

# Query rewriting for ontology-based (big) data access

*Maurizio Lenzerini*

Dipartimento di Ingegneria Informatica  
Automatica e Gestionale Antonio Ruberti

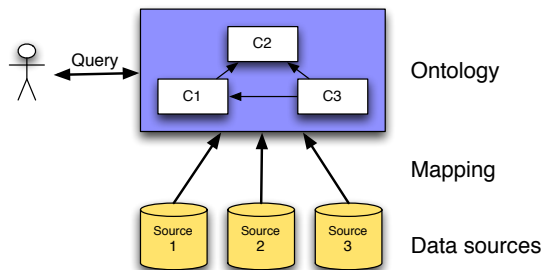


SAPIENZA  
UNIVERSITÀ DI ROMA

*Global scientific data infrastructures: The findability challenge*

Taormina, May 10–11, 2012

# Ontology-based data access



- **Ontology**, used as the conceptual layer to give clients a unified conceptual specification of the domain of interest.
- **Data sources**, representing external, independent, heterogeneous, storage (or, more generally, computational) structures.
- **Mappings**, used to semantically link data at the sources to the ontology.

- **Ontology**
  - terminology,
  - taxonomy,
  - conceptual schema,
  - logical theory.
- **Data sources**
  - databases,
  - KR assertional components,
  - semi-structured data,
  - documents.
- **Mappings**
  - query-to-query,
  - one-to-one,
  - annotations.

**Challenge:** optimal compromise between expressive power and effectiveness of the reasoning system.

An OBDA system is a triple  $\langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$ , where

- $\mathcal{T}$  is the ontology, expressed as a logical theory in a specific logical language
- $\mathcal{S}$  is the source database (whose signature is disjoint from the one of  $\mathcal{T}$ )
- $\mathcal{M}$  is a set of mapping assertions; in the general case, each one has the form

$$\Phi(\vec{x}) \rightsquigarrow \Psi(\vec{x})$$

where

- $\Phi(\vec{x})$  is a query over  $\mathcal{S}$ , returning values for  $\vec{x}$
- $\Psi(\vec{x})$  is a query over  $\mathcal{T}$ , whose free variables are from  $\vec{x}$ .

Let  $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$  be a Tarskian interpretation for the signature of  $\mathcal{T}$ .

Def.: **Model**

$\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$  is a **model** of  $\langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$  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.: **Mapping satisfaction**

We say that  $\mathcal{I}$  satisfies  $\Phi(\vec{x}) \rightsquigarrow \Psi(\vec{x})$  wrt a database  $\mathcal{S}$ , if the sentence

$$\forall \vec{x} (\Psi(\vec{x}) \rightarrow \Phi(\vec{x}))$$

is true in  $\mathcal{I} \cup \mathcal{S}$ .

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

$$\text{cert}(q, \langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle) = \bigcap_{\mathcal{I} \text{ model of } \langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle} q^{\mathcal{I}}$$

# Abstracting from the mapping

In this talk, we abstract from the mapping, because we want to focus on ontologies.

We assume that  $\mathcal{M}$  is a GAV mapping, and we denote by  $\mathcal{M}(\mathcal{S})$  the database obtained by “transferring” the data from the sources to the alphabet of the ontology.

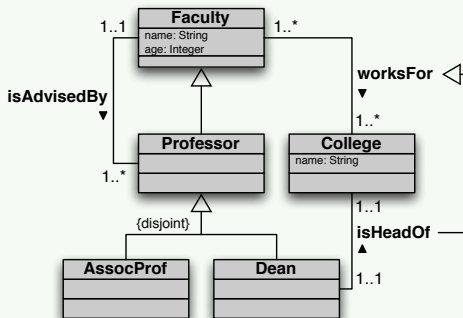
$\mathcal{M}(\mathcal{S})$  can be seen as a set of facts built on the alphabet of  $\mathcal{T}$  (i.e., a set of ground atomic formulas in logic, or simply, an ABox, in DL terminology).

In practice, to go from a query over  $\mathcal{M}(\mathcal{S})$  to a query over  $\mathcal{S}$ , we can rewrite the query based on  $\mathcal{M}$ .

# Ontology-based data access: queries

In principle, we are interested in First-Order Logic (FOL), which is the standard query language for databases.

## Example



$q(nf, af, nd) \leftarrow \text{worksFor}(f, c) \wedge \neg \text{isHeadOf}(d, c) \wedge \text{name}(f, nf) \wedge \text{name}(d, nd) \wedge \text{age}(f, x) \wedge \text{age}(d, x)$

# Query language for user queries

## Fact

Answering FOL queries is **undecidable**, even for very simple ontology and mapping languages.

