Semantic Technologies for Data Integration using OWL2 QL

ESWC 2009 Tutorial

Domenico Lembo Riccardo Rosati

Dipartimento di Informatica e Sistemistica
Sapienza Università di Roma, Italy

6th European Semantic Web Conference
Heraklion, Greece – May 31, 2009

Organization of the Tutorial

- Part 1: Ontology-based Data Integration: Models, Languages, and Reasoning
- Part 2: OWL2 QL
- Part 3: Expressive queries and constraints over OWL2 QL ontologies
- Part 4: Tools for Ontology-Based Data Integration

Part 1

Ontology-based Data Integration: Models, Languages, and Reasoning

Outline

1. Introduction
2. Description Logics
3. Querying data through ontologies
4. *DL-Lite*$_{\mathcal{R}}$: an ontology language for accessing data
5. Ontology-based data integration
6. References

Outline

1. Introduction
2. Description Logics
3. Querying data through ontologies
4. *DL-Lite*$_\mathcal{R}$: an ontology language for accessing data
5. Ontology-based data integration
6. References

SAPIENZA
UNIVERSITÀ DI ROMA

Ontologies

Definition

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

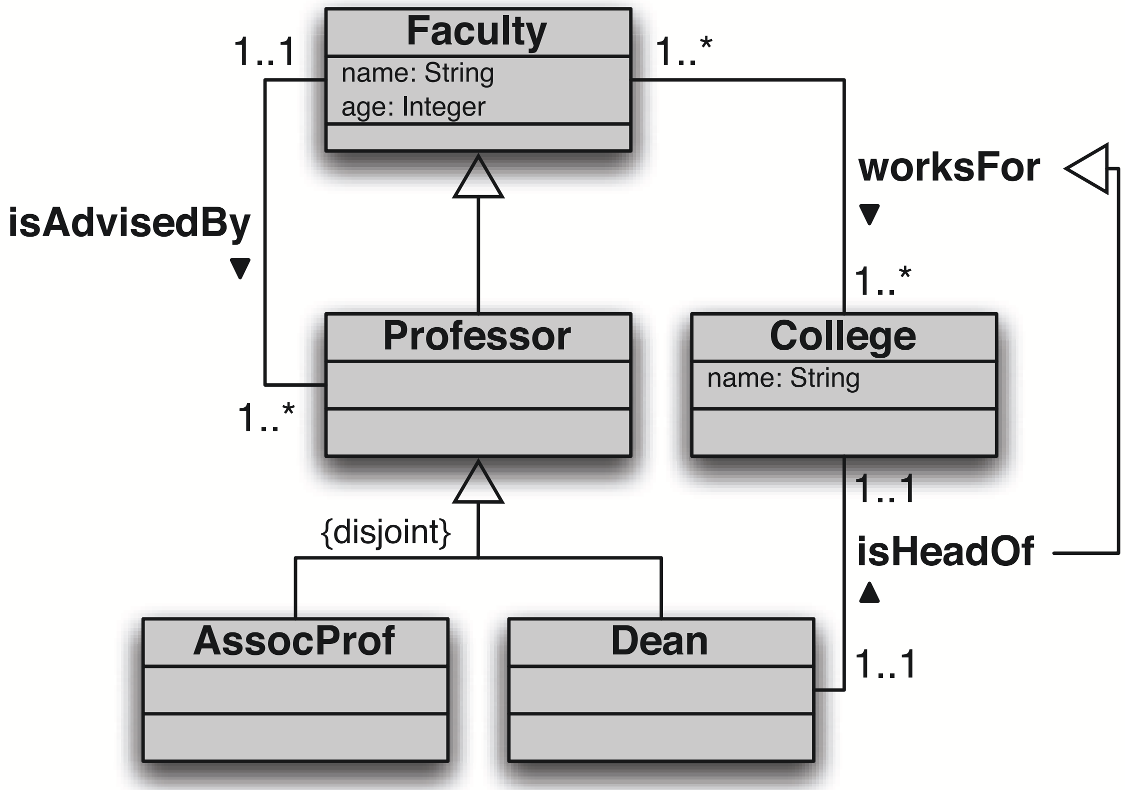
The specification of an ontology comprises several levels:

- **Meta-level**: specifies a set of modeling categories.
- **Intensional level**: specifies a set of conceptual elements (instances of categories) 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.

In this tutorial we focus on the intensional and extensional levels

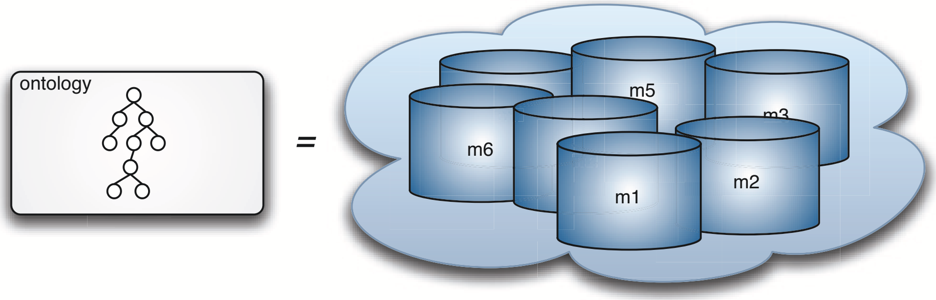
(The intensional level of) an Ontology is typically **rendered as a diagram** (e.g., Semantic Network, Entity-Relationship schema, UML Class Diagram).

Example: ontology rendered as UML Class Diagram



Ontologies and Reasoning

- Ontologies are **logical theories**, and several interpretations may exist that satisfy them (*incomplete information*)



- Reasoning over ontologies amounts to make logical **inference** over them
  - Intensional reasoning: concept/relationship satisfiability, concept/relationship subsumption, etc.
  - Ontology reasoning: ontology satisfiability, instance checking, query answering.

Ontology languages vs. query languages

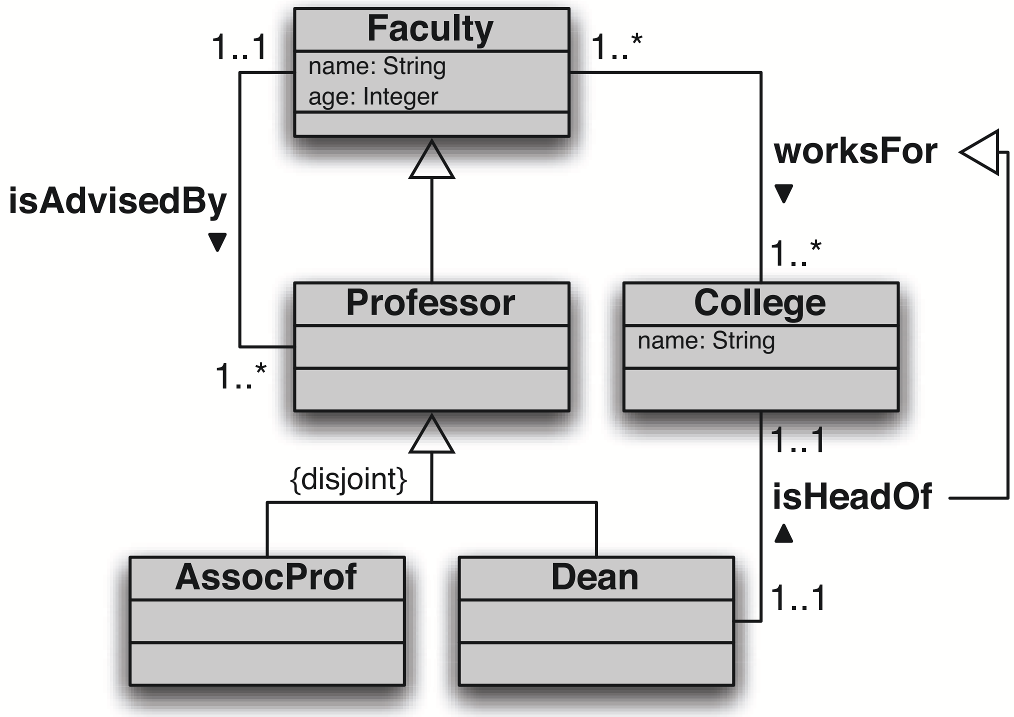
Ontology languages:

- Tailored for capturing intensional relationships.
- 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.

Query languages:

- Allow for general forms of joins.
- May be highly expressive, but computational problems may arise:
  - Full SQL (or equivalently, first-order logic):...
  - ....in the presence of incomplete information, query answering becomes **undecidable** (FOL validity).

Example of query



$$q(nf, af, nd) \leftarrow \exists f, c, d, ad.$$
$$\mathsf{worksFor}(f, c) \wedge \mathsf{isHeadOf}(d, c) \wedge \mathsf{name}(f, nf) \wedge \mathsf{name}(d, nd) \wedge$$
$$\mathsf{age}(f, af) \wedge \mathsf{age}(d, ad) \wedge af = ad$$

Query: return name, age, and name of dean of all faculty that have the same age as their dean.

Ontologies and data

- The best current ontology reasoning systems can deal with a moderately large instance level. $\leadsto 10^4$ individuals *(and this is a big achievement of the last years)!*
- But data of interests in typical information systems (and in data integration) are much **larger**
$\leadsto 10^6 - 10^9$ individuals
- The best technology to deal with large amounts of data are **relational databases**.

Question:

How can we use ontologies together with large amounts of data?

Challenges when integrating data into ontologies

Deal with well-known tradeoff between expressive power of the ontology language and complexity of dealing with (i.e., performing inference over) ontologies in that language.

Requirements come from the specific setting:

- We have to fully take into account the ontology.
  $\leadsto$ **inference**
- We have to deal very large amounts of data.
  $\leadsto$ **relational databases**
- We want flexibility in querying the data.
  $\leadsto$ **expressive query language**
- We want to keep the data in the sources, and not move it around.
  $\leadsto$ **map** data sources to the ontology (Virtual Data Integration)

Questions addressed in this tutorial

1. Which is the "right" **ontology language**?
2. Which is the "right" **query 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 and integration** fully take into account all these issues?

Outline

1. Introduction
2. Description Logics
3. Querying data through ontologies
4. *DL-Lite*$_{\mathcal{R}}$: an ontology language for accessing data
5. Ontology-based data integration
6. References

What are Description Logics?

Description Logics [1] are **logics** specifically designed to represent and reason on structured knowledge:

The domain 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.

Description language

A description language indicates how to form concepts and roles, and is characterized by a set of **constructs** for building **complex concepts** and **roles** starting from atomic ones.

Formal semantics is given in terms of interpretations.

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.

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	$\neg A$	¬Doctor	$\Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$
conjunction	$C \sqcap D$	Hum ⊓ Male	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
(unqual.) exist. res.	$\exists R$	∃hasChild	$\{ o \mid \exists o'.\, (o, o') \in R^{\mathcal{I}} \}$
value restriction	$\forall R.C$	∀hasChild.Male	$\{o \mid \forall o'.\, (o, o') \in R^{\mathcal{I}} \rightarrow o' \in C^{\mathcal{I}}\}$
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.

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).

SAPIENZA
UNIVERSITÀ DI ROMA

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**:

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)
  (**functional** P) (**reflexive** P) $\cdots$

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 knowledge base – 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:

$$\begin{array}{rcl} \mathsf{Father} & \equiv & \mathsf{Human} \sqcap \mathsf{Male} \sqcap \exists \mathsf{hasChild} \\ \mathsf{HappyFather} & \sqsubseteq & \mathsf{Father} \sqcap \forall \mathsf{hasChild}.\mathsf{HappyPerson} \\ \mathsf{HappyAnc} & \sqsubseteq & \forall \mathsf{descendant}.\mathsf{HappyFather} \\ \mathsf{Teacher} & \sqsubseteq & \neg \mathsf{Doctor} \sqcap \neg \mathsf{Lawyer} \end{array}$$

- Inclusion assertions on roles:

$$\mathsf{hasChild} \quad \sqsubseteq \quad \mathsf{descendant}$$

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

ABox membership assertions:

- Teacher($\mathtt{mary}$), hasFather($\mathtt{mary}, \mathtt{john}$), HappyAnc($\mathtt{john}$)

Semantics of a Description Logics knowledge base

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**.

- $A(c)$ is satisfied by $\mathcal{I}$ if $c^{\mathcal{I}} \in A^{\mathcal{I}}$.
- $P(c_1, c_2)$ is satisfied by $\mathcal{I}$ if $(c_1^{\mathcal{I}}, c_2^{\mathcal{I}}) \in P^{\mathcal{I}}$.

SAPIENZA
UNIVERSITÀ DI ROMA

Models of a Description Logics ontology

Model of a DL knowledge base

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 $\mathcal{A}$.

$\mathcal{O}$ is said to be **satisfiable** if it admits a model.

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

Logical implication

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

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., $\mathcal{T} \models C_1 \sqcap C_2 \equiv \bot$.

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

Reasoning over a DL 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 $\mathcal{O}$, 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 $\mathcal{O}$, i.e., whether $\mathcal{O} \models R(c_1, c_2)$.
- **Query Answering:** see later ...

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., [11]).
- 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++, ...) [12].

Relationship between DLs and ontology formalisms

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

DLs vs. OWL

DLs provide the foundations for standard ontology languages.

Different versions of the W3C standard **Web Ontology Language (OWL)** have been defined as syntactic variants of certain DLs:

- **OWL 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.
- **OWL 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.

Description Logics vs. OWL2

- A new version of OWL, **OWL2**, is currently being standardized by the W3C.
- The design aim of OWL2 was to address user requirements for more expressivity of the language, while still preserving decidability of reasoning.
- **OWL2 DL** is a variant of $\mathcal{SROIQ}(D)$, which adds to OWL1 DL several features:
  - qualified number restrictions ($\mathcal{Q}$)
  - regular role hierarchies ($\mathcal{R}$)
  - better treatment of datatypes
- The **OWL2 profiles** (OWL2 EL, OWL2 QL, and OWL2 RL) are variant of specific DLs designed to have tractable reasoning.

DL constructs vs. OWL constructs

OWL construct	DL construct	Example
`ObjectIntersectionOf`	$C_1 \sqcap \cdots \sqcap C_n$	Human $\sqcap$ Male
`ObjectUnionOf`	$C_1 \sqcup \cdots \sqcup C_n$	Doctor $\sqcup$ Lawyer
`ObjectComplementOf`	$\neg C$	$\neg$Male
`ObjectOneOf`	$\{a_1\} \sqcup \cdots \sqcup \{a_n\}$	{john} $\sqcup$ {mary}
`ObjectAllValuesFrom`	$\forall P.C$	$\forall$hasChild.Doctor
`ObjectSomeValuesFrom`	$\exists P.C$	$\exists$hasChild.Lawyer
`ObjectMaxCardinality`	$(\leq n\, P)$	($\leq$ 1 hasChild)
`ObjectMinCardinality`	$(\geq n\, P)$	($\geq$ 2 hasChild)

...

Note: all constructs come also in the `Data...` instead of `Object...` variant.

DL axioms vs. OWL axioms

OWL axiom	DL syntax	Example
`SubClassOf`	$C_1 \sqsubseteq C_2$	Human $\sqsubseteq$ Animal $\sqcap$ Biped
`EquivalentClasses`	$C_1 \equiv C_2$	Man $\equiv$ Human $\sqcap$ Male
`DisjointClasses`	$C_1 \sqsubseteq \neg C_2$	Man $\sqsubseteq$ $\neg$Female
`SameIndividual`	$\{a_1\} \equiv \{a_2\}$	{presBush} $\equiv$ {G.W.Bush}
`DifferentIndividuals`	$\{a_1\} \sqsubseteq \neg\{a_2\}$	{john} $\sqsubseteq$ $\neg${peter}
`SubObjectPropertyOf`	$P_1 \sqsubseteq P_2$	hasDaughter $\sqsubseteq$ hasChild
`EquivalentObjectProperties`	$P_1 \equiv P_2$	hasCost $\equiv$ hasPrice
`InverseObjectProperties`	$P_1 \equiv P_2^-$	hasChild $\equiv$ hasParent$^-$
`TransitiveObjectProperty`	$P^+ \sqsubseteq P$	ancestor$^+$ $\sqsubseteq$ ancestor
`FunctionalObjectProperty`	(**functional** P)	(**functional** hasFather)

...

Outline

1 Introduction

2 Description Logics

3 Querying data through ontologies

4 *DL-Lite*$_{\mathcal{R}}$: an ontology language for accessing data

5 Ontology-based data integration

6 References

Conjunctive queries (CQs)

Which query language to use? we need a decidable query language more expressive than simple concept (or role) expressions.

A **conjunctive query (CQ)** is a first-order query of the form

$$q(\vec{x}) \leftarrow \exists \vec{y}. R_1(\vec{x},\vec{y}) \wedge \cdots \wedge R_k(\vec{x},\vec{y})$$

where each $R_i(\vec{x},\vec{y})$ is an atom using (some of) the **distinguished** variables $\vec{x}$, the **non-distinguished** variables $\vec{y}$, and possibly constants.

