Reasoning for Ontology Engineering and Usage

Part 4: Integrating Data into Ontologies

Diego Calvanese[1], Giuseppe De Giacomo[2],
Demo by Mariano Rodríguez-Muro[1]

[1] Free University of Bozen-Bolzano,

[2] Sapienza Università di Roma

Tutorial at 7th Int. Semantic Web Conference (ISWC 2008)
Karlsruhe – Oct. 27, 2008

Outline

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

Outline

Ontologies and data

- We have seen that current DL reasoning systems can deal with relatively large ABoxes. $\leadsto 10^4$ individuals
- This is small, if compared to data found in various contexts: biological data, scientific data, enterprise data, ...
  $\leadsto 10^5 - 10^9$ 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 sourses to the ontology (cf. Data Integration)

Questions addressed in this part of the tutorial

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

Outline

Ontology languages vs. query languages

Which query language to use?

Two extreme cases:

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

Conjunctive queries (CQs)

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 free variables $\vec{x}$, the existentially quantified variables $\vec{y}$, and possibly constants.

We will also use the simpler Datalog notation:

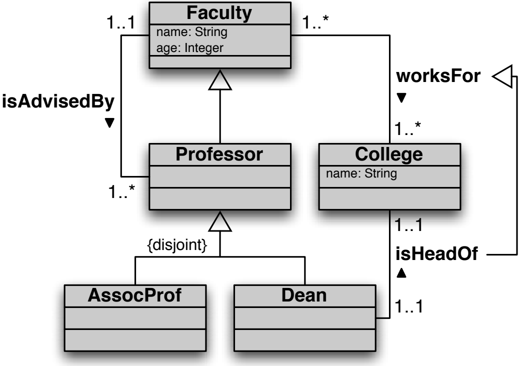
$$q(\vec{x}) \leftarrow R_1(\vec{x},\vec{y}),\ldots,R_k(\vec{x},\vec{y})$$

Note:

- CQs contain no disjunction, no negation, no universal quantification.
- 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

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



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

Conjunctive queries and SQL – Example

Relational alphabet:

worksFor(fac, coll), isHeadOf(dean, coll), name(p, n), age(p, a)

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

Expressed in SQL:

```
SELECT NF.name, AF.age, ND.name
FROM worksFor W, isHeadOf H, name NF, name ND, age AF, age AD
WHERE W.fac = NF.p   AND  W.fac = AF.p  AND
      H.dean = ND.p  AND  H.dean = AD.p  AND
      W.coll = H.coll  AND  AF.a = AD.a
```

Expressed as a CQ:

$$\begin{aligned} q(nf, af, nd) \leftarrow\ & \mathsf{worksFor}(f1, c1),\ \mathsf{isHeadOf}(d1, c2), \\ & \mathsf{name}(f2, nf),\ \mathsf{name}(d2, nd),\ \mathsf{age}(f3, af),\ \mathsf{age}(d3, ad), \\ & f1 = f2,\ f1 = f3,\ d1 = d2,\ d1 = d3,\ c1 = c2,\ af = ad \end{aligned}$$

Query answering under different assumptions

There are fundamentally different assumptions when addressing query answering in different settings:

- **traditional database assumption**
- **knowledge representation assumption**

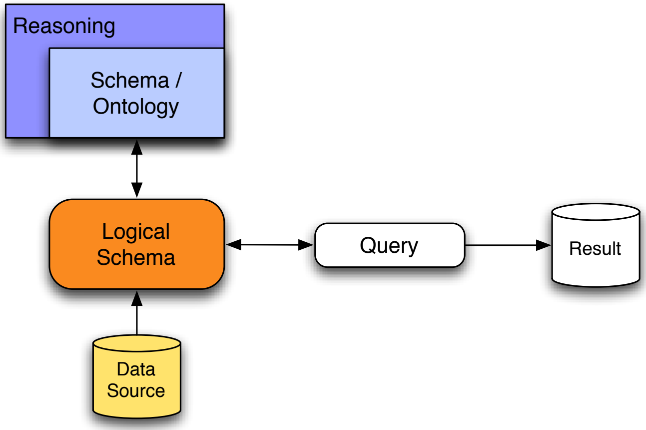
Note: for the moment we assume to deal with an ordinary ABox, which however may be very large and thus is stored in a database.