**Unions of conjunctive queries** (UCQs) do not suffer from this problem. **Conjunctive queries (CQ)** are queries of the form (Datalog notation)

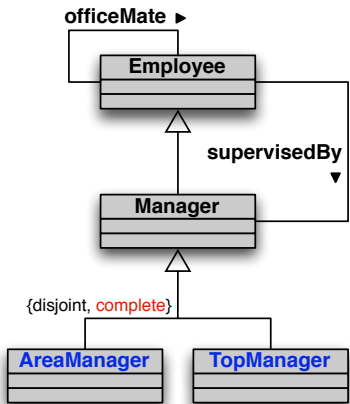
$$q(\vec{x}) \leftarrow R_1(\vec{x}, \vec{y}), \dots, R_k(\vec{x}, \vec{y})$$

where the lhs is the query head, the rhs is the body, and each  $R_i(\vec{x}, \vec{y})$  is an atom using (some of) the free variables  $\vec{x}$ , the existentially quantified variables  $\vec{y}$ , and possibly constants.

- Correspond to SQL/relational algebra **select-project-join (SPJ) queries** – the most frequently asked queries.
- They can also be written as **SPARQL** queries.
- A **Union of CQs (UCQ)** is a set of CQs with the same head predicate.

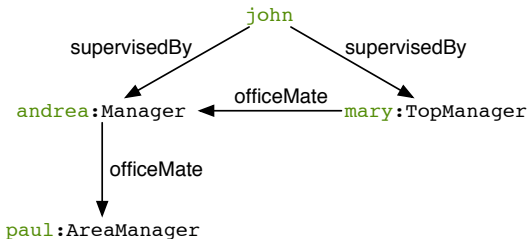


# QA in OBDA – Example<sup>(\*)</sup>



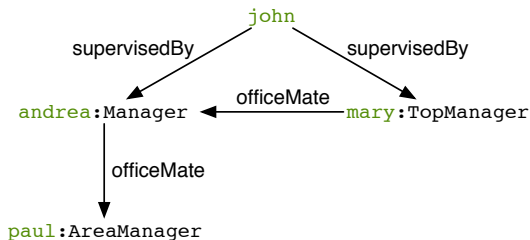
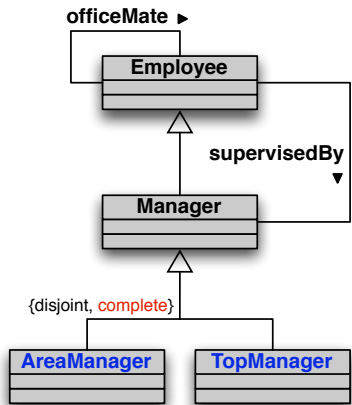
Manager is **partitioned into** AreaManager and TopManager.

- Employee  $\supseteq$  { john }
- Manager  $\supseteq$  { andrea }
- AreaManager  $\supseteq$  { paul }
- TopManager  $\supseteq$  { mary }
- supervisedBy  $\supseteq$  { (john, andrea), (john, mary) }
- officeMate  $\supseteq$  { (mary, andrea), (andrea, paul) }



<sup>(\*)</sup> Due to [Andrea Schaerf 1993].

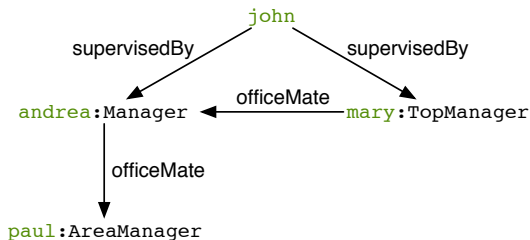
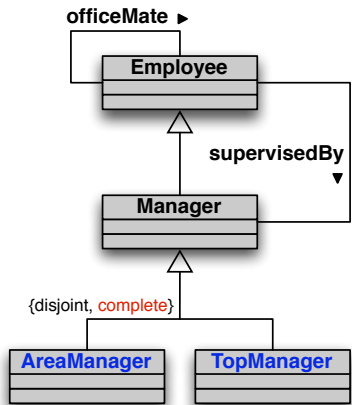
# QA in OBDA – Example (cont'd)



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

Answer: ???

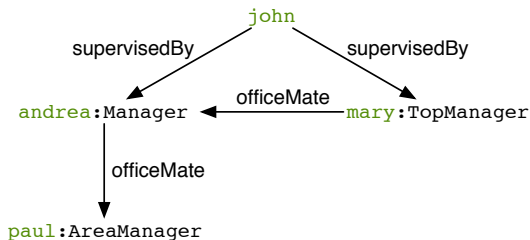
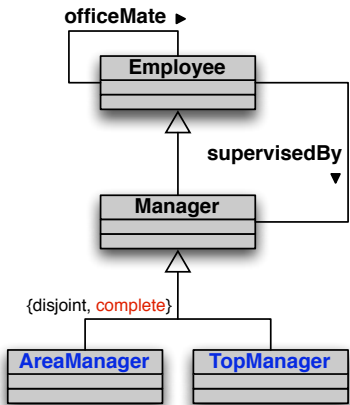
# QA in OBDA – Example (cont'd)



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

Answer: ???