We will also use the simpler Datalog notation:

$$q(\vec{x}) \leftarrow R_1(\vec{x},\vec{y}), \ldots, R_k(\vec{x},\vec{y})$$

A union of CQs (UCQ) is e set of CQs.

Note:

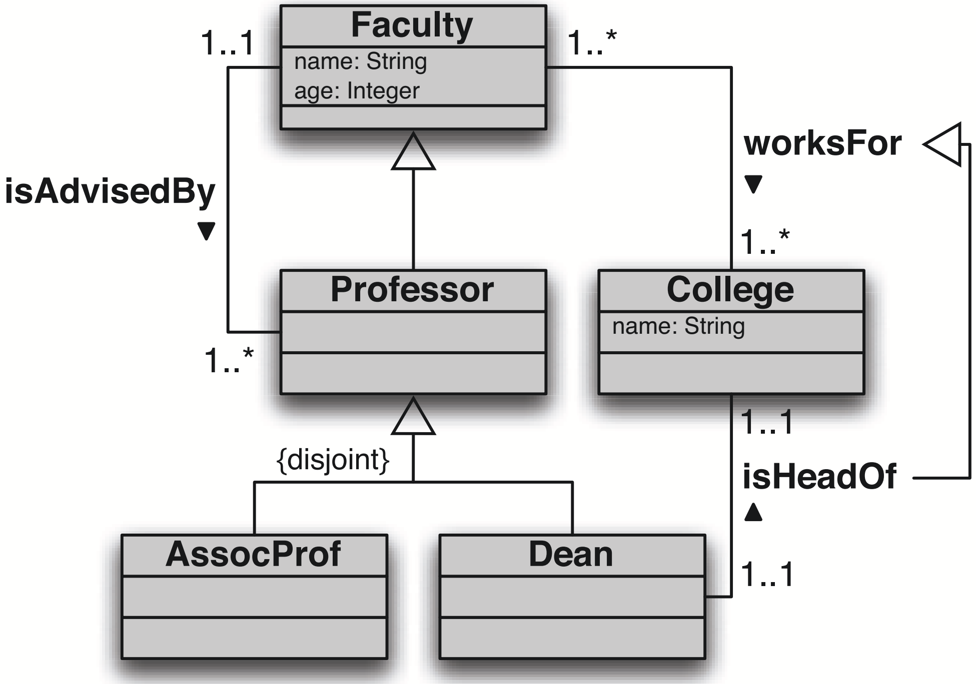
- Correspond to SQL/relational algebra **select-project-join (SPJ) queries** – the most frequently asked queries.
- They can also be written as **SPARQL** queries.

Example of conjunctive query

$$
\begin{array}{rcl}
\text{Professor} & \sqsubseteq & \text{Faculty} \\
\text{AssocProf} & \sqsubseteq & \text{Professor} \\
\text{Dean} & \sqsubseteq & \text{Professor} \\
\exists\text{worksFor} & \sqsubseteq & \text{Faculty} \\
\exists\text{worksFor}^- & \sqsubseteq & \text{College} \\
\text{Faculty} & \sqsubseteq & \exists\text{worksFor} \\
\text{College} & \sqsubseteq & \exists\text{worksFor}^- \\
\exists\text{isHeadOf} & \sqsubseteq & \text{Dean} \\
\exists\text{isHeadOf}^- & \sqsubseteq & \text{College} \\
\text{Dean} & \sqsubseteq & \exists\text{isHeadOf} \\
\text{College} & \sqsubseteq & \exists\text{isHeadOf}^- \\
\text{isHeadOf} & \sqsubseteq & \text{worksFor} \\
\end{array}
$$

(**functional** isHeadOf)
(**functional** isHeadOf$^-$)

$\vdots$



$$
q(nf, af, nd) \leftarrow \exists f, c, d, ad. \; \text{worksFor}(f, c) \wedge \text{isHeadOf}(d, c) \wedge \text{name}(f, nf) \wedge \text{name}(d, nd) \wedge \text{age}(f, af) \wedge \text{age}(d, ad) \wedge af = ad
$$

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 $q(\vec{x}) \leftarrow \exists \vec{y}.\, conj(\vec{x}, \vec{y})$ 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 $\mathcal{O}$.

Note: if q is boolean, i.e., q is an existential sentence, we write $\mathcal{O} \models q$ iff q evaluates to true in every model $\mathcal{I}$ of $\mathcal{O}$, $\mathcal{O} \not\models q$ otherwise.

Data complexity

Various parameters affect the complexity of query answering over an ontology.

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

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

Complexity of query answering in DLs

Problem of rewriting is related to **complexity of query answering**.

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

	Combined complexity	Data complexity
Plain databases	NP-complete	in LogSpace (2)
OWL 2 (and less)	2ExpTime-complete	coNP-hard (1)

(1) Already for a TBox with a single disjunction (2) This is what we need to scale with the data.

Questions

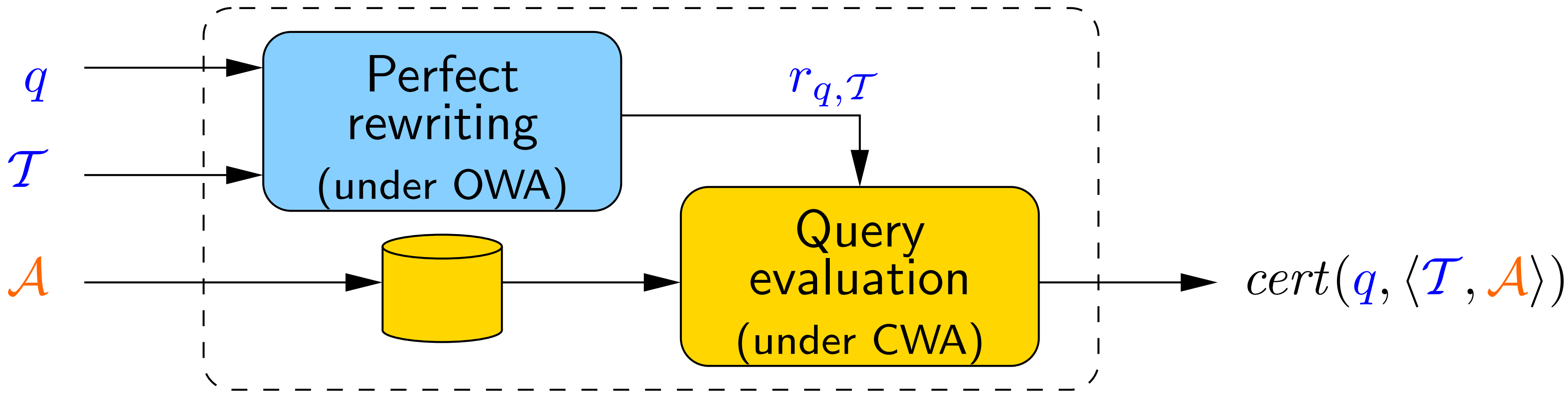
- Can we find interesting families of DLs for which the query answering problem can be solved efficiently (i.e., in LogSpace)?
- If yes, can we leverage relational database technology for query answering?

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 $\mathcal{T}$.

$\leadsto$ 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 $r_{q,\mathcal{T}}$.

Language of the rewriting

The expressiveness of the ontology language affects the **query language into which we are able to rewrite CQs**:

- When we can rewrite into **FOL/SQL** (query answering is **FOL-rewritable**)
  $\rightsquigarrow$ Query evaluation can be done in SQL, i.e., via an **RDBMS** (*Note:* FOL is in LogSpace).
- 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

1 Introduction

2 Description Logics

3 Querying data through ontologies

4 *DL-Lite*$_{\mathcal{R}}$: an ontology language for accessing data

5 Ontology-based data integration

6 References

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 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.
- We present $DL\text{-}Lite_{\mathcal{R}}$, a member of the *DL-Lite* family.
- $DL\text{-}Lite_{\mathcal{R}}$ essentially corresponds to **OWL2 QL**, one of the three candidates **OWL2 Profiles**.
- Extends (the DL fragment of) the ontology language **RDFS**.

DL-Lite$_{\mathcal{A}}$ ontologies

TBox assertions:

- Concept inclusion assertions: $Cl \sqsubseteq Cr$, with:

$$
\begin{array}{rcl}
Cl & \longrightarrow & A \mid \exists Q \\
Cr & \longrightarrow & A \mid \exists Q \mid \neg A \mid \neg\exists Q \\
Q & \longrightarrow & P \mid P^{-}
\end{array}
$$

- Property inclusion assertions: $Q \sqsubseteq R$, with:

$$
R \longrightarrow Q \mid \neg Q
$$

ABox assertions: $A(c)$, $P(c_1, c_2)$, with c_1, c_2 constants

Note: *DL-Lite*$_{\mathcal{R}}$ can be straightforwardly adapted to distinguish also between object and data properties (attributes).

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

Construct	Syntax	Example	Semantics
atomic conc.	A	Doctor	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
exist. restr.	$\exists Q$	$\exists$child$^-$	$\{d \mid \exists e. (d, e) \in Q^{\mathcal{I}}\}$
at. conc. neg.	$\neg A$	$\neg$Doctor	$\Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$
conc. neg.	$\neg \exists Q$	$\neg\exists$child	$\Delta^{\mathcal{I}} \setminus (\exists Q)^{\mathcal{I}}$
atomic role	P	child	$P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
inverse role	P^-	child$^-$	$\{(o, o') \mid (o', o) \in P^{\mathcal{I}}\}$
role negation	$\neg Q$	$\neg$manages	$(\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}) \setminus Q^{\mathcal{I}}$
conc. incl.	$Cl \sqsubseteq Cr$	Father $\sqsubseteq$ $\exists$child	$Cl^{\mathcal{I}} \subseteq Cr^{\mathcal{I}}$
role incl.	$Q \sqsubseteq R$	hasFather $\sqsubseteq$ child$^-$	$Q^{\mathcal{I}} \subseteq R^{\mathcal{I}}$
mem. asser.	$A(c)$	Father(bob)	$c^{\mathcal{I}} \in A^{\mathcal{I}}$
mem. asser.	$P(c_1, c_2)$	child(bob, ann)	$(c_1^{\mathcal{I}}, c_2^{\mathcal{I}}) \in P^{\mathcal{I}}$

DL-Lite$_{\mathcal{R}}$ (as all DLs of the *DL-Lite* family) adopts the Unique Name Assumption (UNA), i.e., different individuals denote different objects (we will come back on this aspect in Part 2 and Part 3).

SAPIENZA UNIVERSITÀ DI ROMA

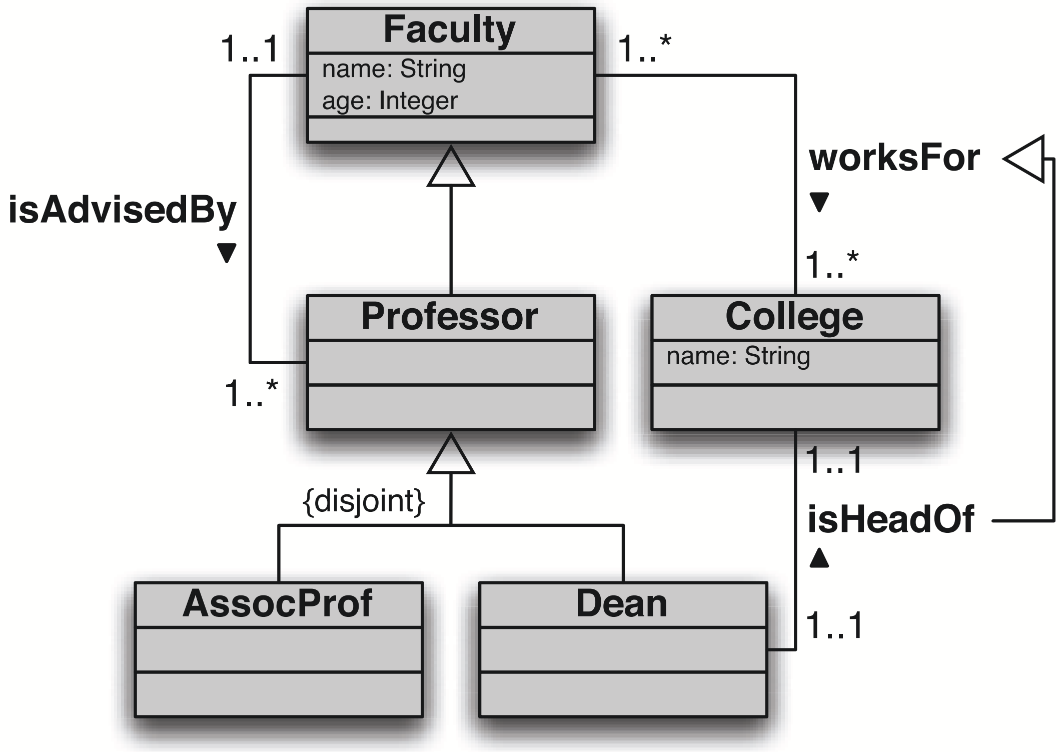
Capturing basic ontology constructs in *DL-Lite*$_\mathcal{R}$

ISA between classes	$A_1 \sqsubseteq A_2$
Disjointness between classes	$A_1 \sqsubseteq \neg A_2$
Domain and range of properties	$\exists P \sqsubseteq A_1 \quad \exists P^- \sqsubseteq A_2$
Mandatory participation *(min card = 1)*	$A_1 \sqsubseteq \exists P \quad A_2 \sqsubseteq \exists P^-$
ISA between properties	$Q_1 \sqsubseteq Q_2$
Disjointness between properties	$Q_1 \sqsubseteq \neg Q_2$

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

Note2: *DL-Lite*$_\mathcal{R}$ cannot capture **functionality** on roles (*max card = 1*)

Example



$$
\begin{array}{rcl}
\mathsf{Professor} & \sqsubseteq & \mathsf{Faculty} \\
\mathsf{AssocProf} & \sqsubseteq & \mathsf{Professor} \\
\mathsf{Dean} & \sqsubseteq & \mathsf{Professor} \\
\exists \mathsf{worksFor} & \sqsubseteq & \mathsf{Faculty} \\
\exists \mathsf{worksFor}^- & \sqsubseteq & \mathsf{College} \\
\mathsf{Faculty} & \sqsubseteq & \exists \mathsf{worksFor} \\
\mathsf{College} & \sqsubseteq & \exists \mathsf{worksFor}^- \\
\exists \mathsf{isHeadOf} & \sqsubseteq & \mathsf{Dean} \\
\exists \mathsf{isHeadOf}^- & \sqsubseteq & \mathsf{College} \\
\mathsf{Dean} & \sqsubseteq & \exists \mathsf{isHeadOf} \\
\mathsf{College} & \sqsubseteq & \exists \mathsf{isHeadOf}^- \\
\mathsf{isHeadOf} & \sqsubseteq & \mathsf{worksFor} \\
 & \vdots &
\end{array}
$$

Query answering in *DL-Lite*$_{\mathcal{R}}$

- We study answering of (U)CQs over *DL-Lite*$_{\mathcal{R}}$ ontologies via query rewriting.
- We first consider query answering over **satisfiable ontologies**, i.e., that admit at least one model.
- Then, we show how to exploit query answering over satisfiable ontologies to establish ontology satisfiability.

Remark

we call **positive inclusions (PIs)** assertions of the form

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

whereas we call negative inclusions (NIs) assertions of the form

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

Query answering in *DL-Lite*$_\mathcal{R}$ (cont'd)

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 $\mathcal{T}$, **reformulate** q as a **union** $r_{q,\mathcal{T}}$ **of CQs**.
2. Evaluate $r_{q,\mathcal{T}}$ directly over $\mathcal{A}$ managed in **secondary storage via a RDBMS**.

Correctness of this procedure shows FOL-rewritability of query answering in *DL-Lite*$_\mathcal{R}$
$\leadsto$ Query answering over *DL-Lite*$_\mathcal{R}$ ontologies can be done using RDMBS technology.

Query answering in *DL-Lite*$_\mathcal{R}$: Query rewriting

Intuition: Use the PIs as basic rewriting rules

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

$$\mathsf{AssProfessor} \sqsubseteq \mathsf{Professor}$$

$$\text{as a logic rule:} \quad \mathsf{Professor}(z) \leftarrow \mathsf{AssProfessor}(z)$$

Basic rewriting step:

when the atom unifies with the **head** of the rule.

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

Towards the computation of the perfect rewriting, we add to the input query above the following query

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

We say that the PI $\mathsf{AssProfessor} \sqsubseteq \mathsf{Professor}$ **applies** to the atom $\mathsf{Professor}(x)$.

