End-User Access to Big Data Using Ontologies
Part 3: Ontology Based Data Access

Diego Calvanese

Free University of Bozen-Bolzano

unibz
Fakultät für Informatik
Facoltà di Scienze e Tecnologie informatiche
Faculty of Computer Science

International Winter School on Big Data
Tarragona, Spain, January 26–30, 2015
Part 3

Ontology Based Data Access
Outline of Part 3

1. The *DL-Lite* family of tractable Description Logics
   - Basic features of *DL-Lite*
   - Syntax and semantics of *DL-Lite*
   - Members of the *DL-Lite* family
   - Properties of *DL-Lite*

2. Reasoning in *DL-Lite*
   - Query answering over satisfiable ontologies
   - Ontology satisfiability
   - Complexity of reasoning in *DL-Lite*

3. Linking ontologies to relational data
   - The impedance mismatch problem
   - Ontology-Based Data Access systems
   - Query answering in Ontology-Based Data Access systems
   - Optimizing OBDA in Ontop

4. Conclusions and further work
Outline of Part 3

1. The *DL-Lite* family of tractable Description Logics
   - Basic features of *DL-Lite*
   - Syntax and semantics of *DL-Lite*
   - Members of the *DL-Lite* family
   - Properties of *DL-Lite*

2. Reasoning in *DL-Lite*

3. Linking ontologies to relational data

4. Conclusions and further work
Outline of Part 3

1. The *DL-Lite* family of tractable Description Logics
   - Basic features of *DL-Lite*
     - Syntax and semantics of *DL-Lite*
     - Members of the *DL-Lite* family
     - Properties of *DL-Lite*

2. Reasoning in *DL-Lite*

3. Linking ontologies to relational data

4. Conclusions and further work
The *DL-Lite* family

- A family of DLs optimized according to the tradeoff between expressive power and *complexity* of query answering, with emphasis on *data*.

- Carefully designed to have nice computational properties for answering UCQs (i.e., computing certain answers):
  - The same data complexity as relational databases.
  - In fact, query answering can be delegated to a relational DB engine.
  - The DLs of the *DL-Lite* family are essentially the maximally expressive ontology languages enjoying these nice computational properties.

- Captures conceptual modeling formalism.

The *DL-Lite* family provides new foundations for Ontology-Based Data Access.
Basic features of $DL$-$Lite_A$

$DL$-$Lite_A$ is an expressive member of the $DL$-$Lite$ family.

- Takes into account the distinction between objects and values:
  - Objects are elements of an abstract interpretation domain.
  - Values are elements of concrete data types, such as integers, strings, etc.
  - Values are connected to objects through attributes (rather than roles).

- Captures most of UML class diagrams and Extended ER diagrams.

- Enjoys nice computational properties, both w.r.t. the traditional reasoning tasks, and w.r.t. query answering (see later).
The OWL 2 QL Profile

OWL 2 defines three profiles: OWL 2 QL, OWL 2 EL, OWL 2 RL [Motik et al., 2009]

- Each profile corresponds to a syntactic fragment (i.e., a sub-language) of OWL 2 DL that is targeted towards a specific use.
- The restrictions in each profile guarantee better computational properties than those of OWL 2 DL.

The **OWL 2 QL** profile is derived from the DLs of the *DL-Lite* family:

- “[It] includes most of the main features of conceptual models such as UML class diagrams and ER diagrams.”
- “[It] is aimed at applications that use very large volumes of instance data, and where query answering is the most important reasoning task. In OWL 2 QL, conjunctive query answering can be implemented using conventional relational database systems.”
Outline of Part 3

1. The DL-Lite family of tractable Description Logics
   - Basic features of DL-Lite
   - Syntax and semantics of DL-Lite
   - Members of the DL-Lite family
   - Properties of DL-Lite

