# Ontologies and Databases

Diego Calvanese

KRDB Research Centre
Free University of Bozen-Bolzano


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

Reasoning Web Summer School 2009
September 3–4, 2009
Bressanone, Italy

# Overview of the Tutorial

1. Introduction to ontology-based data access
   1. Introduction to ontologies
   2. Ontology languages
   3. Query answering in databases
   4. Querying databases and ontologies
2. Description Logics and the *DL-Lite* family
   5. Quick overview of DLs
   6. Reasoning and query answering in DLs
   7. The *DL-Lite* family of tractable DLs
3. Reasoning in the *DL-Lite* family
   8. TBox reasoning
   9. TBox & ABox reasoning
   10. Beyond *DL-Lite*
4. Linking ontologies to relational data
   11. The impedance mismatch problem
   12. Ontology-Based Data Access systems
   13. Query answering in OBDA systems
   14. The QuOnto OBDA system
5. Conclusions and references

# Part I

# Introduction to ontology-based data access

# Outline of Part 1

# Outline of Part 1

# Outline of Part 1

# New challenges in data management

One of the key challenges in complex systems today is the management of data:

- The **amount** of data has increased enormously.
- The **complexity** of the data has increased:
  structured $\rightsquigarrow$ semi-structured $\rightsquigarrow$ unstructured
- The data may be of **low quality**, e.g., incomplete, inconsistent, not *crisp*.
- Data is increasingly **distributed** and **heterogeneous**, but nevertheless needs to be accessed in a uniform way.
- Data needs to be consumed not only by humans, but also by machines.

Traditional database management systems are not sufficient anymore to fulfill today's data management requirements.

# Addressing data management challenges

Several efforts come from the database area:

- New kinds of databases are studied, to manage semi-structured (XML), and probabilistic data.
- Data integration is one of the major challenges for the future or IT.
  E.g., the market for data integration software is estimated to grow from $2.5 billion in 2007 to $3.8 billion in 2012 (+8.7% per year) [IDC. Worldwide Data Integration and Access Software 2008-2012 Forecast. Doc No. 211636 (2008)].

On the other hand, management of complex kinds of information has traditionally been the concern of **Knowledge Representation** in AI:

- Research in AI and KR can bring new insights, solutions, techniques, and technologies.
- **However, what has been done in KR needs to be adapted / extended / tuned to address the new challenges coming from today's data management.**

# Ontologies

One important area of KR deals with the study of formalisms for the structured representation of information:

- By grounding the used formalisms in logic, the information is provided with a **formal semantics** (i.e., a meaning).
- The logic-based formalization allows one to provide automated support for tasks related to data management, by means of **logic-based inference**.
- Computational aspects are of concern, so that the automated reasoning support can be done effectively, and built into tools.

Several proposals and studies in the last 30 years of formalisms, techniques, algorithms, and tools:

$$\text{Description Logics} \rightsquigarrow \textbf{Ontologies}$$

In this tutorial we are looking into using ontologies for managing data.

# Levels of an ontology

**Def.: Ontology**

is a representation scheme that describes a **formal conceptualization** of a domain of interest.

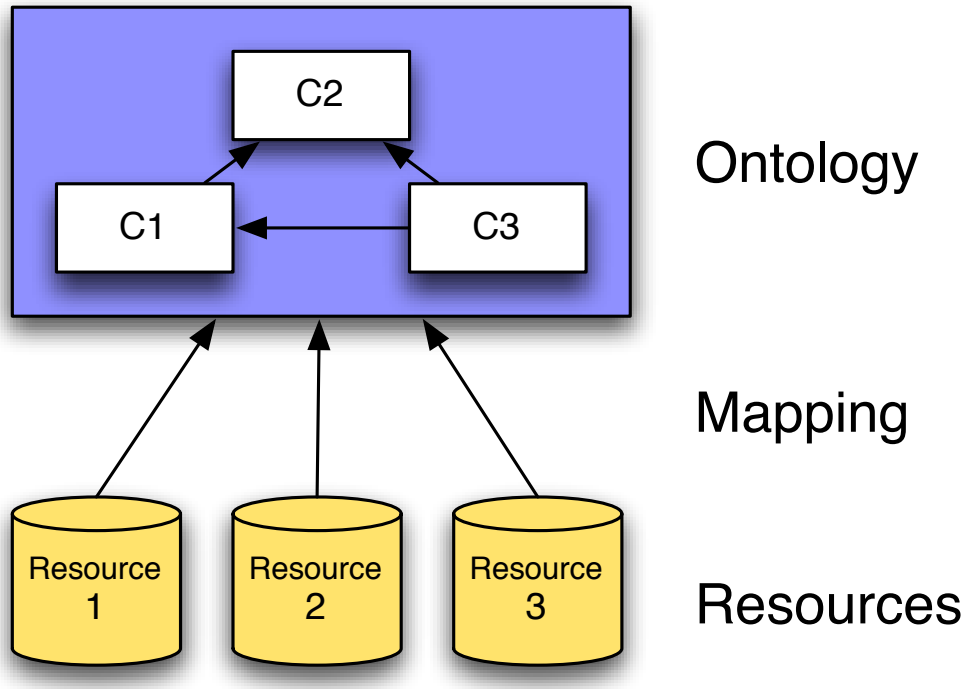
The specification of an ontology usually comprises two distinct levels:

- **Intensional level**: specifies a set of conceptual elements and of rules to describe the conceptual structures of the domain.
- **Extensional level**: specifies a set of instances of the conceptual elements described at the intensional level.

*Note:* an ontology may specify also a **meta-level**, which specifies a set of modeling categories of which the conceptual elements are instances.

# Outline of Part 1

# Ontologies at the core of information systems




The usage of all system resources (data and services) is done through the domain conceptualization.

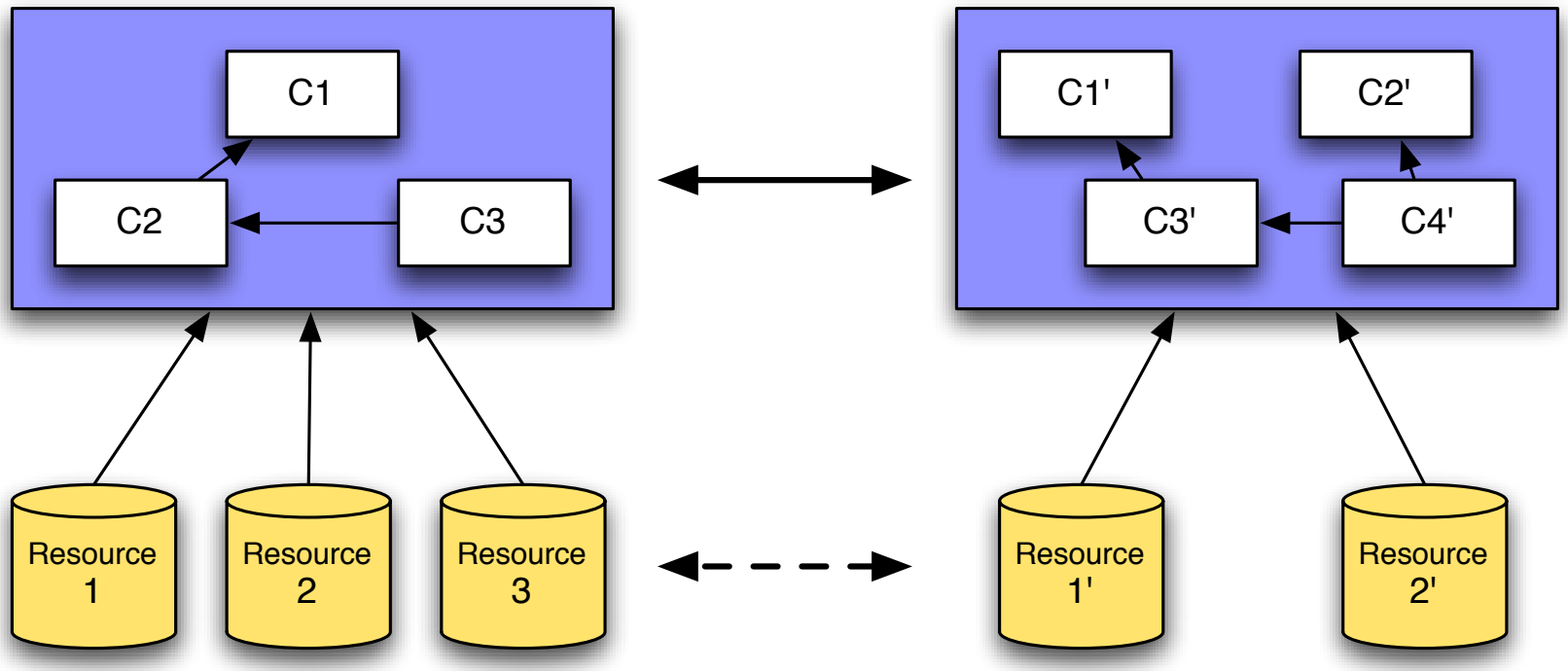
# Ontology mediated access to data

Desiderata: achieve **logical transparency** in access to data:

- **Hide** to the user where and how data are stored.
- Present to the user a **conceptual view** of the data.
- Use a **semantically rich formalism** for the conceptual view.

*This setting is similar to the one of Data Integration. The difference is that here the ontology provides a rich conceptual description of the data managed by the system.*

# Ontologies at the core of cooperation




The cooperation between systems is done at the level of the conceptualization.

# Outline of Part 1

# Three novel challenges

1. **Languages**
2. Methodologies
3. **Tools**

. . . for specifying, building, and managing ontologies to be used in information systems.

# Challenge 1: Ontology languages

- Several proposals for ontology languages have been made.
- **Tradeoff** between **expressive power** of the language and **computational complexity** of dealing with (i.e., performing inference over) ontologies specified in that language.
- Usability needs to be addressed.

In this tutorial:

- We discuss variants of ontology languages suited for managing **ontologies in information systems**.
- We discuss in depth the above mentioned **tradeoff** ...
- ... paying particular attention to the aspects related to data management.

# Challenge 2: Methodologies

- Developing and dealing with ontologies is a complex and challenging task.
- Developing **good ontologies** is even more challenging.
- It requires to master the technologies based on semantics, which in turn requires good knowledge about the languages, their semantics, and the implications it has w.r.t. reasoning over the ontology.

In this tutorial:

- We study in depth the **semantics of ontologies**, with an emphasis on their relationship to data in information sources.
- We thus lay the **foundations for the development of methodologies**, though we do not present specific ontology-development methodologies here.

# Challenge 3: Tools

- According to the principle that "there is no meaning without a language with a formal semantics", the formal semantics becomes the solid basis for dealing with ontologies.
- Hence every kind of access to an ontology (to extract information, to modify it, etc.), requires to **fully** take into account its semantics.
- We need to resort to tools that provide capabilities to perform **automated reasoning** over the ontology, and the kind of reasoning should be **sound** and **complete** w.r.t. the formal semantics.

In this tutorial:

- We discuss the requirements for such ontology management tools.
- We discuss the technologies behind a tool that has been specifically designed for optimized access to information sources through ontologies.

# A challenge across the three challenges: Scalability

When we want to use ontologies to access information sources, we have to address the three challenges of languages, methodologies, and tools by taking into account **scalability** w.r.t.:

- the size of (the intensional level of) the ontology
- the number of ontologies
- the **size of the information sources** that are accessed through the ontology/ontologies.

In this tutorial we pay particular attention to the third aspect, since we work under the realistic assumption that the extensional level (i.e., the data) largely dominates in size the intensional level of an ontology.

# Outline of Part 1

# Outline of Part 1

# Elements of an ontology language

- **Syntax**
  - Alphabet
  - Languages constructs
  - Sentences to assert knowledge
- **Semantics**
  - Formal meaning
- **Pragmatics**
  - Intended meaning
  - Usage

# Static vs. dynamic aspects

The aspects of the domain of interest that can be modeled by an ontology language can be classified into:

- **Static aspects**
  - Are related to the structuring of the domain of interest.
  - Supported by virtually all languages.
- Dynamic aspects
  - Are related to how the elements of the domain of interest evolve over time.
  - Supported only by some languages, and only partially (cf. services).

Before delving into the dynamic aspects, we need a good understanding of the static ones.

In this course we concentrate essentially on the static aspects.

# Outline of Part 1

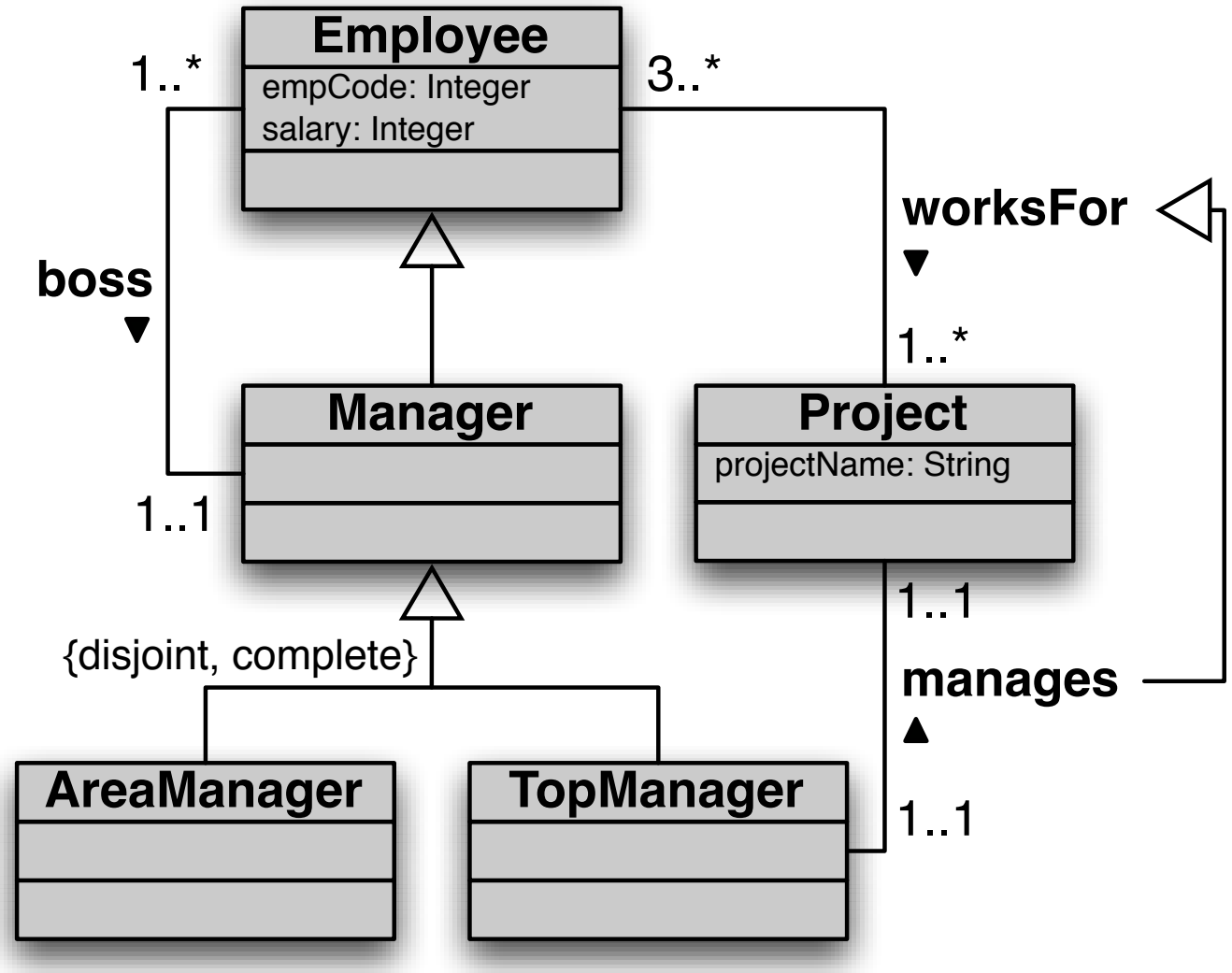
# Intensional level of an ontology language

An ontology language for expressing the intensional level usually includes:

- Concepts
- Properties of concepts
- Relationships between concepts, and their properties
- Axioms
- Queries

Ontologies are typically **rendered as diagrams** (e.g., Semantic Networks, Entity-Relationship schemas, UML Class Diagrams).

# Example: ontology rendered as UML Class Diagram



# Concepts

**Def.: Concept**

Is an element of an ontology that denotes a collection of instances (e.g., the set of "employees").

We distinguish between:

- Intensional definition:
  specification of name, properties, relations, ...
- Extensional definition:
  specification of the instances

Concepts are also called **classes**, **entity types**, **frames**.

# Properties

**Def.: Property**

Is an element of an ontology that qualifies another element (e.g., a concept or a relationship).

Property definition (intensional and extensional):

- Name
- Type: may be either
  - atomic (integer, real, string, enumerated, ...), or
    e.g., eye-color $\longrightarrow$ { blu, brown, green, grey }
  - structured (date, set, list, ...)
    e.g., date $\longrightarrow$ day/month/year
- The definition may also specify a default value.

Properties are also called **attributes**, **features**, **slots**, **data properties**.

# Relationships

**Def.: Relationship**

Is an element of an ontology that expresses an association among concepts.

We distinguish between:

- Intensional definition:
  specification of involved concepts
  e.g., worksFor is defined on Employee and Project
- Extensional definition:
  specification of the instances of the relationship, called facts
  e.g., worksFor(domenico, tones)

Relationships are also called **associations**, **relationship types**, **roles**, **object properties**.

# Axioms

**Def.: Axiom**

Is a logical formula that expresses at the intensional level a condition that must be satisified by the elements at the extensional level.

Different kinds of axioms/conditions:

- subclass relationships, e.g., $\text{Manager} \sqsubseteq \text{Employee}$
- equivalences, e.g., $\text{Manager} \equiv \text{AreaManager} \sqcup \text{TopManager}$
- disjointness, e.g., $\text{AreaManager} \sqcap \text{TopManager} \equiv \bot$
- (cardinality) restrictions,
  e.g., each Employee worksFor at least 3 Project
- ...

Axioms are also called **assertions**.
A special kind of axioms are **definitions**.

# Extensional level of an ontology language

At the extensional level we have individuals and facts:

- An **instance** represents an individual (or object) in the extension of a concept.
  e.g., domenico is an instance of Employee
- A **fact** represents a relationship holding between instances.
  e.g., worksFor(domenico, tones)

# Outline of Part 1

# Comparison with other formalisms

- Ontology languages vs. knowledge representation languages:

  Ontologies **are** knowledge representation schemas.

- Ontology vs. logic:

  Logic is **the** tool for assigning semantics to ontology languages.

- Ontology languages vs. conceptual data models:

  Conceptual schemas **are** special ontologies, suited for conceptualizing a single logical model (database).

- Ontology languages vs. programming languages:

  Class definitions **are** special ontologies, suited for conceptualizing a single structure for computation.

# Classification of ontology languages

- Graph-based
  - Semantic networks
  - Conceptual graphs
  - UML class diagrams, Entity-Relationship schemas
- Frame based
  - Frame Systems
  - OKBC, XOL
- Logic based
  - **Description Logics** (e.g., $\mathcal{SHOIQ}$, $\mathcal{DLR}$, *DL-Lite*, OWL, ...)
  - Rules (e.g., RuleML, LP/Prolog, F-Logic)
  - First Order Logic (e.g., KIF)
  - Non-classical logics (e.g., non-monotonic, probabilistic)

# Outline of Part 1

# Outline of Part 1

# FOL queries

We assume we are given a relational alphabet $\Sigma$, i.e., a set of relation symbols, each with an associated arity.

**Def.: A FOL query $\varphi(x_1, \ldots, x_k)$ over $\Sigma$ (of arity $k$)**

... is a FOL formula over $\Sigma$ with free variables $x_1, \ldots, x_k$.

Such a query is evaluated w.r.t. a FOL interpretation $\mathcal{I}$ and an assignment $\alpha$ of elements of the domain of $\mathcal{I}$ to $x_1, \ldots, x_k$, i.e., we ask whether:

$$\mathcal{I}, \alpha \models \varphi$$

Given a query $\varphi(x_1, \ldots, x_k)$, we denote with $\langle a_1, \ldots, a_k \rangle$ the assignment that assigns $a_i$ to $x_i$, for $i \in \{1, \ldots, k\}$.

Note:

- The interpretation $\mathcal{I}$ corresponds to the database over which the query is evaluated.
- The assignments $\alpha$ to the free variables of $\varphi$ such that $\mathcal{I}, \alpha \models \varphi$ are the **answer to $\varphi$ over $\mathcal{I}$**, denoted $\varphi^{\mathcal{I}}$.

# FOL boolean queries

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

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

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

Such an answer is

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

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

# Query evaluation

Let us consider:

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

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

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

# Query evaluation problem

## Definitions

- **Query answering problem**: given a finite interpretation $\mathcal{I}$ and a FOL query $\varphi(x_1, \ldots, x_k)$, 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 – Complexity measures [Vardi, 1982]

## Def.: 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.

## Def.: Data complexity

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

## Def.: 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).

# Outline of Part 1

# (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

**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:

$\mathsf{Person}(\mathsf{name}, \mathsf{age})$, $\mathsf{Lives}(\mathsf{person}, \mathsf{city})$, $\mathsf{Manages}(\mathsf{boss}, \mathsf{employee})$

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)

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

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

# 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 truth 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 $\texttt{Truth}(\mathcal{I}, \alpha, \varphi)$ is defined inductively as follows.