Query answering in *DL-Lite*$_{\mathcal{R}}$: Query rewriting (cont'd)

Consider now the query

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

$$\mathsf{Professor} \sqsubseteq \exists\mathsf{teaches}$$

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

We add to the reformulation the query

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

Conversely, for the query

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

$$\mathsf{Professor} \sqsubseteq \exists \mathsf{teaches}$$

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

$\mathsf{teaches}(x, \mathtt{databases})$ does not unify with $\mathsf{teaches}(z_1, z_2)$, since the **existentially quantified variable** z_2 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**

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

Query answering in $DL\text{-}Lite_{\mathcal{R}}$: Query rewriting (cont'd)

An analogous behavior with join variables

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

$$\mathsf{Professor} \sqsubseteq \exists\mathsf{teaches}$$

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

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

Conversely, the PI

$$\exists\mathsf{teaches}^- \sqsubseteq \mathsf{Course}$$

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

applies to the atom $\mathsf{Course}(y)$.

We add to the perfect rewriting the query

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

Query answering in $DL\text{-}Lite_{\mathcal{R}}$: Query rewriting (cont'd)

We now have the query

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

The PI Professor $\sqsubseteq$ $\exists$teaches

as a logic rule: $teaches(z_1, z_2) \leftarrow Professor(z_1)$

does not apply to $teaches(x, y)$ nor $teaches(z, y)$, since y is in join.

However, we can transform the above query by unifying the atoms $teaches(x, y)$, $teaches(z_1, y)$. This rewriting step is called **reduce**, and produces the following query

$$q(x) \leftarrow teaches(x, y)$$

We can now apply the PI above, and add to the reformulation the query

$$q(x) \leftarrow Professor(x)$$

Answering by rewriting in *DL-Lite*$_{\mathcal{R}}$: The algorithm

1. Rewrite the CQ q into a UCQs: apply to q in all possible ways the PIs in the TBox $\mathcal{T}$.
2. This corresponds to exploiting ISAs, role typings, and mandatory participations to obtain new queries that could contribute to the answer.
3. Unifying atoms can make applicable rules that could not be applied otherwise.
4. The UCQs resulting from this process is the **perfect rewriting** $r_{q,\mathcal{T}}$.
5. $r_{q,\mathcal{T}}$ is then **encoded into SQL** and evaluated over $\mathcal{A}$ managed in **secondary storage via a RDBMS**, to return the set $cert(q, \mathcal{O})$.

Notice that NIs play no role in the process above: **when the ontology is satisfiable, we can ignore NIs and answer queries as NIs were not specified in $\mathcal{T}$.**

Query answering in $\textit{DL-Lite}_{\mathcal{R}}$: Example

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

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

Perfect Rewriting:
$$
\begin{aligned}
\text{q}(x) &\leftarrow \text{teaches}(x, y), \text{Course}(y)\\
\text{q}(x) &\leftarrow \text{teaches}(x, y), \text{teaches}(z, y)\\
\text{q}(x) &\leftarrow \text{teaches}(x, z)\\
\text{q}(x) &\leftarrow \text{Professor}(x)
\end{aligned}
$$

ABox: $\text{teaches}(\texttt{John}, \texttt{databases})$
$\text{Professor}(\texttt{Mary})$

It is easy to see that the evaluation of $r_{q,\mathcal{T}}$ over $\mathcal{A}$ in this case produces the set $\{\texttt{John}, \texttt{Mary}\}$.

Satisfiability of ontologies with only PIs

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

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

Theorem

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

DL-Lite$_{\mathcal{R}}$ ontologies

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

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

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

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

- Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, and $\mathcal{T}_P$ be the set of PIs in $\mathcal{T}$.
- For each NI N between concepts (resp. roles) in $\mathcal{T}$, we ask $\langle \mathcal{T}_P, \mathcal{A} \rangle$ if there exists some individual (resp. pair of individuals) that contradicts N, i.e., we pose over $\langle \mathcal{T}_P, \mathcal{A} \rangle$ a boolean CQ q_N such that $\langle \mathcal{T}_P, \mathcal{A} \rangle \models q_N$ **iff** $\langle \mathcal{T}_P \cup \{N\}, \mathcal{A} \rangle$ **is unsatisfiable**.
- To verify if $\langle \mathcal{T}_P, \mathcal{A} \rangle \models q_N$ we use the query rewriting algorithm for CQs over satisfiable *DL-Lite*$_{\mathcal{R}}$ ontologies, i.e., **we compute the perfect rewriting** $r_{q_N, \mathcal{T}_P}$, and evaluate it (in fact its SQL encoding) over $\mathcal{A}$ seen as a database.

$\mathcal{O}$ is unsatisfiable iff there exists a NI $N \in \mathcal{T}$ such that the evaluation of $r_{q_N, \mathcal{T}_P}$ over $\mathcal{A}$ seen as a database returns *true*.

Satisfiability of a *DL-Lite*$_{\mathcal{R}}$ ontology is reduced to evaluation of a UCQs over $\mathcal{A}$. $\leadsto$ Ontology satisfiability in *DL-Lite*$_{\mathcal{R}}$ can be done using RDMBS technology.

Example

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

NI N: Professor $\sqsubseteq \neg$Student

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

Perfect Rewriting $r_{q,\mathcal{T}_P}$: $q() \leftarrow \mathsf{Student}(x), \mathsf{Professor}(x)$
$q() \leftarrow \mathsf{Student}(x), \mathsf{teaches}(x, y)$

ABox $\mathcal{A}$: teaches(John, databases)
Student(John)

It is easy to see that $r_{q,\mathcal{T}_P}$ evaluates to *true* over $\mathcal{A}$, and that therefore $\mathcal{O}$ is unsatisfiable.

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

Ontology satisfiability and all classical DL reasoning tasks are:

- Efficiently tractable in the size of TBox (i.e., PTIME).
- Very efficiently tractable in the size of the ABox (i.e., LOGSPACE).

Query answering for CQs and UCQs is:

- PTIME in the size of TBox.
- LOGSPACE 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.

Can we go beyond *DL-Lite*$_{\mathcal{R}}$?

By adding essentially any other DL construct, e.g., union ($\sqcup$), value restriction ($\forall R.C$), etc., without some limitations we lose these nice computational properties (see later).

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

	lhs	rhs	funct.	Prop. incl.	Data complexity of query answering
0	*DL-Lite*$_\mathcal{R}$		$-$	$\surd$	in LogSpace
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**

- Giving up property inclusions from *DL-Lite*$_\mathcal{R}$ allows for having functional roles, remaining in LogSpace (cf. *DL-Lite*$_\mathcal{F}$). Prop. incl. and funct. can be also used together (cf. *DL-Lite*$_\mathcal{A}$), *provided that functional properties are not specialized*.
- 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! $\leadsto$ **No** hope of including **covering constraints**.

Sapienza Università di Roma

Outline

1. Introduction
2. Description Logics
3. Querying data through ontologies
4. *DL-Lite*$_{\mathcal{R}}$: an ontology language for accessing data
5. Ontology-based data integration
6. References

Data integration

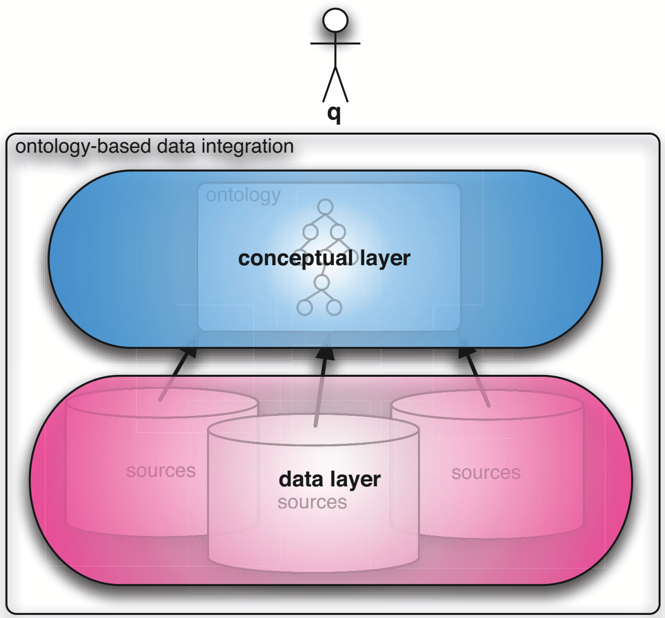
Data integration is the problem of providing unified and transparent access to a set of autonomous and heterogeneous sources.

From [Bernstein & Haas, CACM Sept. 2008]:

- Large enterprises spend a great deal of time and money on information integration (e.g., 40% of information-technology shops' budget).
- Market for data integration software 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 (Apr. 2008)]
- Data integration is a large and growing part of science, engineering, and biomedical computing.

Ontology-based data integration: conceptual layer & data layer

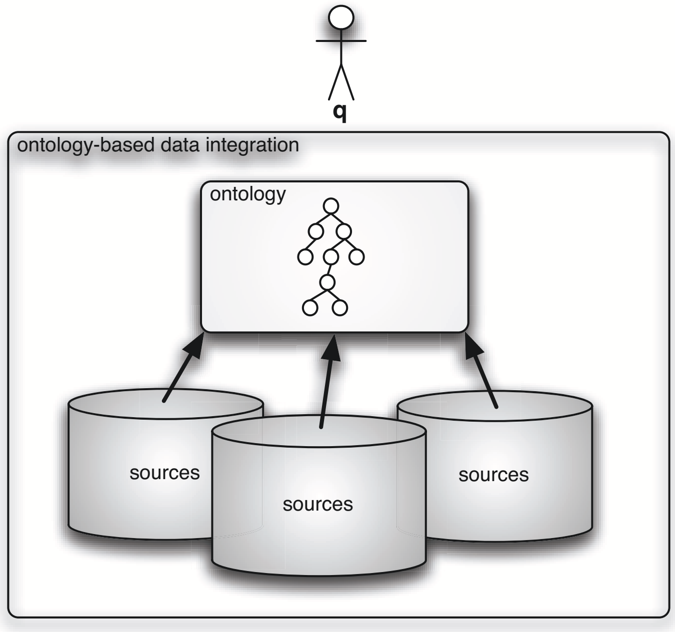
Ontology-based data integration is based on the idea of decoupling information access from data storage.



Clients access only the **conceptual layer** ... while the **data layer**, hidden to clients, manages the data.

$\rightsquigarrow$ *Technological concerns (and changes) on the managed data become fully transparent to the clients.*

Ontology-based data integration: architecture

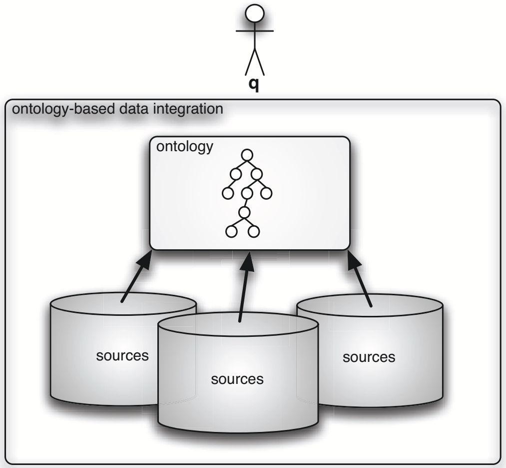


Based on three main components:

- **Ontology**, used as the conceptual layer to give clients a unified conceptual "global view" of the data.
- **Data sources**, these are external, independent, heterogeneous, multiple information systems.
- **Mappings**, which semantically link data at the sources with the ontology *(key issue!)*

Ontology-based data integration: the conceptual layer

The ontology is used as the conceptual layer, to give clients a unified conceptual global view of the data.

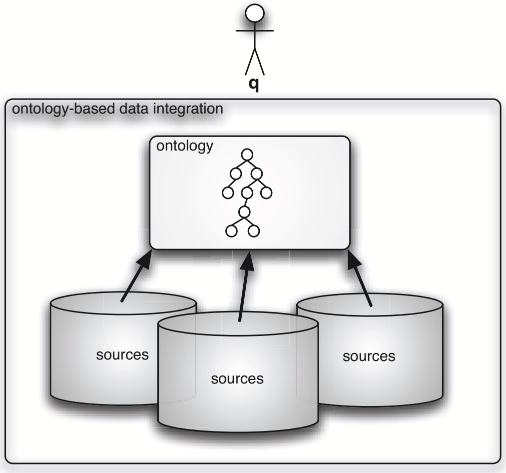


Note: in standard information systems, UML Class Diagram or ER is used at **design time**, ...

... here we use ontologies at **runtime**!

Ontology-based data integration: the sources

Data sources are external, independent, heterogeneous, multiple information systems.

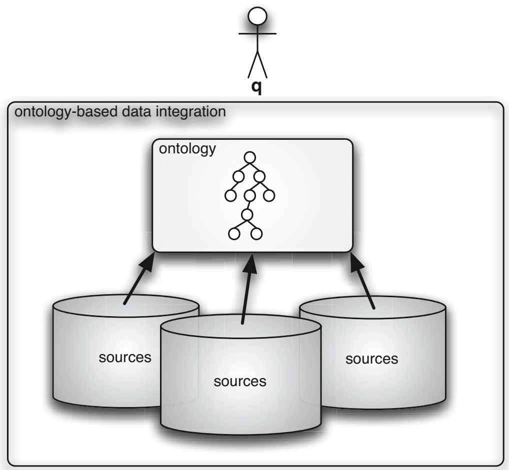


By now we have industrial solutions for:

- Distributed database systems & Distributed query optimization
- Tools for source wrapping
- Systems for database federation, e.g., IBM Information Integrator

Ontology-based data integration: the sources

Data sources are external, independent, heterogeneous, multiple information systems.



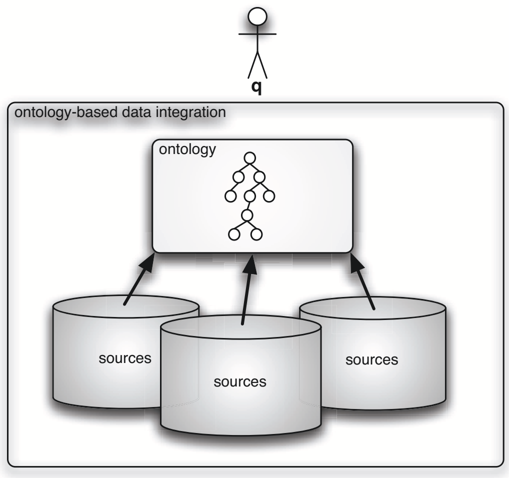
Based on these industrial solutions we can:

1. Wrap the sources and see all of them as relational databases.
2. Use federated database tools to see the multiple sources as a single one.

$\rightsquigarrow$ We can see the sources as a single (remote) relational database.

Ontology-based data integration: mappings

Mappings semantically link data at the sources with the ontology.



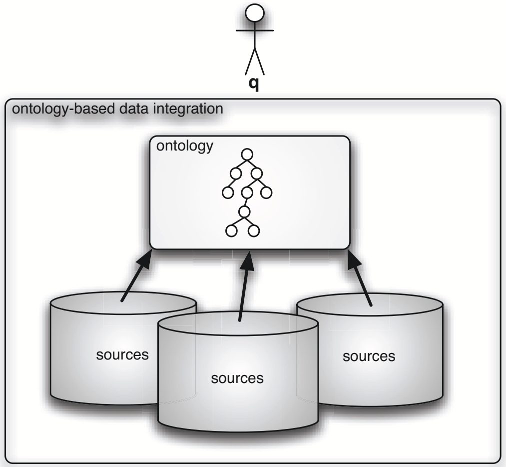
Scientific literature on data integration in databases has shown that ...

... generally we cannot simply **map** single relations to single elements of the global view (the ontology) ...

... we need to rely on **queries**!

Ontology-based data integration: mappings

Mappings semantically link data at the sources with the ontology.



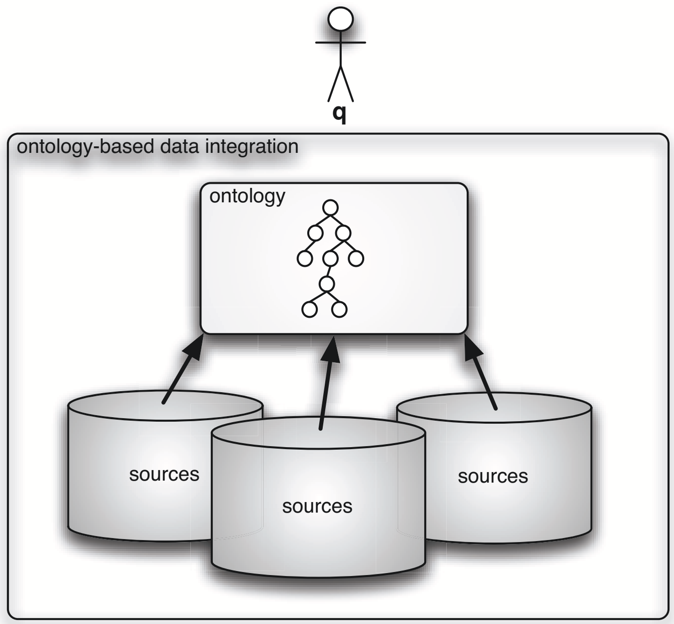
Several general forms of mappings based on queries have been considered:

- GAV: map a query over the source to an element in the global view
  – most used form of mappings
- LAV: map a relation in the source to a query over the global view
  – mathematically elegant, but difficult to use in practice (data in the sources are not clean enough!)
- GLAV: map a query over the sources to a query over the global view
  – the most general form of mappings

This is a key issue (more on this later).

