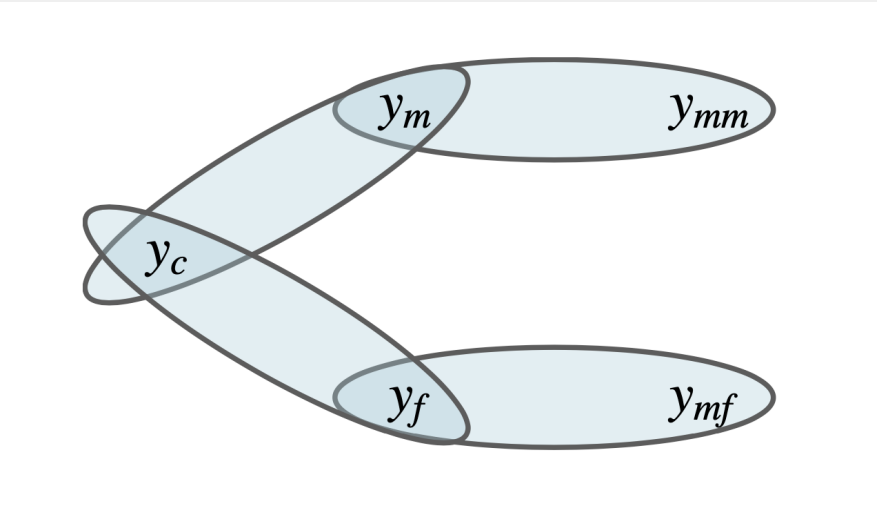


$\rightsquigarrow$ cyclic query

Example: Acyclic CQs

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

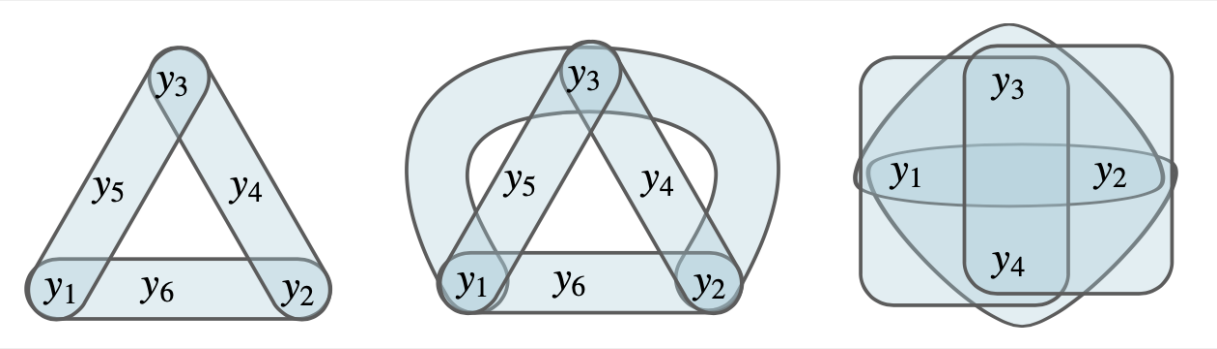
$$\exists y_c, y_m, y_f, y_{mm}, y_{mf}.\text{mother}(y_c, y_m) \wedge \text{father}(y_c, y_f) \wedge \text{married}(y_m, y_{mm}) \wedge \text{married}(y_{mf}, y_f)$$



$\rightsquigarrow$ acyclic query

Defining Acyclic Queries

Queries in general are hypergraphs
$\rightsquigarrow$ 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