Query answering under the database assumption

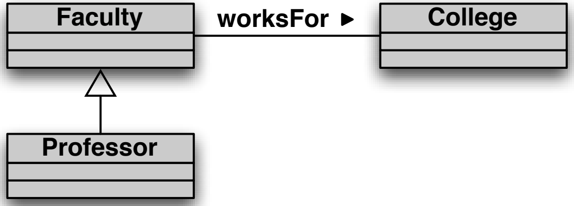
- Data are completely specified (CWA), and typically large.
- Schema/intensional information used in the design phase.
- At **runtime**, the data is assumed to satisfy the schema, and therefore the **schema is not used**.
- Queries allow for complex navigation paths in the data (cf. SQL).

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

Query answering under the database assumption (cont'd)



Query answering under the database assumption – Example



For each class/property we have a (complete) table in the database.

DB: Faculty = { john, mary, nick }
Professor = { john, nick }
College = { collA, collB }
worksFor = { (john,collA), (mary,collB) }

Query: $q(x) \leftarrow \exists c.\,\mathsf{Professor}(x), \mathsf{College}(c), \mathsf{worksFor}(x, c)$

Answer: { john }

Query answering under the KR assumption

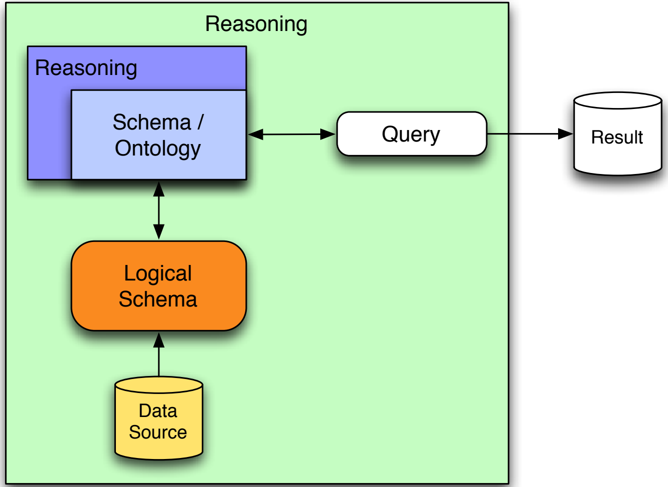
- An ontology imposes constraints on the data.
- Actual data may be incomplete or inconsistent w.r.t. such constraints.
- The system has to take into account the constraints during query answering, and overcome incompleteness or inconsistency.

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

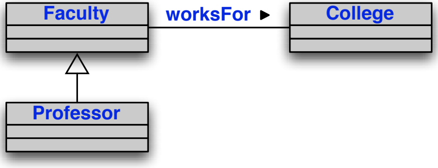
Note:

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

Query answering under the KR assumption (cont'd)



Query answering under the KR assumption – Example



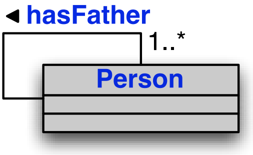
The tables in the database may be **incompletely specified**, or even missing for some classes/properties.

DB: Professor $\supseteq$ { john, nick }
College $\supseteq$ { collA, collB }
worksFor $\supseteq$ { (john,collA), (mary,collB) }

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

Answer: { john, nick, mary }

Query answering under the KR assumption – Example 2



Each person has a father, who is a person.

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

Queries: $q_1(x,y) \leftarrow \text{hasFather}(x,y)$

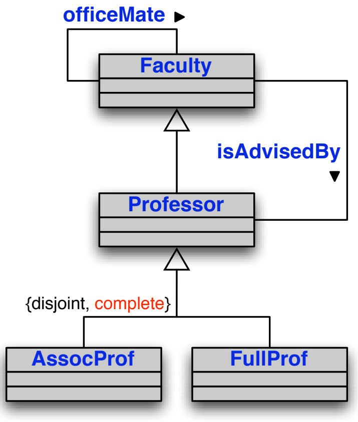
$q_2(x) \leftarrow \exists y.\, \text{hasFather}(x,y)$

$q_3(x) \leftarrow \exists y_1,y_2,y_3.\, \text{hasFather}(x,y_1), \text{hasFather}(y_1,y_2), \text{hasFather}(y_2,y_3)$

$q_4(x,y_3) \leftarrow \exists y_1,y_2.\, \text{hasFather}(x,y_1), \text{hasFather}(y_1,y_2), \text{hasFather}(y_2,y_3)$

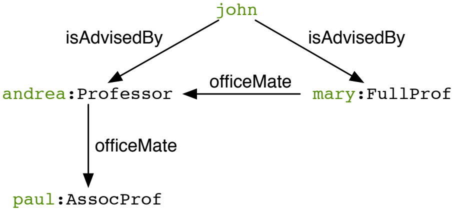
Answers: to q_1: { (`john`,`nick`), (`nick`,`toni`) }
to q_2: { `john`, `nick`, `toni` }
to q_3: { `john`, `nick`, `toni` }
to q_4: { }

