Knowledge Representation and Ontologies

Part 3: Query Answering in Databases and Ontologies

Diego Calvanese

Faculty of Computer Science
Master of Science in Computer Science

A.Y. 2011/2012

Freie Universität Bozen
Libera Università di Bolzano
Free University of Bozen · Bolzano

Part 3

Query answering in databases and ontologies

Outline of Part 3

Outline of Part 3

1. Query answering in databases
   - First-order logic queries
   - Query evaluation problem
   - Conjunctive queries and homomorphisms
   - Unions of conjunctive queries
2. Querying databases and ontologies
3. Query answering in Description Logics
4. References

Outline of Part 3

Queries

- A **query** is a mechanism to extract new information from given information stored in some form. The extracted information is called the **answer** to the query.
- In the most general sense, a query is an arbitrary (computable) function, from some input to some output.
- Typically, one is interested in queries expressed in some (restricted) query language that provides guarantees on the computational properties of computing answers to queries.
- Here we consider queries that:
  - are expressed over a relational alphabet, and
  - return as result a relation, i.e., a set of tuples of objects satisfying a certain condition.

A very prominent query language of this form is **first-order logic**.

First-order logic

We consider now first-order logic with equality (FOL) as a mechanism to express queries.

- FOL is the logic to speak about **objects**, which constitute the domain of discourse (or universe).
- FOL is concerned about **properties** of these objects and **relations** over objects (corresponding to unary and n-ary **predicates**, respectively).
- 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:

- Each variable is a term, i.e., $Vars \subseteq Terms$;
- 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 $t_1, \ldots, t_k \in \mathit{Terms}$ and P^k is a k-ary predicate, then $P^k(t_1, \ldots, t_k) \in \mathit{Formulas}$ (atomic formulas).
- If $t_1, t_2 \in \mathit{Terms}$, then $t_1 = t_2 \in \mathit{Formulas}$.
- If $\varphi \in \mathit{Formulas}$ and $\psi \in \mathit{Formulas}$ then
  - $\neg\varphi \in \mathit{Formulas}$
  - $\varphi \land \psi \in \mathit{Formulas}$
  - $\varphi \lor \psi \in \mathit{Formulas}$
  - $\varphi \rightarrow \psi \in \mathit{Formulas}$
- If $\varphi \in \mathit{Formulas}$ and $x \in \mathit{Vars}$ then
  - $\exists x.\varphi \in \mathit{Formulas}$
  - $\forall x.\varphi \in \mathit{Formulas}$
- Nothing else is in $\mathit{Formulas}$.

Note: a predicate of arity 0 is a proposition (as in propositional logic).

Interpretations

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

$$\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$$

where:

- $\Delta^{\mathcal{I}}$ is the interpretation domain (a set of objects);
- $\cdot^{\mathcal{I}}$ is the interpretation function that interprets predicates and function symbols as follows:
  - 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, $k \geq 1$, then $f_i^{\mathcal{I}} : \Delta^{\mathcal{I}} \times \cdots \times \Delta^{\mathcal{I}} \longrightarrow \Delta^{\mathcal{I}}$
  - 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 $x \in Vars$
- $\hat{\alpha}(f(t_1, \ldots, t_k)) = f^{\mathcal{I}}(\hat{\alpha}(t_1), \ldots, \hat{\alpha}(t_k))$

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

Truth in an interpretation wrt an assignment

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

$$
\begin{array}{ll}
\mathcal{I}, \alpha \models P(t_1, \ldots, t_k), & \text{if } (\hat{\alpha}(t_1), \ldots, \hat{\alpha}(t_k)) \in P^{\mathcal{I}} \\
\mathcal{I}, \alpha \models t_1 = t_2, & \text{if } \hat{\alpha}(t_1) = \hat{\alpha}(t_2) \\
\mathcal{I}, \alpha \models \neg\varphi, & \text{if } \mathcal{I}, \alpha \not\models \varphi \\
\mathcal{I}, \alpha \models \varphi \land \psi, & \text{if } \mathcal{I}, \alpha \models \varphi \text{ and } \mathcal{I}, \alpha \models \psi \\
\mathcal{I}, \alpha \models \varphi \lor \psi, & \text{if } \mathcal{I}, \alpha \models \varphi \text{ or } \mathcal{I}, \alpha \models \psi \\
\mathcal{I}, \alpha \models \varphi \rightarrow \psi, & \text{if } \mathcal{I}, \alpha \models \varphi \text{ implies } \mathcal{I}, \alpha \models \psi \\
\mathcal{I}, \alpha \models \exists x.\varphi, & \text{if for some } a \in \Delta^{\mathcal{I}} \text{ we have } \mathcal{I}, \alpha[x \mapsto a] \models \varphi \\
\mathcal{I}, \alpha \models \forall x.\varphi, & \text{if for every } a \in \Delta^{\mathcal{I}} \text{ we have } \mathcal{I}, \alpha[x \mapsto a] \models \varphi
\end{array}
$$

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

$$
\begin{array}{l}
\alpha[x \mapsto a](x) = a \\
\alpha[x \mapsto a](y) = \alpha(y), \qquad \text{for } y \neq x
\end{array}
$$

Note: we have assumed that variables are standardized apart.

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 **bound**.
- 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 $(x_1, \ldots, x_k)$, then we sometimes write it as $\varphi(x_1, \ldots, x_k)$, and say that φ has **arity** k.

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

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

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

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

- the empty tuple $()$, if $\mathcal{I} \models \varphi$
- the empty set $\emptyset$, 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 $\varphi^{\mathcal{I}} = \Delta^{\mathcal{I}} \times \cdots \times \Delta^{\mathcal{I}}$ 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.