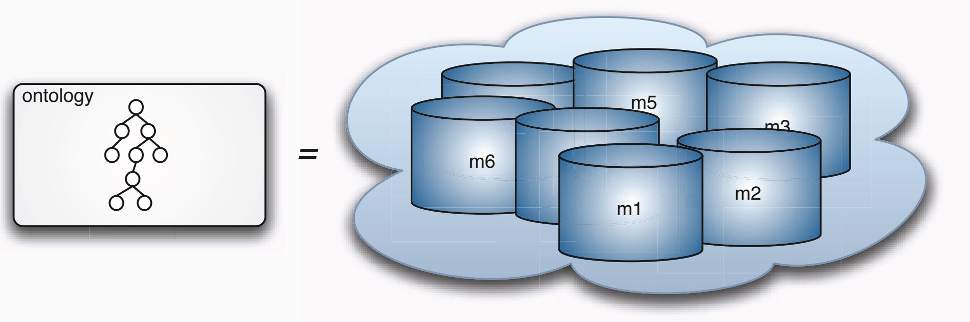
Ontology-based data integration: incomplete information

It is assumed, even in standard data integration, that the information that the global view has on the data is incomplete!



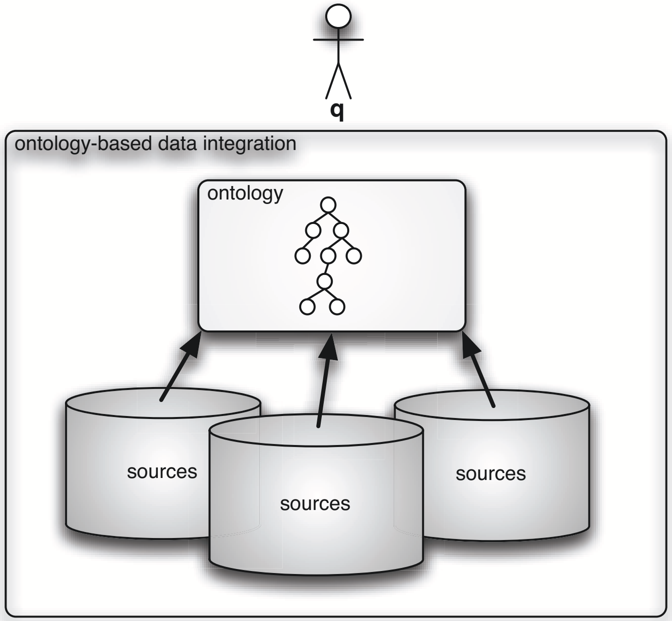
Important

Ontologies are logical theories $\rightsquigarrow$ they are perfectly suited to deal with **incomplete information**!



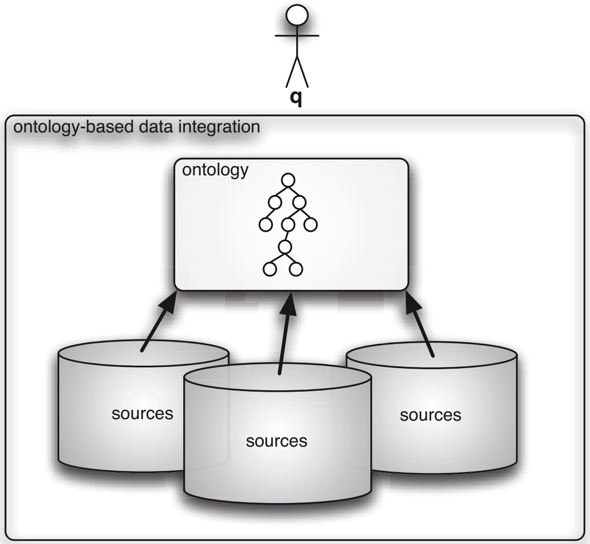
- Query answering amounts to compute **certain answers**, given the global view, the mapping and the data at the sources ...
- ... but query answering may be costly in ontologies (even without mapping and sources).

Ontology-based data integration: the *DL-Lite* solution



- We require the data sources to be **wrapped** and presented as relational sources. $\leadsto$ *"standard technology"*
- We make use of a **data federation tool**, such as IBM Information Integrator, to present the yet to be (semantically) integrated sources as a single relational database. $\leadsto$ *"standard technology"*
- We make use of the **DL-Lite** technology presented above for the conceptual view on the data, to **exploit effectiveness of query answering**. $\leadsto$ *"new technology"*

Ontology-based data integration: the *DL-Lite* solution



Are we done? Not yet!

- The (federated) source database is **external** and **independent** from the conceptual view (the ontology).
- **Mappings** relate information in the sources to the ontology. $\leadsto$ sort of virtual ABox

  We use GAV (global-as-view) mappings: the result of an (arbitrary) SQL query on the source database is considered a (partial) extension of a concept/role.
- Moreover, we properly deal with the notorious **impedance mismatch problem**!

Impedance mismatch problem

The impedance mismatch problem

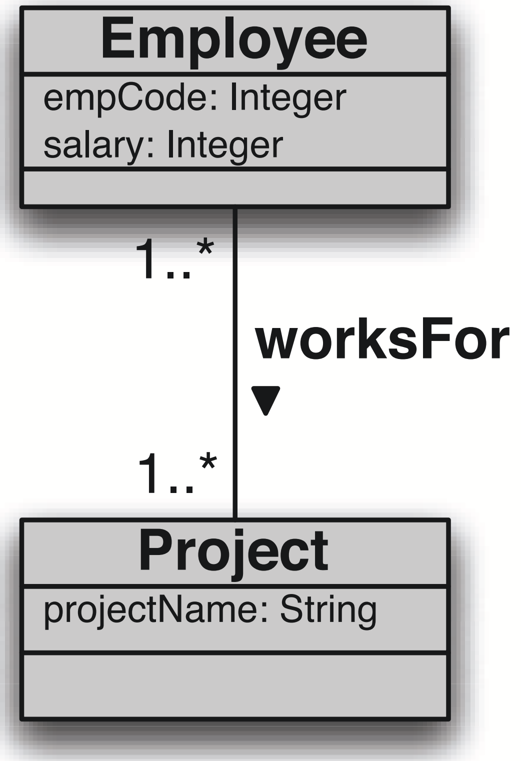
- In **relational databases**, information is represented in forms of tuples of **values**.
- In **ontologies** (or more generally object-oriented systems or conceptual models), information is represented using both **objects** and values ...
  - ... with objects playing the main role, ...
  - ... and values a subsidiary role as fillers of object's attributes.

$\leadsto$ *How do we reconcile these views?*

Solution: We need **constructors** to create objects of the ontology out of tuples of values in the database.

Note: from a formal point of view, such constructors can be simply Skolem functions!

Impedance mismatch – Example



Actual data is stored in a DB:

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

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

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

...

From the domain analysis it turns out that:

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

pers and **proj** are Skolem functions.

If VRD56B25 is a *SSN*, then **pers**(VRD56B25) is an **object term** denoting a person.

Impedance mismatch: the technical solution

Creating object identifiers

- Let Γ_V be the alphabet of constants (values) appearing in the sources.
- We introduce an alphabet Λ of **function symbols**, each with an associated arity, specifying the number of arguments it accepts.
- To denote objects, i.e., instances of concepts in the ontology, we use **object terms** of the form $\mathbf{f}(d_1, \ldots, d_n)$, with $\mathbf{f} \in \Lambda$ of arity n, and each d_i a value constant in Γ_V.

$\leadsto$ *No confusion between the values stored in the database and the terms denoting objects.*

Formalization of ontology with mappings to data sources

An **ontology with mappings** is characterized by a triple $\mathcal{O}_m = \langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$ such that:

- $\mathcal{T}$ is a TBox;
- $\mathcal{S}$ is a (federated) relational database representing the sources;
- $\mathcal{M}$ is a set of **mapping assertions**, each one of the form*

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

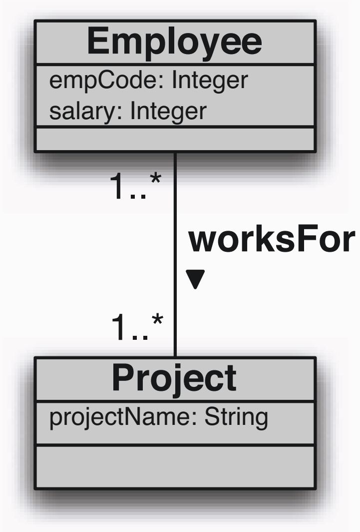
  where

  - $\Phi(\vec{x})$ is an arbitrary SQL query over $\mathcal{S}$, returning attributes $\vec{x}$
  - $\Psi(f(\vec{x}), \vec{x})$ is (the body of) a conjunctive query over $\mathcal{T}$ **without non-distinguished variables**, whose variables, possibly occurring in terms, i.e., $f(\vec{x})$, are from $\vec{x}$.

* Note: this is a form of GAV mapping

Ontology with mappings – Example

TBox $\mathcal{T}$ (UML)



federated schema of the DB $\mathcal{S}$

$D_1[SSN: \text{String}, PrName: \text{String}]$
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

. . .

Mapping $\mathcal{M}$

M_1:
```
SELECT SSN, PrName
FROM D1
```
$\leadsto$ Employee(**pers**(*SSN*)),
Project(**proj**(*PrName*)),
projectName(**proj**(*PrName*), *PrName*),
workFor(**pers**(*SSN*), **proj**(*PrName*))

M_2:
```
SELECT SSN, Salary
FROM D2, D3
WHERE D2.Code = D3.Code
```
$\leadsto$ Employee(**pers**(*SSN*)),
salary(**pers**(*SSN*), *Salary*)

Semantics

Def.: Semantics of mappings

We say that $\mathcal{I}= (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ satisfies $\Phi(\vec{x}) \leadsto \Psi(f(\vec{x}), \vec{x})$ wrt a database $\mathcal{S}$, if for every tuple of values $\vec{v}$ in the answer of the SQL query $\Phi(\vec{x})$ over $\mathcal{S}$, and for each ground atom X in $\Psi(f(\vec{v}), \vec{v})$, we have that:

- if X has the form $A(s)$, then $s^{\mathcal{I}} \in A^{\mathcal{I}}$;
- if X has the form $P(s_1, s_2)$, then $(s_1^{\mathcal{I}}, s_2^{\mathcal{I}}) \in P^{\mathcal{I}}$.

Def.: Semantics of ontologies with mappings

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

- $\mathcal{I}$ is a model of $\mathcal{T}$;
- $\mathcal{I}$ satisfies $\mathcal{M}$ wrt $\mathcal{S}$, i.e., satisfies every assertion in $\mathcal{M}$ wrt $\mathcal{S}$.

Def.: The certain answers to $q(\vec{x})$ over $\mathcal{O}_m = \langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$...

... denoted $cert(q, \mathcal{O}_m)$, are the **tuples $\vec{c}$ of constants of $\mathcal{S}$** such that $\vec{c} \in q^{\mathcal{I}}$, for every model $\mathcal{I}$ of $\mathcal{O}_m$.

DL-Lite$_{\mathcal{R}}$ query answering for data integration

Given a (U)CQ q and $\mathcal{O}_m = \langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$ (assumed satisfiable, i.e., there exists at least one model for $\mathcal{O}_m$), we compute $cert(q, \mathcal{O}_m)$ as follows:

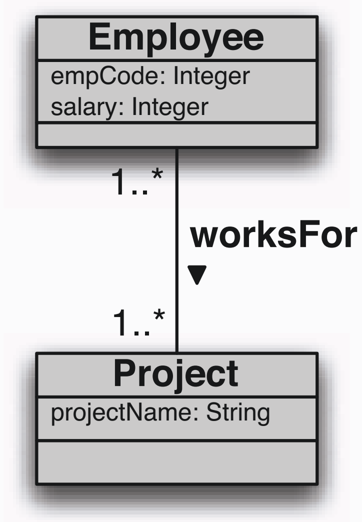
1. Using $\mathcal{T}$, **reformulate** CQ q as a union $r_{q,\mathcal{T}}$ of CQs.
2. Using $\mathcal{M}$, **unfold** $r_{q,\mathcal{T}}$ to obtain a union $unfold(r_{q,\mathcal{T}})$ of CQs.
3. **Evaluate** $unfold(r_{q,\mathcal{T}})$ directly over $\mathcal{S}$ using RDBMS technology.

Correctness of this algorithm shows FOL-reducibility of query answering.
$\rightsquigarrow$ Query answering can again be done using **RDBMS technology**.

SAPIENZA
UNIVERSITÀ DI ROMA

Example – query rewriting

TBox $\mathcal{T}$ (UML)



TBox $\mathcal{T}$ (*DL-Lite*$_\mathcal{R}$)

$$
\begin{array}{rcl}
\text{Employee} & \sqsubseteq & \exists\text{worksFor} \\
\exists\text{worksFor} & \sqsubseteq & \text{Employee} \\
\exists\text{worksFor}^- & \sqsubseteq & \text{Project} \\
\text{Project} & \sqsubseteq & \exists\text{worksFor}^- \\
& \vdots &
\end{array}
$$

Consider the query $q(x) \leftarrow \text{worksFor}(x, y)$

the perfect rewriting is

$$
\begin{array}{rcccl}
r_{q,\mathcal{T}} & = & q(x) & \leftarrow & \text{worksFor}(x, y) \\
& & q(x) & \leftarrow & \text{Employee}(x)
\end{array}
$$

Example – splitting the mapping

To compute $unfold(r_{q,\mathcal{T}})$, we first **split** $\mathcal{M}$ as follows (always possible, since queries in the right-hand side of assertions in $\mathcal{M}$ are without non-distinguished variables):

$M_{1,1}$: `SELECT SSN, PrName FROM D`$_1$ $\leadsto$ Employee(**pers**(*SSN*))

$M_{1,2}$: `SELECT SSN, PrName FROM D`$_1$ $\leadsto$ Project(**proj**(*PrName*))

$M_{1,3}$: `SELECT SSN, PrName FROM D`$_1$ $\leadsto$ projectName(**proj**(*PrName*), *PrName*)

$M_{1,4}$: `SELECT SSN, PrName FROM D`$_1$ $\leadsto$ workFor(**pers**(*SSN*), **proj**(*PrName*))

$M_{2,1}$: `SELECT SSN, Salary FROM D`$_2$`, D`$_3$ `WHERE D`$_2$`.Code = D`$_3$`.Code` $\leadsto$ Employee(**pers**(*SSN*))

$M_{2,2}$: `SELECT SSN, Salary FROM D`$_2$`, D`$_3$ `WHERE D`$_2$`.Code = D`$_3$`.Code` $\leadsto$ salary(**pers**(*SSN*), *Salary*)

Example – unfolding

Then, we unify each atom of the query

$$
\begin{array}{rcrcl}
r_{q,\mathcal{T}} & = & q(x) & \leftarrow & \mathsf{worksFor}(x, y) \\
 & & q(x) & \leftarrow & \mathsf{Employee}(x)
\end{array}
$$

with the right-hand side of the assertion in the split mapping, and substitute such atom with the left-hand side of the mapping

$$
\begin{array}{rcl}
q(\mathbf{pers}(\mathit{SSN})) & \leftarrow & \texttt{SELECT SSN, PrName} \\
 & & \texttt{FROM } \mathtt{D}_1 \\
q(\mathbf{pers}(\mathit{SSN})) & \leftarrow & \texttt{SELECT SSN, Salary} \\
 & & \texttt{FROM } \mathtt{D}_2\texttt{, } \mathtt{D}_3 \\
 & & \texttt{WHERE } \mathtt{D}_2\texttt{.CODE = } \mathtt{D}_3\texttt{.CODE}
\end{array}
$$

The construction of object terms can be pushed into the SQL query, by resorting to SQL functions to manipulate strings (e.g., string concat).

Example – SQL query over the source database

```
SELECT concat(concat('pers (',SSN),')')
FROM D1
UNION
SELECT concat(concat('pers (',SSN),')')
FROM D2, D3
WHERE D2.Code = D3.Code
```

Computational complexity of query answering

Theorem

Query answering in a *DL-Lite*$_{\mathcal{R}}$ ontology with mappings $\mathcal{O} = \langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$ 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** $\mathcal{S}$, in fact FOL-rewritable.