QA under the KR assumption – Andrea's Example

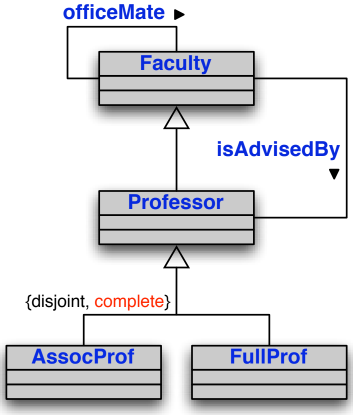


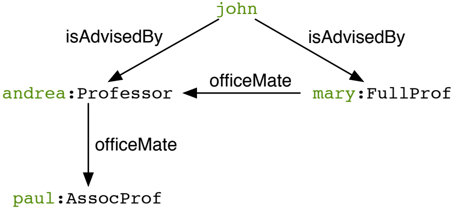
$$\mathsf{FullProf} \equiv \mathsf{AssocProf} \sqcup \mathsf{FullProf}$$

$$\begin{aligned}
\mathsf{Faculty} &\supseteq \{\, \mathtt{andrea}, \mathtt{nick}, \mathtt{mary}, \mathtt{john} \,\} \\
\mathsf{Professor} &\supseteq \{\, \mathtt{andrea}, \mathtt{nick}, \mathtt{mary} \,\} \\
\mathsf{AssocProf} &\supseteq \{\, \mathtt{nick} \,\} \\
\mathsf{FullProf} &\supseteq \{\, \mathtt{mary} \,\} \\
\mathsf{isAdvisedBy} &\supseteq \{\, (\mathtt{john,andrea}), (\mathtt{john,mary}) \,\} \\
\mathsf{officeMate} &\supseteq \{\, (\mathtt{mary,andrea}), (\mathtt{andrea,nick}) \,\}
\end{aligned}$$



QA under the KR assumption – Andrea's Example (cont'd)





$$q() \leftarrow \exists y, z.\ \text{isAdvisedBy}(\texttt{john}, y),\ \text{FullProf}(y),\ \text{officeMate}(y, z),\ \text{AssocProf}(z)$$

Answer: **yes!**

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

Query answering when accessing data through ontologies

We have to face the difficulties of both assumptions:

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

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 $cert(q, \mathcal{O})$

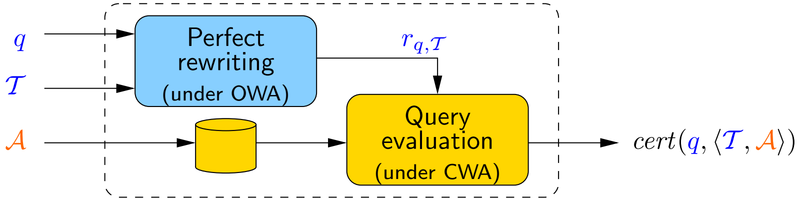
... are the **tuples $\vec{c}$ of constants of $\mathcal{A}$** such that $\vec{c} \in q^{\mathcal{I}}$, for every model $\mathcal{I}$ of $\mathcal{O}$.

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

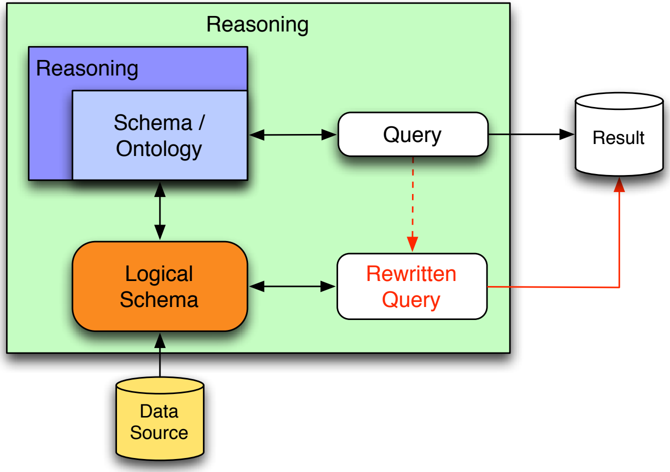


Query answering can **always** be thought as done in two phases:

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

Query rewriting (cont'd)



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**.
  $\leadsto$ 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.
  $\leadsto$ Query evaluation requires (at least) linear recursion.
- When we can rewrite into a PTime-hard language.
  $\leadsto$ Query evaluation requires full recursion (e.g., Datalog).
- When we can rewrite into a coNP-hard language.
  $\leadsto$ Query evaluation requires (at least) power of Disjunctive Datalog.

