

Metodi Formali per il Software e i Servizi

FOL & Conjunctive Queries

Giuseppe De Giacomo

Sapienza Università di Roma
Laurea Magistrale in Ingegneria Informatica

2008/09

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

First-order logic

- ▶ First-order logic (FOL) is the logic to speak about **objects**, which are the domain of discourse or universe.
- ▶ FOL is concerned about **properties** of these objects and **relations** over objects (resp., unary and n -ary **predicates**).
- ▶ FOL also has **functions** including **constants** that denote objects.

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

FOL syntax – Terms

We first introduce:

- ▶ A set $Vars = \{x_1, \dots, x_n\}$ of **individual variables** (i.e., variables that denote single objects).
- ▶ A set of **functions symbols**, each of given arity ≥ 0 .
Functions of arity 0 are called **constants**.

Def.: The set of **Terms** is defined inductively as follows:

- ▶ $Vars \subseteq Terms$;
- ▶ If $t_1, \dots, t_k \in Terms$ and f^k is a k -ary function symbol, then $f^k(t_1, \dots, t_k) \in Terms$;
- ▶ Nothing else is in $Terms$.

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

FOL syntax – Formulas

Def.: The set of **Formulas** is defined inductively as follows:

- ▶ If $t_1, \dots, t_k \in Terms$ and P^k is a k -ary predicate, then $P^k(t_1, \dots, t_k) \in Formulas$ (atomic formulas).
- ▶ If $t_1, t_2 \in Terms$, then $t_1 = t_2 \in Formulas$.
- ▶ If $\varphi \in Formulas$ and $\psi \in Formulas$ then
 - ▶ $\neg\varphi \in Formulas$
 - ▶ $\varphi \wedge \psi \in Formulas$
 - ▶ $\varphi \vee \psi \in Formulas$
 - ▶ $\varphi \rightarrow \psi \in Formulas$
- ▶ If $\varphi \in Formulas$ and $x \in Vars$ then
 - ▶ $\exists x.\varphi \in Formulas$
 - ▶ $\forall x.\varphi \in Formulas$
- ▶ Nothing else is in $Formulas$.

Note: a predicate of arity 0 is a proposition of propositional logic.

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Interpretations

Given an **alphabet** of predicates $P_1, P_2, \dots$ and functions $f_1, f_2, \dots$, each with an associated arity, a FOL **interpretation** is:

$$\mathcal{I} = (\Delta^{\mathcal{I}}, P_1^{\mathcal{I}}, P_2^{\mathcal{I}}, \dots, f_1^{\mathcal{I}}, f_2^{\mathcal{I}}, \dots)$$

where:

- ▶ $\Delta^{\mathcal{I}}$ is the domain (a set of objects)
- ▶ if P_i is a k -ary predicate, then $P_i^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \dots \times \Delta^{\mathcal{I}}$ (k times)
- ▶ if f_i is a k -ary function, then $f_i^{\mathcal{I}} : \Delta^{\mathcal{I}} \times \dots \times \Delta^{\mathcal{I}} \longrightarrow \Delta^{\mathcal{I}}$ (k times)
- ▶ if f_i is a constant (i.e., a 0-ary function), then $f_i^{\mathcal{I}} : () \longrightarrow \Delta^{\mathcal{I}}$ (i.e., f_i denotes exactly one object of the domain)

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Assignment

Let $Vars$ be a set of (individual) variables.

Def.: Given an interpretation $\mathcal{I}$, an **assignment** is a function

$$\alpha : Vars \longrightarrow \Delta^{\mathcal{I}}$$

that assigns to each variable $x \in Vars$ an object $\alpha(x) \in \Delta^{\mathcal{I}}$.

It is convenient to extend the notion of assignment to terms. We can do so by defining a function $\hat{\alpha} : Terms \longrightarrow \Delta^{\mathcal{I}}$ inductively as follows:

- ▶ $\hat{\alpha}(x) = \alpha(x)$, if $x \in Vars$
- ▶ $\hat{\alpha}(f(t_1, \dots, t_k)) = f^{\mathcal{I}}(\hat{\alpha}(t_1), \dots, \hat{\alpha}(t_k))$

Note: for constants $\hat{\alpha}(c) = c^{\mathcal{I}}$.

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Truth in an interpretation wrt an assignment

We define when a FOL formula φ is **true** in an interpretation $\mathcal{I}$ wrt an assignment α , written $\mathcal{I}, \alpha \models \varphi$:

- ▶ $\mathcal{I}, \alpha \models P(t_1, \dots, t_k)$ if $(\hat{\alpha}(t_1), \dots, \hat{\alpha}(t_k)) \in P^{\mathcal{I}}$
- ▶ $\mathcal{I}, \alpha \models t_1 = t_2$ if $\hat{\alpha}(t_1) = \hat{\alpha}(t_2)$
- ▶ $\mathcal{I}, \alpha \models \neg\varphi$ if $\mathcal{I}, \alpha \not\models \varphi$
- ▶ $\mathcal{I}, \alpha \models \varphi \wedge \psi$ if $\mathcal{I}, \alpha \models \varphi$ and $\mathcal{I}, \alpha \models \psi$
- ▶ $\mathcal{I}, \alpha \models \varphi \vee \psi$ if $\mathcal{I}, \alpha \models \varphi$ or $\mathcal{I}, \alpha \models \psi$
- ▶ $\mathcal{I}, \alpha \models \varphi \rightarrow \psi$ if $\mathcal{I}, \alpha \models \varphi$ implies $\mathcal{I}, \alpha \models \psi$
- ▶ $\mathcal{I}, \alpha \models \exists x.\varphi$ if for some $a \in \Delta^{\mathcal{I}}$ we have $\mathcal{I}, \alpha[x \mapsto a] \models \varphi$
- ▶ $\mathcal{I}, \alpha \models \forall x.\varphi$ if for every $a \in \Delta^{\mathcal{I}}$ we have $\mathcal{I}, \alpha[x \mapsto a] \models \varphi$

Here, $\alpha[x \mapsto a]$ stands for the new assignment obtained from α as follows:

$$\begin{aligned}\alpha[x \mapsto a](x) &= a \\ \alpha[x \mapsto a](y) &= \alpha(y) \quad \text{for } y \neq x\end{aligned}$$

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Open vs. closed formulas

Definitions

- ▶ A variable x in a formula φ is **free** if x does not occur in the scope of any quantifier, otherwise it is **bounded**.
- ▶ An **open formula** is a formula that has some free variable.
- ▶ A **closed formula**, also called **sentence**, is a formula that has no free variables.