# Nondeterministic CQ evaluation algorithm

```
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** 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 $\mathrm{NP}$-complete.

We exploit 3-colorability to show $\mathrm{NP}$-hardness of conjunctive query evaluation.

# Reduction from 3-colorability to CQ evaluation

Let $G = (V, E)$ be an undirected graph. 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 $V = \{x_1, \ldots, x_n\}$, then consider the boolean conjunctive query defined as:

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

**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.

# Outline of Part 1

# 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 $h : \Delta^{\mathcal{I}} \rightarrow \Delta^{\mathcal{J}}$ 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 $\exists x_1, \ldots, x_n.conj$

**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

- $\Delta^{\mathcal{I}_q} = \{x_1, \ldots, x_n\} \cup \{c \mid c \text{ constant occurring in } q\}$, i.e., all the variables and constants in $q$;
- $c^{\mathcal{I}_q} = c$, for each constant $c$ in $q$;
- $(t_1, \ldots, t_k) \in P^{\mathcal{I}_q}$ 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 $\mathcal{I}_q$ 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 $\alpha$ be an assignment to the existential variables that makes $q$ true in $\mathcal{I}$, and let $\hat{\alpha}$ be its extension to constants. Then $\hat{\alpha}$ is a homomorphism from $\mathcal{I}_q$ to $\mathcal{I}$.

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

# Canonical interpretation and (boolean) CQ evaluation

The previous result can be rephrased as follows:

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

Finding a homomorphism between two interpretations (aka relational structures) is also known as solving a **Constraint Satisfaction Problem** (CSP), a problem well-studied in AI – see also [Kolaitis and Vardi, 1998].

# Outline of Part 1

# 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 set of distinguished variables.

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

# Datalog notation for UCQs

A union of conjunctive queries

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

is written in **datalog notation** as

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

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

*Note:* in general, we omit the set brackets.

# Evaluation of UCQs

From the definition of FOL query we have that:

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

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** 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

# Outline of Part 1

# 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**.

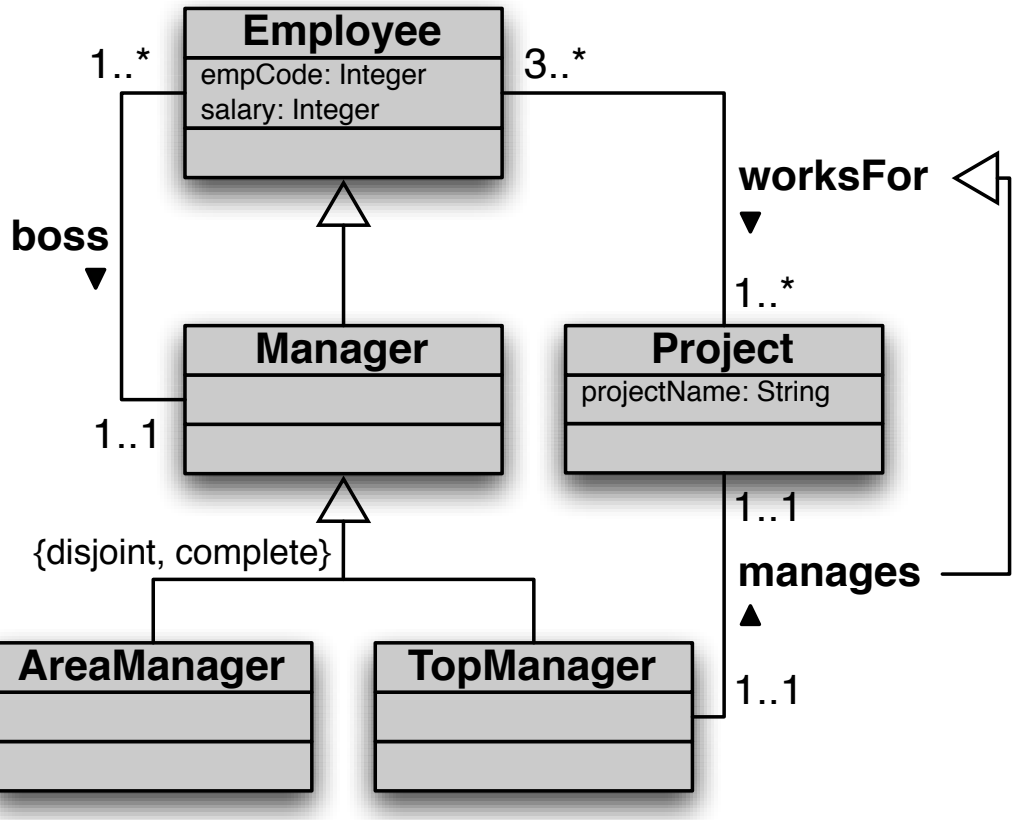
**Def.: Query**

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




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

# 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.

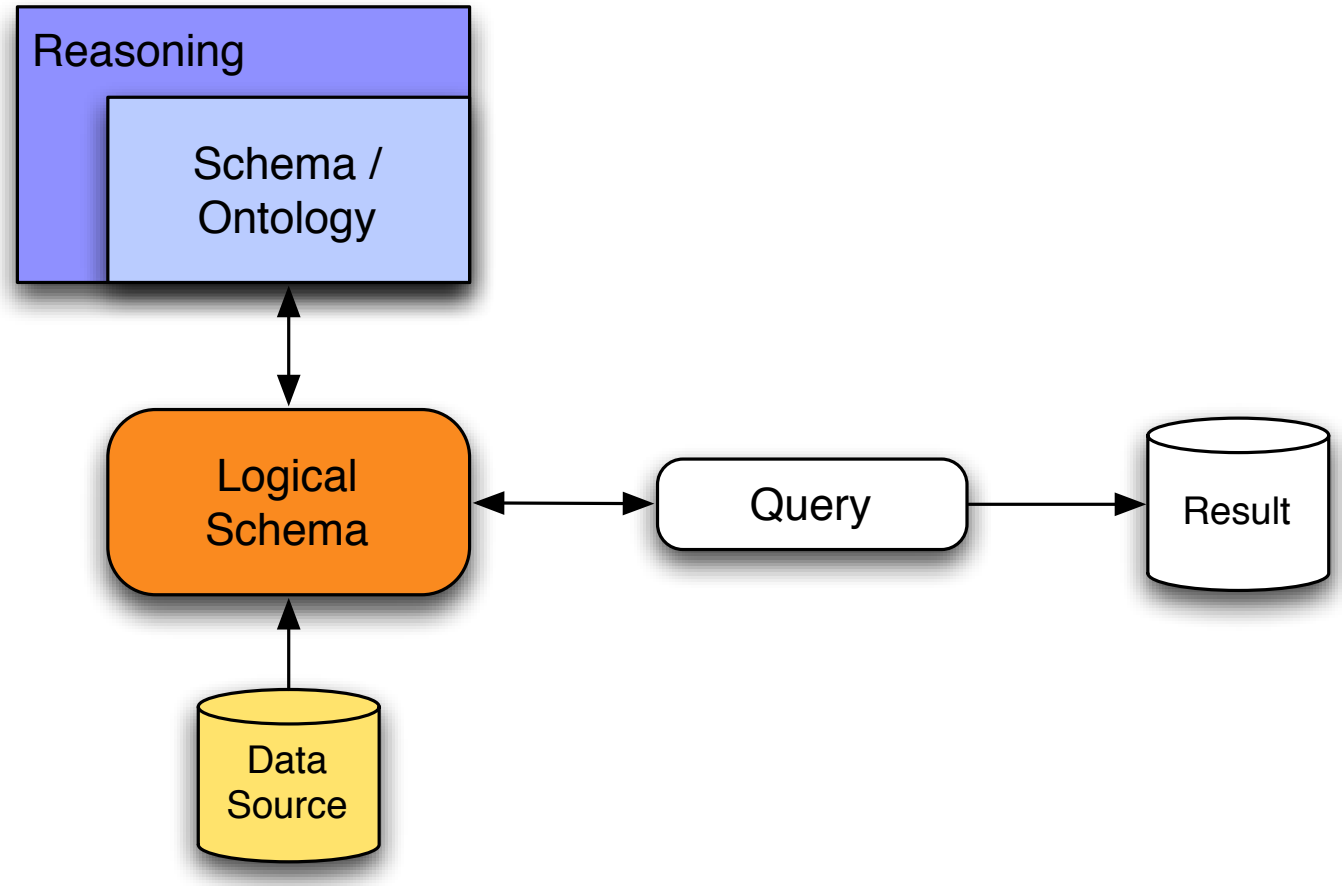
# Outline of Part 1

# Query answering in traditional databases

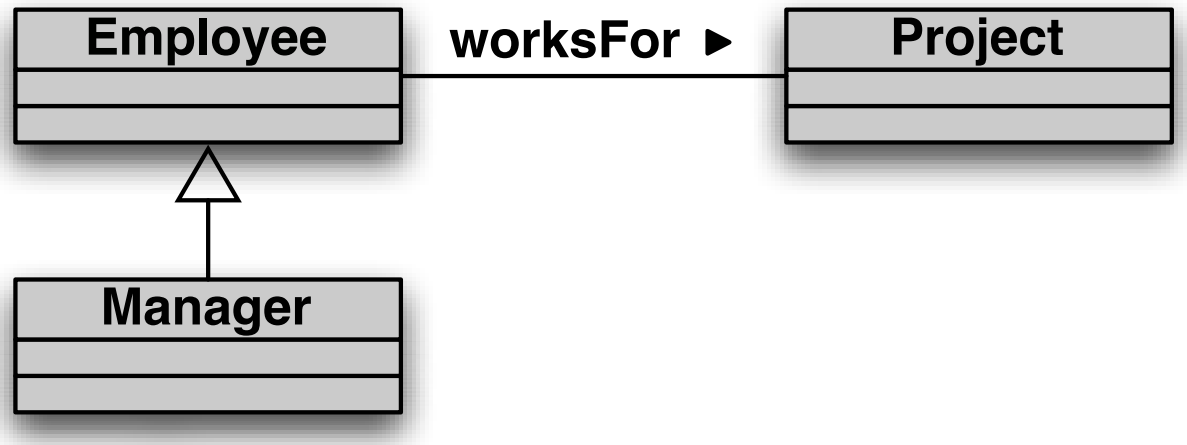
- Data are completely specified (CWA), and typically large.
- Schema/intensional information used in the design phase.
- At **runtime**, the data is assumed to satisfy the schema, and therefore the **schema is not used**.
- Queries allow for complex navigation paths in the data (cf. SQL).

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

**Answer:** { john }

# Outline of Part 1

# Query answering in ontologies

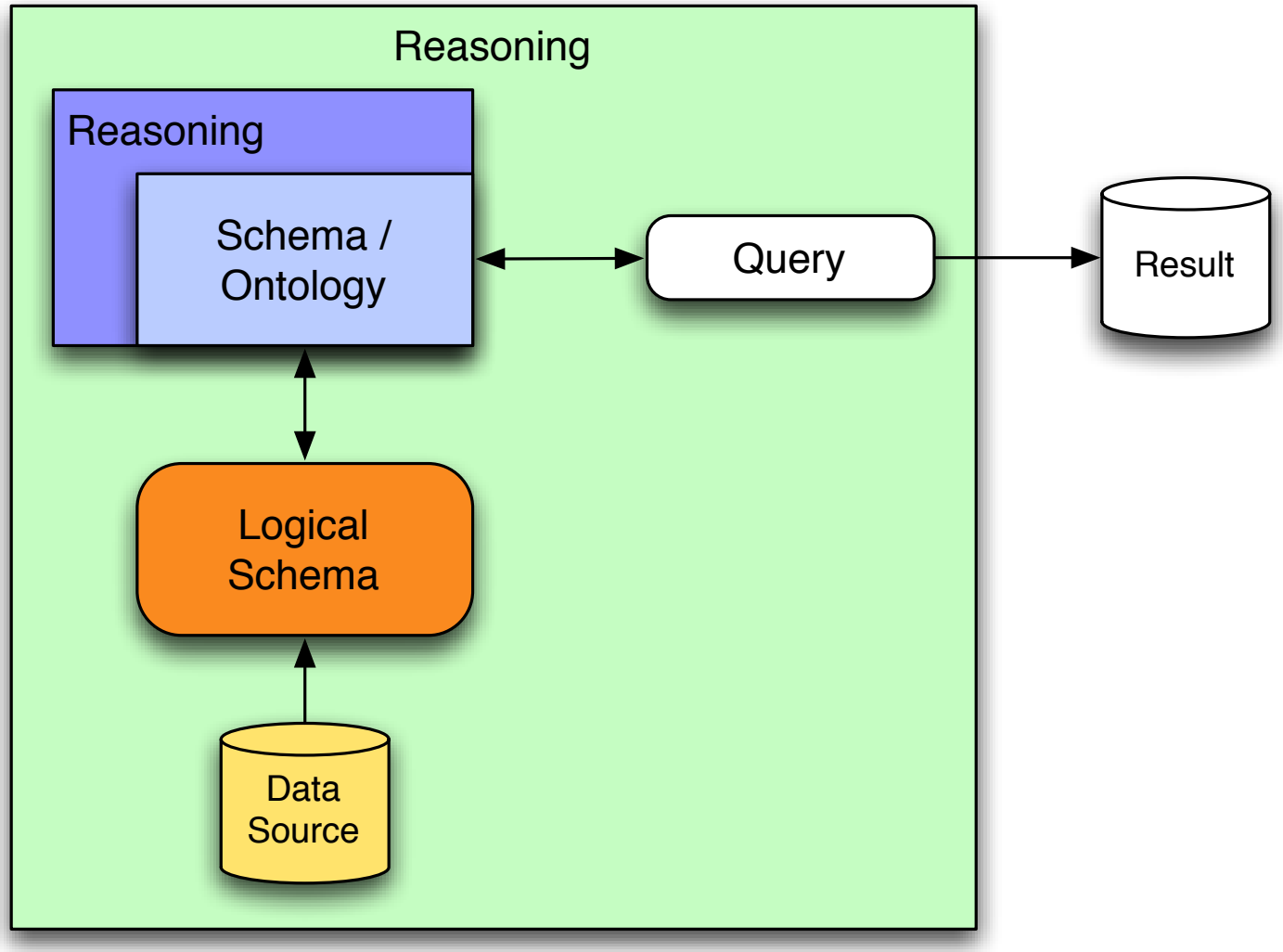
- An ontology (or conceptual schema, or knowledge base) imposes constraints on the data.
- Actual data may be incomplete or inconsistent w.r.t. such constraints.
- The system has to take into account the constraints during query answering, and overcome incompleteness or inconsistency.

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

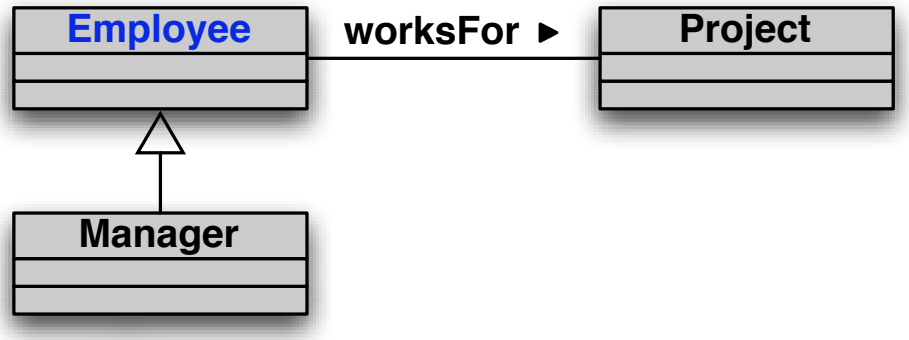
Note:

- 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



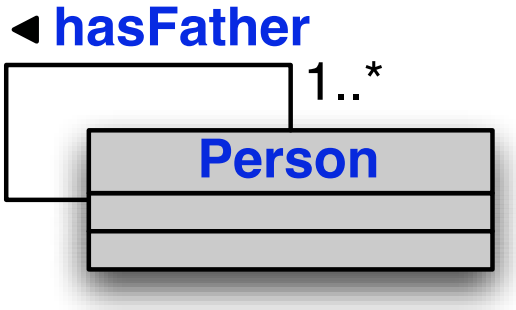

The tables in the database may be **incompletely specified**, or even missing for some classes/properties.

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

Query: $q(x) \leftarrow \text{Employee}(x)$

Answer: { `john`, `nick`, `mary` }

# Query answering in ontologies – Example 2



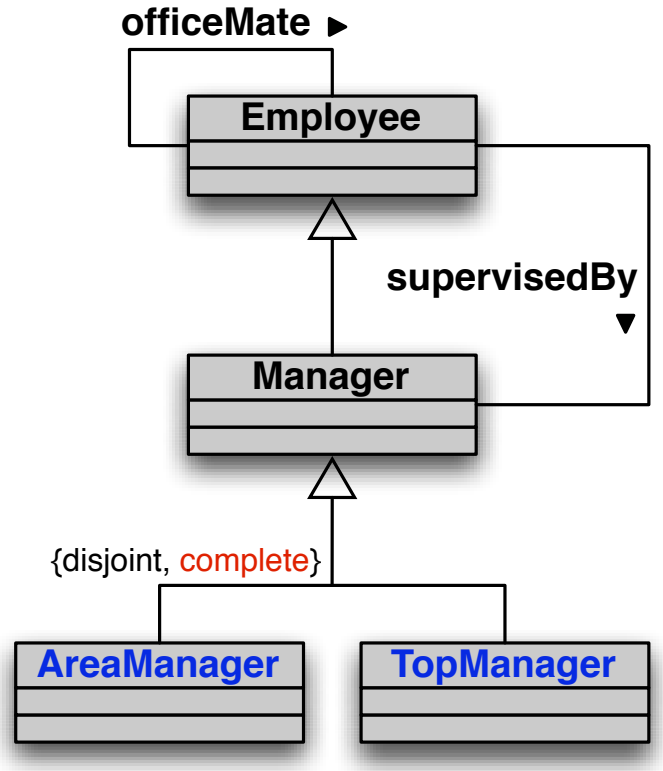

Each person has a father, who is a person.

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

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

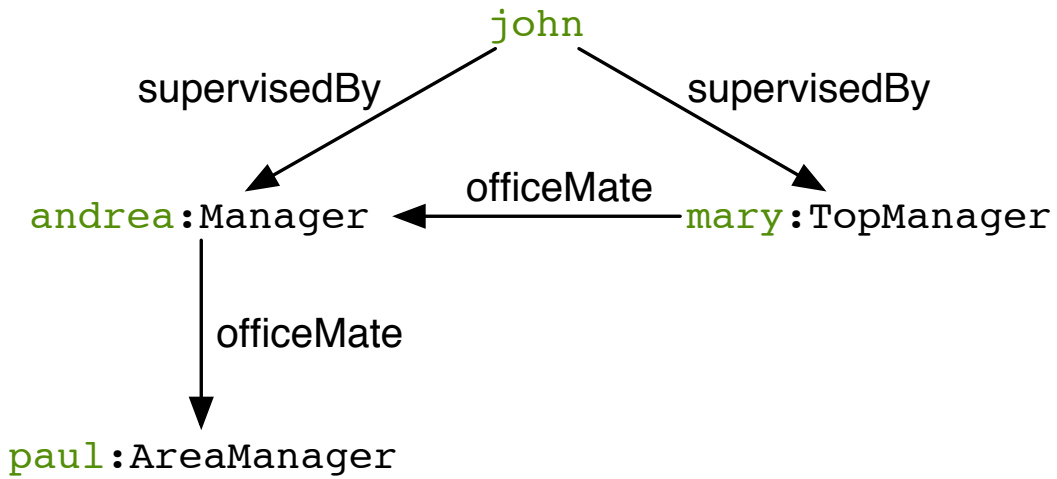
Answers: to $q_1$: { (john,nick), (nick,toni) }
to $q_2$: { john, nick, toni }
to $q_3$: { john, nick, toni }
to $q_4$: { }

# QA in ontologies – Andrea's Example(*)



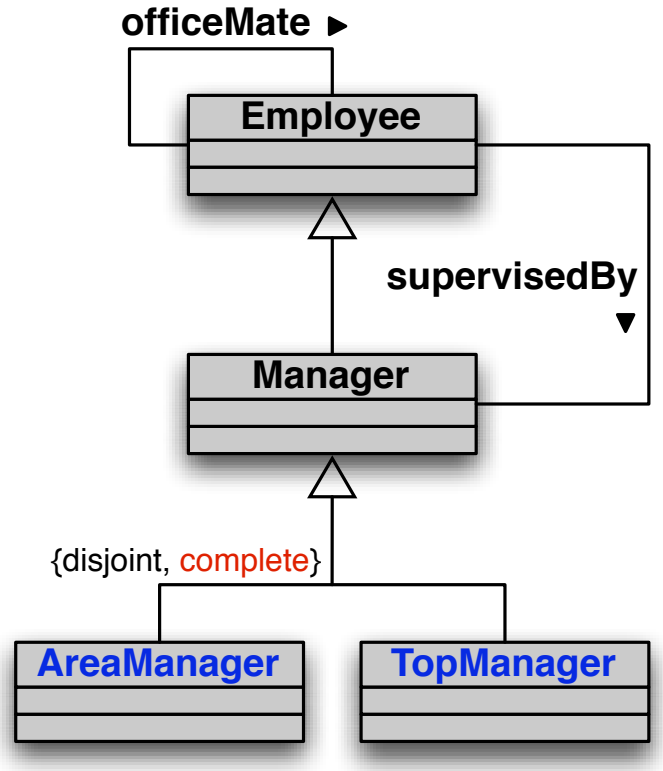

Manager is **partitioned into** AreaManager and TopManager.

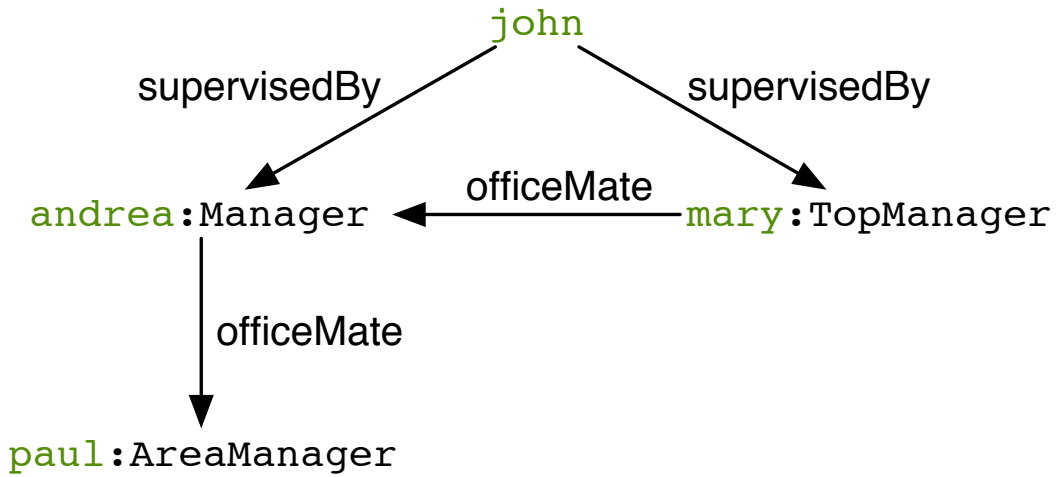
$$\begin{aligned}
\text{Employee} &\supseteq \{\ \texttt{andrea}, \texttt{nick}, \texttt{mary}, \texttt{john}\ \} \\
\text{Manager} &\supseteq \{\ \texttt{andrea}, \texttt{nick}, \texttt{mary}\ \} \\
\text{AreaManager} &\supseteq \{\ \texttt{nick}\ \} \\
\text{TopManager} &\supseteq \{\ \texttt{mary}\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




(*) Due to Andrea Schaerf [Schaerf, 1993].

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




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

Answer: { john }

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

# Outline of Part 1

# 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 $\rightsquigarrow$ 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**.

# Part II

# Description Logics and the *DL-Lite* family

# Outline of Part 2

# Outline of Part 2

# Outline of Part 2

# What are Description Logics?

Description Logics (DLs) [Baader *et al.*, 2003] are **logics** specifically designed to represent and reason on structured knowledge.

The domain of interest is composed of objects and is structured into:

- concepts, which correspond to classes, and denote sets of objects
- roles, which correspond to (binary) relationships, and denote binary relations on objects

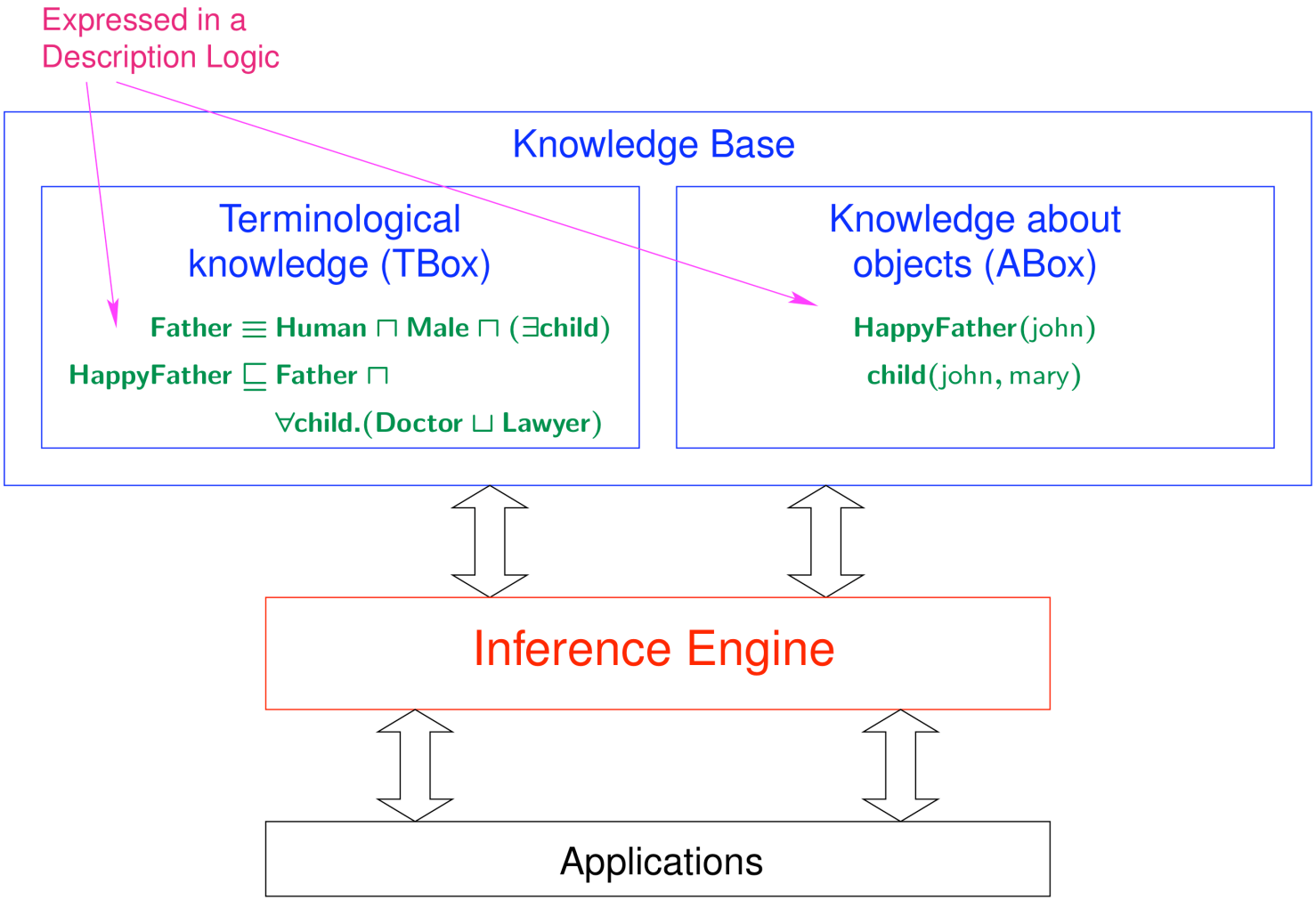
The knowledge is asserted through so-called assertions, i.e., logical axioms.

# Ingredients of a Description Logic

A **DL** is characterized by:

1. A **description language**: how to form concepts and roles
   $\mathsf{Human} \sqcap \mathsf{Male} \sqcap \exists \mathsf{hasChild} \sqcap \forall \mathsf{hasChild}.(\mathsf{Doctor} \sqcup \mathsf{Lawyer})$
2. A mechanism to **specify knowledge** about concepts and roles (i.e., a TBox)
   $\mathcal{T} = \{ \mathsf{Father} \equiv \mathsf{Human} \sqcap \mathsf{Male} \sqcap \exists \mathsf{hasChild},$
   $\mathsf{HappyFather} \sqsubseteq \mathsf{Father} \sqcap \forall \mathsf{hasChild}.(\mathsf{Doctor} \sqcup \mathsf{Lawyer}) \}$
3. A mechanism to specify **properties of objects** (i.e., an ABox)
   $\mathcal{A} = \{ \mathsf{HappyFather}(\mathtt{john}), \quad \mathsf{hasChild}(\mathtt{john}, \mathtt{mary}) \}$
4. A set of **inference services**: how to reason on a given KB
   $\mathcal{T} \models \mathsf{HappyFather} \sqsubseteq \exists \mathsf{hasChild}.(\mathsf{Doctor} \sqcup \mathsf{Lawyer})$
   $\mathcal{T} \cup \mathcal{A} \models (\mathsf{Doctor} \sqcup \mathsf{Lawyer})(\mathtt{mary})$

# Architecture of a Description Logic system



# Description language

A description language provides the means for defining:

- concepts, corresponding to classes: interpreted as sets of objects;
- roles, corresponding to relationships: interpreted as binary relations on objects.

To define concepts and roles:

- We start from a (finite) alphabet of **atomic concepts** and **atomic roles**, i.e., simply names for concept and roles.
- Then, by applying specific **constructors**, we can build **complex concepts** and **roles**, starting from the atomic ones.

A **description language** is characterized by the set of constructs that are available for that.

# Semantics of a description language

The **formal semantics** of DLs is given in terms of interpretations.

Def.: An **interpretation** $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of:

- a nonempty set $\Delta^{\mathcal{I}}$, the domain of $\mathcal{I}$
- an interpretation function $\cdot^{\mathcal{I}}$, which maps
  - each individual $c$ to an element $c^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$
  - each atomic concept $A$ to a subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$
  - each atomic role $P$ to a subset $P^{\mathcal{I}}$ of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$

The interpretation function is extended to complex concepts and roles according to their syntactic structure.

We adopt **the unique name assumption**, i.e., when $c_1 \neq c_2$ then $c_1^{\mathcal{I}} \neq c_2^{\mathcal{I}}$.

# Concept constructors

| Construct | Syntax | Example | Semantics |
|---|---|---|---|
| atomic concept | $A$ | Doctor | $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ |
| atomic role | $P$ | hasChild | $P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ |
| atomic negation | $\neg A$ | $\neg$Doctor | $\Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$ |
| conjunction | $C \sqcap D$ | Hum $\sqcap$ Male | $C^{\mathcal{I}} \cap D^{\mathcal{I}}$ |
| (unqual.) exist. res. | $\exists R$ | $\exists$hasChild | $\{ a \mid \exists b. (a, b) \in R^{\mathcal{I}} \}$ |
| value restriction | $\forall R.C$ | $\forall$hasChild.Male | $\{a \mid \forall b. (a, b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}$ |
| bottom | $\bot$ | | $\emptyset$ |

($C$, $D$ denote arbitrary concepts and $R$ an arbitrary role)

The above constructs form the basic language $\mathcal{AL}$ of the family of $\mathcal{AL}$ languages.

# Additional concept and role constructors

| Construct | $\mathcal{AL}\cdot$ | Syntax | Semantics |
|---|---|---|---|
| disjunction | $\mathcal{U}$ | $C \sqcup D$ | $C^{\mathcal{I}} \cup D^{\mathcal{I}}$ |
| top | | $\top$ | $\Delta^{\mathcal{I}}$ |
| qual. exist. res. | $\mathcal{E}$ | $\exists R.C$ | $\{\ a \mid \exists b.\, (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\ \}$ |
| (full) negation | $\mathcal{C}$ | $\neg C$ | $\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$ |
| number restrictions | $\mathcal{N}$ | $(\geq k\, R)$ | $\{\ a \mid \#\{b \mid (a,b) \in R^{\mathcal{I}}\} \geq k\ \}$ |
| | | $(\leq k\, R)$ | $\{\ a \mid \#\{b \mid (a,b) \in R^{\mathcal{I}}\} \leq k\ \}$ |
| qual. number restrictions | $\mathcal{Q}$ | $(\geq k\, R.\, C)$ | $\{\ a \mid \#\{b \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \geq k\ \}$ |
| | | $(\leq k\, R.\, C)$ | $\{\ a \mid \#\{b \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \leq k\ \}$ |
| inverse role | $\mathcal{I}$ | $R^-$ | $\{\ (a,b) \mid (b,a) \in R^{\mathcal{I}}\ \}$ |
| role closure | $reg$ | $\mathcal{R}^*$ | $(R^{\mathcal{I}})^*$ |

*Many different DL constructs and their combinations have been investigated.*

# Further examples of DL constructs

- Disjunction: $\forall \mathsf{hasChild}.(\mathsf{Doctor} \sqcup \mathsf{Lawyer})$
- Qualified existential restriction: $\exists \mathsf{hasChild}.\mathsf{Doctor}$
- Full negation: $\neg(\mathsf{Doctor} \sqcup \mathsf{Lawyer})$
- Number restrictions: $(\geq 2\, \mathsf{hasChild}) \sqcap (\leq 1\, \mathsf{sibling})$
- Qualified number restrictions: $(\geq 2\, \mathsf{hasChild}.\, \mathsf{Doctor})$
- Inverse role: $\forall \mathsf{hasChild}^{-}.\mathsf{Doctor}$
- Reflexive-transitive role closure: $\exists \mathsf{hasChild}^{*}.\mathsf{Doctor}$

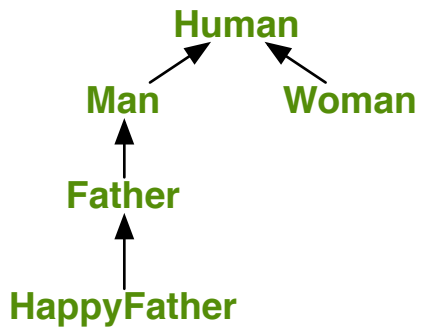
# Reasoning on concept expressions

An interpretation $\mathcal{I}$ is a **model** of a concept $C$ if $C^{\mathcal{I}} \neq \emptyset$.

**Basic reasoning tasks:**

1. **Concept satisfiability**: does $C$ admit a model?
2. **Concept subsumption** $C \sqsubseteq D$: does $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ hold for all interpretations $\mathcal{I}$?

Subsumption is used to build the concept hierarchy:




*Note: (1) and (2) are mutually reducible if DL is propositionally closed.*

# Complexity of reasoning on concept expressions

**Complexity of concept satisfiability: [Donini *et al.*, 1997]**

| | |
|---|---|
| $\mathcal{AL}$, $\mathcal{ALN}$ | PTIME |
| $\mathcal{ALU}$, $\mathcal{ALUN}$ | NP-complete |
| $\mathcal{ALE}$ | coNP-complete |
| $\mathcal{ALC}$, $\mathcal{ALCN}$, $\mathcal{ALCI}$, $\mathcal{ALCQI}$ | PSPACE-complete |

*Observations:*

- Two sources of complexity:
  - union ($\mathcal{U}$) of type NP,
  - existential quantification ($\mathcal{E}$) of type coNP.

  When they are combined, the complexity jumps to PSPACE.
- Number restrictions ($\mathcal{N}$) do not add to the complexity.

# Outline of Part 2

# Structural properties vs. asserted properties

We have seen how to build complex **concept and roles expressions**, which allow one to denote classes with a complex structure.

However, in order to represent real world domains, one needs the ability to **assert properties** of classes and relationships between them (e.g., as done in UML class diagrams).

The assertion of properties is done in DLs by means of an **ontology** (or knowledge base).

# Description Logics ontology (or knowledge base)

Is a pair $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is a **TBox** and $\mathcal{A}$ is an **ABox**:

## Def.: Description Logics **TBox**

Consists of a set of **assertions** on concepts and roles:

- Inclusion assertions on concepts: $C_1 \sqsubseteq C_2$
- Inclusion assertions on roles: $R_1 \sqsubseteq R_2$
- Property assertions on (atomic) roles:
  (**transitive** $P$) (**symmetric** $P$) (**domain** $P$ $C$)
  (**functional** $P$) (**reflexive** $P$) (**range** $P$ $C$) $\cdots$

## Def.: Description Logics **ABox**

Consists of a set of **membership assertions** on individuals:

- for concepts: $A(c)$
- for roles: $P(c_1, c_2)$

(we use $c_i$ to denote individuals)

# Description Logics ontology – Example

*Note:* We use $C_1 \equiv C_2$ as an abbreviation for $C_1 \sqsubseteq C_2,\ C_2 \sqsubseteq C_1$.

TBox assertions:

- Inclusion assertions on concepts:

$$\begin{aligned} \text{Father} &\equiv \text{Human} \sqcap \text{Male} \sqcap \exists \text{hasChild} \\ \text{HappyFather} &\sqsubseteq \text{Father} \sqcap \forall \text{hasChild}.(\text{Doctor} \sqcup \text{Lawyer} \sqcup \text{HappyPerson}) \\ \text{HappyAnc} &\sqsubseteq \forall \text{descendant}.\text{HappyFather} \\ \text{Teacher} &\sqsubseteq \neg\text{Doctor} \sqcap \neg\text{Lawyer} \end{aligned}$$

- Inclusion assertions on roles:

$$\text{hasChild} \sqsubseteq \text{descendant} \qquad \text{hasFather} \sqsubseteq \text{hasChild}^-$$

- Property assertions on roles:
  (**transitive** descendant), (**reflexive** descendant), (**functional** hasFather)

ABox membership assertions:

- Teacher(`mary`), hasFather(`mary`, `john`), HappyAnc(`john`)

# Semantics of a Description Logics ontology

The semantics is given by specifying when an interpretation $\mathcal{I}$ **satisfies** an assertion:

- $C_1 \sqsubseteq C_2$ is satisfied by $\mathcal{I}$ if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$.
- $R_1 \sqsubseteq R_2$ is satisfied by $\mathcal{I}$ if $R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$.
- A property assertion (**prop** $P$) is satisfied by $\mathcal{I}$ if $P^{\mathcal{I}}$ is a relation that has the property **prop**.
  (Note: domain and range assertions can be expressed by means of concept inclusion assertions.)
- $A(c)$ is satisfied by $\mathcal{I}$ if $c^{\mathcal{I}} \in A^{\mathcal{I}}$.
- $P(c_1, c_2)$ is satisfied by $\mathcal{I}$ if $(c_1^{\mathcal{I}}, c_2^{\mathcal{I}}) \in P^{\mathcal{I}}$.

**Def.: Model of a DL ontology**

An interpretation $\mathcal{I}$ is a **model** of $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ if it satisfies all assertions in $\mathcal{T}$ and all assertions in $\mathcal{A}$.

# Outline of Part 2

# Relationship between DLs and ontology formalisms

- DLs are nowadays advocated to provide the foundations for ontology languages.
- Different versions of the W3C standard **Web Ontology Language (OWL)** have been defined as syntactic variants of certain DLs.
- DLs are also ideally suited to capture the fundamental features of conceptual modeling formalism used in information systems design:
  - **Entity-Relationship diagrams**, used in database conceptual modeling
  - **UML Class Diagrams**, used in the design phase of software applications

We briefly overview these correspondences, highlighting essential DL constructs, also in light of the tradeoff between expressive power and computational complexity of reasoning.

# DLs vs. OWL

The Web Ontology Language (OWL) comes in different variants:

- **OWL1 Lite** is a variant of the DL $\mathcal{SHIF}(D)$, where:
  - $\mathcal{S}$ stands for $\mathcal{ALC}$ extended with transitive roles,
  - $\mathcal{H}$ stands for role hierarchies (i.e., role inclusion assertions),
  - $\mathcal{I}$ stands for inverse roles,
  - $\mathcal{F}$ stands for functionality of roles,
  - $(D)$ stand for data types, which are necessary in any practical knowledge representation language.
- **OWL1 DL** is a variant of $\mathcal{SHOIN}(D)$, where:
  - $\mathcal{O}$ stands for nominals, which means the possibility of using individuals in the TBox (i.e., the intensional part of the ontology),
  - $\mathcal{N}$ stands for (unqualified) number restrictions.

# DLs vs. OWL2

A new version of OWL, **OWL2**, is currently being standardized by the W3C:

- **OWL2 DL** is a variant of $\mathcal{SROIQ}(D)$, which adds to OWL1 DL several constructs, while still preserving decidability of reasoning.
  - $\mathcal{Q}$ stands for qualified number restrictions.
  - $\mathcal{R}$ stands for regular role hierarchies, where role chaining might be used in the left-hand side of role inclusion assertions, with suitable acyclicity conditions.

- OWL2 defines also three **profiles**: OWL2 QL, OWL2 EL, OWL2 RL.
  - Each profile corresponds to a syntactic fragment (i.e., a sub-language) of OWL2 DL that is targeted towards a specific use.
