Query Answering and Rewriting in Ontology-based Data Access

Riccardo Rosati

DIAG, Sapienza Università di Roma

SAPIENZA
UNIVERSITÀ DI ROMA

KR 2014, Vienna, July 20, 2014

Outline

- Ontology-based Query Answering (OBQA)
  - problem, languages, example, some complexity results
- The query rewriting approach
  - the idea, FO-rewritability
- Query rewriting in OBQA
  - PerfectRef, results, problems, Requiem, Presto, Rapid, ...
- Ontology-based Data Access (OBDA)
  - problem, languages, example, some complexity results
- Query rewriting in OBDA
  - mapping unfolding, example, problem, optimizations

Outline

Description Logics

Description Logics 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.

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: e.g., (**functional** P)

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)

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-Lite*$_{\mathcal{A}}$, an expressive member of the *DL-Lite* family.
- *DL-Lite*$_{\mathcal{A}}$ provides robust foundations for Ontology-Based Data Access.

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

TBox assertions:

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

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

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

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

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

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

Construct	Syntax	Example	Semantics
atomic conc.	A	Doctor	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
exist. restr.	$\exists Q$	$\exists \text{child}^-$	$\{d \mid \exists e.\, (d, e) \in Q^{\mathcal{I}}\}$
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.	$B \sqsubseteq C$	Father $\sqsubseteq \exists$child	$B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$
role incl.	$Q \sqsubseteq R$	hasFather $\sqsubseteq \text{child}^-$	$Q^{\mathcal{I}} \subseteq R^{\mathcal{I}}$
funct. asser.	(**funct** Q)	(**funct** succ)	$\forall d, e, e'.(d, e) \in Q^{\mathcal{I}} \wedge (d, e') \in Q^{\mathcal{I}} \rightarrow e = e'$
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$_{\mathcal{A}}$ (as all DLs of the *DL-Lite* family) adopts the Unique Name Assumption (UNA), i.e., different individuals denote different objects.

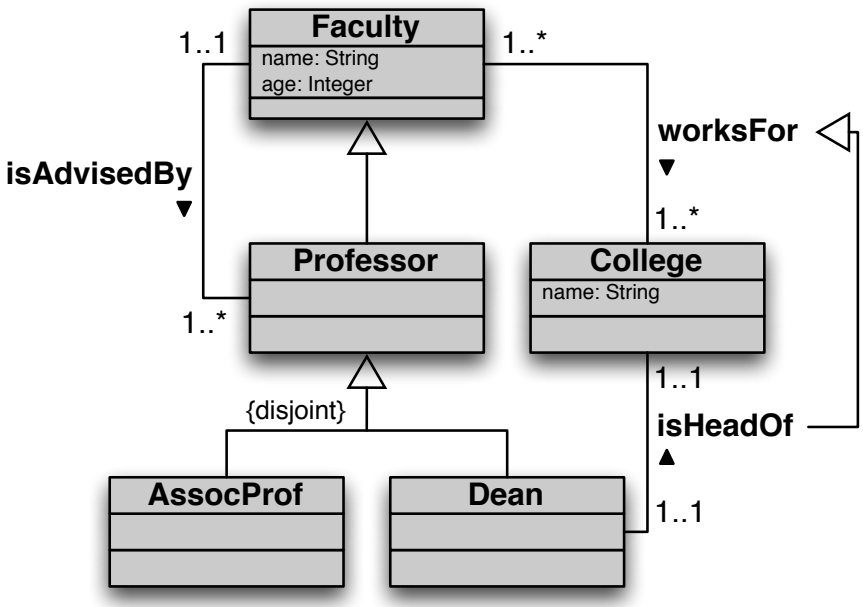
Capturing basic ontology constructs in *DL-Lite*$_{\mathcal{A}}$

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)*	(**funct** P) (**funct** P^-)
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**).

Note2: *DL-Lite*$_{\mathcal{A}}$ can be extended to capture also **min cardinality constraints** ($A \sqsubseteq \leq nQ$) and **max cardinality constraints** ($A \sqsubseteq \geq nQ$) (not considered here for simplicity).

Example



$$
\begin{aligned}
\mathit{Professor} &\sqsubseteq \mathit{Faculty}\\
\mathit{AssocProf} &\sqsubseteq \mathit{Professor}\\
\mathit{Dean} &\sqsubseteq \mathit{Professor}\\
\mathit{AssocProf} &\sqsubseteq \neg \mathit{Dean}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{Faculty} &\sqsubseteq \exists \mathit{age}\\
\exists \mathit{age}^{-} &\sqsubseteq \texttt{xsd:integer}\\
&(\textbf{funct}\ \mathit{age})
\end{aligned}
$$

$$
\begin{aligned}
\exists \mathit{worksFor} &\sqsubseteq \mathit{Faculty}\\
\exists \mathit{worksFor}^{-} &\sqsubseteq \mathit{College}\\
\mathit{Faculty} &\sqsubseteq \exists \mathit{worksFor}\\
\mathit{College} &\sqsubseteq \exists \mathit{worksFor}^{-}
\end{aligned}
$$

$$
\begin{aligned}
\exists \mathit{isHeadOf} &\sqsubseteq \mathit{Dean}\\
\exists \mathit{isHeadOf}^{-} &\sqsubseteq \mathit{College}\\
\mathit{Dean} &\sqsubseteq \exists \mathit{isHeadOf}\\
\mathit{College} &\sqsubseteq \exists \mathit{isHeadOf}^{-}\\
\mathit{isHeadOf} &\sqsubseteq \mathit{worksFor}\\
&(\textbf{funct}\ \mathit{isHeadOf})\\
&(\textbf{funct}\ \mathit{isHeadOf}^{-})\\
&\vdots
\end{aligned}
$$

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 the logical underpinning of **OWL2 QL**, one of the 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.

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 functional assertion (**functional** P) is satisfied by $\mathcal{I}$ if the relation $P^{\mathcal{I}}$ is a (partial) function.

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

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.
- 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, Hermit, Pellet, Racer, Fact++, . . .)

Queries over DL ontologies

- Ontology-based Query Answering: answering queries over TBox + ABox
- query languages:
  conjunctive queries (CQ), **unions of CQ (UCQ)**
- CQ: expression of the form

$$\underset{(head)}{q(t_1, \ldots, t_n)} \leftarrow \underset{(body)}{\alpha_1, \ldots, \alpha_m}$$

  - α_i is either a concept atom $C(t)$ or a role atom $R(t_1, t_2)$
  - every term t_i is either a variable or an individual name
  - every variable occurring in the head also occurs in the body
  - n (number of arguments in the head) is the arity of the CQ
- UCQ: set of CQs of the same arity
- Boolean (U)CQ: CQs without variables in the head
- semantics: **certain answers**

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

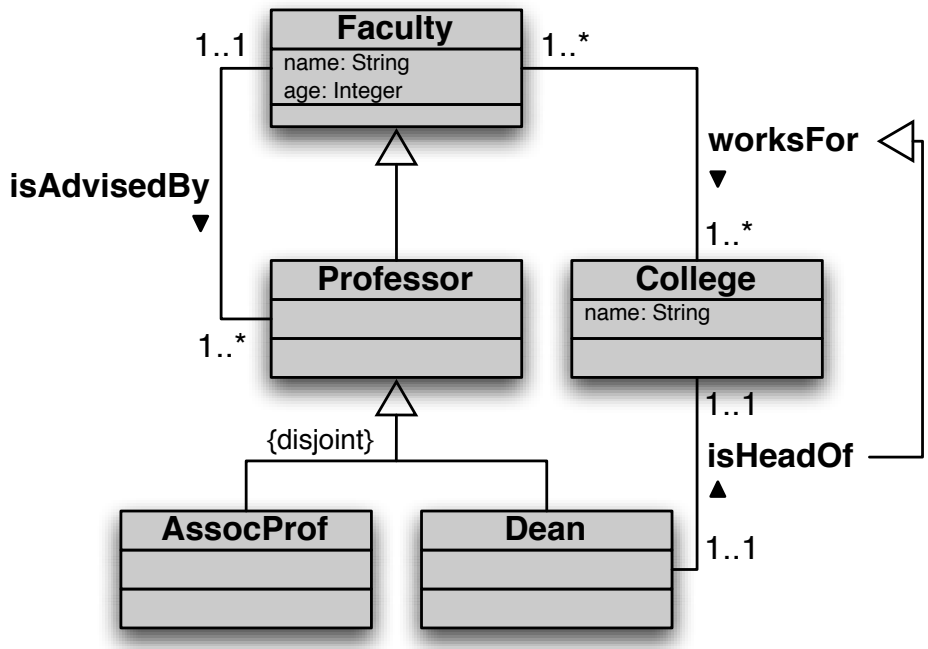
Def.: The **certain answers** to $q(\vec{x})$ over $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, denoted $cert(q, \mathcal{O})$

... 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: when q is boolean, 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.