2. Reasoning in DL-Lite

3. Linking ontologies to relational data

4. Conclusions and further work
Syntax of the $DL-Lite_{A}$ description language

- **Role expressions** (*object properties* in OWL 2 QL):
  - atomic role: $P$
  - basic role: $Q ::= P \mid P$
  - arbitrary role: $R ::= Q \mid \neg Q$ (to express disjointness)

- **Concept expressions** (*classes* in OWL 2 QL):
  - atomic concept: $A$
  - basic concept: $B ::= A \mid \exists Q \mid \delta(U)$
  - arbitrary concept: $C ::= \top_C \mid B \mid \neg B$ (to express disjointness)

- **Attribute expressions** (*data properties* in OWL 2 QL):
  - atomic attribute: $U$
  - arbitrary attribute: $V ::= U \mid \neg U$ (to express disjointness)

- **Value-domain expressions** (*datatypes* in OWL 2 QL):
  - attribute range: $\rho(U)$
  - RDF datatypes: $T_i$
  - top domain: $\top_D$
### Semantics of $DL-Lite_A$ – Objects vs. values

<table>
<thead>
<tr>
<th>Interpretation domain $\Delta^T$</th>
<th>Objects</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain of objects $\Delta^T_O$</td>
<td>Domain of values $\Delta^T_V$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Alphabet $\Gamma$ of constants</th>
<th>Objects</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object constants $\Gamma_O$</td>
<td>Value constants $\Gamma_V$</td>
<td></td>
</tr>
<tr>
<td>$c^T \in \Delta^T_O$</td>
<td>$d^T = \text{val}(d)$ given a priori</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Unary predicates</th>
<th>Objects</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concept $C$</td>
<td>RDF datatype $T_i$</td>
<td></td>
</tr>
<tr>
<td>$C^T \subseteq \Delta^T_O$</td>
<td>$T_i^T \subseteq \Delta^T_V$ given a priori</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Binary predicates</th>
<th>Objects</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role $R$</td>
<td>Attribute $V$</td>
<td></td>
</tr>
<tr>
<td>$R^T \subseteq \Delta^T_O \times \Delta^T_O$</td>
<td>$V^T \subseteq \Delta^T_O \times \Delta^T_V$</td>
<td></td>
</tr>
</tbody>
</table>
### Semantics of the $DL-Lite_A$ Constructs

<table>
<thead>
<tr>
<th>Construct</th>
<th>Syntax</th>
<th>Example</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>atomic role</td>
<td>$P$</td>
<td>child</td>
<td>$P^I \subseteq \Delta^I_O \times \Delta^I_O$</td>
</tr>
<tr>
<td>inverse role</td>
<td>$P^-$</td>
<td>child$^-$</td>
<td>${(o, o') \mid (o', o) \in P^I}$</td>
</tr>
<tr>
<td>role negation</td>
<td>$\neg Q$</td>
<td>$\neg$manages</td>
<td>$(\Delta^I_O \times \Delta^I_O) \setminus Q^I$</td>
</tr>
<tr>
<td>atomic concept</td>
<td>$A$</td>
<td>Doctor</td>
<td>$A^I \subseteq \Delta^I_O$</td>
</tr>
<tr>
<td>existential restriction</td>
<td>$\exists Q$</td>
<td>$\exists$child$^-$</td>
<td>${o \mid \exists o'. (o, o') \in Q^I}$</td>
</tr>
<tr>
<td>concept negation</td>
<td>$\neg B$</td>
<td>$\neg$child</td>
<td>$\Delta^I \setminus B^I$</td>
</tr>
<tr>
<td>attribute domain</td>
<td>$\delta(U)$</td>
<td>$\delta$(salary)</td>
<td>${o \mid \exists v. (o, v) \in U^I}$</td>
</tr>
<tr>
<td>top concept</td>
<td>$\top_C$</td>
<td></td>
<td>$\top^I_C = \Delta^I_O$</td>
</tr>
<tr>
<td>atomic attribute</td>
<td>$U$</td>
<td>salary</td>
<td>$U^I \subseteq \Delta^I_O \times \Delta^I_V$</td>
</tr>
<tr>
<td>attribute negation</td>
<td>$\neg U$</td>
<td>$\neg$salary</td>
<td>$(\Delta^I_O \times \Delta^I_V) \setminus U^I$</td>
</tr>
<tr>
<td>top domain</td>
<td>$\top_D$</td>
<td></td>
<td>$\top^I_D = \Delta^I_V$</td>
</tr>
<tr>
<td>datatype</td>
<td>$T_i$</td>
<td>xsd:int</td>
<td>$T_{i}^I \subseteq \Delta^I_V$ (predefined)</td>
</tr>
<tr>
<td>attribute range</td>
<td>$\rho(U)$</td>
<td>$\rho$(salary)</td>
<td>${v \mid \exists o. (o, v) \in U^I}$</td>
</tr>
<tr>
<td>object constant</td>
<td>$c$</td>
<td>john</td>
<td>$c^I \in \Delta^I_O$</td>
</tr>
<tr>
<td>value constant</td>
<td>$d$</td>
<td>’john’</td>
<td>$\text{val}(d) \in \Delta^I_V$ (predefined)</td>
</tr>
</tbody>
</table>
**DL-Lite** assertions