  - The restrictions in each profile guarantee better computational properties than those of OWL2 DL.
  - The **OWL2 QL** profile is derived from the DLs of the *DL-Lite* family (see later).

# DLs vs. UML Class Diagrams

There is a tight correspondence between variants of DLs and UML Class Diagrams [Berardi *et al.*, 2005].

- We can devise two transformations:
  - one that associates to each UML Class Diagram $\mathcal{D}$ a DL TBox $\mathcal{T}_\mathcal{D}$.
  - one that associates to each DL TBox $\mathcal{T}$ a UML Class Diagram $\mathcal{D}_\mathcal{T}$.
- The transformations are not model-preserving, but are based on a correspondence between instantiations of the Class Diagram and models of the associated ontology.
- The transformations are **satisfiability-preserving**, i.e., a class $C$ is consistent in $\mathcal{D}$ iff the corresponding concept is satisfiable in $\mathcal{T}$.

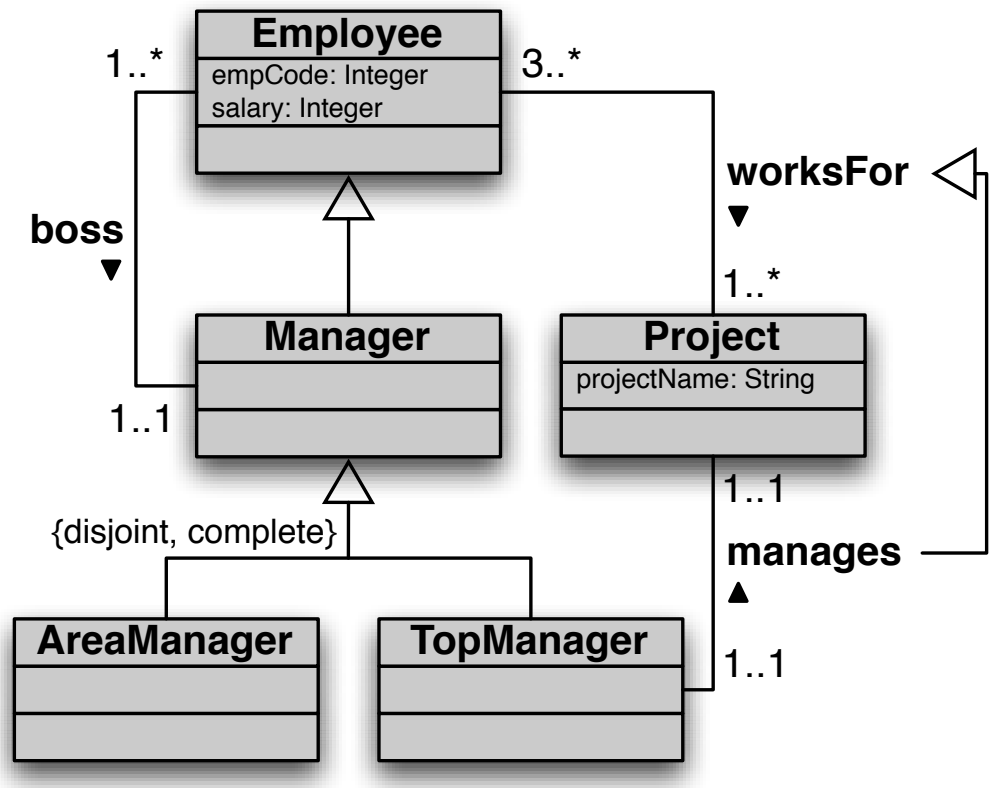
# Encoding UML Class Diagrams in DLs

The ideas behind the encoding of a UML Class Diagram $\mathcal{D}$ in terms of a DL TBox $\mathcal{T}_\mathcal{D}$ are quite natural:

- Each class is represented by an atomic concept.
- Each attribute is represented by a role.
- Each binary association is represented by a role.
- Each non-binary association is reified, i.e., represented as a concept connected to its components by roles.
- Each part of the diagram is encoded by suitable assertions.

We illustrate the encoding by means of an example.

# Encoding UML Class Diagrams in DLs – Example




$$
\begin{array}{rcl}
\text{Manager} & \sqsubseteq & \text{Employee} \\
\text{AreaManager} & \sqsubseteq & \text{Manager} \\
\text{TopManager} & \sqsubseteq & \text{Manager} \\
\text{Manager} & \sqsubseteq & \text{AreaManager} \sqcup \text{TopManager} \\
\text{AreaManager} & \sqsubseteq & \neg\text{TopManager} \\
\exists \text{salary} & \sqsubseteq & \text{Employee} \\
\exists \text{salary}^- & \sqsubseteq & \text{Integer} \\
\exists \text{worksFor} & \sqsubseteq & \text{Employee} \\
\exists \text{worksFor}^- & \sqsubseteq & \text{Project} \\
\text{Employee} & \sqsubseteq & \exists \text{worksFor} \\
\text{Project} & \sqsubseteq & (\geq 3\, \text{worksFor}^-) \\
(\textbf{funct}\ \text{manages}) & & \\
(\textbf{funct}\ \text{manages}^-) & & \\
\text{manages} & \sqsubseteq & \text{worksFor} \\
\multicolumn{3}{c}{\ldots}
\end{array}
$$

*Note:* Domain and range of associations are expressed by means of concept inclusions.

# Encoding DL TBoxes in UML Class Diagrams

The encoding of an $\mathcal{ALC}$ TBox $\mathcal{T}$ in terms of a UML Class Diagram $\mathcal{T}_\mathcal{D}$ is based on the following observations:

- We can restrict the attention to $\mathcal{ALC}$ TBoxes, that are constituted by concept inclusion assertions of a simplified form (single atomic concept on the left, and a single concept constructor on the right).
- For each such inclusion assertion, the encoding introduces a portion of UML Class Diagram, that may refer to some common classes.

By means of the two encodings, one can relate reasoning in DLs to reasoning on UML Class Diagrams.

# Outline of Part 2

# Outline of Part 2

# Logical implication

The fundamental reasoning service from which all other ones can be easily derived is ...

**Def.: Logical implication**

An ontology $\mathcal{O}$ **logically implies** an assertion $\alpha$, written $\mathcal{O} \models \alpha$, if $\alpha$ is satisfied by all models of $\mathcal{O}$.

We can provide an analogous definition for a TBox $\mathcal{T}$ instead of an ontology $\mathcal{O}$.

# TBox reasoning

- **Concept Satisfiability:** $C$ is satisfiable wrt $\mathcal{T}$, if there is a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}}$ is not empty, i.e., $\mathcal{T} \not\models C \equiv \bot$.
- **Subsumption:** $C_1$ is subsumed by $C_2$ wrt $\mathcal{T}$, if for every model $\mathcal{I}$ of $\mathcal{T}$ we have $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$, i.e., $\mathcal{T} \models C_1 \sqsubseteq C_2$.
- **Equivalence:** $C_1$ and $C_2$ are equivalent wrt $\mathcal{T}$ if for every model $\mathcal{I}$ of $\mathcal{T}$ we have $C_1^{\mathcal{I}} = C_2^{\mathcal{I}}$, i.e., $\mathcal{T} \models C_1 \equiv C_2$.
- **Disjointness:** $C_1$ and $C_2$ are disjoint wrt $\mathcal{T}$ if for every model $\mathcal{I}$ of $\mathcal{T}$ we have $C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} = \emptyset$, i.e., $\mathcal{T} \models C_1 \sqcap C_2 \equiv \bot$.
- **Functionality implication:** A functionality assertion $(\textbf{funct}\ R)$ is logically implied by $\mathcal{T}$ if for every model $\mathcal{I}$ of $\mathcal{T}$, we have that $(o, o_1) \in R^{\mathcal{I}}$ and $(o, o_2) \in R^{\mathcal{I}}$ implies $o_1 = o_2$, i.e., $\mathcal{T} \models (\textbf{funct}\ R)$.

*Analogous definitions hold for role satisfiability, subsumption, equivalence, and disjointness.*

# Reasoning over an ontology

- **Ontology Satisfiability:** Verify whether an ontology $\mathcal{O}$ is satisfiable, i.e., whether $\mathcal{O}$ admits at least one model.
- **Concept Instance Checking:** Verify whether an individual $c$ is an instance of a concept $C$ in every model of $\mathcal{O}$, i.e., whether $\mathcal{O} \models C(c)$.
- **Role Instance Checking:** Verify whether a pair $(c_1, c_2)$ of individuals is an instance of a role $R$ in every model of $\mathcal{O}$, i.e., whether $\mathcal{O} \models R(c_1, c_2)$.
- **Query Answering:** see later ...

# Reasoning in Description Logics – Example

TBox:

- Inclusion assertions on concepts:

$$\begin{aligned} \text{Father} &\equiv \text{Human} \sqcap \text{Male} \sqcap \exists \text{hasChild} \\ \text{HappyFather} &\sqsubseteq \text{Father} \sqcap \forall \text{hasChild}.(\text{Doctor} \sqcup \text{Lawyer} \sqcup \text{HappyPerson}) \\ \text{HappyAnc} &\sqsubseteq \forall \text{descendant}.\text{HappyFather} \\ \text{Teacher} &\sqsubseteq \neg\text{Doctor} \sqcap \neg\text{Lawyer} \end{aligned}$$

- Inclusion assertions on roles:

$$\text{hasChild} \sqsubseteq \text{descendant} \qquad \text{hasFather} \sqsubseteq \text{hasChild}^{-}$$

- Property assertions on roles:
  (**transitive** descendant), (**reflexive** descendant), (**functional** hasFather)

The above TBox logically implies: $\text{HappyAncestor} \sqsubseteq \text{Father}$.

- Membership assertions:
  $\text{Teacher}(\texttt{mary})$, $\text{hasFather}(\texttt{mary}, \texttt{john})$, $\text{HappyAnc}(\texttt{john})$

The above TBox and ABox logically imply: $\text{HappyPerson}(\texttt{mary})$

# Complexity of reasoning over DL ontologies

Reasoning over DL ontologies is much more complex than reasoning over concept expressions:

Bad news:

- without restrictions on the form of TBox assertions, reasoning over DL ontologies is already EXPTIME-**hard**, even for very simple DLs (see, e.g., [Donini, 2003]).

Good news:

- We can add a lot of expressivity (i.e., essentially all DL constructs seen so far), while still staying within the EXPTIME upper bound.
- There are DL reasoners that perform reasonably well in practice for such DLs (e.g, Racer, Pellet, Fact++, ...) [Möller and Haarslev, 2003].

# Reasoning on UML Class Diagrams using DLs

- The two encodings seen previously show that DL TBoxes and UML Class Diagrams essentially have the **same expressive power**.
- Since the encodings are polynomial (actually LogSpace), we also get that reasoning over UML Class Diagrams has the same complexity as reasoning over ontologies in expressive DLs, i.e., it is ExpTime-complete.
- The high complexity is caused by:
  1. the possibility to use disjunction (covering constraints)
  2. the interaction between role inclusions and functionality constraints (maximum 1 cardinality)

Without (1) and restricting (2), reasoning becomes simpler [Artale *et al.*, 2007]:

- NLogSpace-complete in combined complexity
- in LogSpace in data complexity (see later)

# Efficient reasoning on UML Class Diagrams

We are interested in using UML Class Diagrams to specify ontologies in the context of ontology-based data access.

**Questions**

- Which is the right combination of constructs to allow in UML Class Diagrams to be used for OBDA?
- Are there techniques for query answering in this case that can be derived from Description Logics?
- Can query answering be done efficiently in the size of the data?
- If yes, can we leverage relational database technology for query answering?

# Outline of Part 2

# 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.

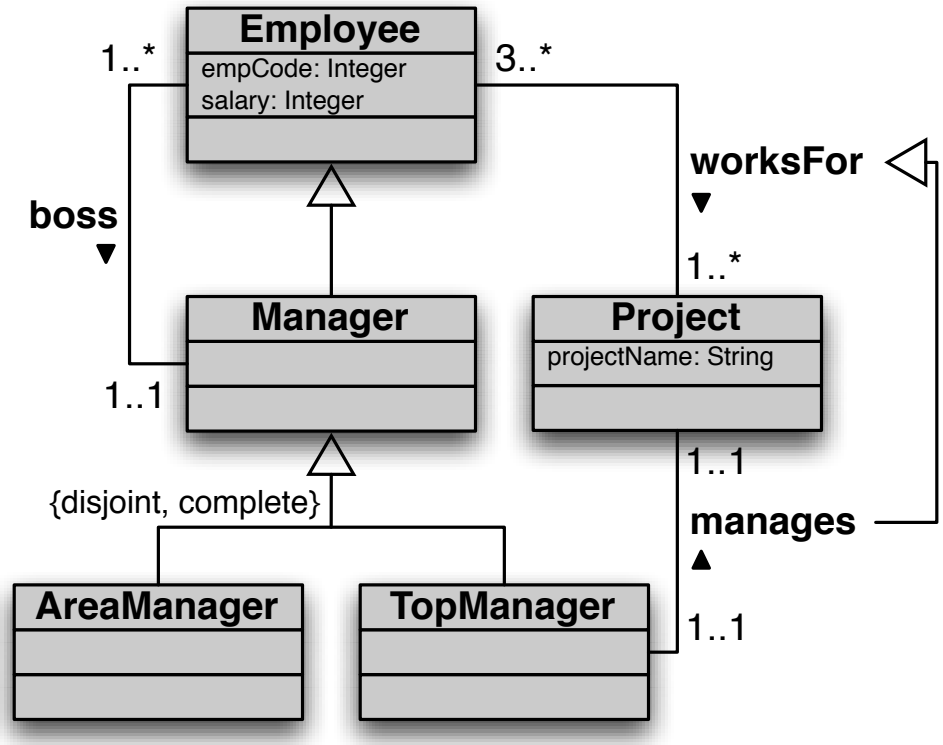
**Def.: A conjunctive query $q(\vec{x})$ over an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$**

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 2

# 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 $\mathcal{O}$**

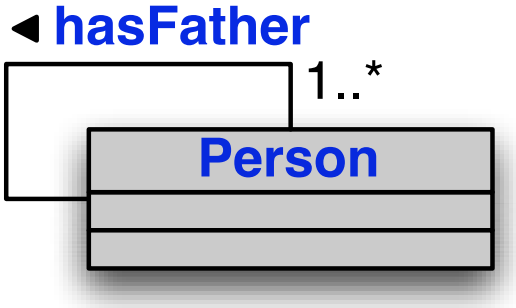
Is the problem of **computing the certain answers** to a query over $\mathcal{O}$.

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 $\mathcal{T}$: $\exists \mathsf{hasFather} \sqsubseteq \mathsf{Person}$
$\exists \mathsf{hasFather}^- \sqsubseteq \mathsf{Person}$
$\mathsf{Person} \sqsubseteq \exists \mathsf{hasFather}$

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

Queries:

$q_1(x, y) \leftarrow \mathsf{hasFather}(x, y)$
$q_2(x) \leftarrow \exists y.\, \mathsf{hasFather}(x, y)$
$q_3(x) \leftarrow \exists y_1, y_2, y_3.\, \mathsf{hasFather}(x, y_1) \wedge \mathsf{hasFather}(y_1, y_2) \wedge \mathsf{hasFather}(y_2, y_3)$
$q_4(x, y_3) \leftarrow \exists y_1, y_2.\, \mathsf{hasFather}(x, y_1) \wedge \mathsf{hasFather}(y_1, y_2) \wedge \mathsf{hasFather}(y_2, y_3)$

**Certain answers:**

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

# Unions of conjunctive queries

We consider also unions of CQs over an ontology.

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

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

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**

$$q(x) \leftarrow (\mathsf{Manager}(x) \wedge \mathsf{worksFor}(x, \mathtt{tones})) \;\vee\; (\exists y.\, \mathsf{boss}(x, y) \wedge \mathsf{worksFor}(y, \mathtt{tones}))$$

In datalog notation:

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

# Outline of Part 2

# Data and combined complexity

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).

$\rightsquigarrow$ **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

Query answering has been studied extensively for (unions of) CQs and various ontology languages:

| | Combined complexity | Data complexity |
|---|---|---|
| Plain databases | NP-complete | in $\mathrm{AC}^0$ (1) |
| OWL 2 (and less) | 2ExpTime-complete (3) | coNP-hard (2) |

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

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

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

**Questions**

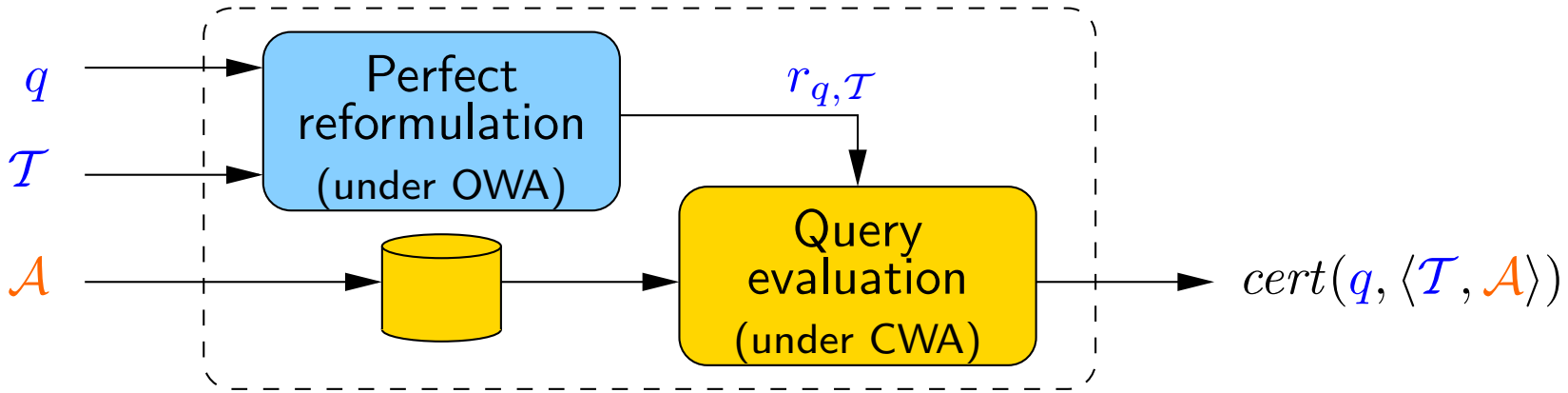
- Can we find interesting (description) logics for which query answering can be done efficiently (i.e., in $\mathrm{AC}^0$)?
- If yes, can we leverage relational database technology for query answering?

# 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 $Q$ be a query language and $\mathcal{L}$ be 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).
  $\rightsquigarrow$ Query evaluation can be done in SQL, i.e., via an RDBMS (*Note:* FOL is in $\mathrm{AC}^0$).
- When we can rewrite into an NLogSpace-hard language.
  $\rightsquigarrow$ Query evaluation requires (at least) linear recursion.
- When we can rewrite into a PTime-hard language.
  $\rightsquigarrow$ Query evaluation requires full recursion (e.g., Datalog).
- When we can rewrite into a coNP-hard language.
  $\rightsquigarrow$ Query evaluation requires (at least) power of Disjunctive Datalog.

# Outline of Part 2

5 A quick overview of Description Logics

6 Reasoning and query answering in Description Logics

7 The *DL-Lite* family of tractable Description Logics
- The *DL-Lite* family
- Properties of *DL-Lite*

# Outline of Part 2

# The *DL-Lite* family

- A family of DLs optimized according to the tradeoff between expressive power and **complexity** of query answering, with emphasis on **data**.
- Carefully designed to have nice computational properties for answering UCQs (i.e., computing certain answers):
  - The same data complexity as relational databases.
  - In fact, query answering can be delegated to a relational DB engine.
  - The DLs of the *DL-Lite* family are essentially the maximally expressive ontology languages enjoying these nice computational properties.
- Captures conceptual modeling formalism.

The *DL-Lite* family provides new foundations for Ontology-Based Data Access.

# Basic features of *DL-Lite*$_{\mathcal{A}}$

*DL-Lite*$_{\mathcal{A}}$ is an expressive member of the *DL-Lite* family.

- Takes into account the distinction between **objects** and **values**:
  - Objects are elements of an abstract interpretation domain.
  - Values are elements of concrete data types, such as integers, strings, ecc.
  - Values are connected to objects through **attributes** (rather than roles).
- Is equipped with identification constraints.
- Captures most of UML class diagrams and Extended ER diagrams.
- Enjoys **FOL-rewritability**, and hence is $\mathrm{AC}^0$ in data complexity.

# Syntax of the *DL-Lite*$_{\mathcal{A}}$ description language

- Concept expressions: atomic concept $A$

$$\begin{array}{lcl} B & \longrightarrow & A \mid \exists Q \mid \delta(U) \\ C & \longrightarrow & \top_C \mid B \mid \neg B \end{array}$$

- Role expressions: atomic role $P$

$$\begin{array}{lcl} Q & \longrightarrow & P \mid P^- \\ R & \longrightarrow & Q \mid \neg Q \end{array}$$

- Value-domain expressions: each $T_i$ is one of the RDF datatypes

$$\begin{array}{lcl} E & \longrightarrow & \rho(U) \\ F & \longrightarrow & \top_D \mid T_1 \mid \cdots \mid T_n \end{array}$$

- Attribute expressions: atomic attribute $U$

$$V \longrightarrow U \mid \neg U$$

# Semantics of *DL-Lite*$_{\mathcal{A}}$ – Objects vs. values

| | Objects | Values |
|---|---|---|
| Interpretation domain $\Delta^{\mathcal{I}}$ | Domain of objects $\Delta_O^{\mathcal{I}}$ | Domain of values $\Delta_V^{\mathcal{I}}$ |
| Alphabet $\Gamma$ of constants | Object constants $\Gamma_O$<br>$c^{\mathcal{I}} \in \Delta_O^{\mathcal{I}}$ | Value constants $\Gamma_V$<br>$d^{\mathcal{I}} = \mathit{val}(d)$ given a priori |
| Unary predicates | Concept $C$<br>$C^{\mathcal{I}} \in \Delta_O^{\mathcal{I}}$ | RDF datatype $T_i$<br>$T_i^{\mathcal{I}} \in \Delta_V^{\mathcal{I}}$ is given a priori |
| Binary predicates | Role $R$<br>$R^{\mathcal{I}} \in \Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}}$ | Attribute $V$<br>$V^{\mathcal{I}} \in \Delta_O^{\mathcal{I}} \times \Delta_V^{\mathcal{I}}$ |

# Semantics of the *DL-Lite*$_\mathcal{A}$ constructs

| Construct | Syntax | Example | Semantics |
|---|---|---|---|
| top concept | $\top_C$ | | $\top_C^{\mathcal{I}} = \Delta_O^{\mathcal{I}}$ |
| atomic concept | $A$ | Doctor | $A^{\mathcal{I}} \subseteq \Delta_O^{\mathcal{I}}$ |
| existential restriction | $\exists Q$ | $\exists$child$^-$ | $\{o \mid \exists o'.\, (o, o') \in Q^{\mathcal{I}}\}$ |
| concept negation | $\neg B$ | $\neg\exists$child | $\Delta^{\mathcal{I}} \setminus B^{\mathcal{I}}$ |
| attribute domain | $\delta(U)$ | $\delta$(salary) | $\{o \mid \exists v.\, (o, v) \in U^{\mathcal{I}}\}$ |
| atomic role | $P$ | child | $P^{\mathcal{I}} \subseteq \Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}}$ |
| inverse role | $P^-$ | child$^-$ | $\{(o, o') \mid (o', o) \in P^{\mathcal{I}}\}$ |
| role negation | $\neg Q$ | $\neg$manages | $(\Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}}) \setminus Q^{\mathcal{I}}$ |
| top domain | $\top_D$ | | $\top_D^{\mathcal{I}} = \Delta_V^{\mathcal{I}}$ |
| datatype | $T_i$ | `xsd:int` | $val(T_i) \subseteq \Delta_V^{\mathcal{I}}$ |
| attribute range | $\rho(U)$ | $\rho$(salary) | $\{v \mid \exists o.\, (o, v) \in U^{\mathcal{I}}\}$ |
| atomic attribute | $U$ | salary | $U^{\mathcal{I}} \subseteq \Delta_O^{\mathcal{I}} \times \Delta_V^{\mathcal{I}}$ |
| attribute negation | $\neg U$ | $\neg$salary | $(\Delta_O^{\mathcal{I}} \times \Delta_V^{\mathcal{I}}) \setminus U^{\mathcal{I}}$ |
| object constant | $c$ | `john` | $c^{\mathcal{I}} \in \Delta_O^{\mathcal{I}}$ |
| value constant | $d$ | `'john'` | $val(d) \in \Delta_V^{\mathcal{I}}$ |

# *DL-Lite*$_{\mathcal{A}}$ assertions

TBox assertions can have the following forms:

- Inclusion assertions:

  $B \sqsubseteq C$ concept inclusion  $E \sqsubseteq F$ value-domain inclusion
  $Q \sqsubseteq R$ role inclusion  $U \sqsubseteq V$ attribute inclusion

- Functionality assertions:

  (**funct** $Q$) role functionality  (**funct** $U$) attribute functionality

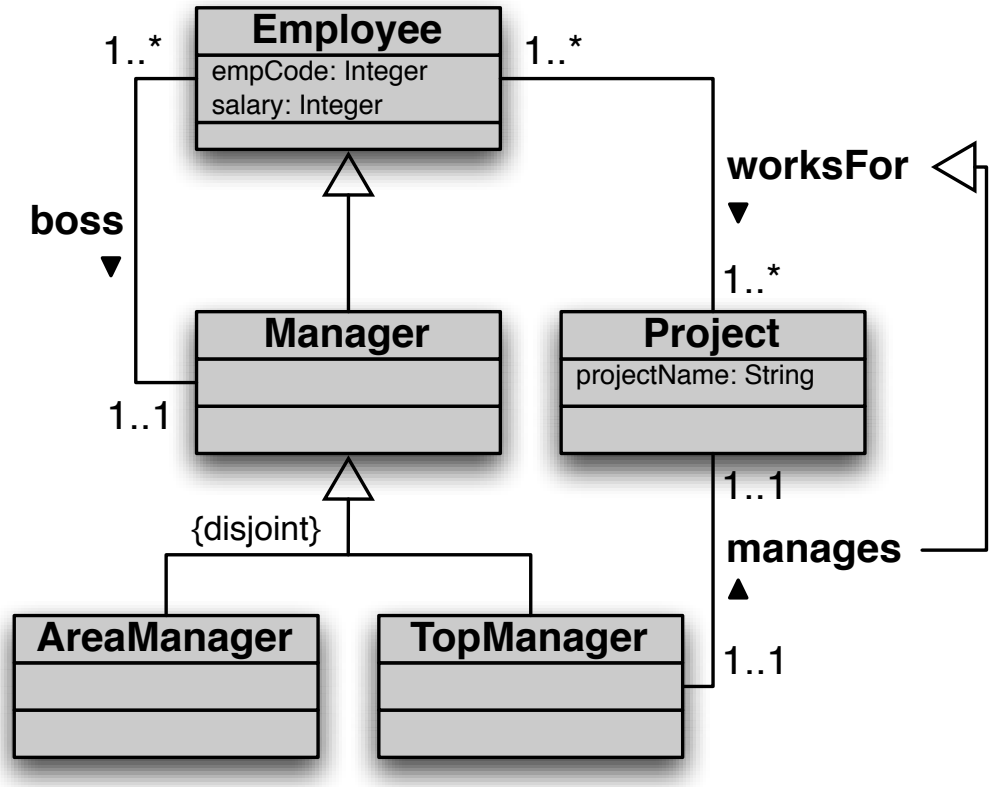
- Identification constraints: (**id** $B\ I_1, \ldots, I_n$)
  where each $I_j$ is a role, an inverse role, or an attribute

ABox assertions: $A(c)$, $P(c, c')$, $U(c, d)$,
where $c$, $c'$ are object constants and $d$ is a value constant

# Semantics of the *DL-Lite*$_{\mathcal{A}}$ assertions

| Assertion | Syntax | Example | Semantics |
|---|---|---|---|
| conc. incl. | $B \sqsubseteq C$ | Father $\sqsubseteq \exists$child | $B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$ |
| role incl. | $Q \sqsubseteq R$ | father $\sqsubseteq$ anc | $Q^{\mathcal{I}} \subseteq R^{\mathcal{I}}$ |
| v.dom. incl. | $E \sqsubseteq F$ | $\rho$(age) $\sqsubseteq$ `xsd:int` | $E^{\mathcal{I}} \subseteq F^{\mathcal{I}}$ |
| attr. incl. | $U \sqsubseteq V$ | offPhone $\sqsubseteq$ phone | $U^{\mathcal{I}} \subseteq V^{\mathcal{I}}$ |
| role funct. | (**funct** $Q$) | (**funct** father) | $\forall o, o_1, o_2.(o, o_1) \in Q^{\mathcal{I}} \wedge (o, o_2) \in Q^{\mathcal{I}} \rightarrow o_1 = o_2$ |
| att. funct. | (**funct** $U$) | (**funct** ssn) | $\forall o, v, v'.(o, v) \in U^{\mathcal{I}} \wedge (o, v') \in U^{\mathcal{I}} \rightarrow v = v'$ |
| id const. | (**id** $B$ $I_1, \ldots, I_n$) | (**id** Person name, dob) | $I_1, \ldots, I_n$ identify instances of $B$ |
| mem. asser. | $A(c)$ | Father(bob) | $c^{\mathcal{I}} \in A^{\mathcal{I}}$ |