Example of conjunctive query

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



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

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:

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

OBQA vs. QA over relational databases (summary)

similarities:

- ABox = database instance
- TBox = integrity constraints over the DB schema (e.g., keys, foreign keys)
- UCQ is a subclass of relational algebra and SQL

OBQA vs. QA over relational databases (summary)

differences:

- syntax: DB allows for predicates of **arbitrary arity**, only **unary and binary** predicates allowed by DL
- syntax: different classes of axioms/constraints allowed
- semantics: OWA vs. CWA
  - DB assumes data is **complete**
  - DL assumes the ABox (and the TBox too) is an **incomplete** specification of the world
  - DB has a **single model** (the DB istance itself)
  - KB has **multiple models**
- semantics: finite vs. infinite interpretation structures
  - DB interpreted over a **finite** model, KB interpreted over (possibly) **infinite** models

Query answering under different assumptions

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

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

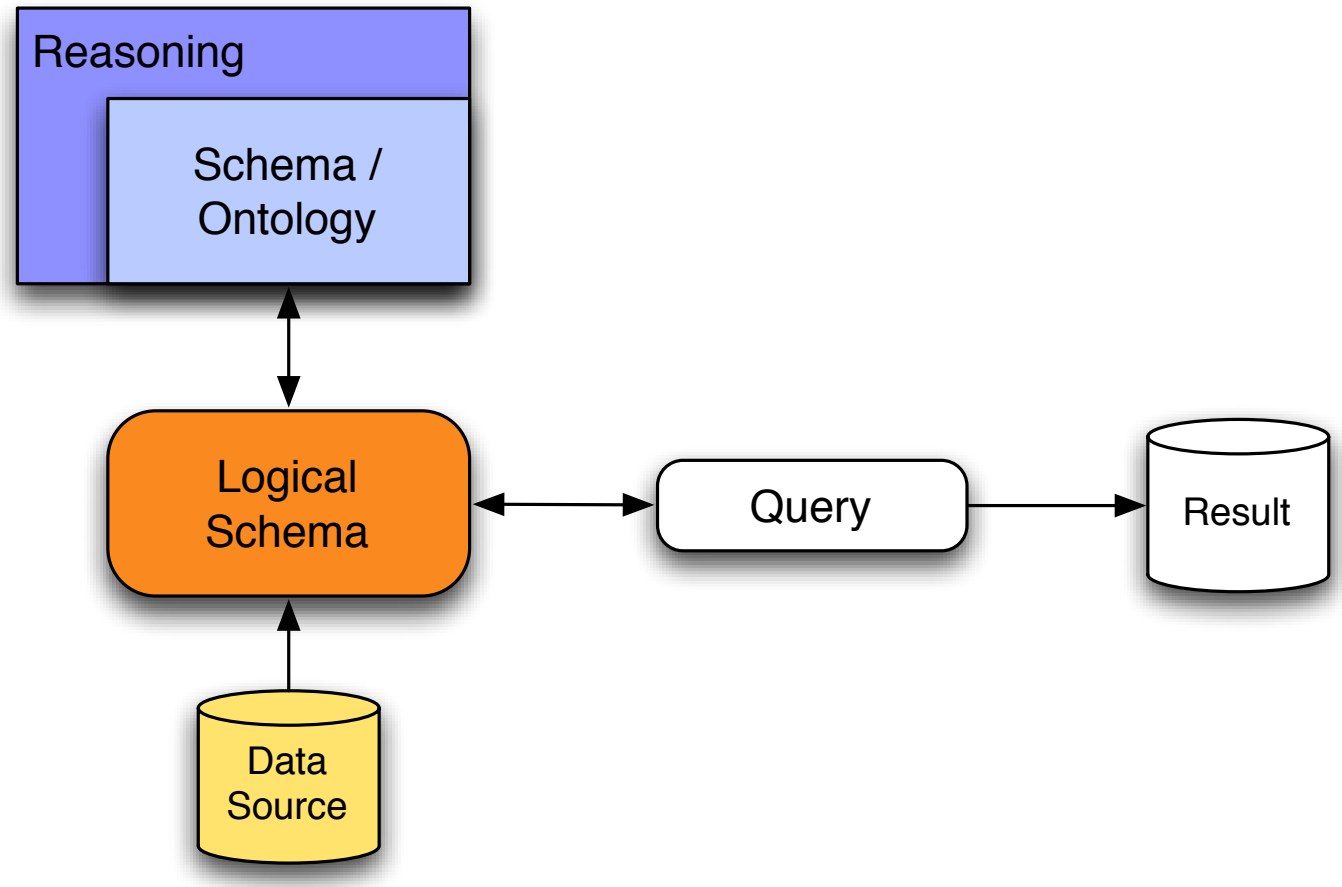
Note: for the moment we assume to deal with an ordinary ABox, which however may be very large and thus is stored in a database.

Query answering under the database assumption

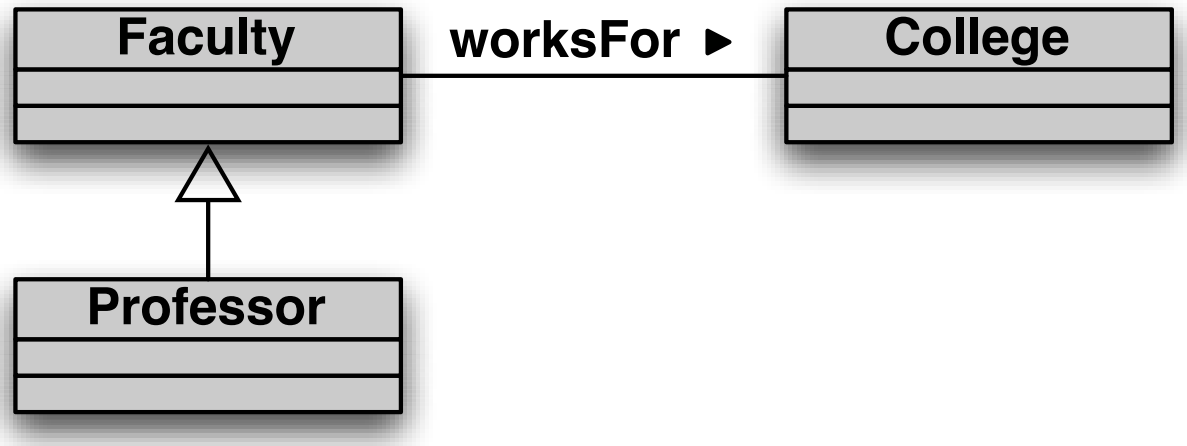
- Data are completely specified (CWA), and typically large.
- Schema/intensional information used in the design phase.
- At **runtime**, the data is assumed to satisfy the schema, and therefore the **schema is not used**.
- Queries allow for complex navigation paths in the data (cf. SQL).

$\rightsquigarrow$ 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`, `paul` }
Professor = { `john`, `paul` }
College = { `collA`, `collB` }
worksFor = { (`john`,`collA`), (`mary`,`collB`) }

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

Answer: { `john` }

Query answering under the KR assumption

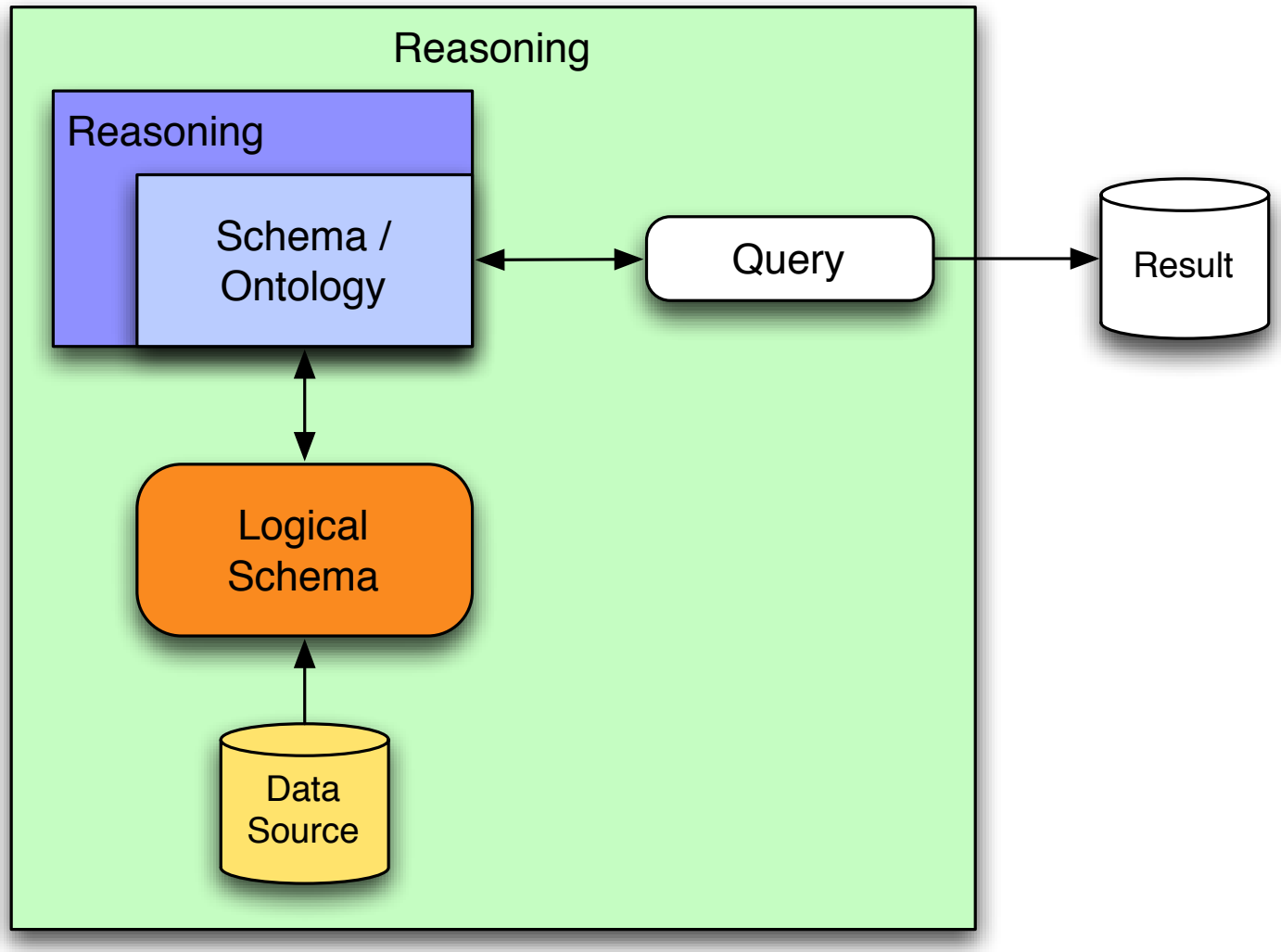
- an ontology imposes constraints on the data.
- actual data may be incomplete or inconsistent w.r.t. such constraints.
- the system has to take into account the constraints during query answering, and overcome incompleteness or inconsistency.
- **implicit** answers (besides the ones explicitly stored in the data) can be retrieved

