MAI4CAREU

Master programmes in Artificial Intelligence 4 Careers in Europe

University of Cyprus

MAI649: PRINCIPLES OF ONTOLOGICAL DATABASES

Relational Model

Andreas Pieris

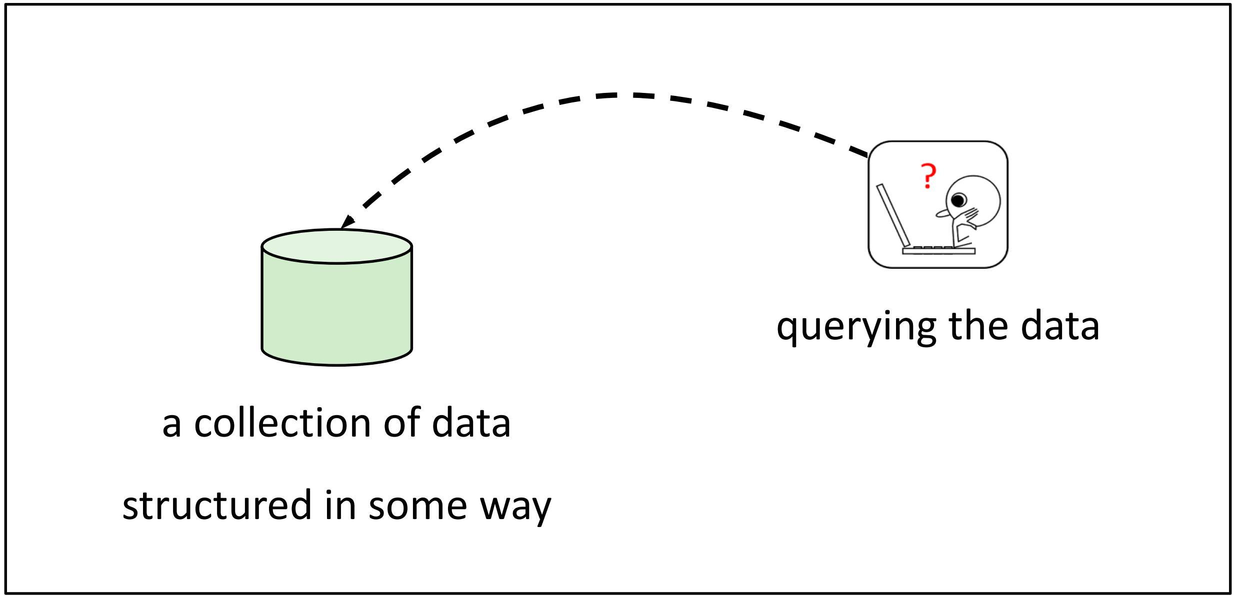
Spring 2022-2023

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

Learning Outcomes

- Abstract data and queries from their physical implementation, and formalize them in a rigorous way - relational model
- Analyze the complexity of evaluating relational (algebra and calculus) queries
- Analyze the complexity of static analysis of relational (algebra and calculus) queries

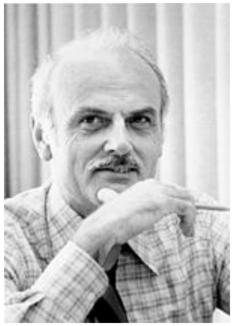


Data Model

mathematical abstraction for structuring the data

independent from the physical implementation

Relational Model



Edgar F. Codd
(1923 - 2003)
Turing Award 1981

- Many ad hoc models before 1970
  - Hard to work with
  - Hard to reason about
- **1970: Relational Model by Edgar Frank Codd**
  - Data are stored in relations (or tables)
  - Queried using a declarative language
  - DBMS converts declarative queries into procedural queries that are optimized and executed
- Key Advantages
  - Simple and clean mathematical model (based on logic)
  - Separation of declarative and procedural

Relational Databases

Database Schema: a finite set of **relation names** together with their **attributes names**

Flight	origin:string	destination:string	airline:string

Airport	code:string	city:string

+

Database Instance: data conforming to the schema

VIE	LHR	BA
LHR	EDI	BA
LGW	GLA	U2
LCA	VIE	OS

VIE	Vienna
LHR	London
LGW	London
LGW	Larnaca
GLA	Glasgow
EDI	Edinburgh

Relational Databases

Flight	origin:string	destination:string	airline:string
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code:string	city:string
	VIE	Vienna
	LHR	London
	LGW	London
	LGW	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

- Ignore attribute types - data elements are coming from a countably infinite set **Const** (constant values)
- A relational database is a *finite* set of relational atoms

Relational Databases

$$\left\{ \begin{array}{ll} & \text{Airport(VIE,Vienna),} \\ \text{Flight(VIE,LHR,BA),} & \text{Airport(LHR,London),} \\ \text{Flight(LHR,EDI,BA),} & \text{Airport(LGW,London),} \\ \text{Flight(LGW,GLA,U2),} & \text{Airport(LGW,Larnaca),} \\ \text{Flight(LCA,VIE,OS),} & \text{Airport(GLA,Glasgow),} \\ & \text{Airport(EDI,Edinburgh)} \end{array} \right\}$$

…we will keep using the table representation without the attribute types

Querying: Relational Algebra

List all the airlines

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

Querying: Relational Algebra

List all the airlines

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

{BA, U2, OS}

π_{airline} Flight

Querying: Relational Algebra

List the codes of the airports in London

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

Querying: Relational Algebra

List the codes of the airports in London

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

{LHR, LGW}

π_{code} $(\sigma_{city=\text{'London'}}$ Airport)

Querying: Relational Algebra

List the airlines that fly directly from London to Glasgow

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

Querying: Relational Algebra

List the airlines that fly directly from London to Glasgow

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

$$\pi_{\text{airline}} \; ((\text{Flight} \;\bowtie_{\text{origin=code}} (\sigma_{\text{city='London'}} \; \text{Airport})) \bowtie_{\text{destination=code}} (\sigma_{\text{city='Glasgow'}} \; \text{Airport}))$$

Querying: Relational Algebra

$$\pi_{\text{airline}} \left(\left(\text{Flight} \bowtie_{\text{origin=code}} \underbrace{\left(\sigma_{\text{city='London'}} \text{Airport} \right)}_{} \right) \bowtie_{\text{destination=code}} \underbrace{\left(\sigma_{\text{city='Glasgow'}} \text{Airport} \right)}_{} \right)$$

code	**city**
LHR	London
LGW	London

code	**city**
GLA	Glasgow

Querying: Relational Algebra

$$\pi_{airline}\ ((\text{Flight}\ \bowtie_{origin=code}\ \underbrace{(\sigma_{city=\text{'London'}}\ \text{Airport}))}\ \bowtie_{destination=code}\ \underbrace{(\sigma_{city=\text{'Glasgow'}}\ \text{Airport}))}$$

code	city
LHR	London
LGW	London

code	city
GLA	Glasgow

origin	destination	airline	code	city
LHR	EDI	BA	LHR	London
LGW	GLA	U2	LGW	London

Querying: Relational Algebra

$$\pi_{\text{airline}} \left(\left(\text{Flight} \bowtie_{\text{origin=code}} \underbrace{\left(\sigma_{\text{city='London'}} \text{Airport} \right)} \right) \bowtie_{\text{destination=code}} \underbrace{\left(\sigma_{\text{city='Glasgow'}} \text{Airport} \right)} \right)$$

code	city
LHR	London
LGW	London

code	city
GLA	Glasgow

origin	destination	airline	code	city
LHR	EDI	BA	LHR	London
LGW	GLA	U2	LGW	London

origin	destination	airline	code	city	code	city
LGW	GLA	U2	LGW	London	GLA	Glasgow

Querying: Relational Algebra

$\pi_{airline}\ ((\text{Flight}\ \bowtie_{origin=code}\ (\sigma_{city=\text{'London'}}\ \text{Airport}))\ \bowtie_{destination=code}\ (\sigma_{city=\text{'Glasgow'}}\ \text{Airport}))$

code	city
LHR	London
LGW	London

code	city
GLA	Glasgow

origin	destination	airline	code	city
LHR	EDI	BA	LHR	London
LGW	GLA	U2	LGW	London

origin	destination	airline	code	city	code	city
LGW	GLA	U2	LGW	London	GLA	Glasgow

airline
U2

Relational Algebra

- **Selection: σ**
- **Projection: π**
- **Cross product:** $\times$
- Natural join: ⋈
- **Rename: ρ**
- **Difference:** $\setminus$
- **Union:** ∪
- Intersection: ∩

in bold are the primitive operators

Formal definition can be found in Chapter 4 of PDB

Querying: Domain Relational Calculus

List all the airlines

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

Querying: Domain Relational Calculus

List all the airlines

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

{BA, U2, OS}

$\{z \mid \exists x \exists y \text{ Flight}(x,y,z)\}$

Querying: Domain Relational Calculus

List the codes of the airports in London

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

Querying: Domain Relational Calculus

List the codes of the airports in London

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

{LHR, LGW}

$\{x \mid \exists y\ Airport(x,y) \ \land\ y = London\}$

Querying: Domain Relational Calculus

List the airlines that fly directly from London to Glasgow

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

Querying: Domain Relational Calculus

List the airlines that fly directly from London to Glasgow

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	$\boxed{\text{U2}}$
	LCA	VIE	OS

⇩

{U2}

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

$\{z \mid \exists x \exists y \exists u \exists v\ \text{Airport}(x,u) \wedge u = \text{London} \wedge \text{Airport}(y,v) \wedge v = \text{Glasgow} \wedge \text{Flight}(x,y,z)\}$

Domain Relational Calculus

(see Chapter 4 of PDB & additional material on relational calculus)

$\{x_1,\dots,x_k \mid \phi\}$

first-order formula with

free variables $\{x_1,\dots,x_k\}$

But, we can express "**problematic**" queries, i.e., depend on the domain

$\{x \mid \forall y\, R(x,y)\}$ $\{x \mid \neg R(x)\}$ $\{x,y \mid R(x) \vee R(y)\}$

Domain Relational Calculus

(see Chapter 4 of PDB & additional material on relational calculus)

$\{x_1,\ldots,x_k \mid \phi\}$

first-order formula with

free variables $\{x_1,\ldots,x_k\}$

But, we can express "**problematic**" queries, i.e., depend on the domain

$\{x \mid \forall y\, R(x,y)\}$ $\{x \mid \neg R(x)\}$ $\{x,y \mid R(x) \vee R(y)\}$

domain = $\{1,2,3\}$

D = $\{R(1,1), R(1,2)\}$

Ans = { }

Domain Relational Calculus

(see Chapter 4 of PDB & additional material on relational calculus)

$\{x_1,\ldots,x_k \mid \phi\}$

first-order formula with

free variables $\{x_1,\ldots,x_k\}$

But, we can express "**problematic**" queries, i.e., depend on the domain

$\{x \mid \forall y\, R(x,y)\}$ $\{x \mid \neg R(x)\}$ $\{x,y \mid R(x) \vee R(y)\}$

domain = $\{1,2\}$

$D = \{R(1,1), R(1,2)\}$

Ans = $\{1\}$

Domain Relational Calculus

(see Chapter 4 of PDB & additional material on relational calculus)

$\{x_1,\ldots,x_k \mid \phi\}$

first-order formula with

free variables $\{x_1,\ldots,x_k\}$

But, we can express "**problematic**" queries, i.e., depend on the domain

$\{x \mid \forall y\, R(x,y)\}$ $\{x \mid \neg R(x)\}$ $\{x,y \mid R(x) \vee R(y)\}$

…thus, we adopt the active domain semantics - quantified variables range over the active domain, i.e., the constants occurring in the input database

Algebra = Calculus

(see Chapter 6 of PDB)

A fundamental theorem (assuming the active domain semantics):

Theorem: The following query languages are equally expressive

- Relational Algebra (**RA**)
- Domain Relational Calculus (**DRC**)
- Tuple Relational Calculus (**TRC**)

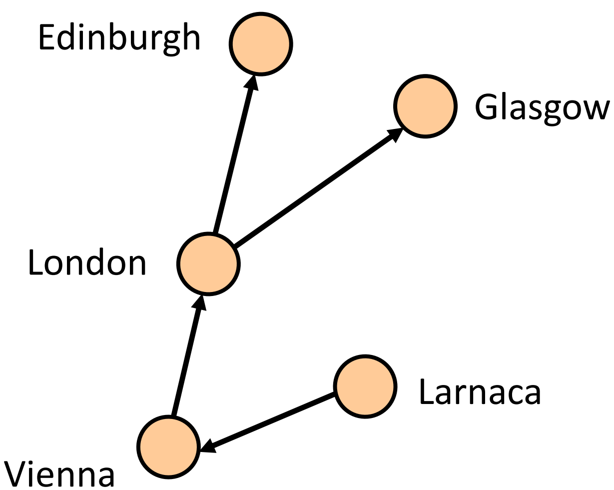
Note: Tuple relational calculus is the declarative language introduce by Codd. Domain relational calculus has been introduced later as a formalism closer to first-order logic

Quiz!

Is Glasgow reachable from Vienna?

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh



$$\{ \mid \exists x \exists y \exists z \exists w \exists v \; \mathrm{Airport}(x,\mathrm{Vienna}) \wedge \mathrm{Airport}(y,\mathrm{Glasgow}) \wedge \mathrm{Flight}(x,z,w) \wedge \mathrm{Flight}(z,y,v)\}$$

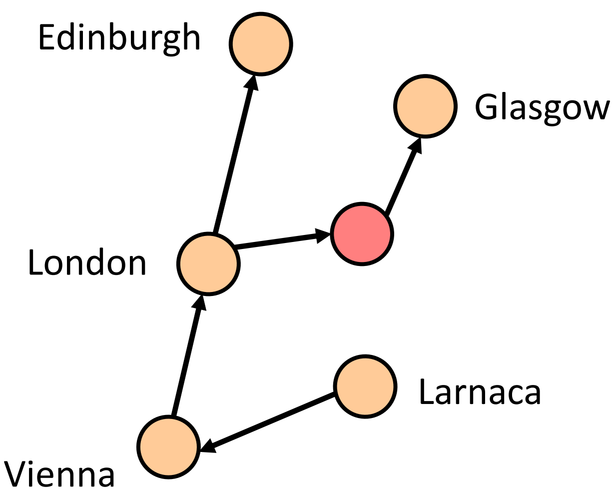
YES

Quiz!

Is Glasgow reachable from Vienna?

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh



$\{ \mid \exists x \exists y \exists z \exists w \exists v \;\; \mathrm{Airport}(x,\mathrm{Vienna}) \;\wedge\; \mathrm{Airport}(y,\mathrm{Glasgow}) \;\wedge\; \mathrm{Flight}(x,z,w) \;\wedge\; \mathrm{Flight}(z,y,v)\}$

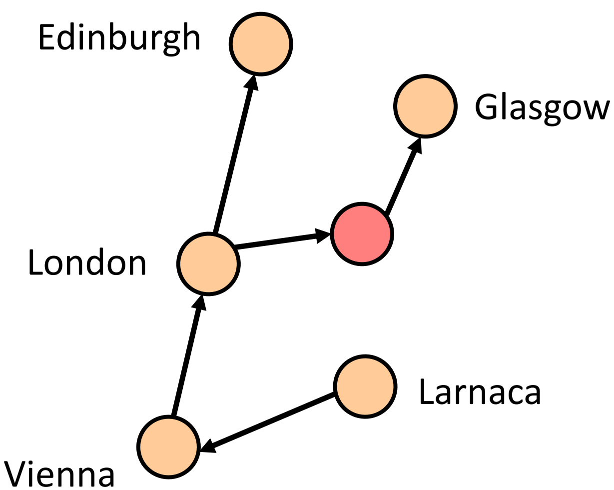
NO

Quiz!

Is Glasgow reachable from Vienna?

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh



$\{ \mid \exists x \exists y \exists z \exists w \exists v \;\; \text{Airport}(x,\text{Vienna}) \;\wedge\; \text{Airport}(y,\text{Glasgow}) \;\wedge$

$\exists z_1 \exists w_1 \qquad \text{Flight}(x,z,w) \qquad \text{Flight}(z,y,v)\}$

$\wedge\; \text{Flight}(z,z_1,w_1) \;\wedge$

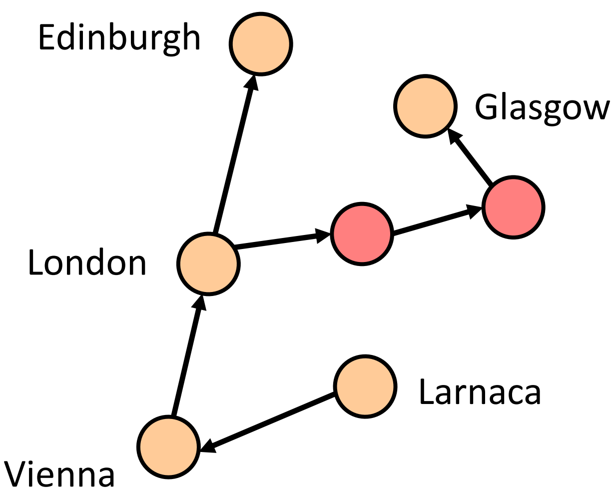
YES

Quiz!

Is Glasgow reachable from Vienna?

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh



$\{ \mid \exists x \exists y \exists z \exists w \exists v\ \text{Airport}(x,\text{Vienna}) \wedge \text{Airport}(y,\text{Glasgow}) \wedge$

$\exists z_1 \exists w_1 \quad \text{Flight}(x,z,w) \quad \text{Flight}(z,y,v)\}$

$\wedge\ \text{Flight}(z,z_1,w_1) \ \wedge$

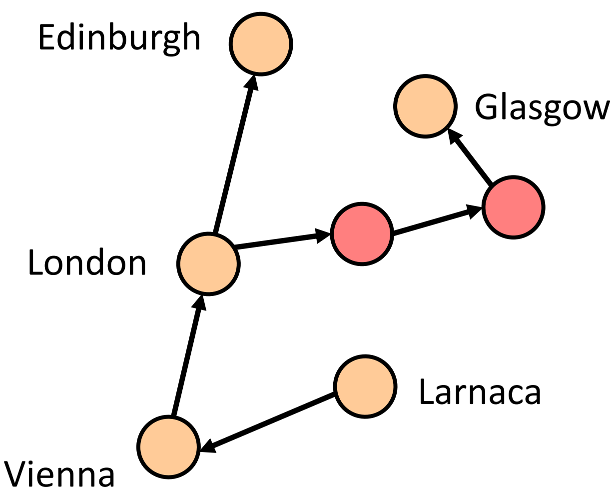
NO

Quiz!

Is Glasgow reachable from Vienna?

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh



Recursive query - not expressible in **RA/DRC/TRC**

(unless we bound the number of intermediate stops)

Complexity of Query Languages

- The goal is to understand the complexity of evaluating a query over a database
- Our main technical tool is complexity theory - see additional material
- What to measure? Queries may have a large output, and it would be unfair to count the output as “complexity”
- We therefore consider the following decision problems:
  - Query Output Tuple (QOT)
  - Boolean Query Evaluation (BQE)

Complexity of Query Languages

Some useful notation:

- Given a database D, and a query Q, Q(D) is the answer to Q over D
- **adom**(D) is the active domain of D - the constants occurring in D
- We write Q/k for the fact that the arity of Q is $k \geq 0$

L is some query language; for example, **RA**, **DRC**, etc. - we will see more query languages

QOT(**L**)

Input: a database D, a query Q/k ∈ **L**, a tuple of constants **t** ∈ **adom**(D)^k

Question: t ∈ Q(D)?

Complexity of Query Languages

Some useful notation:

- Given a database D, and a query Q, Q(D) is the answer to Q over D
- **adom**(D) is the active domain of D - the constants occurring in D
- We write Q/k for the fact that the arity of Q is $k \geq 0$

L is some query language; for example, **RA**, **DRC**, etc. - we will see more query languages

BQE(**L**)

Input: a database D, a Boolean query Q ∈ **L**

Question: is Q(D) non-empty?

Complexity of Query Languages

QOT(**L**)

Input: a database D, a query Q/k ∈ **L**, a tuple of constants **t** ∈ **adom**(D)^k

Question: **t** ∈ Q(D)?

BQE(**L**)

Input: a database D, a Boolean query Q ∈ **L**

Question: is Q(D) non-empty?

Theorem: QOT(**L**) $\equiv_L$ BQE(**L**), where **L** ∈ {**RA**, **DRC**, **TRC**}

($\equiv_L$ means logspace-equivalent)

Complexity of Query Languages

(let us show this for domain relational calculus)

Theorem: QOT(**DRC**) $\equiv_L$ BQE(**DRC**)

Proof: ($\leq_L$) Consider a database D, a k-ary query $Q = \{x_1,\ldots,x_k \mid \phi\}$, and a tuple $(t_1,\ldots,t_k)$

Let $Q_{bool} = \{ \mid \exists x_1 \cdots \exists x_k \, (\phi \wedge x_1 = t_1 \wedge x_2 = t_2 \wedge \cdots \wedge x_k = t_k) \}$

Clearly, $(t_1,\ldots,t_k) \in Q(D)$ iff $Q_{bool}(D)$ is non-empty

($\geq_L$) Trivial - a Boolean domain RC query is a domain RC query

Complexity Measures

- Combined complexity - both D and Q are part of the input

- Data complexity - input D, fixed Q

BQE[Q](**L**)

Input: a database D

Question: is Q(D) non-empty?

Complexity of **RA**, **DRC**, **TRC**

Theorem: For **L** ∈ {**RA**, **DRC**, **TRC**} the following hold:

- BQE(**L**) is PSPACE-complete (combined complexity)
- BQE[Q](**L**) is in LOGSPACE, for a fixed query Q ∈ **L** (data complexity)

Proof hints:

- Recursive algorithm that uses polynomial space in Q and logarithmic space in D
- Reduction from QSAT (a standard PSPACE-hard problem)

Evaluating (Boolean) DRC Queries

Evaluation(D,ϕ) - for brevity we write ϕ instead of $\{ \mid \phi \}$

- **If** $\phi = R(t_1,\ldots,t_k)$, **then** YES iff $R(t_1,\ldots,t_k) \in D$
- **If** $\phi = \psi_1 \land \psi_2$, **then** YES iff Evaluation(D,ψ_1) = YES and Evaluation(D,ψ_2) = YES
- **If** $\phi = \neg\psi$, **then** NO iff Evaluation(D,ψ) = YES
- **If** $\phi = \exists x\ \psi(x)$, **then** YES iff for some $t \in$ **adom**(D), Evaluation(D,$\psi(t)$) = YES

$$\psi_1 \lor \psi_2 \equiv \neg\neg(\psi_1 \lor \psi_2) \equiv \neg(\neg\psi_1 \land \neg\psi_2)$$

$$\forall x\ \psi(x) \equiv \neg\neg(\forall x\ \psi(x)) \equiv \neg(\exists x\ \neg\psi(x))$$

Evaluating (Boolean) DRC Queries

Lemma: It holds that

- Evaluation(D,ϕ) always terminates - *this is trivial*
- Evaluation(D,ϕ) = YES iff $Q(D)$ is non-empty, where $Q = \{ \mid \phi\}$ - *trivial since it simply implements the semantics*
- Evaluation(D,ϕ) uses $O(||\phi||^2 \cdot \log ||D||)$ space

Proof idea:

- It is clear that the recursion depth is $O(||\phi||)$
- We can show by induction on the structure of ϕ that each recursive call uses space $O(||\phi|| \cdot \log ||D||)$. This relies on an encoding of the database that allows us to check whether $R(t_1,\ldots,t_k) \in D$ using space $O(||\phi|| \cdot \log ||D||)$
- Consequently, the overall space used is $O(||\phi||^2 \cdot \log ||D||)$

Complexity of **RA**, **DRC**, **TRC**

Theorem: For **L** ∈ {**RA**, **DRC**, **TRC**} the following hold:

- BQE(**L**) is PSPACE-complete (combined complexity)
- BQE[Q](**L**) is in LOGSPACE, for a fixed query Q ∈ **L** (data complexity)

Proof hints:

- Recursive algorithm that uses polynomial space in Q and logarithmic space in D
- Reduction from QSAT (a standard PSPACE-hard problem)

Other Important Algorithmic Problems

SAT(**L**)

Input: a query $Q \in \mathbf{L}$

Question: is there a database D such that $Q(D)$ is non-empty?

EQUIV(**L**)

Input: two queries $Q_1 \in \mathbf{L}$ and $Q_2 \in \mathbf{L}$

Question: $Q_1 \equiv Q_2$? or $Q_1(D) = Q_2(D)$ for every database D?

CONT(**L**)

Input: two queries $Q_1 \in \mathbf{L}$ and $Q_2 \in \mathbf{L}$

Question: $Q_1 \subseteq Q_2$? or $Q_1(D) \subseteq Q_2(D)$ for every database D?

Other Important Algorithmic Problems

SAT(**L**)

Input: a query Q ∈ **L**

Question: is there a (finite) database D such that Q(D) is non-empty?

EQUIV(**L**)

Input: two q

Question: $Q_1 \equiv Q_2$? or $Q_1(D) = Q_2(D)$ for every (finite) database D?

these problems are important for optimization purposes

CONT(**L**)

Input: two queries Q_1 ∈ **L** and Q_2 ∈ **L**

Question: $Q_1 \subseteq Q_2$? or $Q_1(D) \subseteq Q_2(D)$ for every (finite) database D?

Other Important Algorithmic Problems

SAT(**L**) **Input:** a query Q ∈ **L** **Question:** is there a database D such that Q(D) is non-empty?

- If the answer is no, then the input query Q makes no sense
- Query evaluation becomes trivial - the answer is always NO!

Other Important Algorithmic Problems

EQUIV(**L**)

Input: two queries $Q_1 \in \mathbf{L}$ and $Q_2 \in \mathbf{L}$

Question: $Q_1 \equiv Q_2$? or $Q_1(D) = Q_2(D)$ for every database D?

- Replace a query Q_1 with a query Q_2 that is easier to evaluate
- But, we have to be sure that $Q_1(D) = Q_2(D)$ for every database D

Other Important Algorithmic Problems

CONT(**L**)

Input: two queries $Q_1 \in \mathbf{L}$ and $Q_2 \in \mathbf{L}$

Question: $Q_1 \subseteq Q_2$? or $Q_1(D) \subseteq Q_2(D)$ for every database D?

- Approximate a query Q with a query Q' that is easier to evaluate
- But, we have to be sure that $Q'(D) \subseteq Q(D)$ for every database D
- Moreover, equivalence boils down to two containment checks

SAT is Undecidable

Theorem: For **L** ∈ {**RA**, **DRC**, **TRC**}, SAT(**L**) is undecidable

Proof hint: By reduction from the halting problem.

Given a Turing machine M, we can construct a query Q_M ∈ **L** such that:

M halts on the empty string iff there exists a database D such that Q(D) is non-empty

Note: Actually, this result goes back to the 1950 when

Boris A. Trakhtenbrot proved that the problem of deciding

whether a first-order sentence has a finite model is undecidable

EQUIV and CONT are Undecidable

An easy consequence of the fact that SAT is undecidable is that:

Theorem: For **L** ∈ {**RA**, **DRC**, **TRC**}, EQUIV(**L**) and CONT(**L**) are undecidable

Proof: By reduction from the complement of SAT(**L**)

- Consider a query Q ∈ **L** - i.e., an instance of SAT(**L**)
- Let Q' be a query that is unsatisfiable, i.e., Q'(D) is empty for every D
- For example, when **L** = **DRC**, Q' can be the query $\{ \mid \exists x\, R(x) \land \neg R(x)\}$
- Clearly, Q is unsatisfiable iff Q ≡ Q' (or even Q ⊆ Q')

Recap

- The main languages for querying relational databases are:
  - Relational Algebra (**RA**)
  - Domain Relational Calcuclus (**DRC**)
  - Tuple Relational Calculus (**TRC**)

$\boxed{\textbf{RA = DRC = TRC}}$

(under the active domain semantics)

- Evaluation is decidable, and highly tractable in data complexity
  - **Foundations of the database industry**
  - The core of SQL is equally expressive to **RA/DRC/TRC**

- Satisfiability, equivalence and containment are undecidable
  - **Perfect query optimization is impossible**

MAI4CAREU | Master programmes in Artificial Intelligence 4 Careers in Europe

University of Cyprus

MAI649: PRINCIPLES OF ONTOLOGICAL DATABASES

Thank You!

Andreas Pieris

Spring 2022-2023

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

MAI4CAREU | Master programmes in Artificial Intelligence 4 Careers in Europe

University of Cyprus

MAI649: PRINCIPLES OF ONTOLOGICAL DATABASES

Conjunctive Queries

Andreas Pieris

Spring 2022-2023

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

Learning Outcomes

- Syntax and semantics of conjunctive queries (a core fragment of relational calculus)

- Analyze the complexity of evaluating conjunctive queries

- Analyze the complexity of static analysis of conjunctive queries

- Minimization of conjunctive queries

So far

- The main languages for querying relational databases are:
  - Relational Algebra (**RA**)
  - Domain Relational Calcuclus (**DRC**)
  - Tuple Relational Calculus (**TRC**)

RA = DRC = TRC

(under the active domain semantics)

- Evaluation is decidable, and highly tractable in data complexity
  - **Foundations of the database industry**
  - The core of SQL is equally expressive to **RA/DRC/TRC**

- Satisfiability, equivalence and containment are undecidable
  - **Perfect query optimization is impossible**

A Crucial Question

Are there interesting sublanguages of **RA/DRC/TRC** for which perfect query optimization is possible?