# QA in OBDA – Example (cont'd)

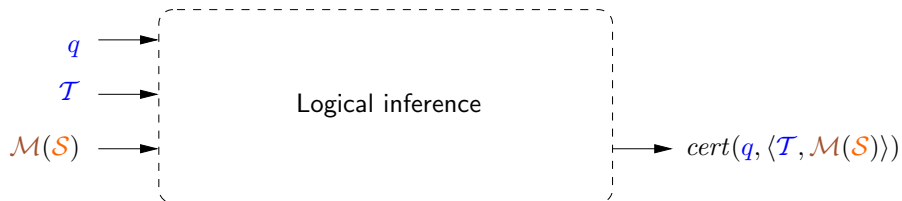


$q(x) \leftarrow \exists y, z. \text{supervisedBy}(x, y), \text{TopManager}(y), \text{officeMate}(y, z), \text{AreaManager}(z)$

Answer: { john }

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

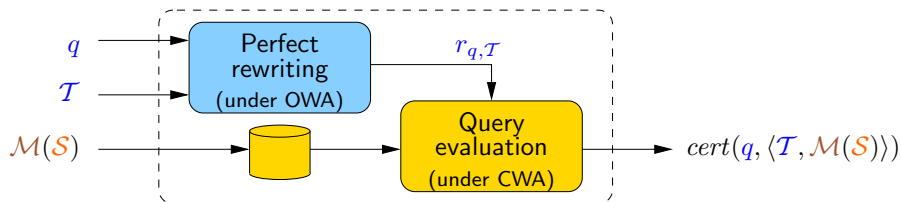
# Inference in query answering



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

→ Query answering by **query rewriting**.

# Query rewriting

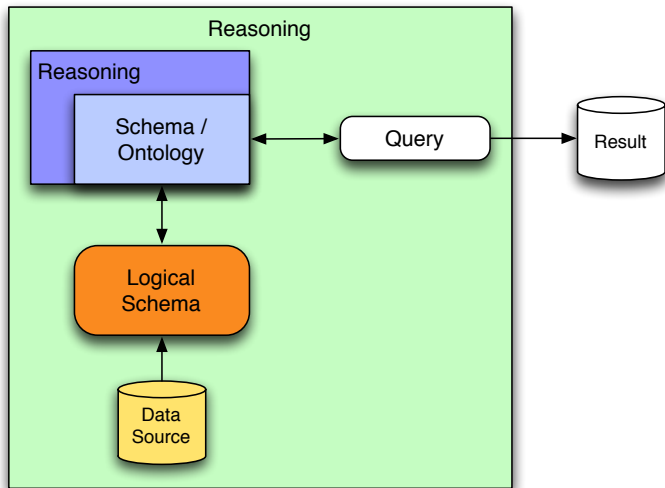


Query answering can **always** be thought as done in two phases:

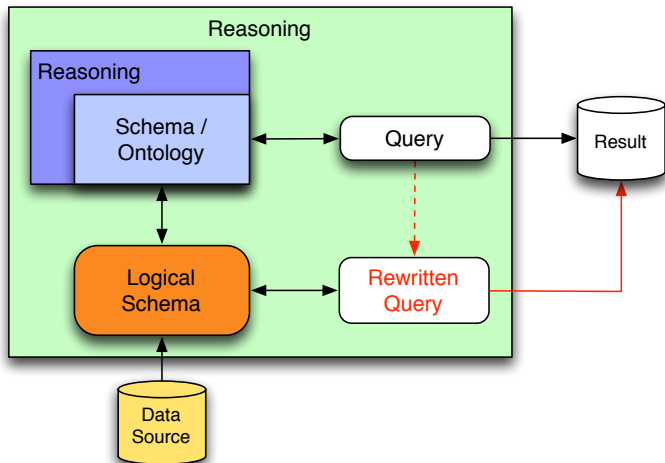
- 1 **Perfect rewriting**: produce from  $q$  and the TBox  $\mathcal{T}$  a new query  $r_{q,\mathcal{T}}$  (called the perfect rewriting of  $q$  w.r.t.  $\mathcal{T}$ ).
- 2 **Query evaluation**: evaluate  $r_{q,\mathcal{T}}$  over the  $\mathcal{M}(\mathcal{S})$  seen as a complete database (and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

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

# Query rewriting (cont'd)



# Query rewriting (cont'd)





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 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.  
↪ Query evaluation requires (at least) **linear recursion**.
- When we can rewrite into a **PTIME-hard** language.  
↪ Query evaluation requires full recursion (e.g., **Datalog**).
- When we can rewrite into a **CONP-hard** language.  
↪ Query evaluation requires (at least) power of **Disjunctive Datalog**.