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

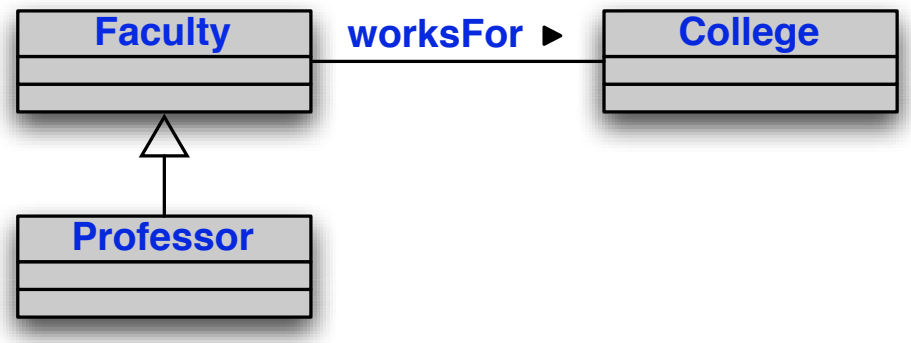
Note:

- 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



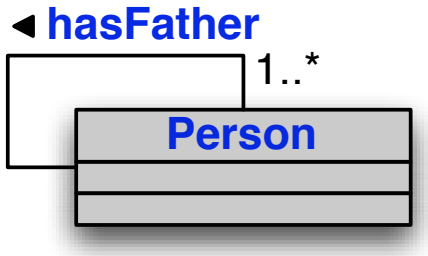
The tables in the database may be **incompletely specified**, or even missing for some classes/properties.

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

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

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

Query answering under the KR assumption – Example 2



Each person has a father, who is a person.

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

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

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

Complexity of OBQA

Various parameters affect the complexity of query answering over an ontology. We get different complexity measures:

- **Data complexity**: only the size of the ABox matters. TBox and query are considered fixed.
- **Schema complexity**: only the size of the TBox matters. ABox and query are considered fixed.
- **Combined complexity**: no parameter is considered fixed.

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

$\rightsquigarrow$ We consider **data complexity** as the relevant complexity measure.

Some decidability and complexity results

- CARIN [Levy & Rousset, 1996]: decidability of CQ answering in $\mathcal{ALCNR}$
- decidability of CQ answering in $\mathcal{DLR}$ [Calvanese et al., 1998]
- tractability (FO-rewritability) of CQ answering in *DL-Lite* [Calvanese et al., 2005;2007]
- complexity of CQ answering in the extended *DL-Lite* family [Artale et al., 2009]
- tractability of CQ answering in $\mathcal{EL}$ [Lutz, 2007; R., 2007]
- tractability of CQ answering in Horn-$\mathcal{SHIQ}$ [Eiter et al., 2008]
- complexity of CQ answering for expressive non-Horn DLs [Lutz, 2008]
- SHIQ, SHOIQ [Glimm et al, 2008; Ortiz et al., 2009; Glimm et al., 2014]
- decidability of CQ answering in OWL 2 still unknown

Outline

Query answering techniques

Query answering in OBQA requires to derive implicit extensional information using the TBox

One can think of solving OBQA through this simple strategy:

1. first "expand" the ABox computing all the extensional consequences of the TBox and the ABox
2. then, discard the TBox and evaluate (in the standard database way) the query on the ABox

Unfortunately, for many DLs this might be too expensive, or even impossible

Expanding the ABox

Example in $\mathit{DL\text{-}Lite}_{\mathcal{A}}$:

$$\mathcal{T} = \{\mathit{Person} \sqsubseteq \exists \mathit{hasFather},\ \exists \mathit{hasFather}^- \sqsubseteq \mathit{Person}\}$$
$$\mathcal{A} = \{\mathit{Person}(\mathit{joe})\}$$

Expansion of $\mathcal{A}$:

$$\mathcal{A}_1 = \mathcal{A} \cup \{\mathit{hasFather}(\mathit{joe}, n_1)\} \quad \text{due to } \mathit{Person} \sqsubseteq \exists \mathit{hasFather}$$
$$\mathcal{A}_2 = \mathcal{A}_1 \cup \{\mathit{Person}(n_1)\} \quad \text{due to } \exists \mathit{hasFather}^- \sqsubseteq \mathit{Person}$$
$$\mathcal{A}_3 = \mathcal{A}_2 \cup \{\mathit{hasFather}(a, n_2)\} \quad \text{due to } \mathit{Person} \sqsubseteq \exists \mathit{hasFather}$$
$$\mathcal{A}_4 = \mathcal{A}_3 \cup \{\mathit{Person}(n_2)\} \quad \text{due to } \exists \mathit{hasFather}^- \sqsubseteq \mathit{Person}$$
$$\mathcal{A}_5 = \ldots$$

In this case, an ABox $\mathcal{A}'$ such that, for every CQ q, $\mathit{ans}(q, \mathcal{A}') = \mathit{cert}(q, \langle \mathcal{T}, \mathcal{A} \rangle)$, **must necessarily be infinite**

The chase and the canonical model

- this expansion of $\mathcal{A}$ w.r.t. $\mathcal{T}$ is called **the chase** of $\langle \mathcal{T}, \mathcal{A} \rangle$
- the chase produces a so-called **canonical model** of $\langle \mathcal{T}, \mathcal{A} \rangle$, i.e., an ABox $\mathcal{A}'$ such that, for every CQ q, $ans(q, \mathcal{A}') = cert(q, \langle \mathcal{T}, \mathcal{A} \rangle)$
- the canonical model always exists for *DL-Lite*$_\mathcal{A}$ and for all Horn DLs
- however, for *DL-Lite*$_\mathcal{A}$ (and for many other Horn DLs) the canonical model may be **infinite** (due to the presence of cyclic inclusion axioms in the TBox)
- **for non-Horn DLs, the canonical model does not exist** as soon as there are "disjunctive" axioms in the TBox
- in DLs, the existence of the canonical model is tightly related to the tractability of conjunctive query answering (w.r.t. data complexity)

To materialize or not to materialize?

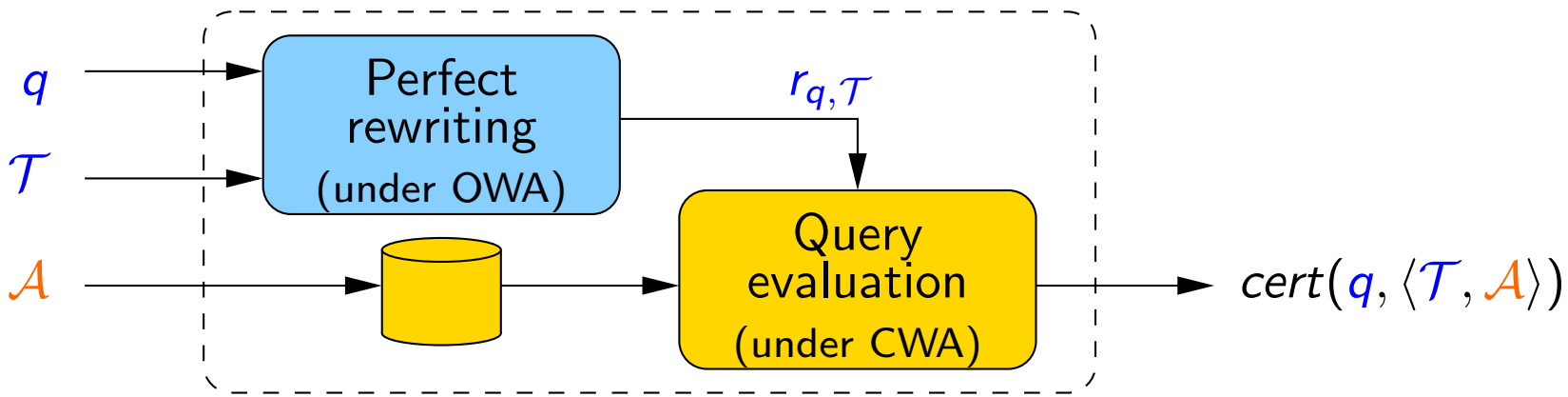
- for the above reasons, many approaches to OBQA do not materialize the canonical model
- instead, they adopt an alternative reasoning strategy based on **query rewriting**
- main advantage: **data structures are not changed by OBQA**, the approach is completely **virtual**
- from now on, we will focus on these approaches
- however, interesting approaches take a **combined** approach that mix (partial) materialization of the canonical model with query rewriting
- in this way it is also possible to go beyond FO-rewritable languages [Lutz et al., 2009;2010;2013]