For **closed formulas** (but not for open formulas) we can define what it means to be **true in an interpretation**, written $\mathcal{I} \models \varphi$, without mentioning the assignment, since the assignment α does not play any role in verifying $\mathcal{I}, \alpha \models \varphi$.

Instead, open formulas are strongly related to **queries** — cf. relational databases.

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

FOL queries

Def.: A **FOL query** is an (open) FOL formula.

When φ is a FOL query with free variables $(x_1, \dots, x_k)$, then we sometimes write it as $\varphi(x_1, \dots, x_k)$, and say that φ has **arity** k .

Given an interpretation $\mathcal{I}$, we are interested in those assignments that map the variables $x_1, \dots, x_k$ (and only those). We write an assignment α s.t. $\alpha(x_i) = a_i$, for $i = 1, \dots, k$, as $\langle a_1, \dots, a_k \rangle$.

Def.: Given an interpretation $\mathcal{I}$, the **answer to a query** $\varphi(x_1, \dots, x_k)$ is

$$\varphi(x_1, \dots, x_k)^{\mathcal{I}} = \{ \langle a_1, \dots, a_k \rangle \mid \mathcal{I}, \langle a_1, \dots, a_k \rangle \models \varphi(x_1, \dots, x_k) \}$$

Note: We will also use the notation $\varphi^{\mathcal{I}}$, which keeps the free variables implicit, and $\varphi(\mathcal{I})$ making apparent that φ becomes a functions from interpretations to set of tuples.

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

FOL boolean queries

Def.: A **FOL boolean query** is a FOL query without free variables.

Hence, the answer to a boolean query $\varphi()$ is defined as follows:

$$\varphi()^{\mathcal{I}} = \{ () \mid \mathcal{I}, \langle \rangle \models \varphi() \}$$

Such an answer is

- ▶ $()$, if $\mathcal{I} \models \varphi$
- ▶ $\emptyset$, if $\mathcal{I} \not\models \varphi$.

As an obvious convention we read $()$ as “true” and $\emptyset$ as “false”.

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

FOL formulas: logical tasks

Definitions

- ▶ **Validity:** φ is **valid** iff for all $\mathcal{I}$ and α we have that $\mathcal{I}, \alpha \models \varphi$.
- ▶ **Satisfiability:** φ is **satisfiable** iff there exists an $\mathcal{I}$ and α such that $\mathcal{I}, \alpha \models \varphi$, and **unsatisfiable** otherwise.
- ▶ **Logical implication:** φ **logically implies** ψ , written $\varphi \models \psi$ iff for all $\mathcal{I}$ and α , if $\mathcal{I}, \alpha \models \varphi$ then $\mathcal{I}, \alpha \models \psi$.
- ▶ **Logical equivalence:** φ is **logically equivalent** to ψ , iff for all $\mathcal{I}$ and α , we have that $\mathcal{I}, \alpha \models \varphi$ iff $\mathcal{I}, \alpha \models \psi$ (i.e., $\varphi \models \psi$ and $\psi \models \varphi$).

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

FOL queries – Logical tasks

- ▶ **Validity:** if φ is valid, then $\varphi^{\mathcal{I}} = \Delta^{\mathcal{I}} \times \dots \times \Delta^{\mathcal{I}}$ for all $\mathcal{I}$, i.e., the query always returns all the tuples of $\mathcal{I}$.
- ▶ **Satisfiability:** if φ is satisfiable, then $\varphi^{\mathcal{I}} \neq \emptyset$ for some $\mathcal{I}$, i.e., the query returns at least one tuple.
- ▶ **Logical implication:** if φ logically implies ψ , then $\varphi^{\mathcal{I}} \subseteq \psi^{\mathcal{I}}$ for all $\mathcal{I}$, written $\varphi \subseteq \psi$, i.e., the answer to φ is contained in that of ψ in every interpretation. This is called **query containment**.
- ▶ **Logical equivalence:** if φ is logically equivalent to ψ , then $\varphi^{\mathcal{I}} = \psi^{\mathcal{I}}$ for all $\mathcal{I}$, written $\varphi \equiv \psi$, i.e., the answer to the two queries is the same in every interpretation. This is called **query equivalence** and corresponds to query containment in both directions.

Note: These definitions can be extended to the case where we have **axioms**, i.e., **constraints** on the admissible interpretations.

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Query evaluation

Let us consider:

- ▶ a **finite alphabet**, i.e., we have a finite number of predicates and functions, and
- ▶ a **finite interpretation** $\mathcal{I}$, i.e., an interpretation (over the finite alphabet) for which $\Delta^{\mathcal{I}}$ is finite.

Then we can consider query evaluation as an algorithmic problem, and study its computational properties.

Note: To study the **computational complexity** of the problem, we need to define a corresponding decision problem.