Outline of Part 3

1. Query answering in databases
   - First-order logic queries
   - Query evaluation problem
   - Conjunctive queries and homomorphisms
   - Unions of conjunctive queries
2. Querying databases and ontologies
3. Query answering in Description Logics
4. References

Query evaluation

Let us consider a **finite interpretation** $\mathcal{I}$, i.e., an interpretation (over the finite alphabet) for which $\Delta^{\mathcal{I}}$ is finite.

Note: whenever we have to evaluate a query, we are only interested in the interpretation of the relation and function symbols that appear in the query, which are **finitely many**.

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 $\varphi(x_1, \ldots, x_k)$, compute

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

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

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

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

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

```
Δ^I TermEval(I,α,t) {
   if (t is x ∈ Vars)
      return α(x);
   if (t is f(t_1,...,t_k))
      return f^I(TermEval(I,α,t_1),...,TermEval(I,α,t_k));
}
```

Note: constants are considered as function symbols of arity 0

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

Query evaluation algorithm (cont'd)

```
boolean Truth(𝓘,α,φ) {
  if (φ is t_1 = t_2)
    return TermEval(𝓘,α,t_1) = TermEval(𝓘,α,t_2);
  if (φ is P(t_1, ..., t_k))
    return P^𝓘(TermEval(𝓘,α,t_1),...,TermEval(𝓘,α,t_k));
  if (φ is ¬ψ)
    return ¬Truth(𝓘,α,ψ);
  if (φ is ψ ∘ ψ')
    return Truth(𝓘,α,ψ) ∘ Truth(𝓘,α,ψ');
  if (φ is ∃x.ψ) {
    boolean b = false;
    for all (a ∈ Δ^𝓘)
       b = b ∨ Truth(𝓘,α[x ↦ a],ψ);
    return b;
  }
  if (φ is ∀x.ψ) {
    boolean b = true;
    for all (a ∈ Δ^𝓘)
       b = b ∧ Truth(𝓘,α[x ↦ a],ψ);
    return b;
  }
}
```

Query evaluation – Results

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

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 $\texttt{Truth}(\mathcal{I}, \alpha, \varphi) = \texttt{true}$.

Proof. Easy, since the structure of the algorithm directly reflects the inductive definition of $\mathcal{I}, \alpha \models \varphi$. □

Query evaluation – Time complexity I

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

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

Proof.

- Each $f^{\mathcal{I}}$ (of arity k) can be represented as a k-dimensional array, hence accessing the required element can be done in time linear in $|\mathcal{I}|$.
- $\texttt{TermEval}(\ldots)$ visits the term, so it generates a polynomial number of recursive calls, hence runs in time polynomial in $(|\mathcal{I}| + |\alpha| + |\varphi|)$.
- Each $P^{\mathcal{I}}$ (of arity k) can be represented as a k-dimensional boolean array, hence accessing the required element can be done in time linear in $|\mathcal{I}|$.

Query evaluation – Time complexity II

- $\texttt{Truth}(\ldots)$ for the boolean cases simply visits the formula, so generates either one or two recursive calls.
- $\texttt{Truth}(\ldots)$ 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 tests is $O(|\mathcal{I}|^{\sharp Vars})$.

Hence the claim holds. $\square$

Query evaluation – Space complexity I

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

The space complexity of $\mathtt{Truth}(\mathcal{I}, \alpha, \varphi)$ 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.

- Each $f^{\mathcal{I}}(\ldots)$ can be represented as a k-dimensional array, hence accessing the required element requires $O(\log |\mathcal{I}|)$ space.
- $\mathtt{TermEval}(\ldots)$ simply visits the term, so it generates a polynomial number of recursive calls. Each activation record has $O(\log |\mathcal{I}|)$ size, and we need $O(|\varphi|)$ activation records.
- Each $P^{\mathcal{I}}(\ldots)$ can be represented as a k-dimensional boolean array, hence accessing the required element requires $O(\log |\mathcal{I}|)$ space.

unibz.it

Query evaluation – Space complexity II

- $\texttt{Truth}(\ldots)$ for the boolean cases simply visits the formula, so generates either one or two recursive calls, each requiring constant space.
- $\texttt{Truth}(\ldots)$ 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 activation records that need to be at the same time on the stack is $O(\sharp \mathit{Vars}) \leq O(|\varphi|)$.

Hence the claim holds. $\square$

Note: the worst case form for the formula is

$$Q_1 x_1.Q_2 x_2.\cdots Q_n x_n.P(x_1, x_2, \ldots, x_{n-1}, x_n).$$

where each Q_i is one of $\forall$ or $\exists$.

Query evaluation – Complexity measures [Vardi, 1982]

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 [Vardi, 1982] 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: in 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 [Vardi, 1982] for hardness

Outline of Part 3

(Union of) Conjunctive queries – (U)CQs

(Unions of) **conjunctive queries** are an important class of queries:

- A (U)CQ is a FOL query using only conjunction, existential quantification (and disjunction).
- Hence, UCQs contain no negation, no universal quantification, and no function symbols besides constants.
- Correspond to SQL/relational algebra **(union) select-project-join (SPJ) queries** – the most frequently asked queries.
- (U)CQs exhibit nice computational and semantic properties, and have been studied extensively in database theory.
- They are important in practice, since relational database engines are specifically optimized for CQs.

Definition of conjunctive queries (CQs)

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

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

where $conj(\vec{x}, \vec{y})$ is a conjunction 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.