Inference in query answering

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

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

Query rewriting

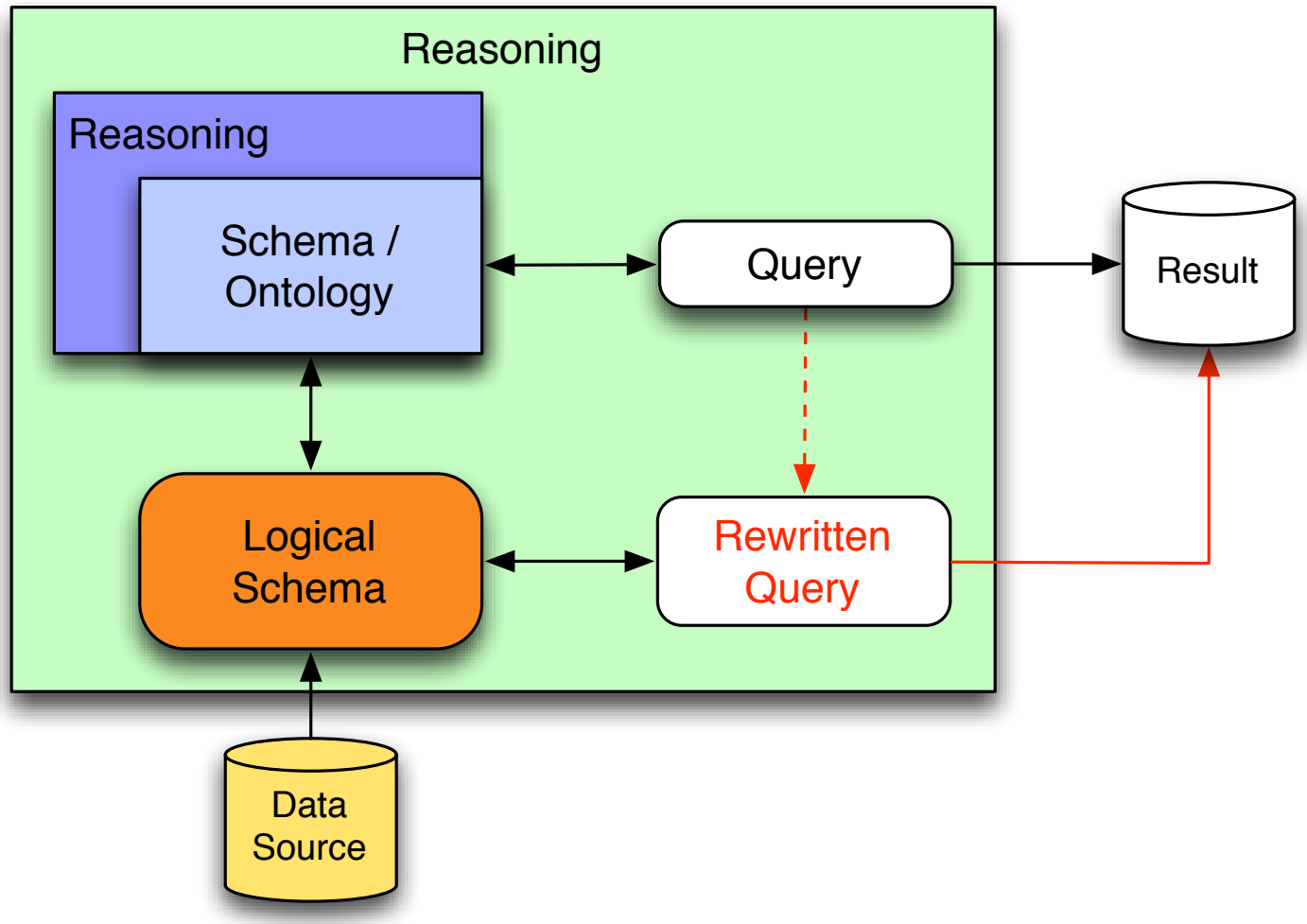


Query answering can **always** be thought as done in two phases:

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

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

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**.
  $\rightsquigarrow$ Query evaluation can be done in SQL, i.e., via an **RDBMS** (*Note:* FOL is in AC^0).
- When we can rewrite into an NLogSpace-hard language.
  $\rightsquigarrow$ 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.

Complexity of query answering in DLs

The rewriting problem 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	AC^0 (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 AC^0)?
- If yes, can we leverage relational database technology for query answering?

Outline

Query rewriting for OBQA

Overview:

- query rewriting for *DL-Lite*$_{\mathcal{A}}$:
  - query rewriting for ontology satisfiability
  - query rewriting for query answering
  - PerfectRef
  - Presto
  - Requiem
  - Rapid
  - incremental query rewriting
- a glimpse beyond *DL-Lite*$_{\mathcal{A}}$

Query rewriting for *DL-Lite*$_{\mathcal{A}}$: Rewriting query atoms

- chase of the ABox = forward chaining
  query rewriting = backward chaining
- essentially, most query rewriting techniques iteratively apply a resolution rule to "expand" the initial query
- e.g., from axiom $C \sqsubseteq D$, i.e., sentence $\forall x(\neg C(x) \vee D(x))$
  and query $q(x) \leftarrow D(x)$
  through resolution we can derive the new query $q(x) \leftarrow C(x)$
- resolution is specialized to the particular class of formulas involved (TBox axioms, CQ)

AtomRewrite: Rewriting query atoms in *DL-Lite*$_{\mathcal{A}}$

AtomRewrite rule: use every positive inclusion axiom as a predicate rewriting rule (from right to left)
e.g.: AtomRewrite uses axiom $C \sqsubseteq D$ to derive $C(x)$ from $D(x)$

Arguments are not affected by the rewriting (they are only propagated)

We can rewrite a role using a concept only if the argument projected out is an existential variable with a single occurrence in the query

e.g.: in $q(x) \leftarrow R(x, y), S(x, z), D(z)$

- we can apply $C \sqsubseteq \exists R$ to atom $R(x, y)$ and generate atom $C(x)$
- we **cannot** apply $D \sqsubseteq \exists S$ to atom $S(x, z)$

AtomRewrite

- for each atom, AtomRewrite can generate at most a linear number of rewritings (w.r.t. TBox size)
- but: the whole rewriting process generates an UCQ having an exponential number of CQs w.r.t. the number of atoms of the initial query

Rewriting query atoms is not enough

Example:

$$\text{TBox: } \mathcal{T} = \{C \sqsubseteq \exists R,\ R \sqsubseteq S\}$$
$$\text{query: } q(x, y) \leftarrow R(x, z), S(y, z)$$

AtomRewrite can only rewrite $S(y, z)$ producing $R(y, z)$. So the rewritten query q' is

$$q'(x, y) \leftarrow R(x, z), S(y, z)$$
$$q'(x, y) \leftarrow R(x, z), R(y, z)$$

this UCQ is not a perfect rewriting:

$$\text{ABox: } \mathcal{A} = \{C(a)\}$$

$\langle a, a \rangle \in \mathit{cert}(q, \langle \mathcal{T}, \mathcal{A} \rangle)$, while q' has no answers over $\mathcal{A}$

the CQ missed by the rewriting is $q(x, x) \leftarrow C(x)$

PerfectRef in a nutshell

PerfectRef [Calvanese et al., 2005] is an algorithm that takes as input a *DL-Lite*$_{\mathcal{A}}$ TBox $\mathcal{T}$ and a CQ q and returns an UCQ q'

q' is computed starting from the UCQ $Q = \{q\}$ and expanding Q by exhaustively applying, to every CQ in Q, the following two rewriting steps:

- AtomRewrite
- Reduce

the Reduce step takes as input a CQ q: if q contains two unifiable atoms with MGU μ, it returns the query $\mu(q)$

PerfectRef in a nutshell

Example (cont.):

$$\text{TBox: } \mathcal{T} = \{C \sqsubseteq \exists R,\ R \sqsubseteq S\}$$
$$\text{query: } q(x, y) \leftarrow R(x, z), S(y, z)$$

1) an AtomRewrite step rewrites $S(z, y)$ using $C \sqsubseteq \exists R$, generating the CQ

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

2) a Reduce step takes the above query and generates the CQ

$$q'(x, x) \leftarrow R(x, z)$$

3) an AtomRewrite step takes the above query and (through $C \sqsubseteq \exists R$) generates the previously missing CQ

$$q'(x, x) \leftarrow C(x)$$

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

- We study answering of UCQs over *DL-Lite*$_{\mathcal{A}}$ 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

$$\begin{array}{lcl} B_1 & \sqsubseteq & B_2 \\ Q_1 & \sqsubseteq & Q_2 \end{array}$$

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

$$\begin{array}{lcl} B_1 & \sqsubseteq & \neg B_2 \\ Q_1 & \sqsubseteq & \neg Q_2 \end{array}$$

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

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 ABox $\mathcal{A}$ such that $\langle \mathcal{T}, \mathcal{A} \rangle$ **is satisfiable**, we have that

$$\langle \mathcal{T}, \mathcal{A} \rangle \models q \textbf{ iff } \langle \mathcal{T}_{\textbf{PI}}, \mathcal{A} \rangle \models q.$$

Proof [intuition]

q is a positive query, i.e., it does not contain atoms with negation nor inequality. $\mathcal{T}_{\text{NI}}$ and $\mathcal{T}_{\text{funct}}$ only contribute to infer new negative consequences, i.e, sentences involving negation.

