Overview of Part 1: First-order queries

1 First-order logic
   1 Syntax of first-order logic
   2 Semantics of first-order logic
   3 First-order logic queries

2 First-order query evaluation
   1 Query evaluation problem
   2 Complexity of query evaluation

3 Conjunctive queries
   1 Evaluation of conjunctive queries
   2 Containment of conjunctive queries
   3 Unions of conjunctive queries
Chapter I

First-Order Logic

Outline

1. Syntax of first-order logic
2. Semantics of first-order logic
3. First-order logic queries
# 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 \( \text{Vars} = \{x_1, \ldots, x_n\} \) of \textit{individual variables} (i.e., variables that denote single objects).
- A set of \textit{functions symbols}, each of given arity \( \geq 0 \).
  Functions of arity 0 are called \textit{constants}.

\textbf{Def.:} The set of \textit{Terms} is defined inductively as follows:

- \( \text{Vars} \subseteq \text{Terms} \);
- If \( t_1, \ldots, t_k \in \text{Terms} \) and \( f^k \) is a \( k \)-ary function symbol, then \( f^k(t_1, \ldots, t_k) \in \text{Terms} \);
- Nothing else is in \text{Terms}.

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

FOL syntax – Formulas

\textbf{Def.:} The set of \textit{Formulas} is defined inductively as follows:

- If \( t_1, \ldots, t_k \in \text{Terms} \) and \( P^k \) is a \( k \)-ary predicate, then \( P^k(t_1, \ldots, t_k) \in \text{Formulas} \) (atomic formulas).
- If \( t_1, t_2 \in \text{Terms} \), then \( t_1 = t_2 \in \text{Formulas} \).
- If \( \varphi \in \text{Formulas} \) and \( \psi \in \text{Formulas} \) then
  - \( \neg \varphi \in \text{Formulas} \)
  - \( \varphi \land \psi \in \text{Formulas} \)
  - \( \varphi \lor \psi \in \text{Formulas} \)
  - \( \varphi \rightarrow \psi \in \text{Formulas} \)
- If \( \varphi \in \text{Formulas} \) and \( x \in \text{Vars} \) then
  - \( \exists x.\varphi \in \text{Formulas} \)
  - \( \forall x.\varphi \in \text{Formulas} \)
- Nothing else is in \text{Formulas}.
Outline

1 Syntax of first-order logic
2 Semantics of first-order logic
3 First-order logic queries

Interpretations

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

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

where:

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

Let $\text{Vars}$ be a set of (individual) variables.

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

$$\alpha : \text{Vars} \rightarrow \Delta^\mathcal{I}$$

that assigns to each variable $x \in \text{Vars}$ 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 $\hat{\alpha} : \text{Terms} \rightarrow \Delta^\mathcal{I}$ inductively as follows:

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

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

Truth in an interpretation wrt an assignment

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

- $\mathcal{I}, \alpha \models P(t_1, \ldots, t_k)$ if $(\hat{\alpha}(t_1), \ldots, \hat{\alpha}(t_k)) \in P^\mathcal{I}$
- $\mathcal{I}, \alpha \models t_1 = t_2$ if $\hat{\alpha}(t_1) = \hat{\alpha}(t_2)$
- $\mathcal{I}, \alpha \models \neg \varphi$ if $\mathcal{I}, \alpha \not\models \varphi$
- $\mathcal{I}, \alpha \models \varphi \land \psi$ if $\mathcal{I}, \alpha \models \varphi$ and $\mathcal{I}, \alpha \models \psi$
- $\mathcal{I}, \alpha \models \varphi \lor \psi$ 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 $\alpha$ as follows:

$$\alpha[x \mapsto a](x) = a$$

$$\alpha[x \mapsto a](y) = \alpha(y) \quad \text{for } y \neq x$$
Open vs. closed formulas

Definitions

- A variable $x$ in a formula $\varphi$ 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 $I \models \varphi$, without mentioning the assignment, since the assignment $\alpha$ does not play any role in verifying $I, \alpha \models \varphi$.