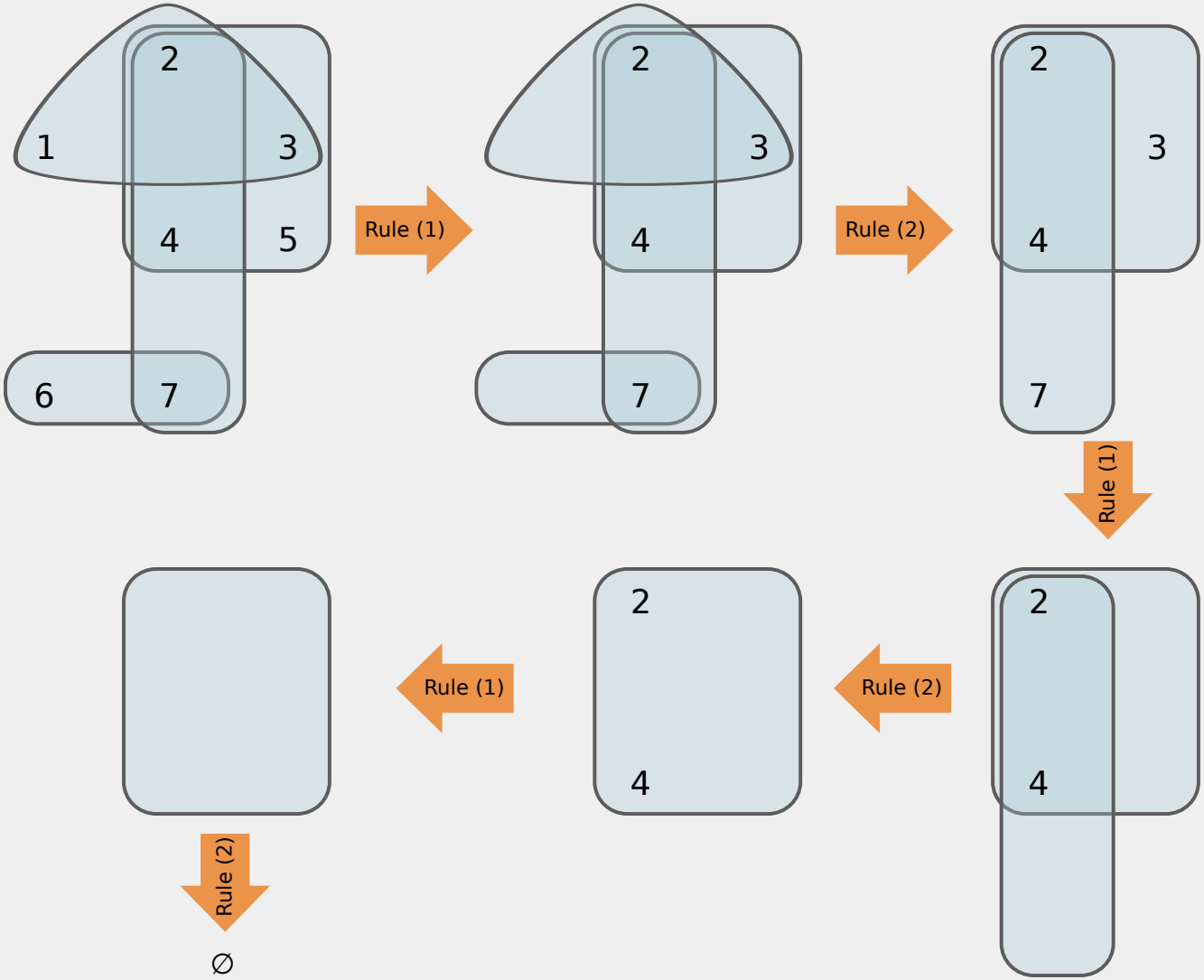
Apply the following simplification rules as long as possible:

(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

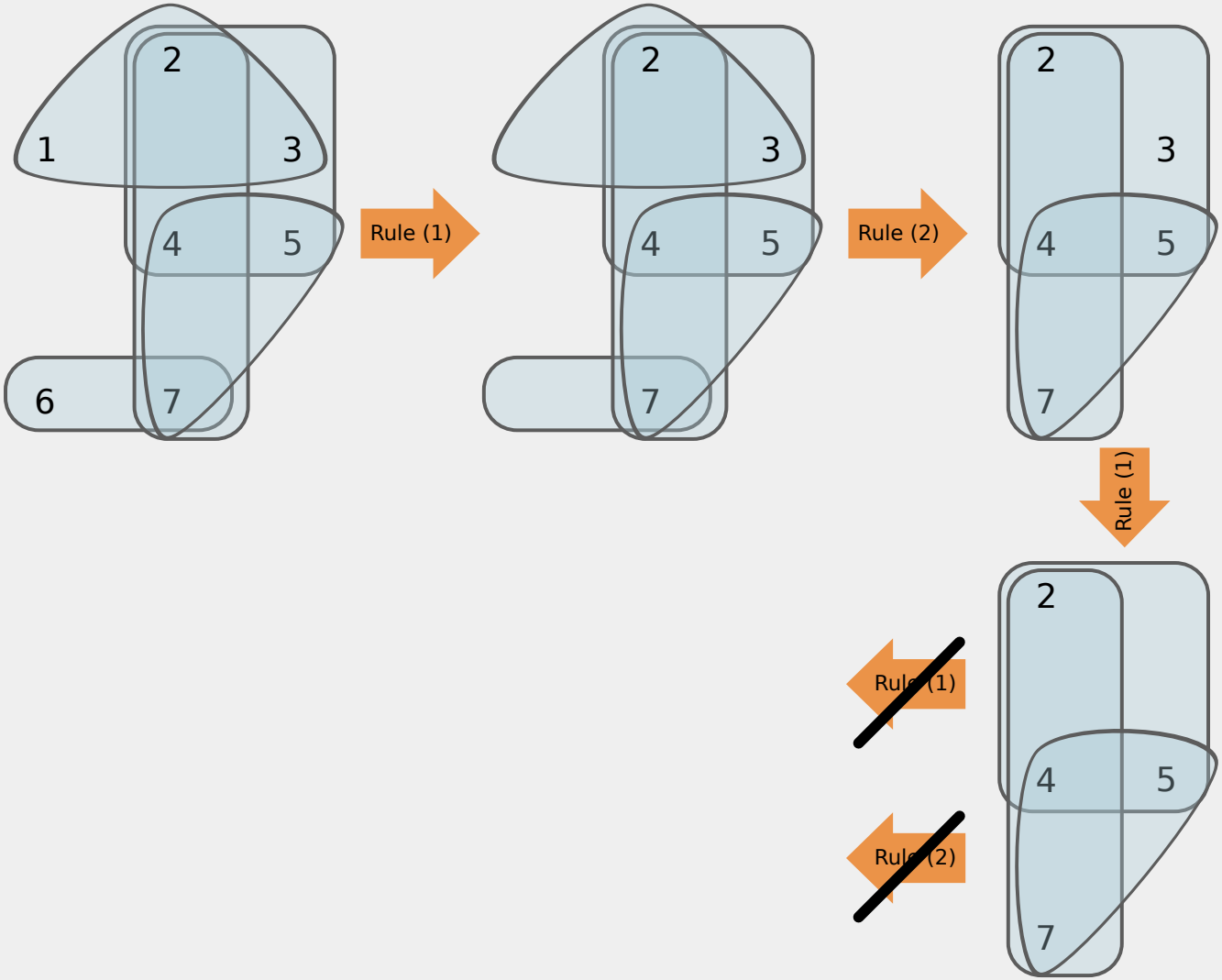
Definition

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



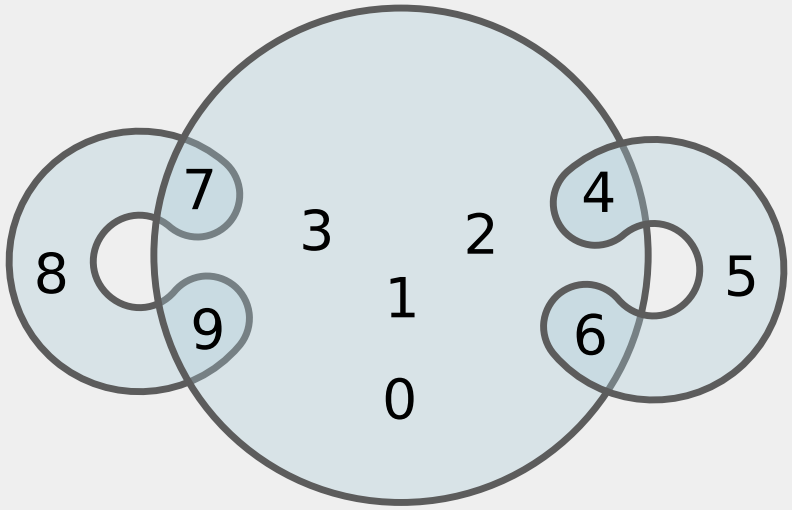
Alternative Version of GYO-Reduction

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.