If q is non-boolean, we have that
$cert(q, \langle \mathcal{T}, \mathcal{A} \rangle) = cert(q, \langle \mathcal{T}_{\text{PI}}, \mathcal{A} \rangle)$.

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

$\langle \mathcal{T}, \emptyset \rangle$ is always satisfiable. That is, inconsistency in *DL-Lite*$_\mathcal{A}$ may arise only when ABox assertions contradict the TBox.

$\langle \mathcal{T}_{\text{PI}}, \mathcal{A} \rangle$, where $\mathcal{T}_{\text{PI}}$ contains only PIs, is always satisfiable. That is, inconsistency in *DL-Lite*$_\mathcal{A}$ may arise only when ABox assertions violate functionalities or NIs.

Example: **TBox** $\mathcal{T}$**:** $\text{Professor} \sqsubseteq \neg\text{Student}$
$\exists\text{teaches} \sqsubseteq \text{Professor}$
$(\textbf{funct}\ \text{teaches}^-)$

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

Violations of functionalities and of NIs can be checked separately!

Satisfiability of *DL-Lite*$_\mathcal{A}$ ontologies: Checking functs

Theorem

Let $\mathcal{T}_{\text{PI}}$ be a TBox with only PIs, and (**funct** Q) a functionality assertion. Then, for every ABox $\mathcal{A}$, $\langle \mathcal{T}_{\text{PI}} \cup \{(\textbf{funct } Q)\}, \mathcal{A} \rangle$ sat iff $\mathcal{A} \not\models \exists x, y, z.Q(x, y) \wedge Q(x, z) \wedge y \neq z$.

Proof [sketch]

$\langle \mathcal{T}_{\text{PI}} \cup \{(\textbf{funct } Q)\}, \mathcal{A} \rangle$ is satisfiable iff $\langle \mathcal{T}_{\text{PI}}, \mathcal{A} \rangle \not\models \neg(\textbf{funct } Q)$. This holds iff $\mathcal{A} \not\models \neg(\textbf{funct } Q)$ (separability property – sophisticated proof). From separability, the claim easily follows, by noticing that (**funct** Q) corresponds to the FOL sentence $\forall x, y, z.Q(x, y) \wedge Q(x, z) \rightarrow y = z$.

For a set of functionalities, we take the union of sentences of the form above (which corresponds to a boolean FOL query).

Checking satisfiability wrt functionalities therefore amounts to evaluate a FOL query over the ABox.

Example

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

The query we associate to the functionality is:

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

which evaluated over the ABox

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

returns true.

Satisfiability of *DL-Lite*$_\mathcal{A}$ ontologies: Checking NIs

Theorem

Let $\mathcal{T}_{\text{PI}}$ be a TBox with only PIs, and $A_1 \sqsubseteq \neg A_2$ a NI. For every ABox $\mathcal{A}$, $\langle \mathcal{T}_{\text{PI}} \cup \{A_1 \sqsubseteq \neg A_2\}, \mathcal{A}\rangle$ sat iff $\langle \mathcal{T}_{\text{PI}}, \mathcal{A}\rangle \not\models \exists x.A_1(x) \wedge A_2(x)$.

Proof [sketch]

$\langle \mathcal{T}_{\text{PI}} \cup \{A_1 \sqsubseteq \neg A_2\}, \mathcal{A}\rangle$ is satisfiable iff $\langle \mathcal{T}_{\text{PI}}, \mathcal{A}\rangle \not\models \neg(A_1 \sqsubseteq \neg A_2)$. The claim follows easily by noticing that $A_1 \sqsubseteq \neg A_2$ corresponds to the FOL sentence $\forall x.A_1(x) \rightarrow \neg A_2(x)$.

The property holds for all kinds of NIs ($A \sqsubseteq \exists Q$, $\exists Q_1 \sqsubseteq \exists Q_2$, etc.)

For a set of NIs, we take the union of sentences of the form above (which corresponds to a UCQ).

Checking satisfiability wrt NIs amounts to answering a UCQ over an ontology with only PIs (this can be reduced to evaluating a UCQ over the ABox – see later).

Example

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

The query we associate to the NI is:

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

whose answer over the ontology

$\exists \mathsf{teaches} \sqsubseteq \mathsf{Professor}$
$\mathsf{teaches}(\mathtt{John}, \mathtt{databases})$
$\mathsf{Student}(\mathtt{John})$
$\mathsf{teaches}(\mathtt{Mark}, \mathtt{databases})$

is true.

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

Satisfiability of a *DL-Lite*$_\mathcal{A}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is reduced to evaluation of a first order query over $\mathcal{A}$, obtained by uniting

(*a*) the FOL query associated to functionalities in $\mathcal{T}$ to

(*b*) the UCQs produced by a rewriting procedure (depending only on the PIs in $\mathcal{T}$) applied to the query associated to NIs in $\mathcal{T}$.

$\rightsquigarrow$ Ontology satisfiability in *DL-Lite*$_\mathcal{A}$ can be done using RDMBS technology.

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

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

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 FO-rewritability of query answering in *DL-Lite*$_{\mathcal{A}}$
$\rightsquigarrow$ Query answering over *DL-Lite*$_{\mathcal{A}}$ ontologies can be done using RDMBS technology.

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

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 (AtomRewrite):
if the atom unifies with the **head** of the rule (with mgu σ)
replace the atom with the **body** of the rule (to which σ is applied).

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

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

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

Query answering in *DL-Lite*$_{\mathcal{A}}$: 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 ($\sigma = \{z_1/x, z_2/y\}$)

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

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

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 `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-Lite*$_{\mathcal{A}}$: 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 ($\sigma = \{z_2/y\}$)

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

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

We now have the query

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

The PI $\mathsf{Professor} \sqsubseteq \exists \mathsf{teaches}$ (corresponding to the logic rule $\mathsf{teaches}(z_1, z_2) \leftarrow \mathsf{Professor}(z_1)$) does not apply to $\mathsf{teaches}(x, y)$ nor $\mathsf{teaches}(z, y)$, since y is a join variable.

However, we can transform the above query by unifying the atoms $\mathsf{teaches}(x, y)$, $\mathsf{teaches}(z_1, y)$. This rewriting step is called **Reduce**, and produces the query $\mathsf{q}(x) \leftarrow \mathsf{teaches}(x, y)$

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

Answering by rewriting in *DL-Lite*$_{\mathcal{A}}$: 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 $cert(q, \mathcal{O})$.

Query answering in *DL-Lite*$_{\mathcal{A}}$: 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: $\text{q}(x) \leftarrow \text{teaches}(x, y), \text{Course}(y)$
$\text{q}(x) \leftarrow \text{teaches}(x, y), \text{teaches}(z, y)$
$\text{q}(x) \leftarrow \text{teaches}(x, z)$
$\text{q}(x) \leftarrow \text{Professor}(x)$

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 $\{\texttt{John}, \texttt{Mary}\}$.

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., AC^0).

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

Query answering for CQs and UCQs is:

- PTIME in the size of TBox.
- AC^0 in the size of the ABox.
- Exponential in the size of the query (NP-complete).
  Bad? . . . not really, this is exactly as in relational DBs.