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: (the distinguished variables are the blue ones)

$$\exists b, e, p_1, c_1, p_2, c_2.\mathsf{Person}(n, a) \land \mathsf{Manages}(b, e) \land \mathsf{Lives}(p1, c1) \land \mathsf{Lives}(p2, c2) \land \\ n = p1 \ \land \ n = e \ \land \ b = p2 \ \land \ c1 = c2$$

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

Datalog notation for CQs

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

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

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

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**;
- $conj'(\vec{x}_1,\vec{y}_1)$ 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 the alphabet $\Sigma = \{E/2\}$ and an **interpretation** $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$. Note that $E^{\mathcal{I}}$ is a binary relation, i.e., $\mathcal{I}$ 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 `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 variable assignment** for the non-distinguished variables;
2. **evaluating** the resulting formula (that has no quantifications).

We define a boolean function for CQ evaluation:

```
boolean ConjTruth(𝓘,α,∃y⃗.conj(x⃗,y⃗)) {
  GUESS assignment α[y⃗ ↦ a⃗] {
     return Truth(𝓘,α[y⃗ ↦ a⃗],conj(x⃗,y⃗));
}
```

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

Nondeterministic CQ evaluation algorithm

Specifically, for CQs, $\texttt{Truth}(\mathcal{I}, \alpha, \varphi)$ is defined as follows:

```
boolean Truth(𝓘,α,φ) {
  if (φ is t_1 = t_2)
    return TermEval(𝓘,α,t_1) = TermEval(𝓘,α,t_2);
  if (φ is P(t_1,...,t_k))
    return P^𝓘(TermEval(𝓘,α,t_1),...,TermEval(𝓘,α,t_k));
  if (φ is ψ ∧ ψ')
    return Truth(𝓘,α,ψ) ∧ Truth(𝓘,α,ψ');
}

Δ^𝓘 TermEval(𝓘,α,t) {
   if (t is a variable x) return α(x);
   if (t is a constant c) return c^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 **in** $\textsc{LogSpace}$

- 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

An undirected 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 an undirected 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 an undirected graph (without edges connecting a node to itself). We consider a relational alphabet consisting of a single binary relation Edge and define:

- An **Interpretation**: $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where:
  - $\Delta^{\mathcal{I}} = \{\mathsf{r}, \mathsf{g}, \mathsf{b}\}$
  - $\mathsf{Edge}^{\mathcal{I}} = \{(\mathsf{r}, \mathsf{g}), (\mathsf{g}, \mathsf{r}), (\mathsf{r}, \mathsf{b}), (\mathsf{b}, \mathsf{r}), (\mathsf{g}, \mathsf{b}), (\mathsf{b}, \mathsf{g})\}$
- A **conjunctive query:** Let $V = \{v_1, \ldots, v_n\}$, then consider the boolean conjunctive query defined as:

$$q_G \; = \; \exists x_1, \ldots, x_n. \bigwedge_{\{v_i, v_j\} \in E} \mathsf{Edge}(x_i, x_j) \wedge \mathsf{Edge}(x_j, x_i)$$

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.

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 $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ and $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ 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 $h : \Delta^{\mathcal{I}} \to \Delta^{\mathcal{J}}$ that preserves constants and relations, i.e., such that:

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

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

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

where $\mathcal{I}_{\alpha,\vec{c}}$ is identical to $\mathcal{I}$ but includes new constants $c_1, \ldots, c_k$ 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 boolean conjunctive query $\quad \exists x_1, \ldots, x_n.conj$

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

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

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

$$\mathcal{I}_q \;=\; (\Delta^{\mathcal{I}_q}, \cdot^{\mathcal{I}_q})$$

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 ([Chandra and Merlin, 1977])

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 $\mathcal{I}$. $\square$

Canonical interpretation and CQ evaluation – Example

Consider the boolean query $q() \leftarrow R_1(x,y), R_2(y,z), R_1(x,z)$.

The canonical interpretation of q is $\mathcal{I}_q = (\Delta^{\mathcal{I}_q}, \cdot^{\mathcal{I}_q})$, where

$$\Delta^{\mathcal{I}_q} = \{x,y,z\}, \qquad R_1^{\mathcal{I}_q} = \{(x,y),(x,z)\} \qquad R_2^{\mathcal{I}_q} = \{(y,z)\}$$

Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, with

$$\Delta^{\mathcal{I}} = \{a,b\}, \qquad R_1^{\mathcal{I}} = \{(a,b)\} \qquad R_2^{\mathcal{I}} = \{(b,b)\}$$