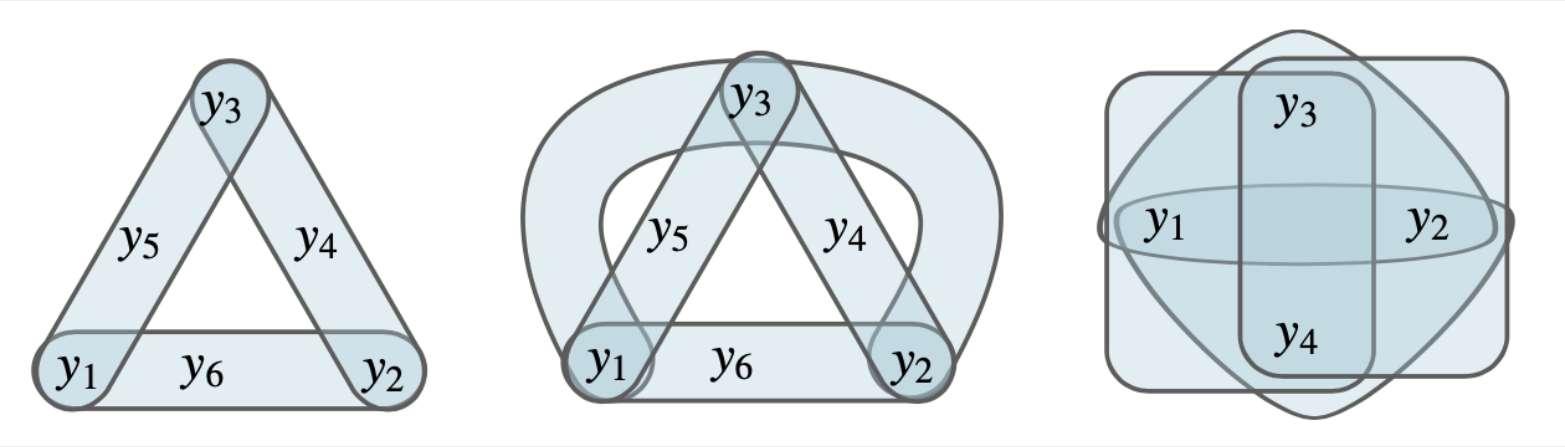
Example:



$\rightsquigarrow$ edges $\langle 4, 5, 6 \rangle$ and $\langle 7, 8, 9 \rangle$ are ears

Examples

Any ears?



GYO'-Reduction

Definition

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 $\langle \emptyset, \emptyset \rangle$

$\rightsquigarrow$ alternative characterization of acyclic hypergraphs

Exercise

Decide if the following conjunctive queries are tree queries by applying (one version of) the GYO algorithm.

1. $\exists x, y, z, v.\ r(x, y) \wedge r(y, z) \wedge r(z, v) \wedge s(x, y, z) \wedge s(y, z, v)$
2. $\exists x, y, z, u, v, w.\ r(x, y) \wedge s(x, z, v) \wedge r(u, z) \wedge t(x, v, u, w)$