Can we move to LAV or GLAV mappings?
Yes, but we have to **strongly limit** *the form of the queries in the mapping (essentially CQs over both the sources and the ontology), if we want to stay in* LOGSPACE.

Outline

1. Introduction
2. Description Logics
3. Querying data through ontologies
4. *DL-Lite*$_{\mathcal{R}}$: an ontology language for accessing data
5. Ontology-based data integration
6. References

SAPIENZA UNIVERSITÀ DI ROMA

References I

[1] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors.
The Description Logic Handbook: Theory, Implementation and Applications.
Cambridge University Press, 2003.

[2] D. Berardi, D. Calvanese, and G. De Giacomo.
Reasoning on UML class diagrams.
Artificial Intelligence, 168(1–2):70–118, 2005.

[3] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, and R. Rosati.
Linking data to ontologies: The description logic *DL-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.

References II

[4] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, R. Rosati, and M. Ruzzi.

Data integration through *DL-Lite*$_{\mathcal{A}}$ ontologies.

In K.-D. Schewe and B. Thalheim, editors, *Revised Selected Papers of the 3rd Int. Workshop on Semantics in Data and Knowledge Bases (SDKB 2008)*, volume 4925 of *Lecture Notes in Computer Science*, pages 26–47. Springer, 2008.

[5] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. 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.

[6] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati.

DL-Lite: Tractable description logics for ontologies.

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

References III

[7] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. 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.

[8] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati.
Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family.
J. of Automated Reasoning, 39(3):385–429, 2007.

[9] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati.
Path-based identification constraints in description logics.
In *Proc. of the 11th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2008)*, pages 231–241, 2008.

References IV

[10] D. Calvanese and M. Rodríguez.
An extension of DIG 2.0 for handling bulk data.
In *Proc. of the 3rd Int. Workshop on OWL: Experiences and Directions (OWLED 2007)*, volume 258 of *CEUR Electronic Workshop Proceedings*, `http://ceur-ws.org/`, 2007.

[11] F. M. Donini.
Complexity of reasoning.
In Baader et al. [1], chapter 3, pages 96–136.

[12] R. Möller and V. Haarslev.
Description logic systems.
In Baader et al. [1], chapter 8, pages 282–305.

[13] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati.
Linking data to ontologies.
J. on Data Semantics, X:133–173, 2008.

References V

[14] A. Poggi, M. Rodriguez, and M. Ruzzi.

Ontology-based database access with DIG-Mastro and the OBDA Plugin for Protégé.

In K. Clark and P. F. Patel-Schneider, editors, *Proc. of the 4th Int. Workshop on OWL: Experiences and Directions (OWLED 2008 DC)*, 2008.

[15] M. Rodriguez-Muro, L. Lubyte, and D. Calvanese.

Realizing ontology based data access: A plug-in for Protégé.

In *Proc. of the 24th Int. Conf. on Data Engineering Workshops (ICDE 2008)*, pages 286–289, 2008.

Semantic Technologies for Ontology-based Data Integration Using OWL2 QL

ESWC 2009 tutorial

Part 2
OWL2 QL

Domenico Lembo, Riccardo Rosati

Dipartimento di Informatica e Sistemistica
Sapienza Università di Roma, Italy

SAPIENZA
UNIVERSITÀ DI ROMA

Overview

1. limitations of OWL
2. OWL2 profiles
3. OWL2 QL

Limitations of OWL DL

- OWL DL (as well as OWL Lite) has inherently intractable worst-case complexity...
  - both with respect to the size of the TBox (schema complexity) $\Rightarrow$ EXPTIME/NEXPTIME-hard
  - and with respect to the size of the ABox (instance/data complexity) $\Rightarrow$ coNP-hard
- this indicates (at the theoretical level) that reasoning over OWL DL ontologies cannot scale up
- what about reasoning in **real ontologies** using **real OWL reasoners**?

Limitations of OWL DL reasoners

performance of OWL-DL reasoners:

- “practically good” for the intensional level
  - the size of a TBox is not likely to scale up too much
- not very good for the extensional level
  - unable to handle instances (ABoxes) of (very) large size...
- ...especially for answering expressive queries (e.g., conjunctive queries)
- very recent efforts to fill this gap in OWL reasoners (e.g., RacerPro)

Handling very large ABoxes

why are these tools so bad with (very) large ABoxes?

two main reasons:

- current algorithms are mainly derived by algorithms defined for purely intensional tasks
  - no real optimization for ABox services
- these algorithms work in main memory
  - bottleneck for very large instances
- query answering is not a standard reasoning task in DL
  - systems have been optimized for standard tasks (consistency, concept subsumption, instance checking)

Limitations of OWL DL

- how to overcome these limitations if we want to build data-intensive Semantic Web applications?
- possible solution: limit the expressive power of the ontology language
  - the idea is sacrifice part of the expressiveness of the ontology language to have more efficient ontology tools
- within the OWL2 initiative, this idea has been formalized through the so-called **OWL2 profiles**

OWL2 profiles

OWL2 profiles = three “tractable fragments” of OWL2 DL:

- **OWL2 EL**
- **OWL2 QL**
- **OWL2 RL**

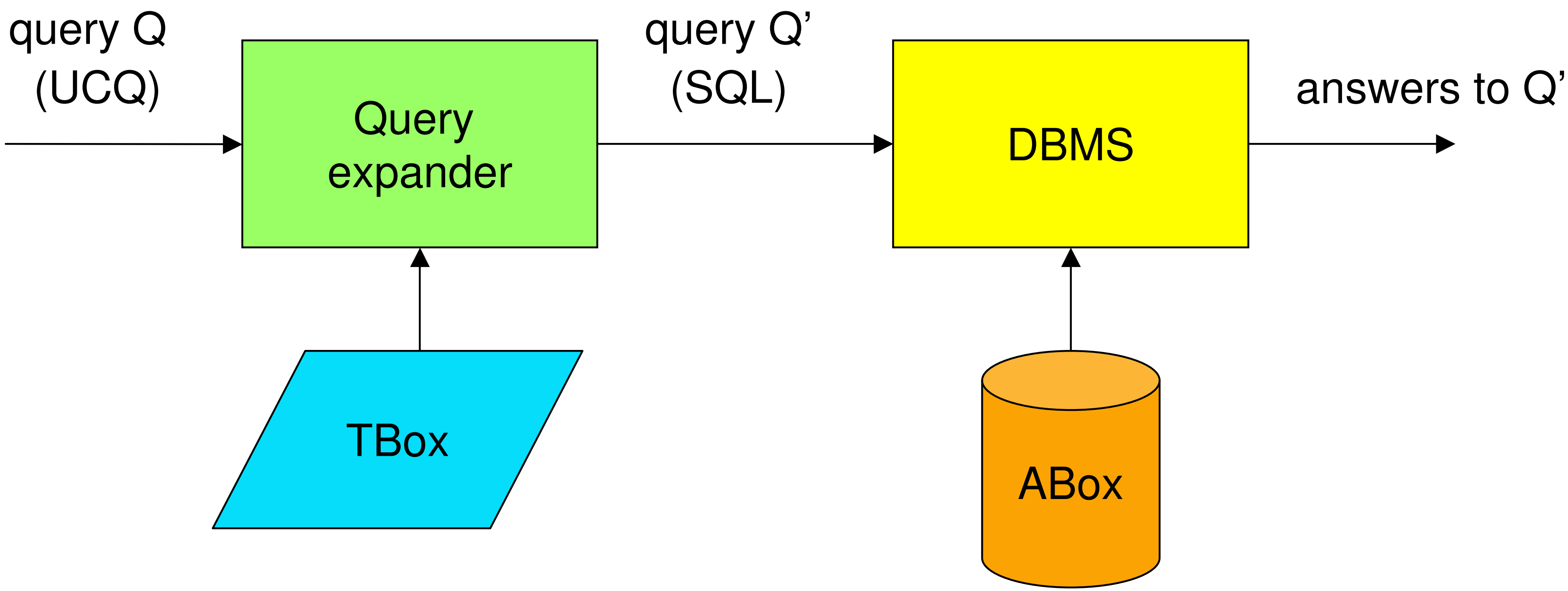
These languages have been designed with different purposes:

- OWL2 EL is tailored for applications employing ontologies that contain very large numbers of classes and/or properties
- OWL2 QL is tailored for applications that use very large volumes of instance data, and where query answering is the most important reasoning task
- OWL2 RL is tailored for applications that “require scalable reasoning without sacrificing too much expressive power”

OWL2 QL

- officially, OWL2 QL stands for... OWL2 QL ;-)
- informally, OWL2 QL can be read as “Query-oriented OWL2 fragment”
- “OWL2 QL is designed so that data (assertions) that are stored in a standard relational database system can be queried through an ontology via a simple rewriting mechanism, i.e., by rewriting the query into an SQL query that is then answered by the RDBMS system, without any changes to the data” [OWL2 Profiles, W3C Working Draft]
- OWL2 QL can be seen as a **“maximal fragment” of OWL2 DL** having the above property (when the query language is union of conjunctive queries)
- as we have seen in part 1, this property corresponds to **first-order (FOL) rewritability of unions of conjunctive queries**
- FOL rewritability $\Rightarrow$ SQL rewritability

OWL2 QL


query Q
(UCQ)
Query expander
query Q'
(SQL)
DBMS
answers to Q'
TBox
ABox

Syntax of OWL2 QL

class expressions are of two kinds: **subclass** and **superclass** expressions

subclassExpression $\sqsubseteq$ superclassExpression

- **subclass expressions** = DL-Lite concept expressions:
  - class name (atomic concept, including *owl:Thing*) A
  - unqualified existential quantification $\exists R$
- **superclass expressions**:
  - subclass expression A or $\exists R$
  - negation of a subclass expression $\neg A$ or $\neg\exists R$
  - conjunction of subclass expressions $C \sqcap D$
  - qualified existential quantification $\exists R.C$

property expressions (same as in DL-Lite and OWL/OWL2):
- property name R
- inverse of a property name R^-

OWL2 QL Axioms

- subclass axioms (**SubClassOf**)
- class expression equivalence (**EquivalentClasses**)
- class expression disjointness (**DisjointClasses**)
- inverse object properties (**InverseObjectProperties**)
- property inclusion (**SubObjectPropertyOf** and **SubDataPropertyOf**)
- property equivalence (**EquivalentObjectProperties** and **EquivalentDataProperties**)
- property domain (**ObjectPropertyDomain** and **DataPropertyDomain**)
- property range (**ObjectPropertyRange** and **DataPropertyRange**)
- disjoint properties (**DisjointObjectProperties** and **DisjointDataProperties**)
- symmetric properties (**SymmetricObjectProperty**)
- reflexive properties (**ReflexiveObjectProperty**)
- irreflexive properties (**IrreflexiveObjectProperty**)
- asymmetric properties (**AsymmetricObjectProperty**)
- assertions (**DifferentIndividuals**, **ClassAssertion**, **ObjectPropertyAssertion**, and **DataPropertyAssertion**)

OWL2 QL vs. DL-Lite$_R$: Syntax

essentially, OWL2 QL corresponds to DL-Lite$_R$

only significant addition to the TBox language:

- qualified existential quantification in superclass expressions:

  $B \sqsubseteq \exists R.C$

  – e.g.: all students are enrolled in a math course:
    $\text{student} \sqsubseteq \exists \text{EnrolledInCourse.Math}$

this is not a “real” language extension:

- qualified existential quantification can be actually simulated (encoded) in DL-Lite$_R$ through the use of auxiliary properties

OWL2 QL vs. DL-Lite$_R$: Syntax

essentially, OWL2 QL corresponds to DL-Lite$_R$

other small additions to the TBox language:

- distinction between objects and values and use of xsd datatypes
  - classes vs. datatypes

    (datatypes are a subset of xsd: (XML Schema) types)
  - object properties vs. data properties
- conjunction in superclass expressions (syntactic sugar)
- property axioms (reflexive, irreflexive, asymmetric)
- owl:Thing

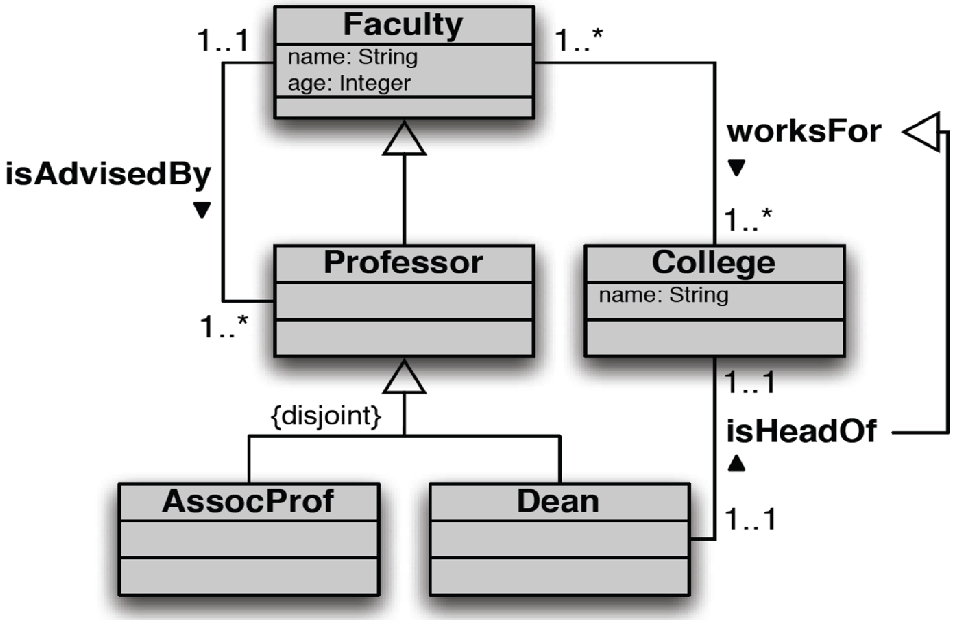
OWL2 QL vs. DL-Lite$_R$: Semantics

semantics:

- is the same as DL-Lite$_R$...
- …except for one aspect: Unique Name Assumption (UNA)
- UNA = different individuals denote different domain elements
- UNA is adopted in DL-Lite
- UNA is not adopted in OWL/OWL2 and thus neither in OWL2 QL
- however, this semantic difference does not affect any of the main reasoning tasks (i.e., reasoning in OWL2 QL is independent of the UNA)
- (at the end of part 3 we will come back to this aspect)

OWL2 QL TBox: Example

TBox of the UML diagram of part 1:



SubClassOf(Professor Faculty)
SubClassOf(Dean Professor)
SubClassOf(AssocProfessor Professor)
DisjointClasses(Dean AssocProfessor)
ObjectPropertyDomain(isHeadOf Dean)
ObjectPropertyRange(isHeadOf College)
SubObjectPropertyOf(isHeadOf worksFor)
InverseObjectProperties(isHeadOf hasHead)
SubClassOf(Dean ObjectSomeValuesFrom(isHeadOf owl:Thing))
SubClassOf(College ObjectSomeValuesFrom(hasHead owl:Thing))
ObjectPropertyDomain(worksFor Faculty)
ObjectPropertyRange(worksFor College)
InverseObjectProperties(worksFor hasMember)
SubClassOf(Faculty ObjectSomeValuesFrom(worksFor owl:Thing))
SubClassOf(College ObjectSomeValuesFrom(hasMember owl:Thing))

OWL2 QL TBox: Example (contd.)