# 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 <sup>(2)</sup>
OWL 2 (and less)	2EXPTIME-complete	CONP-hard <sup>(1)</sup>

<sup>(1)</sup> Already for a TBox with a single disjunction (see example above). <sup>(2)</sup> 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 in OBDA?

# 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 <sup>(2)</sup>
OWL 2 (and less)	2EXPTIME-complete	<b>coNP-hard</b> <sup>(1)</sup>

<sup>(1)</sup> Already for a TBox with a single disjunction (see example above). <sup>(2)</sup>  
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 in OBDA?

# 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 <sup>(2)</sup>
OWL 2 (and less)	2EXPTIME-complete	coNP-hard <sup>(1)</sup>

<sup>(1)</sup> Already for a TBox with a single disjunction (see example above). <sup>(2)</sup> 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 in OBDA?

# 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 <sup>(2)</sup>
OWL 2 (and less)	2EXPTIME-complete	coNP-hard <sup>(1)</sup>

<sup>(1)</sup> Already for a TBox with a single disjunction (see example above). <sup>(2)</sup> 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 in OBDA?

## TBox assertions:

- Class (concept) inclusion assertions:  $B \sqsubseteq C$ , with:

$$\begin{array}{l} B \longrightarrow A \mid \exists Q \\ C \longrightarrow B \mid \neg B \end{array}$$

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

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

- Functionality assertions: (**funct**  $Q$ )
- Proviso:** functional properties cannot be specialized.

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

*Note:* DL-Lite<sub>A</sub> distinguishes also between object and data properties (ignored here).

# Semantics of $DL-Lite_A$

Construct	Syntax	Example	Semantics
atomic conc.	$A$	Doctor	$A^I \subseteq \Delta^I$
exist. restr.	$\exists Q$	$\exists \text{child}^-$	$\{d \mid \exists e. (d, e) \in Q^I\}$
at. conc. neg.	$\neg A$	$\neg \text{Doctor}$	$\Delta^I \setminus A^I$
conc. neg.	$\neg \exists Q$	$\neg \exists \text{child}$	$\Delta^I \setminus (\exists Q)^I$
atomic role	$P$	child	$P^I \subseteq \Delta^I \times \Delta^I$
inverse role	$P^-$	$\text{child}^-$	$\{(o, o') \mid (o', o) \in P^I\}$
role negation	$\neg Q$	$\neg \text{manages}$	$(\Delta^I \times \Delta^I) \setminus Q^I$
conc. incl.	$B \sqsubseteq C$	$\text{Father} \sqsubseteq \exists \text{child}$	$B^I \subseteq C^I$
role incl.	$Q \sqsubseteq R$	$\text{hasFather} \sqsubseteq \text{child}^-$	$Q^I \subseteq R^I$
funct. asser.	( <b>funct</b> $Q$ )	( <b>funct</b> succ)	$\forall d, e, e'. (d, e) \in Q^I \wedge (d, e') \in Q^I \rightarrow e = e'$
mem. asser.	$A(c)$	$\text{Father}(\text{bob})$	$c^I \in A^I$
mem. asser.	$P(c_1, c_2)$	$\text{child}(\text{bob}, \text{ann})$	$(c_1^I, c_2^I) \in P^I$

$DL-Lite_A$  (as all DLs of the  $DL-Lite$  family) adopts the Unique Name Assumption (UNA), i.e., different individuals denote different objects.

## Capturing basic ontology constructs in $DL-Lite_{\mathcal{A}}$

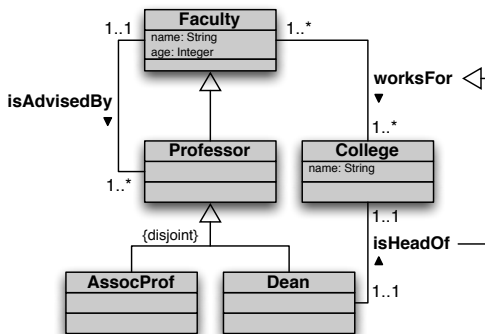
ISA between classes	$A_1 \sqsubseteq A_2$
Disjointness between classes	$A_1 \sqsubseteq \neg A_2$
Domain and range of properties	$\exists P \sqsubseteq A_1 \quad \exists P^- \sqsubseteq A_2$
Mandatory participation ( $min\ card = 1$ )	$A_1 \sqsubseteq \exists P \quad A_2 \sqsubseteq \exists P^-$
Functionality of relations ( $max\ card = 1$ )	<b>(<math>funct\ P</math>)</b> <b>(<math>funct\ P^-</math>)</b>
ISA between properties	$Q_1 \sqsubseteq Q_2$
Disjointness between properties	$Q_1 \sqsubseteq \neg Q_2$

*Note 1:*  $DL-Lite_{\mathcal{A}}$  cannot capture completeness of a hierarchy. This would require **disjunction** (i.e., **OR**).