Conjunctive Queries

= $\{\sigma,\pi,\bowtie\}$-fragment of relational algebra

= relational calculus without $\neg$, $\forall$, $\vee$

= simple SELECT-FROM-WHERE SQL queries
(only AND and equality in the WHERE clause)

Syntax of Conjunctive Queries (**CQ**)

$$Q(\mathbf{x}) \;:=\; \exists \mathbf{y}\,(R_1(\mathbf{v_1}) \;\wedge\; \cdots \;\wedge\; R_m(\mathbf{v_m}))$$

- $R_1,\ldots,R_m$ are relations
- $\mathbf{x}$, $\mathbf{y}$, $\mathbf{v_1},\ldots,\mathbf{v_m}$ are tuples of variables
- each variable mentioned in $\mathbf{v_i}$ appears either in $\mathbf{x}$ or $\mathbf{y}$
- the variables in $\mathbf{x}$ are free called distinguished or output variables

It is very convenient to see conjunctive queries as rule-based queries of the form

$$Q(\mathbf{x}) \;\text{:-}\; \underbrace{R_1(\mathbf{v_1}),\ldots,R_m(\mathbf{v_m})}$$

this is called the body of Q that can be seen as a set of atoms

Conjunctive Queries: Example 1

List all the airlines

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

⇩

{BA, U2, OS}

π_{airline} Flight

Q(z) :- Flight(x,y,z)

{z | ∃x∃y Flight(x,y,z)}

Conjunctive Queries: Example 2

List the codes of the airports in London

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

{LHR, LGW}

π_{code} ($\sigma_{city=\text{'London'}}$ Airport)

{x | ∃y Airport(x,y) ∧ y = London}

Q(x) :- Airport(x,y), y = London

Conjunctive Queries: Example 2

List the codes of the airports in London

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

⇩

{LHR, LGW}

$\pi_{code}\ (\sigma_{city=\text{'London'}}\ \text{Airport})$

$\{x \mid \exists y\ \text{Airport}(x,y)\ \wedge\ y = \text{London}\}$

Q(x) :- Airport(x,London)

Conjunctive Queries: Example 3

List the airlines that fly directly from London to Glasgow

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

$\Downarrow$

{U2}

$$\pi_{\text{airline}} \; ((\text{Flight} \; \bowtie_{\text{origin=code}} \; (\sigma_{\text{city='London'}} \; \text{Airport})) \bowtie_{\text{destination=code}} \; (\sigma_{\text{city='Glasgow'}} \; \text{Airport}))$$

$$\{z \mid \exists x \exists y \exists u \exists v \; \text{Airport}(x,u) \; \wedge \; u = \text{London} \; \wedge \; \text{Airport}(y,v) \; \wedge \; v = \text{Glasgow} \; \wedge \; \text{Flight}(x,y,z)\}$$

Conjunctive Queries: Example 3

List the airlines that fly directly from London to Glasgow

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

⇩

{U2}

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

Q(z) :- Airport(x,London), Airport(y,Glasgow), Flight(x,y,z)

Pattern Matching Problem

List the airlines that fly directly from London to Glasgow

$$\left\{ \begin{array}{ll} & \text{Airport(VIE,Vienna),} \\ \text{Flight(VIE,LHR,BA),} & \text{Airport(LHR,London),} \\ \text{Flight(LHR,EDI,BA),} & \text{Airport(LGW,London),} \\ \text{Flight(LGW,GLA,U2),} & \text{Airport(LCA,Larnaca),} \\ \text{Flight(LCA,VIE,OS),} & \text{Airport(GLA,Glasgow),} \\ & \text{Airport(EDI,Edinburgh)} \end{array} \right\}$$

Q(z) :- Airport(x,London), Airport(y,Glasgow), Flight(x,y,z)

Pattern Matching Problem

List the airlines that fly directly from London to Glasgow

$$\left\{ \begin{array}{ll} & \text{Airport(VIE,Vienna),} \\ \text{Flight(VIE,LHR,BA),} & \text{Airport(LHR,London),} \\ \text{Flight(LHR,EDI,BA),} & \textbf{Airport(LGW,London)}\text{,} \\ \textbf{Flight(LGW,GLA,U2)}\text{,} & \text{Airport(LCA,Larnaca),} \\ \text{Flight(LCA,VIE,OS),} & \textbf{Airport(GLA,Glasgow)}\text{,} \\ & \text{Airport(EDI,Edinburgh)} \end{array} \right\}$$

Q(z) :- Airport(x,London), Airport(y,Glasgow), Flight(x,y,z)

Homomorphism

- Pattern matching - properly formalized via the key notion of homomorphism

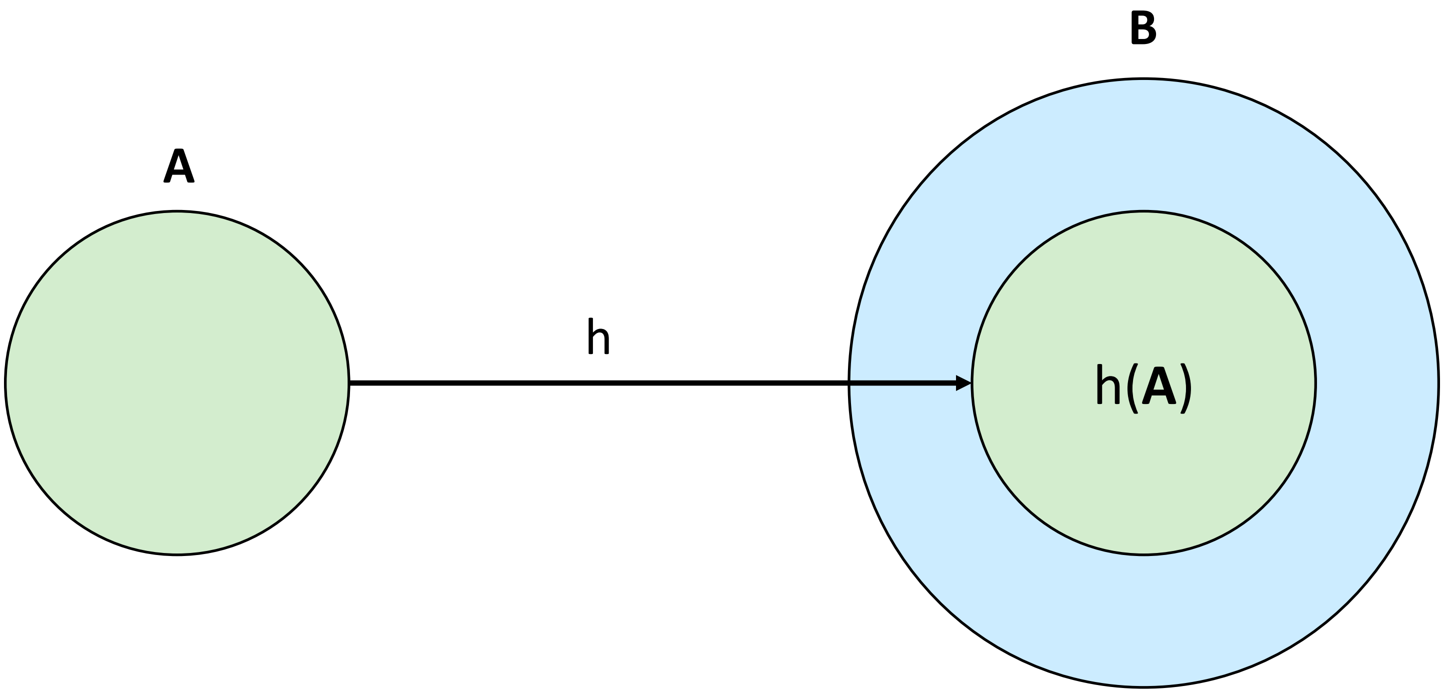
- A substitution from a set of terms **S** to a set of terms **T** is a function $h : \mathbf{S} \rightarrow \mathbf{T}$, i.e., h is a set of mappings of the form $s \mapsto t$, where $s \in \mathbf{S}$ and $t \in \mathbf{T}$

- A homomorphism from a set of atoms **A** to a set of atoms **B** is a substitution $h : \text{terms}(\mathbf{A}) \rightarrow \text{terms}(\mathbf{B})$ such that:

    1. t is a constant value $\Rightarrow$ $h(t) = t$
    2. $R(t_1,\ldots,t_k) \in \mathbf{A} \Rightarrow h(R(t_1,\ldots,t_k)) = R(h(t_1),\ldots,h(t_k)) \in \mathbf{B}$

(terms(**A**) = {t | t is a variable or a constant value that occurs in **A**})

Homomorphism



h : terms(**A**) → terms(**B**) that is the identity on constants

Homomorphism

(a,b,c,d are constants)

A

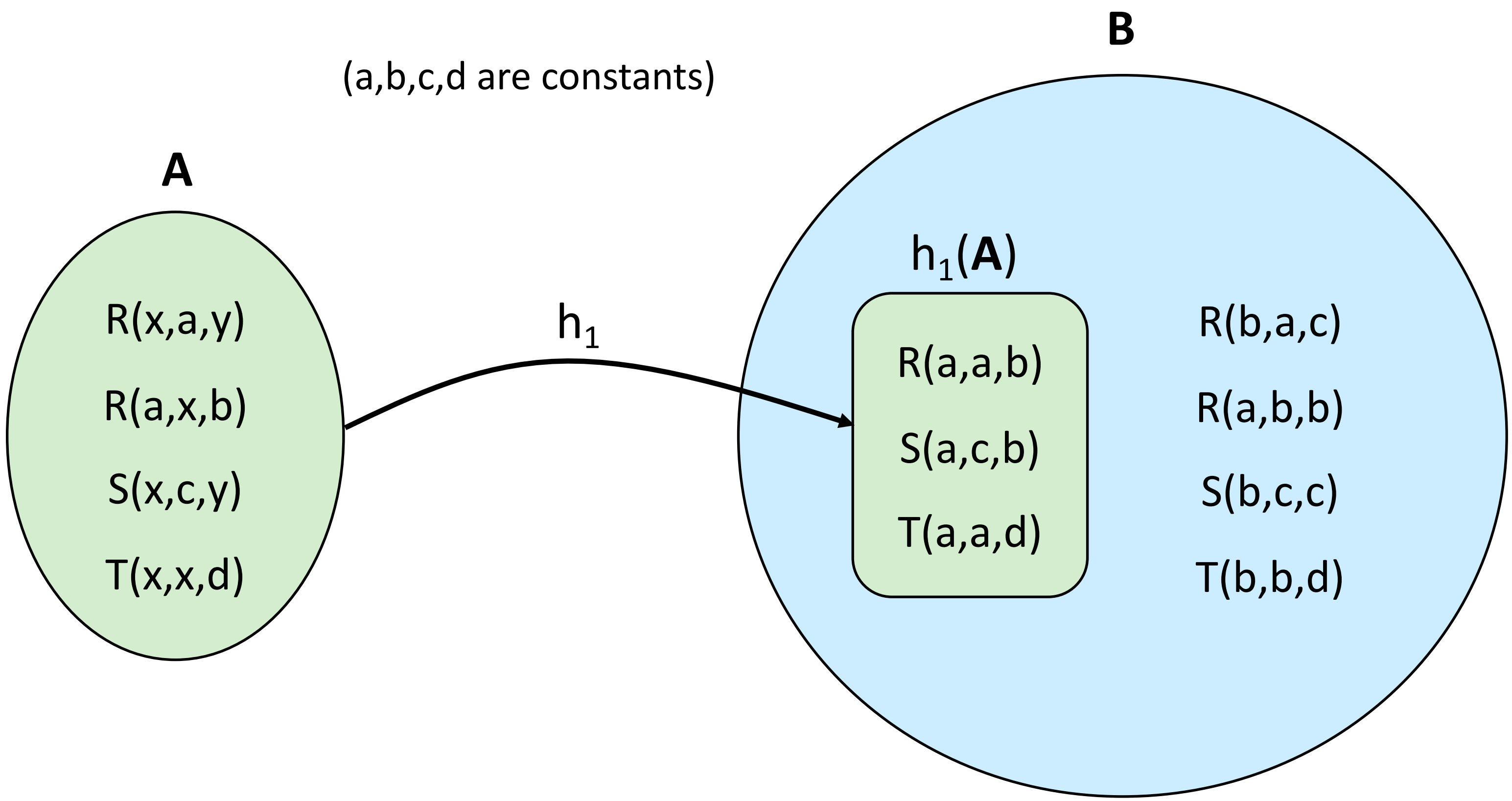
R(x,a,y)
R(a,x,b)
S(x,c,y)
T(x,x,d)

B

R(a,a,b)
S(a,c,b)
T(a,a,d)

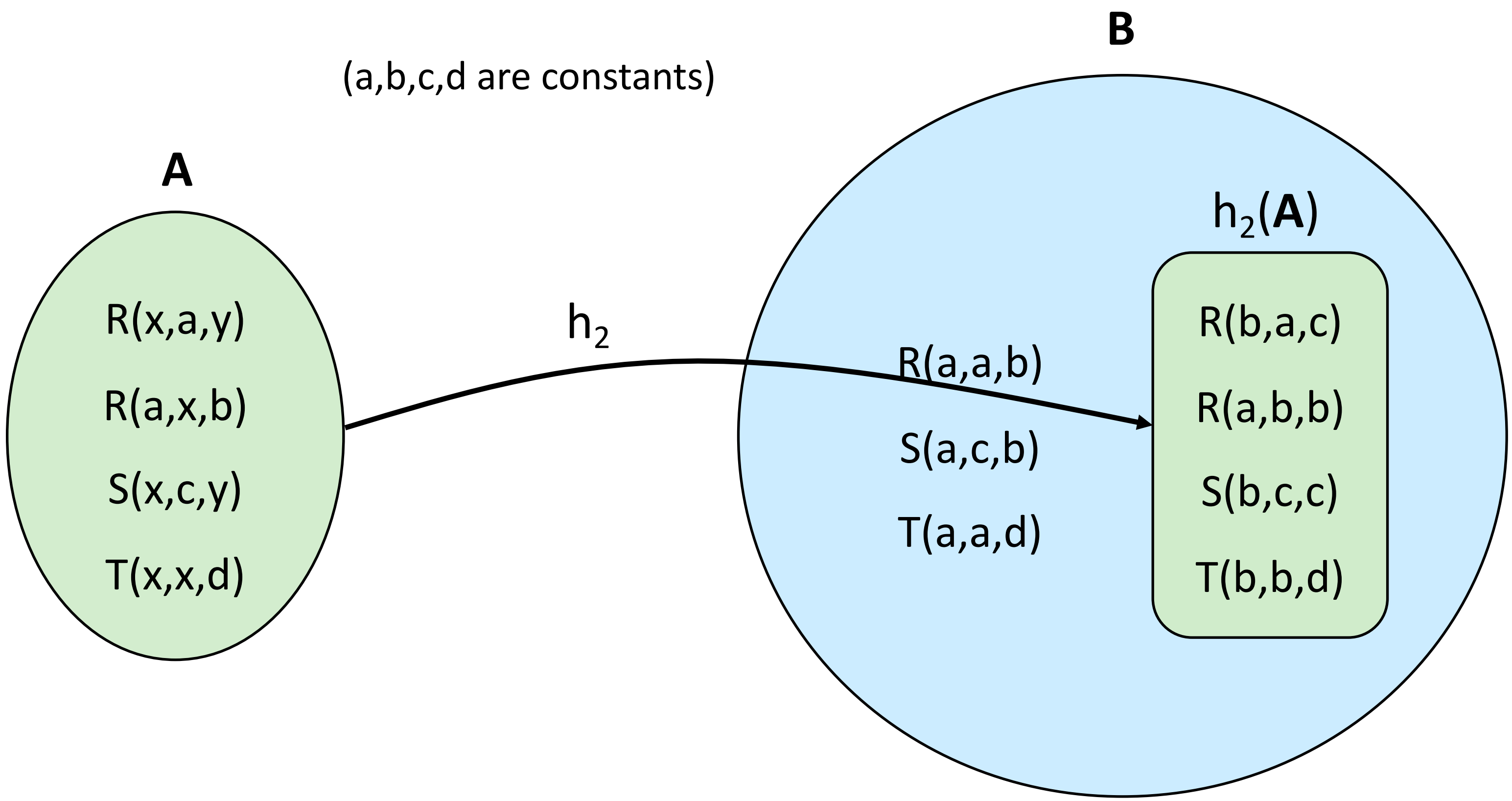
R(b,a,c)
R(a,b,b)
S(b,c,c)
T(b,b,d)

Homomorphism



$h_1 = \{a \mapsto a, b \mapsto b, c \mapsto c, d \mapsto d, x \mapsto a, y \mapsto b\}$

Homomorphism



$h_2 = \{a \mapsto a, b \mapsto b, c \mapsto c, d \mapsto d, x \mapsto b, y \mapsto c\}$

Find the Homomorphisms

$\mathbf{S_1} = \{P(x_1,y_1), P(y_1,z_1), P(z_1,w_1)\}$

$\mathbf{S_2} = \{P(x_2,y_2), P(y_2,z_2), P(z_2,x_2)\}$

$\mathbf{S_3} = \{P(x_3,y_3), P(y_3,x_3)\}$

$\mathbf{S_4} = \{P(x_4,y_4), P(y_4,x_4), P(y_4,y_4)\}$

$\mathbf{S_5} = \{P(x_5,x_5)\}$

Find the Homomorphisms

$$\mathbf{S_1} \quad = \quad \{P(x_1,y_1),\ P(y_1,z_1),\ P(z_1,w_1)\}$$

$\{x_1 \mapsto x_2,\ y_1 \mapsto y_2 ,\ z_1 \mapsto z_2 ,\ w_1 \mapsto x_2\}$

$\{x_1 \mapsto x_3,\ y_1 \mapsto y_3 ,\ z_1 \mapsto x_3 ,\ w_1 \mapsto y_3\}$

$$\mathbf{S_2} \quad = \quad \{P(x_2,y_2),\ P(y_2,z_2),\ P(z_2,x_2)\}$$

$$\mathbf{S_3} \quad = \quad \{P(x_3,y_3),\ P(y_3,x_3)\}$$

$$\mathbf{S_4} \quad = \quad \{P(x_4,y_4),\ P(y_4,x_4),\ P(y_4,y_4)\}$$

$$\mathbf{S_5} \quad = \quad \{P(x_5,x_5)\}$$

Find the Homomorphisms

$$\begin{tikzcd}[column sep=small]
 & \mathbf{S_1} = \{P(x_1,y_1), P(y_1,z_1), P(z_1,w_1)\} \arrow[dl, "{\{x_1 \mapsto x_2,\ y_1 \mapsto y_2,\ z_1 \mapsto z_2,\ w_1 \mapsto x_2\}}"'] \arrow[dr, "{\{x_1 \mapsto x_3,\ y_1 \mapsto y_3,\ z_1 \mapsto x_3,\ w_1 \mapsto y_3\}}"] & \\
\mathbf{S_2} = \{P(x_2,y_2), P(y_2,z_2), P(z_2,x_2)\} \arrow[dr, "{\{x_2 \mapsto y_4,\ y_2 \mapsto x_4,\ z_2 \mapsto y_4\}}"'] & & \mathbf{S_3} = \{P(x_3,y_3), P(y_3,x_3)\} \arrow[dl, "{\{x_3 \mapsto x_4,\ y_3 \mapsto y_4\}}"] \\
 & \mathbf{S_4} = \{P(x_4,y_4), P(y_4,x_4), P(y_4,y_4)\} &
\end{tikzcd}$$

$$\mathbf{S_5} = \{P(x_5,x_5)\}$$

Find the Homomorphisms

$$S_1 = \{P(x_1,y_1),\ P(y_1,z_1),\ P(z_1,w_1)\}$$

$\{x_1 \mapsto x_2,\ y_1 \mapsto y_2\ ,\ z_1 \mapsto z_2\ ,\ w_1 \mapsto x_2\}$

$\{x_1 \mapsto x_3,\ y_1 \mapsto y_3\ ,\ z_1 \mapsto x_3\ ,\ w_1 \mapsto y_3\}$

$$S_2 = \{P(x_2,y_2),\ P(y_2,z_2),\ P(z_2,x_2)\}$$

$$S_3 = \{P(x_3,y_3),\ P(y_3,x_3)\}$$

$\{x_2 \mapsto y_4,\ y_2 \mapsto x_4\ ,\ z_2 \mapsto y_4\}$

$\{x_3 \mapsto x_4,\ y_3 \mapsto y_4\}$

$$S_4 = \{P(x_4,y_4),\ P(y_4,x_4),\ P(y_4,y_4)\}$$

$\{x_4 \mapsto x_5,\ y_4 \mapsto x_5\}$

$$S_5 = \{P(x_5,x_5)\}$$

Find the Homomorphisms

$$
\begin{tikzcd}[column sep=small]
 & \mathbf{S_1} = \{P(x_1,y_1), P(y_1,z_1), P(z_1,w_1)\} \arrow[dl, "\{x_1 \mapsto x_2,\ y_1 \mapsto y_2,\ z_1 \mapsto z_2,\ w_1 \mapsto x_2\}"'] \arrow[dr, "\{x_1 \mapsto x_3,\ y_1 \mapsto y_3,\ z_1 \mapsto x_3,\ w_1 \mapsto y_3\}"] & \\
\mathbf{S_2} = \{P(x_2,y_2), P(y_2,z_2), P(z_2,x_2)\} \arrow[dr, "\{x_2 \mapsto y_4,\ y_2 \mapsto x_4,\ z_2 \mapsto y_4\}"'] & & \mathbf{S_3} = \{P(x_3,y_3), P(y_3,x_3)\} \arrow[dl, "\{x_3 \mapsto x_4,\ y_3 \mapsto y_4\}"] \\
 & \mathbf{S_4} = \{P(x_4,y_4), P(y_4,x_4), P(y_4,y_4)\} \arrow[d, shift right=2, "\{x_4 \mapsto x_5,\ y_4 \mapsto x_5\}"'] & \\
 & \mathbf{S_5} = \{P(x_5,x_5)\} \arrow[u, shift right=2, "\{x_5 \mapsto y_4\}"'] &
\end{tikzcd}
$$

Homomorphisms Compose

$$\begin{tikzcd}
 & \mathbf{S_1} = \{P(x_1,y_1), P(y_1,z_1), P(z_1,w_1)\} \arrow[dl, "\{x_1 \mapsto x_2,\, y_1 \mapsto y_2 ,\, z_1 \mapsto z_2 ,\, w_1 \mapsto x_2\}"'] \arrow[dd, "\{x_1 \mapsto y_4,\, y_1 \mapsto x_4 ,\, z_1 \mapsto y_4 ,\, w_1 \mapsto y_4\}"] \\
\mathbf{S_2} = \{P(x_2,y_2), P(y_2,z_2), P(z_2,x_2)\} \arrow[dr, "\{x_2 \mapsto y_4,\, y_2 \mapsto x_4 ,\, z_2 \mapsto y_4\}"'] & \\
 & \mathbf{S_4} = \{P(x_4,y_4), P(y_4,x_4), P(y_4,y_4)\}
\end{tikzcd}$$

Homomorphisms Compose

$$\begin{tikzcd}
\mathbf{S_1} = \{P(x_1,y_1), P(y_1,z_1), P(z_1,w_1)\} \arrow[dd, "{\{x_1 \mapsto x_4,\ y_1 \mapsto y_4,\ z_1 \mapsto x_4,\ w_1 \mapsto y_4\}}"'] \arrow[dr, "{\{x_1 \mapsto x_3,\ y_1 \mapsto y_3,\ z_1 \mapsto x_3,\ w_1 \mapsto y_3\}}"] & \\
& \mathbf{S_3} = \{P(x_3,y_3), P(y_3,x_3)\} \arrow[dl, "{\{x_3 \mapsto x_4,\ y_3 \mapsto y_4\}}"] \\
\mathbf{S_4} = \{P(x_4,y_4), P(y_4,x_4), P(y_4,y_4)\} &
\end{tikzcd}$$

Semantics of Conjunctive Queries

- A match of a conjunctive query $Q(x_1,\ldots,x_k)$:- body in a database D is a homomorphism h from the set of atoms body to the set of atoms D

- The answer to $Q(x_1,\ldots,x_k)$:- body over D is the set of k-tuples

  $$Q(D) := \{(h(x_1),\ldots,h(x_k)) \mid h \text{ is a match of } Q \text{ in } D\}$$

- The answer consists of the witnesses for the distinguished variables of Q

Pattern Matching Problem

List the airlines that fly directly from London to Glasgow

	Airport(VIE,Vienna),
Flight(VIE,LHR,BA),	Airport(LHR,London),
Flight(LHR,EDI,BA),	Airport(LGW,London),
Flight(LGW,GLA,U2),	Airport(LCA,Larnaca),
Flight(LCA,VIE,OS),	Airport(GLA,Glasgow),
	Airport(EDI,Edinburgh)

Q(z) :- Airport(x,London), Airport(y,Glasgow), Flight(x,y,z)

Pattern Matching Problem

List the airlines that fly directly from London to Glasgow

$$\left\{\begin{array}{ll} & \text{Airport(VIE,Vienna),} \\ \text{Flight(VIE,LHR,BA),} & \text{Airport(LHR,London),} \\ \text{Flight(LHR,EDI,BA),} & \textbf{Airport(LGW,London)}, \\ \textbf{Flight(LGW,GLA,U2)}, & \text{Airport(LCA,Larnaca),} \\ \text{Flight(LCA,VIE,OS),} & \textbf{Airport(GLA,Glasgow)}, \\ & \text{Airport(EDI,Edinburgh)} \end{array}\right\}$$

{x ↦ LGW, y ↦ GLA, z ↦ U2,
London ↦ London, Glasgow ↦ Glasgow}

Q(z) :- Airport(x,London), Airport(y,Glasgow), Flight(x,y,z)

Complexity of **CQ**

Theorem: It holds that:

- BQE(**CQ**) is NP-complete (combined complexity)
- BQE[Q](**CQ**) is in LOGSPACE, for a fixed query Q ∈ **CQ** (data complexity)

Proof:

(NP-membership) Consider a database D, and a Boolean CQ Q :- body

Guess a substitution h : terms(body) → terms(D)

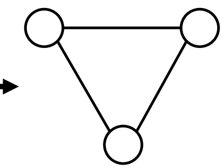
Verify that h is a match of Q in D, i.e., h(body) ⊆ D

(NP-hardness) Reduction from 3-colorability

NP-hardness

(NP-hardness) Reduction from 3-colorability

> 3COL
>
> **Input:** an undirected graph **G** = (V,E)
>
> **Question:** is there a function $c : V \rightarrow \{R,G,B\}$ such that $(v,u) \in E \Rightarrow c(v) \neq c(u)$?

Lemma: G is 3-colorable iff **G** can be mapped to $\mathbf{K_3}$, i.e., **G** $\xrightarrow{\text{hom}}$ 

therefore, **G** is 3-colorable iff there is a match of Q_G in $D = \{E(a,b),E(b,c),E(c,d)\}$

the Boolean CQ that represents **G**

Complexity of **CQ**

Theorem: It holds that:

- BQE(**CQ**) is NP-complete (combined complexity)
- BQE[Q](**CQ**) is in LOGSPACE, for a fixed query Q ∈ **CQ** (data complexity)

Proof:

(NP-membership) Consider a database D, and a Boolean CQ Q :- body

Guess a substitution h : terms(body) → terms(D)

Verify that h is a match of Q in D, i.e., h(body) ⊆ D

(NP-hardness) Reduction from 3-colorability

(LOGSPACE-membership) Inherited from BQE[Q](**DRC**)

What About Optimization of CQs?

SAT(**CQ**)

Input: a query $Q \in \mathbf{CQ}$

Question: is there a (finite) database D such that $Q(D)$ is non-empty?

EQUIV(**CQ**)

Input: two queries $Q_1 \in \mathbf{CQ}$ and $Q_2 \in \mathbf{CQ}$

Question: $Q_1 \equiv Q_2$? or $Q_1(D) = Q_2(D)$ for every (finite) database D?

CONT(**CQ**)

Input: two queries $Q_1 \in \mathbf{CQ}$ and $Q_2 \in \mathbf{CQ}$

Question: $Q_1 \subseteq Q_2$? or $Q_1(D) \subseteq Q_2(D)$ for every (finite) database D?

Canonical Database

- Convert a conjunctive query Q into a database D[Q] - the canonical database of Q

- Given a conjunctive query of the form Q(**x**) :- body, D[Q] is obtained from body by replacing each variable x with a new constant c(x) = $\underline{x}$

- E.g., given Q(x,y) :- R(x,y), P(y,z,w), R(z,x), then D[Q] = {R($\underline{x}$,$\underline{y}$), P($\underline{y}$,$\underline{z}$,$\underline{w}$), R($\underline{z}$,$\underline{x}$)}

- **Note:** The mapping c : {variables in body} → {new constants} is a bijection, where c(body) = D[Q] and c^{-1}(D[Q]) = body

Satisfiability of CQs

SAT(**CQ**)

Input: a query Q ∈ **CQ**

Question: is there a (finite) database D such that Q(D) is non-empty?

Theorem: A query Q ∈ **CQ** is always satisfiable - SAT(**CQ**) ∈ O(1)-time

Proof: Due to its canonical database - Q(D[Q]) is trivially non-empty

Equivalence and Containment of CQs

EQUIV(**CQ**)

Input: two queries $Q_1 \in$ **CQ** and $Q_2 \in$ **CQ**

Question: $Q_1 \equiv Q_2$? or $Q_1(D) = Q_2(D)$ for every (finite) database D?

CONT(**CQ**)

Input: two queries $Q_1 \in$ **CQ** and $Q_2 \in$ **CQ**

Question: $Q_1 \subseteq Q_2$? or $Q_1(D) \subseteq Q_2(D)$ for every (finite) database D?

$Q_1 \equiv Q_2$ iff $Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_1$

$Q_1 \subseteq Q_2$ iff $Q_1 \equiv (Q_1 \wedge Q_2)$

…thus, we can safely focus on CONT(**CQ**)

Homomorphism Theorem

A query homomorphism from $Q_1(x_1,\ldots,x_k)$:- $body_1$ to $Q_2(y_1,\ldots,y_k)$:- $body_2$

is a substitution h : terms($body_1$) → terms($body_2$) such that:

1. h is a homomorphism from $body_1$ to $body_2$
2. $(h(x_1),\ldots,h(x_k)) = (y_1,\ldots,y_k)$

Homomorphism Theorem: Let Q_1 and Q_2 be conjunctive queries. It holds that:

$Q_1 \subseteq Q_2$ iff there exists a query homomorphism from Q_2 to Q_1

Homomorphism Theorem: Example

$Q_1(x,y)$:- $R(x,z)$, $S(z,z)$, $R(z,y)$

$h = \{u \mapsto x, v \mapsto y, w \mapsto z, t \mapsto z\}$

$Q_2(u,v)$:- $R(u,w)$, $S(w,t)$, $R(t,v)$

- h is a query homomorphism from Q_2 to Q_1 $\Rightarrow$ $Q_1 \subseteq Q_2$
- But, there is no homomorphism from Q_1 to Q_2 $\Rightarrow$ $Q_1 \subset Q_2$

Homomorphism Theorem: Proof

Assume that $Q_1(x_1,\ldots,x_k)$:- $body_1$ and $Q_2(y_1,\ldots,y_k)$:- $body_2$

($\Rightarrow$) $Q_1 \subseteq Q_2 \;\Rightarrow\;$ there exists a query homomorphism from Q_2 to Q_1

- Clearly, $(c(x_1),\ldots,c(x_k)) \in Q_1(D[Q_1])$ - recall that $D[Q_1] = c(body_1)$
- Since $Q_1 \subseteq Q_2$, we conclude that $(c(x_1),\ldots,c(x_k)) \in Q_2(D[Q_1])$
- Therefore, there exists a homomorphism h such that $h(body_2) \subseteq D[Q_1] = c(body_1)$

  and $h((y_1,\ldots,y_k)) = (c(x_1),\ldots,c(x_k))$
- By construction, $c^{-1}(c(body_1)) = body_1$

  and $c^{-1}((c(x_1),\ldots,c(x_k))) = (x_1,\ldots,x_k)$
- Therefore, $c^{-1} \circ h$ is a

  query homomorphism from Q_2 to Q_1

$$
\begin{tikzcd}
Q_2(y_1,\ldots,y_k) \arrow[d] & :- & body_2 \arrow[d, "h"] \arrow[dd, bend left=60, "c^{-1} \circ h"] \\
Q_1(c(x_1),\ldots,c(x_k)) \arrow[d] & :- & c(body_1) \arrow[d, "c^{-1}"] \\
Q_1(x_1,\ldots,x_k) & :- & body_1
\end{tikzcd}
$$

Homomorphism Theorem: Proof

Assume that $Q_1(x_1,\ldots,x_k)$:- body_1 and $Q_2(y_1,\ldots,y_k)$:- body_2

($\Leftarrow$) $Q_1 \subseteq Q_2 \;\Leftarrow\;$ there exists a query homomorphism from Q_2 to Q_1

- Consider a database D, and a tuple $\mathbf{t}$ such that $\mathbf{t} \in Q_1(D)$
- We need to show that $\mathbf{t} \in Q_2(D)$
- Clearly, there exists a homomorphism g such that $g(\mathrm{body}_1) \subseteq D$ and $g((x_1,\ldots,x_k)) = \mathbf{t}$
- By hypothesis, there exists a query homomorphism h from Q_2 to Q_1
- Therefore, $g(h(\mathrm{body}_2)) \subseteq D$ and

  $g(h((y_1,\ldots,y_k))) = \mathbf{t}$, which implies that $\mathbf{t} \in Q_2(D)$

$$\begin{tikzcd}
Q_2(y_1,\ldots,y_k) \arrow[d] & {:-} & \mathrm{body}_2 \arrow[d, "h"] \arrow[dd, bend left=60, "g \circ h"] \\
Q_1(x_1,\ldots,x_k) \arrow[d] & {:-} & \mathrm{body}_1 \arrow[d, "g"] \\
\mathbf{t} & & D
\end{tikzcd}$$

Existence of a Query Homomorphism

Theorem: Let Q_1 and Q_2 be conjunctive queries. The problem of deciding whether there exists a query homomorphism from Q_2 to Q_1 is NP-complete

Proof:

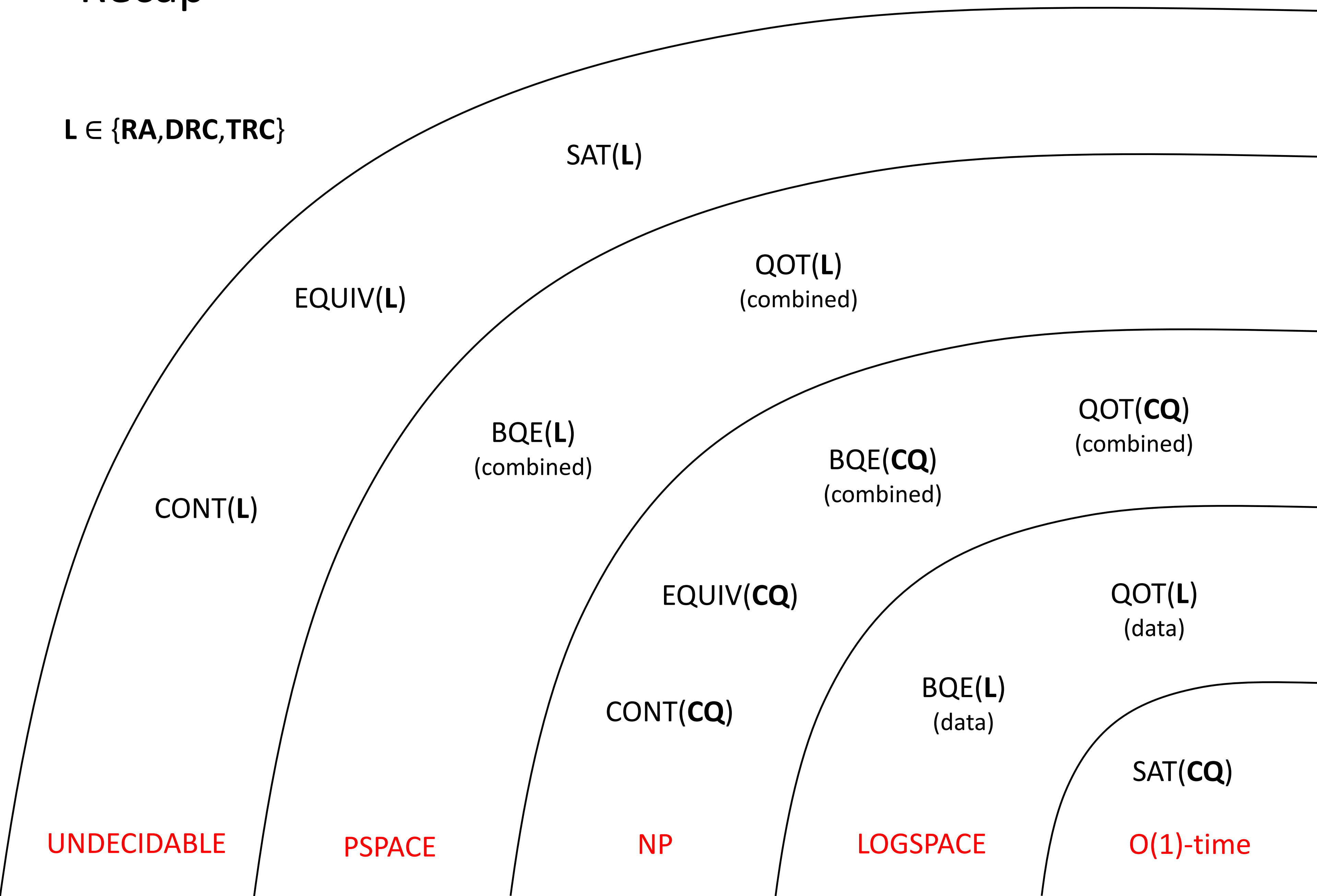
(NP-membership) Guess a substitution, and verify that is a query homomorphism

(NP-hardness) Straightforward reduction from BQE(**CQ**)

By applying the homomorphism theorem we get that:

Corollary: EQUIV(**CQ**) and CONT(**CQ**) are NP-complete

Recap


L ∈ {RA,DRC,TRC}
SAT(L)
EQUIV(L)
CONT(L)
QOT(L) (combined)
BQE(L) (combined)
EQUIV(CQ)
CONT(CQ)
BQE(CQ) (combined)
QOT(CQ) (combined)
QOT(L) (data)
BQE(L) (data)
SAT(CQ)
UNDECIDABLE
PSPACE
NP
LOGSPACE
O(1)-time

Minimizing Conjunctive Queries

- **Goal:** minimize the number of joins in a query

- A conjunctive query Q_1 is minimal if there is no conjunctive query Q_2 such that:
  1. $Q_1 \equiv Q_2$
  2. Q_2 has fewer atoms than Q_1

- The task of CQ minimization is, given a conjunctive query Q, to compute a minimal one that is equivalent to Q

Minimization by Deletion

By exploiting the homomorphism theorem we can show the following:

Theorem: Consider a conjunctive query $Q_1(x_1,\ldots,x_k)$:- $body_1$.