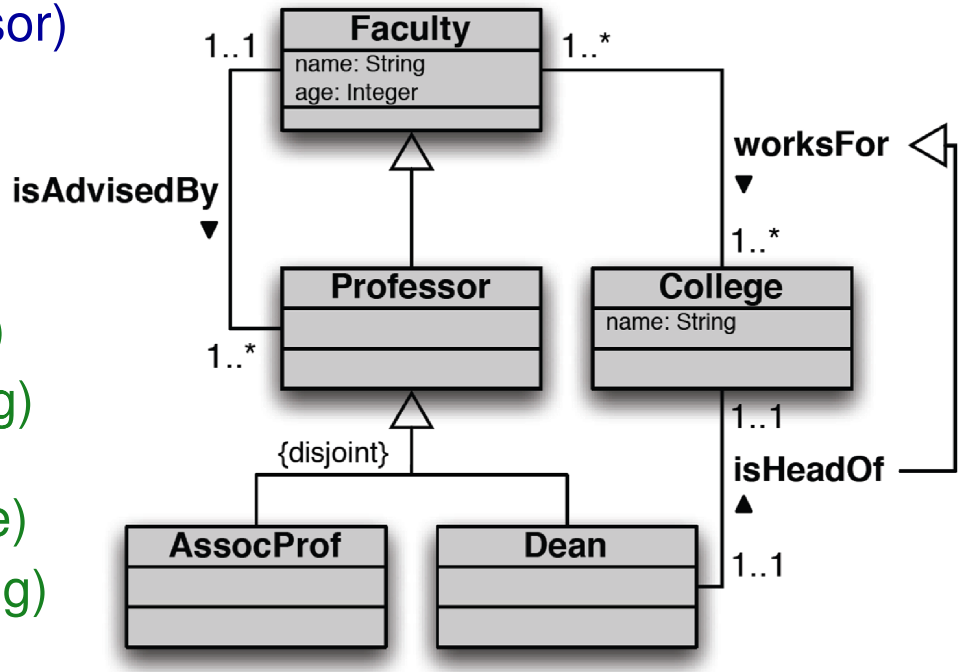
TBox of the UML diagram of part 1:

SubClassOf(Faculty ObjectSomeValuesFrom(isAdvisedBy owl:Thing))
InverseObjectProperties(isAdvisedBy advises)
SubClassOf(Professor ObjectSomeValuesFrom(advises owl:Thing))
ObjectPropertyDomain(isAdvisedBy Faculty)
ObjectPropertyRange(isAdvisedBy Professor)

DataPropertyDomain(age Faculty)
DataPropertyRange(age xsd:int)

DataPropertyDomain(facultyName Faculty)
DataPropertyRange(facultyName xsd:string)

DataPropertyDomain(collegeName College)
DataPropertyRange(collegeName xsd:string)



OWL2 QL vs. RDFS

- OWL2 QL essentially captures RDFS:
  - RDFS classes = classes
  - RDFS properties = properties
  - rdfs:subClassOf = class inclusion
  - rdfs:subPropertyOf = property inclusion
  - rdfs:domain = property domain
  - rdfs:range = property range
- but: OWL2 QL does not allow for meta-modeling (first-order language)
- DL-Lite extends RDFS:
  - "exact" role domain and range
  - concept and role disjointness
  - datatypes
  - ...

Semantic Technologies for Ontology-based Data Integration Using OWL2 QL

ESWC 2009 tutorial

Part 3
Expressive queries and constraints over OWL2 QL ontologies

Domenico Lembo, Riccardo Rosati

Dipartimento di Informatica e Sistemistica
Sapienza Università di Roma, Italy

SAPIENZA
UNIVERSITÀ DI ROMA

Overview

1. Expressive queries
   - motivation
   - EQL
   - EQL-Lite
   - EQL-Lite query answering in OWL2 QL
2. Constraints over OWL2 QL ontologies
   - the notion of CBox
   - denial and epistemic constraints
3. Equality in OWL2 QL
   - functional roles
   - equality assertions
   - semantic issue: Unique Name Assumption

1. Expressive queries over DL and OWL2 QL ontologies

Motivation

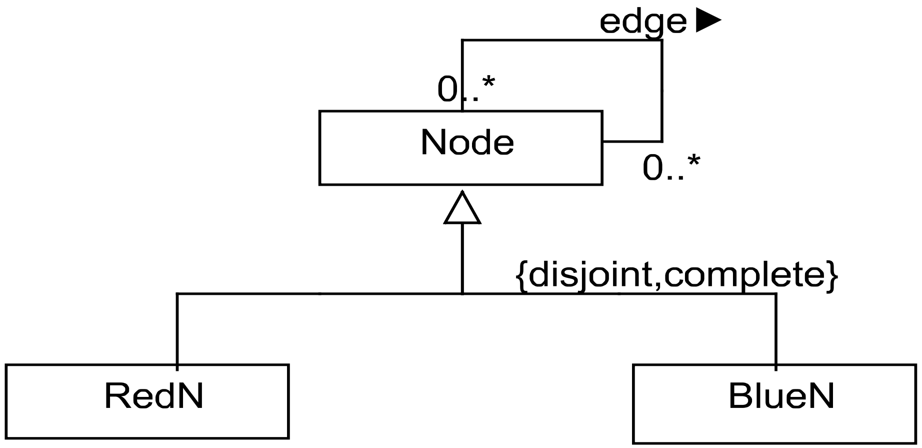
- Good techniques for doing instance checking are known
  - But DLs are poor query languages [Lenzerini-Schaerf-AAAI'91]
  - Also, for most DLs, coNP-hard in data complexity
- Techniques for answering CQs and UCQs are known
  - High complexity for expressive DLs
  - In LOGSPACE (same as SQL in DBs) for DL-Lite/OWL2 QL
- What about going beyond UCQs?
  - FOL/SQL queries over KB are undecidable

But, often users expect to have SQL-like query capabilities!!!

Example

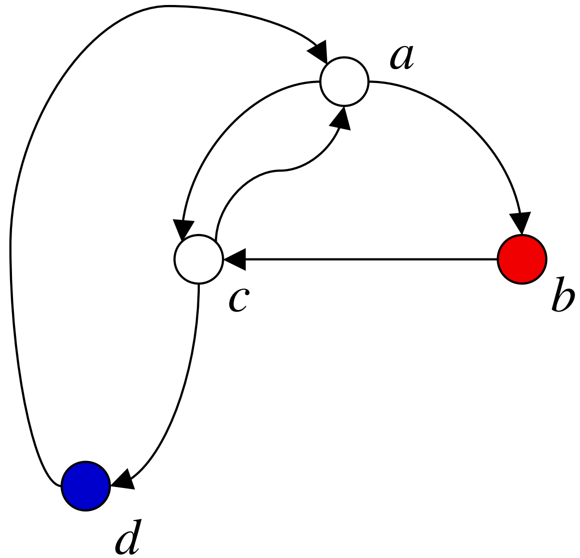
- TBox:

$\exists\, edge^- \sqsubseteq Node$
$\exists\, edge \sqsubseteq Node$
$RedN \sqsubseteq Node$
$BlueN \sqsubseteq Node$
$RedN \sqsubseteq \neg BlueN$
$Node \sqsubseteq RedN \sqcup BlueN$



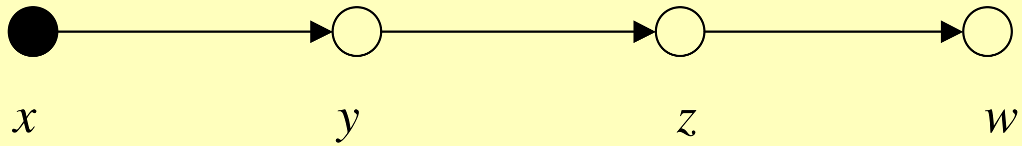
- ABox:

$edge(a,b),\ edge(b,c),$
$edge(c,a),\ edge(c,d),$
$edge(a,c),\ edge(d,a)$
$RedN(b),\ BlueN(d)$

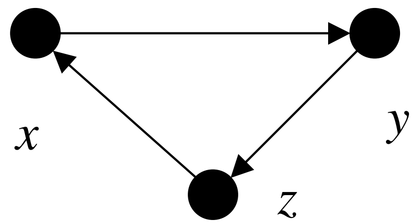


Queries

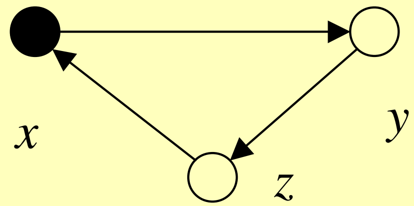
$q(x) \text{ :- } \exists\, y, z, w.\ edge(x,y) \land edge(y,z) \land edge(z,w)$



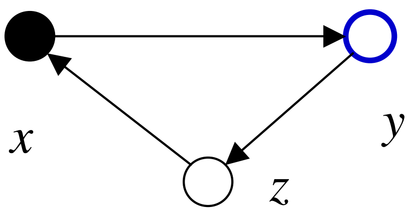
$q(x,y,z) \text{ :- } edge(x,y) \land edge(y,z) \land edge(z,x)$

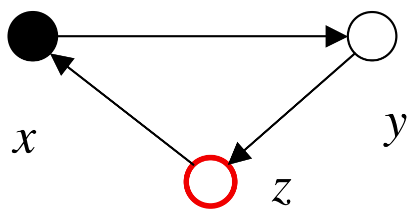


$q(x) \text{ :- } \exists\, y, z.\ edge(x,y) \land edge(y,z) \land edge(z,x)$



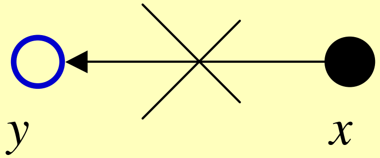
$q(x) \text{ :- } \exists\, y, z.\ edge(x,y) \land edge(y,z) \land edge(z,x) \land (BlueN(y) \lor RedN(z))$



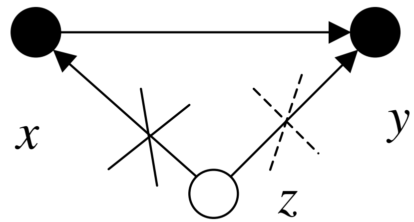


Queries

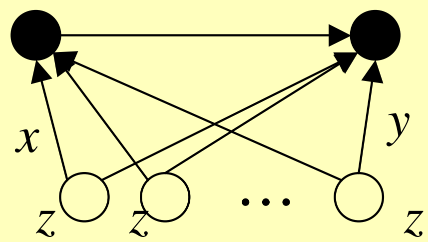
$$q(x) \text{ :- } \exists\, y.\ BlueN(y) \wedge \neg edge(x,y)$$



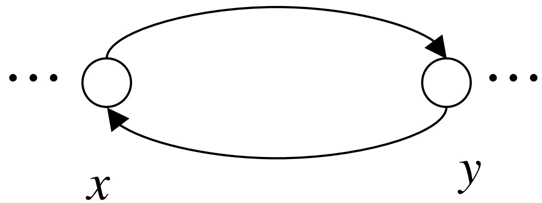
$$q(x,y) \text{ :- } edge(x,y) \wedge \neg\exists\, z.\ (edge\ (z,x) \wedge edge(z,y))$$



$$q(x,y) \text{ :- } edge(x,y) \wedge \forall\, z.\ (edge\ (z,x) \Rightarrow edge(z,y))$$



$$q() \text{ :- } \forall\, x,y.\ (edge\ (x,y) \Rightarrow edge(y,x))$$



An experiment on relational databases

Person

name	birthdate
john	1940
paul	1942
george	1943
richard	null

SQL query:

$$q(x) \text{:- } \exists\, b.(Person(x,b) \wedge b = 1940) \vee$$
$$\exists\, b.(Person(x,b) \wedge b \neq 1940)$$

Answer:

$\{john, paul, george\}$

What about richard? Since the DBMS ***doesn't know*** *his birthdate, the DBMS can't establish whether it is equal to 1940 or different from 1940, hence the DBMS skips it!*

Epistemic Query Language (EQL)

- Let KB be a DL KB, intepreted over
  - a fixed domain Δ, and
  - **standard names**

- EQL = FOL + epistemic operator (minimal knowledge) over KB

$$\varphi ::= A(t) \mid P(t_1,\ldots,t_n) \mid t_1 = t_2 \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \exists\, x.\varphi \mid \mathbf{K}\varphi$$

A : concept name in *KB*
P : role/relation name in *KB*
t : constant in *KB* or variable

Cf:

- Levesque's "*Foundations of a Functional Approach to KR*" [AIJ'84]
- Reiter's "*What should a DB know?*" [JLP'92]
- Levesque & Lakemeyer's "*The Logic of KBs*" [Book, 2001]
- Epistemic operator in DLs [Donini-Lenzerini-Nardi-Schaerf-Nutt-KR'92]

EQL: semantics

- Let KB a KB and φ a EQL formula
- Epistemic interpretation E,w
  - E is the set of **all** models of KB
  - w is **one** such model
- φ true in E,w, written $E,w \models \varphi$

$E,w \models A(c)$	iff	$c \in A^w$
$E,w \models P(c_1,\ldots,c_n)$	iff	$(c_1,\ldots,c_n) \in P^w$
$E,w \models c_1{=}c_2$	iff	$c_1{=}c_2$
$E,w \models \neg\varphi$	iff	$E,w \models \varphi$
$E,w \models \varphi_1 \wedge \varphi_2$	iff	$E,w \models \varphi_1$ and $E,w \models \varphi_2$
$E,w \models \exists x.\varphi(x)$	iff	$E,w \models \varphi(c)$ for some c
$E,w \models \mathbf{K}\varphi$	iff	$E,v \models \varphi$ for all $v \in E$

EQL: objective and subjective formulas

- Objective formulas
  - no occurrence of ***K***
  - talk about what is true in the world
  - example: $\exists\, x, y.\ edge(x,y)$
  - $E,w \models \varphi$ reduces to $w \models \varphi$
- Subjective formulas
  - all atoms under the scope of ***K***
  - talk about what is known by the KB
  - example: $\exists\, x, y.\ \boldsymbol{K}\, edge(x,y)$
  - $E,w \models \varphi$ reduces to $E \models \varphi$
- Non-objective and non-subjective formulas
  - talk about what is true in world in relation to what is known by the KB
  - example: $\exists\, x, y.\ edge(x,y) \wedge \boldsymbol{K}\, edge(x,y)$

EQL: knowledge & logical implication

Fundamental property of EQL: minimal knowledge

$$KB \models \varphi \quad \text{iff} \quad KB \models \mathbf{K}\varphi$$
$$KB \not\models \varphi \quad \text{iff} \quad KB \models \neg\mathbf{K}\varphi$$

In other words:

- $\mathbf{K}\varphi$ can be read as φ is logically implied
- $\neg\mathbf{K}\varphi$ can be read as φ is not logically implied
  i.e.: $\neg\varphi$ is satisfiable

Example:

$$\mathbf{K}edge(a,b) \wedge \mathbf{K}edge(b,c) \wedge \mathbf{K}edge(c,d)$$

can be read:

- edges (a,b), (b,c), (c,d) are known
- edges (a,b), (b,c), (c,d) are logically implied

EQL: queries

- EQL query:

$$q(x_1,\ldots,x_n) \text{ :- } \varphi(x_1,\ldots,x_n)$$

- Answer:

$$ans(q, KB) = \{ (c_1,\ldots,c_n) \mid KB \models \varphi(c_1,\ldots,c_n), \quad c_i \in \Delta \}$$

EQL - CQs without existential variables

Example:

$$q(x,y,z) \text{ :- } edge(x,y) \wedge edge(y,z) \wedge edge(z,x)$$

is equivalent to *(since $KB \models \varphi$ iff $KB \models \mathbf{K}\varphi$)*

$$q(x,y,z) \text{ :- } \mathbf{K}(edge(x,y) \wedge edge(y,z) \wedge edge(z,x))$$

is equivalent to *(since **K** distributes over ANDs)*

$$q(x,y,z) \text{ :- } \mathbf{K}edge(x,y) \wedge \mathbf{K}edge(y,z) \wedge \mathbf{K}edge(z,x)$$

EQL-Lite($\mathcal{Q}$)

- Restriction on EQL, parametric wrt an objective query language $\mathcal{Q}$

- EQL-Lite($\mathcal{Q}$) queries have the form (with α in $\mathcal{Q}$)

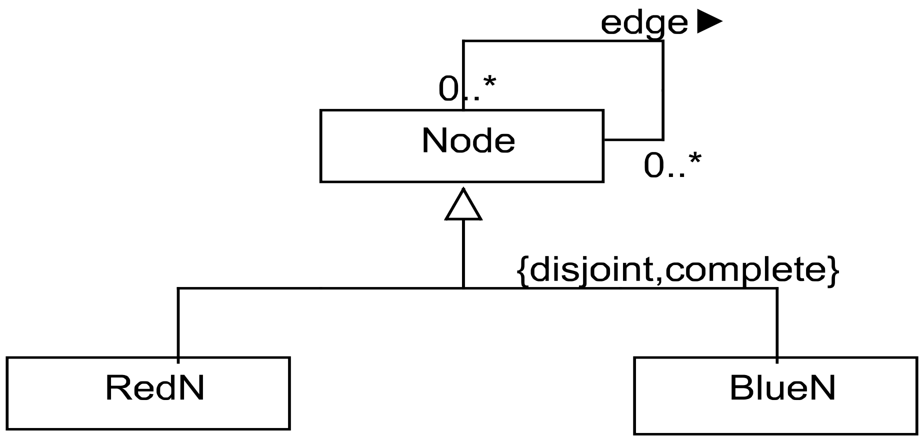
$$\varphi ::= \mathbf{K}\alpha \mid t_1 = t_2 \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \exists\, x.\varphi$$

and are domain independent (cf. relational algebra)