*Note 2:*  $DL-Lite_{\mathcal{A}}$  can be extended to capture also **min cardinality constraints** ( $A \sqsubseteq \leq n Q$ ), **max cardinality constraints** ( $A \sqsubseteq \geq n Q$ ), **identification assertions**, and **denial assertions** (not considered here for simplicity).



# Example



Professor  $\sqsubseteq$  Faculty  
 AssocProf  $\sqsubseteq$  Professor  
 Dean  $\sqsubseteq$  Professor  
 AssocProf  $\sqsubseteq$   $\neg$ Dean

Faculty  $\sqsubseteq$   $\exists$ age  
 $\exists$ age $^-$   $\sqsubseteq$  xsd:integer  
 (func age)

$\exists$ worksFor  $\sqsubseteq$  Faculty  
 $\exists$ worksFor $^-$   $\sqsubseteq$  College  
 Faculty  $\sqsubseteq$   $\exists$ worksFor  
 College  $\sqsubseteq$   $\exists$ worksFor $^-$

$\exists$ isHeadOf  $\sqsubseteq$  Dean  
 $\exists$ isHeadOf $^-$   $\sqsubseteq$  College  
 Dean  $\sqsubseteq$   $\exists$ isHeadOf  
 College  $\sqsubseteq$   $\exists$ isHeadOf $^-$   
 isHeadOf  $\sqsubseteq$  worksFor  
 (func isHeadOf)  
 (func isHeadOf $^-$ )

⋮

- Completely symmetric w.r.t. **direct and inverse roles**: roles are always navigable in the two directions
- TBoxes may contain **cyclic dependencies** (which typically increase the computational complexity of reasoning)

Example:  $A \sqsubseteq \exists P$ ,  $\exists P^{-} \sqsubseteq A$

- Does **not** enjoy the **finite model property**, unless we drop functional assertions.

## Remark

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

$$B_1 \sqsubseteq B_2, \quad Q_1 \sqsubseteq Q_2$$

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

$$B_1 \sqsubseteq \neg B_2, \quad Q_1 \sqsubseteq \neg Q_2$$

## Theorem

Let  $q$  be a boolean UCQs and  $\mathcal{T} = \mathcal{T}_{\text{PI}} \cup \mathcal{T}_{\text{NI}} \cup \mathcal{T}_{\text{funct}}$  be a TBox s.t.

- $\mathcal{T}_{\text{PI}}$  is a set of PIs
- $\mathcal{T}_{\text{NI}}$  is a set of NIs
- $\mathcal{T}_{\text{funct}}$  is a set of functionalities.

For each  $\mathcal{S}$  such that  $\langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$  is **satisfiable**, we have that

$$\langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle \models q \text{ iff } \langle \mathcal{T}_{\text{PI}}, \mathcal{S}, \mathcal{M} \rangle \models q.$$

In other words, we have that  $\text{cert}(q, \langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle) = \text{cert}(q, \langle \mathcal{T}_{\text{PI}}, \mathcal{S}, \mathcal{M} \rangle)$ .

To the aim of answering queries, from now on we assume that  $\mathcal{T}$  contains only Pls.

Given a CQ  $q$  and a satisfiable  $\langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$ , we compute  $\text{cert}(q, \langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle)$  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  $\mathcal{M}(\mathcal{S})$ .

Correctness of this procedure shows FOL-rewritability of query answering in  $DL-Lite_{\mathcal{A}}$

$\rightsquigarrow$  Query answering over  $DL-Lite_{\mathcal{A}}$  ontologies can be done using RDMBS technology.

$\rightsquigarrow$  System implemented: MASTRO

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

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

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

$$\text{AssProfessor} \sqsubseteq \text{Professor}$$

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

**Basic rewriting step:**

**when** the atom unifies with the **head** of the rule (with mgu  $\sigma$ ).

**substitute** the atom with the **body** of the rule (to which  $\sigma$  is applied).

Towards the computation of the perfect rewriting, we add to the input query above the following query ( $\sigma = \{z/x\}$ )

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

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

Consider now the query

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

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

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

We add to the reformulation the query ( $\sigma = \{z_1/x, z_2/y\}$ )

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

Conversely, for the query

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

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

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

$\text{teaches}(x, \text{databases})$  does not unify with  $\text{teaches}(z_1, z_2)$ , since the **existentially quantified variable**  $z_2$  in the head of the rule **does not unify** with the constant  $\text{databases}$ .