**TBox** assertions can have the following forms:

- **Inclusion assertions** (also called positive inclusions):
  
  \[ B_1 \sqsubseteq B_2 \]  
  concept inclusion
  
  \[ Q_1 \sqsubseteq Q_2 \]  
  role inclusion
  
  \[ \rho(U) \sqsubseteq T_i \]  
  value-domain inclusion
  
  \[ U_1 \sqsubseteq U_2 \]  
  attribute inclusion

- **Disjointness assertions** (also called negative inclusions):
  
  \[ B_1 \sqsubseteq \neg B_2 \]  
  concept disjointness
  
  \[ Q_1 \sqsubseteq \neg Q_2 \]  
  role disjointness
  
  \[ U_1 \sqsubseteq \neg U_2 \]  
  attribute disjointness

- **Functionality assertions**:
  
  \[ \text{funct } Q \]  
  role functionality
  
  \[ \text{funct } U \]  
  attribute functionality

- **Identification assertions**:
  
  \[ \text{id } B \ I_1, \ldots, I_n \]  
  where each \( I_j \) is a role, an inverse role, or an attribute

**ABox** assertions:

\[ A(c), \ P(c, c'), \ U(c, d), \]  
where \( c, c' \) are object constants and \( d \) is a value constant
### Semantics of the \( DL-Lite_A \) assertions