Example

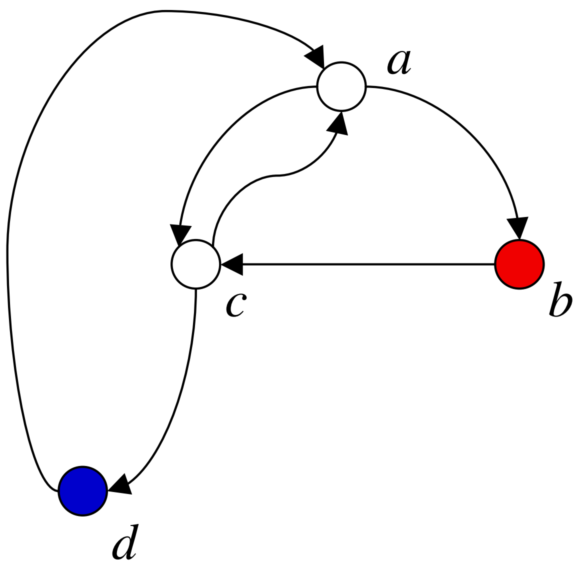
- TBox:

  $\exists\, edge^{-} \sqsubseteq Node$
  $\exists\, edge \sqsubseteq Node$
  $RedN \sqsubseteq Node$
  $BlueN \sqsubseteq Node$
  $RedN \sqsubseteq \neg BlueN$
  $Node \sqsubseteq RedN \sqcup BlueN$



- ABox:

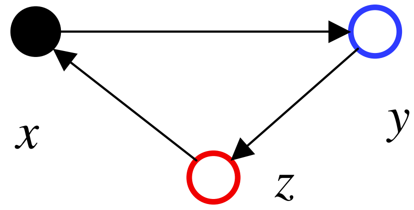
  $edge(a,b)$
  $edge(b,c)$
  $edge(c,a)$
  $edge(c,a)$
  $edge(a,c)$
  $edge(d,a)$
  $RedN(b)$
  $BlueN(d)$



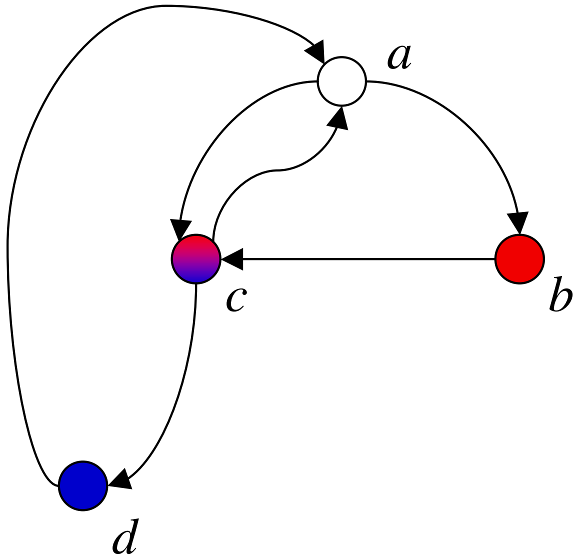
Example

- Query:

$$q(x) \text{ :- } \exists\, y, z.\ edge(x,y) \wedge RedN(y) \wedge edge(y,z) \wedge BlueN(z) \wedge edge(z,x)$$



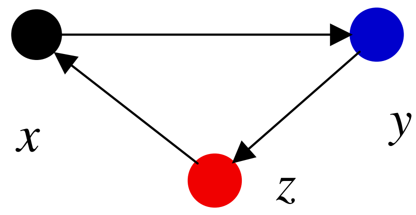
- Answer: $\{a\}$



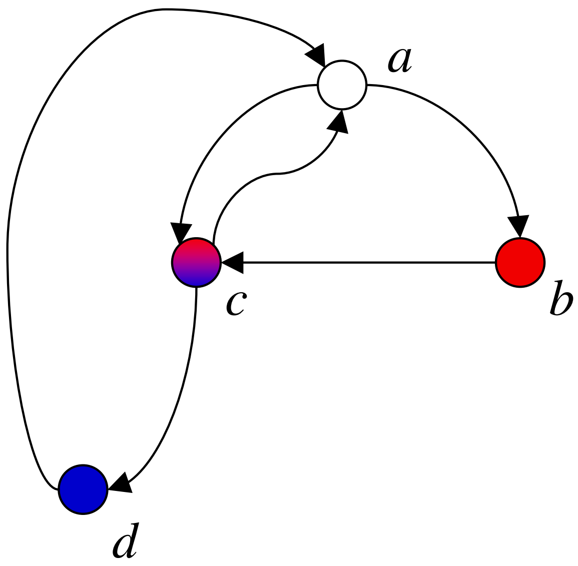
Example

- Query:

$$q(x) \text{ :- } \exists\, y, z.\ \mathbf{K}\, edge(x,y) \wedge \mathbf{K}\, RedN(y) \wedge \mathbf{K}\, edge(y,z) \wedge \mathbf{K}\, BlueB(z) \wedge \mathbf{K}\, edge(z,x)$$



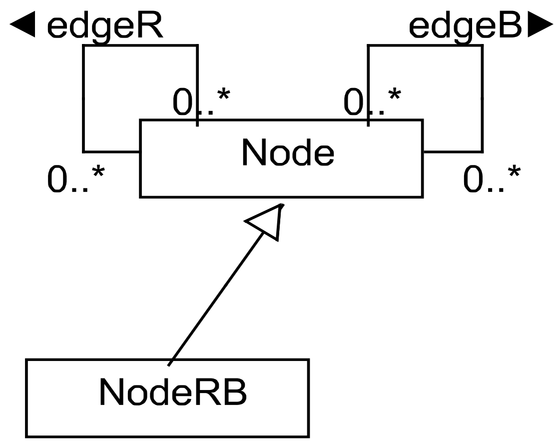
- Answer: { }



Example

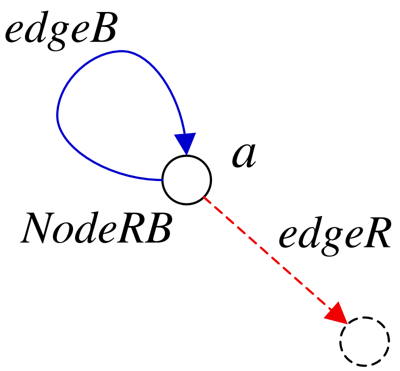
- TBox:

$\exists\, edgeR^{-} \sqsubseteq Node$
$\exists\, edgeR \sqsubseteq Node$
$\exists\, edgeB^{-} \sqsubseteq Node$
$\exists\, edgeB \sqsubseteq Node$
$NodeRB \sqsubseteq \exists\, edgeR$
$NodeRB \sqsubseteq \exists\, edgeB$



- ABox:

$edgeB(a,a)$
$NodeRB(a)$



Queries

- Query:

  $$q1(x) \text{ :- } \exists\, y, z, w.\ edgeB(x,y) \land edgeR(x,z) \land edgeR(y,z)$$

  Answer: $\{a\}$

- Query:

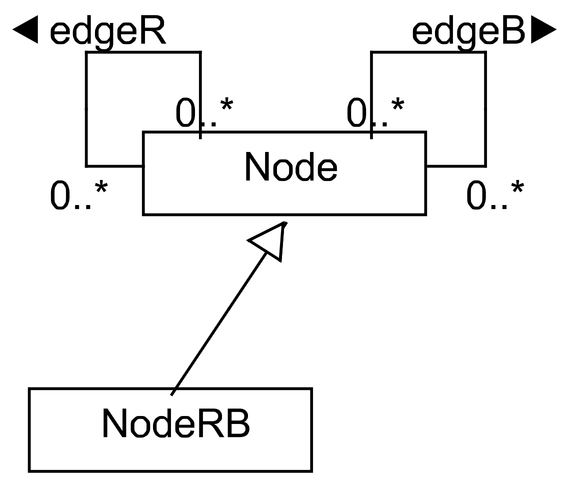
  $$q2(x,y,z) \text{ :- } edgeB(x,y) \land edgeR(x,z) \land edgeR(y,z)$$

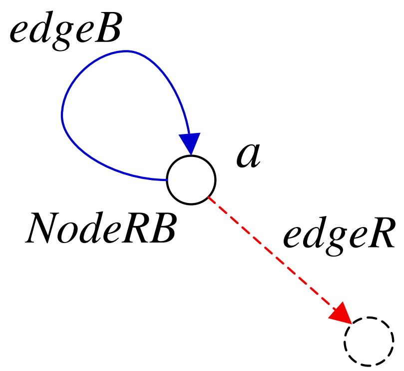
  Answer: $\{\,\}$

- Query:

  $$q3(x) \text{ :- } \exists\, y, z, w.\ \mathbf{K}\, edgeB(x,y) \land \mathbf{K}\, edgeR(x,z) \land \mathbf{K}\, edgeR(y,z)$$

  Answer: $\{\,\}$





EQL-Lite($\mathcal{Q}$): main result

- A $\mathcal{Q}$ query α is $\mathcal{KB}$-range restricted iff ans(α,$\mathcal{KB}$) is finite.

- An EQL-Lite($\mathcal{Q}$) query is $\mathcal{KB}$-range restricted iff all α appearing in it are $\mathcal{KB}$-range restricted.

- **Thm:** if ans(α,$\mathcal{KB}$) is finite, then it contains only individuals occurring in $\mathcal{KB}$.

- **Thm:** Let $\mathcal{KB}$ be a KB expressed in the DL $\mathcal{L}$ and let ***C*** be the data complexity of answering queries in *Q* over KBs in $\mathcal{L}$, then,

  answering a $\mathcal{KB}$-range restricted EQL-Lite($\mathcal{Q}$) is in LOGSPACE^{C} w.r.t. data complexity.

EQL-Lite on UCQs in $\mathcal{SHIQ}$ KB

- $\mathcal{Q} \rightarrow$ UCQs
- KB $\rightarrow \mathcal{SHIQ}$ (or variants)

- Answering UCQs is coNP-complete w.r.t. data complexity for $\mathcal{SHIQ}$
- Answering EQL-Lite queries is $\text{LOGSPACE}^{\text{coNP}}$

EQL-Lite on UCQs in $\mathcal{ALCQI}$ KB

- $\mathcal{Q} \rightarrow$ UCQs
- KB $\rightarrow \mathcal{ALCQI}$

- Answering UCQs in $\mathcal{ALCQI}$ KBs is coNP-complete w.r.t. data complexity

- Answering EQL-Lite queries is $\text{LOGSPACE}^{\text{coNP}}$

EQL-Lite on UCQs in $\mathcal{EL}$ KB

- $\mathcal{Q} \rightarrow$ UCQs
- KB $\rightarrow \mathcal{EL}$ (in fact any member of the $\mathcal{EL}$ family)

- Answering UCQs in $\mathcal{EL}$ KBs is PTIME-complete w.r.t. data complexity

- Answering EQL-Lite queries is PTIME-complete

EQL-Lite on UCQs in DL-Lite/OWL2QL KB

- $\mathcal{Q} \rightarrow$ UCQs
- KB $\rightarrow$ DL-Lite (in fact any member of the DL-lite family)

- Answering UCQs in every DL-Lite is in LOGSPACE w.r.t. data complexity, actually UCQs are FOL rewritable

- Answering EQL-Lite(UCQ) queries in every DL-Lite (and thus in OWL2QL) is in LOGSPACE w.r.t. data complexity:
  – actually, **EQL-Lite(UCQ) queries are FOL reducible (and thus rewritable in SQL)**:
    - external EQL-Lite query = SQL query, with one SQL subquery for every UCQ q within the scope of a **K** operator
    - subquery for **K**q = SQL query corresponding to the FOL rewriting of q with respect to the DL-Lite/OWL2 QL TBox

UCQ vs. EQL-Lite(UCQ)

- what can we express in EQL-Lite(UCQ) that is not expressible in UCQ?
- negation (difference):
  - e.g.: return all non-working students:
  - q(x) :- **K** Student(x) $\wedge$ $\neg$**K** ($\exists$z. WorksFor(x,y))
- universal quantification:
  - e.g.: return all happy fathers, (happy father = father having all happy children)
  - q(x) :- $\exists$y. **K** Father(x,y) $\wedge$ $\neg$**K** ($\exists$z. Father(x,z), unhappy(z))
- inequality, comparison operators, ...

Summary

- EQL-Lite can be seen as a semantically well characterized approximation of FOL queries
- EQL-Lite is based on a controlled use of the epistemic (minimal knowledge) operator
- Jumping from $\mathcal{Q}$ to EQL-Lite($\mathcal{Q}$) is (almost) for free
- EQL-Lite on UCQs over DL-Lite/OWL2 QL is FOL-rewritable (SQL!)
- EQL-Lite is very interesting also for modeling constraints over ontologies
- **SparSQL** = our concrete syntax for EQL-Lite(UCQ)
  - external query written in SQL
  - UCQs within K operators written in SPARQL (unions of basic graph patterns)

2. Constraints over OWL2 QL ontologies

The Constraint Box (CBox)

- general idea: add a **Constraint Box (CBox)** to a DL knowledge base
- KB = (K, CB) where
  - K is a DL knowledge base
  - CB is a CBox
- the CBox contains intensional knowledge, like the TBox...
- ...but: **both syntax and semantics of the CBox are “different” from the TBox**
- idea proposed in different forms in the past
  - Reiter 1990
  - Motik et al., 2007

Constraints and queries

our view of a CBox:

- a CBox is a set of constraints, where

constraint = negation of a query

- a constraints can be written as: **query $\rightarrow$ false** (or: **query $\rightarrow \perp$**)
- given **KB=(K,CB)** where K DL knowledge base and CB is a CBox:
  - an interpretation satisfies a constraint if the corresponding query is not satisfied (empty answer)
  - the models of KB are the models of K that satisfy all constraints in CB

Denial and epistemic constraints

- first-order semantics:
  - if query = UCQ, we speak of a **denial constraint**
- epistemic semantics:
  - if query = EQL-Lite(UCQ) we speak of **EQL-Lite constraints** (or **epistemic constraints**)

- epistemic constraints follow the notion of **integrity constraint** for a knowledge base proposed in [Reiter 1990]

Denial constraints

Examples:

consider an ontology with:

- concept student
- roles Teaches, EnrolledInCourse, Father

(1) no student can be enrolled both in course c1 and course c2:

$(\exists x.\ \text{student}(x) \wedge \text{EnrolledInCourse}(x,c1) \wedge \text{EnrolledInCourse}(x,c2)) \rightarrow \bot$

(2) no student can be enrolled in a course taught by her/his father:

$(\exists x.\ \text{student}(x) \wedge \text{EnrolledInCourse}(x,y) \wedge \text{Father}(x,z) \wedge \text{Teaches}(z,y)) \rightarrow \bot$

notice: denial constraints must be satisfied **by all domain elements** (both named and unnamed individuals)

Epistemic constraints

Examples:

(1) every student must be enrolled either in course c1 or in course c2:

$(\exists x.\ \mathbf{K}\ \text{student}(x) \land$

$\neg\mathbf{K}(\text{EnrolledInCourse}(x,c1) \lor \text{EnrolledInCourse}(x,c2))\ ) \rightarrow \bot$

(2) every student that is not enrolled neither in course c1 nor in course c2 must be enrolled in course c3:

$(\exists x.\ \mathbf{K}\ \text{student}(x) \land$

$\neg\mathbf{K}(\text{EnrolledInCourse}(x,c1) \lor \text{EnrolledInCourse}(x,c2)) \land$

$\neg\mathbf{K}\ \text{EnrolledInCourse}(x,c3)\ ) \rightarrow \bot$

notice: the above epistemic constraints must be satisfied **by known (named) individuals only**

Reasoning in the presence of a CBox

in every DL:

- reasoning over a KB with CBox **KB=(K,CB)** can be reduced to query answering over K
- a preliminary step is needed which checks consistency of K with respect to the CBox:
  - for every constraint C in the CBox, verify that the corresponding query Q(C) has an empty answer over K
  - if K does not pass this test, then the overall KB is inconsistent
  - otherwise, we can discard CB and proceed by reasoning over K

$\Rightarrow$ adding this kind of CBox is “almost for free” in DL systems supporting (expressive) query answering

Reasoning under CBox in OWL2 QL

- adding a CBox to DL-Lite/OWL2 QL ontologies does not increase the worst-case complexity of reasoning
  – query answering (as well as all the other reasoning tasks) is still first-order (FOL) rewritable:
    - UCQs are FOL-rewritable
    - EQL-Lite(UCQ) queries are FOL rewritable
- most importantly, if in KB=(K,CB) K (and in particular its ABox) is static, the presence of the CBox CB in KB can be processed off-line (not at query evaluation time)

3. Equalities in DL-Lite/OWL2 QL

Forms of equalities in DL-Lite

Can we speak about equality in DL-Lite?