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 (see Andrea's example).
(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?

Outline

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

$\textit{DL-Lite}_{\mathcal{A}}$ provides robust foundations for Ontology-Based Data Access.

TBox assertions:

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

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

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

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

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

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

Note: *DL-Lite*$_{\mathcal{A}}$ distinguishes also between object and data properties (ignored here).

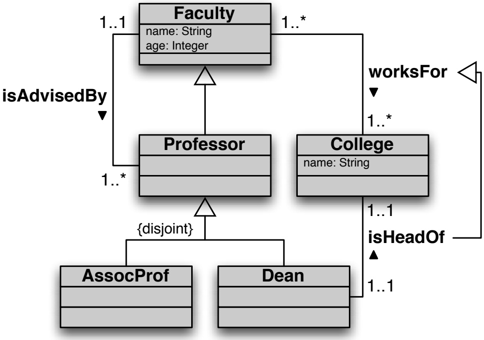
Semantics of the *DL-Lite*$_{\mathcal{A}}$ assertions

Assertion	Syntax	Example	Semantics
class incl.	$B \sqsubseteq C$	Father $\sqsubseteq$ $\exists$child	$B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$
o-prop. incl.	$Q \sqsubseteq R$	father $\sqsubseteq$ anc	$Q^{\mathcal{I}} \subseteq R^{\mathcal{I}}$
v.dom. incl.	$E \sqsubseteq F$	ρ(age) $\sqsubseteq$ `xsd:int`	$E^{\mathcal{I}} \subseteq F^{\mathcal{I}}$
d-prop. incl.	$U \sqsubseteq V$	offPhone $\sqsubseteq$ phone	$U^{\mathcal{I}} \subseteq V^{\mathcal{I}}$
o-prop. funct.	(**funct** Q)	(**funct** father)	$\forall o, o, o''.(o, o') \in Q^{\mathcal{I}} \land (o, o'') \in Q^{\mathcal{I}} \rightarrow o' = o''$
d-prop. funct.	(**funct** U)	(**funct** ssn)	$\forall o, v, v'.(o, v) \in U^{\mathcal{I}} \land (o, v') \in U^{\mathcal{I}} \rightarrow v = v'$
mem. asser.	$A(c)$	Father(`bob`)	$c^{\mathcal{I}} \in A^{\mathcal{I}}$
mem. asser.	$P(c_1, c_2)$	child(`bob`, `ann`)	$(c_1^{\mathcal{I}}, c_2^{\mathcal{I}}) \in P^{\mathcal{I}}$
mem. asser.	$U(c, d)$	phone(`bob`, `'2345'`)	$(c^{\mathcal{I}}, \mathit{val}(d)) \in U^{\mathcal{I}}$

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)*	$(\textbf{funct}\ P) \quad (\textbf{funct}\ P^-)$
ISA between properties	$Q_1 \sqsubseteq Q_2$
Disjointness between properties	$Q_1 \sqsubseteq \neg Q_2$

Example



$$
\begin{array}{rcl}
\text{Professor} & \sqsubseteq & \text{Faculty} \\
\text{AssocProf} & \sqsubseteq & \text{Professor} \\
\text{Dean} & \sqsubseteq & \text{Professor} \\
\text{AssocProf} & \sqsubseteq & \neg\text{Dean} \\
\\
\text{Faculty} & \sqsubseteq & \exists \mathit{age} \\
\exists \mathit{age}^- & \sqsubseteq & \texttt{xsd:int} \\
 & (\textbf{funct}\ \mathit{age}) & \\
\\
\exists\text{worksFor} & \sqsubseteq & \text{Faculty} \\
\exists\text{worksFor}^- & \sqsubseteq & \text{College} \\
\text{Faculty} & \sqsubseteq & \exists\text{worksFor} \\
\text{College} & \sqsubseteq & \exists\text{worksFor}^- \\
\\
\exists\text{isHeadOf} & \sqsubseteq & \text{Dean} \\
\exists\text{isHeadOf}^- & \sqsubseteq & \text{College} \\
\text{Dean} & \sqsubseteq & \exists\text{isHeadOf} \\
\text{College} & \sqsubseteq & \exists\text{isHeadOf}^- \\
\text{isHeadOf} & \sqsubseteq & \text{worksFor} \\
 & (\textbf{funct}\ \text{isHeadOf}) & \\
 & (\textbf{funct}\ \text{isHeadOf}^-) & \\
 & \vdots &
\end{array}
$$

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

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

- Captures all the basic constructs of **UML Class Diagrams** and of the **ER Model** ...
- ... **except covering constraints** in generalizations.
- Is **one of** the three candidate **OWL 2 Profiles**.
- Extends (the DL fragment of) the ontology language **RDFS**.
- Is completely symmetric w.r.t. **direct and inverse properties**.
- Does **not** enjoy the **finite model property**, i.e., reasoning and query answering differ depending on whether we consider or not also infinite models.