Definition

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

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

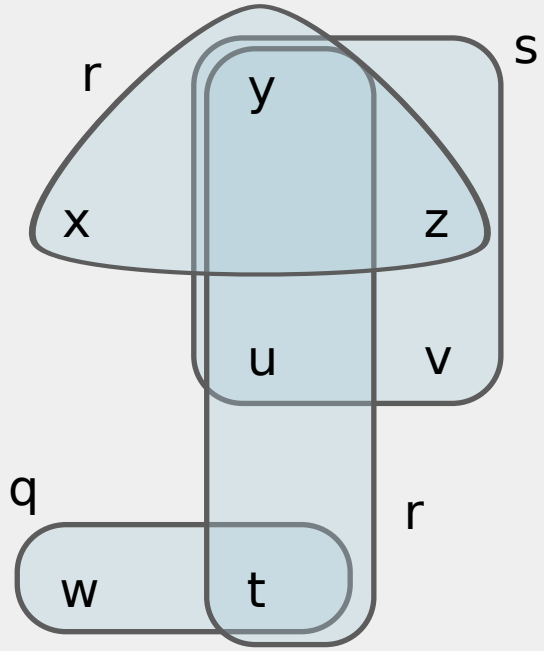
Definition

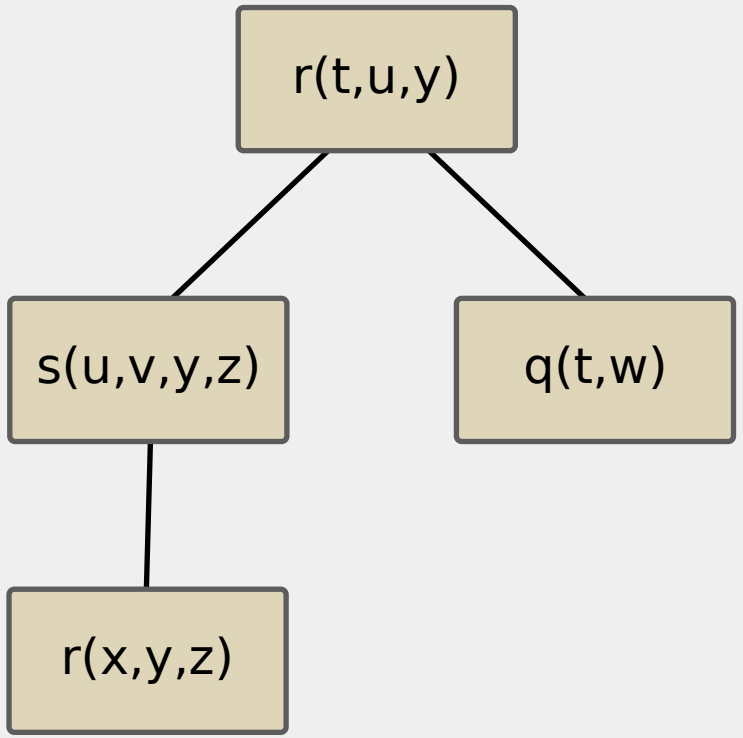
A join tree of a (B)CQ is an arrangement of its query atoms in a tree structure T, such that for each variable x, the atoms that refer to x are a connected subtree of T.