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Query evaluation problem

Definitions

- ▶ **Query answering problem**: given a finite interpretation $\mathcal{I}$ and a FOL query $\varphi(x_1, \dots, x_k)$, compute

$$\varphi^{\mathcal{I}} = \{(a_1, \dots, a_k) \mid \mathcal{I}, \langle a_1, \dots, a_k \rangle \models \varphi(x_1, \dots, x_k)\}$$

- ▶ **Recognition problem (for query answering)**: given a finite interpretation $\mathcal{I}$, a FOL query $\varphi(x_1, \dots, x_k)$, and a tuple $(a_1, \dots, a_k)$, with $a_i \in \Delta^{\mathcal{I}}$, check whether $(a_1, \dots, a_k) \in \varphi^{\mathcal{I}}$, i.e., whether

$$\mathcal{I}, \langle a_1, \dots, a_k \rangle \models \varphi(x_1, \dots, x_k)$$

Note: The recognition problem for query answering is the decision problem corresponding to the query answering problem.

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Query evaluation algorithm

We define now an algorithm that computes the function $\text{Truth}(\mathcal{I}, \alpha, \varphi)$ in such a way that $\text{Truth}(\mathcal{I}, \alpha, \varphi) = \text{true}$ iff $\mathcal{I}, \alpha \models \varphi$.

We make use of an auxiliary function $\text{TermEval}(\mathcal{I}, \alpha, t)$ that, given an interpretation $\mathcal{I}$ and an assignment α , evaluates a term t returning an object $o \in \Delta^{\mathcal{I}}$:

```
 $\Delta^{\mathcal{I}}$  TermEval( $\mathcal{I}, \alpha, t$ ) {  
  if ( $t$  is  $x \in \text{Vars}$ )  
    return  $\alpha(x)$ ;  
  if ( $t$  is  $f(t_1, \dots, t_k)$ )  
    return  $f^{\mathcal{I}}(\text{TermEval}(\mathcal{I}, \alpha, t_1), \dots, \text{TermEval}(\mathcal{I}, \alpha, t_k))$ ;  
}
```

Then, $\text{Truth}(\mathcal{I}, \alpha, \varphi)$ can be defined by structural recursion on φ .

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Query evaluation algorithm (cont'd)

```
boolean Truth( $\mathcal{I}, \alpha, \varphi$ ) {  
  if ( $\varphi$  is  $t_1 = t_2$ )  
    return  $\text{TermEval}(\mathcal{I}, \alpha, t_1) = \text{TermEval}(\mathcal{I}, \alpha, t_2)$ ;  
  if ( $\varphi$  is  $P(t_1, \dots, t_k)$ )  
    return  $P^{\mathcal{I}}(\text{TermEval}(\mathcal{I}, \alpha, t_1), \dots, \text{TermEval}(\mathcal{I}, \alpha, t_k))$ ;  
  if ( $\varphi$  is  $\neg\psi$ )  
    return  $\neg\text{Truth}(\mathcal{I}, \alpha, \psi)$ ;  
  if ( $\varphi$  is  $\psi \circ \psi'$ )  
    return  $\text{Truth}(\mathcal{I}, \alpha, \psi) \circ \text{Truth}(\mathcal{I}, \alpha, \psi')$ ;  
  if ( $\varphi$  is  $\exists x.\psi$ ) {  
    boolean b = false;  
    for all ( $a \in \Delta^{\mathcal{I}}$ )  
      b = b  $\vee$   $\text{Truth}(\mathcal{I}, \alpha[x \mapsto a], \psi)$ ;  
    return b;  
  }  
  if ( $\varphi$  is  $\forall x.\psi$ ) {  
    boolean b = true;  
    for all ( $a \in \Delta^{\mathcal{I}}$ )  
      b = b  $\wedge$   $\text{Truth}(\mathcal{I}, \alpha[x \mapsto a], \psi)$ ;  
    return b;  
  }  
}
```

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Query evaluation – Results

Theorem (Termination of $\text{Truth}(\mathcal{I}, \alpha, \varphi)$)

The algorithm Truth terminates.

Proof. Immediate. □

Theorem (Correctness)

The algorithm Truth is sound and complete, i.e., $\mathcal{I}, \alpha \models \varphi$ if and only if $\text{Truth}(\mathcal{I}, \alpha, \varphi) = \text{true}$.

Proof. Easy, since the algorithm is very close to the semantic definition of $\mathcal{I}, \alpha \models \varphi$. □

◀ ◻ ▶ ◀ ☐ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Query evaluation – Time complexity I

Theorem (Time complexity of $\text{Truth}(\mathcal{I}, \alpha, \varphi)$)

The time complexity of $\text{Truth}(\mathcal{I}, \alpha, \varphi)$ is $(|\mathcal{I}| + |\alpha| + |\varphi|)^{|\varphi|}$, i.e., polynomial in the size of $\mathcal{I}$ and exponential in the size of φ .

Proof.

- ▶ $f^{\mathcal{I}}$ (of arity k) can be represented as k -dimensional array, hence accessing the required element can be done in time linear in $|\mathcal{I}|$.
- ▶ $\text{TermEval}(\dots)$ visits the term, so it generates a polynomial number of recursive calls, hence is time polynomial in $(|\mathcal{I}| + |\alpha| + |\varphi|)$.
- ▶ $P^{\mathcal{I}}$ (of arity k) can be represented as k -dimensional boolean array, hence accessing the required element can be done in time linear in $|\mathcal{I}|$.
- ▶ $\text{Truth}(\dots)$ for the boolean cases simply visits the formula, so generates either one or two recursive calls.

◀ ◻ ▶ ◀ ☐ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Query evaluation – Time complexity II

- ▶ $\text{Truth}(\dots)$ for the quantified cases $\exists x.\varphi$ and $\forall x.\psi$ involves looping for all elements in $\Delta^{\mathcal{I}}$ and testing the resulting assignments.
- ▶ The total number of such testings is $O(|\mathcal{I}|^{\#Vars})$.