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

Outline

1. Syntax of first-order logic
2. Semantics of first-order logic
3. First-order logic queries
**FOL queries**

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

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

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

**Def.:** Given an interpretation \( I \), the answer to a query \( \varphi(x_1, \ldots, x_k) \) is

\[
\varphi(x_1, \ldots, x_k)^I = \{ \langle a_1, \ldots, a_k \rangle \mid I, \langle a_1, \ldots, a_k \rangle \models \varphi(x_1, \ldots, x_k) \}
\]

**Note:** We will also use the notation \( \varphi^I \), which keeps the free variables implicit, and \( \varphi(I) \) making apparent that \( \varphi \) 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:

\[
\varphi()^I = \{ () \mid I, () \models \varphi() \}
\]

Such an answer is

- \( () \), if \( I \models \varphi \)
- \( \emptyset \), if \( I \not\models \varphi \).

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

---

D. Calvanese

Part 1: First-Order Queries

KBDB – 2007/2008 (15/66)
FOL formulas: logical tasks

Definitions

- **Validity**: \( \varphi \) is valid iff for all \( I \) and \( \alpha \) we have that \( I, \alpha \models \varphi \).

- **Satisfiability**: \( \varphi \) is satisfiable iff there exists an \( I \) and \( \alpha \) such that \( I, \alpha \models \varphi \), and unsatisfiable otherwise.

- **Logical implication**: \( \varphi \) logically implies \( \psi \), written \( \varphi \models \psi \) iff for all \( I \) and \( \alpha \), if \( I, \alpha \models \varphi \) then \( I, \alpha \models \psi \).

- **Logical equivalence**: \( \varphi \) is logically equivalent to \( \psi \), iff for all \( I \) and \( \alpha \), we have that \( I, \alpha \models \varphi \) iff \( I, \alpha \models \psi \) (i.e., \( \varphi \models \psi \) and \( \psi \models \varphi \)).

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

FOL queries – Logical tasks

- **Validity**: if \( \varphi \) is valid, then \( \varphi^I = \Delta^I \times \cdots \times \Delta^I \) for all \( I \), i.e., the query always returns all the tuples of \( I \).

- **Satisfiability**: if \( \varphi \) is satisfiable, then \( \varphi^I \neq \emptyset \) for some \( I \), i.e., the query returns at least one tuple.

- **Logical implication**: if \( \varphi \) logically implies \( \psi \), then \( \varphi^I \subseteq \psi^I \) for all \( I \), written \( \varphi \preceq \psi \), i.e., the answer to \( \varphi \) is contained in that of \( \psi \) in every interpretation. This is called query containment.

- **Logical equivalence**: if \( \varphi \) is logically equivalent to \( \psi \), then \( \varphi^I = \psi^I \) for all \( I \), 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.
Chapter II

First-Order Query Evaluation

Outline

4 Query evaluation problem

5 Complexity of query evaluation
Let us consider:

- a finite alphabet, i.e., we have a finite number of predicates and functions, and
- a finite interpretation $I$, i.e., an interpretation (over the finite alphabet) for which $\Delta^I$ 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 $I$ and a FOL query $\varphi(x_1, \ldots, x_k)$, compute

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

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

$$I, \langle a_1, \ldots, a_k \rangle \models \varphi(x_1, \ldots, x_k)$$

**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 $\text{Truth}(I, \alpha, \varphi)$ in such a way that $\text{Truth}(I, \alpha, \varphi) = \text{true}$ iff $I, \alpha \models \varphi$.

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

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

Then, $\text{Truth}(I, \alpha, \varphi)$ can be defined by structural recursion on $\varphi$. 
Query evaluation algorithm (cont’d)

```java
boolean Truth(I, α, φ) {
    if (φ is t₁ = t₂)
        return TermEval(I, α, t₁) = TermEval(I, α, t₂);
    if (φ is P(t₁, ..., tₖ))
        return P(I)(TermEval(I, α, t₁), ..., TermEval(I, α, tₖ));
    if (φ is ¬ψ)
        return ¬Truth(I, α, ψ);
    if (φ is ψ o ψ′)
        return Truth(I, α, ψ) o Truth(I, α, ψ′);
    if (φ is ∃x.ψ) {
        boolean b = false;
        for all (a ∈ Δ_I)
            b = b \lor Truth(I, α[x↦→a], ψ);
        return b;
    }
    if (φ is ∀x.ψ) {
        boolean b = true;
        for all (a ∈ Δ_I)
            b = b \land Truth(I, α[x↦→a], ψ);
        return b;
    }
}
```

Theorem (Termination of Truth(I, α, φ))

The algorithm Truth terminates.

Proof. Immediate.