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

Example: Join Tree

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





Processing Join Trees Efficiently

Join trees can be processed in polynomial time

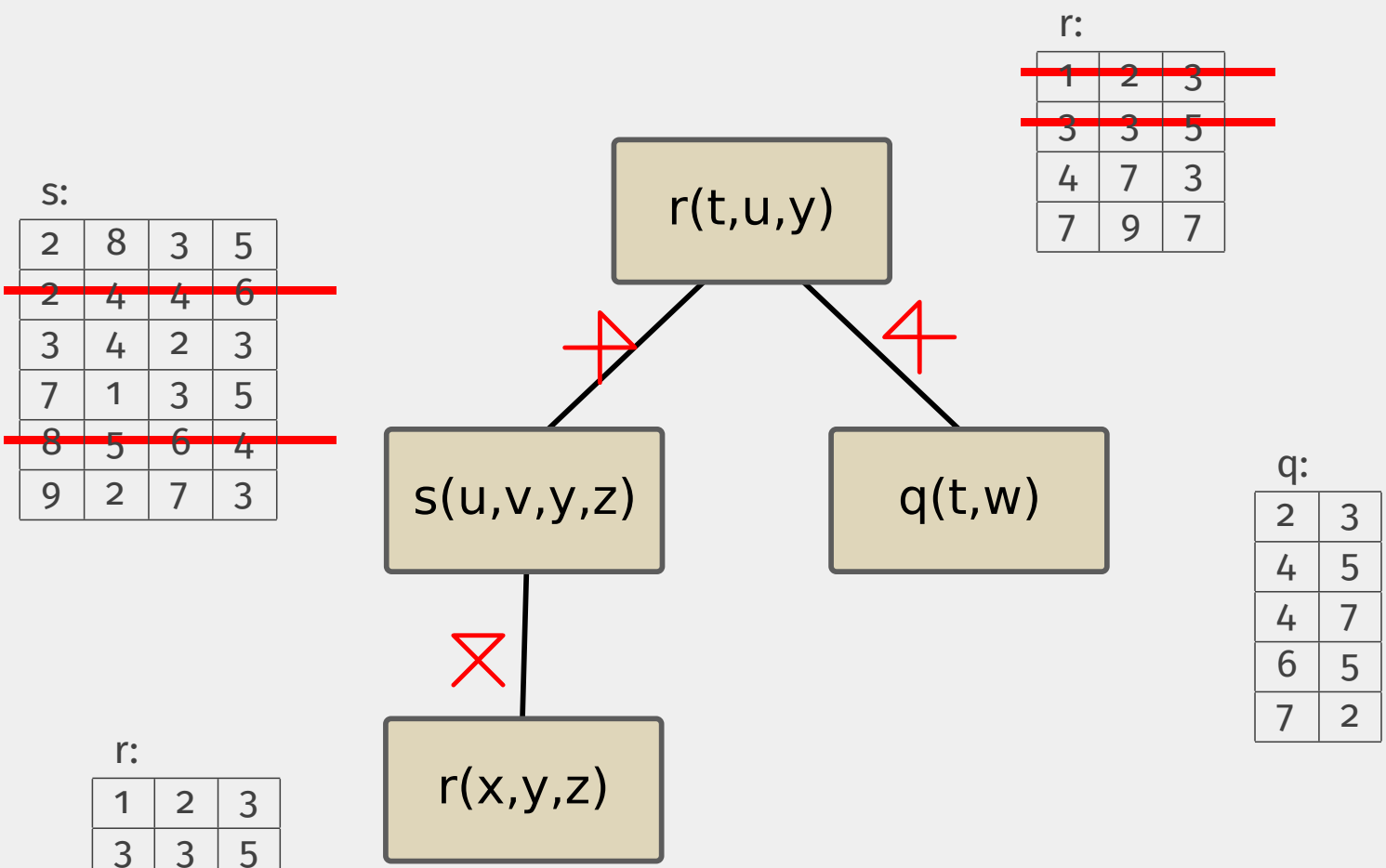
Key ingredient: the semijoin operation

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 be processed by computing semijoins bottom-up
$\rightsquigarrow$ Yannakakis' Algorithm

Yannakakis' Algorithm by Example



Yannakakis' Algorithm: Summary

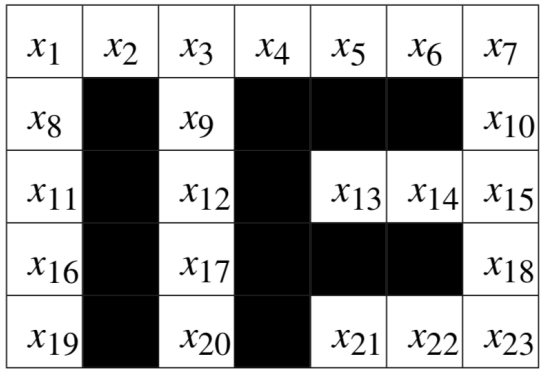
Polynomial time procedure for answering BCQs

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

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

Exercise: Yannakakis' Algorithm

Solve the following combinatorial crossword puzzle using Yannakakis' algorithm (in spirit). Specify the join tree that you are using.