Hence the claim holds. □

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Query evaluation – Space complexity I

Theorem (Space complexity of $\text{Truth}(\mathcal{I}, \alpha, \varphi)$)

The space complexity of $\text{Truth}(\mathcal{I}, \alpha, \varphi)$ is $|\varphi| \cdot (|\varphi| \cdot \log |\mathcal{I}|)$, i.e., logarithmic in the size of $\mathcal{I}$ and polynomial in the size of φ .

Proof.

- ▶ $f^{\mathcal{I}}(\dots)$ can be represented as k -dimensional array, hence accessing the required element requires $O(\log |\mathcal{I}|)$;
- ▶ $\text{TermEval}(\dots)$ simply visits the term, so it generates a polynomial number of recursive calls. Each activation record has a constant size, and we need $O(|\varphi|)$ activation records;
- ▶ $P^{\mathcal{I}}(\dots)$ can be represented as k -dimensional boolean array, hence accessing the required element requires $O(\log |\mathcal{I}|)$;
- ▶ $\text{Truth}(\dots)$ for the boolean cases simply visits the formula, so generates either one or two recursive calls, each requiring constant size;
- ▶ $\text{Truth}(\dots)$ for the quantified cases $\exists x.\varphi$ and $\forall x.\psi$ involves looping for all elements in $\Delta^{\mathcal{I}}$ and testing the resulting assignments;

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Query evaluation – Space complexity II

- ▶ The total number of activation records that need to be at the same time on the stack is $O(\#Vars) \leq O(|\varphi|)$.

Hence the claim holds. □

Note: the worst case form for the formula is

$$\forall x_1. \exists x_2. \dots \forall x_{n-1}. \exists x_n. P(x_1, x_2, \dots, x_{n-1}, x_n).$$

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Query evaluation – Complexity measures [Var82]

Definition (Combined complexity)

The **combined complexity** is the complexity of $\{\langle \mathcal{I}, \alpha, \varphi \rangle \mid \mathcal{I}, \alpha \models \varphi\}$, i.e., interpretation, tuple, and query are all considered part of the input.

Definition (Data complexity)

The **data complexity** is the complexity of $\{\langle \mathcal{I}, \alpha \rangle \mid \mathcal{I}, \alpha \models \varphi\}$, i.e., the query φ is fixed (and hence not considered part of the input).

Definition (Query complexity)

The **query complexity** is the complexity of $\{\langle \alpha, \varphi \rangle \mid \mathcal{I}, \alpha \models \varphi\}$, i.e., the interpretation $\mathcal{I}$ is fixed (and hence not considered part of the input).

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Query evaluation – Combined, data, query complexity

Theorem (Combined complexity of query evaluation)

The complexity of $\{\langle \mathcal{I}, \alpha, \varphi \rangle \mid \mathcal{I}, \alpha \models \varphi\}$ is:

- ▶ time: exponential
- ▶ space: PSPACE-complete — see [Var82] for hardness

Theorem (Data complexity of query evaluation)

The complexity of $\{\langle \mathcal{I}, \alpha \rangle \mid \mathcal{I}, \alpha \models \varphi\}$ is:

- ▶ time: polynomial
- ▶ space: LOGSPACE

Theorem (Query complexity of query evaluation)

The complexity of $\{\langle \alpha, \varphi \rangle \mid \mathcal{I}, \alpha \models \varphi\}$ is:

- ▶ time: exponential
- ▶ space: PSPACE-complete — see [Var82] for hardness

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ≡ ↺ 🔍 ↻

Conjunctive queries (CQs)

Def.: A **conjunctive query (CQ)** is a FOL query of the form

$$\exists \vec{y}. \text{conj}(\vec{x}, \vec{y})$$

where $\text{conj}(\vec{x}, \vec{y})$ is a conjunction (i.e., an “and”) of atoms and equalities, over the free variables $\vec{x}$, the existentially quantified variables $\vec{y}$, and possibly constants.

Note:

- ▶ CQs contain no disjunction, no negation, no universal quantification, and no function symbols besides constants.
- ▶ Hence, they correspond to relational algebra **select-project-join (SPJ) queries**.
- ▶ CQs are the most frequently asked queries.

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ≡ ↺ 🔍 ↻

Conjunctive queries and SQL – Example

Relational alphabet:

Person(name, age), Lives(person, city), Manages(boss, employee)

Query: return name and age of all persons that live in the same city as their boss.

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Conjunctive queries and SQL – Example

Relational alphabet:

Person(name, age), Lives(person, city), Manages(boss, employee)

Query: return name and age of all persons that live in the same city as their boss.

Expressed in SQL:

```
SELECT P.name, P.age
FROM Person P, Manages M, Lives L1, Lives L2
WHERE P.name = L1.person AND P.name = M.employee AND
      M.boss = L2.person AND L1.city = L2.city
```

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Conjunctive queries and SQL – Example

Relational alphabet:

Person(name, age), Lives(person, city), Manages(boss, employee)

Query: return name and age of all persons that live in the same city as their boss.

Expressed in SQL:

```
SELECT P.name, P.age
FROM Person P, Manages M, Lives L1, Lives L2
WHERE P.name = L1.person AND P.name = M.employee AND
      M.boss = L2.person AND L1.city = L2.city
```

Expressed as a CQ:

$$\exists b, e, p_1, c_1, p_2, c_2. \text{Person}(n, a) \wedge \text{Manages}(b, e) \wedge \text{Lives}(p_1, c_1) \wedge \text{Lives}(p_2, c_2) \wedge \\ n = p_1 \wedge n = e \wedge b = p_2 \wedge c_1 = c_2$$

Navigation icons: back, forward, search, etc.

Conjunctive queries and SQL – Example

Relational alphabet:

Person(name, age), Lives(person, city), Manages(boss, employee)

Query: return name and age of all persons that live in the same city as their boss.