Then h defined as follows is a homomorphism from $\mathcal{I}_q$ to $\mathcal{I}$:

$$h(x) = a, \qquad h(y) = b, \qquad h(z) = b$$

This shows that $\mathcal{I} \models q$.

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 (i.e., relational structures) is also known as solving a **Constraint Satisfaction Problem** (CSP), a problem well-studied in AI – see also [Kolaitis and Vardi, 1998].

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

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

(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 ([Chandra and Merlin, 1977])

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.

"⇒" 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 $\mathcal{I}_{q_1(\vec{c})} \models q_2(\vec{c})$.

"⇐" Assume that $\mathcal{I}_{q_1(\vec{c})} \models q_2(\vec{c})$.

- By [Chandra and Merlin, 1977] 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 [Chandra and Merlin, 1977] 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 [Chandra and Merlin, 1977] 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 $q_1(\vec{x}) \subseteq q_2(\vec{x})$. $\square$

Query containment for CQs

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

Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$.
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 $(a_1, \ldots, a_k) \in P^{\mathcal{I}}$, $q_{\mathcal{I}}$ contains one atom $P(a_1, \ldots, a_k)$ (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-completenss 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.

Outline of Part 3

Union of conjunctive queries (UCQs)

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

$$\bigvee_{i=1,\ldots,n} \exists \vec{y}_i . conj_i(\vec{x}, \vec{y}_i)$$

where each $\exists \vec{y}_i . conj_i(\vec{x}, \vec{y}_i)$ is a conjunctive query (note that all CQs in a UCQ have the same set of distinguished variables).

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

Datalog notation for UCQs

A union of conjunctive queries

$$q \;=\; \bigvee_{i=1,\ldots,n} \exists \vec{y_i}.\mathit{conj}_i(\vec{x},\vec{y_i})$$

is written in **datalog notation** as

$$\begin{array}{llll} \{ & q(\vec{x}) & \leftarrow & \mathit{conj}'_1(\vec{x},\vec{y_1}') \\ & & \vdots & \\ & q(\vec{x}) & \leftarrow & \mathit{conj}'_n(\vec{x},\vec{y_n}') \quad \} \end{array}$$

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

Note: normally, we omit the set brackets.

Evaluation of UCQs

From the definition of FOL query we have that:

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

if and only if

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

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)

$\{\langle \mathcal{I}, \alpha, q\rangle \mid \mathcal{I}, \alpha \models q\}$ 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 **in** $\textsc{LogSpace}$ (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, the following holds:
$\{q_1, \ldots, q_k\} \subseteq \{q'_1, \ldots, q'_n\}$ iff for each q_i there is a q'_j such that $q_i \subseteq q'_j$.

Proof.
"$\Leftarrow$" Obvious.

"$\Rightarrow$" If the containment holds, then we have
$\{q_1(\vec{c}), \ldots, q_k(\vec{c})\} \subseteq \{q'_1(\vec{c}), \ldots, q'_n(\vec{c})\}$, 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 $\mathcal{I}_{q_i(\vec{c})} \models \{q_1(\vec{c}), \ldots, q_k(\vec{c})\}$.
- By the containment, we have that $\mathcal{I}_{q_i(\vec{c})} \models \{q'_1(\vec{c}), \ldots, q'_n(\vec{c})\}$. I.e., there exists a $q'_j(\vec{c})$ such that $\mathcal{I}_{q_i(\vec{c})} \models q'_j(\vec{c})$.
- Hence, by [Chandra and Merlin, 1977] on containment of CQs, we have that $q_i \subseteq q'_j$. $\square$

Query containment for UCQs – Complexity

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.

Outline of Part 3

Query answering

In ontology-based data access we are interested in a reasoning service that is not typical in ontologies (or in a FOL theory, or in UML class diagrams, or in a knowledge base) but it is very common in databases: **query answering**.

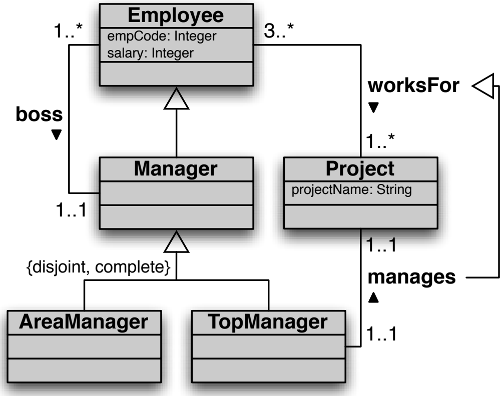
Def.: Query

Is an expression at the intensional level denoting a set of tuples of individuals satisfying a given condition.

Def.: Query Answering

Is the reasoning service that actually computes the answer to a query.

Example of query over an ontology



$$q(ce, cm, sa) \leftarrow \exists e, p, m.$$
$$\text{worksFor}(e,p) \land \text{manages}(m,p) \land \text{boss}(m,e) \land \text{empCode}(e,ce) \land$$