1 hor.:

B	R	I	S	T	O	L
C	A	R	A	M	E	L
P	H	A	R	A	O	H
S	P	I	N	A	C	H
T	S	U	N	A	M	I

1 vert.:

C	L	E	A	R
H	U	M	A	N
P	E	A	C	E
S	H	A	R	K
T	I	G	E	R

3 vert.:

H	A	P	P	Y
I	N	F	E	R
L	A	B	O	R
L	A	T	E	R
U	N	T	I	L

7 vert.:

H	E	A	R	T
H	O	N	E	Y
I	R	O	N	Y
L	O	G	I	C
M	A	G	I	C

13 hor.:

A	N	D
C	A	T
D	I	M
L	A	G
W	I	N

21 hor.:

A	R	C
F	E	E
L	O	W
T	W	O
W	A	Y

Exercise: Acyclicity and Constants

How can we deal with BCQs that feature constants? E.g.,

$$\exists x, y, z.\text{mother}(x, y) \land \text{father}(x, z) \land \text{bornIn}(y, \texttt{"Montpellier"}) \land \text{bornIn}(z, \texttt{"Montpellier"})$$

Discussion

Is the above query acyclic? Can we solve it in polynomial time?

8. Query Optimisation

Static Query Optimisation

Can we optimise query execution without looking at the database?

Queries are logical formulas, so some things might follow …

Query equivalence:
Will the queries Q_1 and Q_2 return the same answers over any database?

- In symbols: $Q_1 \equiv Q_2$
- We have seen many examples of equivalent transformations in exercises
- Several uses for optimisation:
  - $\rightsquigarrow$ DBMS could run the “nicer” of two equivalent queries
  - $\rightsquigarrow$ DBMS could use cached results of one query for the other
  - $\rightsquigarrow$ Also applicable to equivalent subqueries

Static Query Optimisation (2)

Other things that could be useful:

- Query emptiness: Will query Q never have any results?

  $\rightsquigarrow$ Special equivalence with an "empty query"
  (e.g., $x \neq x$ or $R(x) \wedge \neg R(x)$)
  $\rightsquigarrow$ Empty (sub)queries could be answered immediately

- Query containment: Will the query Q_1 return a subset of the results of query Q_2? (in symbols: $Q_1 \sqsubseteq Q_2$)

  $\rightsquigarrow$ Generalisation of equivalence:
  $Q_1 \equiv Q_2$ if and only if $Q_1 \sqsubseteq Q_2$ and $Q_2 \sqsubseteq Q_1$

- Query minimisation: Given a query Q, can we find an equivalent query Q' that is "as simple as possible."

The Impossibility of FO Query Optimisation

Theorem: Boris Trakhtenbrot, 1950

Finite-model reasoning of first-order logic is undecidable.

Corollary

All of the following decision problems are undecidable:

- Query equivalence
- Query emptiness
- Query containment

$\rightsquigarrow$ "perfect" FO query optimisation is impossible

Optimisation for Conjunctive Queries

Optimisation is simpler for conjunctive queries

Example

Conjunctive query containment:

$$Q_1 : \quad \exists x, y, z.\ R(x, y) \wedge R(y, y) \wedge R(y, z)$$
$$Q_2 : \quad \exists u, v, w, t.\ R(u, v) \wedge R(v, w) \wedge R(w, t)$$

Q_1 find R-paths of length two with a loop in the middle
Q_2 find R-paths of length three

$\rightsquigarrow$ in a loop one can find paths of any length
$\rightsquigarrow Q_1 \sqsubseteq Q_2$

Deciding Conjunctive Query Containment

Consider conjunctive queries $Q_1[x_1, \ldots, x_n]$ and $Q_2[y_1, \ldots, y_n]$.

Definition

A query homomorphism from Q_2 to Q_1 is a mapping μ from terms (constants or variables) in Q_2 to terms in Q_1 such that:

- μ does not change constants, i.e., $\mu(c) = c$ for every constant c
- $x_i = \mu(y_i)$ for each $i = 1, \ldots, n$
- if Q_2 has a query atom $R(t_1, \ldots, t_m)$ then Q_1 has a query atom $R(\mu(t_1), \ldots, \mu(t_m))$