If Q_1 is equivalent to a conjunctive query $Q_2(y_1,\ldots,y_k)$:- $body_2$ where $|body_2| < |body_1|$,

then Q_1 is equivalent to a query $Q_3(x_1,\ldots,x_k)$:- $body_3$ such that $body_3 \subseteq body_1$

$$\Downarrow$$

The above theorem says that to minimize a conjunctive query $Q_1(x_1,\ldots,x_k)$:- body we simply need to remove some atoms from body

Minimization Procedure

Minimization($Q(x_1,\dots,x_k)$:- body)

While there is an atom $\alpha \in$ body such that the variables $x_1,\dots,x_k$ appear in body $\setminus \{\alpha\}$, and

there is a query homomorphism from $Q(x_1,\dots,x_k)$:- body to $Q(x_1,\dots,x_k)$:- body $\setminus \{\alpha\}$ **do**

body := body $\setminus \{\alpha\}$

Return $Q(x_1,\dots,x_k)$:- body

Note: if there is a query homomorphism from $Q(x_1,\dots,x_k)$:- body to $Q(x_1,\dots,x_k)$:- body $\setminus \{\alpha\}$, then the two queries are equivalent since there is trivially a query homomorphism from the latter to the former query

Minimization Procedure: Example

(a,b,c,d are constants)

Q(x) :- R(x,y), R(x,b), R(a,b), R(u,c), R(u,v), S(a,c,d)

$\{y \mapsto b\}$

Q(x) :- R(x,b), R(a,b), R(u,c), R(u,v), S(a,c,d)

$\{v \mapsto c\}$

$\boxed{\text{Q(x) :- R(x,b), R(a,b), R(u,c), S(a,c,d)}}$

minimal query

Note: the mapping x $\mapsto$ a is not valid since x is a distinguished variable

Uniqueness of Minimal Queries

Natural question: does the order in which we remove atoms from the body of the input conjunctive query matter?

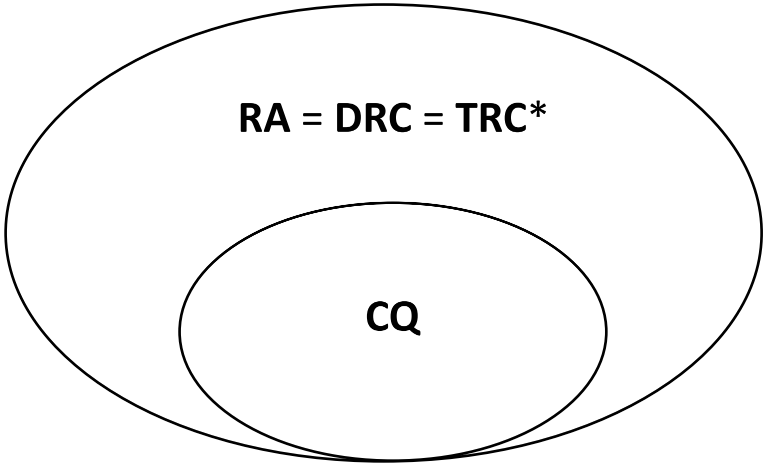
Theorem: Consider a conjunctive query Q. Let Q_1 and Q_2 be minimal conjunctive queries such that $Q_1 \equiv Q$ and $Q_2 \equiv Q$. Then, Q_1 and Q_2 are isomorphic (i.e., they are the same up to variable renaming)

Therefore, given a conjunctive query Q, the result of Minimization(Q) is unique (up to variable renaming) and is called the core of Q

Recap

- The main relational query languages - **RA/DRC/TRC**
  - Evaluation is decidable - foundations of the database industry
  - Perfect query optimization is impossible

- Conjunctive queries - an important query language
  - All the relevant algorithmic problems are decidable
  - Query minimization



*under the active domain semantics

MAI4CAREU

Master programmes in Artificial Intelligence 4 Careers in Europe

University of Cyprus

MAI649: PRINCIPLES OF ONTOLOGICAL DATABASES

Thank You!

Andreas Pieris

Spring 2022-2023

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

MAI4CAREU | Master programmes in Artificial Intelligence 4 Careers in Europe

University of Cyprus

MAI649: PRINCIPLES OF ONTOLOGICAL DATABASES

Fast CQ Evaluation

Andreas Pieris

Spring 2022-2023

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

Learning Outcomes

- Acyclicity of conjunctive queries
- Analyze the complexity of evaluating acyclic conjunctive queries
- Semantic acyclicity of conjunctive queries

Complexity of **CQ**

Theorem: It holds that:

- BQE(**CQ**) is NP-complete (combined complexity)
- BQE[Q](**CQ**) is in LOGSPACE, for a fixed query Q ∈ **CQ** (data complexity)

Proof:

(NP-membership) Consider a database D, and a Boolean CQ Q :- body

Guess a substitution h : terms(body) → terms(D)

Verify that h is a match of Q in D, i.e., h(body) ⊆ D

(NP-hardness) Reduction from 3-colorability

(LOGSPACE-membership) Inherited from BQE[Q](**DRC**)

Complexity of **CQ**

Theorem: It holds that:

- BQE(**CQ**) is NP-complete (combined complexity)
- BQE[Q](**CQ**) is in LOGSPACE, for a fixed query Q ∈ **CQ** (data complexity)

Evaluating a CQ Q over a database D takes time $||D||^{O(||Q||)}$

Minimizing Conjunctive Queries

- Database theory has developed principled methods for optimizing CQs:
  - Find an equivalent CQ with minimal number of atoms (the core)
  - Provides a notion of "true" optimality

Q(x) :- R(x,y), R(x,b), R(a,b), R(u,c), R(u,v), S(a,c,d)

{y ↦ b}

Q(x) :- R(x,b), R(a,b), R(u,c), R(u,v), S(a,c,d)

{v ↦ c}

Q(x) :- R(x,b), R(a,b), R(u,c), S(a,c,d)

minimal query

Minimizing Conjunctive Queries

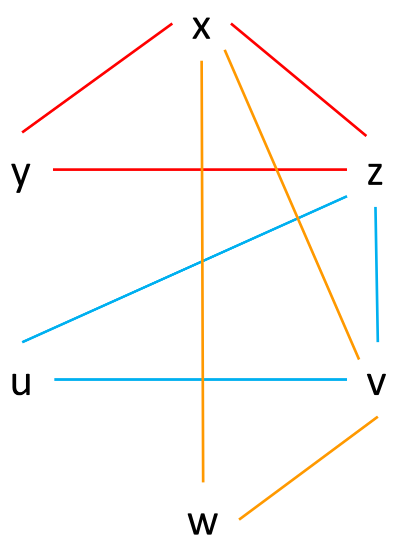
- But, a minimal equivalent CQ might not be easier to evaluate - remains NP-hard

- "Good" classes of CQs for which query evaluation is tractable (*in combined complexity*):
  - Graph-based
  - Hypergraph-based

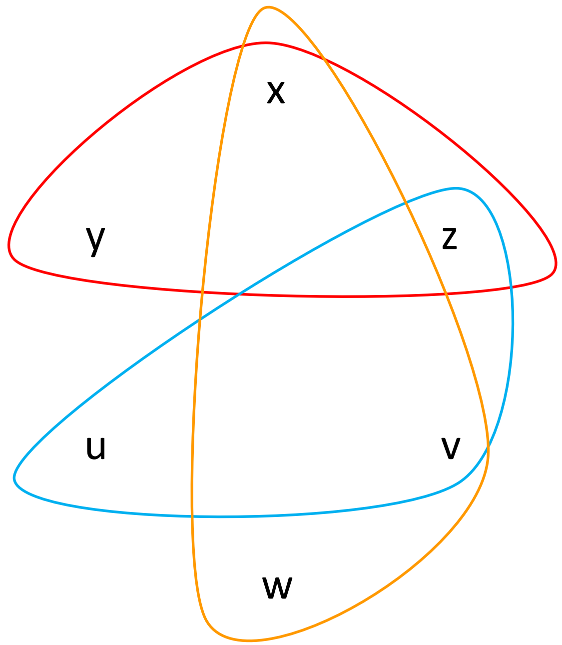
(Hyper)graph of Conjunctive Queries

Q :- R(x,y,z), R(z,u,v), R(v,w,x)

graph of Q - G(Q)



hypergraph of Q - H(Q)



“Good” Classes of Conjunctive Queries

measures how close a graph is to a tree

- Graph-based
  - CQs of bounded treewidth - their graph has bounded treewidth

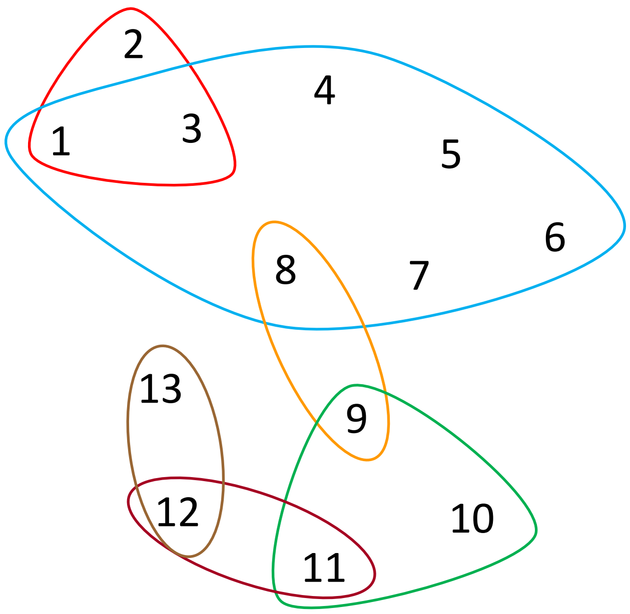
measures how close a hypergraph is to an acyclic one

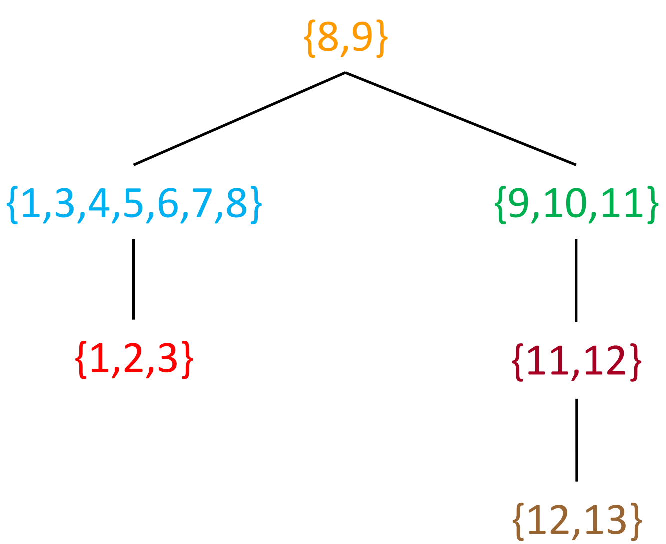
- Hypergraph-based:
  - CQs of bounded hypertree width - their hypergraph has bounded hypertree width
  - **Acyclic CQs - their hypergraph has hypertree width 1**

Acyclic Hypergraphs

A join tree of a hypergraph **H** = (V,E) is a labeled tree **T** = (N,F,L), where L : N → E such that:

1. For each hyperedge e ∈ E of **H**, there exists n ∈ N such that e = L(n)
2. For each node u ∈ V of **H**, the set {n ∈ N | u ∈ L(n)} induces a **connected** subtree of **T**

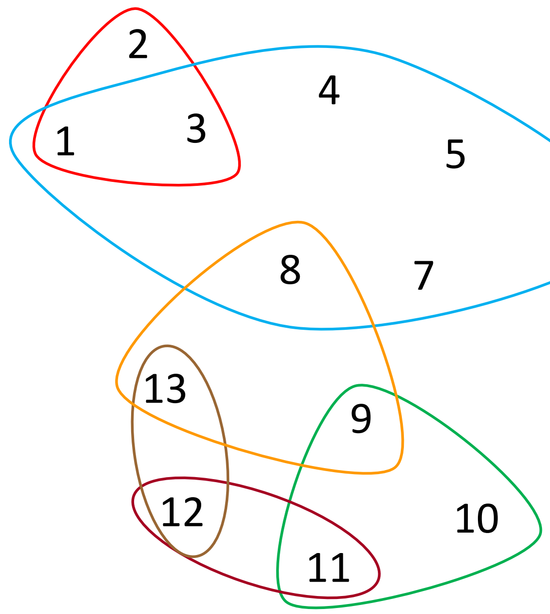


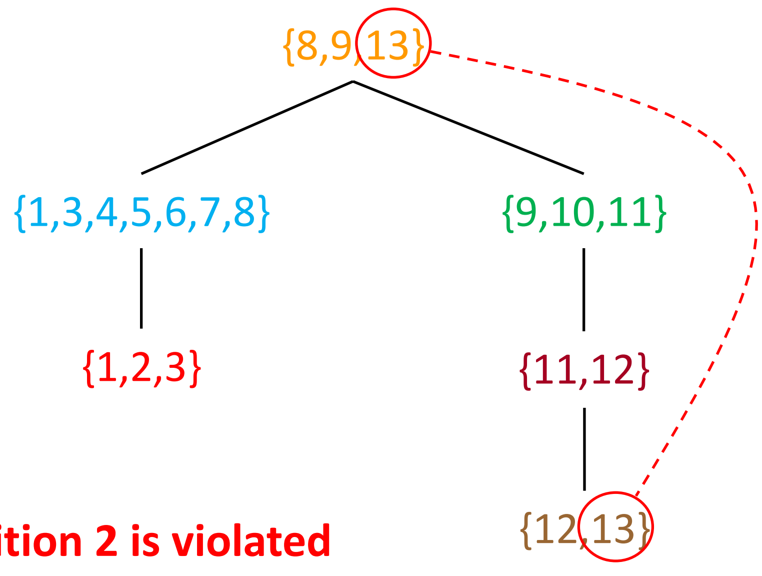


Acyclic Hypergraphs

A join tree of a hypergraph **H** = (V,E) is a labeled tree **T** = (N,F,L), where L : N → E such that:

1. For each hyperedge e ∈ E of **H**, there exists n ∈ N such that e = L(n)
2. For each node u ∈ V of **H**, the set {n ∈ N | u ∈ L(n)} induces a **connected** subtree of **T**





condition 2 is violated

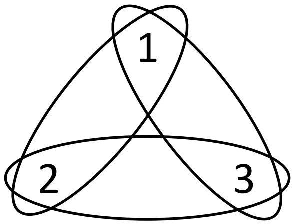
Acyclic Hypergraphs

A join tree of a hypergraph **H** = (V,E) is a labeled tree **T** = (N,F,L), where L : N → E such that:

1. For each hyperedge e ∈ E of **H**, there exists n ∈ N such that e = L(n)

2. For each node u ∈ V of **H**, the set {n ∈ N | u ∈ L(n)} induces a **connected** subtree of **T**

Definition: A hypergraph is acyclic if it has a join tree



prime example of a cyclic hypergraph

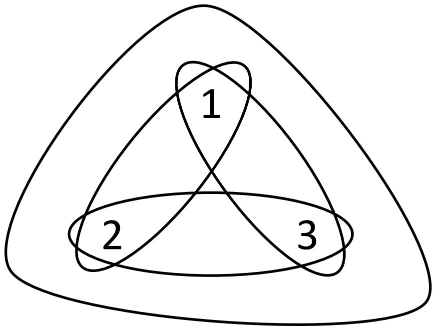
Acyclic Hypergraphs

A join tree of a hypergraph **H** = (V,E) is a labeled tree **T** = (N,F,L), where L : N → E such that:

1. For each hyperedge e ∈ E of **H**, there exists n ∈ N such that e = L(n)

2. For each node u ∈ V of **H**, the set {n ∈ N | u ∈ L(n)} induces a **connected** subtree of **T**

Definition: A hypergraph is acyclic if it has a join tree



but this is acyclic

Relevant Algorithmic Tasks

ACYCLICITY

Input: a query Q ∈ **CQ**

Question: is Q acyclic? or is H(Q) acyclic?

{Q ∈ **CQ** | H(Q) is acyclic}

BQE(**ACQ**)

Input: a database D, a Boolean query Q ∈ **ACQ**

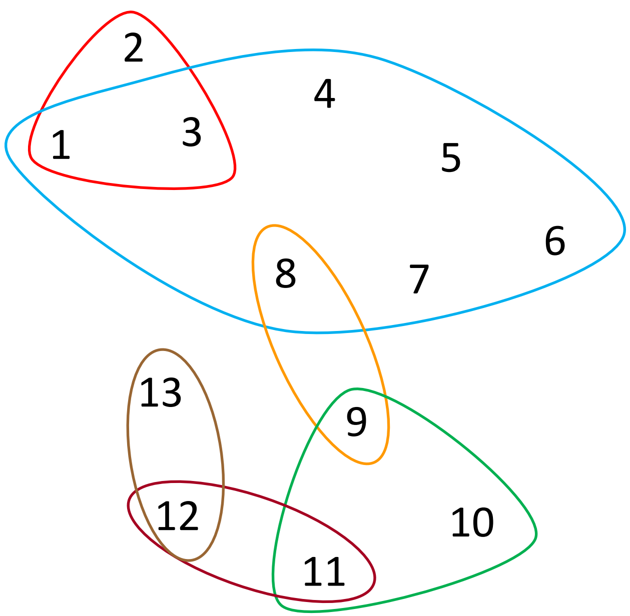
Question: is Q(D) non-empty?

Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge

2. Eliminate hyperedges that are empty or contained in other hyperedges



{8,9}

{1,3,4,5,6,7,8}

{9,10,11}

{1,2,3}

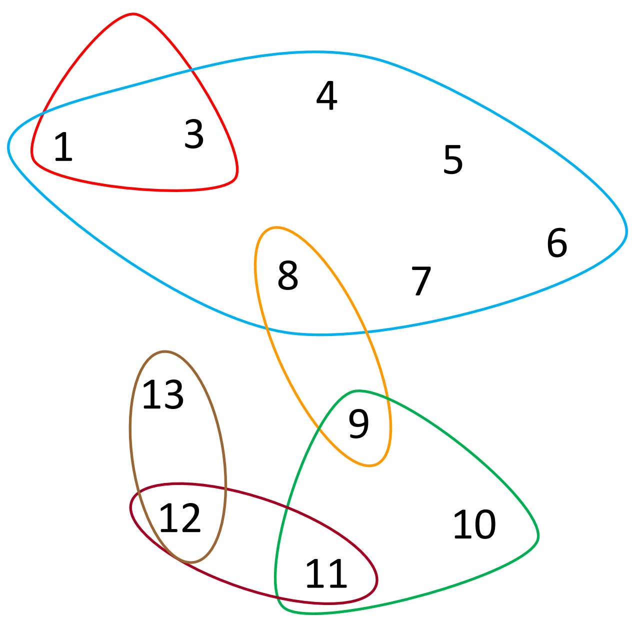
{11,12}

{12,13}

Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge
2. Eliminate hyperedges that are empty or contained in other hyperedges



{8,9}

{1,3,4,5,6,7,8}

{9,10,11}

{1,2,3}

{11,12}

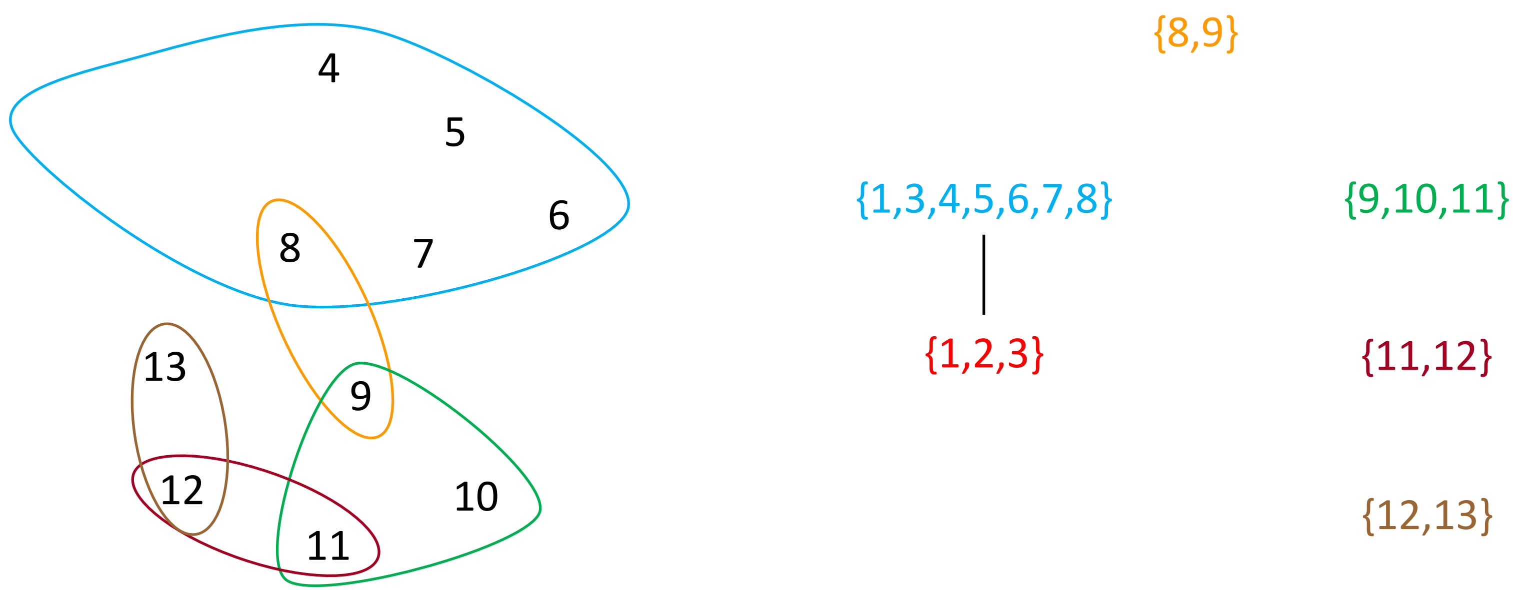
{12,13}

Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge

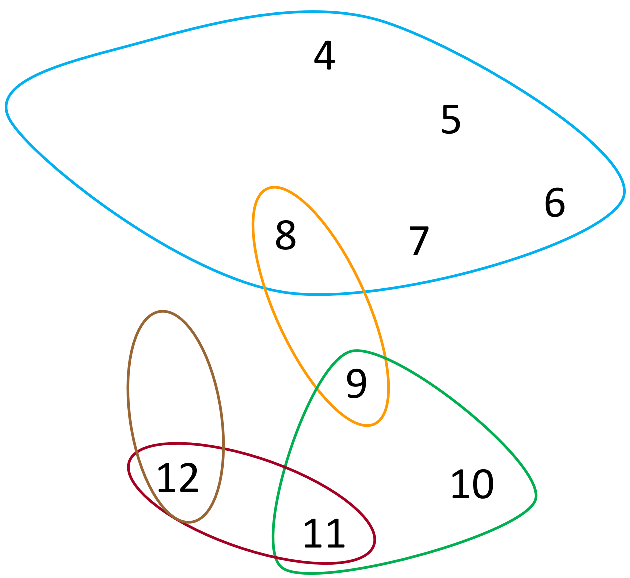
2. Eliminate hyperedges that are empty or contained in other hyperedges



Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge
2. Eliminate hyperedges that are empty or contained in other hyperedges



{8,9}

{1,3,4,5,6,7,8}

{9,10,11}

{1,2,3}

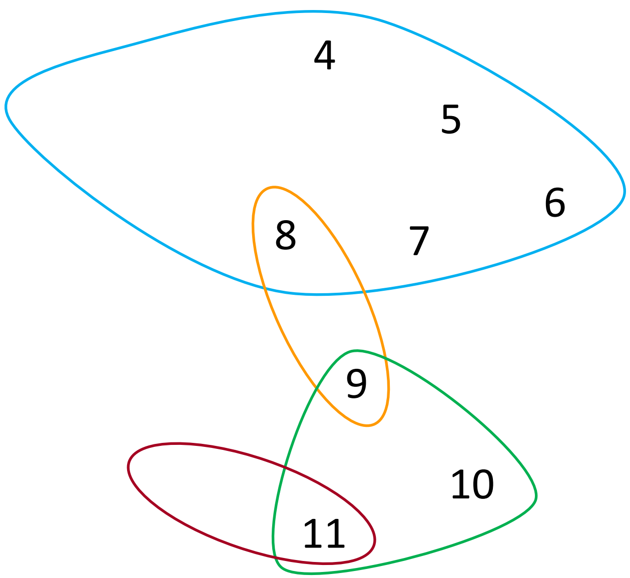
{11,12}

{12,13}

Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge
2. Eliminate hyperedges that are empty or contained in other hyperedges



{8,9}

{1,3,4,5,6,7,8}

{1,2,3}

{9,10,11}

{11,12}

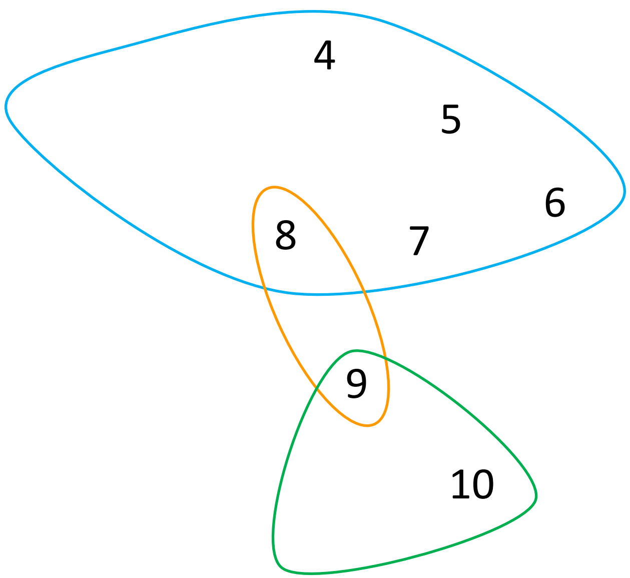
{12,13}

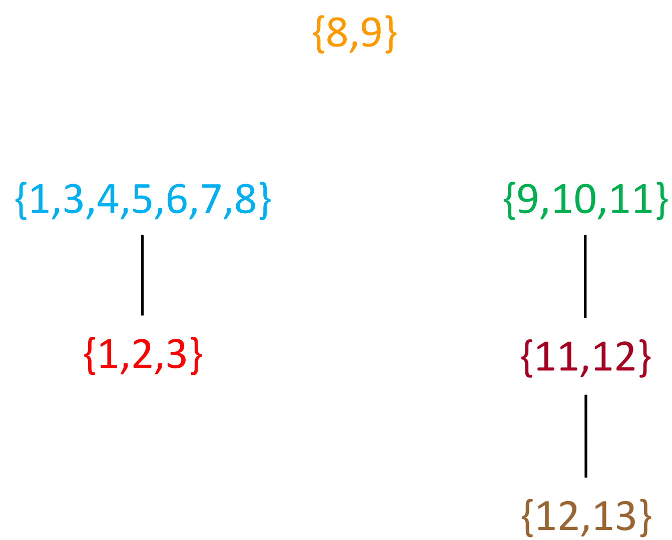
Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge

2. Eliminate hyperedges that are empty or contained in other hyperedges



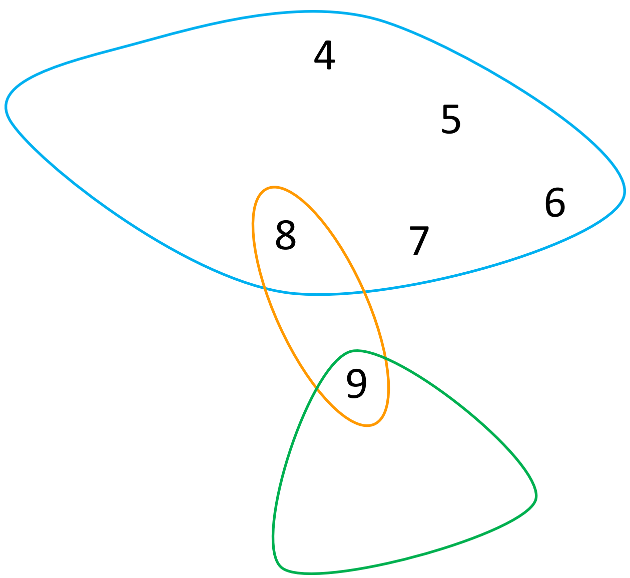


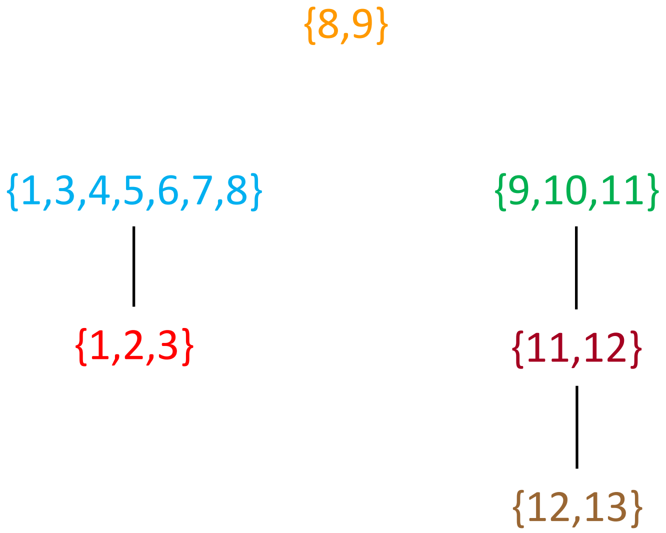
Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge

2. Eliminate hyperedges that are empty or contained in other hyperedges



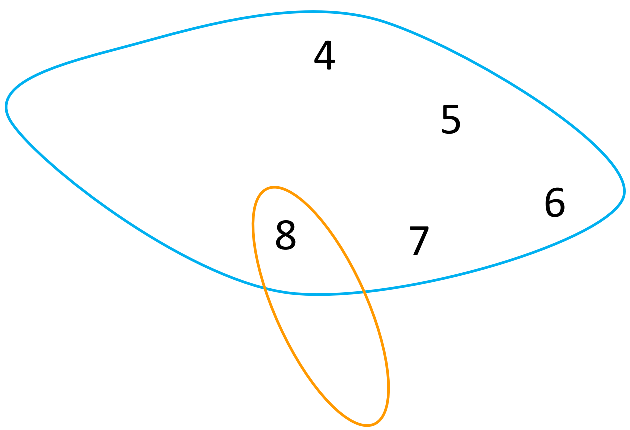


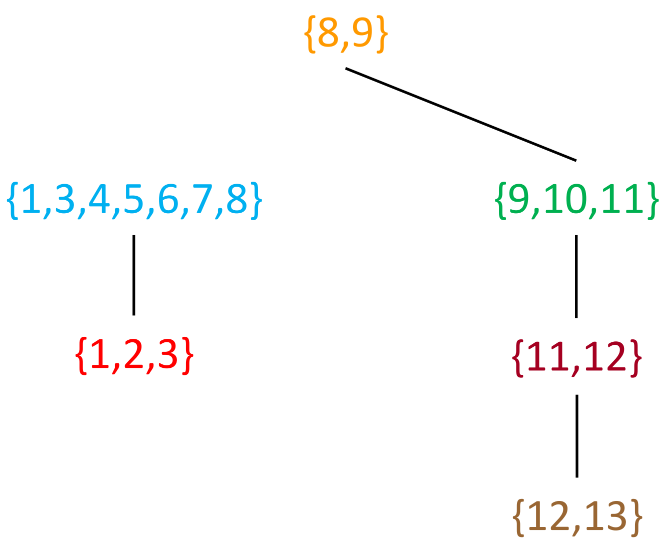
Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge

2. Eliminate hyperedges that are empty or contained in other hyperedges



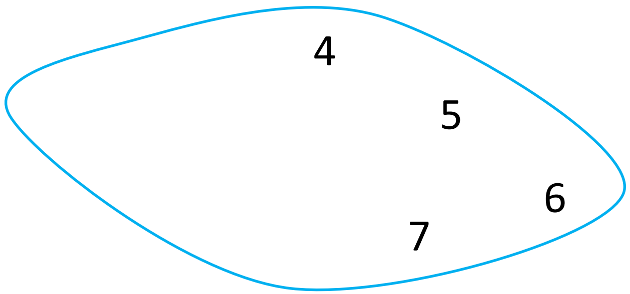


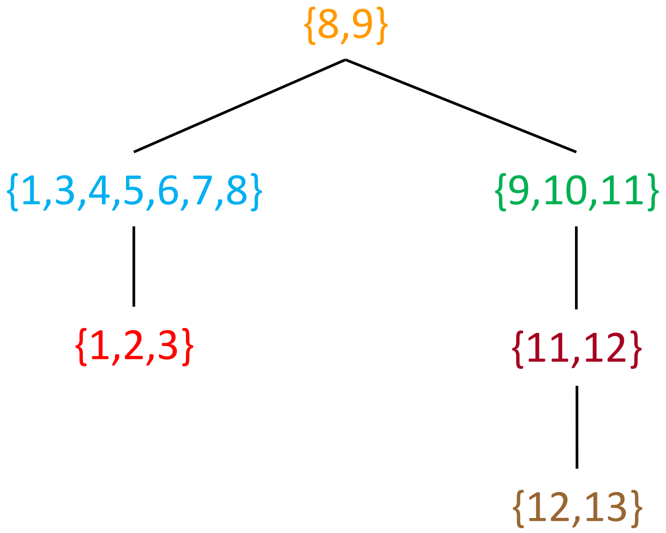
Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge

2. Eliminate hyperedges that are empty or contained in other hyperedges

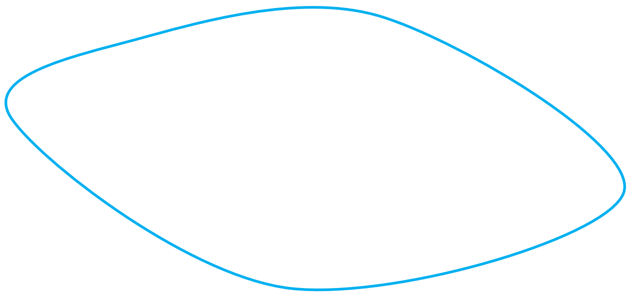


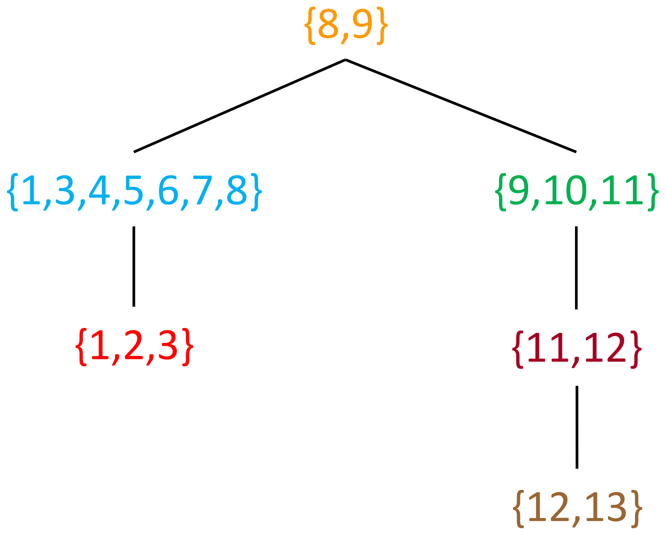


Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge
2. Eliminate hyperedges that are empty or contained in other hyperedges





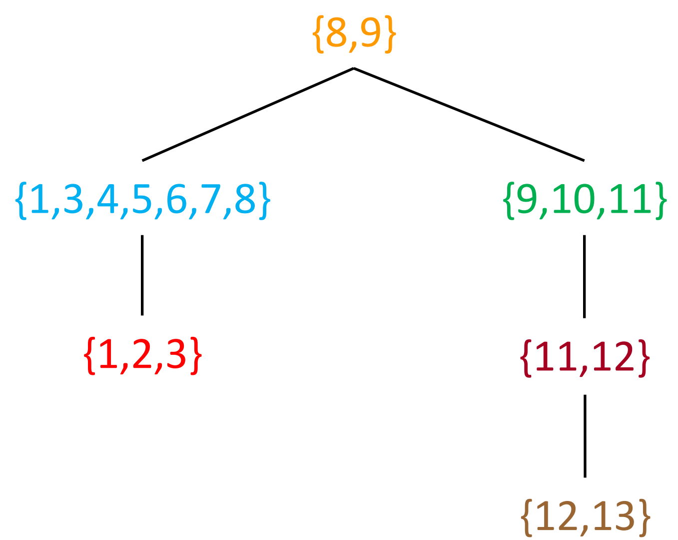
Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge

2. Eliminate hyperedges that are empty or contained in other hyperedges

(Ø,Ø)



Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge

2. Eliminate hyperedges that are empty or contained in other hyperedges

Theorem: A hypergraph **H** is acyclic iff GYO(**H**) = (∅,∅)

$\Downarrow$

checking whether **H** is acyclic is feasible in polynomial time, and if it is the case, a join tree can be found in polynomial time

$\Downarrow$

Theorem: ACYCLICITY is in PTIME

Checking Acyclicity

Theorem: ACYCLICITY is in PTIME

NOTE: actually, we can check whether a CQ is acyclic in time O(||Q||)

linear time in the size Q

Evaluating Acyclic CQs

Theorem: BQE(**ACQ**) is in PTIME

NOTE: actually, if H(Q) is acyclic, then Q can be evaluated in time $O(||D|| \cdot ||Q||)$

linear time in the size of D and Q

Yannakaki's Algorithm

Dynamic programming algorithm over the join tree

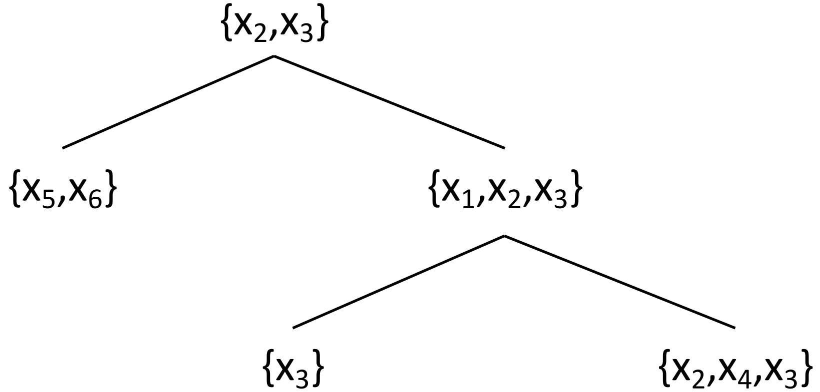
Given a database D, and an acyclic Boolean CQ Q

1. Compute the join tree **T** of H(Q)
2. Assign to each node of **T** the corresponding relation of D
3. Compute semi-joins in a bottom up traversal of **T**
4. Return YES if the resulting relation at the root of **T** is non-empty;

   otherwise, return NO

Yannakaki's Algorithm: Step 1

$Q \ \text{:-}\ R_1(x_1,x_2,x_3),\ R_2(x_2,x_3),\ R_2(x_5,x_6),\ R_3(x_3),\ R_4(x_2,x_4,x_3)$



Yannakaki's Algorithm: Step 2

x_2	x_3
c_1	b_2
c_1	b_1
c_4	b_6

x_5	x_6
c_1	b_2
c_1	b_1
c_4	b_6

x_1	x_2	x_3
s_1	c_1	b_1
s_1	c_1	b_2
s_3	c_3	b_1
s_3	c_1	b_4
s_2	c_2	b_3

x_3
b_1
b_2

x_2	x_4	x_3
c_1	a_1	b_1
c_1	a_1	b_2
c_1	a_2	b_2

Yannakaki's Algorithm: Step 3

x_2	x_3
c_1	b_2
c_1	b_1
c_4	b_6

x_5	x_6
c_1	b_2
c_1	b_1
c_4	b_6

x_1	x_2	x_3
s_1	c_1	b_1
s_1	c_1	b_2
s_3	c_3	b_1
$\cancel{s_3}$	$\cancel{c_1}$	$\cancel{b_4}$
s_2	c_2	b_3

x_3
b_1
b_2

x_2	x_4	x_3
c_1	a_1	b_1
c_1	a_1	b_2
c_1	a_2	b_2

Yannakaki’s Algorithm: Step 3

x_2	x_3
c_1	b_2
c_1	b_1
c_4	b_6

x_5	x_6
c_1	b_2
c_1	b_1
c_4	b_6

x_1	x_2	x_3
s_1	c_1	b_1
s_1	c_1	b_2
s_3	c_3	b_1
~~s_3~~	~~c_1~~	~~b_4~~
~~s_2~~	~~c_2~~	~~b_3~~

x_3
b_1
b_2

x_2	x_4	x_3
c_1	a_1	b_1
c_1	a_1	b_2
c_1	a_2	b_2

Yannakaki's Algorithm: Step 3

x_2	x_3
c_1	b_2
c_1	b_1
c_4	b_6

x_5	x_6
c_1	b_2
c_1	b_1
c_4	b_6

x_1	x_2	x_3
s_1	c_1	b_1
s_1	c_1	b_2
$\cancel{s_3}$	$\cancel{c_3}$	$\cancel{b_1}$
$\cancel{s_3}$	$\cancel{c_1}$	$\cancel{b_4}$
$\cancel{s_2}$	$\cancel{c_2}$	$\cancel{b_3}$

x_3
b_1
b_2

x_2	x_4	x_3
c_1	a_1	b_1
c_1	a_1	b_2
c_1	a_2	b_2

Yannakaki's Algorithm: Step 3

x_2	x_3
c_1	b_2
c_1	b_1
$\cancel{c_4}$	$\cancel{b_6}$

x_5	x_6
c_1	b_2
c_1	b_1
c_4	b_6

x_1	x_2	x_3
s_1	c_1	b_1
s_1	c_1	b_2
$\cancel{s_3}$	$\cancel{c_3}$	$\cancel{b_1}$
$\cancel{s_3}$	$\cancel{c_1}$	$\cancel{b_4}$
$\cancel{s_2}$	$\cancel{c_2}$	$\cancel{b_3}$

x_3
b_1
b_2

x_2	x_4	x_3
c_1	a_1	b_1
c_1	a_1	b_2
c_1	a_2	b_2

Yannakaki’s Algorithm: Step 4

x_2	x_3
c_1	b_2
c_1	b_1
~~c_4~~	~~b_6~~

YES

x_5	x_6
c_1	b_2
c_1	b_1
c_4	b_6

x_1	x_2	x_3
s_1	c_1	b_1
s_1	c_1	b_2
~~s_3~~	~~c_3~~	~~b_1~~
~~s_3~~	~~c_1~~	~~b_4~~
~~s_2~~	~~c_2~~	~~b_3~~

x_3
b_1
b_2

x_2	x_4	x_3
c_1	a_1	b_1
c_1	a_1	b_2
c_1	a_2	b_2

Acyclic CQs: Recap

ACYCLICITY

Input: a query Q ∈ **CQ**

Question: is Q acyclic? or is H(Q) acyclic?

BQE(**ACQ**)

Input: a database D, a Boolean query Q ∈ **ACQ**

Question: is Q(D) non-empty?

both problems are feasible in linear time

Query Optimization

Replace a given CQ with one that is much faster to execute

or

Replace a given CQ with one that falls in a “good” class of CQs

preferably, with an acyclic CQ

since evaluation is in linear time

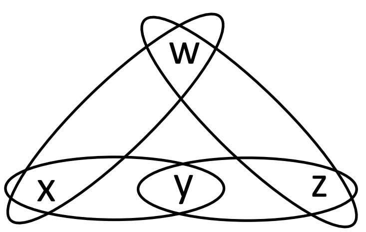
Semantic Acyclicity

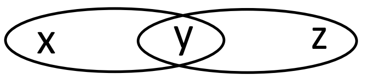
Definition: A CQ Q is semantically acyclic if there exists an acyclic CQ Q′ such that Q ≡ Q′

Q(x,z) :- R(x,y), R(y,z), R(x,w), R(w,z)

$\{w \mapsto y, z \mapsto y\}$

Q(x,z) :- R(x,y), R(y,z)





Relevant Algorithmic Tasks

SemACYCLICITY

Input: a query Q ∈ **CQ**

Question: is there an acyclic CQ Q’ such that Q ≡ Q’?

{Q ∈ **CQ** | Q semantically acyclic}

BQE(**SACQ**)

Input: a database D, a Boolean query Q ∈ **SACQ**

Question: is Q(D) non-empty?

Checking Semantic Acyclicity

Theorem: A CQ Q is semantically acyclic iff its core is acyclic

Theorem: SemACYCLICITY is NP-complete

Proof idea (upper bound):

- If Q is semantically acyclic, then there exists an acyclic CQ Q' such that $|Q'| \leq |Q|$ and $Q \equiv Q'$ (why?)
- Then, we can guess in polynomial time:
  - An acyclic CQ Q' such that $|Q'| \leq |Q|$
  - A mapping $h_1 : \text{terms}(Q) \rightarrow \text{terms}(Q')$
  - A mapping $h_2 : \text{terms}(Q') \rightarrow \text{terms}(Q)$
- And verify in polynomial time that h_1 is a query homomorphism from Q to Q' (i.e., $Q' \subseteq Q$), and h_2 is a query homomorphism from Q' to Q (i.e., $Q \subseteq Q'$)

Evaluating Semantically Acyclic CQs

$$f(||Q||) \quad + \quad O(||D|| \cdot ||Q||)$$

compute the equivalent acyclic CQ

evaluate the equivalent acyclic CQ

an improvement compare to $||D||^{O(||Q||)}$ for evaluating arbitrary CQs

Theorem: BQE(**SACQ**) is fixed-parameter tractable

Evaluating Semantically Acyclic CQs

Theorem: BQE(**SACQ**) is in PTIME

assuming Q belongs to **SACQ**: Q(D) is non-empty $\Leftrightarrow$ Q $\rightarrow_{\exists 1c}$ D

the duplicator has a winning strategy

for the existential 1-cover game,

which can be checked in polynomial time

Semantically Acyclic CQs: Recap

SemACYCLICITY

Input: a query Q ∈ **CQ**

Question: is there an acyclic CQ Q′ such that Q ≡ Q′?

NP-complete - but no database is involved

BQE(**SACQ**)

Input: a database D, a Boolean query Q ∈ **SACQ**

Question: is Q(D) non-empty?

in PTIME (combined complexity)

Recap

- "Good" classes of CQs for which query evaluation is tractable - conditions based on the graph or hypergraph of the CQ

- Acyclic CQs - their hypergraph is acyclic, can be checked in linear time

- Evaluating acyclic CQs is feasible in linear time (Yannakaki's algorithm)

- Semantic acyclicity - difficult to check, but ensures tractable evaluation

MAI4CAREU | Master programmes in Artificial Intelligence 4 Careers in Europe

University of Cyprus

MAI649: PRINCIPLES OF ONTOLOGICAL DATABASES

Thank You!

Andreas Pieris

Spring 2022-2023

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

MAI4CAREU Master programmes in Artificial Intelligence 4 Careers in Europe

University of Cyprus

MAI649: PRINCIPLES OF ONTOLOGICAL DATABASES

Adding Recursion - Datalog

Andreas Pieris

Spring 2022-2023

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

Learning Outcomes

- Syntax and semantics of Datalog (CQs + recursion)

- Analyze the complexity of evaluating Datalog queries

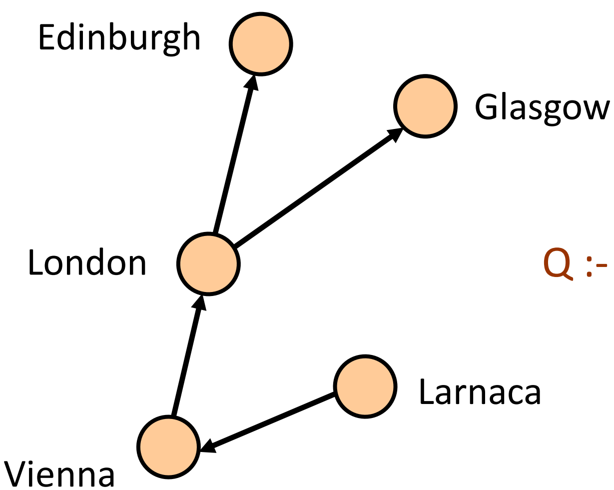
- Static analysis of Datalog queries

Limits of CQs

Is Glasgow reachable from Vienna?

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh



Q :- Airport(x,Vienna), Airport(y,Glasgow), Flight(x,z,w), Flight(z,y,v)

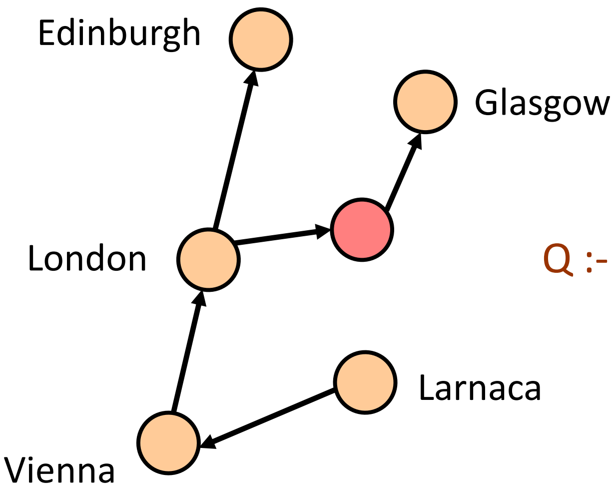
YES

Limits of CQs

Is Glasgow reachable from Vienna?

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh



Q :- Airport(x,Vienna), Airport(y,Glasgow), Flight(x,z,w), Flight(z,y,v)

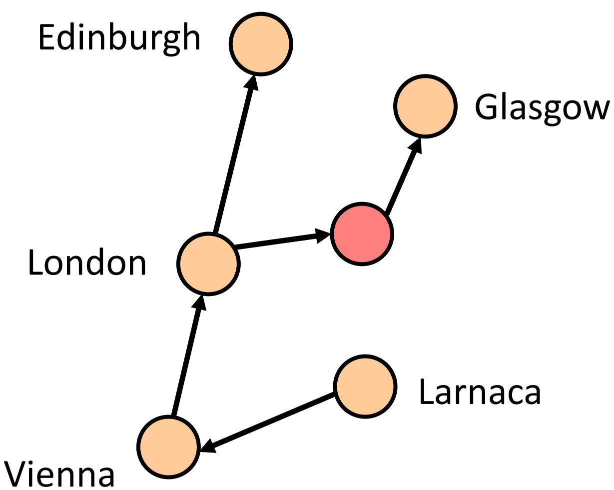
NO

Limits of CQs

Is Glasgow reachable from Vienna?

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh



Q :- Airport(x,Vienna), Airport(y,Glasgow), Flight(x,z,w), Flight(z,z_1,w_1), Flight(z,y,v)

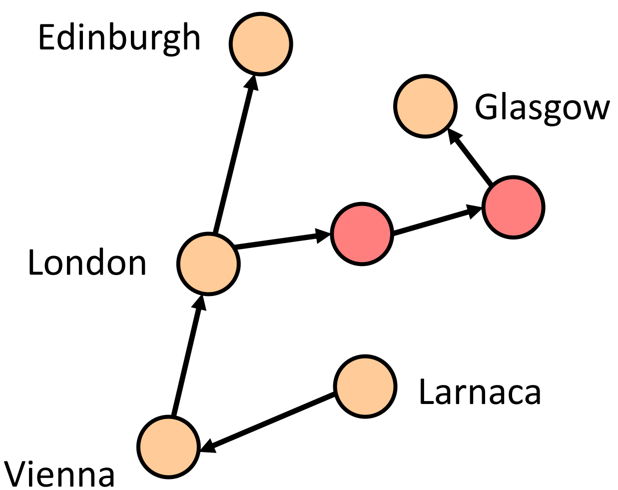
YES

Limits of CQs

Is Glasgow reachable from Vienna?

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh



Q :- Airport(x,Vienna), Airport(y,Glasgow), Flight(x,z,w), Flight(z,z_1,w_1), Flight(z,y,v)

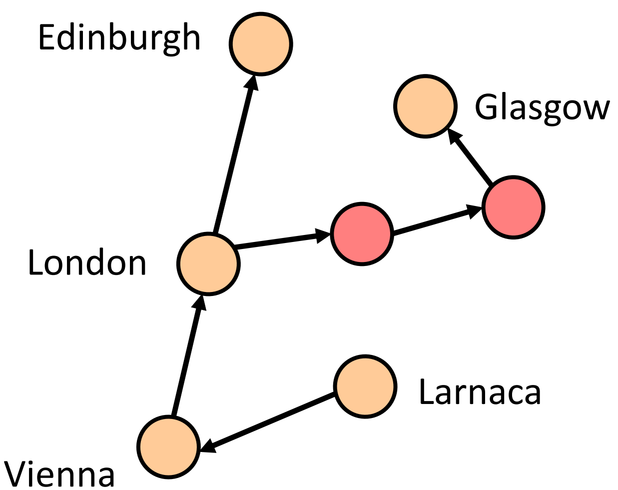
NO

Limits of CQs

Is Glasgow reachable from Vienna?

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh



Recursive query - not expressible in **CQ**

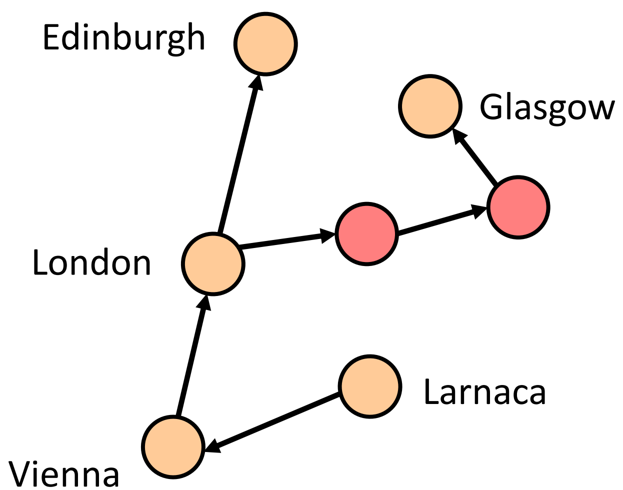
(or even in **RA** and **RC**)

A Possible Strategy

Is Glasgow reachable from Vienna?

Flight	origin	destination	airline
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	code	city
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh



- List all the pairs (a,b) such that b is reachable from a
- Check if there exists a pair (a,b) such that a is in Vienna and b is in Glasgow

A Possible Strategy

Is Glasgow reachable from Vienna?

Flight	origin	destination	airline

Airport	code	city

- List all the pairs (a,b) such that b is reachable from a

Reachable(x,y) :- Flight(x,y,z)

Reachable(x,w) :- Flight(x,y,z), Reachable(y,w)

- Check if there exists a pair (a,b) such that a is in Vienna and b is in Glasgow

Answer() :- Airport(x,Vienna), Airport(y,Glasgow), Reachable(x,y)

A Possible Strategy

Is Glasgow reachable from Vienna?

Flight	origin	destination	airline

Airport	code	city

- List all the pairs (a,b) such that b is reachable from a

Reachable(x,y) :- Flight(x,y,z)

Reachable(x,w) :- Flight(x,y,z), Reachable(y,w) - recursion

- Check if there exists a pair (a,b) such that a is in Vienna and b is in Glasgow

Answer() :- Airport(x,Vienna), Airport(y,Glasgow), Reachable(x,y)

A Possible Strategy

Is Glasgow reachable from Vienna?

Flight	origin	destination	airline

Airport	code	city

- List all the pairs (a,b) such that b is reachable from a

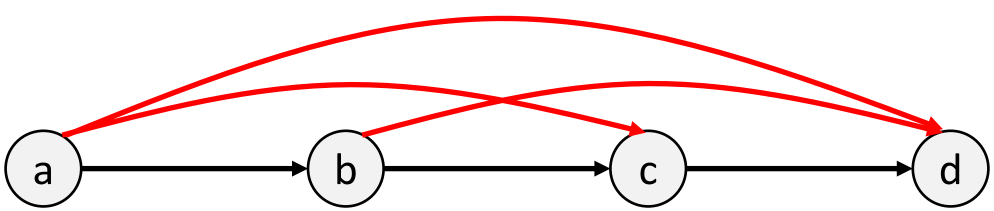
Reachable(x,y) :- Flight(x,y,z)

Reachable(x,w) :- Flight(x,y,z), Reachable(y,w) - recursion

DATALOG

Datalog at First Glance

Transitive closure of a graph



Datalog at First Glance

Edge	start	end
	a	b
	b	c
	c	d

TrClosure(x,y) :- Edge(x,y)

TrClosure(x,y) :- Edge(x,z), TrClosure(z,y)

Answer(x,y) :- TrClosure(x,y)

Answer	start	end
	a	b
	a	c
	a	d
	b	c
	b	d
	c	d

Datalog at First Glance

- **Semantics:** a mapping from databases of the **extensional** schema to databases of the **intensional** schema, and the answer is determined by the output relation

Edge	start	end
	a	b
	b	c
	c	d

Answer	start	end
	a	b
	a	c
	a	d
	b	c
	b	d
	c	d

- Equivalent ways for defining the semantics
  - **Model-theoretic:** logical sentences asserting a property of the result
  - **Fixpoint:** solution of a fixpoint procedure

Syntax of Datalog

A **Datalog rule** is an expression of the form

$$\underbrace{S(\mathbf{x})}_{\text{head}} \;\text{:-}\; \underbrace{R_1(\mathbf{x_1}),\ldots,R_n(\mathbf{x_n})}_{\text{body}}$$

- $n \geq 0$ (the body might be empty)
- $S, R_1, \ldots, R_n$ are relation names
- $\mathbf{x}, \mathbf{x_1}, \ldots, \mathbf{x_n}$ are tuples of variables
- each variable in the head occurs also in the body (**safety condition**)

Syntax of Datalog

- **Datalog program P:** a finite set of Datalog rules
- **Extensional relation:** does not occur in the head of a rule of P
- **Intensional relation:** occurs in the head of some rule of P
- EDB(P) is the set of extensional relations of P
- IDB(P) is the set of intensional relations of P

the **schema** of P

SCH(P) = EDB(P) ∪ IDB(P)

- **Datalog query Q:** a pair of the form (P, Answer), where P is a Datalog program, and Answer a distinguished intensional relation, the **output relation**

Example of Datalog

Is Glasgow reachable from Vienna?

Flight	origin	destination	airline

Airport	code	city

$$P = \left\{ \begin{array}{l} \text{Reachable}(x,y) \text{ :- } \text{Flight}(x,y,z) \\ \text{Reachable}(x,w) \text{ :- } \text{Flight}(x,y,z),\ \text{Reachable}(y,w) \\ \text{Answer}() \text{ :- } \text{Airport}(x,\text{Vienna}),\ \text{Airport}(y,\text{Glasgow}),\ \text{Reachable}(x,y) \end{array} \right\}$$

EDB(P) = {Flight, Airport}   IDB(P) = {Reachable, Answer}

Q = (P, Answer)

Semantics of Datalog

…it relies on the notion of immediate consequence operator

- Given a database D and a Datalog program P, an atom $R(a_1,\dots,a_n)$ is an **immediate consequence** for D and P if:
  - $R(a_1,\dots,a_n)$ belongs to D, or
  - There exists a rule $R(x_1,\dots,x_n)$:- body in P, and a homomorphism h from body to D such that $R(h(x_1),\dots,h(x_n)) = R(a_1,\dots,a_n)$

- $T_P(D) = \{R(a_1,\dots,a_n) \mid R(a_1,\dots,a_n)$ is an immediate consequence for D and P$\}$

- The immediate consequence operator T_P should be understood as a function from databases of SCH(P) to databases of SCH(P)

Semantics of Datalog

...it relies on the notion of immediate consequence operator

Theorem: For every Datalog program P and database D of EDB(P), the immediate consequence operator T_P has a minimum **fixpoint** containing D

a database D' is a fixpoint of T_P if $T_P(D') = D'$

the semantics of P on D, denoted P(D), is the minimum fixpoint of P containing D

for a Datalog query Q = (P, Answer), Q(D) = {t | Answer(t) ∈ P(D)}

...how do we compute P(D)?

Semantics of Datalog

…it relies on the notion of immediate consequence operator

$T_{P,0}(D) = D$ and $T_{P,i+1}(D) = T_P(T_{P,i}(D))$

$$\mathbf{T_{P,\infty}(D) = T_{P,0}(D) \cup T_{P,1}(D) \cup T_{P,2}(D) \cup T_{P,3}(D) \cup \cdots}$$

Semantics of Datalog: Example

…it relies on the notion of immediate consequence operator

D = {Edge(a,b), Edge(b,c), Edge(c,d)}

$$P = \left\{ \begin{array}{l} \text{TrClosure(x,y) :- Edge(x,y)} \\ \text{TrClosure(x,y) :- Edge(x,z), TrClosure(z,y)} \\ \text{Answer(x,y) :- TrClosure(x,y)} \end{array} \right\}$$

$T_{P,0}(D) = D$

$T_{P,1}(D) = T_P(T_{P,0}(D)) = D \cup$ {TrClosure(a,b), TrClosure(b,c), TrClosure(c,d)}

$T_{P,2}(D) = T_P(T_{P,1}(D)) = T_{P,1}(D) \cup$ {TrClosure(a,c), TrClosure(b,d), Answer(a,b), Answer(b,c), Answer(c,d)}

$T_{P,3}(D) = T_P(T_{P,2}(D)) = T_{P,2}(D) \cup$ {TrClosure(a,d), Answer(a,c), Answer(b,d)}

$T_{P,4}(D) = T_P(T_{P,3}(D)) = T_{P,3}(D) \cup$ {Answer(a,d)}

$T_{P,5}(D) = T_P(T_{P,4}(D)) = T_{P,4}(D)$

$\boxed{T_{P,\infty}(D) = T_{P,4}(D)}$

Semantics of Datalog

…it relies on the notion of immediate consequence operator

$$T_{P,0}(D) = D \qquad \text{and} \qquad T_{P,i+1}(D) = T_P(T_{P,i}(D))$$

$$\mathbf{T_{P,\infty}(D) = T_{P,0}(D) \cup T_{P,1}(D) \cup T_{P,2}(D) \cup T_{P,3}(D) \cup \cdots}$$

Theorem: For every Datalog program P and database D of EDB(P), $P(D) = T_{P,\infty}(D)$

Complexity of **DATALOG**

QOT(**DATALOG**)

Input: a database D, a Datalog query Q/k, a tuple of constants **t** ∈ **adom**(D)^k

Question: **t** ∈ Q(D)? (i.e., whether Answer(**t**) ∈ P(D))

Theorem: It holds that:

- QOT(**DATALOG**) is EXPTIME-complete (combined complexity)
- QOT[Q](**DATALOG**) is PTIME-complete, for a fixed Datalog query Q (data complexity)

Complexity of **DATALOG**

- Recall that $P(D) = T_{P,\infty}(D)$
- Computing $T_{P,i}(D)$ takes time

$$O(|P| \cdot |\mathbf{adom}(D)|^{\text{maxvar}} \cdot \text{maxbody} \cdot |T_{P,i-1}(D)|)$$

where <u>maxvar</u> is the maximum number of variables in a rule-body, and <u>maxbody</u> is the maximum number of atoms in a rule-body

- It is clear that $|T_{P,i-1}(D)| \leq |T_{P,\infty}(D)|$, and thus, computing $T_{P,i}(D)$ takes time

$$O(|P| \cdot |\mathbf{adom}(D)|^{\text{maxvar}} \cdot \text{maxbody} \cdot |T_{P,\infty}(D)|)$$

- Consequently, computing $T_{P,\infty}(D)$ takes time

$$O(|P| \cdot |\mathbf{adom}(D)|^{\text{maxvar}} \cdot \text{maxbody} \cdot |T_{P,\infty}(D)|^2)$$

- It is not difficult to verify that

$$|T_{P,\infty}(D)| \leq |SCH(P)| \cdot |\mathbf{adom}(D)|^{\text{maxarity}}$$

where <u>maxarity</u> is the maximum arity over all relations of $SCH(P)$

- Consequently, $T_{P,\infty}(D)$ can be computed in time

$$O(|P| \cdot |\mathbf{adom}(D)|^{\text{maxvar}} \cdot \text{maxbody} \cdot |SCH(P)|^2 \cdot |\mathbf{adom}(D)|^{2\text{maxarity}})$$

Complexity of **DATALOG**

QOT(**DATALOG**)

Input: a database D, a Datalog query Q/k, a tuple of constants **t** ∈ **adom**(D)^k

Question: **t** ∈ Q(D)? (i.e., whether Answer(**t**) ∈ P(D))

Theorem: It holds that:

- QOT(**DATALOG**) is EXPTIME-complete (combined complexity)
- QOT[Q](**DATALOG**) is PTIME-complete, for a fixed Datalog query Q (data complexity)

P(D) can be computed in time

$$O(|P| \cdot |\mathbf{adom}(D)|^{\text{maxvar}} \cdot \text{maxbody} \cdot |SCH(P)|^2 \cdot |\mathbf{adom}(D)|^{2\text{maxarity}})$$

What About Optimization of Datalog?

SAT(**DATALOG**)

Input: a query $Q \in$ **DATALOG**

Question: is there a (finite) database D such that $Q(D)$ is non-empty?