The weak side of the query rewriting approach

- main problem:
  the size of the rewriting produced by PerfectRef is exponential w.r.t. the size of the initial query
- this problem is actually unavoidable: in general, the perfect rewriting of a CQ over a *DL-Lite*$_{\mathcal{A}}$ TBox may be in the worst case exponential, if the rewritten query is a UCQ
- the same holds even if we go beyond UCQ and allow for arbitrary FO queries [Kikot et al., 2011;2012]
- using additional predicates/constants, it is possible to produce polynomial perfect rewritings of CQs in nonrecursive Datalog [Gottlob et al., 2012]
- nevertheless, several optimization of PerfectRef have been proposed, to improve both the execution time of query rewriting and the size of the rewritten query

Requiem [Perez Urbina et al., 2006]

- through the Reduce step, PerfectRef solves incompleteness of previous approaches
- however, the Reduce step is applied in a very naive, exhaustive way
- in the vast majority of cases, this is not needed
- Requiem is an algorithm that improves this part of the computation
- in addition, it provides a native treatment of qualified existential restrictions
- the algorithm has then extended to more expressive DLs (up to $\mathcal{ELHIO}$)

Requiem [Perez Urbina et al., 2006]

Main optimizations for *DL-Lite*$_{\mathcal{A}}$:

- single rewriting step: avoids unification steps separated from resolution/rewriting step (as in Reduce)
  - to do so, it first encodes the TBox into clauses with functional terms
  - then, it uses a specialized resolution rule for such clauses
  - this allows for avoiding useless unification (Reduce) steps
  - this is more effective mainly in the presence of qualified existential restrictions (beyond *DL-Lite*$_{\mathcal{A}}$)
- also performs elimination of redundant CQs (through a CQ containment check)

Presto [R. et al., 2010]

Idea 1: divide computation of rewriting in two phases:

phase 1: elimination of existential join variables
purpose: make the Reduce step of PerfectRef totally useless

phase 2: "unfolding"
corresponds to the application of AtomRewrite to the query produced by phase 1

Idea 2: use nonrecursive Datalog instead of UCQ, at least for internal representation of the query

Elimination of join variables in Presto: Example

TBox: $\{D \sqsubseteq \exists R,\ D \sqsubseteq \exists S,\ R \sqsubseteq S\}$
query: $q(x) \leftarrow C(x), R(x,z), S(x,z)$

Question: can join variable z be eliminated? i.e., does z disappear in some rewriting of this query?

The algorithm looks for (a specialized notion of) **most general subsumees (MGS)** of the concept expressions $\exists R$, $\exists S$ in the TBox

In our example, D is an MGS of $\exists R$, $\exists S$ (notice: axiom $R \sqsubseteq S$ is actually necessary in order to conclude this)

The algorithm rewrites all the atoms where z occurs using the MGS (and unification), producing a new query $q(x) \leftarrow C(x), D(x)$

This corresponds to a sequence of AtomRewrite and Reduce steps

Rapid [Chortaras et al., 2011]

- similar to Presto
- divides computation in two steps:
  1. shrinking phase
     same purpose as Presto: eliminate existential join variables
  2. unfolding phase
     again, corresponds to application of AtomRewrite
- additional optimization: generation of core rewritings
  - no subsumed CQs in the final UCQ
  - no redundant atoms in CQs

Incremental query rewriting [Venetis et al., 2012]

- exploits the property that the rewritings of a query atom are (mostly) independent on the other atoms of the query
- e.g., if Q is a (already computed) perfect rewriting of query $q \leftarrow body$, the rewriting of query $q \leftarrow body, \alpha$ can be obtained by rewriting atom α only and then combining such a rewriting with Q
- it can also compute query rewritings from scratch, by rewriting single query atoms and then combining the rewritings
- the performance is competitive with the previous algorithms even when computing rewritings from scratch

Other FO-rewritable ontology languages

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

Within DL:
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 [Calvanese et al., 2006;Artale et al., 2009]

Outside DL:
The following languages have been considered:

- n-ary extensions of DL (*DLR-Lite*)
- constraint languages for relational schemas:
  - tuple-generating dependencies and equality-generating dependencies (i.e., embedded database dependencies)
  - a.k.a. Datalog+/−, existential rules

Tuple-generating dependencies (TGDs)

- TGD = sentence of the form

$$\forall x_1, \ldots, x_k \, (\alpha_1 \wedge \ldots \wedge \alpha_n \rightarrow \exists y_1, \ldots, y_h \, (\beta_1 \wedge \ldots \wedge \beta_m))$$

  where

  - every α_i is an atom whose terms are constants and variables from $\{x_1, \ldots, x_k\}$
  - every β_i is an atom whose terms are constants and variables from $\{x_1, \ldots, x_k \; y_1, \ldots, y_h\}$

- TGDs generalize Horn-DLs
- in general, reasoning under TGDs is undecidable
- recent, notable amount of research on identifying decidable/tractable/FO-rewritable subclasses of TGDs

FO-rewritable classes of TGDs

- linear TGDs [Calì et al., 2003; Calì et al., 2009]
- multi-linear TGDs [Calì et al., 2009]
- sticky TGDs, sticky-join TGDs [Calì et al., 2010]
- domain-restricted TGDs [Baget et al., 2011]
- AGRD TGDs [Baget et al., 2011]
- weakly recursive TGDs [Civili et al., 2012]

Query rewriting techniques outside DLs

- linear TGDs [Calì et al., 2003]
- *DLR-Lite* [Calvanese et al., 2007]
- sticky TGDs, sticky-join TGDs [Gottlob et al., 2011]
- more general algorithm for TGDs [König et al., 2012]
- ...

FO-rewritability and the Unique Name Assumption

Remark: like *DL-Lite*$_{\mathcal{A}}$, all these languages adopt the Unique Name Assumption

In the absence of UNA, **FO-rewritability of CQs is lost as soon as the ontology language allows for deriving equalities** between constants (individuals)

E.g., role functionality axioms in *DL-Lite*$_{\mathcal{A}}$ may impose equalities between constants (functionality of role R and the presence of $R(a, b)$ and $R(a, c)$ in the ABox imply $b = c$)

In these cases, it would be necessary to encode the equality predicate in the perfect rewriting of queries, which is not possible using FO queries (since equality is a transitive property).

Outline

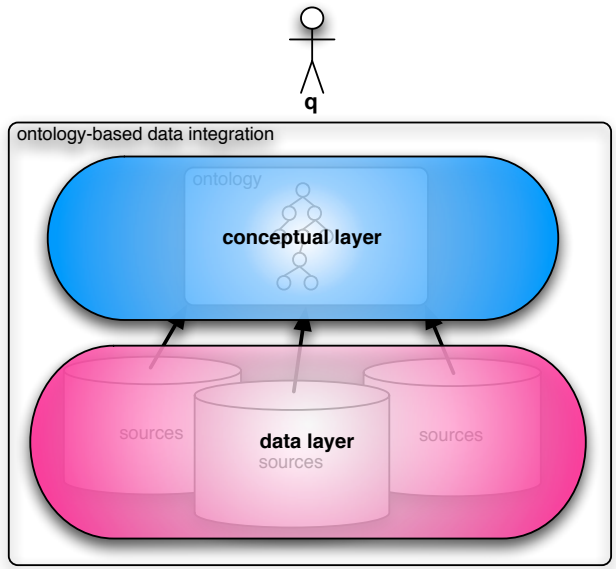
Data integration

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

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

Ontology-based data access: conceptual & data layer

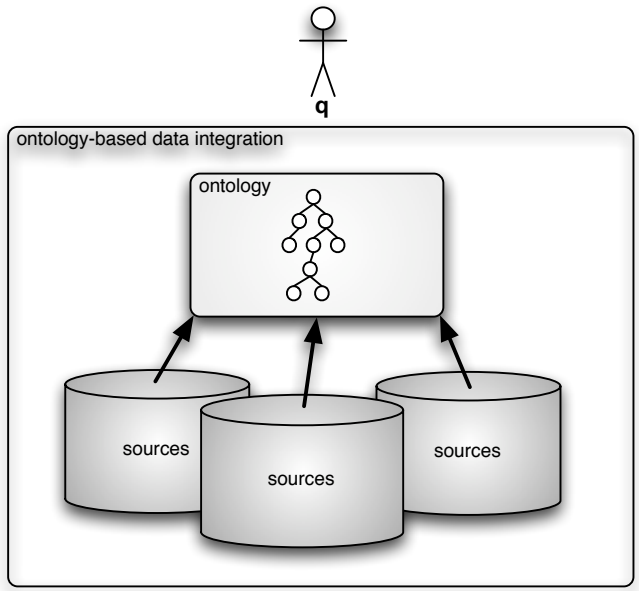
Ontology-based data access is based on the idea of decoupling information access from data storage.



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 access: architecture

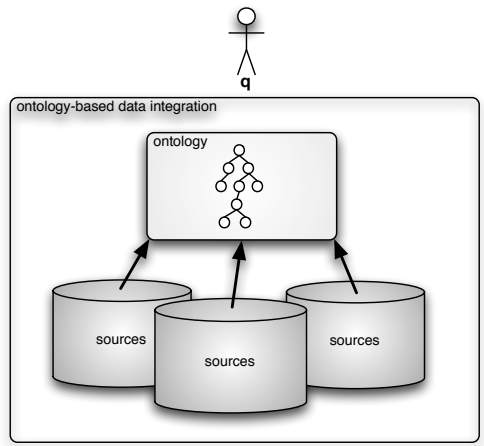


Based on three main components:

- **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 access: the conceptual layer

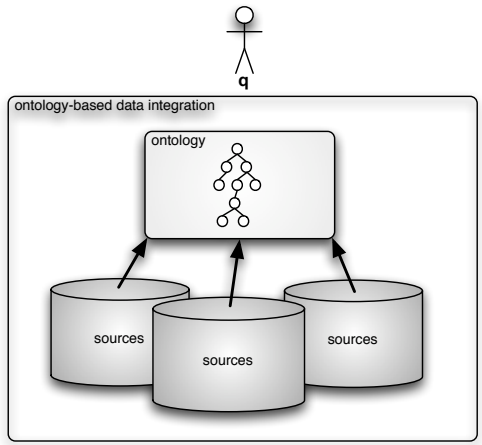
The ontology is used as the conceptual layer, to give clients a unified conceptual global view of the data.