Expressed in SQL:

```
SELECT P.name, P.age
FROM Person P, Manages M, Lives L1, Lives L2
WHERE P.name = L1.person AND P.name = M.employee AND
      M.boss = L2.person AND L1.city = L2.city
```

Expressed as a CQ:

$$\exists b, e, p_1, c_1, p_2, c_2. \text{Person}(n, a) \wedge \text{Manages}(b, e) \wedge \text{Lives}(p_1, c_1) \wedge \text{Lives}(p_2, c_2) \wedge \\ n = p_1 \wedge n = e \wedge b = p_2 \wedge c_1 = c_2$$

Or simpler: $\exists b, c. \text{Person}(n, a) \wedge \text{Manages}(b, n) \wedge \text{Lives}(n, c) \wedge \text{Lives}(b, c)$

Navigation icons: back, forward, search, etc.

Datalog notation for CQs

A CQ $q = \exists \vec{y}. \text{conj}(\vec{x}, \vec{y})$ can also be written using **datalog notation** as

$$q(\vec{x}_1) \leftarrow \text{conj}'(\vec{x}_1, \vec{y}_1)$$

where $\text{conj}'(\vec{x}_1, \vec{y}_1)$ is the list of atoms in $\text{conj}(\vec{x}, \vec{y})$ obtained by equating the variables $\vec{x}, \vec{y}$ according to the equalities in $\text{conj}(\vec{x}, \vec{y})$.

As a result of such an equality elimination, we have that $\vec{x}_1$ and $\vec{y}_1$ can contain constants and multiple occurrences of the same variable.

Def.: In the above query q , we call:

- ▶ $q(\vec{x}_1)$ the **head**;
- ▶ $\text{conj}'(\vec{x}_1, \vec{y}_1)$ the **body**;
- ▶ the variables in $\vec{x}_1$ the **distinguished variables**;
- ▶ the variables in $\vec{y}_1$ the **non-distinguished variables**.

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Conjunctive queries – Example

- ▶ Consider an **interpretation** $\mathcal{I} = (\Delta^{\mathcal{I}}, E^{\mathcal{I}})$, where $E^{\mathcal{I}}$ is a binary relation – *note that such interpretation is a (directed) graph*.
- ▶ The following **CQ** q returns all nodes that participate to a triangle in the graph:

$$\exists y, z. E(x, y) \wedge E(y, z) \wedge E(z, x)$$

- ▶ The query q in **datalog notation** becomes:

$$q(x) \leftarrow E(x, y), E(y, z), E(z, x)$$

- ▶ The query q in **SQL** is (we use $\text{Edge}(f, s)$ for $E(x, y)$):

```
SELECT E1.f
FROM Edge E1, Edge E2, Edge E3
WHERE E1.s = E2.f AND E2.s = E3.f AND E3.s = E1.f
```

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Nondeterministic evaluation of CQs

Since a CQ contains only existential quantifications, we can evaluate it by:

1. **guessing a truth assignment** for the non-distinguished variables;
2. **evaluating** the resulting formula (that has no quantifications).

```
boolean ConjTruth( $\mathcal{I}, \alpha, \exists \vec{y}. conj(\vec{x}, \vec{y})$ ) {  
    GUESS assignment  $\alpha[\vec{y} \mapsto \vec{a}]$  {  
        return Truth( $\mathcal{I}, \alpha[\vec{y} \mapsto \vec{a}], conj(\vec{x}, \vec{y})$ );  
    }  
}
```

where $\text{Truth}(\mathcal{I}, \alpha, \varphi)$ is defined as for FOL queries, considering only the required cases.

Nondeterministic CQ evaluation algorithm

```
boolean Truth( $\mathcal{I}, \alpha, \varphi$ ) {  
    if ( $\varphi$  is  $t_1 = t_2$ )  
        return TermEval( $\mathcal{I}, \alpha, t_1$ ) == TermEval( $\mathcal{I}, \alpha, t_2$ );  
    if ( $\varphi$  is  $P(t_1, \dots, t_k)$ )  
        return  $P^{\mathcal{I}}$ (TermEval( $\mathcal{I}, \alpha, t_1$ ), ..., TermEval( $\mathcal{I}, \alpha, t_k$ ));  
    if ( $\varphi$  is  $\psi \wedge \psi'$ )  
        return Truth( $\mathcal{I}, \alpha, \psi$ ) & Truth( $\mathcal{I}, \alpha, \psi'$ );  
}
```

```
 $\Delta^{\mathcal{I}}$  TermEval( $\mathcal{I}, \alpha, t$ ) {  
    if ( $t$  is a variable  $x$ ) return  $\alpha(x)$ ;  
    if ( $t$  is a constant  $c$ ) return  $c^{\mathcal{I}}$ ;  
}
```

CQ evaluation – Combined, data, and query complexity

Theorem (Combined complexity of CQ evaluation)

$\{\langle \mathcal{I}, \alpha, q \rangle \mid \mathcal{I}, \alpha \models q\}$ is *NP-complete* — see below for hardness

- ▶ time: exponential
- ▶ space: polynomial

Theorem (Data complexity of CQ evaluation)

$\{\langle \mathcal{I}, \alpha \rangle \mid \mathcal{I}, \alpha \models q\}$ is *LOGSPACE*

- ▶ time: polynomial
- ▶ space: logarithmic

Theorem (Query complexity of CQ evaluation)

$\{\langle \alpha, q \rangle \mid \mathcal{I}, \alpha \models q\}$ is *NP-complete* — see below for hardness

- ▶ time: exponential
- ▶ space: polynomial

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

3-colorability

A graph is *k-colorable* if it is possible to assign to each node one of k colors in such a way that every two nodes connected by an edge have different colors.

Def.: *3-colorability* is the following decision problem

Given a graph $G = (V, E)$, is it 3-colorable?