EQUIV(**DATALOG**)

Input: two queries $Q_1 \in$ **DATALOG** and $Q_2 \in$ **DATALOG**

Question: $Q_1 \equiv Q_2$? or $Q_1(D) = Q_2(D)$ for every database D?

CONT(**DATALOG**)

Input: two queries $Q_1 \in$ **DATALOG** and $Q_2 \in$ **DATALOG**

Question: $Q_1 \subseteq Q_2$? or $Q_1(D) \subseteq Q_2(D)$ for every database D?

What About Optimization of Datalog?

SAT(**DATALOG**)

Input: a query $Q \in$ **DATALOG**

Question: is there a (finite) database D such that $Q(D)$ is non-empty?

EQUIV(**DATALOG**)

Input: two queries $Q_1 \in$ **DATALOG** and $Q_2 \in$ **DATALOG**

Question: $Q_1 \equiv Q_2$? or $Q_1(D) = Q_2(D)$ for every database D?

UNDECIDABLE

CONT(**DATALOG**)

Input: two queries $Q_1 \in$ **DATALOG** and $Q_2 \in$ **DATALOG**

Question: $Q_1 \subseteq Q_2$? or $Q_1(D) \subseteq Q_2(D)$ for every database D?

What About Optimization of Datalog?

SAT(**DATALOG**)

Input: a query $Q \in$ **DATALOG**

Question: is there a (finite) database D such that $Q(D)$ is non-empty?

Theorem: SAT(**DATALOG**) is in EXPTIME

Lemma: Given a Datalog query $Q = (P, \text{Answer})$, Q is satisfiable iff $Q(D_P) \neq \emptyset$,

where $D_P = \{R(b_1,\ldots,b_m) \mid R \in \text{EDB}(P) \text{ and } b_i \in \{\star,a_1,\ldots,a_n\}\}$, with $a_1,\ldots,a_n$ being the

constants occurring in the rules of P, and $\star$ being a new constant not in $\{a_1,\ldots,a_n\}$

Recap

- Recursive queries are not expressible via relational algebra or calculus
- Adding recursion to CQs → Datalog
- Fixpoint semantics of Datalog based on the immediate consequence operator
- Evaluating Datalog queries is EXPTIME-complete in combined complexity and PTIME-complete in data complexity
- We can check for satisfiability of Datalog queries, but equivalence and containment are undecidable (perfect query optimization not possible)

MAI4CAREU | Master programmes in Artificial Intelligence 4 Careers in Europe

University of Cyprus

MAI649: PRINCIPLES OF ONTOLOGICAL DATABASES

Thank You!

Andreas Pieris

Spring 2022-2023

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

MAI4CAREU | Master programmes in Artificial Intelligence 4 Careers in Europe

University of Cyprus

MAI649: PRINCIPLES OF ONTOLOGICAL DATABASES

Ontological Databases

Andreas Pieris

Spring 2022-2023

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

Learning Outcomes

- Syntax and semantics of existential rules
- Ontological query answering and universal models
- Ontology-based data access

Querying Relational Databases

List the codes of teaching staff

Lecturer	id	Name
	1	Alice
	2	Bob
	3	Tom
	4	Mary

Course	code	organiser
	CS100	2
	CS200	1
	CS300	5

Q(x) :- TeachingStaff(x,y)

Querying Relational Databases

List the codes of teaching staff

Lecturer	id	Name
	1	Alice
	2	Bob
	3	Tom
	4	Mary

Course	code	organiser
	CS100	2
	CS200	1
	CS300	5

Lecturers are teaching staff

Course organisers are teaching staff

Q(x) :- TeachingStaff(x,y)

Querying Relational Databases

List the codes of teaching staff

Lecturer	id	Name
	1	Alice
	2	Bob
	3	Tom
	4	Mary

Course	code	organiser
	CS100	2
	CS200	1
	CS300	5

$\forall x \forall y\ (Lecturer(x,y) \rightarrow TeachingStaff(x,y))$

$\forall x \forall y\ (Course(x,y) \rightarrow \exists z\ TeachingStaff(y,z))$

$Q(x)\ \text{:-}\ TeachingStaff(x,y)$

Querying Relational Databases

List the codes of teaching staff

Lecturer	id	Name
	1	Alice
	2	Bob
	3	Tom
	4	Mary

Course	code	organiser
	CS100	2
	CS200	1
	CS300	5

{1, 2, 3, 4, 5}

$\forall x \forall y\, (\mathrm{Lecturer}(x,y) \rightarrow \mathrm{TeachingStaff}(x,y))$

$\forall x \forall y\, (\mathrm{Course}(x,y) \rightarrow \exists z\, \mathrm{TeachingStaff}(y,z))$

Q(x) :- TeachingStaff(x,y)

Some Terminology

- Our basic vocabulary:
  - A countable set **Const** of constants - domain of a database
  - A countable set **Nulls** of marked nulls - globally ∃-quantified variables
  - A countable set **Vars** of variables - used in rules and queries

- A term is a constant, marked null, or variable

- An atom has the form $R(t_1,\ldots,t_n)$ - R is an n-ary relation and t_i's are terms

- An instance is a (*possibly infinite*) set of atoms with constants and nulls

- A database is a finite instance with only constants

Syntax of Existential Rules

An existential rule is an expression

$$\forall \mathbf{x} \forall \mathbf{y} \, (\underbrace{\varphi(\mathbf{x},\mathbf{y})}_{\text{body}} \rightarrow \exists \mathbf{z} \, \underbrace{\psi(\mathbf{x},\mathbf{z})}_{\text{head}})$$

- $\mathbf{x}$,$\mathbf{y}$ and $\mathbf{z}$ are tuples of variables of **Vars**
- $\varphi(\mathbf{x},\mathbf{y})$ and $\psi(\mathbf{x},\mathbf{z})$ are (constant-free) conjunctions of atoms

…also known as tuple-generating dependencies

Semantics of Existential Rules

- An instance J is a model of the rule

$$\sigma = \forall \mathbf{x} \forall \mathbf{y} \, (\varphi(\mathbf{x},\mathbf{y}) \rightarrow \exists \mathbf{z} \, \psi(\mathbf{x},\mathbf{z}))$$

written as $J \models \sigma$, if the following holds:

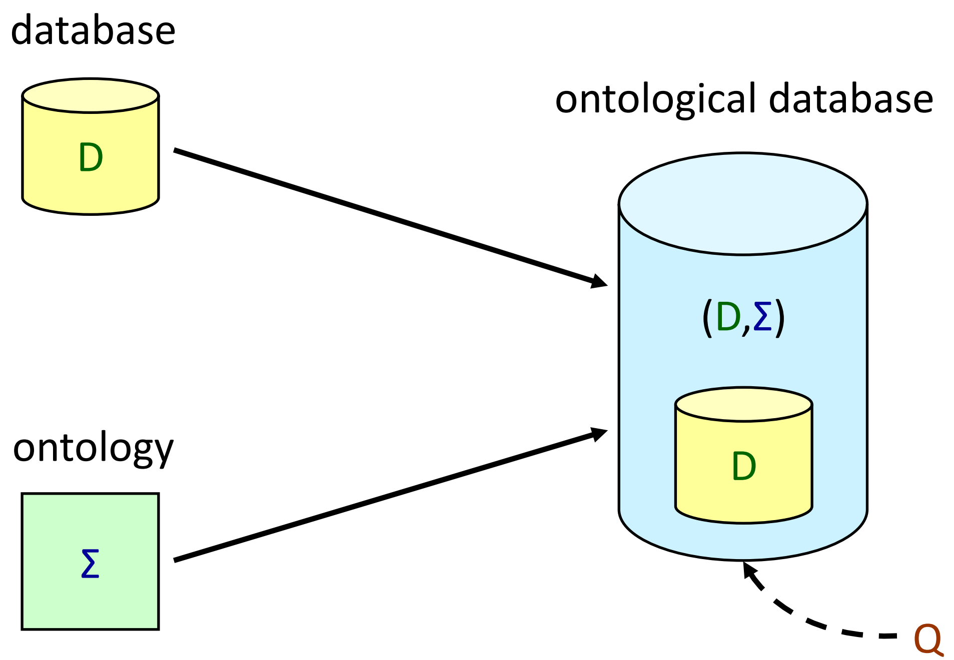
whenever there exists a homomorphism h such that $h(\varphi(\mathbf{x},\mathbf{y})) \subseteq J$,

then there exists $g \supseteq h_{|\mathbf{x}}$ such that $g(\psi(\mathbf{x},\mathbf{z})) \subseteq J$

$\{t \mapsto h(t) \mid t \in \mathbf{x}\}$ - the restriction of h to **x**

- Given a set Σ of existential rules, J is a model of Σ, written as $J \models \Sigma$, if, for each $\sigma \in \Sigma$, $J \models \sigma$

Ontological Query Answering (OQA)



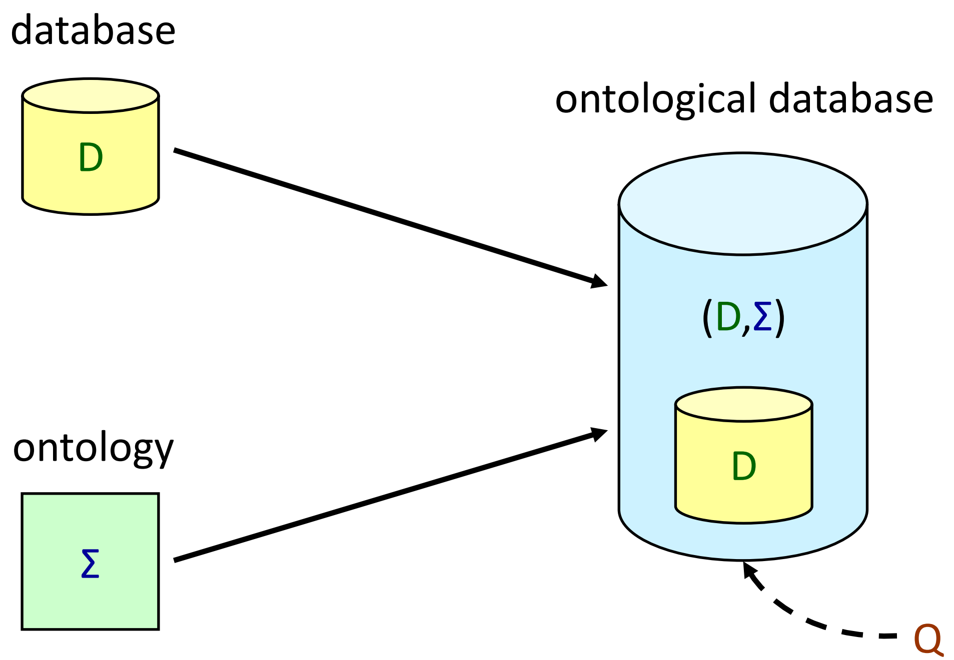
existential rules

$\forall \mathbf{x} \forall \mathbf{y} \, (\varphi(\mathbf{x},\mathbf{y}) \rightarrow \exists \mathbf{z} \, \psi(\mathbf{x},\mathbf{z}))$

conjunctive query

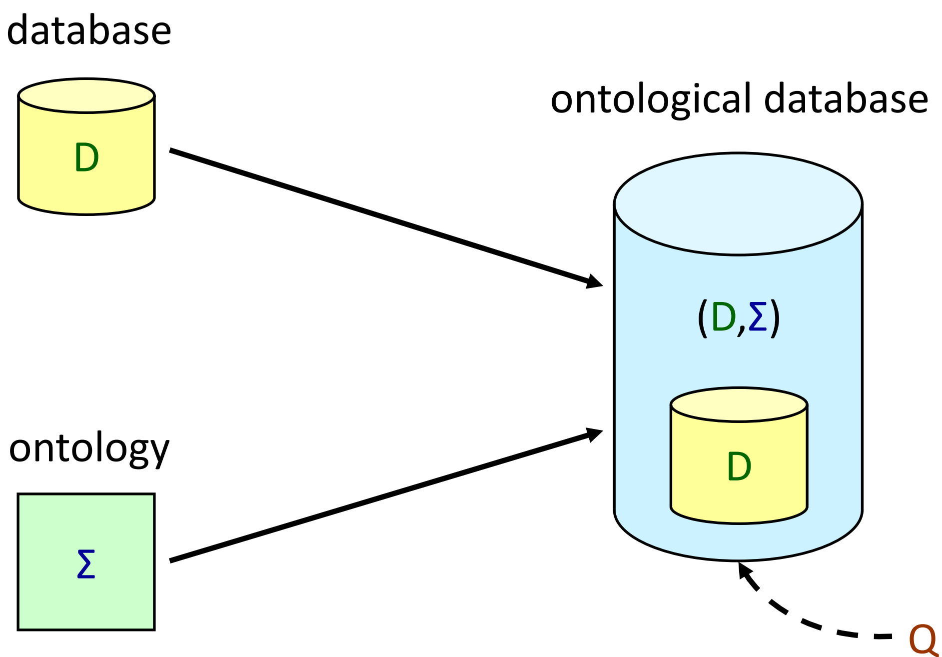
$Q(\mathbf{x}) \ :- \ R_1(\mathbf{v_1}),\ldots,R_m(\mathbf{v_m})$

Ontological Query Answering (OQA)



$$\text{models}(D,\Sigma) \;=\; \{J \mid J \supseteq D \ \text{ and } \ J \models \Sigma\}$$

Ontological Query Answering (OQA)



$$\text{Answer}(Q,D,\Sigma) \quad = \quad \bigcap_{J \,\in\, \text{models}(D,\Sigma)} \quad Q(J)$$

Exercise: Compute the Certain Answers

D = {Person(john), Person(bob), Person(tom),

hasFather(john,bob), hasFather(bob,tom)}

Σ = {$\forall x$ (Person(x) $\rightarrow$ $\exists y$ hasFather(x,y)),

$\forall x \forall y$ (hasFather(x,y) $\rightarrow$ Person(x) $\wedge$ Person(y))}

Q_1(x,y) :- hasFather(x,y)

Q_2(x) :- hasFather(x,y)

Q_3(x) :- hasFather(x,y), hasFather(y,z), hasFather(z,w)

Q_4(x,w) :- hasFather(x,y), hasFather(y,z), hasFather(z,w)

Exercise: Compute the Certain Answers

D = {Person(john), Person(bob), Person(tom),

hasFather(john,bob), hasFather(bob,tom)}

Σ = $\{\forall x\ (\text{Person}(x) \rightarrow \exists y\ \text{hasFather}(x,y)),$

$\forall x \forall y\ (\text{hasFather}(x,y) \rightarrow \text{Person}(x) \wedge \text{Person}(y))\}$

$Q_1(x,y)$:- hasFather(x,y)

{(john,bob), (bob,tom)}

Exercise: Compute the Certain Answers

D = {Person(john), Person(bob), Person(tom),

hasFather(john,bob), hasFather(bob,tom)}

Σ = $\{\forall x\ (\text{Person}(x) \rightarrow \exists y\ \text{hasFather}(x,y)),$

$\forall x \forall y\ (\text{hasFather}(x,y) \rightarrow \text{Person}(x) \land \text{Person}(y))\}$

$Q_2(x)$:- hasFather(x,y)

{(john), (bob), (tom)}

Exercise: Compute the Certain Answers

$D = \{Person(john), Person(bob), Person(tom),$
$hasFather(john,bob), hasFather(bob,tom)\}$

$\Sigma = \{\forall x\, (Person(x) \rightarrow \exists y\, hasFather(x,y)),$
$\forall x \forall y\, (hasFather(x,y) \rightarrow Person(x) \land Person(y))\}$

$Q_3(x)$:- $hasFather(x,y), hasFather(y,z), hasFather(z,w)$

$\{(john), (bob), (tom)\}$

Exercise: Compute the Certain Answers

D = {Person(john), Person(bob), Person(tom),

hasFather(john,bob), hasFather(bob,tom)}

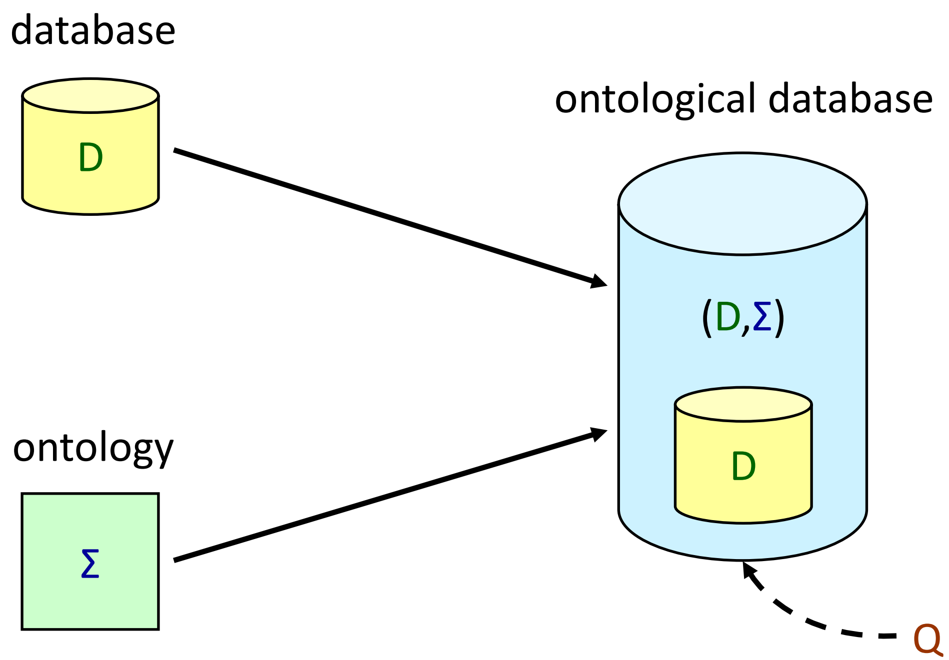
Σ = {$\forall x$ (Person(x) $\rightarrow$ $\exists y$ hasFather(x,y)),

$\forall x \forall y$ (hasFather(x,y) $\rightarrow$ Person(x) $\wedge$ Person(y))}

Q_4(x,w) :- hasFather(x,y), hasFather(y,z), hasFather(z,w)

{ }

Ontological Query Answering (OQA)



$$\text{Answer}(Q,D,\Sigma) \quad = \quad \bigcap_{J \,\in\, \text{models}(D,\Sigma)} Q(J)$$

keep only tuples with constants

Ontological Query Answering (OQA)

ontology language based on existential rules

OQA(**L**)

Input: a database D, a set of existential rules $\Sigma \in$ **L**, a CQ Q/k, a tuple of constants **t** $\in$ **adom**(D)^k

Question: **t** $\in$ Answer(Q,D,Σ)?

BOQA(**L**)

Input: a database D, a set of existential rules $\Sigma \in$ **L**, a Boolean query Q

Question: is Answer(Q,D,Σ) non-empty?

Theorem: OQA(**L**) $\equiv_L$ BOQA(**L**) for every language **L**

($\equiv_L$ means logspace-equivalent)

Data Complexity of BOQA

input D, fixed Σ and Q

BOQA[Σ,Q](**L**)

Input: a database D

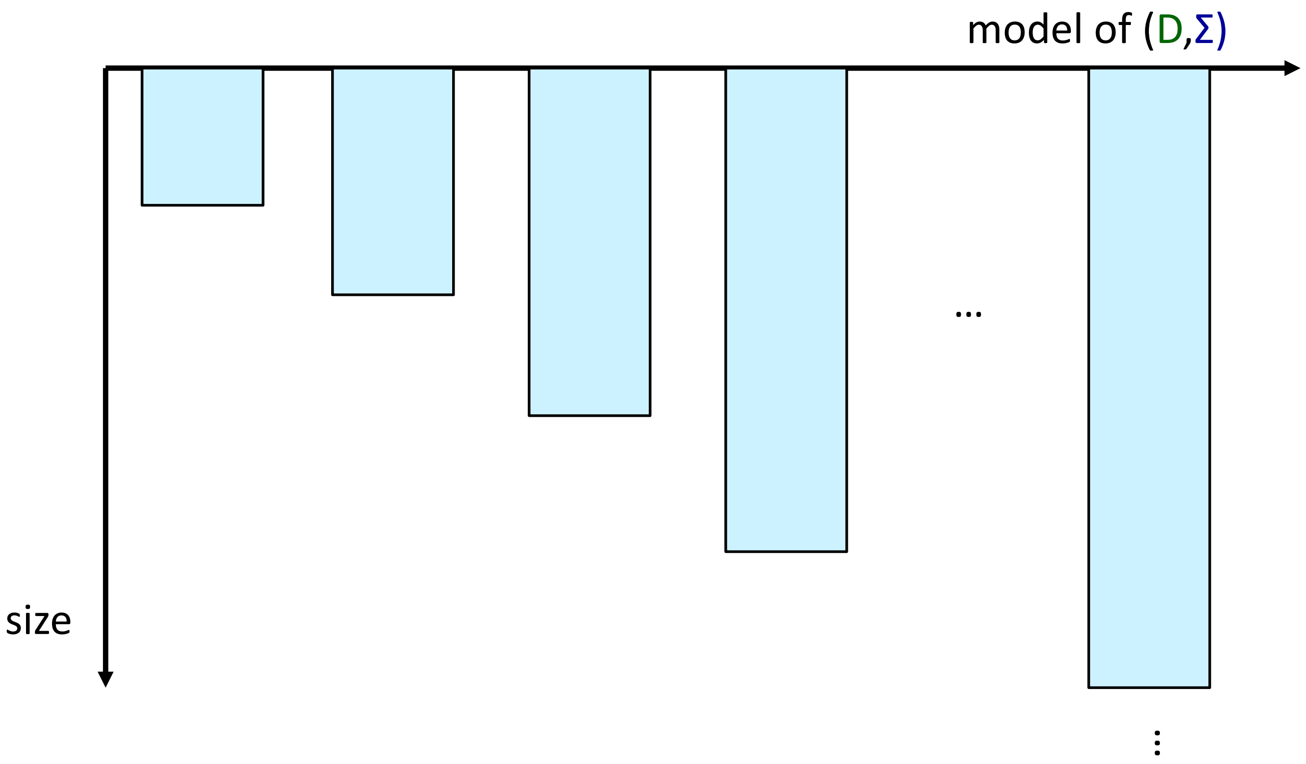
Question: is Answer(Q,D,Σ) non-empty?

Why is OQA technically challenging?

What is the right tool for tackling this problem?

The Two Dimensions of Infinity

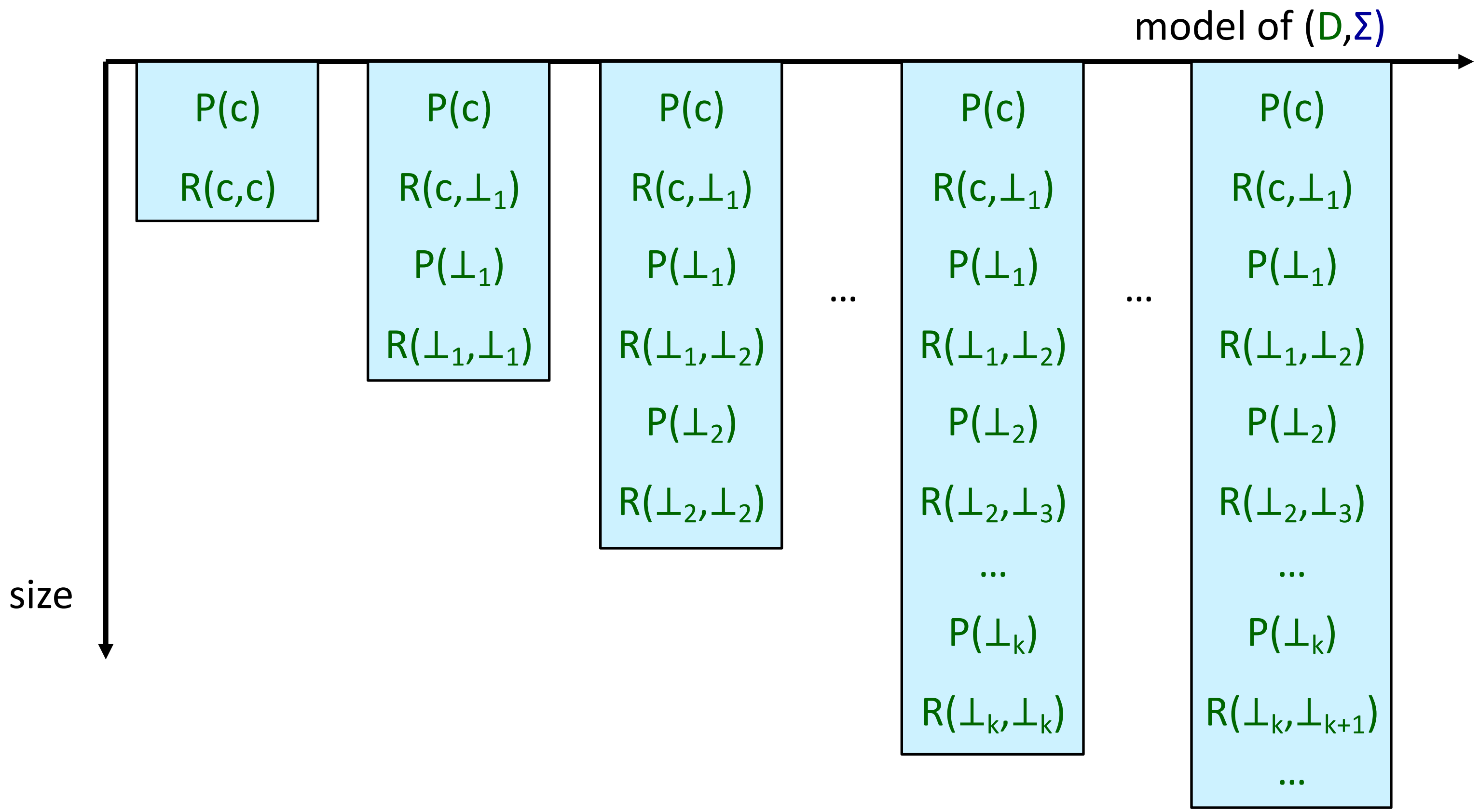
Consider a database D, and a set of existential rules Σ



(D,Σ) admits infinitely many models, of possibly infinite size

The Two Dimensions of Infinity

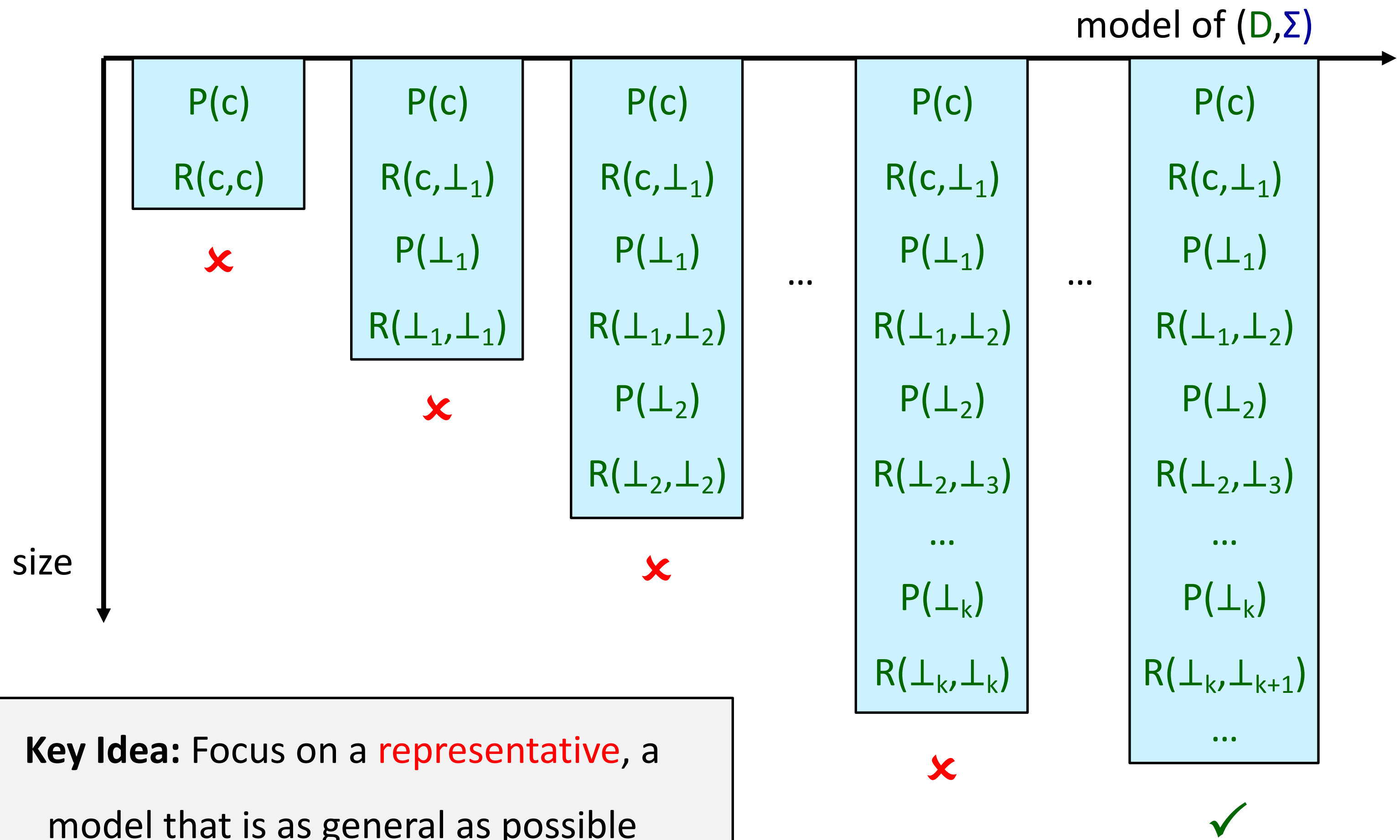
$D \ = \ \{P(c)\}$ $\qquad\qquad$ $\Sigma = \{\forall x\, (P(x) \rightarrow \exists y\, (R(x,y) \wedge P(y)))\}$



$\perp_1, \perp_2, \perp_3, \ldots$ are marked nulls from **Nulls**

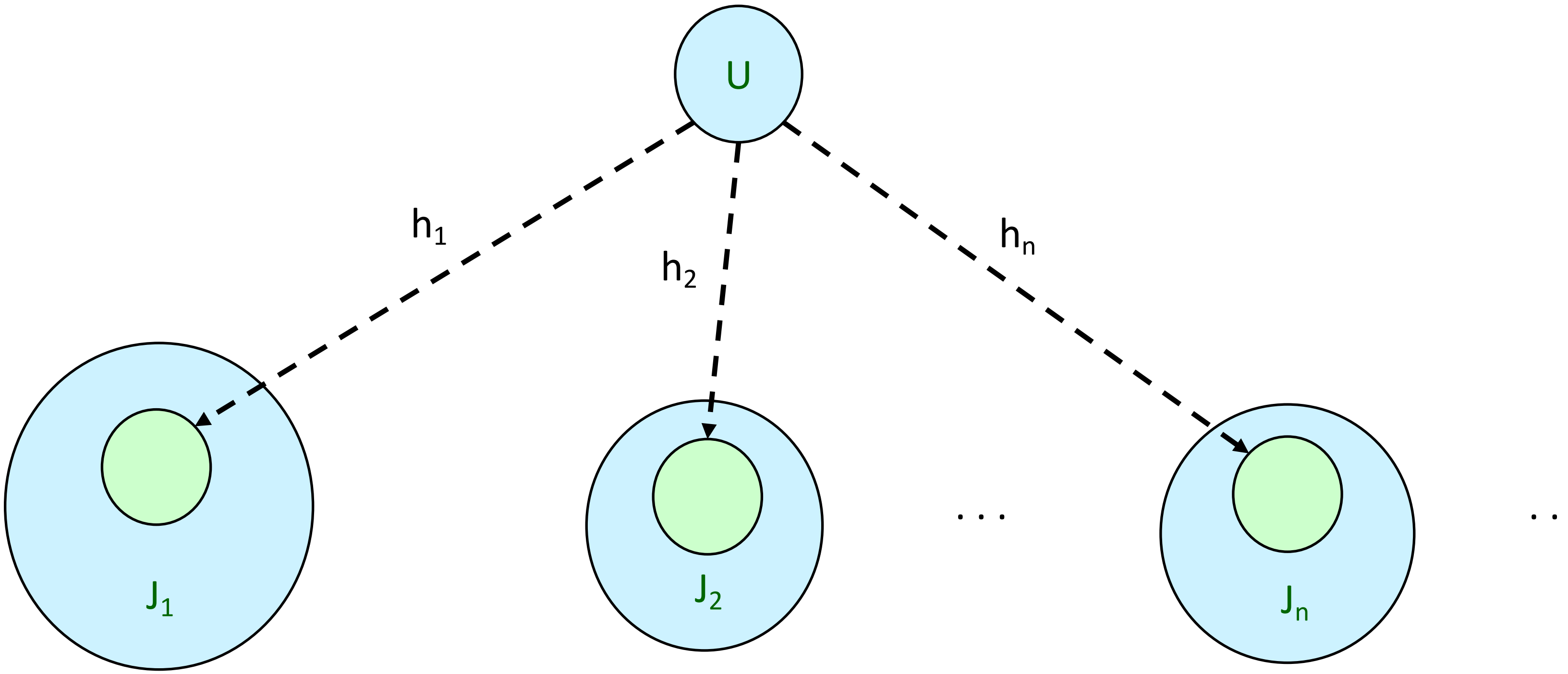
The Two Dimensions of Infinity

$D = \{P(c)\}$ $\Sigma = \{\forall x\, (P(x) \rightarrow \exists y\, (R(x,y) \wedge P(y)))\}$