Theorem (Correctness)

The algorithm Truth is sound and complete, i.e., I, α |= ϕ if and only if Truth(I, α, φ) = true.

Proof. Easy, since the algorithm is very close to the semantic definition of I, α |= ϕ.
Query evaluation problem

Complexity of query evaluation

Chap. 2: First-Order Query Evaluation

Outline

4 Query evaluation problem

5 Complexity of query evaluation

Query evaluation – Time complexity I

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

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

Proof.

- \( f^I \) (of arity \( k \)) can be represented as \( k \)-dimensional array, hence accessing the required element can be done in time linear in \(|I|\).

- \( \text{TermEval}(\ldots) \) visits the term, so it generates a polynomial number of recursive calls, hence is time polynomial in \((|I| + |\alpha| + |\varphi|)\).
Query evaluation – Time complexity II

- \( P^I \) (of arity \( k \)) can be represented as \( k \)-dimensional boolean array, hence accessing the required element can be done in time linear in \( |I| \).

- \( \text{Truth}(\ldots) \) for the boolean cases simply visits the formula, so generates either one or two recursive calls.

- \( \text{Truth}(\ldots) \) for the quantified cases \( \exists x. \varphi \) and \( \forall x. \psi \) involves looping for all elements in \( \Delta I \) and testing the resulting assignments.

- The total number of such testings is \( O(|I|^\#\text{Vars}) \).

Hence the claim holds.

Proof:

- \( f^I(\ldots) \) can be represented as \( k \)-dimensional array, hence accessing the required element requires \( O(\log |I|) \);

- \( \text{TermEval}(\ldots) \) 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;

- \( P^I(\ldots) \) can be represented as \( k \)-dimensional boolean array, hence accessing the required element requires \( O(\log |I|) \);

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

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

Proof.

- \( f^I(\ldots) \) can be represented as \( k \)-dimensional array, hence accessing the required element requires \( O(\log |I|) \);

- \( \text{TermEval}(\ldots) \) 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;

- \( P^I(\ldots) \) can be represented as \( k \)-dimensional boolean array, hence accessing the required element requires \( O(\log |I|) \);
Query evaluation – Space complexity II

- \textit{Truth}(\ldots) for the boolean cases simply visits the formula, so generates either one or two recursive calls, each requiring constant size;
- \textit{Truth}(\ldots) for the quantified cases \(\exists x.\varphi\) and \(\forall x.\psi\) involves looping for all elements in \(\Delta^I\) and testing the resulting assignments;
- The total number of activation records that need to be at the same time on the stack is \(O(\#\text{Vars}) \leq O(|\varphi|)\).

Hence the claim holds.

\textbf{Note:} the worst case form for the formula is

\[ \forall x_1.\exists x_2.\cdots\forall x_{n-1}.\exists x_n. P(x_1, x_2, \ldots, x_{n-1}, x_n). \]

---

Query evaluation – Complexity measures \cite{Var82}

\textbf{Definition (Combined complexity)}

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

\textbf{Definition (Data complexity)}

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

\textbf{Definition (Query complexity)}

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

Complexity of query evaluation

Chap. 2: First-Order Query Evaluation

Query evaluation – Combined, data, query complexity

Theorem (Combined complexity of query evaluation)

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

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

Theorem (Data complexity of query evaluation)

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

- time: polynomial
- space: in LogSPACE

Theorem (Query complexity of query evaluation)

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

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

Chapter III

Conjunctive Queries
Outline

6 Evaluation of conjunctive queries

7 Containment of conjunctive queries

8 Unions of conjunctive queries
Conjunctive queries (CQs)

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

\[ \exists \bar{y}. \text{conj}(\bar{x}, \bar{y}) \]