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

The same holds for the following query, where  $y$  is **distinguished**

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

## Query answering in $DL\text{-Lite}_{\mathcal{A}}$ : Query rewriting (cont'd)

An analogous behavior with join variables

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

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

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

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

Conversely, the PI

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

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

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

We add to the perfect rewriting the query ( $\sigma = \{z_2/y\}$ )

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



## Query answering in $DL\text{-Lite}_{\mathcal{A}}$ : Query rewriting (cont'd)

We now have the query

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

The PI

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

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

does not apply to  $\text{teaches}(x, y)$  nor  $\text{teaches}(z, y)$ , since  $y$  is a join variable.

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

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

We can now apply the PI above ( $\sigma\{z_1/x, z_2/y\}$ ), and add to the reformulation the query

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

**Algorithm**  $PerfectRef(q, \mathcal{T}_P)$

**Input:** conjunctive query  $q$ , set of  $DL-Lite_{\mathcal{A}}$  PIs  $\mathcal{T}_P$

**Output:** union of conjunctive queries  $PR$

$PR := \{q\};$

**repeat**

$PR' := PR;$

**for each**  $q \in PR'$  **do**

        (a) **for each**  $g$  in  $q$  **do**

**for each** PI  $I$  in  $\mathcal{T}_P$  **do**

**if**  $I$  is applicable to  $g$

**then**  $PR := PR \cup \{q[g/(g, I)]\}$

        (b) **for each**  $g_1, g_2$  in  $q$  **do**

**if**  $g_1$  and  $g_2$  unify

**then**  $PR := PR \cup \{\tau(reduce(q, g_1, g_2))\};$

**until**  $PR' = PR;$

**return**  $PR$

- 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**  $r_{q,\mathcal{T}}$ .

## Query answering in $DL\text{-Lite}_{\mathcal{A}}$ : Example

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

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

**Perfect Rewriting:**  $q(x) \leftarrow \text{teaches}(x, y), \text{Course}(y)$   
 $q(x) \leftarrow \text{teaches}(x, y), \text{teaches}(z, y)$   
 $q(x) \leftarrow \text{teaches}(x, z)$   
 $q(x) \leftarrow \text{Professor}(x)$

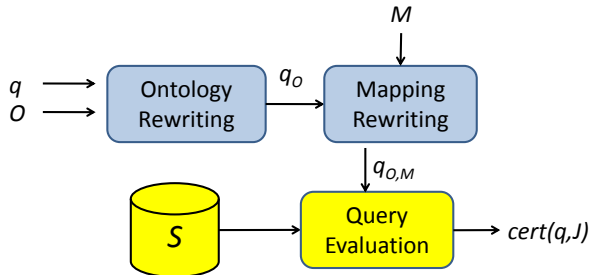
$\mathcal{M}(\mathcal{S})$ :  $\text{teaches}(\text{John}, \text{databases})$   
 $\text{Professor}(\text{Mary})$

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

# Query answering for OBDA

Based on **query rewriting** – given an (U)CQ  $q$ , and  $\mathcal{J} = \langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$ :

- 1 **Perfect reformulation**: rewrite  $q$  into the perfect reformulation  $q_{\mathcal{T}}$  of  $q$  w.r.t.  $\mathcal{T}$ , which turns out to be a UCQ –  $q_{\mathcal{T}}$  is such that  $\text{cert}(q, \langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle) = \text{cert}(q_{\mathcal{T}}, \langle \emptyset, \mathcal{S}, \mathcal{M} \rangle)$
- 2 **Unfolding**: compute the rewriting  $q_{\mathcal{T}, \mathcal{M}}$  of  $q_{\mathcal{T}}$  w.r.t.  $\mathcal{M}$ , which is a query over  $\mathcal{S}$  –  $q_{\mathcal{T}, \mathcal{M}}$  is such that  $\text{cert}(q_{\mathcal{T}}, \langle \emptyset, \mathcal{S}, \mathcal{M} \rangle) = q_{\mathcal{T}, \mathcal{M}}^{\mathcal{S}}$
- 3 **Evaluation**: evaluate  $q_{\mathcal{T}, \mathcal{M}}$  over the source database  $\mathcal{S}$ .



**Complexity:**  $\text{AC}^0$  in the size of the **database**  $\mathcal{S}$ , in fact FOL-rewritable.

# Complexity

$n$  : query size

$m$  : number of predicate symbols in  $\mathcal{T}$  or query  $q$

The number of distinct conjunctive queries generated by the algorithm is less than or equal to  $(m \times (n + 1)^2)^n$ , which corresponds to the maximum number of executions of the repeat-until cycle of the algorithm.