Key Idea: Focus on a representative, a model that is as general as possible

Universal Models (a.k.a. Canonical Models)



An instance U is a universal model of (D,Σ) if the following holds:

1. U is a model of (D,Σ)

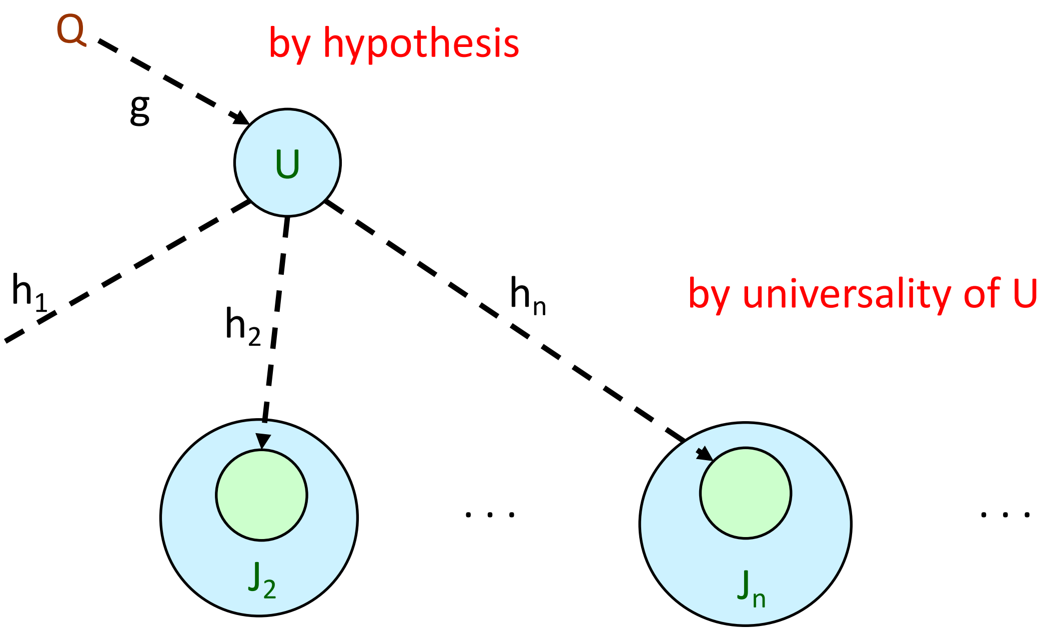
2. for each J ∈ models(D,Σ), there exists a homomorphism h such that h(U) ⊆ J

Query Answering via Universal Models

Theorem: Answer(Q,D,Σ) is non-empty iff Q(U) is non-empty, where U a universal model of (D,Σ)

Proof: (⇒) Trivial since, for every J ∈ models(D,Σ), Q(J) is non-empty

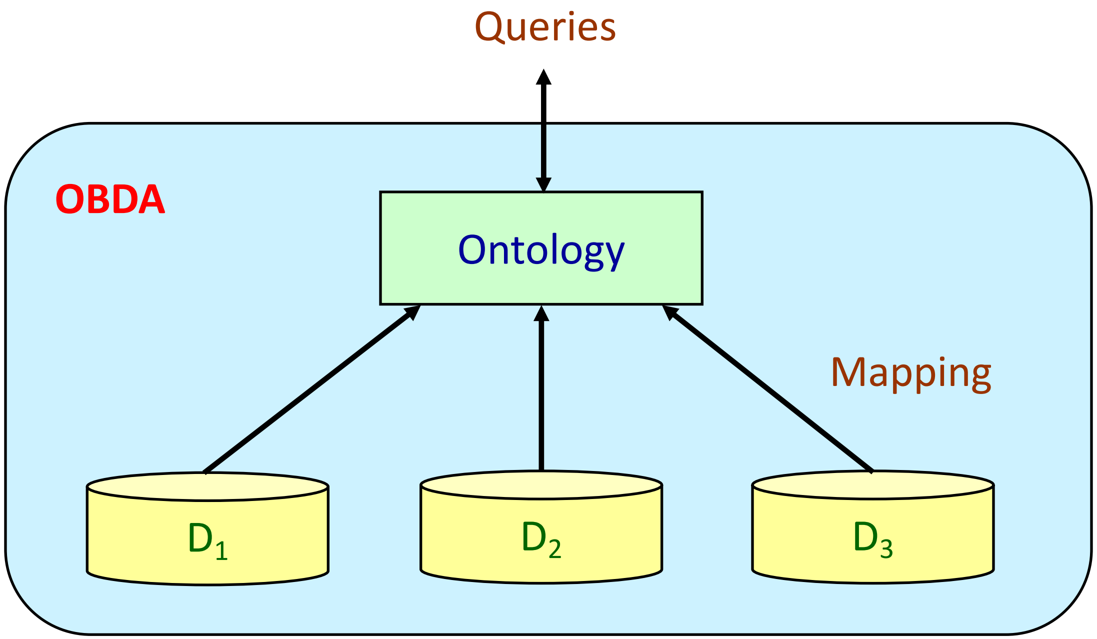
(⇐) By exploiting the universality of U



∀J ∈ models(D,Σ), ∃h such that h(g(Q)) ⊆ J ⇒ ∀J ∈ models(D,Σ), Q(J) is non-empty

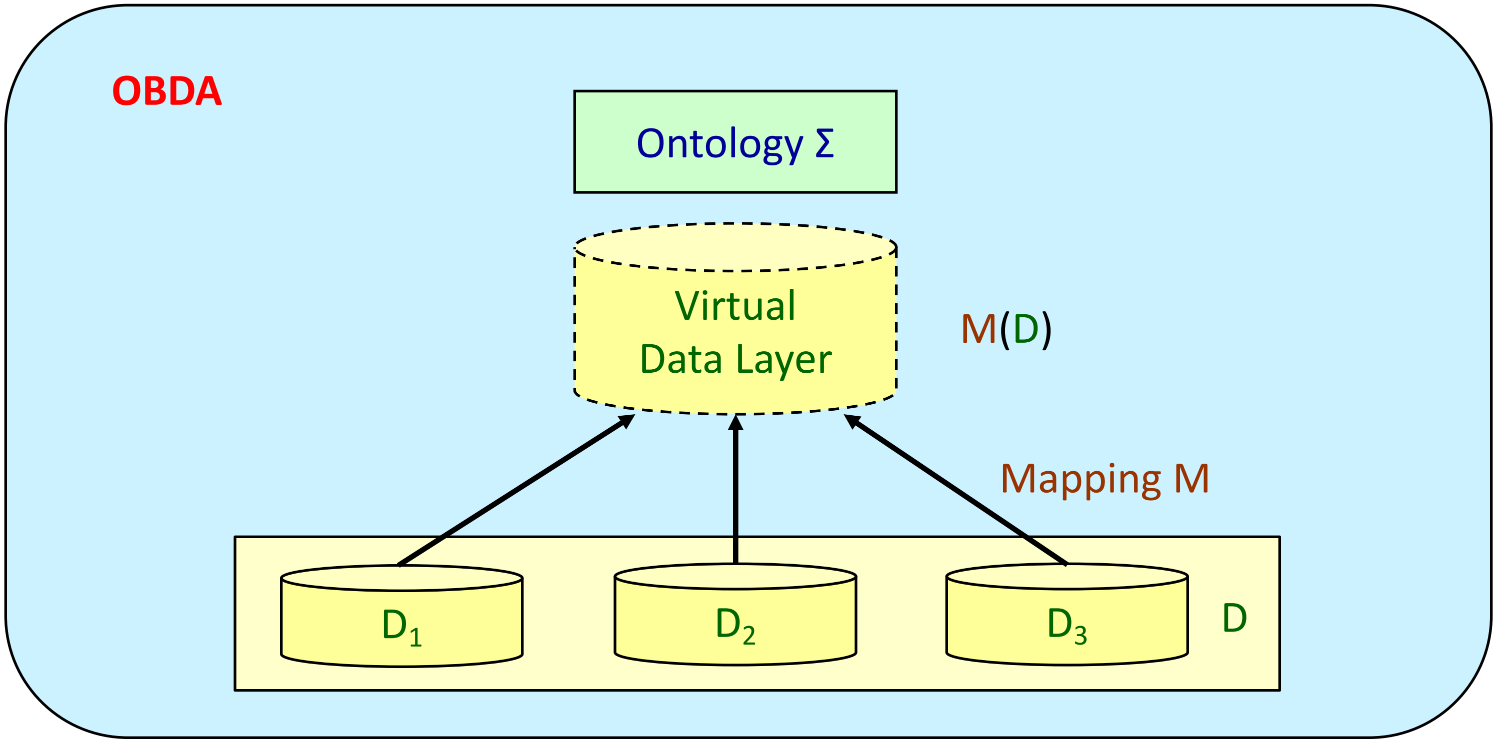
⇒ Answer(Q,D,Σ) is non-empty

Ontology-based Data Access: Architecture



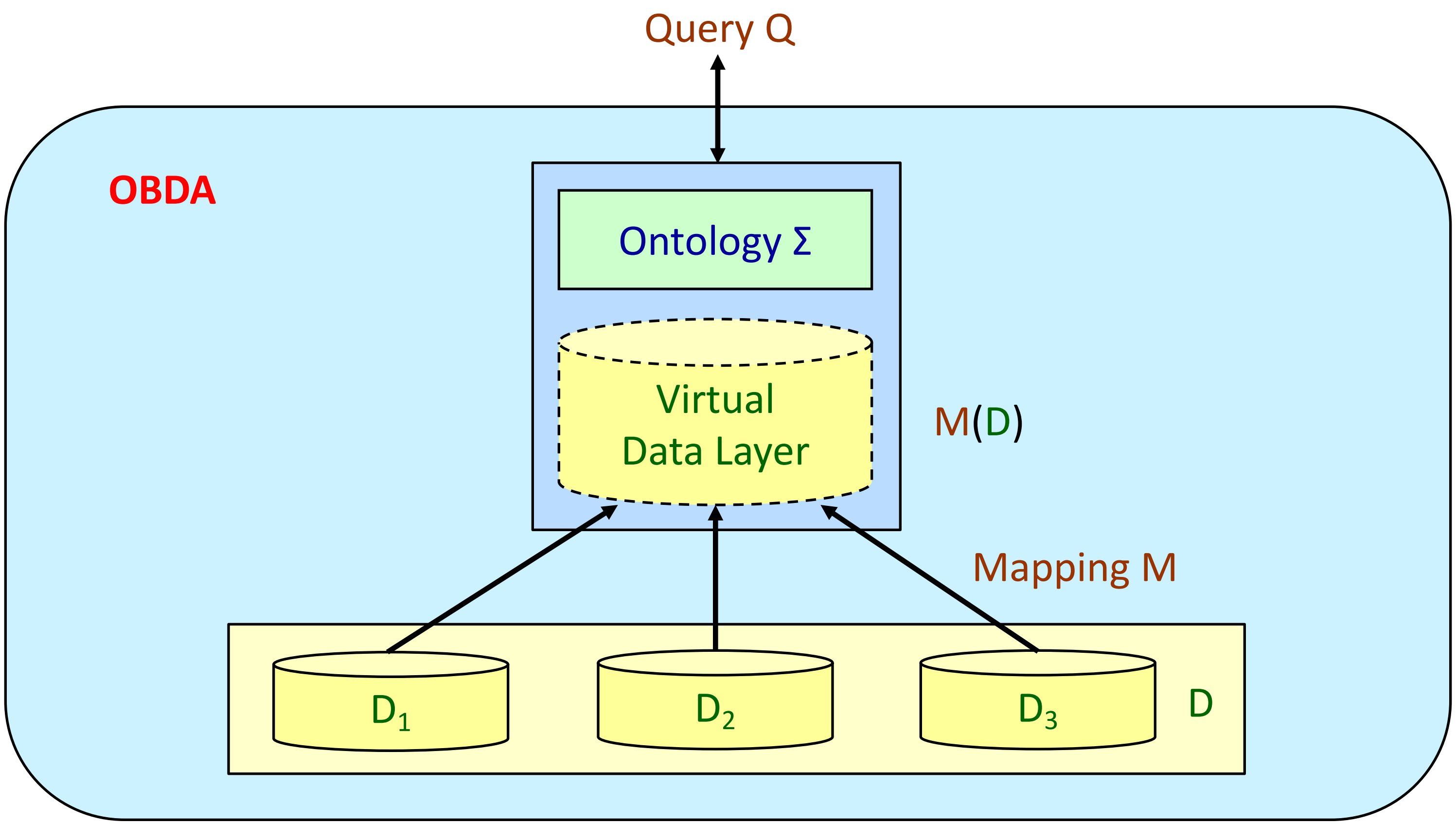
- Ontology: provides a unified conceptual "global view" of the data
- Data Sources: external and independent (possibly multiple and heterogeneous)
- Mapping: semantically link data at the sources with the ontology

Query Answering in OBDA



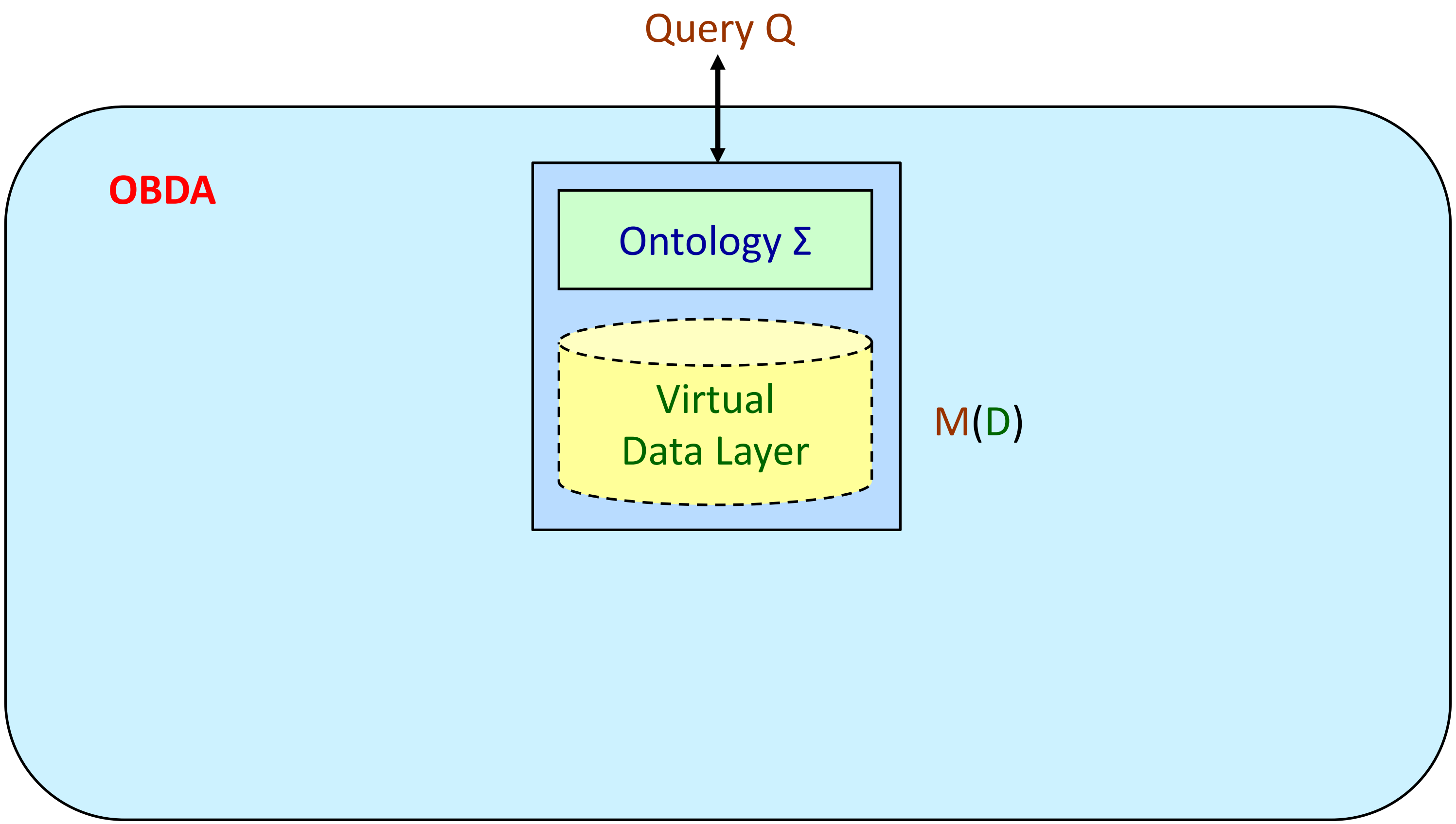
- The sources and the mapping define a virtual data layer M(D)

Query Answering in OBDA



- The sources and the mapping define a virtual data layer M(D)
- Queries are answered against the ontological database (M(D), Σ)

Query Answering in OBDA



Ontological Query Answering

MAI4CAREU

Master programmes in Artificial Intelligence 4 Careers in Europe

University of Cyprus

MAI649: PRINCIPLES OF ONTOLOGICAL DATABASES

Thank You!

Andreas Pieris

Spring 2022-2023

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

MAI4CAREU | Master programmes in Artificial Intelligence 4 Careers in Europe

University of Cyprus

MAI649: PRINCIPLES OF ONTOLOGICAL DATABASES

Ontological Query Answering

Andreas Pieris

Spring 2022-2023

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action
No 2020-EU-IA-0087, co-financed by the EU CEF Telecom
under GA nr. INEA/CEF/ICT/A2020/2267423

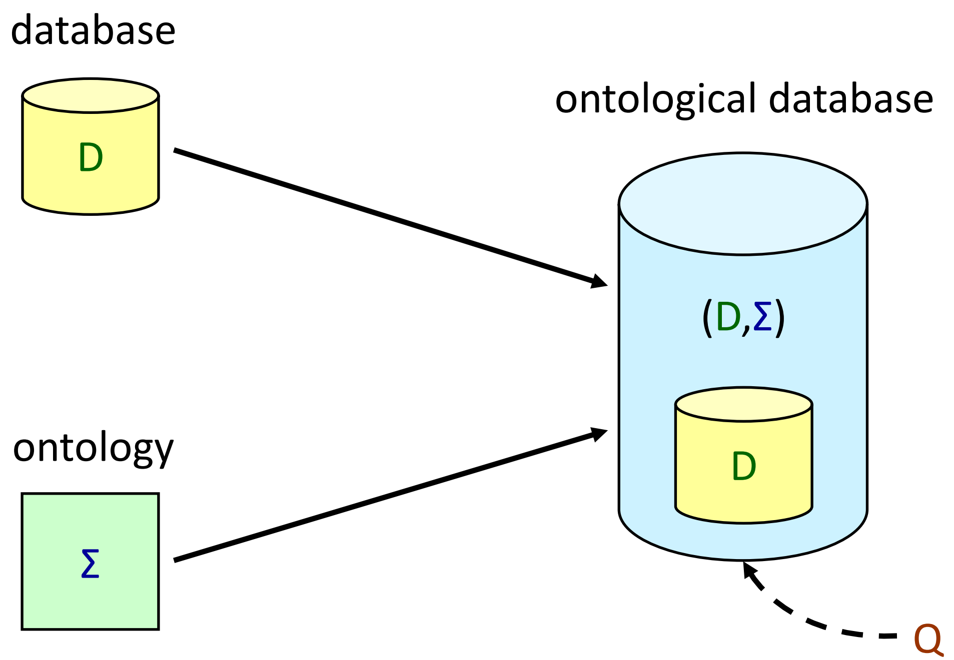
Learning Outcomes

- Ontological query answering via the chase procedure - forward-chaining

- Ontological query answering via query rewriting - backward-chaining

- Linear existential rules

Ontological Query Answering (OQA)



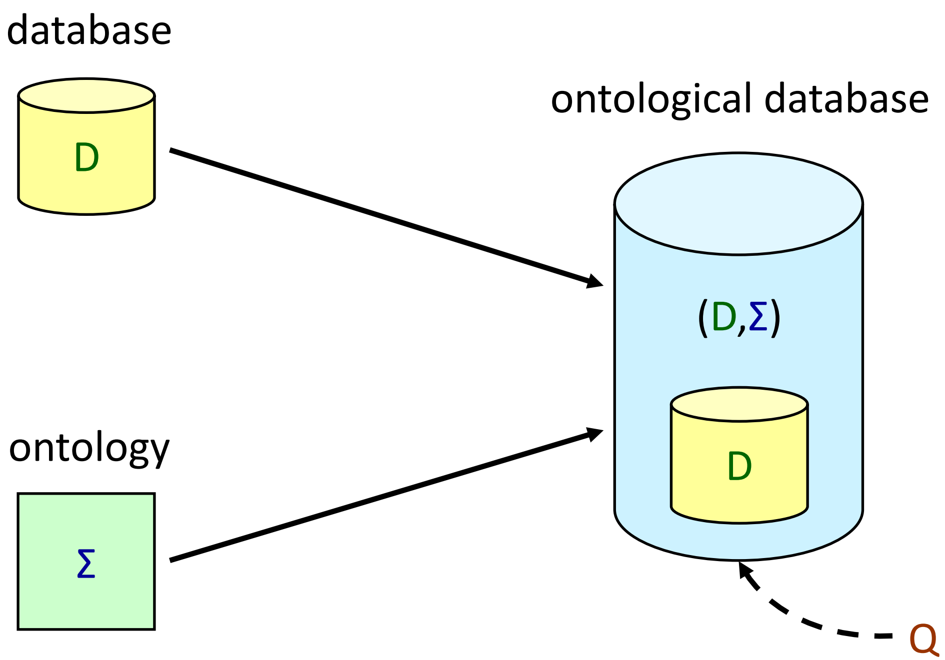
existential rules

$\forall \mathbf{x} \forall \mathbf{y}\ (\varphi(\mathbf{x},\mathbf{y}) \rightarrow \exists \mathbf{z}\ \psi(\mathbf{x},\mathbf{z}))$

conjunctive query

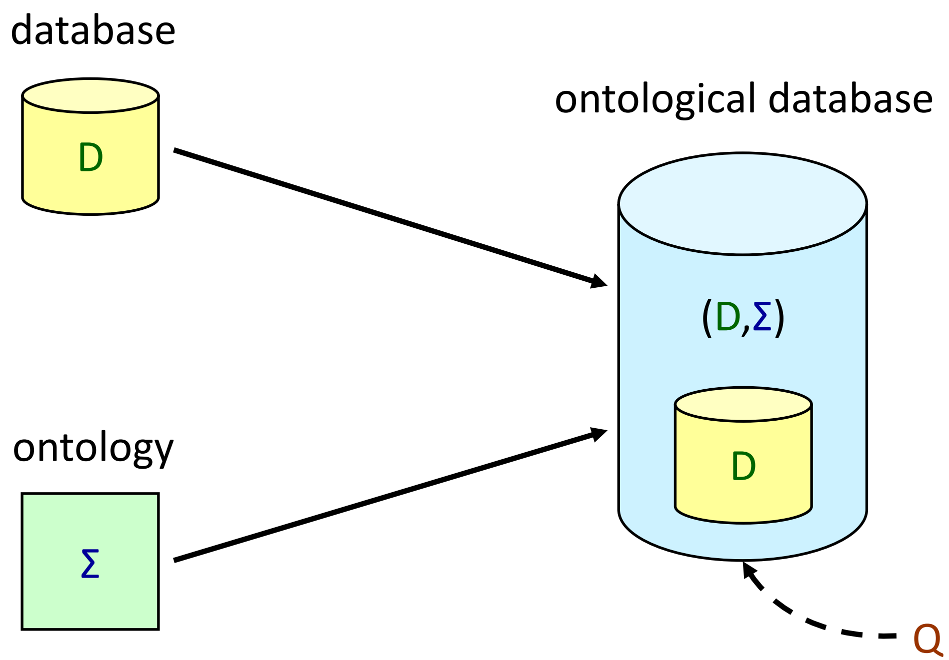
$Q(\mathbf{x})\ \ \text{:-}\ \ R_1(\mathbf{v_1}),\ldots,R_m(\mathbf{v_m})$

Ontological Query Answering (OQA)



models(D,Σ) = {J | J ⊇ D and J ⊨ Σ}

Ontological Query Answering (OQA)



$$\mathrm{Answer}(Q,D,\Sigma) \quad = \quad \bigcap_{J \,\in\, \mathrm{models}(D,\Sigma)} Q(J)$$

Ontological Query Answering (OQA)

ontology language based on existential rules

OQA(**L**)

Input: a database D, a set of existential rules $\Sigma \in$ **L**, a CQ Q/k, a tuple of constants **t** $\in$ **adom**(D)^k

Question: **t** $\in$ Answer(Q,D,Σ)?

BOQA(**L**)

Input: a database D, a set of existential rules $\Sigma \in$ **L**, a Boolean query Q

Question: is Answer(Q,D,Σ) non-empty?

Theorem: OQA(**L**) $\equiv_L$ BOQA(**L**) for every language **L**

($\equiv_L$ means logspace-equivalent)

Data Complexity of BOQA

input D, fixed Σ and Q

BOQA[Σ,Q](**L**)

Input: a database D

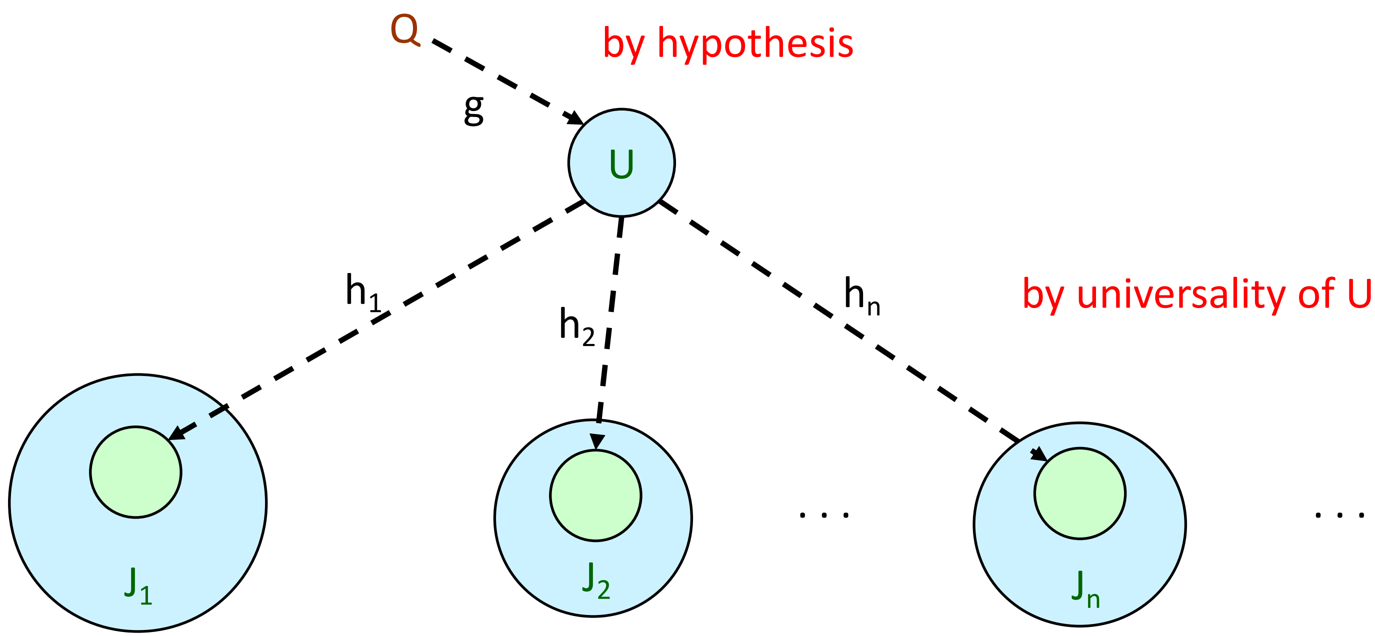
Question: is Answer(Q,D,Σ) non-empty?

Query Answering via Universal Models

Theorem: Answer(Q,D,Σ) is non-empty iff $Q(U)$ is non-empty, where U a universal model of (D,Σ)

Proof: ($\Rightarrow$) Trivial since, for every $J \in$ models(D,Σ), $Q(J)$ is non-empty

($\Leftarrow$) By exploiting the universality of U



$\forall J \in$ models(D,Σ), $\exists h$ such that $h(g(Q)) \subseteq J$ $\Rightarrow$ $\forall J \in$ models(D,Σ), $Q(J)$ is non-empty

$\Rightarrow$ Answer(Q,D,Σ) is non-empty

The Chase Procedure

D

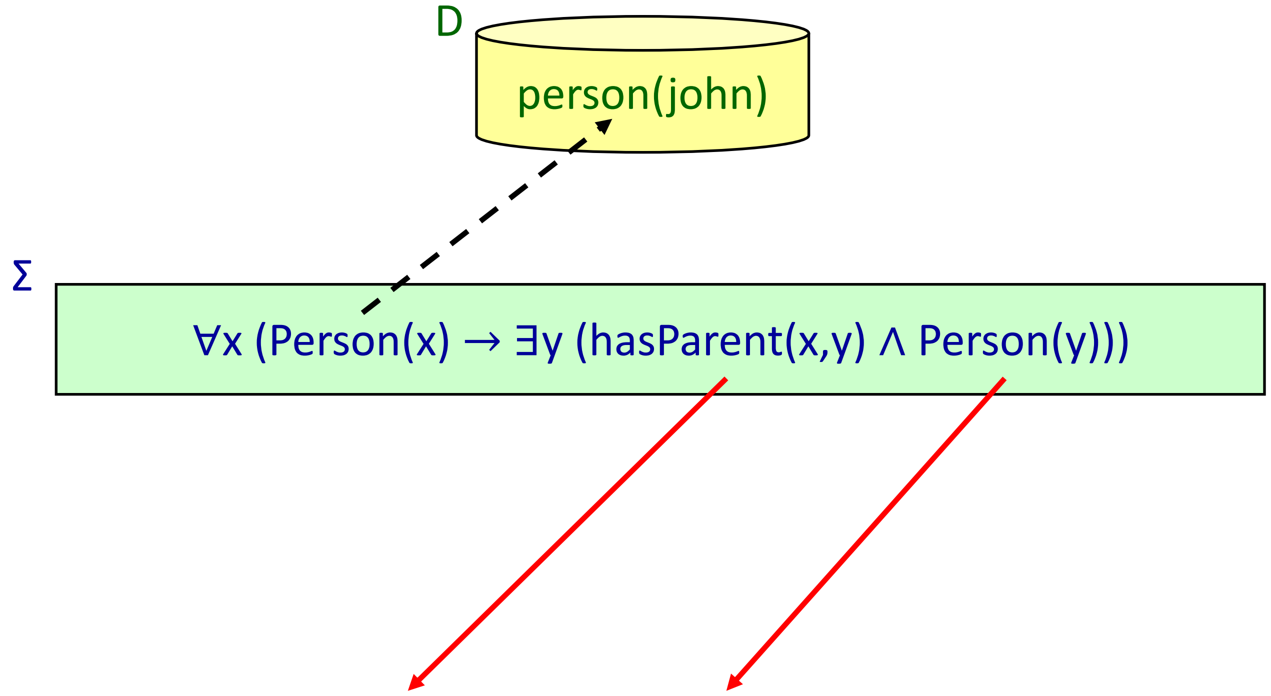
person(john)

Σ

$\forall x\,(\mathrm{Person}(x) \rightarrow \exists y\,(\mathrm{hasParent}(x,y) \wedge \mathrm{Person}(y)))$

$\mathrm{chase}(D,\Sigma) = D \cup$

The Chase Procedure



$\text{chase}(D,\Sigma) = D \cup \{\text{hasParent}(\text{john},\perp_1), \text{Person}(\perp_1)$

The Chase Procedure

D

person(john)

Σ

$\forall x\,(\mathrm{Person}(x) \rightarrow \exists y\,(\mathrm{hasParent}(x,y) \wedge \mathrm{Person}(y)))$

$\mathrm{chase}(D,\Sigma) = D \cup \{\mathrm{hasParent}(\mathrm{john},\perp_1), \mathrm{Person}(\perp_1),$

$\mathrm{hasParent}(\perp_1,\perp_2), \mathrm{Person}(\perp_2)$

The Chase Procedure

D

person(john)

Σ

$$\forall x\, (Person(x) \rightarrow \exists y\, (hasParent(x,y) \land Person(y)))$$

$$chase(D,\Sigma) = D \cup \{hasParent(john,\perp_1),\ Person(\perp_1),$$

$$hasParent(\perp_1,\perp_2),\ Person(\perp_2),$$

$$hasParent(\perp_2,\perp_3),\ Person(\perp_3)$$

The Chase Procedure

D

person(john)

Σ

$\forall x\,(Person(x) \rightarrow \exists y\,(hasParent(x,y) \wedge Person(y)))$

$chase(D,\Sigma) = D \cup \{hasParent(john,\perp_1), Person(\perp_1),$

$hasParent(\perp_1,\perp_2), Person(\perp_2),$

$hasParent(\perp_2,\perp_3), Person(\perp_3), \ldots$

infinite instance

The Chase Procedure: Formal Definition

- Chase step - the building block of the chase procedure

- A rule $\sigma = \forall \mathbf{x} \forall \mathbf{y}\, (\varphi(\mathbf{x},\mathbf{y}) \rightarrow \exists \mathbf{z}\, \psi(\mathbf{x},\mathbf{z}))$ is applicable to an instance J if:

  1. There exists a homomorphism h such that $h(\varphi(\mathbf{x},\mathbf{y})) \subseteq J$
  2. There is no $g \supseteq h_{|\mathbf{x}}$ such that $g(\psi(\mathbf{x},\mathbf{z})) \subseteq J$

$J = \{R(a), P(a,b)\}$

$h = \{x \mapsto a\}$ $\quad$ $g = \{x \mapsto a, y \mapsto b\}$

$\forall x\, (R(x) \rightarrow \exists y\, P(x,y))$

✗

$J = \{R(a), P(b,a)\}$

$h = \{x \mapsto a\}$ $\quad$ ✗

$\forall x\, (R(x) \rightarrow \exists y\, P(x,y))$

✓

The Chase Procedure: Formal Definition

- Chase step - the building block of the chase procedure

- A rule $\sigma = \forall \mathbf{x} \forall \mathbf{y} \, (\varphi(\mathbf{x},\mathbf{y}) \rightarrow \exists \mathbf{z} \, \psi(\mathbf{x},\mathbf{z}))$ is applicable to an instance J if:

    1. There exists a homomorphism h such that $h(\varphi(\mathbf{x},\mathbf{y})) \subseteq J$

    2. There is no $g \supseteq h_{|\mathbf{x}}$ such that $g(\psi(\mathbf{x},\mathbf{z})) \subseteq J$

- Let $J_+ = J \cup \{g(\psi(\mathbf{x},\mathbf{z}))\}$, where $g \supseteq h_{|\mathbf{x}}$ and $g(\mathbf{z})$ are "fresh" nulls not in J

- The result of applying σ to J is J_+, denoted $J[\sigma,h]J_+$ - single chase step

The Chase Procedure: Formal Definition

- A finite chase of D w.r.t. Σ is a finite sequence

$$D[\sigma_1,h_1]J_1[\sigma_2,h_2]J_2[\sigma_3,h_3]J_3 \cdots J_{n-1}[\sigma_n,h_n]J_n$$

and chase(D,Σ) is defined as the instance J_n

all applicable rules will eventually be applied

- An infinite chase of D w.r.t. Σ is a fair finite sequence

$$D[\sigma_1,h_1]J_1[\sigma_2,h_2]J_2[\sigma_3,h_3]J_3 \ldots J_{n-1}[\sigma_n,h_n]J_n \ldots$$

and chase(D,Σ) is defined as the instance $D \cup J_1 \cup J_2 \cup J_3 \cup \cdots \cup J_n \cup \cdots$

least fixpoint of a monotonic operator - chase step

Chase: A Universal Model

Theorem: chase(D,Σ) is a universal model of (D,Σ)

the result of the chase after $k \geq 0$ applications of the chase step

Proof:

- By construction, chase(D,Σ) ∈ models(D,Σ)
- It remains to show that chase(D,Σ) can be mapped into every other model of (D,Σ)
- Fix an arbitrary instance J ∈ models(D,Σ). We need to show that there exists h such that h(chase(D,Σ)) ⊆ J
- **By induction on the number of applications of the chase step, we show that for every $k \geq 0$, there exists h_k such that $h_k(\text{chase}^{[k]}(D,\Sigma)) \subseteq J$, and h_k is compatible with h_{k-1}**
- Clearly, $h_0 \cup h_1 \cup \cdots \cup h_n \cup \cdots$ is a well-defined homomorphism that maps chase(D,Σ) to J
- The claim follows with $h = h_0 \cup h_1 \cup \cdots \cup h_n \cup \cdots$

Chase: Uniqueness Property

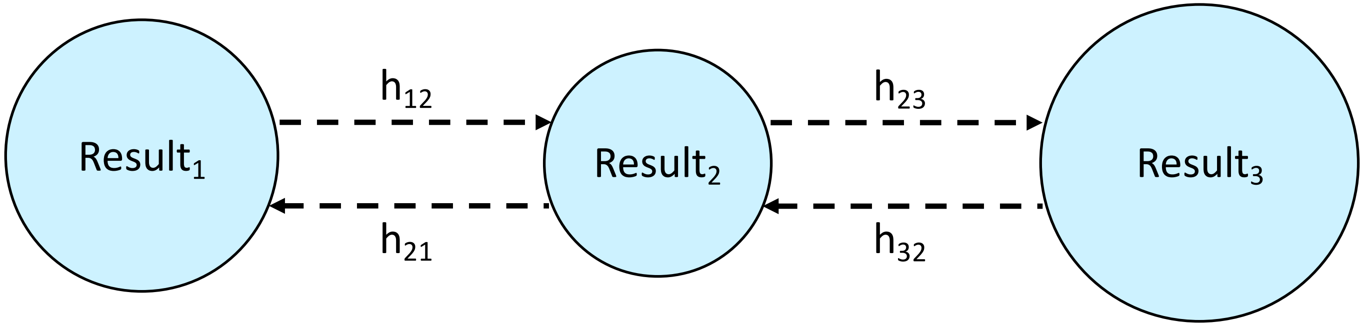
- The result of the chase is not unique - depends on the order of rule application

$D = \{P(a)\}$ $\sigma_1 = \forall x\, (P(x) \rightarrow \exists y\, R(y))$ $\sigma_2 = \forall x\, (P(x) \rightarrow R(x))$

$\text{Result}_1 = \{P(a), R(\perp), R(a)\}$ σ_1 then σ_2

$\text{Result}_2 = \{P(a), R(a)\}$ σ_2 then σ_1

- But, it is unique up to homomorphic equivalence



- Thus, it is unique for query answering purposes

Query Answering via the Chase

Theorem: Answer(Q,D,Σ) is non-empty iff Q(U) is non-empty, where U a universal model of (D,Σ)

&

Theorem: chase(D, Σ) is a universal model of (D,Σ)

⇓

Corollary: Answer(Q,D,Σ) is non-empty iff Q(chase(D,Σ)) is non-empty

- We can tame the first dimension of infinity by exploiting the chase procedure
- What about the second dimension of infinity? - the chase may be infinite

Can we tame the second dimension of infinity?

Undecidability of Ontological Query Answering

arbitrary existential rules

Theorem: OBQA(∃**RULES**) is undecidable

Proof Idea : By simulating a deterministic Turing machine with an empty tape.

Encode the computation of a DTM M with an empty tape using a database D, a set Σ of existential rules, and a Boolean CQ Q such that Answer(Q,D,Σ) is non-empty iff M accepts

Gaining Decidability

By restricting the database

- Answer(Q,{Start(c)},Σ) is non-empty iff the DTM M accepts
- The problem is undecidable even for singleton databases
- No much to do in this direction

By restricting the query language

- Answer(Q :- Accept(x),D,Σ) is non-empty iff the DTM M accepts
- The problem is undecidable already for atomic queries
- No much to do in this direction

By restricting the ontology language

- Achieve a good trade-off between expressive power and complexity
- Field of intense research
- Any ideas?

Source of Non-termination

D

person(john)

Σ

$\forall x\, (Person(x) \rightarrow \exists y\, (hasParent(x,y) \wedge Person(y)))$

$chase(D,\Sigma) = D \cup \{hasParent(john,\perp_1), Person(\perp_1),$

$hasParent(\perp_1,\perp_2), Person(\perp_2),$

$hasParent(\perp_2,\perp_3), Person(\perp_3), \ldots$

1. Existential quantification

2. Recursive definitions

infinite instance

Termination of the Chase

- Drop the existential quantification
  - We obtain the class of **full existential rules**
  - Very close to Datalog

- Drop the recursive definitions
  - We obtain the class of **acyclic existential rules**
  - Also known as non-recursive existential rules

Our Simple Example

D

person(john)

Σ

$\forall x\, (Person(x) \rightarrow \exists y\, (hasParent(x,y) \wedge Person(y)))$

$chase(D,\Sigma) = D \cup \{hasParent(john,\perp_1), Person(\perp_1),$

$hasParent(\perp_1,\perp_2), Person(\perp_2),$

$hasParent(\perp_2,\perp_3), Person(\perp_3), \ldots$

Existential quantification & recursive definitions

are key features for modelling ontologies

Key Question

We need classes of existential rules such that

- Existential quantification and recursive definition coexist

  $\Rightarrow$ the chase may be infinite

- BOQA is decidable, and tractable w.r.t. the data complexity

$\Downarrow$

Tame the infinite chase:

Deal with infinite structures without explicitly building them

Linear Existential Rules

- A linear existential rule is an existential rule of the form

$$\forall \mathbf{x} \forall \mathbf{y}\, (P(\mathbf{x},\mathbf{y}) \rightarrow \exists \mathbf{z}\, \psi(\mathbf{x},\mathbf{z}))$$

single atom

- We denote **LINEAR** the class of linear existential rules

- But, is this a reasonable ontology language?

3 OWL 2 QL

The OWL 2 QL profile is designed so that sound and complete query answering is in LOGSPACE (more precisely, in AC^0) with respect to the size of the data (assertions), while providing many of the main features necessary to express conceptual models such as UML class diagrams and ER diagrams. In particular, this profile contains the intersection of RDFS and OWL 2 DL. It is designed so that data (assertions) that is stored in a standard relational database system can be queried through an ontology via a simple rewriting mechanism, i.e., by rewriting the query into an SQL query that is then answered by the RDBMS system, without any changes to the data.

OWL 2 QL is based on the DL-Lite family of description logics [*DL-Lite*]. Several variants of DL-Lite have been described in the literature, and DL-Lite$_R$ provides the logical underpinning for OWL 2 QL. DL-Lite$_R$ does not require the unique name assumption (UNA), since making this assumption would have no impact on the semantic consequences of a DL-Lite$_R$ ontology. More expressive variants of DL-Lite, such as DL-Lite$_A$, extend DL-Lite$_R$ with functional properties, and these can also be extended with keys; however, for query answering to remain in LOGSPACE, these extensions require UNA and need to impose certain global restrictions on the interaction between properties used in different types of axiom. Basing OWL 2 QL on DL-Lite$_R$ avoids practical problems involved in the explicit axiomatization of UNA. Other variants of DL-Lite can also be supported on top of OWL 2 QL, but may require additional restrictions on the structure of ontologies.

3.1 Feature Overview

OWL 2 QL is defined not only in terms of the set of supported constructs, but it also restricts the places in which these constructs are allowed to occur. The allowed usage of constructs in class expressions is summarized in Table 1.

Table 1. Syntactic Restrictions on Class Expressions in OWL 2 QL

Subclass Expressions	Superclass Expressions
a class existential quantification (**ObjectSomeValuesFrom**) where the class is limited to *owl:Thing* existential quantification to a data range (**DataSomeValuesFrom**)	a class intersection (**ObjectIntersectionOf**) negation (**ObjectComplementOf**) existential quantification to a class (**ObjectSomeValuesFrom**) existential quantification to a data range (**DataSomeValuesFrom**)

OWL 2 QL supports the following axioms, constrained so as to be compliant with the mentioned restrictions on class expressions:

- subclass axioms (**SubClassOf**)
- class expression equivalence (**EquivalentClasses**)
- class expression disjointness (**DisjointClasses**)
- inverse object properties (**InverseObjectProperties**)
- property inclusion (**SubObjectPropertyOf** not involving property chains and **SubDataPropertyOf**)
- property equivalence (**EquivalentObjectProperties** and **EquivalentDataProperties**)
- property domain (**ObjectPropertyDomain** and **DataPropertyDomain**)
- property range (**ObjectPropertyRange** and **DataPropertyRange**)
- disjoint properties (**DisjointObjectProperties** and **DisjointDataProperties**)

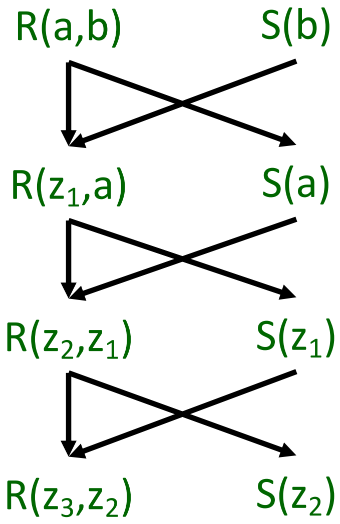
https://www.w3.org/TR/owl2-profiles/#OWL_2_QL

Chase Graph

The chase can be naturally seen as a graph - chase graph

$D = \{R(a,b), S(b)\}$

$$\Sigma = \begin{cases} \forall x \forall y\, (R(x,y) \wedge S(y) \rightarrow \exists z\, R(z,x)) \\ \forall x \forall y\, (R(x,y) \rightarrow S(x)) \end{cases}$$

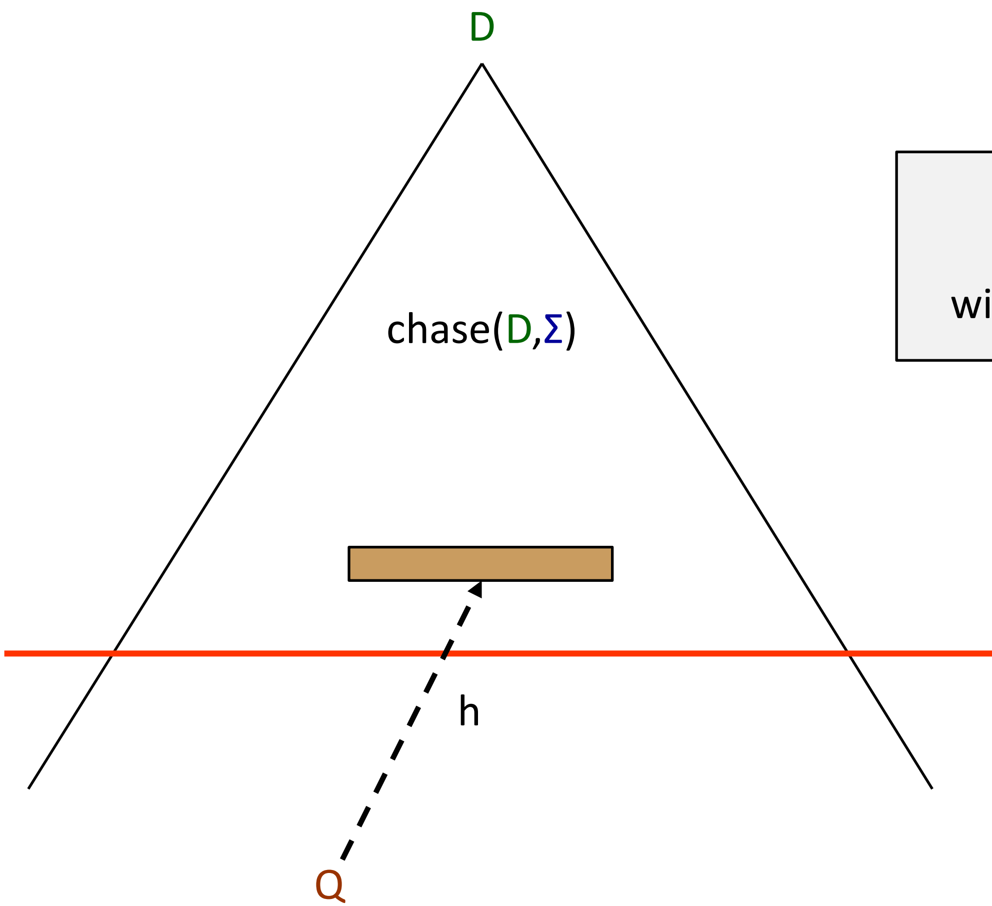


For **LINEAR** the chase graph is a forest

Bounded Derivation-Depth Property

For **LINEAR**, $k = |Q| \cdot m$

with $m = |sch(\Sigma)| \cdot (2 \cdot maxarity)^{maxarity}$



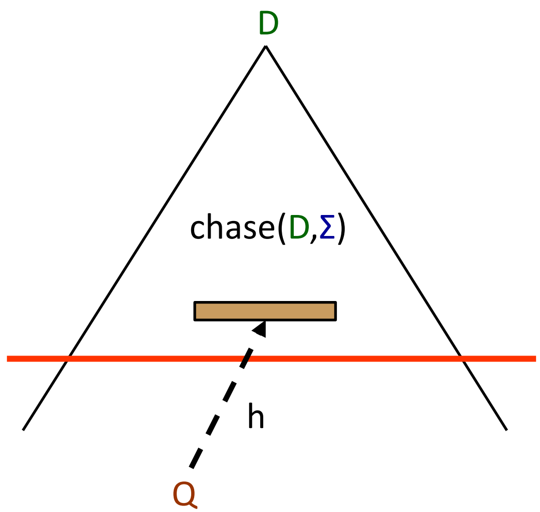
depth k that does not depend on D

chase graph up to depth k

$Q(chase(D,\Sigma))$ is non-empty $\Rightarrow$ $Q(chase^{k}(D,\Sigma))$ is non-empty

The Blocking Algorithm for **LINEAR**

Theorem: BOQA[Σ,Q](**LINEAR**) is in PTIME for a fixed set Σ, and a Boolean CQ Q



$$k = |Q| \cdot |sch(\Sigma)| \cdot (2 \cdot \text{maxarity})^{\text{maxarity}}$$

The Blocking Algorithm for LINEAR

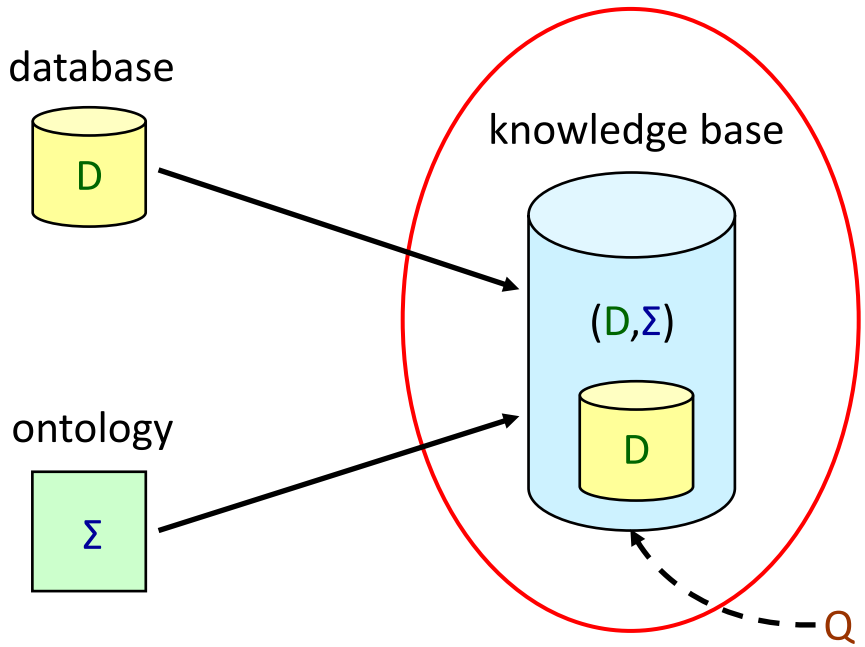
Theorem: BOQA[Σ,Q](**LINEAR**) is in PTIME for a fixed set Σ, and a Boolean CQ Q

but, we can do better

Theorem: BOQA[Σ,Q](**LINEAR**) is in LOGSPACE for a fixed set Σ, and a Boolean CQ Q

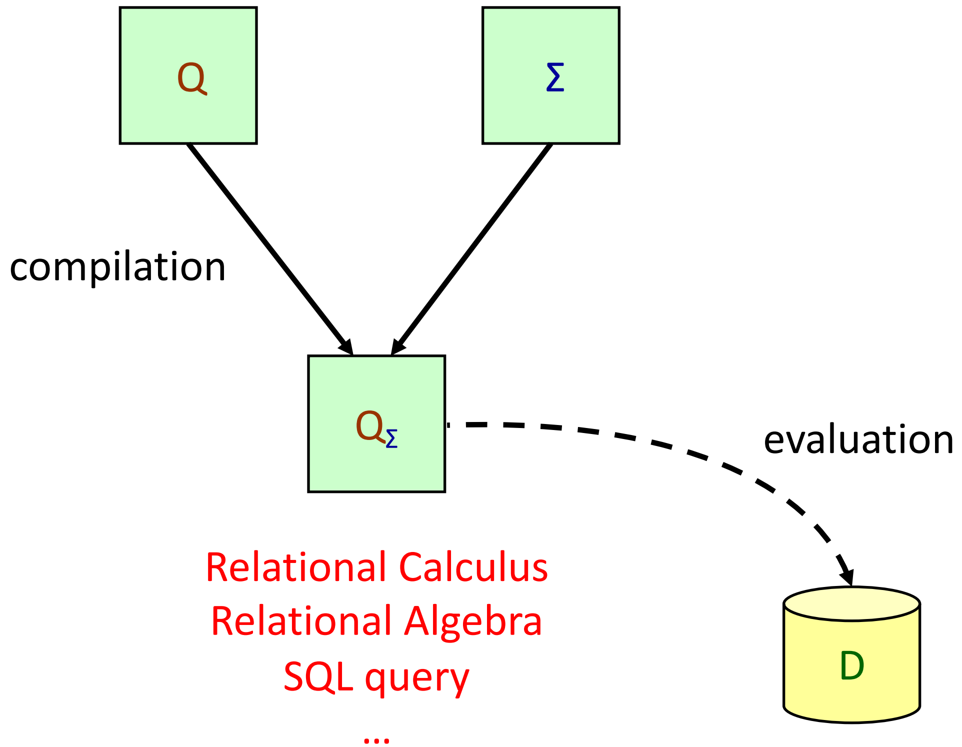
Scalability in OQA

Exploit standard RDBMSs - efficient technology for answering CQs



But in the OQA setting we have to query a knowledge base, not just a relational database

Query Rewriting



for every database D, Answer(Q,D,Σ) is non-empty iff Q_Σ(D) is non-empty

Query Rewriting: Formal Definition

Consider a class of existential rules **L**, and a query language **Q**.

BOQA(**L**) is **Q**-rewritable if, for every $\Sigma \in$ **L** and Boolean CQ Q,

we can construct a Boolean query $Q_{\Sigma} \in$ **Q** such that,

for every database D, Answer(Q,D,Σ) is non-empty iff Q_{Σ}(D) is non-empty

NOTE: The construction of Q_{Σ} is database-independent

An Example

$\Sigma = \{\forall x\, (P(x) \rightarrow T(x)),\ \ \forall x \forall y\, (R(x,y) \rightarrow S(x))\}$

Q :- S(x), U(x,y), T(y)

$Q_\Sigma = \{$Q :- S(x), U(x,y), T(y),

Q_1 :- S(x), U(x,y), P(y),

Q_2 :- R(x,z), U(x,y), T(y),

Q_3 :- R(x,z), U(x,y), P(y)$\}$

An Example

$\Sigma = \{\forall x \forall y\, (R(x,y) \wedge P(y) \rightarrow P(x))\}$

Q :- $P(c)$

$Q_\Sigma = \{Q$:- $P(c)$,

Q_1 :- $R(c,y_1), P(y_1)$,

Q_2 :- $R(c,y_1), R(y_1,y_2), P(y_2)$,

Q_3 :- $R(c,y_1), R(y_1,y_2), R(y_2,y_3), P(y_3)$,

… $\}$

- This cannot be written as a finite first-order query
- It can be written as Q :- $R(c,x), R^*(x,y), P(y)$, but transitive closure is not FO-expressible

Query Rewriting for **LINEAR**

union of conjunctive queries

Theorem: **LINEAR** is UCQ-rewritable

$\Downarrow$

Theorem: BOQA[Σ,Q](**LINEAR**) is in LOGSPACE for a fixed set Σ, and a Boolean CQ Q

…it also tells us that for answering CQs in the presence of **LINEAR** ontologies,

we can exploit standard database technology

UCQ-Rewritings

- The standard algorithm for computing UCQ-rewritings performs an exhaustive application of the following two steps:
    1. Rewriting
    2. Minimization

- We are going to see the version of the algorithm that assumes normalized existential rules, where only one atom appears in the head

Normalization Procedure

$$\forall \mathbf{x} \forall \mathbf{y}\, (\varphi(\mathbf{x},\mathbf{y}) \rightarrow \exists \mathbf{z}\, (P_1(\mathbf{x},\mathbf{z}) \wedge \cdots \wedge P_n(\mathbf{x},\mathbf{z})))$$

$$\downarrow$$

$$\forall \mathbf{x} \forall \mathbf{y}\, (\varphi(\mathbf{x},\mathbf{y}) \rightarrow \exists \mathbf{z}\, \mathrm{Auxiliary}(\mathbf{x},\mathbf{z}))$$

$$\forall \mathbf{x} \forall \mathbf{z}\, (\mathrm{Auxiliary}(\mathbf{x},\mathbf{z}) \rightarrow P_1(\mathbf{x},\mathbf{z}))$$

$$\forall \mathbf{x} \forall \mathbf{z}\, (\mathrm{Auxiliary}(\mathbf{x},\mathbf{z}) \rightarrow P_2(\mathbf{x},\mathbf{z}))$$

...

$$\forall \mathbf{x} \forall \mathbf{z}\, (\mathrm{Auxiliary}(\mathbf{x},\mathbf{z}) \rightarrow P_n(\mathbf{x},\mathbf{z}))$$

NOTE : Linearity is preserved, and we obtain an equivalent ontology w.r.t. query answering

UCQ-Rewritings

- The standard algorithm for computing UCQ-rewritings performs an exhaustive application of the following two steps:
    1. Rewriting
    2. Minimization

- We are going to see the version of the algorithm that assumes normalized existential rules, where only one atom appears in the head

Rewriting Step

$\Sigma = \{\forall x \forall y\ (project(x) \land inArea(x,y) \rightarrow \exists z\ hasCollaborator(z,y,x))\}$

Q :- $hasCollaborator(u,db,v)$

$g = \{x \mapsto v,\ y \mapsto db,\ z \mapsto u\}$

$hasCollaborator(u,db,v)$

Thus, we can simulate a chase step by applying a backward resolution step

$Q_\Sigma = \{Q$:- $hasCollaborator(u,db,v),$

Q_1 :- $project(v), inArea(v,db)\}$

Unsound Rewritings

$\Sigma = \{\forall x \forall y\ (project(x) \wedge\ inArea(x,y) \rightarrow \exists z\ hasCollaborator(z,y,x))\}$

Q :- $hasCollaborator(c,db,v)$

(c is a constant)

$g = \{x \mapsto v,\ y \mapsto db,\ z \mapsto c\}$

$hasCollaborator(c,db,v)$

After applying the rewriting step we obtain the following UCQ

$Q_\Sigma = \{Q$:- $hasCollaborator(c,db,v),$

Q_1 :- $project(v), inArea(v,db)\}$

Unsound Rewritings

$\Sigma = \{\forall x \forall y\ (project(x) \wedge inArea(x,y) \rightarrow \exists z\ hasCollaborator(z,y,x))\}$

Q :- hasCollaborator(c,db,v)

Q_Σ = {Q :- hasCollaborator(c,db,v),

Q_1 :- project(v), inArea(v,db)}

- Consider the database D = {project(a), inArea(a,db)}
- Clearly, $Q_\Sigma(D)$ is non-empty
- However, Answer(Q,D,Σ) is empty since there is no way to obtain an atom of the form hasCollaborator(c,db,_) during the chase

Unsound Rewritings

$\Sigma = \{\forall x \forall y\, (project(x) \wedge inArea(x,y) \rightarrow \exists z\, hasCollaborator(z,y,x))\}$

Q :- $hasCollaborator(c,db,v)$

$$Q_\Sigma = \{Q :- hasCollaborator(c,db,v),$$

$$Q_1 :- project(v), inArea(v,db)\}$$

the information about the constant c in the original query is lost after the application of the rewriting step since c is unified with an ∃-variable

Unsound Rewritings

$\Sigma = \{\forall x \forall y\ (\mathrm{project}(x) \wedge \mathrm{inArea}(x,y) \rightarrow \exists z\ \mathrm{hasCollaborator}(z,y,x))\}$

Q :- hasCollaborator(v,db,v)

$g = \{x \mapsto v,\ y \mapsto db,\ z \mapsto v\}$

hasCollaborator(v,db,v)

After applying the rewriting step we obtain the following UCQ

Q_Σ = {Q :- hasCollaborator(v,db,v),

Q_1 :- project(v), inArea(v,db)}

Unsound Rewritings

$\Sigma = \{\forall x \forall y\ (project(x) \wedge inArea(x,y) \rightarrow \exists z\ hasCollaborator(z,y,x))\}$

Q :- $hasCollaborator(v,db,v)$

$$Q_\Sigma = \{Q \text{ :- } hasCollaborator(v,db,v), \quad Q_1 \text{ :- } project(v), inArea(v,db)\}$$

- Consider the database $D = \{project(a), inArea(a,db)\}$
- Clearly, $Q_\Sigma(D)$ is non-empty
- However, Answer(Q,D,Σ) is empty since there is no way to obtain an atom of the form $hasCollaborator(t,db,t)$ during the chase

Unsound Rewritings

$\Sigma = \{\forall x \forall y\, (project(x) \wedge inArea(x,y) \rightarrow \exists z\, hasCollaborator(z,y,x))\}$

Q :- hasCollaborator(v,db,v)

$Q_\Sigma = \{$Q :- hasCollaborator(v,db,v),

Q_1 :- project(v), inArea(v,db)$\}$

the fact that v in the original query participates in a join is lost after the application of the rewriting step since v is unified with an ∃-variable

Applicability Condition

Consider a Boolean CQ Q, an atom α in Q, and a (normalized) rule σ.

We say that σ is applicable to α if the following conditions hold:

1. head(σ) and α unify via h

2. For every variable x in head(σ):
   1. If h(x) is a constant, then x is a $\forall$-variable
   2. If h(x) = h(y), where y is a shared variable of α, then x is a $\forall$-variable

3. If x is an $\exists$-variable of head(σ), and y is a variable in head(σ) such that $x \neq y$, then $h(x) \neq h(y)$

...but, although it is crucial for soundness, may destroy completeness

Incomplete Rewritings

Σ = {$\forall x \forall y$ (project(x) $\wedge$ inArea(x,y) $\rightarrow \exists z$ hasCollaborator(z,y,x)),

$\forall x \forall y \forall z$ (hasCollaborator(x,y,z) $\rightarrow$ collaborator(x))}

Q :- hasCollaborator(u,v,w), collaborator(u))

Q_Σ = {Q :- hasCollaborator(u,v,w), collaborator(u),

Q_1 :- hasCollaborator(u,v,w), hasCollaborator(u,v',w')

- Consider the database D = {project(a), inArea(a,db)}
- Clearly, Q over chase(D,Σ) = D $\cup$ {hasCollaborator(z,db,a), collaborator(z)} is non-empty
- However, Q_Σ(D) is empty

Incomplete Rewritings

$\Sigma = \{\forall x \forall y\, (\text{project}(x) \wedge \text{inArea}(x,y) \rightarrow \exists z\, \text{hasCollaborator}(z,y,x)),$

$\forall x \forall y \forall z\, (\text{hasCollaborator}(x,y,z) \rightarrow \text{collaborator}(x))\}$

Q :- hasCollaborator(u,v,w), collaborator(u))

Q_Σ = {Q :- hasCollaborator(u,v,w), collaborator(u),

Q_1 :- hasCollaborator(u,v,w), hasCollaborator(u,v',w')

Q_2 :- project(u), inArea(u,v)

but, we cannot obtain the last query due to the applicablity condition

Incomplete Rewritings

Σ = {$\forall x \forall y$ (project(x) $\wedge$ inArea(x,y) $\rightarrow \exists z$ hasCollaborator(z,y,x)),

$\forall x \forall y \forall z$ (hasCollaborator(x,y,z) $\rightarrow$ collaborator(x))}

Q :- hasCollaborator(u,v,w), collaborator(u))

Q_Σ = {Q :- hasCollaborator(u,v,w), collaborator(u),

Q_1 :- hasCollaborator(u,v,w), hasCollaborator(u,v',w')

Q_2 :- hasCollaborator(u,v,w) - by minimization

Q_3 :- project(w), inArea(w,v) - by rewriting

Q_Σ(D) is non-empty, where D = {project(a), inArea(a,db)}

UCQ-Rewritings

- The standard algorithm for computing UCQ-rewritings performs an exhaustive application of the following two steps:

    1. Rewriting

    2. Minimization

- We are going to see the version of the algorithm that assumes normalized existential rules, where only one atom appears in the head

The Rewriting Algorithm

Q_Σ := {Q}

repeat

Q_{aux} := Q_Σ

foreach disjunct q of Q_{aux} **do**

//Rewriting Step

foreach atom α in q **do**

foreach rule σ in Σ **do**

if σ is applicable to α **then**

q_{rew} := rewrite(q,α,σ) //we resolve α using σ

if q_{rew} does not appear in Q_Σ (modulo variable renaming) **then**

Q_Σ := $Q_\Sigma \cup \{q_{rew}\}$

//Minimization Step

foreach pair of atoms α,β in q that unify **do**

q_{min} := minimize(q,α,β) //we apply the most general unifier of α and β on q

if q_{min} does not appear in Q_Σ (modulo variable renaming) **then**

Q_Σ := $Q_\Sigma \cup \{q_{min}\}$

until $Q_{aux} = Q_\Sigma$

return Q_Σ

Termination

Theorem: The rewriting algorithm terminates under **LINEAR**

Proof Idea:

- **Key observation:** the size of each partial rewriting is at most the size of the given CQ Q
- Thus, each partial rewriting can be transformed into an equivalent query that contains at most ($|Q| \cdot$ maxarity) variables
- The number of queries that can be constructed using a finite number of predicates and a finite number of variables is finite
- Therefore, only finitely many partial rewritings can be constructed - in general, exponentially many

Size of the Rewriting

- Ideally, we would like to construct UCQ-rewritings of polynomial size
- But, the standard rewritng algorithm produces rewritings of exponential size
- Can we do better? **NO!!!**

$\Sigma = \{\forall x\, (R_k(x) \rightarrow P_k(x))\}$ for $k \in \{1,\ldots,n\}$ $\quad Q$:- $P_1(x), \ldots, P_n(x)$

Q :- $P_1(X), \ldots, P_n(X)$

$P_1(X) \vee R_1(X)$ $\quad\quad P_n(X) \vee R_n(X)$

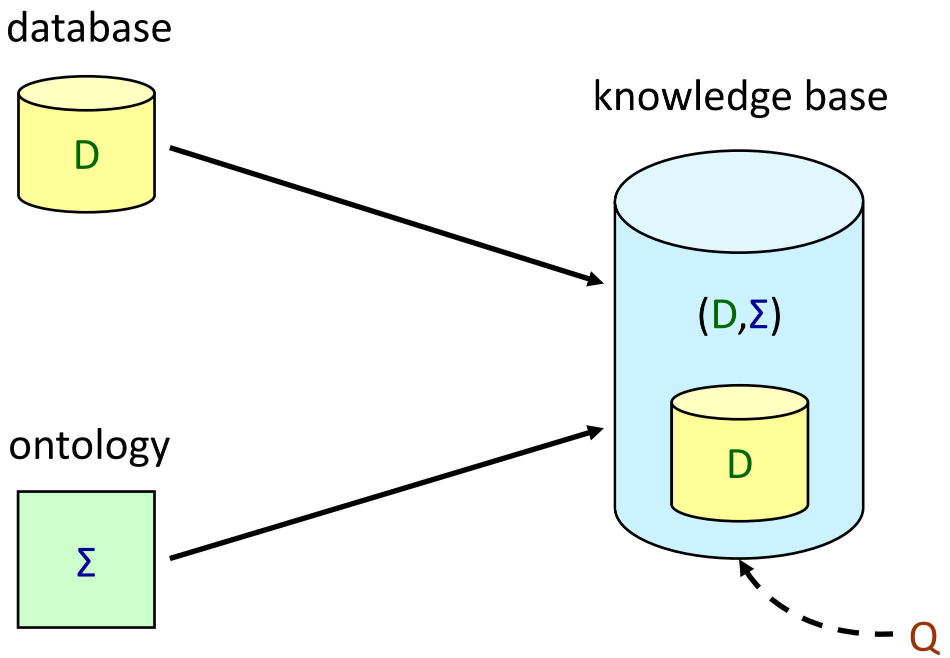
thus, we need to consider 2^n disjuncts

Size of the Rewriting

- Ideally, we would like to construct UCQ-rewritings of polynomial size
- But, the standard rewritng algorithm produces rewritings of exponential size
- Can we do better? **NO!!!**

  - Although the standard rewriting algorithm is worst-case optimal, it can be significantly improved
  - Optimization techniques can be applied in order to compute efficiently small rewritings - field of intense research

Recap



existential rules

$\forall \mathbf{x} \forall \mathbf{y}\, (\varphi(\mathbf{x},\mathbf{y}) \rightarrow \exists \mathbf{z}\, \psi(\mathbf{x},\mathbf{z}))$

conjunctive query

$Q(\mathbf{x}) \text{ :- } R_1(\mathbf{v_1}),\ldots,R_m(\mathbf{v_m})$

in general, this is an undecidable problem, but well-behaved ontology languages exists - **LINEAR**

MAI4CAREU Master programmes in Artificial Intelligence 4 Careers in Europe

University of Cyprus

MAI649: PRINCIPLES OF ONTOLOGICAL DATABASES

Thank You!