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

Expressed in SQL:

```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:

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

Or simpler: \[ \exists b, c. \text{Person}(n, a) \land \text{Manages}(b, n) \land \text{Lives}(n, c) \land \text{Lives}(b, c) \]
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:

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

- 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 $E(x, y)$):

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

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

boolean Truth(\(I, \alpha, \varphi\)) {
    if (\(\varphi\) is \(t_1 = t_2\))
        return TermEval(\(I, \alpha, t_1\)) = TermEval(\(I, \alpha, t_2\));
    if (\(\varphi\) is \(P(t_1, \ldots, t_k)\))
        return \(P^I(\text{TermEval}(I, \alpha, t_1), \ldots, \text{TermEval}(I, \alpha, t_k))\);
    if (\(\varphi\) is \(\psi \land \psi'\))
        return Truth(\(I, \alpha, \psi\) \(\land\) Truth(\(I, \alpha, \psi'\));
}

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

CQ evaluation – Combined, data, and query complexity

Theorem (Combined complexity of CQ evaluation)
\{\langle I, \alpha, q \rangle \mid I, \alpha \models q \}\text{ is NP-complete} — see below for hardness.
- time: exponential
- space: polynomial

Theorem (Data complexity of CQ evaluation)
\{\langle I, \alpha \rangle \mid I, \alpha \models q \}\text{ is in LOGSPACE}
- time: polynomial
- space: logarithmic

Theorem (Query complexity of CQ evaluation)
\{\langle \alpha, q \rangle \mid I, \alpha \models q \}\text{ 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.

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**: $I = (\Delta^I, E^I)$ where:
  - $\Delta^I = \{r, g, b\}$
  - $E^I = \{(r, g), (g, r), (r, b), (b, r), (g, b), (b, g)\}$

- A **conjunctive query**: Let $V = \{x_1, \ldots, x_n\}$, then consider the boolean conjunctive query defined as:

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

**Theorem**

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

**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.

*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.
Outline

6 Evaluation of conjunctive queries
7 Containment of conjunctive queries
8 Unions of conjunctive queries

Homomorphism

Let $\mathcal{I} = (\Delta^\mathcal{I}, P^\mathcal{I}, \ldots, c^\mathcal{I}, \ldots)$ and $\mathcal{J} = (\Delta^\mathcal{J}, P^\mathcal{J}, \ldots, c^\mathcal{J}, \ldots)$ 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:

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

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.
## Recognition problem and boolean query evaluation

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

$$I, \alpha \models q(x_1, \ldots, x_k) \iff I, \alpha, \vec{c} \models q(c_1, \ldots, c_k)$$

where $I, \alpha, \vec{c}$ is identical to $I$ but includes new constants $c_1, \ldots, c_k$ that are interpreted as $c_i = \alpha(x_i)$. 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 $\exists x_1, \ldots, x_n. \text{conj}$

**Def.: The canonical interpretation $I_q$ associated with $q$** is the interpretation $I_q = (\Delta I_q, P I_q, \ldots, c I_q, \ldots)$, where

- $\Delta I_q = \{x_1, \ldots, x_n\} \cup \{c \mid c \text{ constant occurring in } q\}$, i.e., all the variables and constants in $q$;
- $c I_q = c$, for each constant $c$ in $q$;
- $(t_1, \ldots, t_k) \in P I_q$ iff the atom $P(t_1, \ldots, t_k)$ 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 \( I_q \) is defined as

\[
I_q = (\Delta I_q, E I_q, c I_q)
\]

where

- \( \Delta I_q = \{y, z, c\} \)
- \( E I_q = \{(c, y), (y, z), (z, c)\} \)
- \( c I_q = c \)

Canonical interpretation and (boolean) CQ evaluation

**Theorem ([CM77])**

For boolean CQs, \( I \models q \) iff there exists a homomorphism from \( I_q \) to \( I \).

**Proof.**

“⇒” Let \( I \models q \), let \( \alpha \) be an assignment to the existential variables that makes \( q \) true in \( I \), and let \( \hat{\alpha} \) be its extension to constants. Then \( \hat{\alpha} \) is a homomorphism from \( I_q \) to \( I \).

“⇐” Let \( h \) be a homomorphism from \( I_q \) to \( I \). Then restricting \( h \) to the variables only we obtain an assignment to the existential variables that makes \( q \) true in \( I \).
The previous result can be rephrased as follows:

\[(The\ recognition\ problem\ associated\ to)\ \text{query}\ evaluation\ can\ be\ reduced\ to\ finding\ a\ homomorphism.\]