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

Based on query reformulation: given an (U)CQ and an ontology:

1. **Compute its perfect rewriting**, which turns out to be a UCQ.
2. **Evaluate the perfect rewriting** on the ABox seen as a DB.

To compute the perfect rewriting, starting from the original (U)CQ, iteratively get a CQ to be processed and either:

- **expand** positive inclusions & **simplify** redundant atoms, or
- **unify** atoms in the CQ to obtain a more specific CQ to be further expanded.

Each result of the above steps is added to the queries to be processed.

Note: negative inclusions and functionalities play a role in ontology satisfiability, but not in query answering.

Query answering in $\mathit{DL\text{-}Lite}_{\mathcal{A}}$ – Example

TBox: $\mathsf{Professor} \sqsubseteq \exists \mathsf{worksFor}$
$\exists \mathsf{worksFor}^- \sqsubseteq \mathsf{College}$

Query: $\mathsf{q}(x) \leftarrow \mathsf{worksFor}(x, y), \mathsf{College}(y)$

Perfect Reformulation:
$$\begin{aligned}
\mathsf{q}(x) &\leftarrow \mathsf{worksFor}(x, y), \mathsf{College}(y)\\
\mathsf{q}(x) &\leftarrow \mathsf{worksFor}(x, y), \mathsf{worksFor}(_, y)\\
\mathsf{q}(x) &\leftarrow \mathsf{worksFor}(x, _)\\
\mathsf{q}(x) &\leftarrow \mathsf{Professor}(x)
\end{aligned}$$

ABox: $\mathsf{worksFor}(\mathtt{john}, \mathtt{collA})$ $\quad$ $\mathsf{Professor}(\mathtt{john})$
$\mathsf{worksFor}(\mathtt{mary}, \mathtt{collB})$ $\quad$ $\mathsf{Professor}(\mathtt{nick})$

Evaluating the last two queries over the ABox (seen as a DB) produces as answer $\{\mathtt{john}, \mathtt{nick}, \mathtt{mary}\}$.

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

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).

In fact, reasoning can be done by constructing suitable FOL/SQL queries and evaluating them over the ABox (**FOL-rewritability**).

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*$_{\mathcal{A}}$?

No! By adding essentially any additional constructor we lose these nice computational properties.

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

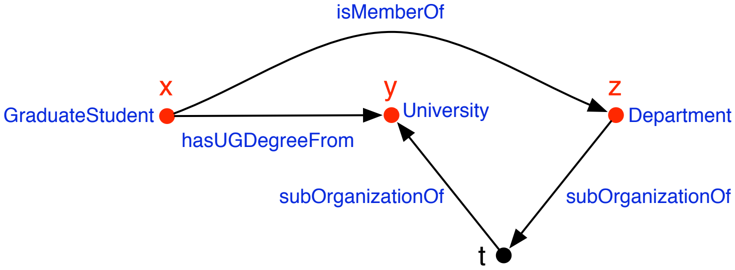
	lhs	rhs	funct.	Prop. incl.	Data complexity of query answering
0	*DL-Lite*$_{\mathcal{A}}$		$\surd^*$	$\surd^*$	in LogSpace
1	$A \mid \exists P.A$	A	–	–	NLogSpace-hard
2	A	$A \mid \forall P.A$	–	–	NLogSpace-hard
3	A	$A \mid \exists P.A$	$\surd$	–	NLogSpace-hard
4	$A \mid \exists P.A \mid A_1 \sqcap A_2$	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**

Notes:

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

Example of query

$$q(x,y,z) \leftarrow \text{GraduateStudent}(x),\ \text{University}(y),\ \text{Department}(z),\ \text{hasUndergraduateDegreeFrom}(x,y),\ \text{isMemberOf}(x,z),\ \text{subOrganizationOf}(z,t),\ \text{subOrganizationOf}(t,y)$$




```
SELECT ?X ?Y ?Z WHERE
  ?X rdf:type 'GraduateStudent'  .  ?Y rdf:type 'University' .
  ?Z rdf:type 'Department'  .
  ?X :hasUndergraduateDegreeFrom ?Y  .  ?X :isMemberOf ?Z  .
  ?Z subOrganizationOf ?T  .  ?T subOrganizationOf ?Y
```

Outline

Ontologies in *DL-Lite*$_{\mathcal{A}}$

As said, *DL-Lite*$_{\mathcal{A}}$ can be seen as a fragment of OWL 2 specifically designed to deal with **large amounts of data**:

- It allows for **query answering** (and checking ontology satisfiability) in LogSpace wrt the size of the data (i.e., the ABox).
- Reasoning with data in *DL-Lite*$_{\mathcal{A}}$ can be **delegated to a relational DBMS**.
- *DL-Lite*$_{\mathcal{A}}$ captures all constructs in typical conceptual models, such as **UML Class Diagrams** and **ER**.