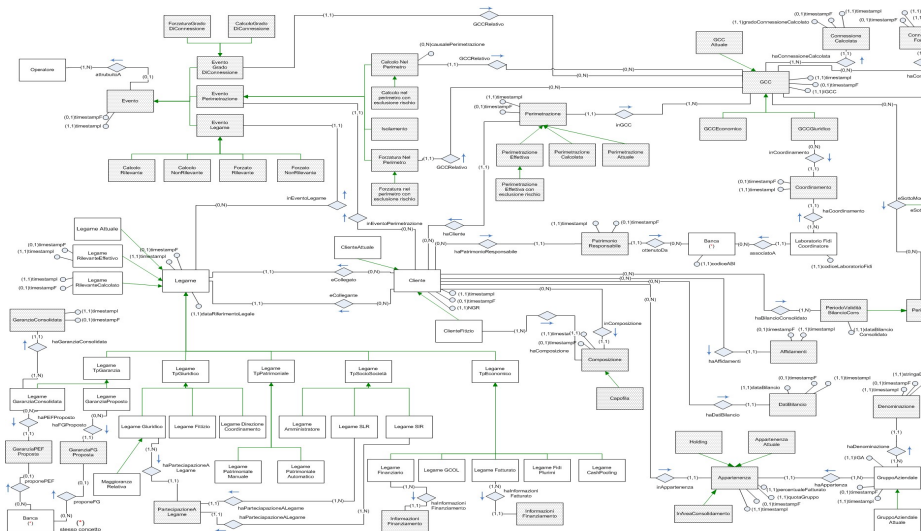
Query answering for CQs and UCQs is:

- **P**TIME in the size of **T**Box.
- **AC**<sup>0</sup> in the size of the  $\mathcal{M}(\mathcal{S})$ .
- Exponential in the size of the **query**.

Can we go beyond *DL-Lite<sub>A</sub>* and remain in **AC**<sup>0</sup>?

By adding essentially any other DL construct (without limitations) we lose this nice computational properties.

# Complexity counts



For realistic ontologies, systems based on *PerfectRef* works for queries with at most 7-8 atoms.

Two sources of complexity wrt query:

- conjunctive query evaluation is **NP-complete** – complexity comes from the need of matching the query and the data  
     $\leadsto$  **unavoidable!**
- the rewritten query has exponential size wrt the original query – complexity comes from the need of comparing the query and the ontology  
     $\leadsto$  **avoidable?**



# Avoiding exponential blow-up: first attempt

Idea: avoid rewriting whatsoever!

Unfortunately, this idea does not work:

Theorem (Calvanese et al, JAR 2007)

Given  $\mathcal{M}(\mathcal{S})$ , there is no database  $B$  such that for every query  $q$ ,  
 $\text{cert}(q, \langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle) = q^B$

## Avoiding exponential blow-up: second attempt

Presto [Rosati, KR 2010] is the current rewriting algorithm used in **Mastro**, based on the following ideas for improving the performance of *PerfectRef*:

- centering the rewriting around the query variables rather than the query atoms – this allows for
  - collapsing sequences of rewriting steps into single steps and
  - dramatically pruning the solution space of the algorithm
- going beyond the disjunctive normal form (UCQ) of the rewritten – query nonrecursive datalog queries are produced

## Avoiding exponential blow-up: second attempt

Presto replaces the “atom-rewrite” and “reduce” rules of *PerfectRef* with a rule (based on MGS) that eliminates existential join variables, where **elimination of an existential join variable** means that the variable turns into a non-join existential variable (through unification steps)

This makes the rewriting produced by Presto **exponential with respect to the number of eliminable existential join variables in the query** (notice: in practice, very often the majority of existential join variables in a CQ are not eliminable)

Previous algorithms produce rewritings whose size is exponential with respect to the number of atoms in the conjunctive query

↪ **dramatic reduction of the size of the query generated. Presto can handle queries with about 30 atoms.**