Note: in standard information systems, UML Class Diagram or ER is used at **design time**, ...
... here we use ontologies at **runtime**!

Ontology-based data access: the sources

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

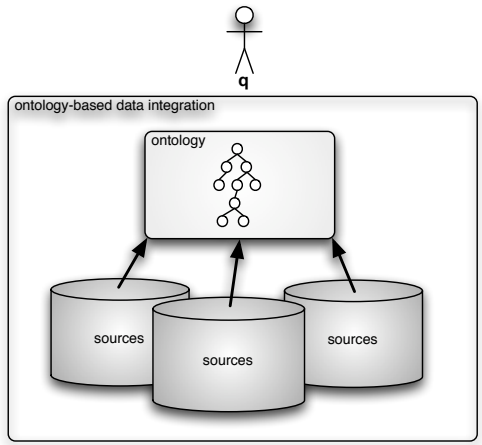


By now we have industrial solutions for:

- Distributed database systems & Distributed query optimization
- Tools for source wrapping
- Systems for database federation

Ontology-based data access: the sources

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



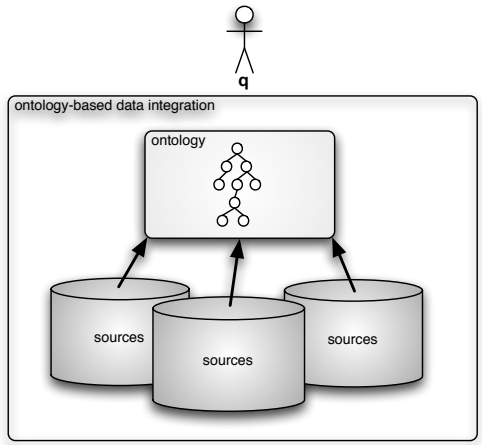
Based on these industrial solutions we can:

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 access: mappings

Mappings semantically link data at the sources with the ontology.

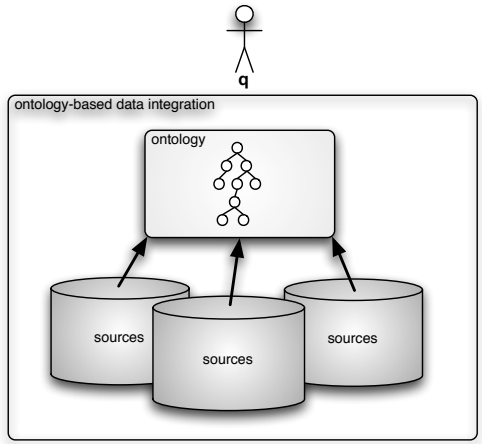


Scientific literature on data integration in databases has shown that ...

... 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 access: mappings

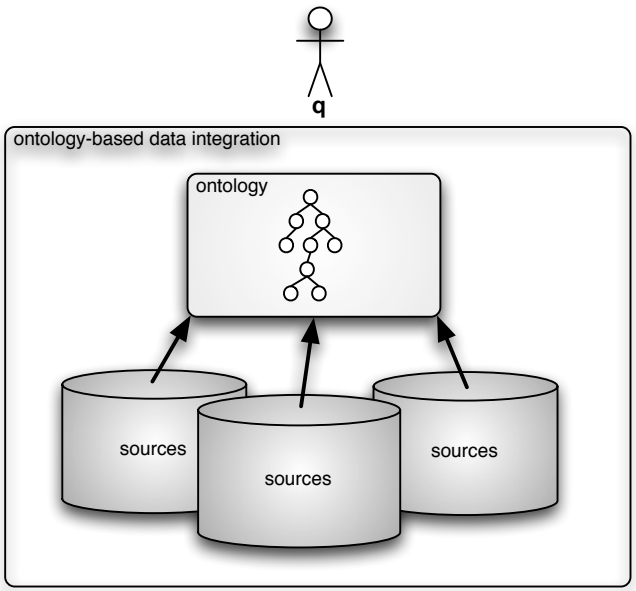


Several general forms of mappings based on queries have been considered:

- 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*

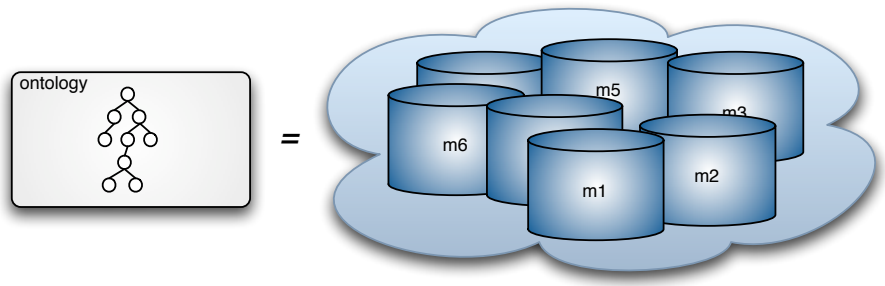
Ontology-based data access: incomplete information

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**!



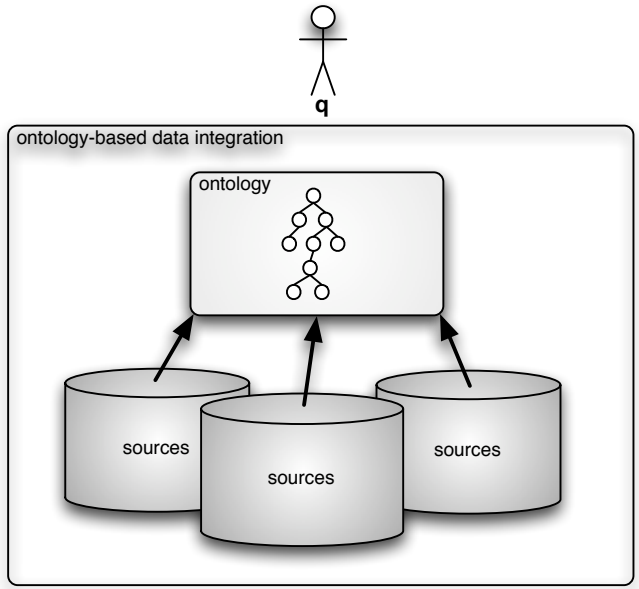
- Query answering amounts to compute **certain answers**, given the global view, the mapping and the data at the sources ...
- ... but query answering may be costly in ontologies (even without mapping and sources).

Query answering in OBDA

We have to face the difficulties of both DB and KB assumptions:

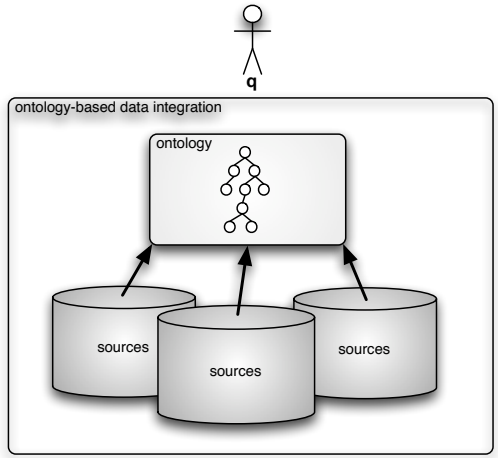
- The actual **data** is stored in external information sources (i.e., databases), and thus its size is typically **very large**.
- The ontology introduces **incompleteness** of information, and we have to do logical inference, rather than query evaluation.
- We want to take into account at **runtime** 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.

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



- We require the data sources to be **wrapped** and presented as relational sources. $\rightsquigarrow$ *"standard technology"*
- We make use of a **data federation tool** to present the yet to be (semantically) integrated sources as a single relational database. $\rightsquigarrow$ *"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**. $\rightsquigarrow$ *"new technology"*

Ontology-based data access: 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. $\rightsquigarrow$ define in fact a **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

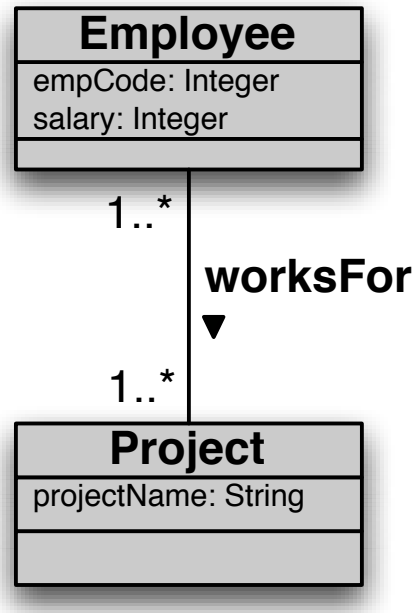
The impedance mismatch problem

- 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.

$\rightsquigarrow$ *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:

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

...

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.