Ontology interoperation

Next we look at **semantical interoperation** between an **ontology-based system** and **external systems**, such as traditional information systems.

- In doing this, we consider for concreteness an actual scenario, where semantic interoperation is the core issue ...
  $\leadsto$ ... **data integration** ...
- ... and we show the notable advantages that ontologies can bring to this scenario.

Data integration

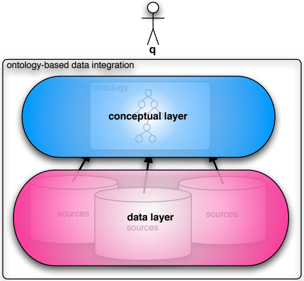
Data integration is the problem of providing unified and transparent access to a set of autonomous and heterogeneous sources.

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

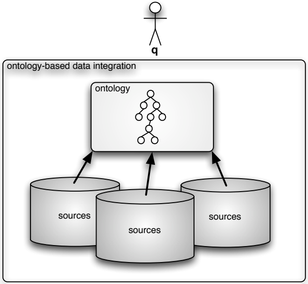
Ontology-based data integration is based on the idea of decoupling information access from data storage.



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

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

Ontology-based data integration: architecture

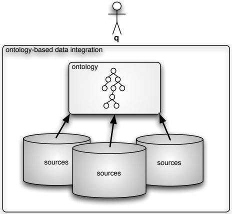


Based on three main components:

- **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.

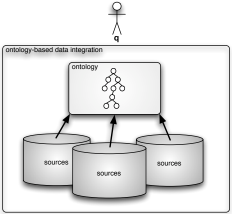


Note: in standard information systems, UML Class Diagram or ER is used at **design time**, ...

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

Ontology-based data integration: the sources

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

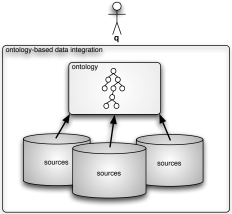


By now we have industrial solutions for:

- Distributed database systems
- Distributed query optimization
- Tools for source wrapping
- Systems for database federation, e.g., IBM Information Integrator
  but notice no open-source federated databases yet!

Ontology-based data integration: the sources

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



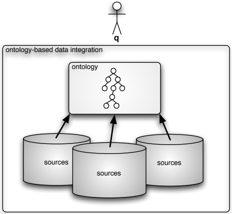
Based on these industrial solutions we can:

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.

$\leadsto$ 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.



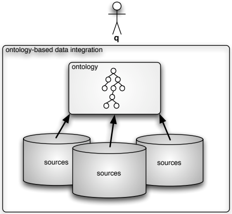
Scientific literature on data integration in databases has shown that ...

... 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.



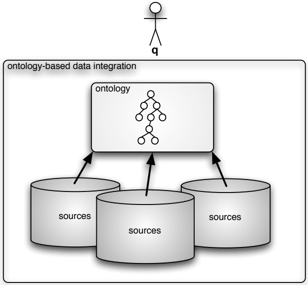
Several general forms of mappings based on queries have been considered:

- 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 practically useless (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).

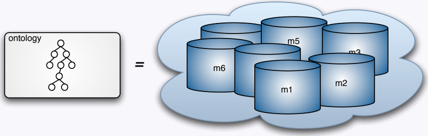
Ontology-based data integration: incomplete information

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



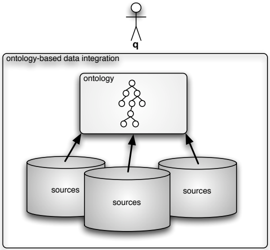
Important

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



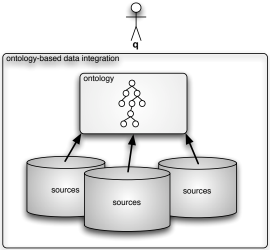
- Query answering amounts to compute **certain answers**, given the global view, the mapping and the data at the sources ...
- ... but query answering may be costly in ontologies (even without mapping and sources).