| mem. asser. | $P(c_1, c_2)$ | child(bob, ann) | $(c_1^{\mathcal{I}}, c_2^{\mathcal{I}}) \in P^{\mathcal{I}}$ |
| mem. asser. | $U(c, d)$ | phone(bob, '2345') | $(c^{\mathcal{I}}, \mathit{val}(d)) \in U^{\mathcal{I}}$ |

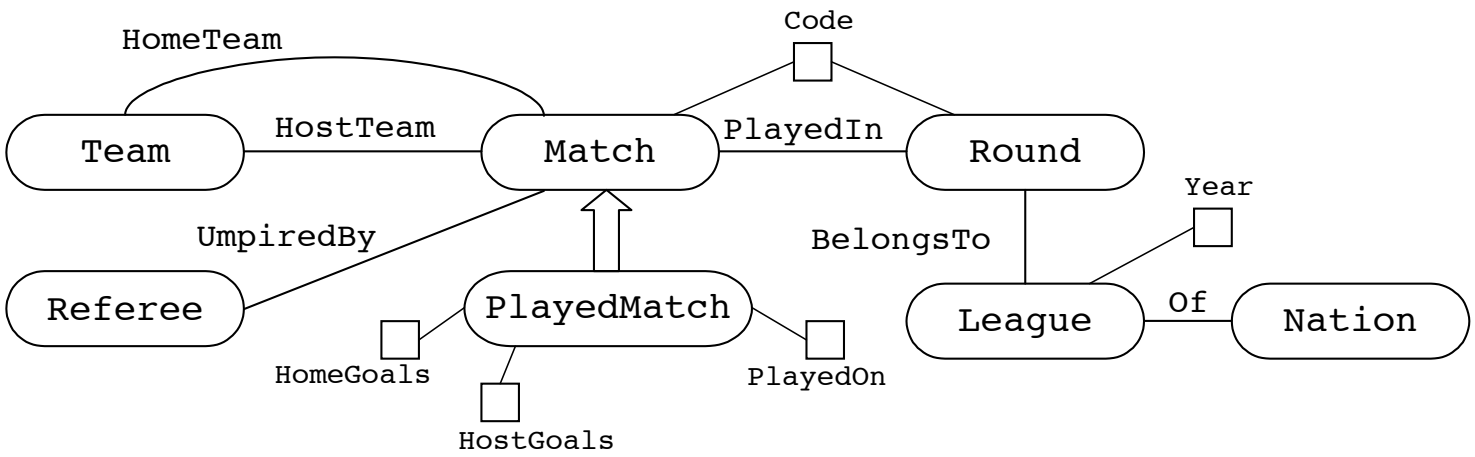
# *DL-Lite*$_{\mathcal{A}}$ – Example




$$
\begin{aligned}
\text{Manager} &\sqsubseteq \text{Employee}\\
\text{AreaManager} &\sqsubseteq \text{Manager}\\
\text{TopManager} &\sqsubseteq \text{Manager}\\
\text{AreaManager} &\sqsubseteq \neg\text{TopManager}\\
\text{Employee} &\sqsubseteq \delta(\text{empCode})\\
\delta(\text{empCode}) &\sqsubseteq \text{Employee}\\
\rho(\text{empCode}) &\sqsubseteq \texttt{xsd:int}\\
&(\textbf{funct}\ \text{empCode})\\
&(\textbf{id}\ \text{Employee empCode})\\
\exists\text{worksFor} &\sqsubseteq \text{Employee}\\
\exists\text{worksFor}^{-} &\sqsubseteq \text{Project}\\
\text{Employee} &\sqsubseteq \exists\text{worksFor}\\
\text{Project} &\sqsubseteq \exists\text{worksFor}^{-}\\
&(\textbf{funct}\ \text{manages})\\
&(\textbf{funct}\ \text{manages}^{-})\\
\text{manages} &\sqsubseteq \text{worksFor}\\
&\vdots
\end{aligned}
$$

*Note:* *DL-Lite*$_{\mathcal{A}}$ cannot capture completeness of a hierarchy. This would require **disjunction** (i.e., **OR**).

# Identification assertions – Example



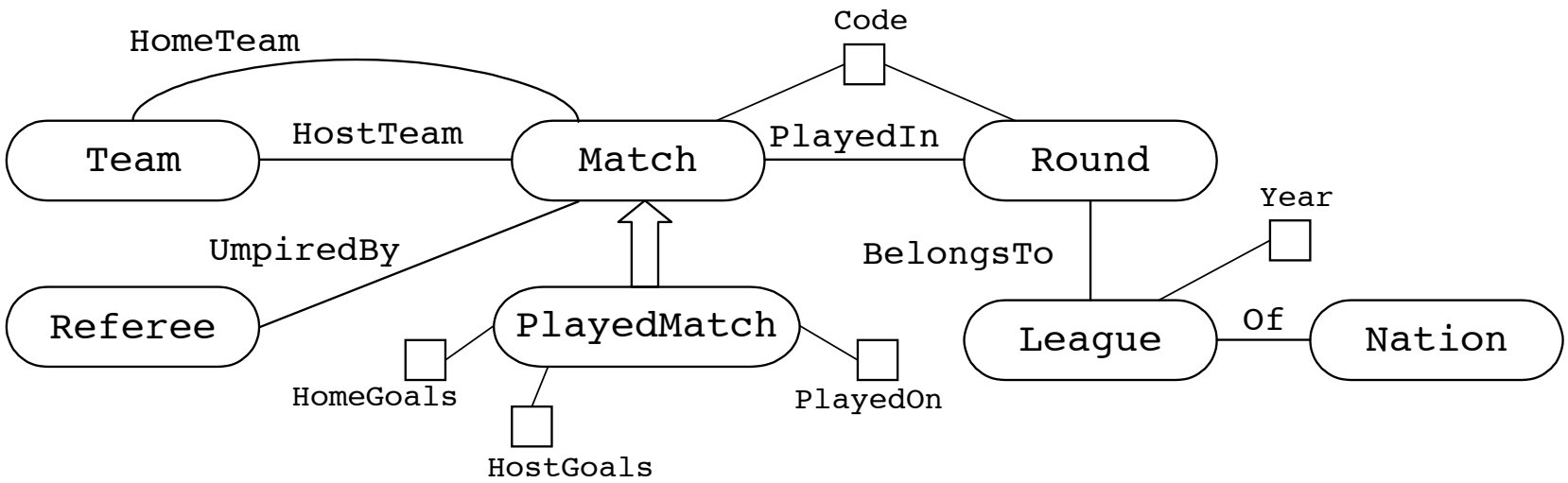

## What we would like to additionally capture:

1. No two leagues with the same year and the same nation exist
2. Within a certain league, the code associated to a round is unique
3. Every match is identified by its code within its round
4. Every referee can umpire at most one match in the same round
5. No team can be the home team of more than one match per round
6. No team can be the host team of more than one match per round

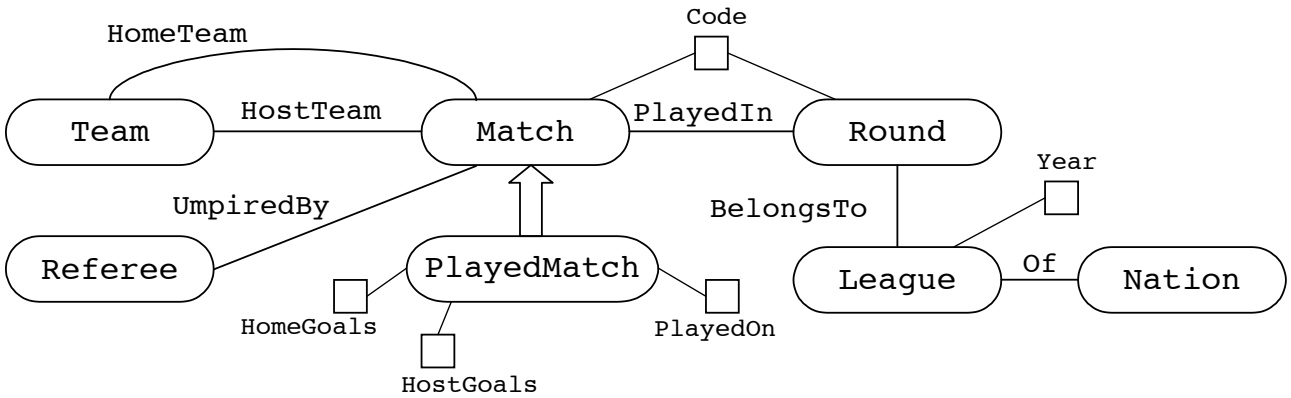
# Identification assertions – Example (cont'd)

$\mathsf{League} \sqsubseteq \exists \mathsf{of}$
$\exists \mathsf{of} \sqsubseteq \mathsf{League}$
$\exists \mathsf{of}^- \sqsubseteq \mathsf{Nation}$
$\mathsf{Round} \sqsubseteq \exists \mathsf{belongsTo}$
$\exists \mathsf{belongsTo} \sqsubseteq \mathsf{Round}$
$\exists \mathsf{belongsTo}^- \sqsubseteq \mathsf{League}$
$\mathsf{Match} \sqsubseteq \exists \mathsf{playedIn}$
...

$\mathsf{PlayedMatch} \sqsubseteq \mathsf{Match}$
$\mathsf{Match} \sqsubseteq \delta(\mathsf{code})$
$\mathsf{Round} \sqsubseteq \delta(\mathsf{code})$
$\mathsf{PlayedMatch} \sqsubseteq \delta(\mathsf{playedOn})$
...
$\rho(\mathsf{playedOn}) \sqsubseteq \mathtt{xsd:date}$
$\rho(\mathsf{code}) \sqsubseteq \mathtt{xsd:int}$
...



# Identification assertions – Example (cont'd)




1. No two leagues with the same year and the same nation exist
2. Within a certain league, the code associated to a round is unique
3. Every match is identified by its code within its round
4. Every referee can umpire at most one match in the same round
5. No team can be the home team of more than one match per round
6. No team can be the host team of more than one match per round

(**id** League of, year)
(**id** Round belongsTo, code)
(**id** Match playedIn, code)
(**id** Match umpiredBy, playedIn)
(**id** Match home, playedIn)
(**id** Match host, playedIn)

# Restriction on TBox assertions in *DL-Lite*$_\mathcal{A}$ ontologies

We will see that, to ensure FOL-rewritability, we have to impose a **restriction** on the use of functionality and role/attribute inclusions.

**Restriction on *DL-Lite*$_\mathcal{A}$ TBoxes**

**No functional or identifying role or attribute can be specialized** by using it in the right-hand side of a role or attribute inclusion assertion.

Formally:

- If $Q \sqsubseteq P$, or $Q \sqsubseteq P^-$, or (**id** $B$ $\ldots, P, \ldots$), or (**id** $B$ $\ldots, P^-, \ldots$) is in $\mathcal{T}$, then (**funct** $P$) and (**funct** $P^-$) are **not in** $\mathcal{T}$.
- If $U' \sqsubseteq U$ or (**id** $B$ $\ldots, U, \ldots$) is in $\mathcal{T}$, then (**funct** $U$) is **not in** $\mathcal{T}$.

# *DL-Lite*$_{\mathcal{F}}$ and *DL-Lite*$_{\mathcal{R}}$

We consider also two sub-languages of *DL-Lite*$_{\mathcal{A}}$ (that trivially obey the previous restriction):

- *DL-Lite*$_{\mathcal{F}}$: Allows for functionality assertions, but does not allow for role inclusion assertions.
- *DL-Lite*$_{\mathcal{R}}$: Allows for role inclusion assertions, but does not allow for functionality assertions.

In both *DL-Lite*$_{\mathcal{F}}$ and *DL-Lite*$_{\mathcal{R}}$ we do not consider data values (and hence drop value domains and attributes).

*Note:* We simply use *DL-Lite* to refer to any of the logics of the *DL-Lite* family.

# Outline of Part 2

5 A quick overview of Description Logics

6 Reasoning and query answering in Description Logics

7 The *DL-Lite* family of tractable Description Logics
- The *DL-Lite* family
- Properties of *DL-Lite*

# Capturing basic ontology constructs in *DL-Lite*$_{\mathcal{A}}$

| | |
|---|---|
| ISA between classes | $A_1 \sqsubseteq A_2$ |
| Disjointness between classes | $A_1 \sqsubseteq \neg A_2$ |
| Mandatory participation to relations | $A_1 \sqsubseteq \exists P \quad A_2 \sqsubseteq \exists P^-$ |
| Domain and range of relations | $\exists P \sqsubseteq A_1 \quad \exists P^- \sqsubseteq A_2$ |
| Functionality of relations | (**funct** $P$) (**funct** $P^-$) |
| ISA between relations | $Q_1 \sqsubseteq Q_2$ |
| Disjointness between relations | $Q_1 \sqsubseteq \neg Q_2$ |
| Domain and range of attributes | $\delta(U) \sqsubseteq A \quad \rho(U) \sqsubseteq T_i$ |
| Mandatory and functional attributes | $A \sqsubseteq \delta(U)$ (**funct** $U$) |
| Identification constraints | (**id** $A\ P, \ldots, P'^-, \ldots, U, \ldots$) |

# Properties of *DL-Lite*

- The TBox may contain **cyclic dependencies** (which typically increase the computational complexity of reasoning).

  Example: $A \sqsubseteq \exists P, \quad \exists P^- \sqsubseteq A$

- In the syntax, we have not included $\sqcap$ on the right hand-side of inclusion assertions, but it can trivially be added, since

$$B \sqsubseteq C_1 \sqcap C_2 \quad \text{is equivalent to} \quad \begin{array}{l} B \sqsubseteq C_1 \\ B \sqsubseteq C_2 \end{array}$$

- A domain assertion on role $P$ has the form: $\exists P \sqsubseteq A_1$
  A range assertion on role $P$ has the form: $\exists P^- \sqsubseteq A_2$

# Properties of *DL-Lite*$_{\mathcal{F}}$

*DL-Lite*$_{\mathcal{F}}$ does **not** enjoy the **finite model property**.

**Example**

TBox $\mathcal{T}$: $\mathsf{Nat} \sqsubseteq \exists \mathsf{succ}$ $\quad \exists \mathsf{succ}^- \sqsubseteq \mathsf{Nat}$

$\mathsf{Zero} \sqsubseteq \mathsf{Nat} \sqcap \neg \exists \mathsf{succ}^-$ $\quad (\mathbf{funct}\ \mathsf{succ}^-)$

ABox $\mathcal{A}$: $\mathsf{Zero}(0)$

$\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ admits only infinite models.
Hence, it is satisfiable, but **not finitely satisfiable**.

Hence, reasoning w.r.t. arbitrary models is different from reasoning w.r.t. finite models only.

# Properties of *DL-Lite*$_{\mathcal{R}}$

- *DL-Lite*$_{\mathcal{R}}$ **does enjoy the finite model property**. Hence, reasoning w.r.t. finite models is the same as reasoning w.r.t. arbitrary models.

- With role inclusion assertions, we can simulate **qualified existential quantification** in the rhs of an inclusion assertion $A_1 \sqsubseteq \exists Q.A_2$.

  To do so, we introduce a new role $Q_{A_2}$ and:

  - the role inclusion assertion $Q_{A_2} \sqsubseteq Q$
  - the concept inclusion assertions: $A_1 \sqsubseteq \exists Q_{A_2}$, $\exists Q^-_{A_2} \sqsubseteq A_2$

  In this way, we can consider $\exists Q.A$ in the right-hand side of an inclusion assertion as an abbreviation.

# Observations on *DL-Lite*$_{\mathcal{A}}$

- Captures all the basic constructs of **UML Class Diagrams** and of the **ER Model** ...
- ... **except covering constraints** in generalizations.
- Extends (the DL fragment of) the ontology language **RDFS**.
- Is completely symmetric w.r.t. **direct and inverse properties**.
- Is at the basis of the **OWL2 QL** profile of OWL2.

# The OWL2 QL Profile

OWL2 defines three **profiles**: OWL2 QL, OWL2 EL, OWL2 RL.

- Each profile corresponds to a syntactic fragment (i.e., a sub-language) of OWL2 DL that is targeted towards a specific use.
- The restrictions in each profile guarantee better computational properties than those of OWL2 DL.

The **OWL2 QL** profile is derived from the DLs of the *DL-Lite* family:

- "[It] includes most of the main features of conceptual models such as UML class diagrams and ER diagrams."
- "[It] is aimed at applications that use very large volumes of instance data, and where query answering is the most important reasoning task. In OWL2 QL, conjunctive query answering can be implemented using conventional relational database systems."

# Complexity of reasoning in *DL-Lite*$_{\mathcal{A}}$

1. We have seen that *DL-Lite*$_{\mathcal{A}}$ can capture the essential features of prominent conceptual modeling formalisms.
2. In the following, we will analyze reasoning in *DL-Lite*, and establish the following characterization of its computational properties:
   - Ontology satisfiability and all classical DL reasoning tasks are:
     - Efficiently tractable in the size of the TBox (i.e., PTIME).
     - Very efficiently tractable in the size of the ABox (i.e., $\mathrm{AC}^0$).
   - Query answering for CQs and UCQs is:
     - PTIME in the size of the TBox.
     - $\mathrm{AC}^0$ in the size of the ABox.
     - Exponential in the size of the **query** (NP-complete).
       Bad? . . . not really, this is exactly as in relational DBs.
3. We will also see that *DL-Lite* is essentially the maximal DL enjoying these nice computational properties.

From (1), (2), and (3) we get that:

*DL-Lite* is a representation formalism that is very well suited to underlie ontology-based data management systems.

# Part III

# Reasoning in the *DL-Lite* family

# Outline of Part 3

# Outline of Part 3

# Outline of Part 3

# Remarks

In the following, we make some simplifying assumptions:

- We ignore the distinction between objects and values, since it is not relevant for reasoning. Hence we do not use value domains and attributes.
- We do not consider identification constraints.

Notation:

- When the distinction between *DL-Lite*$_{\mathcal{R}}$, *DL-Lite*$_{\mathcal{F}}$, or *DL-Lite*$_{\mathcal{A}}$ is not important, we use just the *DL-Lite*.
- $Q$ denotes a basic role, i.e., $Q \longrightarrow P \mid P^-$.
- $R$ denotes a general role, i.e., $R \longrightarrow Q \mid \neg Q$.
- $C$ denotes a general concept, i.e., $C \longrightarrow A \mid \neg A \mid \exists Q \mid \neg \exists Q$, where $A$ is an atomic concept.

# TBox Reasoning services

- **Concept Satisfiability:** $C$ is satisfiable wrt $\mathcal{T}$, if there is a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}}$ is not empty, i.e., $\mathcal{T} \not\models C \equiv \bot$
- **Subsumption:** $C_1$ is subsumed by $C_2$ wrt $\mathcal{T}$, if for every model $\mathcal{I}$ of $\mathcal{T}$ we have $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$, i.e., $\mathcal{T} \models C_1 \sqsubseteq C_2$.
- **Equivalence:** $C_1$ and $C_2$ are equivalent wrt $\mathcal{T}$ if for every model $\mathcal{I}$ of $\mathcal{T}$ we have $C_1^{\mathcal{I}} = C_2^{\mathcal{I}}$, i.e., $\mathcal{T} \models C_1 \equiv C_2$.
- **Disjointness:** $C_1$ and $C_2$ are disjoint wrt $\mathcal{T}$ if for every model $\mathcal{I}$ of $\mathcal{T}$ we have $C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} = \emptyset$, i.e., $\mathcal{T} \models C_1 \sqcap C_2 \equiv \bot$
- **Functionality implication:** A functionality assertion $(\textbf{funct}\ Q)$ is logically implied by $\mathcal{T}$ if for every model $\mathcal{I}$ of $\mathcal{T}$, we have that $(o, o_1) \in Q^{\mathcal{I}}$ and $(o, o_2) \in Q^{\mathcal{I}}$ implies $o_1 = o_2$, i.e., $\mathcal{T} \models (\textbf{funct}\ Q)$.

*Analogous definitions hold for role satisfiability, subsumption, equivalence, and disjointness.*

# From TBox reasoning to ontology (un)satisfiability

Basic reasoning service:

- **Ontology satisfiability**: Verify whether an ontology $\mathcal{O}$ is satisfiable, i.e., whether $\mathcal{O}$ admits at least one model.

In the following, we show how to reduce TBox reasoning to ontology unsatisfiability:

1. We show how to reduce TBox reasoning services to concept/role subsumption.
2. We provide reductions from concept/role subsumption to ontology unsatisfiability.

# Outline of Part 3

# Concept/role satisfiability, equivalence, and disjointness

**Theorem**

1. $C$ is unsatisfiable wrt $\mathcal{T}$ iff $\mathcal{T} \models C \sqsubseteq \neg C$.
2. $\mathcal{T} \models C_1 \equiv C_2$ iff $\mathcal{T} \models C_1 \sqsubseteq C_2$ and $\mathcal{T} \models C_2 \sqsubseteq C_1$.
3. $C_1$ and $C_2$ are disjoint iff $\mathcal{T} \models C_1 \sqsubseteq \neg C_2$.

**Proof (sketch).**

1. "$\Leftarrow$" if $\mathcal{T} \models C \sqsubseteq \neg C$, then $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$, for every model $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ of $\mathcal{T}$; but this holds iff $C^{\mathcal{I}} = \emptyset$.

   "$\Rightarrow$" if $C$ is unsatisfiable, then $C^{\mathcal{I}} = \emptyset$, for every model $\mathcal{I}$ of $\mathcal{T}$. Therefore $C^{\mathcal{I}} \subseteq (\neg C)^{\mathcal{I}}$.
2. Trivial.
3. Trivial. □

*Analogous reductions for role satisfiability, equivalence and disjointness.*

# From implication of functionalities to subsumption

**Theorem**

$\mathcal{T} \models (\textbf{funct}\ Q)$ iff either

- $(\textbf{funct}\ Q) \in \mathcal{T}$ (only for *DL-Lite*$_{\mathcal{F}}$ or *DL-Lite*$_{\mathcal{A}}$), or
- $\mathcal{T} \models Q \sqsubseteq \neg Q$.

**Proof (sketch).**

"$\Leftarrow$" The case in which $(\textbf{funct}\ Q) \in \mathcal{T}$ is trivial.
Instead, if $\mathcal{T} \models Q \sqsubseteq \neg Q$, then $Q^{\mathcal{I}} = \emptyset$ and hence $\mathcal{I} \models (\textbf{funct}\ Q)$, for every model $\mathcal{I}$ of $\mathcal{T}$.

"$\Rightarrow$" When neither $(\textbf{funct}\ Q) \in \mathcal{T}$ nor $\mathcal{T} \models Q \sqsubseteq \neg Q$, we can construct a model of $\mathcal{T}$ that is not a model of $(\textbf{funct}\ Q)$. □

# Outline of Part 3

# From concept subsumption to ontology unsatisfiability

**Theorem**

$\mathcal{T} \models C_1 \sqsubseteq C_2$ iff the ontology $\mathcal{O}_{C_1 \sqsubseteq C_2} = \langle \mathcal{T} \cup \{\hat{A} \sqsubseteq C_1, \hat{A} \sqsubseteq \neg C_2\},\ \{\hat{A}(c)\}\rangle$ is unsatisfiable, where $\hat{A}$ is an atomic concept not in $\mathcal{T}$, and $c$ is a constant.

Intuitively, $C_1$ is subsumed by $C_2$ iff the smallest ontology containing $\mathcal{T}$ and implying both $C_1(c)$ and $\neg C_2(c)$ is unsatisfiable.

**Proof (sketch).**

"$\Leftarrow$" Let $\mathcal{O}_{C_1 \sqsubseteq C_2}$ be unsatisfiable, and suppose that $\mathcal{T} \not\models C_1 \sqsubseteq C_2$. Then there exists a model $\mathcal{I}$ of $\mathcal{T}$ such that $C_1^{\mathcal{I}} \not\subseteq C_2^{\mathcal{I}}$. Hence $C_1^{\mathcal{I}} \setminus C_2^{\mathcal{I}} \neq \emptyset$. We can extend $\mathcal{I}$ to a model of $\mathcal{O}_{C_1 \sqsubseteq C_2}$ by taking $c^{\mathcal{I}} = o$, for some $o \in C_1^{\mathcal{I}} \setminus C_2^{\mathcal{I}}$, and $\hat{A}^{\mathcal{I}} = \{c^{\mathcal{I}}\}$. This contradicts $\mathcal{O}_{C_1 \sqsubseteq C_2}$ being unsatisfiable.

"$\Rightarrow$" Let $\mathcal{T} \models C_1 \sqsubseteq C_2$, and suppose that $\mathcal{O}_{C_1 \sqsubseteq C_2}$ is satisfiable. Then there exists a model $\mathcal{I}$ be of $\mathcal{O}_{C_1 \sqsubseteq C_2}$. Then $\mathcal{I} \models \mathcal{T}$, and $\mathcal{I} \models C_1(c)$ and $\mathcal{I} \models \neg C_2(c)$, i.e., $\mathcal{I} \not\models C_1 \sqsubseteq C_2$. This contradicts $\mathcal{T} \models C_1 \sqsubseteq C_2$. □

# From role subsumption to ont. unsatisfiability

**Theorem**

Let $\mathcal{T}$ be a *DL-Lite*$_\mathcal{R}$ or *DL-Lite*$_\mathcal{A}$ TBox and $R_1$, $R_2$ two general roles.
Then $\mathcal{T} \models R_1 \sqsubseteq R_2$ iff the ontology
$\mathcal{O}_{R_1 \sqsubseteq R_2} = \langle \mathcal{T} \cup \{\hat{P} \sqsubseteq R_1, \hat{P} \sqsubseteq \neg R_2\},\ \{\hat{P}(c_1, c_2)\}\rangle$ is unsatisfiable,
where $\hat{P}$ is an atomic role not in $\mathcal{T}$, and $c_1$, $c_2$ are two constants.

Intuitively, $R_1$ is subsumed by $R_2$ iff the smallest ontology containing $\mathcal{T}$ and implying both $R_1(c_1, c_2)$ and $\neg R_2(c_1, c_2)$ is unsatisfiable.

**Proof (sketch).**

Analogous to the one for concept subsumption. □

Notice that $\mathcal{O}_{R_1 \sqsubseteq R_2}$ *is inherently a DL-Lite*$_\mathcal{R}$ *ontology.*

# From role subsumption to ont. unsatisfiability for *DL-Lite*$_{\mathcal{F}}$

**Theorem**

Let $\mathcal{T}$ be a *DL-Lite*$_{\mathcal{F}}$ TBox, and $Q_1$, $Q_2$ two basic roles such that $Q_1 \neq Q_2$. Then,

1. $\mathcal{T} \models Q_1 \sqsubseteq Q_2$ iff $Q_1$ is unsatisfiable iff
   either $\exists Q_1$ or $\exists Q_1^-$ is unsatisfiable wrt $\mathcal{T}$,
   which can again be reduced to ontology unsatisfiability.
2. $\mathcal{T} \models \neg Q_1 \sqsubseteq Q_2$ iff $\mathcal{T}$ is unsatisfiable.
3. $\mathcal{T} \models Q_1 \sqsubseteq \neg Q_2$ iff
   either $\exists Q_1$ and $\exists Q_2$ are disjoint, or $\exists Q_1^-$ and $\exists Q_2^-$ are disjoint, iff
   either $\mathcal{T} \models \exists Q_1 \sqsubseteq \neg \exists Q_2$, or $\mathcal{T} \models \exists Q_1^- \sqsubseteq \neg \exists Q_2^-$,
   which can again be reduced to ontology unsatisfiability.

Notice that an inclusion of the form $\neg Q_1 \sqsubseteq \neg Q_2$ is equivalent to $Q_2 \sqsubseteq Q_1$, and therefore is considered in the first item.

# Summary

- The results above tell us that we can support TBox reasoning services by relying on the ontology (un)satisfiability service.
- Ontology satisfiability is a form of reasoning over both the TBox and the ABox of the ontology.
- In the following, we first consider other TBox & ABox reasoning services, in particular **query answering**, and then turn back to ontology satisfiability.