<table>
<thead>
<tr>
<th>Assertion</th>
<th>Syntax</th>
<th>Example</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>conc. incl.</td>
<td>( B_1 \sqsubseteq B_2 )</td>
<td>Father ( \sqsubseteq \exists \text{child} )</td>
<td>( B_1^T \sqsubseteq B_2^T )</td>
</tr>
<tr>
<td>role incl.</td>
<td>( Q_1 \sqsubseteq Q_2 )</td>
<td>father ( \sqsubseteq \text{anc} )</td>
<td>( Q_1^T \sqsubseteq Q_2^T )</td>
</tr>
<tr>
<td>v.dom. incl.</td>
<td>( \rho(U) \sqsubseteq T_i )</td>
<td>( \rho(\text{age}) \sqsubseteq \text{xsd:int} )</td>
<td>( \rho(U)^T \sqsubseteq T_i^T )</td>
</tr>
<tr>
<td>attr. incl.</td>
<td>( U_1 \sqsubseteq U_2 )</td>
<td>offPhone ( \sqsubseteq \text{phone} )</td>
<td>( U_1^T \sqsubseteq U_2^T )</td>
</tr>
<tr>
<td>conc. disj.</td>
<td>( B_1 \sqsubseteq \neg B_2 )</td>
<td>Person ( \sqsubseteq \neg \text{Course} )</td>
<td>( B_1^T \sqsubseteq (\neg B_2)^T )</td>
</tr>
<tr>
<td>role disj.</td>
<td>( Q_1 \sqsubseteq \neg Q_2 )</td>
<td>sibling ( \sqsubseteq \neg \text{cousin} )</td>
<td>( Q_1^T \sqsubseteq (\neg Q_2)^T )</td>
</tr>
<tr>
<td>attr. disj.</td>
<td>( U_1 \sqsubseteq \neg U_2 )</td>
<td>offPhn ( \sqsubseteq \neg \text{homePhn} )</td>
<td>( U_1^T \sqsubseteq (\neg U_2)^T )</td>
</tr>
<tr>
<td>role funct.</td>
<td>(\text{funct} ( Q ))</td>
<td>(\text{funct} ( \text{father} ))</td>
<td>( \forall o, o_1, o_2.(o, o_1) \in Q^T \land (o, o_2) \in Q^T \rightarrow o_1 = o_2 )</td>
</tr>
<tr>
<td>att. funct.</td>
<td>(\text{funct} ( U ))</td>
<td>(\text{funct} ( \text{ssn} ))</td>
<td>( \forall o, v, v'.(o, v) \in U^T \land (o, v') \in U^T \rightarrow v = v' )</td>
</tr>
<tr>
<td>id const.</td>
<td>(\text{id} ( B \ I_1, \ldots, I_n ))</td>
<td>(\text{id} ( \text{Person name, dob} ))</td>
<td>( I_1, \ldots, I_n ) identify instances of ( B )</td>
</tr>
<tr>
<td>mem. asser.</td>
<td>( A(c) )</td>
<td>Father(( \text{bob} ))</td>
<td>( c^T \in A^T )</td>
</tr>
<tr>
<td>mem. asser.</td>
<td>( P(c_1, c_2) )</td>
<td>child(( \text{bob, ann} ))</td>
<td>( (c_1^T, c_2^T) \in P^T )</td>
</tr>
<tr>
<td>mem. asser.</td>
<td>( U(c, d) )</td>
<td>phone(( \text{bob, ’2345’} ))</td>
<td>( (c^T, \text{val}(d)) \in U^T ) \text{unibz}</td>
</tr>
</tbody>
</table>
DL-Lite_\textsubscript{A} – Example

Note: DL-Lite_\textsubscript{A} cannot capture completeness of a hierarchy. This would require disjunction (i.e., OR).
<table>
<thead>
<tr>
<th>Outline of Part 3</th>
<th>The DL-Lite family of tractable Description Logics</th>
<th>Reasoning in DL-Lite</th>
<th>Linking ontologies to relational data</th>
<th>Conclusions and further work</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>1.</strong> The <em>DL-Lite</em> family of tractable Description Logics</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Basic features of <em>DL-Lite</em></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Syntax and semantics of <em>DL-Lite</em></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Members of the <em>DL-Lite</em> family</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Properties of <em>DL-Lite</em></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>2.</strong> Reasoning in <em>DL-Lite</em></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>3.</strong> Linking ontologies to relational data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>4.</strong> Conclusions and further work</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Restriction on TBox assertions in $DL-Lite_A$ ontologies

We will see that, to ensure the good computational properties that we aim at, we have to impose a restriction on the use of functionality and role/attribute inclusions.

Restriction on $DL-Lite_A$ TBoxes

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

Formally:

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

2. If $(\text{funct } U)$ or $(\text{id } B \ldots, U, \ldots)$ is in $\mathcal{T}$, then $U' \sqsubseteq U$ is not in $\mathcal{T}$.
We consider also two sub-languages of $DL$-$Lite_A$ (that trivially obey the previous restriction):

- $DL$-$Lite_F$: Allows for functionality assertions, but does not allow for role inclusion assertions.
- $DL$-$Lite_R$: Allows for role inclusion assertions, but does not allow for functionality assertions.

This is the DL that corresponds to the OWL 2 QL profile.