$$\text{empCode}(m,cm) \land \text{salary}(e,sa) \land \text{salary}(m,sa)$$

Query answering under different assumptions

There are two fundamentally different assumptions when addressing query answering:

- **Complete information** on the data, as in traditional databases.
- **Incomplete information** on the data, as in ontologies (aka knowledge bases), but also information integration in databases.

Outline of Part 3

1. Query answering in databases

2. Querying databases and ontologies
   - Query answering in traditional databases
   - Query answering in ontologies
   - Query answering in ontology-based data access

3. Query answering in Description Logics

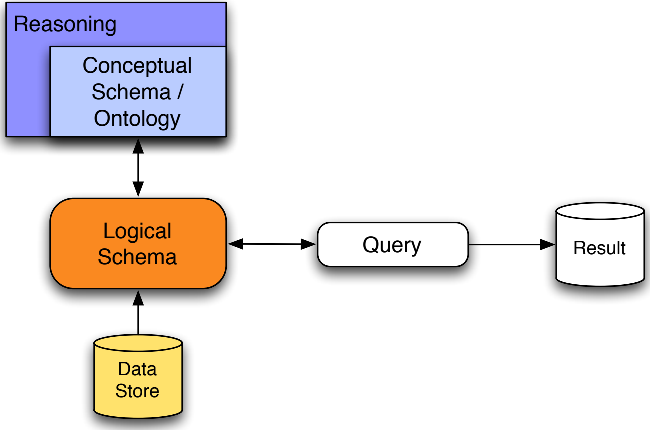
4. References

Query answering in traditional databases

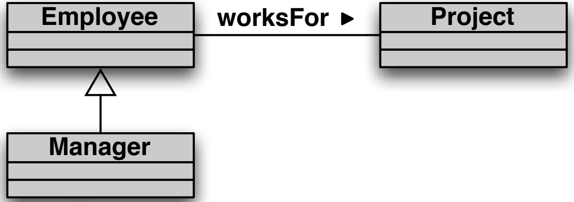
- Data are completely specified (CWA), and typically large.
- Schema/intensional information used in the design phase.
- At **runtime**, the data is assumed to satisfy the schema, and therefore the **schema is not used**.
- Queries allow for complex navigation paths in the data (cf. SQL).

$\rightsquigarrow$ Query answering amounts to **query evaluation**, which is computationally easy.

Query answering in traditional databases (cont'd)



Query answering in traditional databases – Example



For each concept/relationship we have a (complete) table in the DB.

DB: Employee = { john, mary, nick }
Manager = { john, nick }
Project = { prA, prB }
worksFor = { (john,prA), (mary,prB) }

Query: $q(x) \leftarrow \exists p.\, \mathsf{Manager}(x) \land \mathsf{Project}(p) \land \mathsf{worksFor}(x, p)$

Answer: { john }

Outline of Part 3

1. Query answering in databases
2. Querying databases and ontologies
   - Query answering in traditional databases
   - Query answering in ontologies
   - Query answering in ontology-based data access
3. Query answering in Description Logics
4. References

Query answering in ontologies

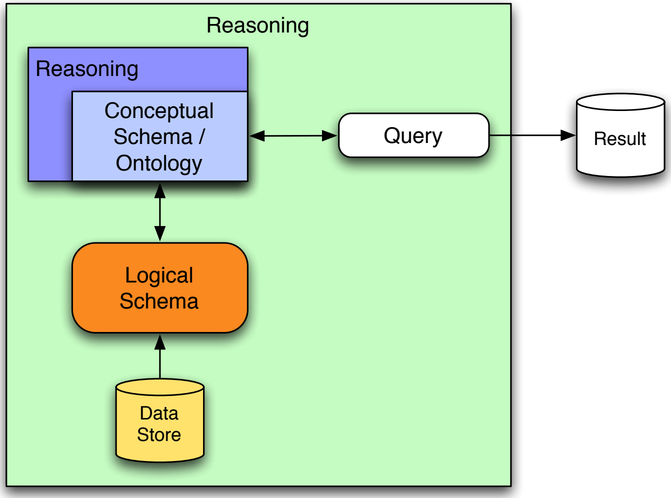
- An ontology (or conceptual schema, or knowledge base) imposes constraints on the data.
- Actual data may be incomplete or inconsistent w.r.t. such constraints.
- The system has to take into account the constraints during query answering, and overcome incompleteness or inconsistency.

$\leadsto$ Query answering amounts to **logical inference**, which is computationally more costly.

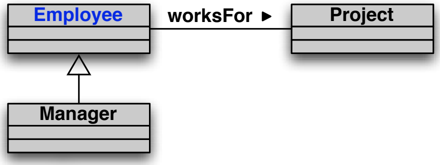
Note:

- The size of the data is not considered critical (comparable to the size of the intensional information).
- Queries are typically simple, i.e., atomic (a class name), and query answering amounts to instance checking.

Query answering in ontologies (cont'd)



Query answering in ontologies – Example



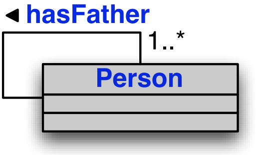
The tables in the database may be **incompletely specified**, or even missing for some classes/properties.

DB: Manager $\supseteq$ { john, nick }
Project $\supseteq$ { prA, prB }
worksFor $\supseteq$ { (john,prA), (mary,prB) }

Query: $q(x) \leftarrow$ Employee(x)

Answer: { john, nick, mary }

Query answering in ontologies – Example 2



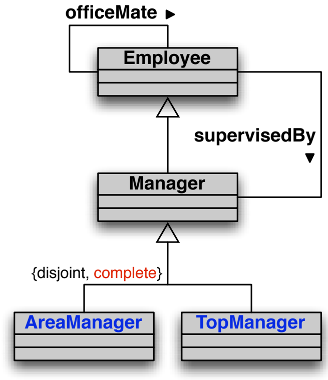
Each person has a father, who is a person.

DB: Person $\supseteq$ { john, nick, toni }
hasFather $\supseteq$ { (john,nick), (nick,toni) }

Queries: $q_1(x, y) \leftarrow$ hasFather(x, y)
$q_2(x) \leftarrow \exists y.$ hasFather(x, y)
$q_3(x) \leftarrow \exists y_1, y_2, y_3.$ hasFather$(x, y_1) \land$ hasFather$(y_1, y_2) \land$ hasFather(y_2, y_3)
$q_4(x, y_3) \leftarrow \exists y_1, y_2.$ hasFather$(x, y_1) \land$ hasFather$(y_1, y_2) \land$ hasFather(y_2, y_3)

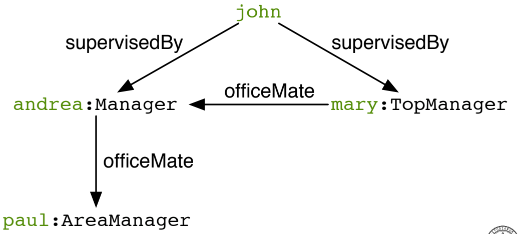
Answers: to q_1: { (john,nick), (nick,toni) }
to q_2: { john, nick, toni }
to q_3: { john, nick, toni }
to q_4: { }

QA in ontologies – Andrea's Example$^{(*)}$



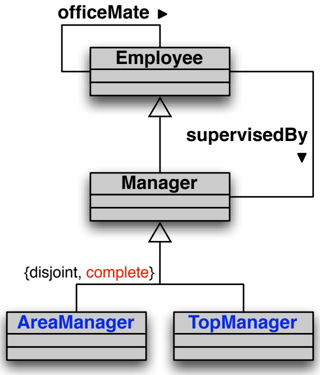
Manager is **partitioned into** AreaManager and TopManager.

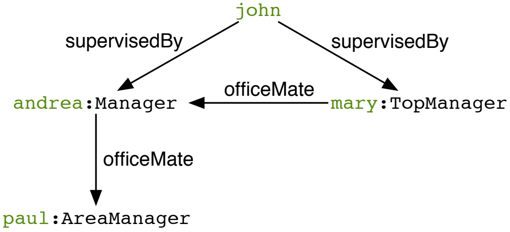
$$
\begin{aligned}
\text{Employee} &\supseteq \{\ \texttt{andrea}, \texttt{paul}, \texttt{mary}, \texttt{john}\ \}\\
\text{Manager} &\supseteq \{\ \texttt{andrea}, \texttt{paul}, \texttt{mary}\ \}\\
\text{AreaManager} &\supseteq \{\ \texttt{paul}\ \}\\
\text{TopManager} &\supseteq \{\ \texttt{mary}\ \}\\
\text{supervisedBy} &\supseteq \{\ (\texttt{john},\texttt{andrea}), (\texttt{john},\texttt{mary})\ \}\\
\text{officeMate} &\supseteq \{\ (\texttt{mary},\texttt{andrea}), (\texttt{andrea},\texttt{paul})\ \}
\end{aligned}
$$



$^{(*)}$ Due to Andrea Schaerf [Schaerf, 1993].

QA in ontologies – Andrea's Example (cont'd)