# Outline of Part 3

8 TBox reasoning

9 TBox & ABox reasoning
- TBox & ABox Reasoning services
- Query answering
- Query answering over satisfiable ontologies
- Ontology satisfiability
- Complexity of reasoning in *DL-Lite*

10 Beyond *DL-Lite*

# Outline of Part 3

# TBox and ABox reasoning services

- **Ontology Satisfiability:** Verify wether an ontology $\mathcal{O}$ is satisfiable, i.e., whether $\mathcal{O}$ admits at least one model.
- **Concept Instance Checking:** Verify wether an individual $c$ is an instance of a concept $C$ in an ontology $\mathcal{O}$, i.e., whether $\mathcal{O} \models C(c)$.
- **Role Instance Checking:** Verify wether a pair $(c_1, c_2)$ of individuals is an instance of a role $Q$ in an ontology $\mathcal{O}$, i.e., whether $\mathcal{O} \models Q(c_1, c_2)$.
- **Query Answering** Given a query $q$ over an ontology $\mathcal{O}$, find all tuples $\vec{c}$ of constants such that $\mathcal{O} \models q(\vec{c})$.

# Query answering and instance checking

For atomic concepts and roles, **instance checking is a special case of query answering**, in which the query is **boolean** and constituted by a single atom in the body.

- $\mathcal{O} \models A(c)$ iff $q() \leftarrow A(c)$ evaluated over $\mathcal{O}$ is true.
- $\mathcal{O} \models P(c_1, c_2)$ iff $q() \leftarrow P(c_1, c_2)$ evaluated over $\mathcal{O}$ is true.

# From instance checking to ontology unsatisfiability

**Theorem**

Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a **DL-Lite** ontology, $C$ a *DL-Lite* concept, and $P$ an atomic role. Then:

- $\mathcal{O} \models C(c)$ iff $\mathcal{O}_{C(c)} = \langle \mathcal{T} \cup \{\hat{A} \sqsubseteq \neg C\},\ \mathcal{A} \cup \{\hat{A}(c)\} \rangle$ is unsatisfiable, where $\hat{A}$ is an atomic concept not in $\mathcal{O}$.
- $\mathcal{O} \models \neg P(c_1, c_2)$ iff $\mathcal{O}_{\neg P(c_1,c_2)} = \langle \mathcal{T},\ \mathcal{A} \cup \{P(c_1, c_2)\} \rangle$ is unsatisfiable.

**Theorem**

Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a **DL-Lite**$_{\mathcal{F}}$ ontology and $P$ an atomic role.
Then $\mathcal{O} \models P(c_1, c_2)$ iff $\mathcal{O}$ is unsatisfiable or $P(c_1, c_2) \in \mathcal{A}$.

**Theorem**

Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a **DL-Lite**$_{\mathcal{R}}$ ontology and $P$ an atomic role.
Then $\mathcal{O} \models P(c_1, c_2)$ iff $\mathcal{O}_{P(c_1,c_2)} = \langle \mathcal{T} \cup \{\hat{P} \sqsubseteq \neg P\},\ \mathcal{A} \cup \{\hat{P}(c_1, c_2)\} \rangle$ is unsatisfiable, where $\hat{P}$ is an atomic role not in $\mathcal{O}$.

# Outline of Part 3

# Certain answers

We recall that

Query answering over an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is a form of **logical implication**:

find all tuples $\vec{c}$ of constants of $\mathcal{A}$ s.t. $\mathcal{O} \models q(\vec{c})$

A.k.a. **certain answers** in databases, i.e., the tuples that are answers to $q$ in **all** models of $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$:

$$cert(q, \mathcal{O}) \;=\; \{\; \vec{c} \;\mid\; \vec{c} \in q^{\mathcal{I}}, \quad \text{for every model } \mathcal{I} \text{ of } \mathcal{O} \;\}$$

*Note:* We have assumed that the answer $q^{\mathcal{I}}$ to a query $q$ over an interpretation $\mathcal{I}$ is constituted by a set of tuples of **constants** of $\mathcal{A}$, rather than objects in $\Delta^{\mathcal{I}}$.

# $\mathcal{Q}$-rewritability for *DL-Lite*

- We now study rewritability of query answering over *DL-Lite* ontologies.
- In particular we will show that *DL-Lite*$_{\mathcal{A}}$ (and hence *DL-Lite*$_{\mathcal{F}}$ and *DL-Lite*$_{\mathcal{R}}$) enjoy FOL-rewritability of answering union of conjunctive queries.

# Query answering vs. ontology satisfiability

- In the case in which an ontology is unsatisfiable, according to the "ex falso quod libet" principle, reasoning is trivialized.
- In particular, query answering is meaningless, since every tuple is in the answer to every query.
- We are not interested in encoding meaningless query answering into the perfect reformulation of the input query. Therefore, before query answering, we will always check ontology satisfiability to single out meaningful cases.

Thus, we proceed as follows:

1. We show how to do **query answering over satisfiable ontologies**.
2. We show how we can exploit the query answering algorithm also to check ontology satisfiability.

# Remark

We call positive inclusions (PIs) assertions of the form

$$\begin{array}{lcl} Cl & \sqsubseteq & A \mid \exists Q \\ Q_1 & \sqsubseteq & Q_2 \end{array}$$

We call negative inclusions (NIs) assertions of the form

$$\begin{array}{lcl} Cl & \sqsubseteq & \neg A \mid \neg \exists Q \\ Q_1 & \sqsubseteq & \neg Q_2 \end{array}$$

# Outline of Part 3

# Query answering over satisfiable ontologies

Given a CQ $q$ and a satisfiable ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, we compute $cert(q, \mathcal{O})$ as follows:

1. Using $\mathcal{T}$, **rewrite** $q$ into a UCQ $r_{q,\mathcal{T}}$ (the perfect rewriting of $q$ w.r.t. $\mathcal{T}$).
2. Encode $r_{q,\mathcal{T}}$ into SQL and **evaluate** it over $\mathcal{A}$ managed in secondary storage **via a RDBMS**, to return $cert(q, \mathcal{O})$.

Correctness of this procedure shows FOL-rewritability of query answering in *DL-Lite*.

$\rightsquigarrow$ Query answering over *DL-Lite* ontologies can be done using RDMBS technology.

# Query rewriting

Consider the query $\mathsf{q}(x) \leftarrow \mathsf{Professor}(x)$

**Intuition:** Use the PIs as basic rewriting rules:

$$\mathsf{AssistantProf} \sqsubseteq \mathsf{Professor}$$

as a logic rule: $\mathsf{Professor}(z) \leftarrow \mathsf{AssistantProf}(z)$

**Basic rewriting step:**

when an atom in the query unifies with the **head** of the rule,

substitute the atom with the **body** of the rule.

We say that the PI inclusion **applies to** the atom.

In the example, the PI $\mathsf{AssistantProf} \sqsubseteq \mathsf{Professor}$ applies to the atom $\mathsf{Professor}(x)$. Towards the computation of the perfect rewriting, we add to the input query above, the query

$$\mathsf{q}(x) \leftarrow \mathsf{AssistantProf}(x)$$

# Query rewriting (cont'd)

Consider the query $\mathsf{q}(x) \leftarrow \mathsf{teaches}(x, y), \mathsf{Course}(y)$

and the PI $\exists \mathsf{teaches}^- \sqsubseteq \mathsf{Course}$

as a logic rule: $\mathsf{Course}(z_2) \leftarrow \mathsf{teaches}(z_1, z_2)$

The PI applies to the atom $\mathsf{Course}(y)$, and we add to the perfect rewriting the query

$$\mathsf{q}(x) \leftarrow \mathsf{teaches}(x, y), \mathsf{teaches}(z_1, y)$$

Consider now the query $\mathsf{q}(x) \leftarrow \mathsf{teaches}(x, y)$

and the PI $\mathsf{Professor} \sqsubseteq \exists \mathsf{teaches}$

as a logic rule: $\mathsf{teaches}(z, f(z)) \leftarrow \mathsf{Professor}(z)$

The PI applies to the atom $\mathsf{teaches}(x, y)$, and we add to the perfect rewriting the query

$$\mathsf{q}(x) \leftarrow \mathsf{Professor}(x)$$

# Query rewriting – Constants

Conversely, for the query $\mathsf{q}(x) \leftarrow \mathsf{teaches}(x, \mathtt{databases})$

and the same PI as before $\mathsf{Professor} \sqsubseteq \exists \mathsf{teaches}$

as a logic rule: $\mathsf{teaches}(z, f(z)) \leftarrow \mathsf{Professor}(z)$

$\mathsf{teaches}(x, \mathtt{databases})$ does not unify with $\mathsf{teaches}(z, f(z))$, since the **skolem term** $f(z)$ in the head of the rule **does not unify** with the constant $\mathtt{databases}$.

In this case, the PI **does not apply** to the atom $\mathsf{teaches}(x, \mathtt{databases})$.

The same holds for the following query, where $y$ is **distinguished**, since unifying $f(z)$ with $y$ would correspond to returning a skolem term as answer to the query:

$$\mathsf{q}(x, y) \leftarrow \mathsf{teaches}(x, y)$$

# Query rewriting – Join variables

An analogous behavior to the one with constants and with distinguished variables holds when the atom contains **join variables** that would have to be unified with skolem terms.

Consider the query $\mathsf{q}(x) \leftarrow \mathsf{teaches}(x, y), \mathsf{Course}(y)$

and the PI $\mathsf{Professor} \sqsubseteq \exists \mathsf{teaches}$

as a logic rule: $\mathsf{teaches}(z, f(z)) \leftarrow \mathsf{Professor}(z)$

The PI above does **not** apply to the atom $\mathsf{teaches}(x, y)$.

# Query rewriting – Reduce step

Consider now the query $\mathsf{q}(x) \leftarrow \mathsf{teaches}(x, y), \mathsf{teaches}(z, y)$

and the PI $\mathsf{Professor} \sqsubseteq \exists \mathsf{teaches}$

as a logic rule: $\mathsf{teaches}(z, f(z)) \leftarrow \mathsf{Professor}(z)$

This PI does not apply to $\mathsf{teaches}(x, y)$ or $\mathsf{teaches}(z, y)$, since $y$ is in join, and we would again introduce the skolem term in the rewritten query.

However, we can transform the above query by unifying the atoms $\mathsf{teaches}(x, y)$ and $\mathsf{teaches}(z, y)$. This rewriting step is called **reduce**, and produces the query

$$\mathsf{q}(x) \leftarrow \mathsf{teaches}(x, y)$$

Now, we can apply the PI above, and add to the rewriting the query

$$\mathsf{q}(x) \leftarrow \mathsf{Professor}(x)$$

# Query rewriting – Summary

Reformulate the CQ $q$ into a set of queries: apply to $q$ and the computed queries in all possible ways the PIs in the TBox $\mathcal{T}$:

$$
\begin{array}{rlcl}
A_1 \sqsubseteq A_2 & \ldots, A_2(x), \ldots & \rightsquigarrow & \ldots, A_1(x), \ldots \\
\exists P \sqsubseteq A & \ldots, A(x), \ldots & \rightsquigarrow & \ldots, P(x, \_), \ldots \\
\exists P^- \sqsubseteq A & \ldots, A(x), \ldots & \rightsquigarrow & \ldots, P(\_, x), \ldots \\
A \sqsubseteq \exists P & \ldots, P(x, \_), \ldots & \rightsquigarrow & \ldots, A(x), \ldots \\
A \sqsubseteq \exists P^- & \ldots, P(\_, x), \ldots & \rightsquigarrow & \ldots, A(x), \ldots \\
\exists P_1 \sqsubseteq \exists P_2 & \ldots, P_2(x, \_), \ldots & \rightsquigarrow & \ldots, P_1(x, \_), \ldots \\
P_1 \sqsubseteq P_2 & \ldots, P_2(x, y), \ldots & \rightsquigarrow & \ldots, P_1(x, y), \ldots \\
\ldots & & &
\end{array}
$$

(_ denotes an **unbound** variable, i.e., a variable that appears only once)

This corresponds to exploiting ISAs, role typing, and mandatory participation to obtain new queries that could contribute to the answer.

Unifying atoms can make rules applicable that were not so before, and is required for completeness of the method.

The UCQ resulting from this process is the **perfect rewriting** $r_{q,\mathcal{T}}$.

# Query rewriting algorithm

**Algorithm** *PerfectRef*$(q, \mathcal{T}_P)$
**Input:** conjunctive query $q$, set of *DL-Lite*$_\mathcal{R}$ PIs $\mathcal{T}_P$
**Output:** union of conjunctive queries $PR$
$PR := \{q\}$;
**repeat**
  $PR' := PR$;
  **for each** $q \in PR'$ **do**
    **for each** $g$ in $q$ **do**
      **for each** PI $I$ in $\mathcal{T}_P$ **do**
        **if** $I$ is applicable to $g$
        **then** $PR := PR \cup \{\, q[g/(g, I)] \,\}$
    **for each** $g_1, g_2$ in $q$ **do**
      **if** $g_1$ and $g_2$ unify
      **then** $PR := PR \cup \{\tau(\mathit{Reduce}(q, g_1, g_2))\}$;
**until** $PR' = PR$;
**return** $PR$

**Notice that NIs or functionality assertions do not play any role in the rewriting of the query.**

# Query answering in *DL-Lite* – An interesting example

TBox: $\text{Person} \sqsubseteq \exists \text{hasFather}$
$\exists \text{hasFather}^- \sqsubseteq \text{Person}$

ABox: $\text{Person}(\texttt{mary})$

Query: $\text{q}(x) \leftarrow \text{Person}(x), \text{hasFather}(x, y_1), \text{hasFather}(y_1, y_2), \text{hasFather}(y_2, y_3)$

$\text{q}(x) \leftarrow \text{Person}(x), \text{hasFather}(x, y_1), \text{hasFather}(y_1, y_2), \text{hasFather}(y_2, \_)$

$\Downarrow$ **Apply** $\text{Person} \sqsubseteq \exists \text{hasFather}$ to the atom $\text{hasFather}(y_2, \_)$

$\text{q}(x) \leftarrow \text{Person}(x), \text{hasFather}(x, y_1), \text{hasFather}(y_1, y_2), \text{Person}(y_2)$

$\Downarrow$ **Apply** $\exists \text{hasFather}^- \sqsubseteq \text{Person}$ to the atom $\text{Person}(y_2)$

$\text{q}(x) \leftarrow \text{Person}(x), \text{hasFather}(x, y_1), \text{hasFather}(y_1, y_2), \text{hasFather}(\_, y_2)$

$\Downarrow$ **Unify** atoms $\text{hasFather}(y_1, y_2)$ and $\text{hasFather}(\_, y_2)$

$\text{q}(x) \leftarrow \text{Person}(x), \text{hasFather}(x, y_1), \text{hasFather}(y_1, y_2)$

$\Downarrow$

$\ldots$

$\text{q}(x) \leftarrow \text{Person}(x), \text{hasFather}(x, \_)$

$\Downarrow$ **Apply** $\text{Person} \sqsubseteq \exists \text{hasFather}$ to the atom $\text{hasFather}(x, \_)$

$\text{q}(x) \leftarrow \text{Person}(x)$

# ABox storage

**ABox** $\mathcal{A}$ stored as a **relational database** in a standard RDBMS as follows:

- For each **atomic concept** $A$ used in the ABox:
  - define a **unary relational table** $\texttt{tab}_A$
  - populate $\texttt{tab}_A$ with each $\langle c \rangle$ such that $A(c) \in \mathcal{A}$
- For each atomic role $P$ used in the ABox,
  - define a binary relational table $\texttt{tab}_P$
  - populate $\texttt{tab}_P$ with each $\langle c_1, c_2 \rangle$ such that $P(c_1, c_2) \in \mathcal{A}$

We denote with $\mathbf{DB}(\mathcal{A})$ the database obtained as above.

# Query evaluation

Let $r_{q,\mathcal{T}}$ be the UCQ returned by the algorithm *PerfectRef*$(q, \mathcal{T})$.

- We denote by **SQL**$(r_{q,\mathcal{T}})$ the encoding of $r_{q,\mathcal{T}}$ into an SQL query over $DB(\mathcal{A})$.
- We indicate with **Eval(SQL**$(r_{q,\mathcal{T}})$**, DB**$(\mathcal{A})$**)** the evaluation of $SQL(r_{q,\mathcal{T}})$ over $DB(\mathcal{A})$.

# Query answering in *DL-Lite*

**Theorem**

Let $\mathcal{T}$ be a *DL-Lite* TBox, $\mathcal{T}_P$ the set of PIs in $\mathcal{T}$, $q$ a CQ over $\mathcal{T}$, and let $r_{q,\mathcal{T}} = \mathit{PerfectRef}(q, \mathcal{T}_P)$. Then, for each ABox $\mathcal{A}$ such that $\langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable, we have that

$$\mathit{cert}(q, \langle \mathcal{T}, \mathcal{A} \rangle) = \mathit{Eval}(\mathit{SQL}(r_{q,\mathcal{T}}), \mathit{DB}(\mathcal{A})).$$

In other words, query answering over a satisfiable *DL-Lite* ontology is FOL-rewritable.

Notice that we did not mention NIs or functionality assertions of $\mathcal{T}$ in the result above. Indeed, **when the ontology is satisfiable, we can ignore NIs and functionalities and answer queries as if they were not specified in $\mathcal{T}$.**

# Query answering in *DL-Lite* – Example

TBox: $\mathsf{Professor} \sqsubseteq \exists \mathsf{teaches}$
$\exists \mathsf{teaches}^- \sqsubseteq \mathsf{Course}$

Query: $\mathsf{q}(x) \leftarrow \mathsf{teaches}(x, y), \mathsf{Course}(y)$

Perfect Rewriting: $\mathsf{q}(x) \leftarrow \mathsf{teaches}(x, y), \mathsf{Course}(y)$
$\mathsf{q}(x) \leftarrow \mathsf{teaches}(x, y), \mathsf{teaches}(\_, y)$
$\mathsf{q}(x) \leftarrow \mathsf{teaches}(x, \_)$
$\mathsf{q}(x) \leftarrow \mathsf{Professor}(x)$

ABox: $\mathsf{teaches}(\mathtt{john}, \mathtt{databases})$
$\mathsf{Professor}(\mathtt{mary})$

It is easy to see that $\mathit{Eval}(\mathit{SQL}(r_{q,\mathcal{T}}), \mathit{DB}(\mathcal{A}))$ in this case produces as answer $\{\mathtt{john}, \mathtt{mary}\}$.

# Outline of Part 3

# Satisfiability of ontologies with only PIs

Let us now consider the problem of establishing whether an ontology is satisfiable.

A first notable result tells us that PIs alone cannot generate ontology unsatisfiability.

**Theorem**

Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a *DL-Lite* ontology where $\mathcal{T}$ contains **only PIs**.
Then, $\mathcal{O}$ is satisfiable.

# Satisfiability of *DL-Lite*$_\mathcal{R}$ ontologies

Unsatisfiability in *DL-Lite*$_\mathcal{R}$ ontology, however can be caused by NIs.

**Example**

TBox $\mathcal{T}$: $\text{Professor} \sqsubseteq \neg\text{Student}$
$\exists\text{teaches} \sqsubseteq \text{Professor}$

ABox $\mathcal{A}$: $\text{teaches}(\texttt{john}, \texttt{databases})$
$\text{Student}(\texttt{john})$

# Checking satisfiability of *DL-Lite*$_\mathcal{R}$ ontologies

Satisfiability of a *DL-Lite*$_\mathcal{R}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is reduced to evaluating a FOL-query (in fact a UCQ) over $DB(\mathcal{A})$.

We proceed as follows: Let $\mathcal{T}_P$ the set of PIs in $\mathcal{T}$.

1. For each NI $N$ between concepts (resp. roles) in $\mathcal{T}$, we ask $\langle \mathcal{T}_P, \mathcal{A} \rangle$ whether there exists some individual (resp. pair of individuals) that contradicts $N$, i.e., we construct over $\langle \mathcal{T}_P, \mathcal{A} \rangle$ a **boolean CQ** $q_N()$ such that
$$\langle \mathcal{T}_P, \mathcal{A} \rangle \models q_N() \quad \text{iff} \quad \langle \mathcal{T}_P \cup \{N\}, \mathcal{A} \rangle \text{ is unsatisfiable}$$
2. We exploit *PerfectRef* to verify whether $\langle \mathcal{T}_P, \mathcal{A} \rangle \models q_N()$, i.e., we compute *PerfectRef*$(q_N, \mathcal{T}_P)$, and evaluate it (in fact, its SQL encoding) over $DB(\mathcal{A})$.

# Satisfiability of *DL-Lite*$_\mathcal{R}$ ontologies – Example

PIs $\mathcal{T}_P$: $\exists \mathsf{teaches} \sqsubseteq \mathsf{Professor}$

NI $N$: $\mathbf{Professor} \sqsubseteq \neg \mathbf{Student}$

Query $q_N$: $q_N() \leftarrow \mathsf{Student}(x), \mathsf{Professor}(x)$

Perfect Rewriting: $q_N() \leftarrow \mathsf{Student}(x), \mathsf{Professor}(x)$
$q_N() \leftarrow \mathsf{Student}(x), \mathsf{teaches}(x, \_)$

ABox $\mathcal{A}$: $\mathsf{teaches}(\mathtt{john}, \mathtt{databases})$
$\mathsf{Student}(\mathtt{john})$

It is easy to see that $\langle \mathcal{T}_P, \mathcal{A} \rangle \models q_N()$, and that the ontology $\langle \mathcal{T}_P \cup \{\mathsf{Professor} \sqsubseteq \neg \mathsf{Student}\},\ \mathcal{A} \rangle$ **is unsatisfiable**.

# Queries for NIs

For each **NI** $N$ in $\mathcal{T}$ we compute a boolean CQ $q_N()$ according to the following rules:

$$
\begin{array}{rcl}
A_1 \sqsubseteq \neg A_2 & \rightsquigarrow & q_N() \leftarrow A_1(x), A_2(x) \\
\exists P \sqsubseteq \neg A \text{ or } A \sqsubseteq \neg \exists P & \rightsquigarrow & q_N() \leftarrow P(x,y), A(x) \\