Ontology-based data integration: the *DL-Lite*$_{\mathcal{A}}$ 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**$_{\mathcal{A}}$ 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*$_{\mathcal{A}}$ 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. $\rightsquigarrow$ 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 exploit the distinction between **objects** and **values** in *DL-Lite*$_{\mathcal{A}}$ to 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: the technical solution

Let Γ_V be the alphabet of constants (values) appearing in the sources. We introduce a new alphabet Λ of function symbols, where each function symbol has an associated arity, specifying the number of arguments it accepts.

We inductively define the set $\tau(\Lambda, \Gamma_V)$ of all (Skolem) terms of the form $f(d_1, \ldots, d_n)$ such that

- $f \in \Lambda$,
- the arity of f is $n > 0$, and
- $d_1, \ldots, d_n \in \Gamma_V$.

We use $\tau(\Lambda, \Gamma_V)$ to denote the instances of concepts in the ontology. The unique name assumption is now enforced on such a set.

$\leadsto$ *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*

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

where

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

*Note: this is a form of GAV mapping

Example

Let $\mathcal{S}$ be the database constituted by a set of relations with the signature:

D_1[SSN,PROJ,D], D_2[SSN,NAME], D_3[CODE,NAME], D_4[CODE,SSN]

- Relation D_1 stores tuples (s, p, d), where s and p are strings and d is a date, such that s is the social security number of a temporary employee, p is the name of the project s/he works for (different projects have different names), and d is the ending date of the employment.
- Relation D_2 stores tuples (s, n) of strings consisting of the social security number s of an employee and her/his name n.
- Relation D_3 stores tuples (c, n) of strings consisting of the code c of a manager and her/his name n.
- Relation D_4 relates managers' code with their social security number.

Example (cont'd)

Consider the ontology with mappings $\mathcal{O}_m = \langle \mathcal{T}, \mathcal{M}, DB \rangle$ such that $\mathcal{T}$ is

$$
\begin{array}{ll}
\mathsf{TempEmp} \sqsubseteq \mathsf{Employee} & \\
\mathsf{Manager} \sqsubseteq \mathsf{Employee} & \mathsf{Project} \sqsubseteq \exists \mathit{projName} \\
\mathsf{Employee} \sqsubseteq \mathsf{Person} & (\mathbf{funct}\ \mathit{projName}) \\
\mathsf{Employee} \sqsubseteq \exists \mathsf{worksFor} & \mathsf{TempEmp} \sqsubseteq \exists \mathit{until} \\
\exists \mathsf{worksFor}^- \sqsubseteq \mathsf{Project} & \exists \mathit{until} \sqsubseteq \exists \mathsf{worksFor} \\
\mathsf{Person} \sqsubseteq \exists \mathit{persName} & (\mathbf{funct}\ \mathit{until}) \\
(\mathbf{funct}\ \mathit{persName}) & \mathsf{Manager} \sqsubseteq \neg \exists \mathit{until}
\end{array}
$$

and $\mathcal{M}$ is defined by using $\Lambda = \{\mathbf{pers}, \mathbf{proj}, \mathbf{mgr}\}$, all of which are function symbols of arity 1.

Example (cont'd)

Mapping assertions $\mathcal{M}$:

M_{m_1} :	SELECT SSN,PROJ,D FROM D_1	$\leadsto$	TempEmp(**pers**(SSN)), worksFor(**pers**(SSN), **proj**(PROJ)), *projName*(**proj**(PROJ), PROJ), *until*(**pers**(SSN), D)
M_{m_2} :	SELECT SSN,NAME FROM D_2	$\leadsto$	Employee(**pers**(SSN)), *persName*(**pers**(SSN), NAME)
M_{m_3} :	SELECT SSN, NAME FROM D_3, D_4 WHERE D_3.CODE=D_4.CODE	$\leadsto$	Manager(**pers**(SSN)), *persName*(**pers**(SSN), NAME)
M_{m_4} :	SELECT CODE, NAME FROM D_3 WHERE CODE NOT IN (SELECT CODE FROM D_4)	$\leadsto$	Manager(**mgr**(CODE)), *persName*(**mgr**(CODE), NAME)

Semantics of mappings

Def.: Semantics of mappings

We say that $\mathcal{I}$ **satisfies** $\Phi(\vec{x}) \leadsto \Psi(f(\vec{x}, \vec{x})$ **wrt a database** $\mathcal{S}$**, if for every tuple of values** $\vec{v}$ **such that** $\vec{v}$ **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

An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ 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 $\mathcal{M}$ w.r.t. $\mathcal{S}$, i.e., satisfies every assertion in $\mathcal{M}$ w.r.t. $\mathcal{S}$.

An ontology with mappings is **satisfiable** if it admits at least one model.