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

Formalization of OBDA

An **OBDA specification** 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 schema representing the sources, possibly with integrity constraints;
- $\mathcal{M}$ is a set of (GAV-style) **mapping assertions**, each one of the form*

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

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 $\vec{x}$.

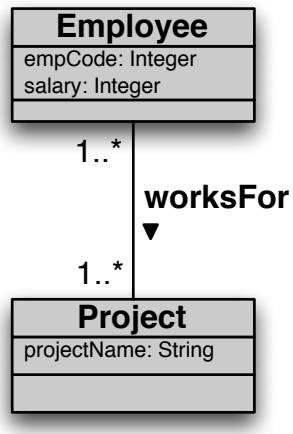
Formalization of OBDA

An **OBDA system** is a pair $\langle \mathcal{O}_m, D \rangle$ where

- $\mathcal{O}_m$ is an OBDA specification $\mathcal{O}_m = \langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$
- D is a legal instance of schema $\mathcal{S}$
  (i.e., D satisfies the integrity constraints in $\mathcal{S}$)

OBDA specification – 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` $\rightsquigarrow$ 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` $\rightsquigarrow$ Employee(**pers**(*SSN*)), salary(**pers**(*SSN*), *Salary*)

Semantics

Def.: **Semantics of mappings**

We say that $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ satisfies $\Phi(\vec{x}) \rightsquigarrow \Psi(f(\vec{x}), \vec{x})$ 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 OBDA**

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

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

Semantics

Def.: The **certain answers** to $q(\vec{x})$ over $\langle \mathcal{O}_m, D \rangle \ldots$

. . . denoted $cert(q, \mathcal{O}_m, D)$, are the **tuples $\vec{t}$ of object terms and constants from D** such that $\vec{t} \in q^{\mathcal{I}}$, for every model $\mathcal{I}$ of $\langle \mathcal{O}_m, D \rangle$.

Outline

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

We do not consider inconsistent OBDA systems (it is possible to check consistency of OBDA system)

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

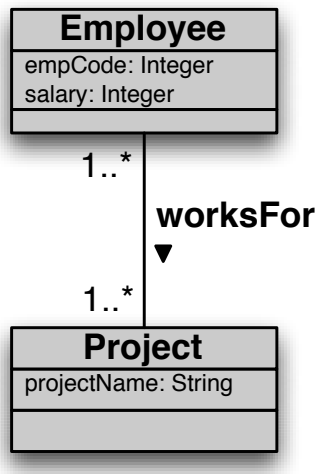
1. Using $\mathcal{T}$, **reformulate** CQ q as a union $r_{q,\mathcal{T}}$ of CQs.
2. Using $\mathcal{M}$, **unfold** $r_{q,\mathcal{T}}$ to obtain a union $unfold(r_{q,\mathcal{T}})$ of CQs.
3. **Evaluate** $unfold(r_{q,\mathcal{T}})$ directly over D using RDBMS technology.

Correctness of this algorithm shows FOL-reducibility of query answering.

$\rightsquigarrow$ Query answering can again be done using **RDBMS technology**.

Example – query rewriting

TBox $\mathcal{T}$ (UML)



TBox $\mathcal{T}$ ($DL\text{-}Lite_{\mathcal{A}}$)

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

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

the perfect rewriting is

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

Example – splitting the mapping

To compute $\mathit{unfold}(r_{q,\mathcal{T}})$, 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 D1` $\rightsquigarrow$ Employee(**pers**(*SSN*))

$M_{1,2}$: `SELECT SSN, PrName FROM D1` $\rightsquigarrow$ Project(**proj**(*PrName*))

$M_{1,3}$: `SELECT SSN, PrName FROM D1` $\rightsquigarrow$ projectName(**proj**(*PrName*), *PrName*)

$M_{1,4}$: `SELECT SSN, PrName FROM D1` $\rightsquigarrow$ workFor(**pers**(*SSN*), **proj**(*PrName*))

$M_{2,1}$:
```
SELECT SSN, Salary
FROM D2, D3
WHERE D2.Code = D3.Code
```
$\rightsquigarrow$ Employee(**pers**(*SSN*))

$M_{2,2}$:
```
SELECT SSN, Salary
FROM D2, D3
WHERE D2.Code = D3.Code
```
$\rightsquigarrow$ salary(**pers**(*SSN*), *Salary*)

Example – unfolding

Then, we unify each atom of the query

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

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

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

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*$_{\mathcal{A}}$ 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. $\mathbf{AC}^0$ in the size of the **database** $\mathcal{S}$, in fact FO-rewritable.

Can we move to LAV or GLAV mappings?
No, if we want to have DL-Lite$_{\mathcal{A}}$ *TBoxes and stay in* AC^0*!*

Alternatively, we can have LAV or GLAV mappings, but we have to **renounce to use role functionalities** *in the TBox and* **limit the form of the queries in the mapping** *(essentially CQs over both the sources and the ontology), if we want to stay in* AC^0*.*

Current OBDA systems

- Mastro [De Giacomo et al., 2012] implements the above query answering technique
- Ontop [Rodriguez-Muro et al, 2013] implements a different technique

  main difference: saturation of mapping to reduce query rewriting over the TBox
- Optique (under development) (EU project)

Remark: we are only considering systems able to deal with the above rich mapping language, without materialization of the ABox

The weak side of query rewriting in OBDA

- as discussed above, the rewriting of a query q w.r.t. TBox may be exponential w.r.t. the size (number of atoms) of q
- in addition, the perfect rewriting of a CQ in OBDA has a second exponential blowup which is due to the mapping
- example: consider an empty TBox and a mapping of the form

$$T_1(x, y) \rightsquigarrow R(x, y)$$
$$T_2(x, y) \rightsquigarrow R(x, y)$$

then the perfect rewriting of query
$q(x_1) \leftarrow R(x_1, x_2), \ldots, R(x_n, x_1)$ consists of the UCQ

$$\bigcup_{j_1,\ldots,j_n \in \{1,2\}} q(x_1) \leftarrow T_{j_1}(x_1, x_2), \ldots, T_{j_n}(x_n, x_1)$$

containing 2^n CQs.

The weak side of query rewriting in OBDA

- in practice, the bottleneck due to the mapping may be worse than the one caused by the TBox
- e.g., if every predicate is associated with 10 mappings assertions, then the mapping query rewriting of a query with 10 atoms produces a UCQ with 10^{10} CQs
- one possible way out is to merge mappings, generating only one mapping for every ontology predicate
- e.g., in the previous example, the mapping would be transformed as follows: $T_1(x, y)$ `UNION` $T_2(x, y) \rightsquigarrow R(x, y)$
- this complicates the structure of the final SQL expression (additional nesting level of subqueries)
- DBMSs do not seem able to effectively deal with such more complex query structures [Calvanese et al., 2012; Di Pinto et al., 2013]

The weak side of query rewriting in OBDA

optimizations to mitigate this problem have been proposed recently, e.g.:

- use the form of the mapping and the database integrity constraints to prune the rewritten query and/or reduce the number of queries generated by the unfolding [Di Pinto et al, 2013]
- perform a merge (factorization) operation on mappings over the same ontology predicate, when the structure of the SQL queries involved is sufficiently simple and follows a common pattern [Rodriguez-Muro et al., 2013]

Outline

Some open problems in OBQA

- further optimization of OBQA query rewriting in *DL-Lite*$_{\mathcal{A}}$ and FO-rewritable languages
- query languages beyond UCQ:
  - FO-queries
    - under classical semantics, this in general implies that FO-rewritability (or even decidability) is lost
    - alternative semantics have been proposed, e.g., epistemic semantics
  - other classes of queries (SPARQL queries, RPQ and extensions)

Some open problems in OBQA

- FO-rewritability of languages is a nice theoretical tool... but it would be important to go beyond *DL-Lite*$_{\mathcal{A}}$ and FO-rewritable languages while keeping query answering "practical"
- a lot of current work on this – some directions:
  - studying FO-rewritability of **single TBoxes**
  - ... and of **single queries** too
  - **approximating** more expressive **TBoxes** to FO-rewritable languages
  - **approximating query answers** over more expressive TBoxes
  - move to **Datalog-rewritable languages** and Datalog data management systems
  - ...

Some open problems in OBDA

- current query rewriting algorithms for OBDA strictly sepatate TBox processing and mapping processing
  - further optimizations might be obtained by a more holistic approach that considers the whole OBDA specification
- efficiency of OBDA query answering in OBDA heavily depends on the underlying data management system and the data structures
- however, current techniques are essentially independent of such aspects
  - further optimizations might be obtained by taking into account these characteristics of the data layer

Conclusions

- a lot of research on OBQA for *DL-Lite*
  - several practical techniques
  - "good" optimizations
- query answering and rewriting in OBDA is less developed
  - more optimizations needed
- theoretical and practical limits of "FO-rewritability approach" still not known
- query rewriting in OBQA and (especially) in OBDA still very challenging
- a lot of potential applications of OBDA in the real world
- OPTIQUE European Project, www.optique-project.eu

Acknowledgments

Many thanks to Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and the OPTIQUE project

Thank you very much for your attention!

P.S.: Thanks to Shqiponja Ahmetaj for pointing out an error in one of the examples