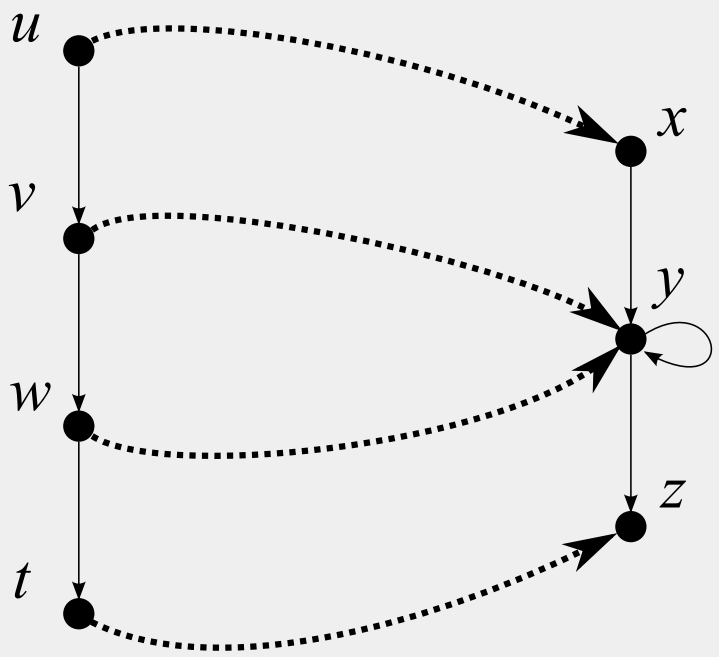
Homomorphism Theorem

$Q_1 \sqsubseteq Q_2$ if and only if there is a query homomorphism $Q_2 \to Q_1$.

Example

$$Q_1 : \quad \exists x, y, z.\ R(x, y) \wedge R(y, y) \wedge R(y, z)$$

$$Q_2 : \quad \exists u, v, w, t.\ R(u, v) \wedge R(v, w) \wedge R(w, t)$$



Implications of the Homomorphism Theorem

The proof has highlighted another useful fact:

The following two are equivalent:

- Finding a homomorphism from Q_2 to Q_1
- Finding a query result for Q_2 over $\mathcal{I}_1$

$\rightsquigarrow$ all complexity results for CQ query answering apply

Theorem

Deciding if $Q_1 \sqsubseteq Q_2$ is NP-complete.

Note that even in the NP-complete case the problem size is rather small (only queries, no databases)

Application: CQ Minimisation

Definition

A conjunctive query Q is minimal if:

- for all subqueries Q' of Q (that is, queries Q' that are obtained by dropping one or more atoms from Q),
- we find that $Q' \not\equiv Q$.

A minimal CQ is also called a core.

It is useful to minimise CQs to avoid unnecessary joins in query answering.

CQ Minimisation the Direct Way

A simple idea for minimising Q:

- Consider each atom of Q, one after the other
- Check if the subquery obtained by dropping this atom is contained in Q

  (Observe that the subquery always contains the original query.)
- If yes, delete the atom; continue with the next atom

Example query $Q[v, w]$

$\exists x, y, z.R(a, y) \wedge R(x, y) \wedge S(y, y) \wedge S(y, z) \wedge S(z, y) \wedge T(y, v) \wedge T(y, w)$

$\rightsquigarrow$ Simpler notation: write as set and mark answer variables

$\{R(a, y), R(x, y), S(y, y), S(y, z), S(z, y), T(y, \bar{v}), T(y, \bar{w})\}$

CQ Minimisation Example

$$\{R(a,y), R(x,y), S(y,y), S(y,z), S(z,y), T(y,\bar{v}), T(y,\bar{w})\}$$

Can we map the left side homomorphically to the right side?

$R(a,y)$	$R(a,y)$	Keep (cannot map constant a)
~~$R(x,y)$~~	~~$R(x,y)$~~	Drop; map $R(x,y)$ to $R(a,y)$
$S(y,y)$	$S(y,y)$	Keep (no other atom of form $S(t,t)$)
~~$S(y,z)$~~	~~$S(y,z)$~~	Drop; map $S(y,z)$ to $S(y,y)$
~~$S(z,y)$~~	~~$S(z,y)$~~	Drop; map $S(z,y)$ to $S(y,y)$
$T(y,\bar{v})$	$T(y,\bar{v})$	Keep (cannot map answer variable)
$T(y,\bar{w})$	$T(y,\bar{w})$	Keep (cannot map answer variable)

Core: $\exists y.R(a,y) \wedge S(y,y) \wedge T(y,v) \wedge T(y,w)$