**Note:** We simply use $DL$-$Lite$ to refer to any of the logics of the $DL$-$Lite$ family.
Outline of Part 3

1. The *DL-Lite* family of tractable Description Logics
   - Basic features of *DL-Lite*
   - Syntax and semantics of *DL-Lite*
   - Members of the *DL-Lite* family
   - Properties of *DL-Lite*

2. Reasoning in *DL-Lite*

3. Linking ontologies to relational data

4. Conclusions and further work
## Capturing basic ontology constructs in $DL$-$Lite_A$

<table>
<thead>
<tr>
<th>Property</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISA between classes</td>
<td>$A_1 \sqsubseteq A_2$</td>
</tr>
<tr>
<td>Disjointness between classes</td>
<td>$A_1 \sqsubseteq \neg A_2$</td>
</tr>
<tr>
<td>Mandatory participation to relations</td>
<td>$A_1 \sqsubseteq \exists P \quad A_2 \sqsubseteq \exists P^-$</td>
</tr>
<tr>
<td>Domain and range of relations</td>
<td>$\exists P \sqsubseteq A_1 \quad \exists P^- \sqsubseteq A_2$</td>
</tr>
<tr>
<td>Functionality of relations</td>
<td>($\text{funct } P$) ($\text{funct } P^-$)</td>
</tr>
<tr>
<td>ISA between relations</td>
<td>$Q_1 \sqsubseteq Q_2$</td>
</tr>
<tr>
<td>Disjointness between relations</td>
<td>$Q_1 \sqsubseteq \neg Q_2$</td>
</tr>
<tr>
<td>Domain and range of attributes</td>
<td>$\delta(U) \sqsubseteq A \quad \rho(U) \sqsubseteq T_i$</td>
</tr>
<tr>
<td>Mandatory and functional attributes</td>
<td>$A \sqsubseteq \delta(U)$ ($\text{funct } U$)</td>
</tr>
<tr>
<td>Identification constraints</td>
<td>($\text{id } A, P, \ldots, P'^-, \ldots, U, \ldots$)</td>
</tr>
</tbody>
</table>
Properties of *DL-Lite*

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

  **Example:** \( A \sqsubseteq \exists P, \exists P^- \sqsubseteq A \)

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

  \[
  B \sqsubseteq C_1 \cap C_2 \quad \text{is equivalent to} \quad B \sqsubseteq C_1 \quad B \sqsubseteq C_2
  \]

- A domain assertion on role \( P \) has the form: \( \exists P \sqsubseteq A_1 \)

- A range assertion on role \( P \) has the form: \( \exists P^- \sqsubseteq A_2 \)
Finite model property

*DL-Lite*₆ (and *DL-Lite₄) does **not** enjoy the **finite model property**.

**Example**

**TBox** T:  
Nat ⊑ ∃succ  
Zero ⊑ Nat ⊓ ¬∃succ⁻  
(\textbf{funct succ⁻})

**ABox** A:  
Zero(0)  

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

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

Notice that instead *DL-Lite₆* does enjoy the finite model property.
### Outline of Part 3

1. **The *DL-Lite* family of tractable Description Logics**

2. **Reasoning in *DL-Lite***
   - Query answering over satisfiable ontologies
   - Ontology satisfiability
   - Complexity of reasoning in *DL-Lite*

3. **Linking ontologies to relational data**

4. **Conclusions and further work**
In the following, we ignore the distinction between objects and values, since it is not relevant for reasoning. Hence we do not use value domains and attributes.

Notation:
- When the distinction between $DL$-$Lite_R$, $DL$-$Lite_F$, or $DL$-$Lite_A$ is not important, we use just $DL$-$Lite$.
- $Q$ denotes a basic role, i.e., $Q \rightarrow P \mid P^\perp$.
- $R$ denotes a general role, i.e., $R \rightarrow Q \mid \neg Q$.
- $C$ denotes a general concept, i.e., $C \rightarrow A \mid \neg A \mid \exists Q \mid \neg \exists Q$, where $A$ is an atomic concept.
Reasoning services

One can show that in *DL-Lite* all TBox reasoning services can be reduced to **ontology satisfiability**.