Theorem

3-colorability is NP-complete.

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

3-colorability

A graph is ***k*-colorable** if it is possible to assign to each node one of k colors in such a way that every two nodes connected by an edge have different colors.

Def.: **3-colorability** is the following decision problem

Given a graph $G = (V, E)$, is it 3-colorable?

Theorem

3-colorability is NP-complete.

We exploit 3-colorability to show NP-hardness of conjunctive query evaluation.

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Reduction from 3-colorability to CQ evaluation

Let $G = (V, E)$ be a graph. We define:

- ▶ An **Interpretation**: $\mathcal{I} = (\Delta^{\mathcal{I}}, E^{\mathcal{I}})$ where:
 - ▶ $\Delta^{\mathcal{I}} = \{r, g, b\}$
 - ▶ $E^{\mathcal{I}} = \{(r, g), (g, r), (r, b), (b, r), (g, b), (b, g)\}$
- ▶ A **conjunctive query**: Let $V = \{x_1, \dots, x_n\}$, then consider the boolean conjunctive query defined as:

$$q_G = \exists x_1, \dots, x_n. \bigwedge_{(x_i, x_j) \in E} E(x_i, x_j) \wedge E(x_j, x_i)$$

Theorem

G is 3-colorable iff $\mathcal{I} \models q_G$.

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

NP-hardness of CQ evaluation

The previous reduction immediately gives us the hardness for combined complexity.

Theorem

CQ evaluation is NP-hard in combined complexity.



NP-hardness of CQ evaluation

The previous reduction immediately gives us the hardness for combined complexity.

Theorem

CQ evaluation is NP-hard in combined complexity.

Note: in the previous reduction, the interpretation does not depend on the actual graph. Hence, the reduction provides also the lower-bound for query complexity.

Theorem

CQ evaluation is NP-hard in query (and combined) complexity.



Homomorphism

Let $\mathcal{I} = (\Delta^{\mathcal{I}}, P^{\mathcal{I}}, \dots, c^{\mathcal{I}}, \dots)$ and $\mathcal{J} = (\Delta^{\mathcal{J}}, P^{\mathcal{J}}, \dots, c^{\mathcal{J}}, \dots)$ be two interpretations over the same alphabet (for simplicity, we consider only constants as functions).

Def.: A **homomorphism** from $\mathcal{I}$ to $\mathcal{J}$

is a mapping $h : \Delta^{\mathcal{I}} \rightarrow \Delta^{\mathcal{J}}$ such that:

- ▶ $h(c^{\mathcal{I}}) = c^{\mathcal{J}}$
- ▶ $h(P^{\mathcal{I}}(a_1, \dots, a_k)) = P^{\mathcal{J}}(h(a_1), \dots, h(a_k))$

Note: An **isomorphism** is a homomorphism that is one-to-one and onto.

Theorem

FOL is unable to distinguish between interpretations that are isomorphic.

Proof. See any standard book on logic. $\square$

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Recognition problem and boolean query evaluation

Consider the recognition problem associated to the evaluation of a query q of arity k . Then

$$\mathcal{I}, \alpha \models q(x_1, \dots, x_k) \quad \text{iff} \quad \mathcal{I}_{\alpha, \vec{c}} \models q(c_1, \dots, c_k)$$

where $\mathcal{I}_{\alpha, \vec{c}}$ is identical to $\mathcal{I}$ but includes new constants $c_1, \dots, c_k$ that are interpreted as $c_i^{\mathcal{I}_{\alpha, \vec{c}}} = \alpha(x_i)$.

That is, we can **reduce the recognition problem to the evaluation of a boolean query**.

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Canonical interpretation of a (boolean) CQ

Let q be a conjunctive query $\exists x_1, \dots, x_n. conj$

Def.: The **canonical interpretation** $\mathcal{I}_q$ associated with q

is the interpretation $\mathcal{I}_q = (\Delta^{\mathcal{I}_q}, P^{\mathcal{I}_q}, \dots, c^{\mathcal{I}_q}, \dots)$, where

- ▶ $\Delta^{\mathcal{I}_q} = \{x_1, \dots, x_n\} \cup \{c \mid c \text{ constant occurring in } q\}$,
i.e., all the variables and constants in q ;
- ▶ $c^{\mathcal{I}_q} = c$, for each constant c in q ;
- ▶ $(t_1, \dots, t_k) \in P^{\mathcal{I}_q}$ iff the atom $P(t_1, \dots, t_k)$ occurs in q .

Sometimes the procedure for obtaining the canonical interpretation is called **freezing** of q .

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Canonical interpretation of a (boolean) CQ – Example

Consider the boolean query q

$$q(c) \leftarrow E(c, y), E(y, z), E(z, c)$$

Then, the canonical interpretation $\mathcal{I}_q$ is defined as

$$\mathcal{I}_q = (\Delta^{\mathcal{I}_q}, E^{\mathcal{I}_q}, c^{\mathcal{I}_q})$$

where

- ▶ $\Delta^{\mathcal{I}_q} = \{y, z, c\}$
- ▶ $E^{\mathcal{I}_q} = \{(c, y), (y, z), (z, c)\}$
- ▶ $c^{\mathcal{I}_q} = c$

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Canonical interpretation and (boolean) CQ evaluation

Theorem ([CM77])

For boolean CQs, $\mathcal{I} \models q$ iff there exists a homomorphism from $\mathcal{I}_q$ to $\mathcal{I}$.

Proof.

“ $\Rightarrow$ ” Let $\mathcal{I} \models q$, let α be an assignment to the existential variables that makes q true in $\mathcal{I}$, and let $\hat{\alpha}$ be its extension to constants. Then $\hat{\alpha}$ is a homomorphism from $\mathcal{I}_q$ to $\mathcal{I}$.