$$q(x) \leftarrow \exists y, z.\, \mathsf{supervisedBy}(x, y) \land \mathsf{TopManager}(y) \land \mathsf{officeMate}(y, z) \land \mathsf{AreaManager}(z)$$

Answer: { john }

To determine this answer, we need to resort to **reasoning by cases**.

Outline of Part 3

Query answering in ontology-based data access

In OBDA, we have to face the difficulties of both settings:

- The actual **data** is stored in external information sources (i.e., databases), and thus its size is typically **very large**.
- The ontology introduces **incompleteness** of information, and we have to do logical inference, rather than query evaluation.
- We want to take into account at **runtime** the **constraints** expressed in the ontology.
- We want to answer **complex database-like queries**.
- We may have to deal with multiple information sources, and thus face also the problems that are typical of data integration.

Questions that need to be addressed

In the context of ontology-based data access:

1. Which is the "right" **query language**?
2. Which is the "right" **ontology language**?
3. How can we bridge the **semantic mismatch** between the ontology and the data sources?
4. How can **tools for ontology-based data access** take into account these issues?

Which language to use for querying ontologies?

Two borderline cases:

1. Just classes and properties of the ontology $\leadsto$ instance checking
   - Ontology languages are tailored for capturing intensional relationships.
   - They are quite **poor as query languages**:
     Cannot refer to same object via multiple navigation paths in the ontology, i.e., allow only for a limited form of JOIN, namely chaining.
2. Full SQL (or equivalently, first-order logic)
   - Problem: in the presence of incomplete information, query answering becomes **undecidable** (FOL validity).

A good tradeoff is to use (unions of) **conjunctive queries**.

Outline of Part 3

Outline of Part 3

Queries over Description Logics ontologies

Traditionally, simple concept (or role) expressions have been considered as queries over DL ontologies.

We have seen that we need more complex forms of queries, such as those used in databases.

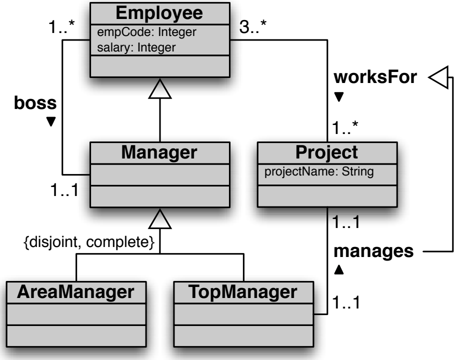
Def.: A **conjunctive query** $q(\vec{x})$ over an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$

is a conjunctive query $\exists \vec{y}.\, conj(\vec{x}, \vec{y})$

- whose **predicate symbols are atomic concept and roles** of $\mathcal{T}$, and
- that may contain constants that are individuals of $\mathcal{A}$.

Remember: a CQ corresponds to a select-project-join SQL query.

Queries over Description Logics ontologies – Example



Conjunctive query over the above ontology:

$$q(x, y) \leftarrow \exists p.\, \mathsf{Employee}(x), \mathsf{Employee}(y), \mathsf{Project}(p), \mathsf{boss}(x, y), \mathsf{worksFor}(x, p), \mathsf{worksFor}(y, p)$$

Outline of Part 3

1. Query answering in databases
2. Querying databases and ontologies
3. Query answering in Description Logics
   - Queries over Description Logics ontologies
   - Certain answers
   - Complexity of query answering
4. References

Certain answers to a query

Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an ontology, $\mathcal{I}$ an interpretation for $\mathcal{O}$, and $q(\vec{x}) = \exists \vec{y}.\, conj(\vec{x}, \vec{y})$ a CQ.

Def.: The **answer** to $q(\vec{x})$ over $\mathcal{I}$, denoted $q^{\mathcal{I}}$

is the set of **tuples $\vec{c}$ of constants of $\mathcal{A}$** such that the formula $\exists \vec{y}.\, conj(\vec{c}, \vec{y})$ evaluates to true in $\mathcal{I}$.

We are interested in finding those answers that hold in all models of an ontology.

Def.: The **certain answers** to $q(\vec{x})$ over $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, denoted $cert(q, \mathcal{O})$

are the **tuples $\vec{c}$ of constants of $\mathcal{A}$** such that $\vec{c} \in q^{\mathcal{I}}$, for every model $\mathcal{I}$ of $\mathcal{O}$.

Query answering in ontologies

Def.: Query answering over an ontology $\mathcal{O}$

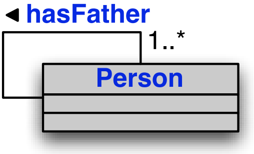
Is the problem of **computing the certain answers** to a query over $\mathcal{O}$.

Computing certain answers is a form of **logical implication**:

$$\vec{c} \in cert(q, \mathcal{O}) \qquad \text{iff} \qquad \mathcal{O} \models q(\vec{c})$$

Note: A special case of query answering is **instance checking**: it amounts to answering the boolean query $q() \leftarrow A(c)$ (resp., $q() \leftarrow P(c_1, c_2)$) over $\mathcal{O}$ (in this case $\vec{c}$ is the empty tuple).