Andreas Pieris

Spring 2022-2023

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

MAI4CAREU | Master programmes in Artificial Intelligence 4 Careers in Europe

University of Cyprus

MAI649: PRINCIPLES OF ONTOLOGICAL DATABASES

Complexity Theory

Andreas Pieris

Spring 2022-2023

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

A Crash Course on Complexity Theory

we recall some fundamental notions from complexity theory that will be heavily used in the context of MAI649 - further details can be found in the standard textbooks

Deterministic Turing Machine (DTM)

$$M \;=\; (S, \Lambda, \Gamma, \delta, s_0, s_{accept}, s_{reject})$$

- S is the set of states
- Λ is the input alphabet, not containing the blank symbol $\sqcup$
- Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Lambda \subseteq \Gamma$
- $\delta : S \times \Gamma \;\rightarrow\; S \times \Gamma \times \{L,R\}$
- s_0 is the initial state
- s_{accept} is the accept state
- s_{reject} is the reject state, where $s_{accept} \neq s_{reject}$

Deterministic Turing Machine (DTM)

$$M \quad = \quad (S, \Lambda, \Gamma, \delta, s_0, s_{accept}, s_{reject})$$

$\delta(s_1, \alpha) = (s_2, \beta, R)$

IF at some time instant τ the machine is in sate s_1, the cursor points to cell κ, and this cell contains α

THEN at instant $\tau+1$ the machine is in state s_2, cell κ contains β, and the cursor points to cell $\kappa+1$

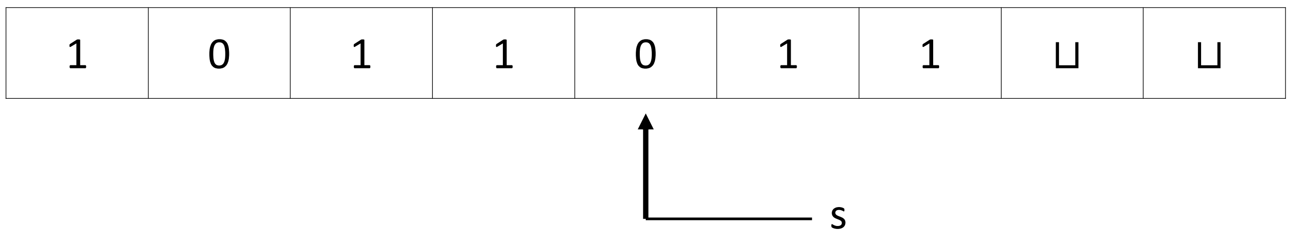
Nondeterministic Turing Machine (NTM)

$$M = (S, \Lambda, \Gamma, \delta, s_0, s_{accept}, s_{reject})$$

- S is the set of states
- Λ is the input alphabet, not containing the blank symbol $\sqcup$
- Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Lambda \subseteq \Gamma$
- $\delta : S \times \Gamma \rightarrow$ power set of $S \times \Gamma \times \{L,R\}$
- s_0 is the initial state
- s_{accept} is the accept state
- s_{reject} is the reject state, where $s_{accept} \neq s_{reject}$

Turing Machine Configuration

A perfect description of the machine at a certain point in the computation

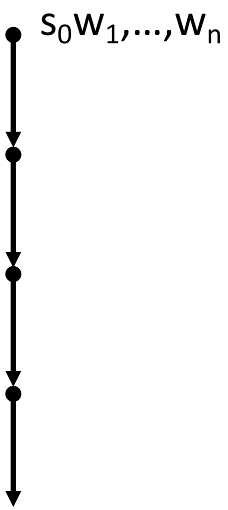


is represented as a string: **1011s011**

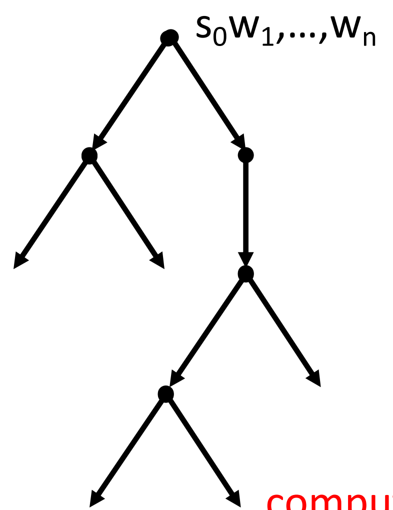
- Initial configuration on input $w_1,\ldots,w_n$ - $s_0 w_1,\ldots,w_n$
- Accepting configuration - $u_1,\ldots,u_k s_{accept} u_{k+1},\ldots,u_{k+m}$
- Rejecting configuration - $u_1,\ldots,u_k s_{reject} u_{k+1},\ldots,u_{k+m}$

Turing Machine Computation

Deterministic



Nondeterministic



computation path

the next configuration is unique

computation tree

Deciding a Problem

(recall that an instance of a decision problem Π is encoded as a word over a certain alphabet Λ - thus, Π is a set of words over Λ, i.e., $\Pi \subseteq \Lambda^*$)

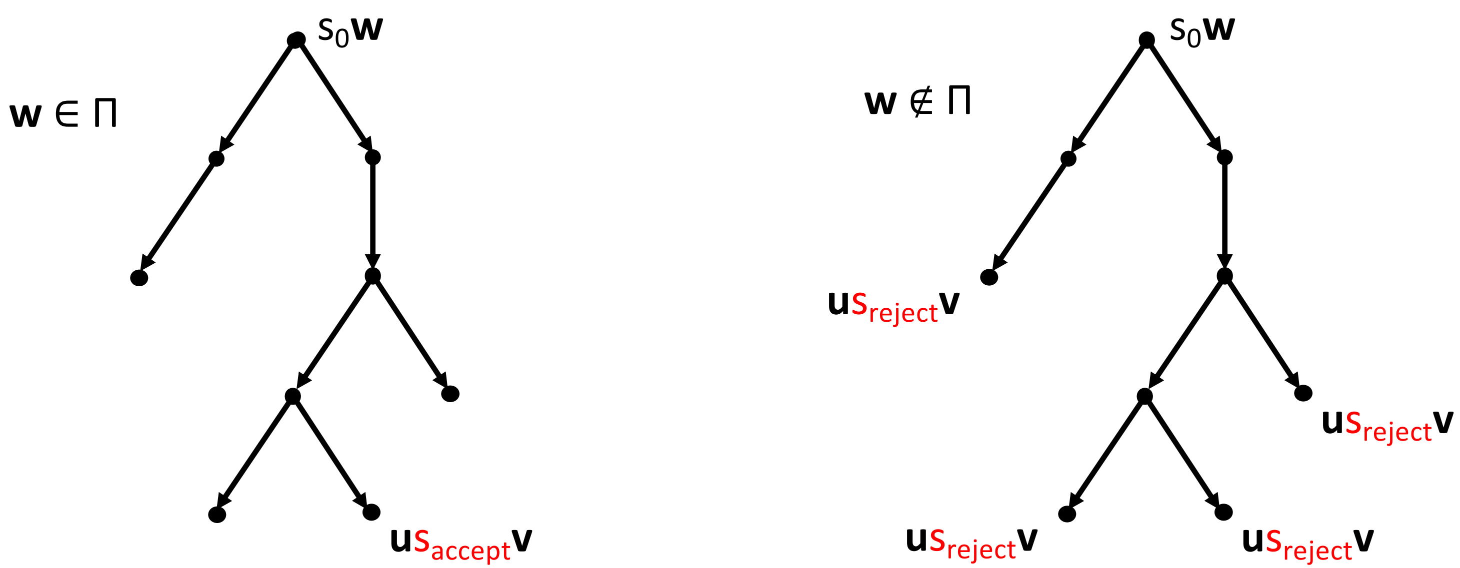
A DTM $M = (S, \Lambda, \Gamma, \delta, s_0, s_{accept}, s_{reject})$ decides a problem Π if, for every $\mathbf{w} \in \Lambda^*$:

- M on input **w** halts in s_{accept} if $\mathbf{w} \in \Pi$
- M on input **w** halts in s_{reject} if $\mathbf{w} \notin \Pi$

Deciding a Problem

A NTM $M = (S, \Lambda, \Gamma, \delta, s_0, s_{accept}, s_{reject})$ decides a problem Π if, for every $\mathbf{w} \in \Lambda^*$:

- The computation tree of M on input **w** is finite
- There exists at least one accepting computation path if $\mathbf{w} \in \Pi$
- There is no accepting computation path if $\mathbf{w} \notin \Pi$



Complexity Classes

Consider a function $f : N \rightarrow N$

$$\mathrm{TIME}(f(n)) = \{\Pi \mid \Pi \text{ is decided by some DTM in time } O(f(n))\}$$

$$\mathrm{NTIME}(f(n)) = \{\Pi \mid \Pi \text{ is decided by some NTM in time } O(f(n))\}$$

$$\mathrm{SPACE}(f(n)) = \{\Pi \mid \Pi \text{ is decided by some DTM using space } O(f(n))\}$$

$$\mathrm{NSPACE}(f(n)) = \{\Pi \mid \Pi \text{ is decided by some NTM using space } O(f(n))\}$$

Complexity Classes

- We can now recall the standard time and space complexity classes:

$$\text{PTIME} = \bigcup_{k>0} \text{TIME}(n^k)$$

$$\text{NP} = \bigcup_{k>0} \text{NTIME}(n^k)$$

$$\text{EXPTIME} = \bigcup_{k>0} \text{TIME}(2^{n^k})$$

$$\text{NEXPTIME} = \bigcup_{k>0} \text{NTIME}(2^{n^k})$$

$$\left.\begin{array}{l}\text{LOGSPACE} = \text{SPACE}(\log n) \\ \text{NLOGSPACE} = \text{NSPACE}(\log n)\end{array}\right\} \text{these definitions are relying on two-tape Turing machines with a read-only and a read/write tape}$$

$$\text{PSPACE} = \bigcup_{k>0} \text{SPACE}(n^k)$$

$$\text{EXPSPACE} = \bigcup_{k>0} \text{SPACE}(2^{n^k})$$

- For every complexity class C we can define its complementary class coC

$$\text{coC} = \{\Lambda^* \setminus \Pi \mid \Pi \in C\}$$

An Alternative Definition for NP

Theorem: Consider a problem $\Pi \subseteq \Lambda^*$. The following are equivalent:

- $\Pi \in NP$
- There is a relation $R \subseteq \Lambda^* \times \Lambda^*$ that is *polynomially decidable* such that

  $\Pi = \{\mathbf{u} \mid \text{there exists } \mathbf{w} \text{ such that } |\mathbf{w}| \leq |\mathbf{u}|^k \text{ and } (\mathbf{u},\mathbf{w}) \in R\}$

witness or certificate

$\{\mathbf{xy} \in \Lambda^* \mid (\mathbf{x},\mathbf{y}) \in R\} \in \text{PTIME}$

Example:

$3SAT = \{\phi \mid \phi \text{ is a 3CNF formula that is satisfiable}\}$

$= \{\phi \mid \phi \text{ is a 3CNF for which there is an assignment } \alpha \text{ such that } |\alpha| \leq |\phi| \text{ and } (\phi,\alpha) \in R\}$

where $R = \{(\phi,\alpha) \mid \alpha \text{ is a satisfying assignment for } \phi\} \in \text{PTIME}$

Relationship Among Complexity Classes

LOGSPACE $\subseteq$ NLOGSPACE $\subseteq$ PTIME $\subseteq$ NP, coNP $\subseteq$

PSPACE $\subseteq$ EXPTIME $\subseteq$ NEXPTIME, coNEXPTIME $\subseteq \cdots$

Some useful notes:

- For a deterministic complexity class C, coC = C
- coNLOGSPACE = NLOGSPACE
- It is generally believed that PTIME $\neq$ NP, but we don't know
- PTIME $\subset$ EXPTIME $\Rightarrow$ at least one containment between them is strict
- PSPACE = NPSPACE, EXPSPACE = NEXPSPACE, etc.
- But, we don't know whether LOGSPACE = NLOGSPACE

Complete Problems

- These are the hardest problems in a complexity class
- A problem that is complete for a class C, it is unlikely to belong in a lower class

- A problem Π is complete for a complexity class C, or simply C-complete, if:
  1. $\Pi \in C$
  2. Π is C-hard, i.e., every problem $\Pi' \in C$ can be efficiently reduced to Π

there exists a logspace algorithm that computes a function f such that

$\mathbf{w} \in \Pi'$ iff $f(\mathbf{w}) \in \Pi$ - in this case we write $\Pi' \leq_L \Pi$

- To show that Π is C-hard it suffices to reduce some C-hard problem Π' to it

Some Complete Problems

- NP-complete
  - SAT (satisfiability of propositional formulas)
  - Many graph-theoretic problems (e.g., 3-colorability)
  - Traveling salesman
  - etc.

- PSPACE-complete
  - Quantified SAT (or simply QSAT)
  - Equivalence of two regular expressions
  - Many games (e.g., Geography)
  - etc.

MAI4CAREU | Master programmes in Artificial Intelligence 4 Careers in Europe

University of Cyprus

MAI649: PRINCIPLES OF ONTOLOGICAL DATABASES

Thank You!

Andreas Pieris

Spring 2022-2023

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

MAI4CAREU | Master programmes in Artificial Intelligence 4 Careers in Europe

University of Cyprus

MAI649: PRINCIPLES OF ONTOLOGICAL DATABASES

First-Order Logic & Relational Calculus

Andreas Pieris

Spring 2022-2023

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423

A Crash Course on First-Order Logic

we recall the syntax and the semantics of first-order logic, and we discuss how first-order logic can be used to define a query language (that is, relational calculus) that will play a crucial role in the context of MAI649

Schemas and Databases

- We assume a countably infinite set **Rel** of relation symbols
- We assume a countably infinite set **Const** of constant values

Definition: A *relational schema* (or simply *schema*) is a finite set $\mathbf{S} = \{R_1,\ldots,R_n\}$, where each R_i, for $i \in \{1,\ldots,n\}$, is a relation symbol from **Rel** of some fixed arity denoted $\text{arity}_{\mathbf{S}}(R_i)$

Definition: A *database instance* (or simply *database*) of a schema **S** is a finite set of relational atoms $R(c_1,\ldots,c_k)$, where $R \in \mathbf{S}$, $\text{arity}_{\mathbf{S}}(R_i) = k$, and $c_i \in$ **Const** for each $i \in \{1,\ldots,k\}$

Syntax of First-Order Logic

- We assume a countably infinite set **Var** of variables
- We call the elements of **Const** and **Var** *terms*

Definition: *First-order (FO) formulae* over a schema **S** are inductively defined as follows:

- If a ∈ **Const** and x,y ∈ **Var**, then $x = a$ and $x = y$ are atomic formulae (*equational atoms*)
- If $u_1,\ldots,u_k$ are (not necessarily distinct) terms, and R ∈ **S** with $\text{arity}_{\mathbf{S}}(R) = k$, then

  $R(u_1,\ldots,u_k)$ is an atomic formula (*relational atom*)
- If φ_1 and φ_2 are FO formulae, then $(\varphi_1 \wedge \varphi_2)$, $(\varphi_1 \vee \varphi_2)$ and $(\neg\varphi_1)$ are FO formulae
- If φ is an FO formula and x ∈ **Var**, then $(\exists x\ \varphi)$ and $(\forall x\ \varphi)$ are FO formulae

Syntax of First-Order Logic: Example

$$\underbrace{\text{Airport}(x,u)}_{\varphi_1} \quad \underbrace{u = \text{London}}_{\varphi_2} \quad \underbrace{\text{Airport}(y,v)}_{\varphi_3} \quad \underbrace{v = \text{Glasgow}}_{\varphi_4} \quad \underbrace{\text{Flight}(x,y,z)}_{\varphi_5}$$

$$(\varphi_1 \wedge \varphi_2)$$

$$((\varphi_1 \wedge \varphi_2) \ \wedge \varphi_3)$$

$$(((\varphi_1 \wedge \varphi_2) \ \wedge \varphi_3) \ \wedge \varphi_4)$$

$$((((\varphi_1 \wedge \varphi_2) \ \wedge \varphi_3) \ \wedge \varphi_4) \ \wedge \varphi_5)$$

$$((((\text{Airport}(x,u) \ \wedge \ u = \text{London}) \ \wedge \ \text{Airport}(y,v)) \ \wedge \ v = \text{Glasgow}) \ \wedge \ \text{Flight}(x,y,z))$$

$$\text{Airport}(x,u) \ \wedge \ u = \text{London} \ \wedge \ \text{Airport}(y,v) \ \wedge \ v = \text{Glasgow} \ \wedge \ \text{Flight}(x,y,z)$$

(for brevity, we may omit the outermost brackets)

Free Variables

essentially, the variables in a formula that are not quantified

Definition: Given an FO formula φ, the set of *free variables of φ*, denoted FV(φ), is:

- $FV(x = a) = \{x\}$
- $FV(x = y) = \{x,y\}$
- $FV(R(u_1,\ldots,u_k)) = \{u_1,\ldots,u_k\} \cap \mathbf{Var}$
- $FV(\varphi_1 \wedge \varphi_2) = FV(\varphi_1 \vee \varphi_2) = FV(\varphi_1) \cup FV(\varphi_2)$
- $FV(\neg\varphi) = FV(\varphi)$
- $FV(\exists x\, \varphi) = FV(\forall x\, \varphi) = FV(\varphi) \setminus \{x\}$

Free Variables: Example

φ = Airport(x,u) $\wedge$ u = London $\wedge$ Airport(y,v) $\wedge$ v = Glasgow $\wedge$ Flight(x,y,z)

FV(φ) = {x,y,z,u,v}

φ = $\exists x \exists y \exists u \exists v$ (Airport(x,u) $\wedge$ u = London $\wedge$ Airport(y,v) $\wedge$ v = Glasgow $\wedge$ Flight(x,y,z))

FV(φ) = {z}

φ = $\exists x \exists y \exists z \exists u \exists v$ (Airport(x,u) $\wedge$ u = London $\wedge$ Airport(y,v) $\wedge$ v = Glasgow $\wedge$ Flight(x,y,z))

FV(φ) = $\emptyset$

Semantics of First-Order Logic

- Given a database D of a schema **S**, and an FO formula φ over **S**, an *assignment for φ over D* is a total function of the form η : FV(φ) → Dom(D) ∪ Dom(φ)

constants occurring in D and φ

- We write η[x/u], where x ∈ **Var** and u ∈ **Const** ∪ **Var**, for the assignment that modifies η by setting η(x) = u

Semantics of First-Order Logic

Definition: Given a database D of a schema **S**, an FO formula φ over **S**, and an assignment η for φ over D, we define when *φ is satisfied in D under η*, denoted $(D,\eta) \vDash \varphi$, as follows:

- If φ is $x = y$, then $(D,\eta) \vDash \varphi$ when $\eta(x) = \eta(y)$
- If φ is $x = a$, then $(D,\eta) \vDash \varphi$ when $\eta(x) = a$
- If φ is $R(u_1,\dots,u_k)$, then $(D,\eta) \vDash \varphi$ when $R(\eta(u_1),\dots,\eta(u_k)) \in D$
- If φ is $\varphi_1 \wedge \varphi_2$, then $(D,\eta) \vDash \varphi$ when $(D,\eta) \vDash \varphi_1$ **and** $(D,\eta) \vDash \varphi_2$
- If φ is $\varphi_1 \vee \varphi_2$, then $(D,\eta) \vDash \varphi$ when $(D,\eta) \vDash \varphi_1$ **or** $(D,\eta) \vDash \varphi_2$
- If φ is $\neg\psi$, then $(D,\eta) \vDash \varphi$ when $(D,\eta) \vDash \psi$ **does not hold**
- If φ is $\exists x\, \psi$, then $(D,\eta) \vDash \varphi$ when $(D,\eta[x/a]) \vDash \psi$ for **some** value $a \in \mathrm{Dom}(D) \cup \mathrm{Dom}(\varphi)$
- If φ is $\forall x\, \psi$, then $(D,\eta) \vDash \varphi$ when $(D,\eta[x/a]) \vDash \psi$ for **each** value $a \in \mathrm{Dom}(D) \cup \mathrm{Dom}(\varphi)$

Semantics of First-Order Logic

The standard priority is

$\neg \quad \wedge \quad \vee \quad \exists, \forall$

$\exists x\ \neg R(x) \wedge S(x)$ we mean $\exists x((\neg R(x)) \wedge S(x))$

notice the difference with $\exists x\ \neg(R(x) \wedge S(x))$

Relational Calculus: Syntax

- We can now use FO formulae to define queries
- We need to specify together with an FO formula a tuple of variables $x_1,\ldots,x_k$ that indicates how the output of the query is formed

Definition: A *relational calculus (RC) query* over a schema **S** is an expression of the form

$$\varphi(x_1,\ldots,x_k)$$

where φ is an FO formula over **S**, $\{x_1,\ldots,x_k\} \subseteq FV(\varphi)$, and each free variable of φ occurs at least once in $x_1,\ldots,x_k$

note that the syntax $\{(x_1,\ldots,x_k) \mid \varphi\}$ is also used

Relational Calculus: Semantics

Consider an RC query $\varphi(x_1,\ldots,x_k)$ over a schema **S**. A database D of a schema **S** *satisfies* $\varphi(x_1,\ldots,x_k)$ using the values $a_1,\ldots,a_k$, denoted $D \models \varphi(a_1,\ldots,a_k)$, if there exists an assignment η for φ over D such that $(\eta(x_1),\ldots,\eta(x_k)) = (a_1,\ldots,a_k)$ and $(D,\eta) \models \varphi$

Definition: Given a database D of a schema **S**, and an RC query $Q = \varphi(x_1,\ldots,x_k)$ over **S**, the *output of Q on D*, denoted Q(D), is defined as the set of tuples

$$\{ (a_1,\ldots,a_k) \in (\mathrm{Dom}(D) \cup \mathrm{Dom}(\varphi))^k \mid D \models \varphi(a_1,\ldots,a_k) \}$$

Relational Calculus: Example

List the airlines that fly directly from London to Glasgow

Flight	*origin*	*destination*	*airline*
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	*code*	*city*
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh

$$Q = \varphi(z)$$

$$\varphi = \exists x \exists y \exists u \exists v\ (\text{Airport}(x,u)\ \wedge\ u = \text{London}\ \wedge\ \text{Airport}(y,v)\ \wedge\ v = \text{Glasgow}\ \wedge\ \text{Flight}(x,y,z))$$

$$Q(D) = \{\ (U2)\ \}$$

Algebra = Calculus

A fundamental relative expressiveness result:

Theorem: Relational Algebra = Relational Calculus

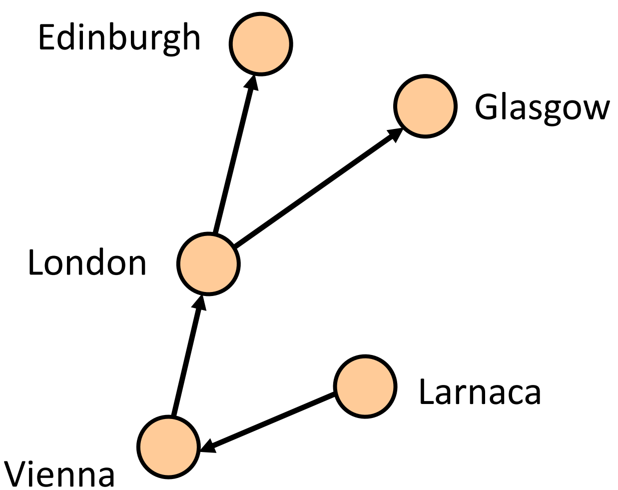
The proof can be found in Chapter 6 of PDB

Quiz!

Is Glasgow reachable from Vienna?

Flight	*origin*	*destination*	*airline*
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	*code*	*city*
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh



$$\exists x \exists y \exists z \exists w \exists v \ \ \text{Airport}(x,\text{Vienna}) \ \wedge \ \text{Airport}(y,\text{Glasgow}) \ \wedge$$

$$\text{Flight}(x,z,w) \ \wedge \ \text{Flight}(z,y,v)$$

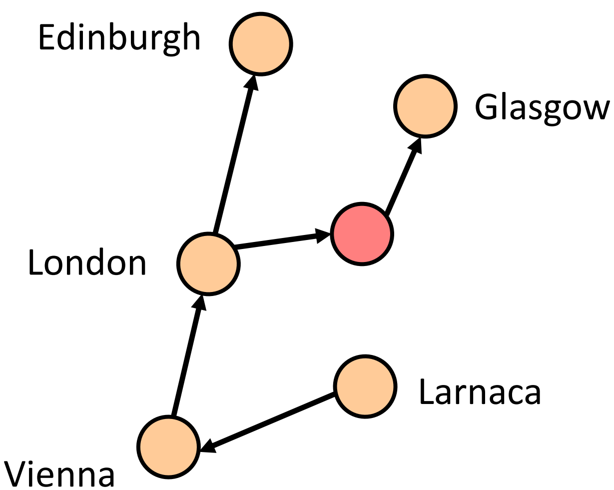
YES

Quiz!

Is Glasgow reachable from Vienna?

Flight	*origin*	*destination*	*airline*
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	*code*	*city*
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh



$$\exists x \exists y \exists z \exists w \exists v \ \ Airport(x,Vienna) \land Airport(y,Glasgow) \land Flight(x,z,w) \land Flight(z,y,v)$$

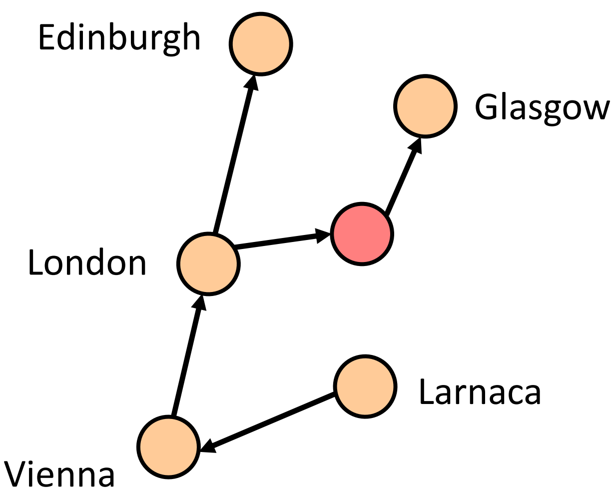
NO

Quiz!

Is Glasgow reachable from Vienna?

Flight	*origin*	*destination*	*airline*
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	*code*	*city*
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh



$$\exists x \exists y \exists z \exists w \exists v \ \ Airport(x,Vienna) \wedge Airport(y,Glasgow) \wedge$$

$$\exists z_1 \exists w_1 \quad Flight(x,z,w) \quad Flight(z,y,v)$$

$$\wedge \ Flight(z,z_1,w_1) \ \wedge$$

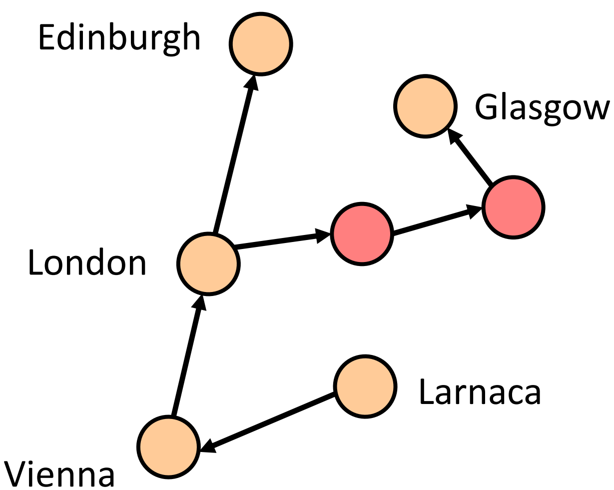
YES

Quiz!

Is Glasgow reachable from Vienna?

Flight	***origin***	***destination***	***airline***
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	***code***	***city***
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh



$\exists x \exists y \exists z \exists w \exists v$ Airport(x,Vienna) $\wedge$ Airport(y,Glasgow) $\wedge$

$\exists z_1 \exists w_1$ Flight(x,z,w) Flight(z,y,v)

$\wedge$ Flight(z,z_1,w_1) $\wedge$

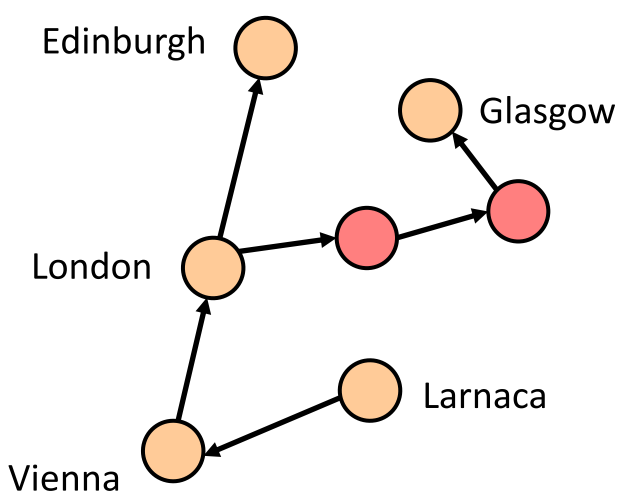
NO

Quiz!

Is Glasgow reachable from Vienna?

Flight	*origin*	*destination*	*airline*
	VIE	LHR	BA
	LHR	EDI	BA
	LGW	GLA	U2
	LCA	VIE	OS

Airport	*code*	*city*
	VIE	Vienna
	LHR	London
	LGW	London
	LCA	Larnaca
	GLA	Glasgow
	EDI	Edinburgh



Recursive query - not expressible in calculus/algebra

(unless we bound the number of intermediate stops)

MAI4CAREU | Master programmes in Artificial Intelligence 4 Careers in Europe

University of Cyprus

MAI649: PRINCIPLES OF ONTOLOGICAL DATABASES

Thank You!

Andreas Pieris

Spring 2022-2023

Co-financed by the European Union
Connecting Europe Facility

This Master is run under the context of Action No 2020-EU-IA-0087, co-financed by the EU CEF Telecom under GA nr. INEA/CEF/ICT/A2020/2267423