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 $\varphi$ and $\psi$ of the same arity, $\varphi$ is contained in $\psi$, denoted $\varphi \subseteq \psi$, if for all interpretations $\mathcal{I}$ and all assignments $\alpha$ 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. □
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
   - $I, \alpha \models q_1(\vec{x})$ implies $I, \alpha \models q_2(\vec{x})$, for all $I$ and $\alpha$; or equivalently
   - $I_{\alpha, \vec{c}} \models q_1(\vec{c})$ implies $I_{\alpha, \vec{c}} \models q_2(\vec{c})$, for all $I_{\alpha, \vec{c}}$, where $\vec{c}$ are new constants, and $I_{\alpha, \vec{c}}$ extends $I$ to the new constants with $c_{I_{\alpha, \vec{c}}} = \alpha(x)$.

2. **Construct the canonical interpretation $I_{q_1(\vec{c})}$ of the CQ $q_1(\vec{c})$ on the left hand side . . .

3. . . and **evaluate on $I_{q_1(\vec{c})}$** the CQ $q_2(\vec{c})$ on the right hand side, i.e., check whether $I_{q_1(\vec{c})} \models q_2(\vec{c})$.

Reducing containment of CQs to CQ evaluation

**Theorem ([CM77])**

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

**Proof.**

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

- Since $I_{q_1(\vec{c})} \models q_1(\vec{c})$ it follows that $I_{q_1(\vec{c})} \models q_2(\vec{c})$.

“⇐” Assume that $I_{q_1(\vec{c})} \models q_2(\vec{c})$.

- By [CM77] on hom., for every $I$ such that $I \models q_1(\vec{c})$ there exists a homomorphism $h$ from $I_{q_1(\vec{c})}$ to $I$.

- On the other hand, since $I_{q_1(\vec{c})} \models q_2(\vec{c})$, again by [CM77] on hom., there exists a homomorphism $h'$ from $I_{q_2(\vec{c})}$ to $I_{q_1(\vec{c})}$.

- The mapping $h \circ h'$ (obtained by composing $h$ and $h'$) is a homomorphism from $I_{q_2(\vec{c})}$ to $I$. Hence, once again by [CM77] on hom., $I \models q_2(\vec{c})$.

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 $I \models q$ can be reduced to query containment.

Let $I = (\Delta^I, P^I, \ldots, e^I, \ldots)$.

We construct the (boolean) CQ $q_I$ as follows:

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

**Theorem**

For CQs, $I \models q$ iff $q_I \subseteq q$.

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.

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

\[ \bigvee_{i=1,\ldots,n} \exists \overline{y}_i \cdot \text{conj}_i(\overline{x}, \overline{y}_i) \]

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

**Note:** Obviously, each conjunctive query is also a of union of conjunctive queries.
### Datalog notation for UCQs

A union of conjunctive queries

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

is written in **datalog notation** as

\[
\{ q(\vec{x}) \leftarrow \text{conj}'_1(\vec{x}, \vec{y}_1') \\
\vdots \\
q(\vec{x}) \leftarrow \text{conj}'_n(\vec{x}, \vec{y}_n') \}
\]

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:

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

if and only if

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

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)**

\[ \{ \langle I, \alpha, q \rangle \mid I, \alpha \models q \} \] is NP-complete.
- time: exponential
- space: polynomial

**Theorem (Data complexity of UCQ evaluation)**

\[ \{ \langle I, q \rangle \mid I, \alpha \models q \} \] is in LOGSPACE (query q fixed).
- time: polynomial
- space: logarithmic

**Theorem (Query complexity of UCQ evaluation)**

\[ \{ \langle \alpha, q \rangle \mid I, \alpha \models q \} \] is NP-complete (interpretation I fixed).
- time: exponential
- space: polynomial

### Query containment for UCQs

**Theorem**

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

**Proof.**

"\( \Rightarrow \)" Obvious.

"\( \Leftarrow \)" If the containment holds, then we have \( \{ q_1(c), \ldots, q_k(c) \} \subseteq \{ q'_1(c), \ldots, q'_n(c) \} \), where \( c \) are new constants:
- Now consider \( I_{q_i}(c) \). We have \( I_{q_i}(c) \models q_i(c) \), and hence \( I_{q_i}(c) \models \{ q_1(c), \ldots, q_k(c) \} \).
- By the containment, we have that \( I_{q_i}(c) \models \{ q'_1(c), \ldots, q'_n(c) \} \). I.e., there exists a \( q'_j(c) \) such that \( I_{q_i}(c) \models q'_j(c) \).
- Hence, by [CM77] on containment of CQs, we have that \( q_i \subseteq q'_j \).

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

We immediately get:

**Theorem**

Containment of UCQs is NP-complete.

---

**References**

Optimal implementation of conjunctive queries in relational data bases.

Conjunctive-query containment and constraint satisfaction.

[Var82] M. Y. Vardi.
The complexity of relational query languages.