Query answering in ontologies – Example



TBox $\mathcal{T}$:

$$\begin{array}{rcl} \exists \mathsf{hasFather} & \sqsubseteq & \mathsf{Person} \\ \exists \mathsf{hasFather}^- & \sqsubseteq & \mathsf{Person} \\ \mathsf{Person} & \sqsubseteq & \exists \mathsf{hasFather} \end{array}$$

ABox $\mathcal{A}$: Person(john), Person(nick), Person(toni)
hasFather(john,nick), hasFather(nick,toni)

Queries:

$$\begin{array}{l} q_1(x,y) \leftarrow \mathsf{hasFather}(x,y) \\ q_2(x) \leftarrow \exists y.\, \mathsf{hasFather}(x,y) \\ q_3(x) \leftarrow \exists y_1,y_2,y_3.\, \mathsf{hasFather}(x,y_1) \land \mathsf{hasFather}(y_1,y_2) \land \mathsf{hasFather}(y_2,y_3) \\ q_4(x,y_3) \leftarrow \exists y_1,y_2.\, \mathsf{hasFather}(x,y_1) \land \mathsf{hasFather}(y_1,y_2) \land \mathsf{hasFather}(y_2,y_3) \end{array}$$

Certain answers:

$$\begin{array}{l} cert(q_1, \langle \mathcal{T}, \mathcal{A} \rangle) = \{ \text{(john,nick), (nick,toni)} \} \\ cert(q_2, \langle \mathcal{T}, \mathcal{A} \rangle) = \{ \text{john, nick, toni} \} \\ cert(q_3, \langle \mathcal{T}, \mathcal{A} \rangle) = \{ \text{john, nick, toni} \} \\ cert(q_4, \langle \mathcal{T}, \mathcal{A} \rangle) = \{ \} \end{array}$$

Unions of conjunctive queries

We consider also unions of CQs over an ontology.

A **union of conjunctive queries** (UCQ) has the form:

$$\exists \vec{y_1}.\, conj(\vec{x}, \vec{y_1}) \lor \cdots \lor \exists \vec{y_k}.\, conj(\vec{x}, \vec{y_k})$$

where each $\exists \vec{y_i}.\, conj(\vec{x}, \vec{y_i})$ is a CQ.

The (certain) answers to a UCQ are defined analogously to those for CQs.

Example

$$\begin{aligned} q(x) \leftarrow\ & (\mathsf{Manager}(x) \land \mathsf{worksFor}(x, \mathtt{tones})) \ \lor \\ & (\exists y.\, \mathsf{boss}(x, y) \land \mathsf{worksFor}(y, \mathtt{tones})) \end{aligned}$$

In datalog notation:

$$\begin{array}{lcl} q(x) & \leftarrow & \mathsf{Manager}(x), \mathsf{worksFor}(x, \mathtt{tones}) \\ q(x) & \leftarrow & \exists y.\, \mathsf{boss}(x, y) \land \mathsf{worksFor}(y, \mathtt{tones}) \end{array}$$

Outline of Part 3

Complexity measures for queries over ontologies

When measuring the complexity of answering a query $q(\vec{x})$ over an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, various parameters are of importance.

Depending on which parameters we consider, we get different complexity measures:

- **Data complexity**: only the size of the ABox (i.e., the data) matters.
  TBox and query are considered fixed.
- **Query complexity**: only the size of the query matters.
  TBox and ABox are considered fixed.
- **Schema complexity**: only the size of the TBox (i.e., the schema) matters.
  ABox and query are considered fixed.
- **Combined complexity**: no parameter is considered fixed.

In the OBDA setting, **the size of the data largely dominates** the size of the conceptual layer (and of the query).

$\leadsto$ **Data complexity** is the relevant complexity measure.

Data complexity of query answering

When studying the complexity of query answering, we need to consider the associated decision problem:

Def.: Recognition problem for query answering

Given an ontology $\mathcal{O}$, a query q over $\mathcal{O}$, and a tuple $\vec{c}$ of constants, **check whether $\vec{c} \in cert(q, \mathcal{O})$**.

We look mainly at the **data complexity** of query answering, i.e., complexity of the recognition problem computed **w.r.t. the size of the ABox only**.

Complexity of query answering in DLs

Studied extensively for (unions of) CQs and various ontology languages:

	Combined complexity	Data complexity
Plain databases	NP-complete	in AC^0 (1)
$\mathcal{ALCI}$, $\mathcal{SH}$, $\mathcal{SHIQ}$, ...	2ExpTime-complete (3)	coNP-complete (2)
OWL 2 (and less)	3ExpTime-hard	coNP-hard

(1) This is what we need to scale with the data.

(2) coNP-hard already for a TBox with a single disjunction [Donini *et al.*, 1994; Calvanese *et al.*, 2006].
In coNP for very expressive DLs [Levy and Rousset, 1998; Ortiz *et al.*, 2006; Glimm *et al.*, 2007].

(3) [Calvanese *et al.*, 1998; Calvanese *et al.*, 2008; Lutz, 2007]

Questions

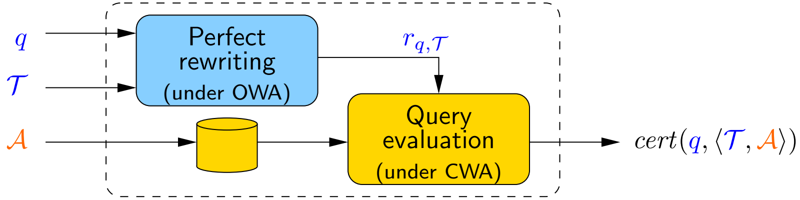
- Can we find interesting (description) logics for which query answering can be done efficiently (i.e., in AC^0)?
- If yes, can we leverage relational database technology for query answering?

Inference in query answering

To be able to deal with data efficiently, we need to separate the contribution of $\mathcal{A}$ from the contribution of q and $\mathcal{T}$.

$\rightsquigarrow$ Query answering by **query rewriting**.