“ $\Leftarrow$ ” Let h be a homomorphism from $\mathcal{I}_q$ to $\mathcal{I}$. Then restricting h to the variables only we obtain an assignment to the existential variables that makes q true in $\mathcal{I}$. □

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Canonical interpretation and (boolean) CQ evaluation

The previous result can be rephrased as follows:

(The recognition problem associated to) **query evaluation can be reduced to finding a homomorphism.**

Finding a homomorphism between two interpretations (aka relational structures) is also known as solving a **Constraint Satisfaction Problem (CSP)**, a problem well-studied in AI – see also [KV98].

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Query containment

Def.: Query containment

Given two FOL queries φ and ψ of the same arity, φ is contained in ψ , denoted $\varphi \subseteq \psi$, if for all interpretations $\mathcal{I}$ and all assignments α we have that

$$\mathcal{I}, \alpha \models \varphi \text{ implies } \mathcal{I}, \alpha \models \psi$$

(In logical terms: $\varphi \models \psi$.)

Note: Query containment is of special interest in query optimization.



Query containment

Def.: Query containment

Given two FOL queries φ and ψ of the same arity, φ is contained in ψ , denoted $\varphi \subseteq \psi$, if for all interpretations $\mathcal{I}$ and all assignments α we have that

$$\mathcal{I}, \alpha \models \varphi \text{ implies } \mathcal{I}, \alpha \models \psi$$

(In logical terms: $\varphi \models \psi$.)

Note: Query containment is of special interest in query optimization.

Theorem

For FOL queries, query containment is undecidable.

Proof: Reduction from FOL logical implication. $\square$



Query containment for CQs

For CQs, query containment $q_1(\vec{x}) \subseteq q_2(\vec{x})$ can be reduced to query evaluation.

1. **Freeze the free variables**, i.e., consider them as constants.
This is possible, since $q_1(\vec{x}) \subseteq q_2(\vec{x})$ iff
 - ▶ $\mathcal{I}, \alpha \models q_1(\vec{x})$ implies $\mathcal{I}, \alpha \models q_2(\vec{x})$, for all $\mathcal{I}$ and α ; or equivalently
 - ▶ $\mathcal{I}_{\alpha, \vec{c}} \models q_1(\vec{c})$ implies $\mathcal{I}_{\alpha, \vec{c}} \models q_2(\vec{c})$, for all $\mathcal{I}_{\alpha, \vec{c}}$, where $\vec{c}$ are new constants, and $\mathcal{I}_{\alpha, \vec{c}}$ extends $\mathcal{I}$ to the new constants with $c^{\mathcal{I}_{\alpha, \vec{c}}} = \alpha(x)$.
2. **Construct the canonical interpretation $\mathcal{I}_{q_1(\vec{c})}$ of the CQ $q_1(\vec{c})$ on the left hand side** ...
3. ... and **evaluate on $\mathcal{I}_{q_1(\vec{c})}$ the CQ $q_2(\vec{c})$ on the right hand side**, i.e., check whether $\mathcal{I}_{q_1(\vec{c})} \models q_2(\vec{c})$.

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Reducing containment of CQs to CQ evaluation

Theorem ([CM77])

For CQs, $q_1(\vec{x}) \subseteq q_2(\vec{x})$ iff $\mathcal{I}_{q_1(\vec{c})} \models q_2(\vec{c})$, where $\vec{c}$ are new constants.

Proof.

“ $\Rightarrow$ ” Assume that $q_1(\vec{x}) \subseteq q_2(\vec{x})$.

- ▶ Since $\mathcal{I}_{q_1(\vec{c})} \models q_1(\vec{c})$ it follows that $\mathcal{I}_{q_1(\vec{c})} \models q_2(\vec{c})$.

“ $\Leftarrow$ ” Assume that $\mathcal{I}_{q_1(\vec{c})} \models q_2(\vec{c})$.

- ▶ By [CM77] on hom., for every $\mathcal{I}$ such that $\mathcal{I} \models q_1(\vec{c})$ there exists a homomorphism h from $\mathcal{I}_{q_1(\vec{c})}$ to $\mathcal{I}$.
- ▶ On the other hand, since $\mathcal{I}_{q_1(\vec{c})} \models q_2(\vec{c})$, again by [CM77] on hom., there exists a homomorphism h' from $\mathcal{I}_{q_2(\vec{c})}$ to $\mathcal{I}_{q_1(\vec{c})}$.
- ▶ The mapping $h \circ h'$ (obtained by composing h and h') is a homomorphism from $\mathcal{I}_{q_2(\vec{c})}$ to $\mathcal{I}$. Hence, once again by [CM77] on hom., $\mathcal{I} \models q_2(\vec{c})$.

So we can conclude that $q_1(\vec{c}) \subseteq q_2(\vec{c})$, and hence $q_1(\vec{x}) \subseteq q_2(\vec{x})$. ◻

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Query containment for CQs

For CQs, we also have that (boolean) query evaluation $\mathcal{I} \models q$ can be reduced to query containment.

Let $\mathcal{I} = (\Delta^{\mathcal{I}}, P^{\mathcal{I}}, \dots, c^{\mathcal{I}}, \dots)$.

We construct the (boolean) CQ $q_{\mathcal{I}}$ as follows:

- ▶ $q_{\mathcal{I}}$ has no existential variables (hence no variables at all);
- ▶ the constants in $q_{\mathcal{I}}$ are the elements of $\Delta^{\mathcal{I}}$;
- ▶ for each relation P interpreted in $\mathcal{I}$ and for each fact $(a_1, \dots, a_k) \in P^{\mathcal{I}}$, $q_{\mathcal{I}}$ contains one atom $P(a_1, \dots, a_k)$ (note that each $a_i \in \Delta^{\mathcal{I}}$ is a constant in $q_{\mathcal{I}}$).