# Avoiding exponential blow-up: second attempt

Ontology	Query ID	Number of rules/CQs of the generated query					Time (msec) to generate the query				
		QPerfRef	Requiem	RequiemF	RequiemG	PRESTO	QPerfRef	Requiem	RequiemF	RequiemG	PRESTO
V	Q1	15	15	15	15	16	1	1	1	15	1
V	Q2	10	10	10	10	11	31	15	15	15	15
V	Q3	72	576	72	72	28	47	328	655	468	15
V	Q4	185	185	185	185	43	94	78	125	125	15
V	Q5	30	30	30	30	14	140	16	31	15	16
V	Q6	1850	1850	1850	1850	53	920	2341	6428	8909	15
V	Q7	7200	7200	7200	7200	83	3323	36596	114599	327463	15
S	Q1	6	6	6	6	7	1	1	1	15	1
S	Q2	2	160	2	2	3	1	109	140	47	1
S	Q3	4	480	4	4	5	46	1248	1779	171	1
S	Q4	4	960	4	4	5	16	2341	3463	47	15
S	Q5	8	2880	8	8	7	47	51481	75349	296	16
S	Q6	8			8	9	32			60153	16
S	Q7	16			16	12	78			164611	16
P1	Q1	2	2	2	2	3	15	1	1	1	1
P1	Q2	2	2	2	2	3	16	1	1	1	1
P1	Q3	2	2	2	2	3	31	1	1	1	1
P1	Q4	2	2	2	2	3	32	15	16	16	1
P1	Q5	2	2	2	2	3	47	15	16	31	15
P1	Q6	2	32	2	32	3	94	7098	7286	7238	15
P1	Q7	2	64	2	64	3	171	168379	172549	172218	16
P5X	Q1	10	14	14	14	11	15	16	16	15	1
P5X	Q2	50	77	25	25	16	46	46	63	63	1
P5X	Q3	250	390	58	58	16	125	297	499	639	15
P5X	Q4	1254	1953	179	179	16	749	6476	12247	16880	15
P5X	Q5	6330	9766	718	718	16	7239	223955	427426	567713	15
P5X	Q6	32338				16	114233				16
P5X	Q7					16					16

# Avoiding exponential blow-up: second attempt

Ontology	Query ID	Number of rules/CQs of the generated query					Time (msec) to generate the query				
		QPerfRef	Requiem	RequiemF	RequiemG	PRESTO	QPerfRef	Requiem	RequiemF	RequiemG	PRESTO
A	Q1	558	114	27	27	69	171	62	94	78	1
A	Q2	1739	74	50	50	52	592	47	63	94	1
A	Q3	4741	104	104	104	55	2200	94	140	374	15
A	Q4	6589	285	224	224	93	2340	156	234	374	15
A	Q5	66068	624	624	624	71	35365	672	1248	2247	16
A	Q6		2496	2496	2496	91		9221	18799	36443	15
A	Q7					131					31
U	Q1	5	2	2	2	6	1	1	1	1	1
U	Q2	1	148	1	1	1	1	78	93	47	1
U	Q3	12	224	4	4	8	31	156	234	15	16
U	Q4	5	1628	2	2	6	1	1998	4430	78	16
U	Q5	25	2960	10	10	11	31	9953	18157	297	16
U	Q6	40	2368	16	16	14	47	7322	14238	725	16
U	Q7	560	33152		224	28	296	1734331		121888	16

- [Calvanese et al, DL 2011] show how to exploit knowledge about inclusion dependencies on  $\mathcal{M}(\mathcal{S})$  in order to produce more compact rewritings
- Prexto [Rosati, ESWC 2012] applies this idea to Presto.

## Avoiding exponential blow-up: another observation

In the worst case, Presto still rewrites into a union of conjunctive queries of exponential size wrt the original query

Is it avoidable?

Theorem (Kikot et al, DL 2011)

Checking whether  $\vec{t} \in \text{cert}(q, \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle)$  is NP-complete in combined complexity, even if  $\mathcal{T}$  is in  $DL\text{-Lite}_{\mathcal{R}}$ , and  $\mathcal{M}(\mathcal{S})$  is constituted by just one atomic assertion  $A(c)$ .

Since answering a FOL query over a database  $A(c)$  can be done in linear time, it follows that **no algorithm can construct a FOL rewriting in polynomial time, unless  $P = NP$ .**

## Avoiding exponential blow-up: other attempts

- Polynomial FOL rewritings exist in the case of  $DL-Lite_{core}$  ( $DL-Lite_{\mathcal{R}}$  without role inclusions) [Kikot et al, DL 2011]. Unfortunately, our experience shows that role inclusions are very important for modeling real-world domains
- Polynomial rewritings exist if we allow rewriting to be expressed in nonrecursive Datalog queries [Gottlob and Schwentick, DL 2011]. Unclear whether this has a practical application.
- Once you accept to rewrite in a language going beyond FOL, then it makes sense to look at more expressive ontology languages (i.e.,  $\mathcal{EL}$ , Datalog+-)

- Is “first-order rewritability” a real limit that cannot be surpassed by data-intensive ontologies?  $\rightsquigarrow$  **real issue** (open research problem).
- Our opinion:
  - FOL rewritability = reuse of relational database technology for query processing
  - more expressive ontology/query languages necessarily require support for (at least linear) recursion
  - currently, there is no available technology for recursive queries (notwithstanding with negation interpreted under the stable model semantics) that is comparable to SQL technology
- Many **research challenges** remain (i.e., rewriting wrt mappings, updates, service and process management, etc.)