Query rewriting



Query answering can **always** be thought as done in two phases:

1. **Perfect rewriting**: produce from q and the TBox $\mathcal{T}$ a new query $r_{q,\mathcal{T}}$ (called the perfect rewriting of q w.r.t. $\mathcal{T}$).
2. **Query evaluation**: evaluate $r_{q,\mathcal{T}}$ over the ABox $\mathcal{A}$ seen as a complete database (and without considering the TBox $\mathcal{T}$).
   $\rightsquigarrow$ Produces $cert(q, \langle \mathcal{T}, \mathcal{A} \rangle)$.

Note: The "always" holds if we pose no restriction on the language in which to express the rewriting $r_{q,\mathcal{T}}$.

$\mathcal{Q}$-rewritability

Let $\mathcal{Q}$ be a query language and $\mathcal{L}$ an ontology language.

Def.: $\mathcal{Q}$-rewritability

For an ontology language $\mathcal{L}$, query answering is **$\mathcal{Q}$-rewritable** if for every TBox $\mathcal{T}$ of $\mathcal{L}$ and for every query q, the perfect reformulation $r_{q,\mathcal{T}}$ of q w.r.t. $\mathcal{T}$ can be expressed in the query language $\mathcal{Q}$.

Notice that the complexity of computing $r_{q,\mathcal{T}}$ or the size of $r_{q,\mathcal{T}}$ do **not** affect data complexity.

Hence, $\mathcal{Q}$-rewritability is tightly related to **data complexity**, i.e.:

- complexity of computing $cert(q, \langle\mathcal{T}, \mathcal{A}\rangle)$ measured in the size of the ABox $\mathcal{A}$ only,
- which corresponds to the **complexity of evaluating $r_{q,\mathcal{T}}$ over $\mathcal{A}$**.

Language of the rewriting

The **expressiveness of the ontology language affects the rewriting language**, i.e., the language into which we are able to rewrite UCQs:

- When we can rewrite into FOL/SQL (i.e., the ontology language enjoys FOL-rewritability).
  $\leadsto$ Query evaluation can be done in SQL, i.e., via an RDBMS
  (*Note:* FOL is in AC^0).
- When we can rewrite into an NLogSpace-hard language.
  $\leadsto$ Query evaluation requires (at least) linear recursion.
- When we can rewrite into a PTime-hard language.
  $\leadsto$ Query evaluation requires full recursion (e.g., Datalog).
- When we can rewrite into a coNP-hard language.
  $\leadsto$ Query evaluation requires (at least) power of Disjunctive Datalog.

References I

[Calvanese *et al.*, 1998] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini.
On the decidability of query containment under constraints.
In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 149–158, 1998.

[Calvanese *et al.*, 2006] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati.
Data complexity of query answering in description logics.
In *Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 260–270, 2006.

[Calvanese *et al.*, 2008] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini.
Conjunctive query containment and answering under description logics constraints.
ACM Trans. on Computational Logic, 9(3):22.1–22.31, 2008.

[Chandra and Merlin, 1977] Ashok K. Chandra and Philip 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.

References II

[Donini *et al.*, 1994] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf.
Deduction in concept languages: From subsumption to instance checking.
J. of Logic and Computation, 4(4):423–452, 1994.

[Glimm *et al.*, 2007] Birte Glimm, Ian Horrocks, Carsten Lutz, and Ulrike Sattler.
Conjunctive query answering for the description logic $\mathcal{SHIQ}$.
In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, pages 399–404, 2007.

[Kolaitis and Vardi, 1998] Phokion G. Kolaitis and Moshe 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.

[Levy and Rousset, 1998] Alon Y. Levy and Marie-Christine Rousset.
Combining Horn rules and description logics in CARIN.
Artificial Intelligence, 104(1–2):165–209, 1998.

References III

[Lutz, 2007] Carsten Lutz.
Inverse roles make conjunctive queries hard.
In *Proc. of the 20th Int. Workshop on Description Logic (DL 2007)*, volume 250 of *CEUR Electronic Workshop Proceedings*, `http://ceur-ws.org/`, pages 100–111, 2007.

[Ortiz *et al.*, 2006] Maria Magdalena Ortiz, Diego Calvanese, and Thomas Eiter.
Characterizing data complexity for conjunctive query answering in expressive description logics.
In *Proc. of the 21st Nat. Conf. on Artificial Intelligence (AAAI 2006)*, pages 275–280, 2006.

[Schaerf, 1993] Andrea Schaerf.
On the complexity of the instance checking problem in concept languages with existential quantification.
J. of Intelligent Information Systems, 2:265–278, 1993.

[Vardi, 1982] Moshe 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.