\exists P^- \sqsubseteq \neg A \text{ or } A \sqsubseteq \neg \exists P^- & \rightsquigarrow & q_N() \leftarrow P(y,x), A(x) \\
\exists P_1 \sqsubseteq \neg \exists P_2 & \rightsquigarrow & q_N() \leftarrow P_1(x,y), P_2(x,z) \\
\exists P_1 \sqsubseteq \neg \exists P_2^- & \rightsquigarrow & q_N() \leftarrow P_1(x,y), P_2(z,x) \\
\exists P_1^- \sqsubseteq \neg \exists P_2 & \rightsquigarrow & q_N() \leftarrow P_1(x,y), P_2(y,z) \\
\exists P_1^- \sqsubseteq \neg \exists P_2^- & \rightsquigarrow & q_N() \leftarrow P_1(x,y), P_2(z,y) \\
P_1 \sqsubseteq \neg P_2 \text{ or } P_1^- \sqsubseteq \neg P_2^- & \rightsquigarrow & q_N() \leftarrow P_1(x,y), P_2(x,y) \\
P_1^- \sqsubseteq \neg P_2 \text{ or } P_1 \sqsubseteq \neg P_2^- & \rightsquigarrow & q_N() \leftarrow P_1(x,y), P_2(y,x)
\end{array}
$$

# From satisfiability to query answering in *DL-Lite*$_\mathcal{R}$

**Lemma (Separation for *DL-Lite*$_\mathcal{R}$)**

Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a *DL-Lite*$_\mathcal{R}$ ontology, and $\mathcal{T}_P$ the set of PIs in $\mathcal{T}$. Then, $\mathcal{O}$ **is unsatisfiable iff there exists a NI** $N \in \mathcal{T}$ **such that** $\langle \mathcal{T}_P, \mathcal{A} \rangle \models q_N()$.

The lemma relies on the following properties:

- NIs do not interact with each other.
- Interaction between NIs and PIs can be managed through *PerfectRef*.

Notably, each NI can be processed individually.

# FOL-rewritability of satisfiability in *DL-Lite*$_\mathcal{R}$

From the previous lemma and the theorem on query answering for satisfiable *DL-Lite*$_\mathcal{R}$ ontologies, we get the following result.

**Theorem**

Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a *DL-Lite*$_\mathcal{R}$ ontology, and $\mathcal{T}_P$ the set of PIs in $\mathcal{T}$. Then, **$\mathcal{O}$ is unsatisfiable iff there exists a NI $N \in \mathcal{T}$ such that $\mathbf{Eval}(\mathbf{SQL}(\mathbf{PerfectRef}(q_N, \mathcal{T}_P)), \mathbf{DB}(\mathcal{A}))$ returns true.**

In other words, satisfiability of a *DL-Lite*$_\mathcal{R}$ ontology can be reduced to FOL-query evaluation.

# Satisfiability of *DL-Lite*$_\mathcal{A}$ ontologies

Unsatisfiability in *DL-Lite*$_\mathcal{A}$ (and *DL-Lite*$_\mathcal{F}$) ontologies can be caused by **NIs** or by **functionality assertions**.

**Example**

TBox $\mathcal{T}$: $\mathsf{Professor} \sqsubseteq \neg \mathsf{Student}$
$\exists \mathsf{teaches} \sqsubseteq \mathsf{Professor}$
$(\mathbf{funct}\ \mathsf{teaches}^-)$

ABox $\mathcal{A}$: $\mathsf{Student}(\mathtt{john})$
$\mathsf{teaches}(\mathtt{john}, \mathtt{databases})$
$\mathsf{teaches}(\mathtt{michael}, \mathtt{databases})$

# Checking satisfiability of *DL-Lite*$_\mathcal{A}$ ontologies

Satisfiability of a *DL-Lite*$_\mathcal{A}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is reduced to evaluating a FOL-query over $DB(\mathcal{A})$.

**We deal with NIs exactly as done in DL-Lite$_\mathcal{R}$ ontologies** (indeed, limited to NIs, *DL-Lite*$_\mathcal{A}$ ontologies are just *DL-Lite*$_\mathcal{R}$ ontologies).

To deal with **functionality assertions**, we proceed as follows:

1. For each functionality assertion $F \in \mathcal{T}$, we ask if there exist two pairs of individuals in $\mathcal{A}$ that contradict $F$, i.e., we pose over $\mathcal{A}$ a boolean FOL query $q_F()$ such that

$$\mathcal{A} \models q_F() \quad \text{iff} \quad \langle \{F\}, \mathcal{A} \rangle \text{ is unsatisfiable.}$$

2. To verify if $\mathcal{A} \models q_F()$, we evaluate $SQL(q_F)$ over $DB(\mathcal{A})$.

# Queries for functionality assertions

For each **functionality assertion** $F$ in $\mathcal{T}$ we compute a boolean FOL query $q_F()$ according to the following rules:

$$\begin{array}{lcl} (\textbf{funct } P) & \rightsquigarrow & q_F() \leftarrow P(x,y), P(x,z), y \neq z \\ (\textbf{funct } P^-) & \rightsquigarrow & q_F() \leftarrow P(x,y), P(z,y), x \neq z \end{array}$$

## Example

Functionality $F$: $(\textbf{funct teaches}^-)$

Query $q_F$: $q_F() \leftarrow \textbf{teaches}(x,y), \textbf{teaches}(z,y), x \neq z$

ABox $\mathcal{A}$: $\texttt{teaches}(\texttt{john}, \texttt{databases})$
$\texttt{teaches}(\texttt{michael}, \texttt{databases})$

It is easy to see that $\mathcal{A} \models q_F()$, and that $\langle \{(\textbf{funct teaches}^-)\}, \mathcal{A} \rangle$, **is unsatisfiable**.

# From satisfiability to query answering in *DL-Lite*$_\mathcal{A}$

**Lemma (Separation for *DL-Lite*$_\mathcal{A}$)**

Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a *DL-Lite*$_\mathcal{A}$ ontology, and $\mathcal{T}_P$ the set of PIs in $\mathcal{T}$. Then, $\mathcal{O}$ is unsatisfiable iff one of the following condition holds:

- $(a)$ There exists a NI $N \in \mathcal{T}$ such that $\langle \mathcal{T}_P, \mathcal{A} \rangle \models q_N()$.
- $(b)$ There exists a functionality assertion $F \in \mathcal{T}$ such that $\mathcal{A} \models q_F()$.

(a) relies on the properties that NIs do not interact with each other, and interaction between NIs and PIs can be managed through *PerfectRef*.

(b) exploits the property that **NIs and PIs do not interact with functionalities**: indeed, no functionality assertions are contradicted in a *DL-Lite*$_\mathcal{A}$ ontology $\mathcal{O}$, beyond those explicitly contradicted by the ABox.

Notably, the lemma asserts that to check ontology satisfiability, each NI and each functionality assertion can be processed individually.

# FOL-rewritability of satisfiability in *DL-Lite*$_\mathcal{A}$

From the previous lemma and the theorem on query answering for satisfiable *DL-Lite*$_\mathcal{A}$ ontologies, we get the following result.

**Theorem**

Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a *DL-Lite*$_\mathcal{A}$ ontology, and $\mathcal{T}_P$ the set of PIs in $\mathcal{T}$. Then, $\mathcal{O}$ is unsatisfiable iff one of the following condition holds:

- $(a)$ There exists a NI $N \in \mathcal{T}$ such that *Eval*(*SQL*(*PerfectRef*$(q_N, \mathcal{T}_P)$), *DB*$(\mathcal{A})$) returns *true*.
- $(b)$ There exists a functionality assertion $F \in \mathcal{T}$ such that *Eval*(*SQL*$(q_F)$, *DB*$(\mathcal{A})$) returns *true*.

In other words, satisfiability of a *DL-Lite*$_\mathcal{A}$ ontology can be reduced to FOL-query evaluation.

# Outline of Part 3

# Complexity of query answering over satisfiable ontologies

## Theorem

**Query answering** over *DL-Lite*$_{\mathcal{A}}$ ontologies is

1. NP-**complete** in the size of **query and ontology** (combined comp.).
2. PTIME in the size of the **ontology**.
3. $\mathrm{AC}^0$ in the size of the **ABox** (data complexity).

## Proof (sketch).

1. **Guess** together the derivation of one of the CQs of the perfect rewriting, and an assignment to its existential variables. Checking the derivation and evaluating the guessed CQ over the ABox is then polynomial in combined complexity. NP-hardness follows from combined complexity of evaluating CQs over a database.
2. The number of CQs in the perfect rewriting is polynomial in the size of the TBox, and we can compute them in PTIME.
3. Is the data complexity of evaluating FOL queries over a DB. □

# Complexity of ontology satisfiability

**Theorem**

Checking satisfiability of *DL-Lite*$_{\mathcal{A}}$ ontologies is

1. PTIME in the size of the **ontology** (combined complexity).
2. LOGSPACE in the size of the **ABox** (data complexity).

**Proof (sketch).**

Follows directly from the algorithm for ontology satisfiability and the complexity of query answering over satisfiable ontologies. □

# Complexity of TBox reasoning

**Theorem**

**TBox reasoning** over *DL-Lite*$_{\mathcal{A}}$ ontologies is PTIME in the size of the **TBox** (schema complexity).

**Proof (sketch).**

Follows from the previous theorem, and from the reduction of TBox reasoning to ontology satisfiability. Indeed, the size of the ontology constructed in the reduction is polynomial in the size of the input TBox. □

# Outline of Part 3

# Outline of Part 3

# Beyond *DL-Lite*

We consider now DL languages that **extend DL-Lite with additional DL constructs** or with combinations of constructs that are not legal in *DL-Lite*.

We show that (essentially) all such extensions of *DL-Lite* make it lose its nice computational properties.

Specifically, we consider the following DL constructs:

| **Construct** | **Syntax** | **Example** | **Semantics** |
|---|---|---|---|
| conjunction | $C_1 \sqcap C_2$ | Doctor $\sqcap$ Male | $C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$ |
| disjunction | $C_1 \sqcup C_2$ | Doctor $\sqcup$ Lawyer | $C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$ |
| qual. exist. restr. | $\exists Q.C$ | $\exists$child.Male | $\{a \mid \exists b.\, (a,b) \in Q^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}$ |
| qual. univ. restr. | $\forall Q.C$ | $\forall$child.Male | $\{a \mid \forall b.\, (a,b) \in Q^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}$ |

# Beyond *DL-Lite*$_\mathcal{A}$: results on data complexity

| | Lhs | Rhs | funct. | Role incl. | Data complexity of query answering |
|---|---|---|---|---|---|
| 0 | *DL-Lite*$_\mathcal{A}$ | | $\surd^*$ | $\surd^*$ | in $\mathrm{AC}^0$ |
| 1 | $A \mid \exists P.A$ | $A$ | – | – | NLogSpace-hard |
| 2 | $A$ | $A \mid \forall P.A$ | – | – | NLogSpace-hard |
| 3 | $A$ | $A \mid \exists P.A$ | $\surd$ | – | NLogSpace-hard |
| 4 | $A \mid \exists P.A \mid A_1 \sqcap A_2$ | $A$ | – | – | PTime-hard |
| 5 | $A \mid A_1 \sqcap A_2$ | $A \mid \forall P.A$ | – | – | PTime-hard |
| 6 | $A \mid A_1 \sqcap A_2$ | $A \mid \exists P.A$ | $\surd$ | – | PTime-hard |
| 7 | $A \mid \exists P.A \mid \exists P^-.A$ | $A \mid \exists P$ | – | – | PTime-hard |
| 8 | $A \mid \exists P \mid \exists P^-$ | $A \mid \exists P \mid \exists P^-$ | $\surd$ | $\surd$ | PTime-hard |
| 9 | $A \mid \neg A$ | $A$ | – | – | coNP-**hard** |
| 10 | $A$ | $A \mid A_1 \sqcup A_2$ | – | – | coNP-**hard** |
| 11 | $A \mid \forall P.A$ | $A$ | – | – | coNP-**hard** |

*Notes:*

- * with the "proviso" of not specializing functional properties.
- NLogSpace and PTime hardness holds already for instance checking.
- For coNP-hardness in line 10, a TBox with a single assertion $A_L \sqsubseteq A_T \sqcup A_F$ suffices! $\rightsquigarrow$ **No** hope of including **covering constraints**.

# Observations

- *DL-Lite*-family is FOL-rewritable, hence LogSpace – holds also with $n$-ary relations $\rightsquigarrow$ *DLR-Lite*$_{\mathcal{F}}$ and *DLR-Lite*$_{\mathcal{R}}$.
- RDFS is a subset of *DL-Lite*$_{\mathcal{R}}$ $\rightsquigarrow$ is FOL-rewritable, hence LogSpace.
- Horn-$\mathcal{SHIQ}$ [Hustadt *et al.*, 2005] is PTime-**hard** even for instance checking (line 8).
- DLP [Grosof *et al.*, 2003] is PTime-**hard** (line 6)
- $\mathcal{EL}$ [Baader *et al.*, 2005] is PTime-**hard** (line 6).
- Although used in ER and UML, no hope of including **covering constraints**, since we get coNP-hardness for trivial DLs (line 10)

# Outline of Part 3

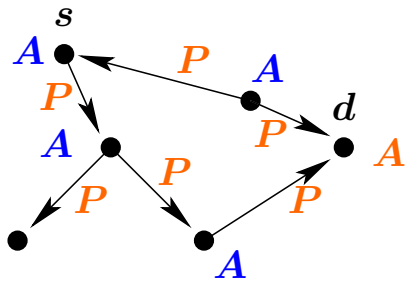
# Qualified existential quantification in the lhs of inclusions

Adding **qualified existential on the lhs** of inclusions makes instance checking (and hence query answering) NLogSpace-hard:

| | $Cl$ | $Cr$ | $Lhs$ | $Rhs$ | Data complexity |
|---|---|---|---|---|---|
| 1 | $A \mid \exists P.A$ | $A$ | – | – | NLogSpace-hard |

Hardness proof is by a reduction from reachability in directed graphs:

- TBox $\mathcal{T}$: a single inclusion assertion $\exists P.A \sqsubseteq A$
- ABox $\mathcal{A}$: encodes graph using $P$ and asserts $A(d)$




Result:

$\langle \mathcal{T}, \mathcal{A} \rangle \models A(s)$ iff $d$ is reachable from $s$ in the graph.

# NLogSpace-hard cases

Instance checking (and hence query answering) is NLogSpace-hard in data complexity for:

| | Lhs | Rhs | $\mathcal{F}$ | $\mathcal{R}$ | Data complexity |
|---|---|---|---|---|---|
| 1 | $A \mid \exists P.A$ | $A$ | – | – | NLogSpace-hard |

By reduction from reachability in directed graphs

| | | | | | |
|---|---|---|---|---|---|
| 2 | $A$ | $A \mid \forall P.A$ | – | – | NLogSpace-hard |

Follows from 1 by replacing $\exists P.A_1 \sqsubseteq A_2$ with $A_1 \sqsubseteq \forall P^-.A_2$, and by replacing each occurrence of $P^-$ with $P'$, for a new role $P'$.

| | | | | | |
|---|---|---|---|---|---|
| 3 | $A$ | $A \mid \exists P.A$ | $\surd$ | – | NLogSpace-hard |

Proved by simulating in the reduction $\exists P.A_1 \sqsubseteq A_2$ via $A_1 \sqsubseteq \exists P^-.A_2$ and (**funct** $P^-$), and by replacing again each occurrence of $P^-$ with $P'$, for a new role $P'$.

# Outline of Part 3

# Path System Accessibility

Instance of Path System Accessibility: $PS = (N, E, S, t)$ with

- $N$ a set of nodes
- $E \subseteq N \times N \times N$ an accessibility relation
- $S \subseteq N$ a set of source nodes
- $t \in N$ a terminal node

**Accessibility** of nodes is defined inductively:

- each $n \in S$ is accessible
- if $(n, n_1, n_2) \in E$ and $n_1$, $n_2$ are accessible, then also $n$ is accessible

Given $PS$, checking whether $t$ is accessible, is PTIME-complete.

# Reduction from Path System Accessibility

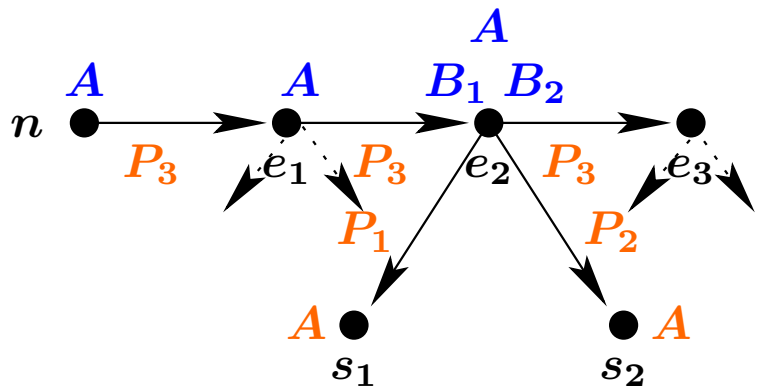
Given an instance $PS = (N, E, S, t)$, we construct

- TBox $\mathcal{T}$ consisting of the inclusion assertions

$$\begin{array}{rcl} \exists P_1.A & \sqsubseteq & B_1 \\ \exists P_2.A & \sqsubseteq & B_2 \end{array} \qquad \begin{array}{rcl} B_1 \sqcap B_2 & \sqsubseteq & A \\ \exists P_3.A & \sqsubseteq & A \end{array}$$

- ABox $\mathcal{A}$ encoding the accessibility relation using $P_1$, $P_2$, and $P_3$, and asserting $A(s)$ for each source node $s \in S$

$$\begin{aligned} e_1 &= (n, \ . \ , \ . \ ) \\ e_2 &= (n, s_1, s_2) \\ e_3 &= (n, \ . \ , \ . \ ) \end{aligned}$$




Result:

$\langle \mathcal{T}, \mathcal{A} \rangle \models A(t)$ iff $t$ is accessible in $PS$.

# Outline of Part 3

# coNP-hard cases

Are obtained when we can use in the query **two concepts that cover another concept**. This forces **reasoning by cases** on the data.

Query answering is coNP-hard in data complexity for:

| | $Cl$ | $Cr$ | $\mathcal{F}$ | $\mathcal{R}$ | Data complexity |
|---|---|---|---|---|---|
| 9 | $A \mid \neg A$ | $A$ | – | – | coNP-hard |
| 10 | $A$ | $A \mid A_1 \sqcup A_2$ | – | – | coNP-hard |
| 11 | $A \mid \forall P.A$ | $A$ | – | – | coNP-hard |

All three cases are proved by adapting the proof of coNP-hardness of instance checking for $\mathcal{ALE}$ by [Donini *et al.*, 1994].

# 2+2-SAT

**2+2-SAT**: satisfiability of a 2+2-CNF formula, i.e., a CNF formula where each clause has exactly 2 positive and 2 negative literals.

Example: $\varphi = c_1 \wedge c_2 \wedge c_3$, with

$$\begin{aligned} c_1 &= v_1 \vee v_2 \vee \neg v_3 \vee \neg v_4 \\ c_2 &= \mathit{false} \vee \mathit{false} \vee \neg v_1 \vee \neg v_4 \\ c_3 &= \mathit{false} \vee v_4 \vee \neg \mathit{true} \vee \neg v_2 \end{aligned}$$

**2+2-SAT is NP-complete** [Donini *et al.*, 1994].

# Reduction from 2+2-SAT

2+2-CNF formula $\varphi = c_1 \wedge \cdots \wedge c_k$ over variables $v_1, \ldots, v_n$, *true*, *false*

- Ontology is over concepts $L$, $T$, $F$, roles $P_1$, $P_2$, $N_1$, $N_2$ and individuals $\mathtt{v}_1, \ldots, \mathtt{v}_n$, $\mathtt{true}$, $\mathtt{false}$, $\mathtt{c}_1, \ldots \mathtt{c}_k$
- ABox $\mathcal{A}_\varphi$ constructed from $\varphi$:
  - for each propositional variable $v_i$: $L(\mathtt{v}_i)$
  - for each clause $c_j = v_{j_1} \vee v_{j_2} \vee \neg v_{j_3} \vee \neg v_{j_4}$:
    $P_1(\mathtt{c}_j, \mathtt{v}_{j_1})$, $P_2(\mathtt{c}_j, \mathtt{v}_{j_2})$, $N_1(\mathtt{c}_j, \mathtt{v}_{j_3})$, $N_2(\mathtt{c}_j, \mathtt{v}_{j_4})$
  - $T(\mathtt{true})$, $F(\mathtt{false})$
- TBox $\mathcal{T} = \{ L \sqsubseteq T \sqcup F \}$
- $q() \leftarrow P_1(c, v_1), P_2(c, v_2), N_1(c, v_3), N_2(c, v_4),$
  $F(v_1), F(v_2), T(v_3), T(v_4)$

*Note:* the TBox $\mathcal{T}$ and the query $q$ do not depend on $\varphi$, hence this reduction works for data complexity.

# Reduction from 2+2-SAT (cont'd)

**Lemma**

$\langle \mathcal{T}, A_\varphi \rangle \not\models q()$ iff $\varphi$ is satisfiable.

**Proof (sketch).**

"$\Rightarrow$" If $\langle \mathcal{T}, A_\varphi \rangle \not\models q()$, then there is a model $\mathcal{I}$ of $\langle \mathcal{T}, A_\varphi \rangle$ s.t. $\mathcal{I} \not\models q()$. We define a truth assignment $\alpha_\mathcal{I}$ by setting $\alpha_\mathcal{I}(v_i) = \mathit{true}$ iff $\mathrm{v}_i^\mathcal{I} \in T^\mathcal{I}$. Notice that, since $L \sqsubseteq T \sqcup F$, if $\mathrm{v}_i^\mathcal{I} \notin T^\mathcal{I}$, then $\mathrm{v}_i^\mathcal{I} \in F^\mathcal{I}$.
It is easy to see that, since $q()$ asks for a false clause and $\mathcal{I} \not\models q()$, for each clause $c_j$, one of the literals in $c_j$ evaluates to *true* in $\alpha_\mathcal{I}$.
"$\Leftarrow$" From a truth assignment $\alpha$ that satisfies $\varphi$, we construct an interpretation $\mathcal{I}_\alpha$ with $\Delta^{\mathcal{I}_\alpha} = \{c_1, \ldots, c_k, v_1, \ldots, v_n, t, f\}$, and:

- $\mathrm{c}_j^{\mathcal{I}_\alpha} = c_j$, $\mathrm{v}_i^{\mathcal{I}_\alpha} = v_i$, $\mathtt{true}^{\mathcal{I}_\alpha} = t$, $\mathtt{false}^{\mathcal{I}_\alpha} = f$
- $T^{\mathcal{I}_\alpha} = \{v_i \mid \alpha(v_i) = \mathit{true}\} \cup \{t\}$, $F^{\mathcal{I}_\alpha} = \{v_i \mid \alpha(v_i) = \mathit{false}\} \cup \{f\}$

It is easy to see that $\mathcal{I}_\alpha$ is a model of $\langle \mathcal{T}, A_\varphi \rangle$ and that $\mathcal{I}_\alpha \not\models q()$. □

# Outline of Part 3

# Combining functionalities and role inclusions – Example

TBox $\mathcal{T}$: $A \sqsubseteq \exists P$ $\quad P \sqsubseteq S$ $\quad \exists P^- \sqsubseteq A$ $\quad (\textbf{funct } S)$

ABox $\mathcal{A}$: $A(c_1),\ S(c_1, c_2),\ S(c_2, c_3),\ \ldots,\ S(c_{n-1}, c_n)$

$$
\begin{array}{rrl}
A(c_1), & A \sqsubseteq \exists P & \models \quad P(c_1, x), \text{ for some } x \\
P(c_1, x), & P \sqsubseteq S & \models \quad S(c_1, x) \\
S(c_1, x), \quad S(c_1, c_2), & (\textbf{funct } S) & \models \quad x = c_2 \\
P(c_1, c_2), & \exists P^- \sqsubseteq A & \models \quad A(c_2) \\