Theorem

For CQs, $\mathcal{I} \models q$ iff $q_{\mathcal{I}} \subseteq q$.

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Query containment for CQs – Complexity

From the previous results and NP-completeness of combined complexity of CQ evaluation, we immediately get:

Theorem

Containment of CQs is NP-complete.

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Query containment for CQs – Complexity

From the previous results and NP-completeness of combined complexity of CQ evaluation, we immediately get:

Theorem

Containment of CQs is NP-complete.

Since CQ evaluation is NP-complete even in query complexity, the above result can be strengthened:

Theorem

Containment $q_1(\vec{x}) \subseteq q_2(\vec{x})$ of CQs is NP-complete, even when q_1 is considered fixed.



Union of conjunctive queries (UCQs)

Def.: A **union of conjunctive queries (UCQ)** is a FOL query of the form

$$\bigvee_{i=1,\dots,n} \exists \vec{y}_i. \text{conj}_i(\vec{x}, \vec{y}_i)$$

where each $\text{conj}_i(\vec{x}, \vec{y}_i)$ is a conjunction of atoms and equalities with free variables $\vec{x}$ and $\vec{y}_i$, and possibly constants.

Note: Obviously, each conjunctive query is also a union of conjunctive queries.



Datalog notation for UCQs

A union of conjunctive queries

$$q = \bigvee_{i=1, \dots, n} \exists \vec{y}_i. \text{conj}_i(\vec{x}, \vec{y}_i)$$

is written in **datalog notation** as

$$\{ \begin{array}{l} q(\vec{x}) \leftarrow \text{conj}'_1(\vec{x}, \vec{y}'_1) \\ \vdots \\ q(\vec{x}) \leftarrow \text{conj}'_n(\vec{x}, \vec{y}'_n) \end{array} \}$$

where each element of the set is the datalog expression corresponding to the conjunctive query $q_i = \exists \vec{y}_i. \text{conj}_i(\vec{x}, \vec{y}_i)$.

Note: in general, we omit the set brackets.



Evaluation of UCQs

From the definition of FOL query we have that:

$$\mathcal{I}, \alpha \models \bigvee_{i=1, \dots, n} \exists \vec{y}_i. \text{conj}_i(\vec{x}, \vec{y}_i)$$

if and only if

$$\mathcal{I}, \alpha \models \exists \vec{y}_i. \text{conj}_i(\vec{x}, \vec{y}_i) \quad \text{for some } i \in \{1, \dots, n\}.$$

Hence to evaluate a UCQ q , we simply evaluate a number (linear in the size of q) of conjunctive queries in isolation.

Hence, **evaluating UCQs has the same complexity as evaluating CQs.**



UCQ evaluation – Combined, data, and query complexity

Theorem (Combined complexity of UCQ evaluation)

$\{\langle \mathcal{I}, \alpha, q \rangle \mid \mathcal{I}, \alpha \models q\}$ is *NP-complete*.

- ▶ time: exponential
- ▶ space: polynomial

Theorem (Data complexity of UCQ evaluation)

$\{\langle \mathcal{I}, q \rangle \mid \mathcal{I}, \alpha \models q\}$ is *LOGSPACE-complete* (query q fixed).

- ▶ time: polynomial
- ▶ space: logarithmic

Theorem (Query complexity of UCQ evaluation)

$\{\langle \alpha, q \rangle \mid \mathcal{I}, \alpha \models q\}$ is *NP-complete* (interpretation $\mathcal{I}$ fixed).

- ▶ time: exponential
- ▶ space: polynomial

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Query containment for UCQs

Theorem

For UCQs, $\{q_1, \dots, q_k\} \subseteq \{q'_1, \dots, q'_n\}$ iff for each q_i there is a q'_j such that $q_i \subseteq q'_j$.

Proof.

“ $\Leftarrow$ ” Obvious.

“ $\Rightarrow$ ” If the containment holds, then we have

$\{q_1(\vec{c}), \dots, q_k(\vec{c})\} \subseteq \{q'_1(\vec{c}), \dots, q'_n(\vec{c})\}$, where $\vec{c}$ are new constants:

- ▶ Now consider $\mathcal{I}_{q_i(\vec{c})}$. We have $\mathcal{I}_{q_i(\vec{c})} \models q_i(\vec{c})$, and hence $\mathcal{I}_{q_i(\vec{c})} \models \{q_1(\vec{c}), \dots, q_k(\vec{c})\}$.
- ▶ By the containment, we have that $\mathcal{I}_{q_i(\vec{c})} \models \{q'_1(\vec{c}), \dots, q'_n(\vec{c})\}$. I.e., there exists a $q'_j(\vec{c})$ such that $\mathcal{I}_{q_i(\vec{c})} \models q'_j(\vec{c})$.
- ▶ Hence, by [CM77] on containment of CQs, we have that $q_i \subseteq q'_j$.

◻

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

Query containment for UCQs – Complexity

From the previous result, we have that we can check $\{q_1, \dots, q_k\} \subseteq \{q'_1, \dots, q'_n\}$ by at most $k \cdot n$ CQ containment checks.

We immediately get:

Theorem

Containment of UCQs is NP-complete.



References

- [CM77] A. K. Chandra and P. M. Merlin.
Optimal implementation of conjunctive queries in relational data bases.
In Proc. of the 9th ACM Symp. on Theory of Computing (STOC'77), pages 77–90, 1977.
- [KV98] P. G. Kolaitis and M. Y. Vardi.
Conjunctive-query containment and constraint satisfaction.
In Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98), pages 205–213, 1998.
- [Var82] M. Y. Vardi.
The complexity of relational query languages.
In Proc. of the 14th ACM SIGACT Symp. on Theory of Computing (STOC'82), pages 137–146, 1982.