- TBox: functional roles:
  - functional(R) with R basic role
  - this is a form of implication involving the equality predicate
  - equivalent to the FOL sentence $\forall x\ (R(x,y) \wedge R(x,z) \rightarrow y=z)$
- ABox: no equality assertion allowed in DL-Lite
  - we cannot state that two names denote the same object
    - e.g., we cannot state that a and b denote the same object (which is expressed in OWL by (a owl:sameAs b))
  - why?
    - because the semantics of DL-Lite is based on the Unique Name Assumption

Unique Name Assumption (UNA)

Unique Name Assumption (UNA):

- different individuals denote different objects

- in every model of the knowledge base, a and b denote different domain elements
- so the equality assertion (a owl:sameAs b) is always inconsistent with respect to the UNA
- thus, under the UNA, equality assertions are useless

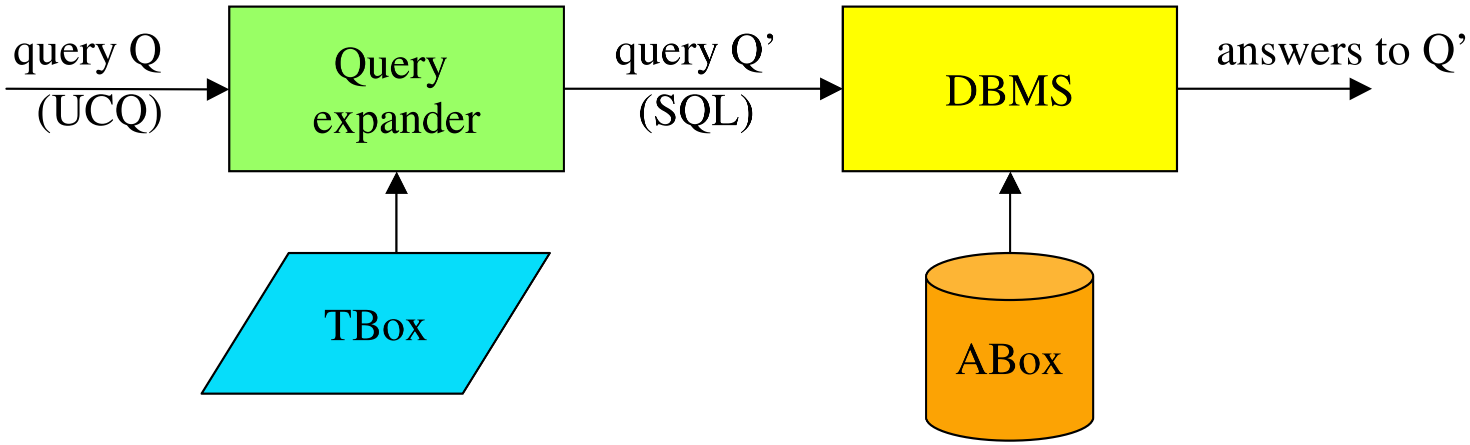
OWL2 QL and the UNA

- OWL2 (as well as OWL) does **not** adopt the UNA
- as an OWL2 profile, also OWL2 QL does not adopt the UNA
- but: OWL2 QL is actually “insensitive” to the Unique Name Assumption
- reasoning in OWL2 QL under UNA is the same as reasoning in OWL2 QL without UNA
- why? because OWL2 QL does not allow for expressing any form of equality in the knowledge base

Functional roles and OWL2QL

functional roles are not allowed in OWL2 QL

- why?
- because adding functional roles to OWL2 QL makes UCQs non-first-order-rewritable
- i.e.: jump in the worst-case complexity of query answering
- and this query answering scheme is not feasible anymore:



DL-Lite$_A$

- there are actually two members of DL-Lite which allow for expressing functional roles:
  - DL-Lite$_F$
  - DL-Lite$_A$ (which is a superset of DL-Lite$_F$)
- essentially, DL-Lite$_A$ is an extension OWL2 QL with functional roles

Functional roles cannot be specialized

- how does DL-Lite$_A$ overcome the computational problem due to functional roles?
- by imposing a **syntactic restriction** on the use of functional roles in role hierarchies:
  - in DL-Lite$_A$, **functional roles cannot be specialized**
  - i.e., if R is declared functional, then no role R’ can be declared as a specialization of R (i.e., R’ $\sqsubseteq$ R is not allowed)
  - under this restriction, UCQs are still first-order rewritable

Functional roles cannot be specialized

if we adopt the same restriction, can we add functional roles to OWL2 QL?

- the above restriction “works” only in the presence of the UNA
- without the UNA, even under this restriction, the presence of functional roles makes UCQs non-first-order rewritable anymore
- so it is impossible to add functional roles in OWL2 QL and retain FOL rewritability of UCQs without adopting the UNA

Beyond the Unique Name Assumption

what if we allow for owl:sameAs assertions in the ABox?

- again, we lose FOL rewritability of query answering
- but: we can think of techniques for handling equalities in an approximate (incomplete) way
- or, we can think of pre-processing the ABox, “materializing” (or “propagating”) all equalities
- however: the above pre-processing is likely to be unrealistic in ontology-based data integration
- we should:
  1. retrieve (off-line) all instance data from all sources
  2. pre-process instances (propagating equalities)
  3. at query evaluation time, we can discard equalities
- step 1 may be impossible and/or computationally too expensive
- moreover, in dynamic scenarios, at step 3 instances may be different than those retrieved at step 1!!

Summary

- we cannot actually add the above forms of equality to OWL2 QL without losing the nice computational properties of query answering
- no real surprise: the equality predicate is recursive and non-expressible in FOL
- we can think of approximate (incomplete) ways for handling equality, without getting out of the FOL rewritability class

Summary

- is “first-order rewritability” a real limit that cannot be surpassed by data-intensive ontologies?
- real issue (open research problem)
- our opinion:
  - FOL rewritability = reuse of relational database technology for query processing
  - more expressive ontologies/queries necessarily require support for (at least linear) **recursion**
  - currently, there is no avaliable technology for recursive queries that is comparable to SQL technology
  - that’s why FOL rewritability is so crucial for us (and for DL-Lite)...
  - e.g., in a real “billion-triples” application with a very large number of instances and *owl:sameAs* assertions, an exact treatment of equality is something we currently cannot afford

References (for Part 2 and Part 3)

On OWL2, OWL2 profiles and OWL2 QL:

- www.w3.org/2007/OWL

On epistemic queries and EQL-Lite:

- Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati: *EQL-Lite: Effective First-Order Query Processing in Description Logics*. IJCAI 2007: 274-279

On (epistemic) constraints, epistemic queries, and SparSQL:

- Claudio Corona, Emma Di Pasquale, Antonella Poggi, Marco Ruzzi, Domenico Fabio Savo: *When OWL met DL-Lite...* SWAP 2008

On first-order rewritability, equality, and UNA:

- Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Riccardo Rosati, Marco Ruzzi: *Data Integration through DL-Lite$_A$ Ontologies*. SDKB 2008: 26-47

Part IV

Tools for Ontology-Based Data Integration

Outline

1. The QuOnto Reasoner
2. A Case Study: Accessing SAPIENZA's database through LUBM ontology

Outline

1. The QuOnto Reasoner
2. A Case Study: Accessing SAPIENZA's database through LUBM ontology

SAPIENZA
Università di Roma

The QuOnto Reasoner

- 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 – *API are available for selected projects upon request*

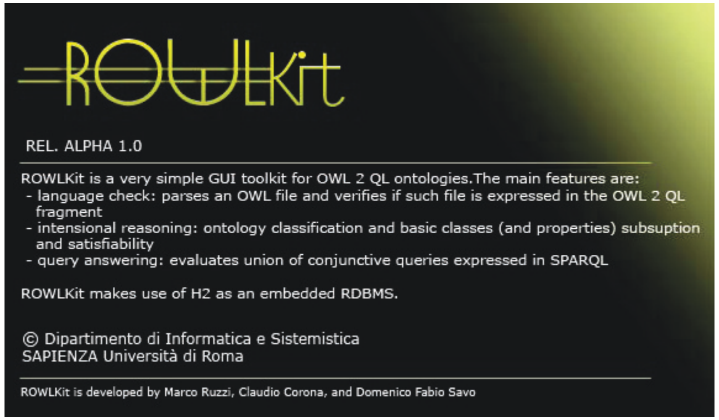
QuOnto wrapped versions
http://www.dis.uniroma1.it/~quonto/

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

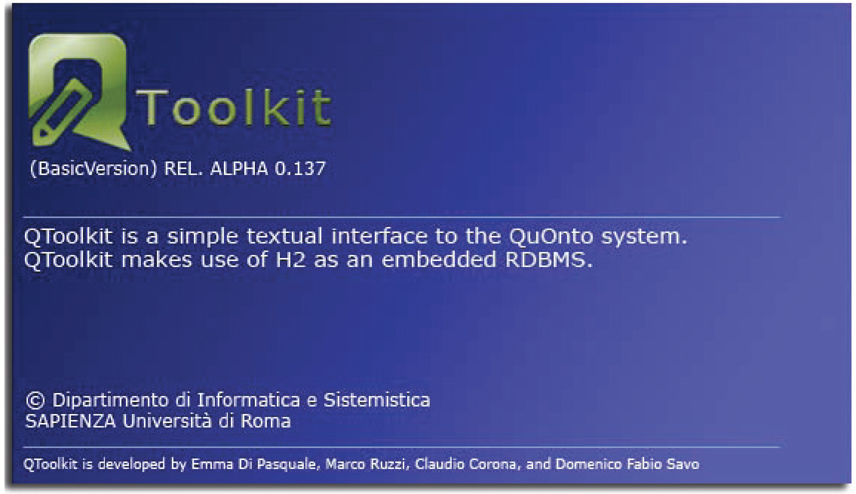
SAPIENZA
UNIVERSITÀ DI ROMA

ROWLkit



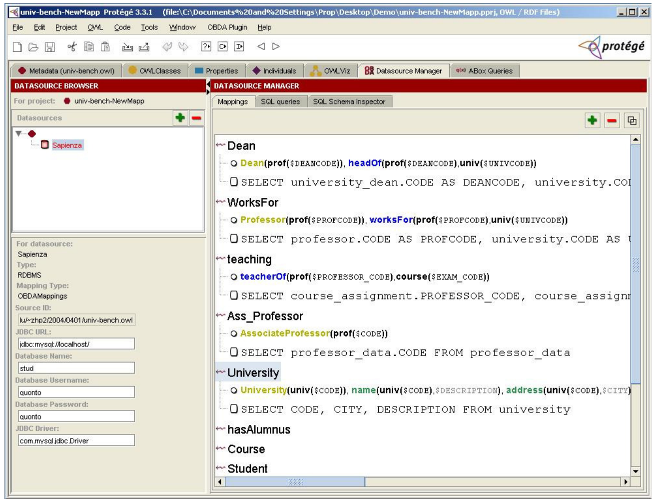
- ROWLKit is a system with a simple GUI to reason over ontologies written in OWL2 QL. At its core it uses QuOnto services enriched with additional features to deal with OWL2 QL ontologies
- It takes as input OWL2 QL ontologies through OWL API
- ROWLKit main services are:
  - Ontology satisfiability check
  - 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).

SAPIENZA Università di Roma



- 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
- Currently available for Protégé 3.3, a version of the plugin for Protégé 4 will be distributed soon

Outline

1. The QuOnto Reasoner

2. A Case Study: Accessing SAPIENZA's database through LUBM ontology

SAPIENZA UNIVERSITÀ DI ROMA

The Case Study

We will show how to access

- an actual large database of the University of Rome "La Sapienza", storing information on professors, students, exams, course assignments, etc. of the school of engineering, referring to the years 1990–1999. . .
- . . . through the university domain ontology provided by the Lehigh University BenchMark (LUBM) (`http://swat.cse.lehigh.edu/projects/lubm/`).

The LUBM Ontology

- In fact LUBM consists of a university domain ontology expressed in OWL, customizable and repeatable synthetic data, a set of test queries, and several performance metrics.
- In this case study we make use only of (an adapted version of) the ontology. Such ontology can be almost completely specified in *DL-Lite*/OWL2 QL and suitably connected through mappings to the SAPIENZA's database.
- Furthermore, to better model the domain of interest of "La Sapienza", we enriched the ontology with (few) additional concepts and roles (e.g., to model exams passed by students).

The SAPIENZA's database

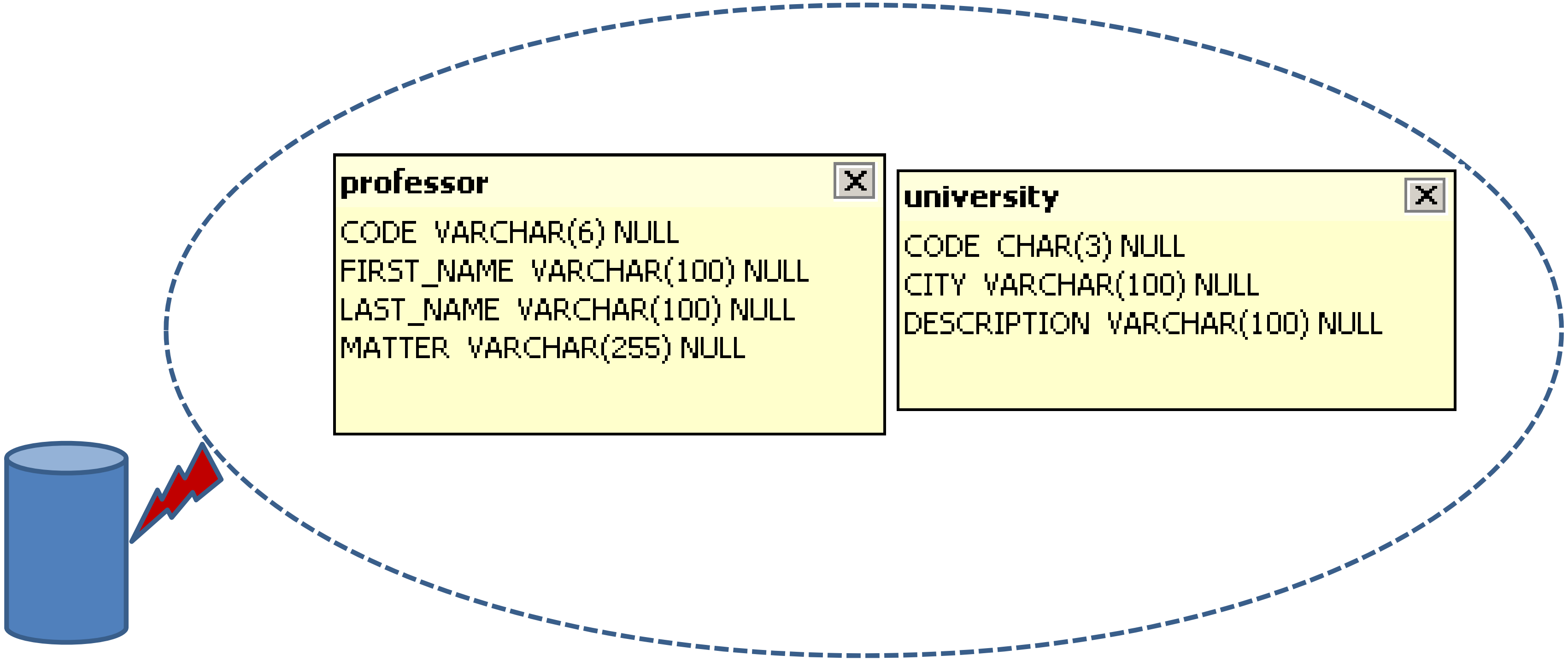
About 250.000 tuples stored in 29 tables in the DBMS MySQL.

Name	Data Length	Rows
bachelor_exam	1 KB	28
bachelor_exam_record	918 KB	17001
career	1932 KB	50633
career_status	1 KB	15
course	184 KB	4722
course_assignment	10 KB	402
course_exam	141 KB	7204
degree	48 KB	397
degree_course	61 KB	1716
enrollment	1 KB	5
exam	793 KB	17144
exam_plan	46 KB	1089
exam_plan_data	533 KB	27130
exam_plan_status	1 KB	3

Name	Data Length	Rows
exam_regularity	1 KB	4
exam_type	1 KB	3
faculty	14 KB	511
high_school	3 KB	69
master_exam	3 KB	66
master_exam_record	4166 KB	106273
modality	1 KB	2
person	0 KB	0
positioning	2 KB	29
professor	11 KB	146
professor_data	2 KB	67
session	1 KB	4
student	2568 KB	16079
university	10 KB	163
university_dean	1 KB	2

SAPIENZA
UNIVERSITÀ DI ROMA

Example of Mapping Assertion




```
SELECT professor.CODE AS PROFCODE, university.CODE AS UNIVCODE
FROM professor, university
WHERE university.DESCRIPTION = 'Sapienza'
```

Professor(**prof**($PROFCODE)), worksFor(**prof**($PROFCODE),**univ**($UNIVCODE))

Acknowledgements

People involved in this work:

- Sapienza Università di Roma
  - Claudio Corona
  - Giuseppe De Giacomo
  - Maurizio Lenzerini
  - Antonella Poggi
  - Riccardo Rosati
  - Marco Ruzzi
  - Domenico Fabio Savo
- Libera Università di Bolzano
  - Diego Calvanese
  - Mariano Rodriguez Muro
- Students (thanks!)