A(c_2), & A \sqsubseteq \exists P & \ldots \\
 & & \models \quad A(c_n)
\end{array}
$$

Hence, we get:

- If we add $B(c_n)$ and $B \sqsubseteq \neg A$, the ontology becomes inconsistent.
- Similarly, the answer to the following query over $\langle \mathcal{T}, \mathcal{A} \rangle$ is *true*:

$$q() \leftarrow A(z_1), S(z_1, z_2), S(z_2, z_3), \ldots, S(z_{n-1}, z_n), A(z_n)$$

# Interaction between functionalities and role inclusions

*Note:* The number of unification steps above **depends on the data**. Hence this kind of deduction cannot be mimicked by a FOL (or SQL) query, since it requires a form of **recursion**. As a consequence, we get:

**Combining functionality and role inclusions is problematic.**

It breaks **separability**, i.e., functionality assertions may force existentially quantified objects to be unified with existing objects.

*Note:* the problems are caused by the **interaction** among:

- an inclusion $P \sqsubseteq S$ between roles,
- a functionality assertion $(\textbf{funct}\ S)$ on the super-role, and
- a cycle of concept inclusion assertions $A \sqsubseteq \exists P$ and $\exists P^- \sqsubseteq A$.

Since we do not want to limit cycles of ISA, we pose suitable restrictions on the combination of functionality and role inclusions

# Interaction between funct. and role inclusions (cont.'d)

**Theorem** [Artale *et al.*, 2009]

Checking the satisfiability of a *DL-Lite* TBox that allows for combining functionality and role inclusions in an unrestricted way is ExpTime-complete.

**Proof (idea).**

- By using functionality and role inclusions, we can simulate $A_1 \sqcap A_2 \sqsubseteq C$

$$
\begin{array}{rll}
A_1 \sqsubseteq \exists R_1 & A_2 \sqsubseteq \exists R_2 & \\
R_1 \sqsubseteq R_{12} & R_2 \sqsubseteq R_{12} & (\textbf{funct } R_{12}) \\
\exists R_1^- \sqsubseteq \exists R_3^- & & \\
\exists R_3 \sqsubseteq C & & \\
R_3 \sqsubseteq R_{23} & R_2 \sqsubseteq R_{23} & (\textbf{funct } R_{23}^-)
\end{array}
$$

- $A \sqsubseteq \exists R.C$ is simulated by $A \sqsubseteq \exists R_C, \quad R_C \sqsubseteq R, \quad \exists R_C^- \sqsubseteq C.$
- $A \sqsubseteq \forall R.C$ can be simulated using reification.

# Complexity of *DL-Lite* with funct. and role inclusions

Let *DL-Lite*$_{\mathcal{FR}}$ be the DL that is the union of *DL-Lite*$_{\mathcal{F}}$ and *DL-Lite*$_{\mathcal{R}}$, i.e., the *DL-Lite* logic that allows for using both role functionality and role inclusions without any restrictions.

**Theorem** [Artale *et al.*, 2009]

For *DL-Lite*$_{\mathcal{FR}}$ ontologies:

- Checking satisfiability of the ontology is
  - EXPTIME-**complete** in the size of the **ontology** (combined complexity).
  - PTIME-**complete** in the size of the **ABox** (data complexity).
- TBox reasoning is EXPTIME-**complete** in the size of the **TBox**.
- Query answering is
  - NP-**complete** in the size of the query and the ontology (comb. com.).
  - EXPTIME-**complete** in the size of the **ontology**.
  - PTIME-**complete** in the size of the **ABox** (data complexity).

# Combining functionalities and role inclusions

We have seen that:

- By including in *DL-Lite* both functionality of roles and role inclusions without restrictions on their interaction, query answering becomes PTIME-hard.
- When the data complexity of query answering is NLOGSPACE or above, the DL does not enjoy FOL-rewritability.

**As a consequence of these results, we get:**

To preserve FOL-rewritability, the restriction on the interaction of functionality and role inclusions of *DL-Lite*$_{\mathcal{A}}$ is necessary.

# Part IV

# Linking ontologies to relational data

# Outline of Part 4

# Outline of Part 4

# Managing ABoxes

In the traditional DL setting, it is assumed that the data is maintained in the ABox of the ontology:

- The ABox is perfectly compatible with the TBox:
  - the vocabulary of concepts, roles, and attributes is the one used in the TBox.
  - The ABox "stores" abstract objects, and these objects and their properties are those returned by queries over the ontology.
- There may be different ways to manage the ABox from a physical point of view:
  - Description Logics reasoners maintain the ABox is main-memory data structures.
  - When an ABox becomes large, managing it in secondary storage may be required, but this is again handled directly by the reasoner.

# Data in external sources

There are several situations where the assumptions of having the data in an ABox managed directly by the ontology system (e.g., a Description Logics reasoner) is not feasible or realistic:

- When the ABox is very large, so that it requires relational database technology.
- When we have no direct control over the data since it belongs to some external organization, which controls the access to it.
- When multiple data sources need to be accessed, such as in Information Integration.

We would like to deal with such a situation by keeping the data in the external (relational) storage, and performing **query answering** by leveraging the capabilities of the **relational engine**.

# The impedance mismatch problem

We have to deal with the **impedance mismatch problem**:

- Sources store data, which is constituted by values taken from concrete domains, such as strings, integers, codes, ...
- Instead, instances of concepts and relations in an ontology are (abstract) objects.

**Solution:**

- We need to specify how to construct from the data values in the relational sources the (abstract) objects that populate the ABox of the ontology.
- This specification is embedded in the mappings between the data sources and the ontology.

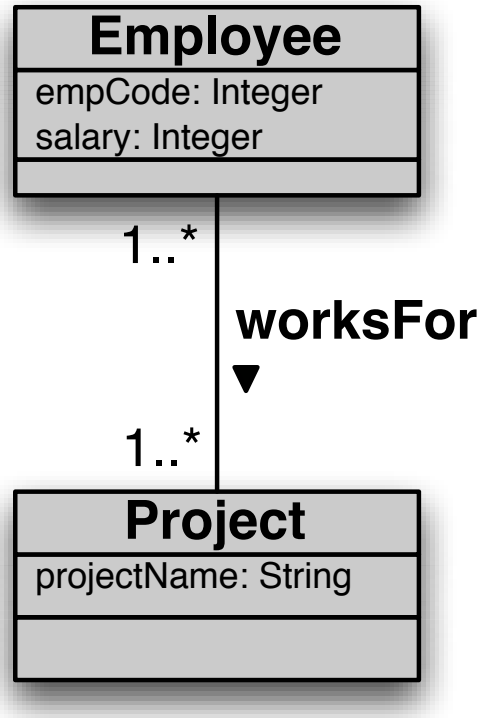
*Note:* the **ABox** is only **virtual**, and the objects are not materialized.

# Solution to the impedance mismatch problem

We need to define a **mapping language** that allows for specifying how to transform data into abstract objects:

- Each mapping assertion maps:
  - a query that retrieves values from a data source to ...
  - a set of atoms specified over the ontology.
- Basic idea: use **Skolem functions** in the atoms over the ontology to "generate" the objects from the data values.
- Semantics of mappings:
  - Objects are denoted by terms (of exactly one level of nesting).
  - Different terms denote different objects (i.e., we make the unique name assumption on terms).

# Impedance mismatch – Example




Actual data is stored in a DB:
– An employee is identified by her SSN.
– A project is identified by its name.

$D_1[SSN: \text{String}, PrName: \text{String}]$
Employees and projects they work for

$D_2[Code: \text{String}, Salary: \text{Int}]$
Employee's code with salary

$D_3[Code: \text{String}, SSN: \text{String}]$
Employee's Code with SSN

...

Intuitively:

- An employee should be created from her SSN: **pers**(*SSN*)
- A project should be created from its name: **proj**(*PrName*)

# Creating object identifiers

We need to associate to the data in the tables objects in the ontology.

- We introduce an alphabet $\Lambda$ of **function symbols**, each with an associated arity.
- To denote values, we use value constants from an alphabet $\Gamma_V$.
- To denote objects, we use **object terms** instead of object constants.
  An object term has the form $\mathbf{f}(d_1, \ldots, d_n)$, with $\mathbf{f} \in \Lambda$, and each $d_i$ a value constant in $\Gamma_V$.

Example

- If a person is identified by her *SSN*, we can introduce a function symbol **pers**/1. If `VRD56B25` is a *SSN*, then **pers**(`VRD56B25`) denotes a person.
- If a person is identified by her *name* and *dateOfBirth*, we can introduce a function symbol **pers**/2. Then **pers**(`Vardi`, `25/2/56`) denotes a person.

# Mapping assertions

Mapping assertions are used to extract the data from the DB to populate the ontology.

We make use of **variable terms**, which are like object terms, but with variables instead of values as arguments of the functions.

**Def.: Mapping assertion between a database and a TBox**

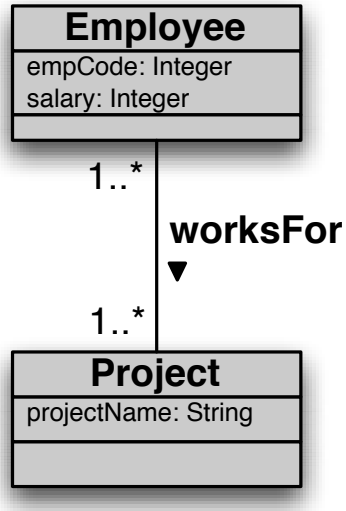
A **mapping assertion** between a database $\mathcal{D}$ and a TBox $\mathcal{T}$ has the form

$$\Phi \rightsquigarrow \Psi$$

where

- $\Phi$ is an arbitrary SQL query of arity $n > 0$ over $\mathcal{D}$.
- $\Psi$ is a conjunctive query over $\mathcal{T}$ of arity $n' > 0$ **without non-distinguished variables**, possibly involving variable terms.

# Mapping assertions – Example




$D_1[SSN: \text{String}, PrName: \text{String}]$
Employees and Projects they work for

$D_2[Code: \text{String}, Salary: \text{Int}]$
Employee's code with salary

$D_3[Code: \text{String}, SSN: \text{String}]$
Employee's code with SSN

...

$m_1$: `SELECT SSN, PrName FROM` $D_1$ $\rightsquigarrow$ Employee(**pers**(*SSN*)), Project(**proj**(*PrName*)), projectName(**proj**(*PrName*), *PrName*), worksFor(**pers**(*SSN*), **proj**(*PrName*))

$m_2$: `SELECT SSN, Salary FROM` $D_2$`,` $D_3$ `WHERE` $D_2$`.Code =` $D_3$`.Code` $\rightsquigarrow$ Employee(**pers**(*SSN*)), salary(**pers**(*SSN*), *Salary*)

# Outline of Part 4

# Ontology-Based Data Access System

The mapping assertions are a crucial part of an Ontology-Based Data Access System.

**Def.: Ontology-Based Data Access System**

is a triple $\mathcal{O} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$, where

- $\mathcal{T}$ is a TBox.
- $\mathcal{D}$ is a relational database.
- $\mathcal{M}$ is a set of mapping assertions between $\mathcal{T}$ and $\mathcal{D}$.

We need to specify the syntax and semantics of mapping assertions.

# Mapping assertions

A mapping assertion in $\mathcal{M}$ has the form

$$\Phi(\vec{x}) \leadsto \Psi(\vec{t}, \vec{y})$$

where

- $\Phi$ is an arbitrary SQL query of arity $n > 0$ over $\mathcal{D}$;
- $\Psi$ is a conjunctive query over $\mathcal{T}$ of arity $n' > 0$ **without non-distinguished variables**;
- $\vec{x}$, $\vec{y}$ are variables, with $\vec{y} \subseteq \vec{x}$;
- $\vec{t}$ are variable terms of the form $f(\vec{z})$, with $f \in \Lambda$ and $\vec{z} \subseteq \vec{x}$.

*Note:* we could consider also mapping assertions between the datatypes of the database and those of the ontology.

# Semantics of mappings

To define the semantics of an OBDA system $\mathcal{O} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$, we first need to define the semantics of mappings.

**Def.: Satisfaction of a mapping assertion with respect to a database**

An interpretation $\mathcal{I}$ **satisfies** a mapping assertion $\Phi(\vec{x}) \leadsto \Psi(\vec{t}, \vec{y})$ in $\mathcal{M}$ **with respect to a database** $\mathcal{D}$, if for each tuple of values $\vec{v} \in \mathit{Eval}(\Phi, \mathcal{D})$, and for each ground atom in $\Psi[\vec{x}/\vec{v}]$, we have that:

- if the ground atom is $A(s)$, then $s^{\mathcal{I}} \in A^{\mathcal{I}}$.
- if the ground atom is $P(s_1, s_2)$, then $(s_1^{\mathcal{I}}, s_2^{\mathcal{I}}) \in P^{\mathcal{I}}$.

Intuitively, $\mathcal{I}$ **satisfies** $\Phi \leadsto \Psi$ w.r.t. $\mathcal{D}$ if all facts obtained by evaluating $\Phi$ over $\mathcal{D}$ and then propagating the answers to $\Psi$, hold in $\mathcal{I}$.

*Note:* $\mathit{Eval}(\Phi, \mathcal{D})$ denotes the result of evaluating $\Phi$ over the database $\mathcal{D}$.
$\Psi[\vec{x}/\vec{v}]$ denotes $\Psi$ where each $x_i$ has been substituted with $v_i$.

# Semantics of an OBDA system

**Def.: Model of an OBDA system**

An interpretation $\mathcal{I}$ is a **model** of $\mathcal{O} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$ if:

- $\mathcal{I}$ is a model of $\mathcal{T}$;
- $\mathcal{I}$ satisfies $\mathcal{M}$ w.r.t. $\mathcal{D}$, i.e., $\mathcal{I}$ satisfies every assertion in $\mathcal{M}$ w.r.t. $\mathcal{D}$.

An OBDA system $\mathcal{O}$ is **satisfiable** if it admits at least one model.

# Outline of Part 4

# Answering queries over an OBDA system

In an OBDA system $\mathcal{O} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$

- Queries are posed over the TBox $\mathcal{T}$.
- The data needed to answer queries is stored in the database $\mathcal{D}$.
- The mapping $\mathcal{M}$ is used to bridge the gap between $\mathcal{T}$ and $\mathcal{D}$.

Two approaches to exploit the mapping:

- bottom-up approach: simpler, but less efficient
- top-down approach: more sophisticated, but also more efficient

*Note:* Both approaches require to first **split** the TBox queries in the mapping assertions into their constituent atoms.

# Splitting of mappings

A mapping assertion $\Phi \rightsquigarrow \Psi$, where the TBox query $\Psi$ is constituted by the atoms $X_1,\ldots,X_k$, can be split into several mapping assertions:

$$\Phi \rightsquigarrow X_1 \qquad \cdots \qquad \Phi \rightsquigarrow X_k$$

This is possible, since $\Psi$ does not contain non-distinguished variables.

## Example

$m_1$: `SELECT SSN, PrName FROM D1` $\rightsquigarrow$ Employee(**pers**(*SSN*)), Project(**proj**(*PrName*)), projectName(**proj**(*PrName*), *PrName*), worksFor(**pers**(*SSN*), **proj**(*PrName*))

is split into

$m_1^1$: `SELECT SSN, PrName FROM D1` $\rightsquigarrow$ Employee(**pers**(*SSN*))

$m_1^2$: `SELECT SSN, PrName FROM D1` $\rightsquigarrow$ Project(**proj**(*PrName*))

$m_1^3$: `SELECT SSN, PrName FROM D1` $\rightsquigarrow$ projectName(**proj**(*PrName*), *PrName*)

$m_1^4$: `SELECT SSN, PrName FROM D1` $\rightsquigarrow$ worksFor(**pers**(*SSN*), **proj**(*PrName*))

# Bottom-up approach to query answering

Consists in a straightforward application of the mappings:

1. Propagate the data from $\mathcal{D}$ through $\mathcal{M}$, materializing an ABox $\mathcal{A}_{\mathcal{M},\mathcal{D}}$ (the constants in such an ABox are values and object terms).
2. Apply to $\mathcal{A}_{\mathcal{M},\mathcal{D}}$ and to the TBox $\mathcal{T}$, the satisfiability and query answering algorithms developed for *DL-Lite*$_{\mathcal{A}}$.

This approach has several drawbacks (hence is only theoretical):

- The technique is no more LogSpace in the data, since the ABox $\mathcal{A}_{\mathcal{M},\mathcal{D}}$ to materialize is in general polynomial in the size of the data.
- $\mathcal{A}_{\mathcal{M},\mathcal{D}}$ may be very large, and thus it may be infeasible to actually materialize it.
- Freshness of $\mathcal{A}_{\mathcal{M},\mathcal{D}}$ with respect to the underlying data source(s) may be an issue, and one would need to propagate source updates (cf. Data Warehousing).

# Top-down approach to query answering

Consists of three steps:

1. **Reformulation:** Compute the perfect reformulation $q_{pr} = \mathit{PerfectRef}(q, \mathcal{T}_P)$ of the original query $q$, using the inclusion assertions of the TBox $\mathcal{T}$ (see later).

2. **Unfolding:** Compute from $q_{pr}$ a new query $q_{unf}$ by unfolding $q_{pr}$ using (the split version of) the mappings $\mathcal{M}$.
   - Essentially, each atom in $q_{pr}$ that unifies with an atom in $\Psi$ is substituted with the corresponding query $\Phi$ over the database.
   - The unfolded query is such that $\mathit{Eval}(q_{unf}, \mathcal{D}) = \mathit{Eval}(q_{pr}, \mathcal{A}_{\mathcal{M},\mathcal{D}})$.

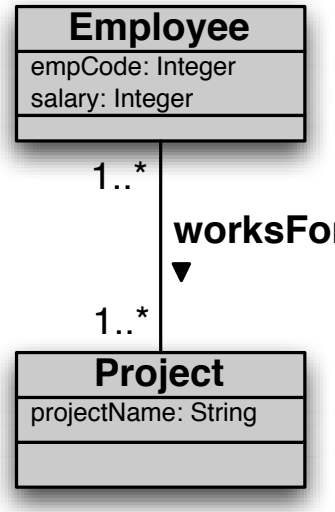
3. **Evaluation:** Delegate the evaluation of $q_{unf}$ to the relational DBMS managing $\mathcal{D}$.

# Unfolding

To unfold a query $q_{pr}$ with respect to a set of mapping assertions:

1. For each non-split mapping assertion $\Phi_i(\vec{x}) \rightsquigarrow \Psi_i(\vec{t}, \vec{y})$:
   1. Introduce a **view symbol** $\mathsf{Aux}_i$ of arity equal to that of $\Phi_i$.
   2. Add a **view definition** $\mathsf{Aux}_i(\vec{x}) \leftarrow \Phi_i(\vec{x})$.
2. For each split version $\Phi_i(\vec{x}) \rightsquigarrow X_j(\vec{t}, \vec{y})$ of a mapping assertion, introduce a **clause** $X_j(\vec{t}, \vec{y}) \leftarrow \mathsf{Aux}_i(\vec{x})$.
3. Obtain from $q_{pr}$ in all possible ways queries $q_{aux}$ defined over the view symbols $\mathsf{Aux}_i$ as follows:
   1. Find a most general unifier $\vartheta$ that unifies each atom $X(\vec{z})$ in the body of $q_{pr}$ with the head of a clause $X(\vec{t}, \vec{y}) \leftarrow \mathsf{Aux}_i(\vec{x})$.
   2. Substitute each atom $X(\vec{z})$ with $\vartheta(\mathsf{Aux}_i(\vec{x}))$, i.e., with the body the unified clause to which the unifier $\vartheta$ is applied.
4. The unfolded query $q_{unf}$ is the **union** of all queries $q_{aux}$, together with the view definitions for the predicates $\mathsf{Aux}_i$ appearing in $q_{aux}$.

# Unfolding – Example




$m_1$: `SELECT SSN, PrName FROM D1` $\rightsquigarrow$ Employee(**pers**(*SSN*)), Project(**proj**(*PrName*)), projectName(**proj**(*PrName*), *PrName*), worksFor(**pers**(*SSN*), **proj**(*PrName*))

$m_2$: `SELECT SSN, Salary FROM D2, D3 WHERE D2.Code = D3.Code` $\rightsquigarrow$ Employee(**pers**(*SSN*)), salary(**pers**(*SSN*), *Salary*)

We define a view $\text{Aux}_i$ for the source query of each mapping $m_i$.

For each (split) mapping assertion, we introduce a clause:

$$\begin{aligned}
\text{Employee}(\mathbf{pers}(\mathit{SSN})) &\leftarrow \text{Aux}_1(\mathit{SSN}, \mathit{PrName}) \\
\text{projectName}(\mathbf{proj}(\mathit{PrName}), \mathit{PrName}) &\leftarrow \text{Aux}_1(\mathit{SSN}, \mathit{PrName}) \\
\text{Project}(\mathbf{proj}(\mathit{PrName})) &\leftarrow \text{Aux}_1(\mathit{SSN}, \mathit{PrName}) \\
\text{worksFor}(\mathbf{pers}(\mathit{SSN}), \mathbf{proj}(\mathit{PrName})) &\leftarrow \text{Aux}_1(\mathit{SSN}, \mathit{PrName}) \\
\text{Employee}(\mathbf{pers}(\mathit{SSN})) &\leftarrow \text{Aux}_2(\mathit{SSN}, \mathit{Salary}) \\
\text{salary}(\mathbf{pers}(\mathit{SSN}), \mathit{Salary}) &\leftarrow \text{Aux}_2(\mathit{SSN}, \mathit{Salary})
\end{aligned}$$

# Unfolding – Example (cont'd)

Query over ontology: employees who work for `tones` and their salary:

$q(e, s) \leftarrow \mathsf{Employee}(e), \mathsf{salary}(e, s), \mathsf{worksFor}(e, p), \mathsf{projectName}(p, \texttt{tones})$

A unifier between the atoms in $q$ and the clause heads is:

$$\vartheta(e) = \mathbf{pers}(\mathit{SSN}) \qquad \vartheta(s) = \mathit{Salary}$$
$$\vartheta(\mathit{PrName}) = \texttt{tones} \qquad \vartheta(p) = \mathbf{proj}(\texttt{tones})$$