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

Given a (U)CQ q and $\mathcal{O} = \langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$ (assumed to be satisfiable), we compute **certain answers** $cert(q, \mathcal{O})$ 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.
$\leadsto$ Query answering can again be done using **RDBMS technology**.
$\leadsto$ Prototype system implemented: Quonto+Integration Module

Example

$\mathcal{M}$ can be encoded in the following portion of a logic program:

$$
\begin{array}{lcl}
\mathsf{TempEmp}(\mathbf{pers}(s)) & \leftarrow & Aux_{11}(s) \\
\mathsf{worksFor}(\mathbf{pers}(s), \mathbf{proj}(p)) & \leftarrow & Aux_{12}(s,p) \\
\mathit{projName}(\mathbf{proj}(p), p) & \leftarrow & Aux_{13}(p) \\
\mathit{until}(\mathbf{pers}(s), d) & \leftarrow & Aux_{14}(s,d) \\
\mathsf{Employee}(\mathbf{pers}(s)) & \leftarrow & Aux_{21}(s) \\
\mathit{persName}(\mathbf{pers}(s), n) & \leftarrow & Aux_{22}(s,n) \\
\mathsf{Manager}(\mathbf{pers}(s)) & \leftarrow & Aux_{31}(s) \\
\mathit{persName}(\mathbf{pers}(s), n) & \leftarrow & Aux_{32}(s,n) \\
\mathsf{Manager}(\mathbf{mgr}(c)) & \leftarrow & Aux_{41}(c) \\
\mathit{persName}(\mathbf{mgr}(c), n) & \leftarrow & Aux_{42}(c,n)
\end{array}
$$

where Aux_{ij} is a predicate denoting the result of the evaluation over $\mathcal{S}$ of the query $\Phi_{m_{ij}}$ in the left-hand side of the mapping $M_{m_{ij}}$.

Example (cont'd)

Consider $q(x) \leftarrow \text{worksFor}(x, y)$, whose reformulation $Q' = r_{q,\mathcal{T}}$ is:

$$\begin{array}{lcl} Q'(x) & \leftarrow & \text{worksFor}(x, y) \\ Q'(x) & \leftarrow & \textit{until}(x, y) \\ Q'(x) & \leftarrow & \text{TempEmp}(x) \\ Q'(x) & \leftarrow & \text{Employee}(x) \\ Q'(x) & \leftarrow & \text{Manager}(x) \end{array}$$

To compute the unfolding of Q', we unify each of its atoms with the left-hand side of the logic program rules corresponding to the mapping assertions in $\mathcal{M}$, and we obtain the following *partial evaluation* of Q':

$$\begin{array}{lcl} q(\textbf{pers}(s)) & \leftarrow & Aux_{12}(s, p) \\ q(\textbf{pers}(s)) & \leftarrow & Aux_{14}(s, d) \\ q(\textbf{pers}(s)) & \leftarrow & Aux_{11}(s) \\ q(\textbf{pers}(s)) & \leftarrow & Aux_{21}(s) \\ q(\textbf{pers}(s)) & \leftarrow & Aux_{31}(s, n) \\ q(\textbf{mgr}(c)) & \leftarrow & Aux_{41}(c, n) \end{array}$$

Example (cont'd)

From the above formulation, it is now possible to derive the corresponding SQL query Q'' that can be directly issued over the database $\mathcal{S}$:

```
SELECT concat(concat('pers (',SSN),')')
FROM D1
UNION
SELECT concat(concat('pers (',SSN),')')
FROM D2
UNION
SELECT concat(concat('pers (',SSN),')')
FROM D3, D4
WHERE D3.CODE=D4.CODE
UNION
SELECT concat(concat('mgr (',CODE),')')
FROM D3
WHERE CODE NOT IN (SELECT CODE FROM D4)
```

Computational complexity of query answering

Theorem

Query answering in a *DL-Lite*$_{\mathcal{A}}$ 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?
No, if we want to stay in $\textsc{LogSpace}$.

Acknowledgements

People involved in this work:

- Domenico Lembo
- Maurizio Lenzerini
- Marco Ruzzi
- Antonella Poggi
- Mariano Rodriguez Muro
- Riccardo Rosati
- Several master and PhD students, including the ones currently active:
  - Claudio Corona
  - Emma Di Pasquale
  - Fabio Savo

Outline

References I

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

[2] 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/Vol-216/, 2006.

[3] 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/Vol-188/, 2005.

References II

[4] 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.

[5] 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.

[6] 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.

[7] 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 III

[8] 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/Vol-258/`, 2007.

[9] 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.

[10] 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 OWL: Experiences and Directions 2008 (OWLED 2008 DC) Workshop*, 2008.

References IV

[11] 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.