After applying $\vartheta$ to $q$, we obtain:

$$q(\mathbf{pers}(\mathit{SSN}), \mathit{Salary}) \leftarrow \mathsf{Employee}(\mathbf{pers}(\mathit{SSN})),\ \mathsf{salary}(\mathbf{pers}(\mathit{SSN}), \mathit{Salary}),\ \mathsf{worksFor}(\mathbf{pers}(\mathit{SSN}), \mathbf{proj}(\texttt{tones})),\ \mathsf{projectName}(\mathbf{proj}(\texttt{tones}), \texttt{tones})$$

Substituting the atoms with the bodies of the unified clauses, we obtain:

$$q(\mathbf{pers}(\mathit{SSN}), \mathit{Salary}) \leftarrow \mathsf{Aux}_1(\mathit{SSN}, \texttt{tones}),\ \mathsf{Aux}_2(\mathit{SSN}, \mathit{Salary}),\ \mathsf{Aux}_1(\mathit{SSN}, \texttt{tones}),\ \mathsf{Aux}_1(\mathit{SSN}, \texttt{tones})$$

# Exponential blowup in the unfolding

When there are multiple mapping assertions for each atom, the unfolded query may be exponential in the original one.

Consider a query: $q(y) \leftarrow A_1(y), A_2(y), \ldots, A_n(y)$

and the mappings: $m_i^1\colon \Phi_i^1(x) \rightsquigarrow A_i(\mathbf{f}(x))$ (for $i \in \{1, \ldots, n\}$)
$m_i^2\colon \Phi_i^2(x) \rightsquigarrow A_i(\mathbf{f}(x))$

We add the view definitions: $\mathsf{Aux}_i^j(x) \leftarrow \Phi_i^j(x)$
and introduce the clauses: $A_i(\mathbf{f}(x)) \leftarrow \mathsf{Aux}_i^j(x)$ (for $i \in \{1, \ldots, n\}$, $j \in \{1, 2\}$).

There is a single unifier, namely $\vartheta(y) = \mathbf{f}(x)$, but each atom $A_i(y)$ in the query unifies with the head of two clauses.

Hence, we obtain one unfolded query

$$q(\mathbf{f}(x)) \leftarrow \mathsf{Aux}_1^{j_1}(x), \mathsf{Aux}_2^{j_2}(x), \ldots, \mathsf{Aux}_n^{j_n}(x)$$

for each possible combination of $j_i \in \{1, 2\}$, for $i \in \{1, \ldots, n\}$.
Hence, we obtain $2^n$ **unfolded queries**.

# Computational complexity of query answering

From the top-down approach to query answering, and the complexity results for *DL-Lite*, we obtain the following result.

**Theorem**

**Query answering** in a *DL-Lite* OBDM system $\mathcal{O} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$ is

1. NP-**complete** in the size of the query.
2. PTIME in the size of the **TBox** $\mathcal{T}$ and the **mappings** $\mathcal{M}$.
3. LOGSPACE in the size of the **database** $\mathcal{D}$.

*Note:* The LOGSPACE result is a consequence of the fact that query answering in such a setting can be reduced to evaluating an SQL query over the relational database.

# Implementation of top-down approach to query answering

To implement the top-down approach, we need to generate an SQL query.

We can follow different strategies:

1. Substitute each view predicate in the unfolded queries with the corresponding SQL query over the source:
   - \+ joins are performed on the DB attributes;
   - \+ does not generate doubly nested queries;
   - – the number of unfolded queries may be exponential.

2. Construct for each atom in the original query a new view. This view takes the union of all SQL queries corresponding to the view predicates, and constructs also the Skolem terms:
   - \+ avoids exponential blow-up of the resulting query, since the union (of the queries coming from multiple mappings) is done before the joins;
   - – joins are performed on Skolem terms;
   - – generates doubly nested queries.

Which method is better, depends on various parameters.
Experiments have shown that (1) behaves better in most cases.

# Towards answering arbitrary SQL queries

- We have seen that answering full SQL (i.e., FOL) queries is undecidable.
- However, we can treat the answers to an UCQ, as "knowledge", and perform further computations on that knowledge.
- This corresponds to applying a knowledge operator to UCQs that are embedded into an arbitrary SQL query (EQL queries) [Calvanese *et al.*, 2007a]
  - The UCQs are answered according to the certain answer semantics.
  - The SQL query is evaluated on the facts returned by the UCQs.
- The approach can be implemented by rewriting the UCQs and embedding the rewritten UCQs into SQL.
- The user "sees" arbitrary SQL queries, but these SQL queries are evaluated according to a weakened semantics.

# Outline of Part 4

# The QuOnto system

- QuOnto is a tool for representing and reasoning over ontologies of the *DL-Lite* family.
- The basic functionalities it offers are:
  - Ontology representation
  - Ontology satisfiability check
  - Intensional reasoning services: concept/property subsumption and disjunction, concept/property satisfiability
  - Query Answering of UCQs
- Includes also support for:
  - Identification path constraints
  - Denial constraints
  - Epistemic queries (EQL-Lite on UCQs)
  - Epistemic constraints (EQL-Lite constraints)
- Reasoning services are highly optimized.
- Can be used with internal and external DBMS (include drivers for Oracle, DB2, IBM Information Integrator, SQL Server, MySQL, etc.).
- Implemented in Java – *APIs are available for selected projects upon request*.

# QuOnto wrapped versions

Several wrapped versions publicly available at:
`http://www.dis.uniroma1.it/~quonto/` (or just google "`quonto`")

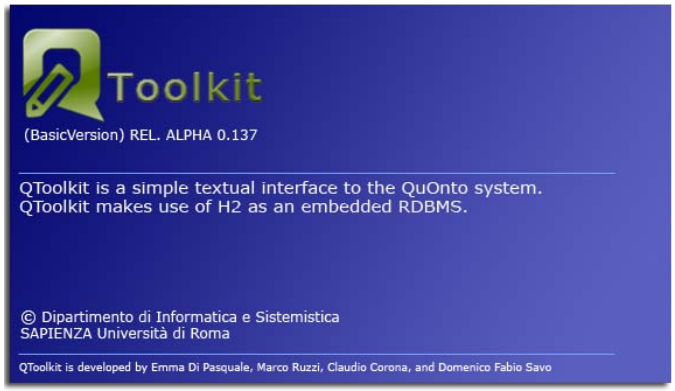
- ROWLkit: first implementation of the OWL2 QL Profile
- QToolKit: simple graphical interface for using QuOnto to reason over *DL-Lite* ontologies
- DIG Server wrapper + OBDA Protégé plugin: for Ontology-based Data Access and Integration through *DL-Lite* ontologies

  *by Mariano Rodriguez Muro, Univ. Bolzano*

# ROWLkit

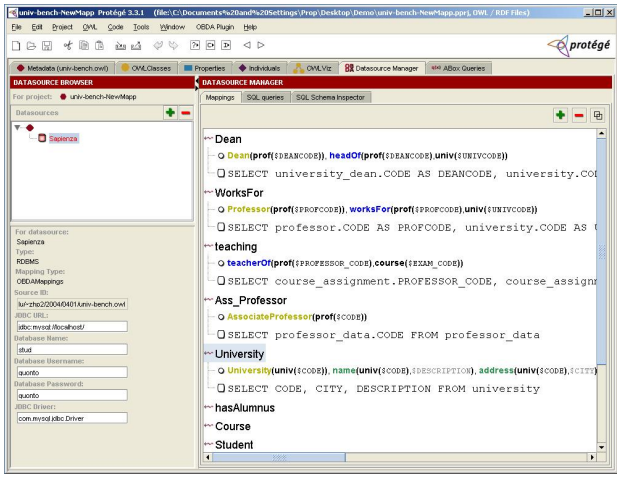

- ROWLKit is a system with a simple GUI to reason over ontologies written in OWL2 QL. At its core it uses QuOnto services enriched with additional features to deal with OWL2 QL ontologies.
- It takes as input OWL2 QL ontologies through OWL API.
- The main services of ROWLKit are:
  - Ontology satisfiability check
  - Intensional reasoning services: concept/property subsumption and disjunction, concept/property satisfiability
  - Query Answering of UCQs – *expressed in SPARQL*
- ROWLKit is written in JAVA and embeds the H2 JAVA relational DBMS for the storage (in main memory) of ABoxes and their querying (support to storage in mass memory is also provided).

# QToolKit




- QToolKit is a simple graphical interface for representing and reasoning over DL-Lite ontologies relying on the QuOnto reasoner.
- It takes as input DL-Lite ontologies specified in the standard OWL functional-style syntax, suitably restricted for *DL-Lite*.
- QToolKit allows for using **all QuOnto reasoning capabilities**. In particular, it allows for answering UCQs (*expressed in Datalog or SPARQL*) and epistemic queries (EQL-Lite on UCQs) (*expressed in SparSQL*) over *DL-Lite*$_{\mathcal{A}}$ ontologies possibly equipped with identification path constraints, denial and epistemic constraints.
- QToolKit stores the ABox in an internal database (no connection to external DBs).

# DIG Server wrapper + OBDA Protégé plugin




- QuOnto offers a DIG 1.1 interface through which it is possible to exploit the mapping capabilities provided by the QuOnto technology and specify mappings between *DL-Lite*$_\mathcal{A}$ ontologies and data managed by external systems (e.g., Oracle, DB2, IBM Information Integrator, etc.).
- An open source plugin for Protégé that extends the ontology editor with facilities to design Mappings towards those external DBMS is available.
- The plugin can be used as a client for QuOnto DIG interface and allows for specifying and querying *DL-Lite*$_\mathcal{A}$ ontologies with mappings.
- Available for Protégé 3.3 and Protégé 4.

Part V

# Conclusions and references

# Query rewriting for more expressive ontology languages

The result presented in this tutorial have recently been extended to more expressive ontology languages, using different techniques:

- In [Artale *et al.*, 2009] various *DL-Lite* extensions are considered, providing a comprehensive treatment of the expressiveness/complexity trade-off for the *DL-Lite* family and related logics:
  - number restrictions besides functionality;
  - conjunction on the left-hand side of inclusions (horn logics);
  - boolean constructs;
  - constraints on roles, such as (ir)reflexivity, (a)symmetry, transitivity;
  - presence and absence of the unique name assumption.
- Alternative query rewriting techniques based on resolution, and applicable also to more expressive logics (leading to recursive rewritings) [Pérez-Urbina *et al.*, 2009].
- Query rewriting techniques for database inspired constraint languages [Calì *et al.*, 2009a; Calì *et al.*, 2009b].

# Further theoretical work

The result presented in this tutorial have also inspired additional work relevant for ontology-based data access:

- We have considered mainly query answering. However, several other ontology-based services are of importance:
  - write-also access: updating a data source through an ontology [De Giacomo *et al.*, 2009];
  - modularity and minimal module extraction [Kontchakov *et al.*, 2008; Kontchakov *et al.*, 2009];
  - privacy aware data access [Calvanese *et al.*, 2008];
  - meta-level reasoning and query answering, a la RDFS [De Giacomo *et al.*, 2008]
  - provenance and explanation [Borgida *et al.*, 2008]
- Reasoning with respect to finite models only [Rosati, 2008].
- We have dealt only with the static aspects of information systems. However a crucial issue is how to deal with **dynamic aspects**. Preliminary results are in [Calvanese *et al.*, 2007c]. The general problem is largely unexplored.

Work on most of these issues is still ongoing.

# Further practical and experimental work

The theoretical results indicate a good computational behaviour in the size of the data. However, performance is a critical issue in practice:

- The rewriting consists of a large number of CQs. Query containment can be used to prune the rewriting. This is already implemented in the QuOnto system, but requires further optimizations.
- The SQL queries generated by the mapping unfolding are not easy to process by the DBMS engine (e.g., they may contain complex joins on skolem terms computed on the fly).
  Different mapping unfolding strategies have a strong impact on computational complexity. Experimentation is ongoing to assess the tradeoff.
- Further extensive experimentations are ongoing:
  - on artificially generated data;
  - on real-world use cases.

# Conclusions

- Ontology-based data access is **ready for prime time**.
- QuOnto provides serious proof of concept of this.
- We are successfully applying QuOnto in various **full-fledged case studies**.
- We are currently **looking for projects** where to apply such technology further!

# Acknowledgements

People involved in this work:

- Sapienza Università di Roma
  - Claudio Corona
  - Giuseppe De Giacomo
  - Domenico Lembo
  - Maurizio Lenzerini
  - Antonella Poggi
  - Riccardo Rosati
  - Marco Ruzzi
  - Domenico Fabio Savo
- Free University of Bozen-Bolzano
  - Mariano Rodriguez Muro
- Many students (thanks!)

This work has been carried out within the EU Strep FET project TONES (Thinking ONtologiES).
http://www.tonesproject.org/

# References I

[Artale *et al.*, 2007] Alessandro Artale, Diego Calvanese, Roman Kontchakov, Vladislav Ryzhikov, and Michael Zakharyaschev.

Reasoning over extended ER models.

In *Proc. of the 26th Int. Conf. on Conceptual Modeling (ER 2007)*, volume 4801 of *Lecture Notes in Computer Science*, pages 277–292. Springer, 2007.

[Artale *et al.*, 2009] Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev.

The *DL-Lite* family and relations.

*J. of Artificial Intelligence Research*, 36, 2009.

[Baader *et al.*, 2003] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors.

*The Description Logic Handbook: Theory, Implementation and Applications*.

Cambridge University Press, 2003.

[Baader *et al.*, 2005] Franz Baader, Sebastian Brandt, and Carsten Lutz.

Pushing the $\mathcal{EL}$ envelope.

In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 364–369, 2005.

# References II

[Berardi *et al.*, 2005] Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo.
Reasoning on UML class diagrams.
*Artificial Intelligence*, 168(1–2):70–118, 2005.

[Borgida *et al.*, 2008] Alexander Borgida, Diego Calvanese, and Mariano Rodriguez-Muro.
Explanation in the *DL-Lite* family of description logics.
In *Proc. of the 7th Int. Conf. on Ontologies, DataBases, and Applications of Semantics (ODBASE 2008)*, volume 5332 of *Lecture Notes in Computer Science*, pages 1440–1457. Springer, 2008.

[Calì and Kifer, 2006] Andrea Calì and Michael Kifer.
Containment of conjunctive object meta-queries.
In *Proc. of the 32nd Int. Conf. on Very Large Data Bases (VLDB 2006)*, pages 942–952, 2006.

[Calì *et al.*, 2009a] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz.
Datalog$^{\pm}$: a unified approach to ontologies and integrity constraints.
In *Proc. of the 12th Int. Conf. on Database Theory (ICDT 2009)*, pages 14–30, 2009.

# References III

[Calì *et al.*, 2009b] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz.

A general Datalog-based framework for tractable query answering over ontologies.

In *Proc. of the 28th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2009)*, pages 77–86, 2009.

[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.*, 2005a] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati.

Tailoring OWL for data intensive ontologies.

In *Proc. of the 1st Int. Workshop on OWL: Experiences and Directions (OWLED 2005)*, volume 188 of *CEUR Electronic Workshop Proceedings*, `http://ceur-ws.org/`, 2005.

# References IV

[Calvanese *et al.*, 2005b] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati.

*DL-Lite*: Tractable description logics for ontologies.

In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, pages 602–607, 2005.

[Calvanese *et al.*, 2006a] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, and Riccardo Rosati.

Linking data to ontologies: The description logic $DL\text{-}Lite_A$.

In *Proc. of the 2nd Int. Workshop on OWL: Experiences and Directions (OWLED 2006)*, volume 216 of *CEUR Electronic Workshop Proceedings*, `http://ceur-ws.org/`, 2006.

[Calvanese *et al.*, 2006b] 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.

# References V

[Calvanese *et al.*, 2007a] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati.

EQL-Lite: Effective first-order query processing in description logics.

In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, pages 274–279, 2007.

[Calvanese *et al.*, 2007b] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati.

Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family.

*J. of Automated Reasoning*, 39(3):385–429, 2007.

[Calvanese *et al.*, 2007c] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati.

Actions and programs over description logic ontologies.

In *Proc. of the 20th Int. Workshop on Description Logic (DL 2007)*, volume 250 of *CEUR Electronic Workshop Proceedings*, `http://ceur-ws.org/`, pages 29–40, 2007.

# References VI

[Calvanese *et al.*, 2008] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati.

View-based query answering over description logic ontologies.

In *Proc. of the 11th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2008)*, pages 242–251, 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.

[De Giacomo *et al.*, 2008] Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati.

Towards higher-order *DL-Lite*.

In *Proc. of the 21st Int. Workshop on Description Logic (DL 2008)*, volume 353 of *CEUR Electronic Workshop Proceedings*, `http://ceur-ws.org/`, 2008.

[De Giacomo *et al.*, 2009] Giuseppe De Giacomo, Maurizio Lenzerini, Antonella Poggi, and Riccardo Rosati.

On instance-level update and erasure in description logic ontologies.

*J. of Logic and Computation, Special Issue on Ontology Dynamics*, 2009.

# References VII

[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.

[Donini *et al.*, 1997] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Werner Nutt.

The complexity of concept languages.

*Information and Computation*, 134:1–58, 1997.

[Donini, 2003] Francesco M. Donini.

Complexity of reasoning.

In Baader et al. [2003], chapter 3, pages 96–136.

[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.

# References VIII

[Grosof *et al.*, 2003] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker.
Description logic programs: Combining logic programs with description logic.
In *Proc. of the 12th Int. World Wide Web Conf. (WWW 2003)*, pages 48–57, 2003.

[Hustadt *et al.*, 2005] Ullrich Hustadt, Boris Motik, and Ulrike Sattler.
Data complexity of reasoning in very expressive description logics.
In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 466–471, 2005.

[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.

[Kontchakov *et al.*, 2008] Roman Kontchakov, Frank Wolter, and Michael Zakharyaschev.
Can you tell the difference between *DL-Lite* ontologies?
In *Proc. of the 11th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2008)*, pages 285–295, 2008.

# References IX

[Kontchakov *et al.*, 2009] R. Kontchakov, L. Pulina, U. Sattler, T. Schneider, P. Selmer, F. Wolter, and M. Zakharyaschev.

Minimal module extraction from DL-Lite ontologies using QBF solvers.

In *Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI 2009)*, pages 836–840, 2009.

[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.

[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.

[Möller and Haarslev, 2003] Ralf Möller and Volker Haarslev.

Description logic systems.

In Baader et al. [2003], chapter 8, pages 282–305.

# References X

[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.

[Pérez-Urbina *et al.*, 2009] Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks.

Tractable query answering and rewriting under description logic constraints.

*J. of Applied Logic*, 2009.

To appear.

[Poggi *et al.*, 2008] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati.

Linking data to ontologies.

*J. on Data Semantics*, X:133–173, 2008.

[Rosati, 2008] Ricardo Rosati.

Finite model reasoning in *DL-Lite*.

In *Proc. of the 5th European Semantic Web Conf. (ESWC 2008)*, 2008.

# References XI

[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.