Hence, in the following, we concentrate on:

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

- **Query answering**: Given a query $q$ over an ontology $\mathcal{O}$, find all tuples $\vec{c}$ of constants such that $\mathcal{O} \models q(\vec{c})$. 
Query answering vs. ontology satisfiability

- In the case in which an ontology is unsatisfiable, according to the “ex falso quod libet” principle, reasoning is trivialized.

- In particular, query answering is meaningless, since every tuple is in the answer to every query.

- We are not interested in encoding meaningless query answering into the perfect reformulation of the input query. Therefore, before query answering, we will always check ontology satisfiability to single out meaningful cases.

Thus, we proceed as follows:

1. We show how to do query answering over satisfiable ontologies.
2. We show how we can exploit the query answering algorithm also to check ontology satisfiability.
Outline of Part 3

1. The DL-Lite family of tractable Description Logics

2. Reasoning in DL-Lite
   - Query answering over satisfiable ontologies
   - Ontology satisfiability
   - Complexity of reasoning in DL-Lite

3. Linking ontologies to relational data

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

1. **Using** $T$, **rewrite** $q$ into a UCQ $r_{q,T}$ (the perfect rewriting of $q$ w.r.t. $T$).
2. **Evaluate** $r_{q,T}$ over $\mathcal{A}$ (simply viewed as data), to return $\text{cert}(q, O)$.

Correctness of this procedure shows **FOL-rewritability** of query answering in $DL$-$Lite$. 
Positive vs. negative inclusions

We call positive inclusions (PIs) assertions of the form

\[ A_1 \sqsubseteq A_2 \]
\[ A_1 \sqsubseteq \exists Q_2 \]
\[ \exists Q_1 \sqsubseteq A_2 \]
\[ \exists Q_1 \sqsubseteq \exists Q_2 \]
\[ Q_1 \sqsubseteq Q_2 \]

We call negative inclusions (NIs) assertions of the form

\[ A_1 \sqsubseteq \neg A_2 \]
\[ A_1 \sqsubseteq \neg \exists Q_2 \]
\[ \exists Q_1 \sqsubseteq \neg A_2 \]
\[ \exists Q_1 \sqsubseteq \neg \exists Q_2 \]
\[ Q_1 \sqsubseteq \neg Q_2 \]
Query rewriting

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

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

\[
\text{AssistantProf} \sqsubseteq \text{Professor}
\]

as a logic rule:

\[
\text{Professor}(z) \leftarrow \text{AssistantProf}(z)
\]

**Basic rewriting step:**

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

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

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

\[
q(x) \leftarrow \text{AssistantProf}(x)
\]
Query rewriting (cont’d)

Consider the query
\[ q(x) \leftarrow \text{teaches}(x,y), \text{Course}(y) \]
and the PI
\[ \exists \text{teaches} \sqsubseteq \exists \text{Course} \]
as a logic rule:
\[ \text{Course}(z_2) \leftarrow \text{teaches}(z_1, z_2) \]

The PI applies to the atom \( \text{Course}(y) \), and we add to the perfect rewriting the query
\[ q(x) \leftarrow \text{teaches}(x,y), \text{teaches}(z_1, y) \]

Consider now the query
\[ q(x) \leftarrow \text{teaches}(x,y) \]
and the PI
\[ \text{Professor} \sqsubseteq \exists \text{teaches} \]
as a logic rule:
\[ \text{teaches}(z, f(z)) \leftarrow \text{Professor}(z) \]

The PI applies to the atom \( \text{teaches}(x,y) \), and we add to the perfect rewriting the query
\[ q(x) \leftarrow \text{Professor}(x) \]
Query rewriting – Constants

Conversely, for the query \( q(x) \leftarrow \text{teaches}(x, f1) \)

and the same PI as before \( \text{Professor} \sqsubseteq \exists \text{teaches} \)
as a logic rule: \( \text{teaches}(z, f(z)) \leftarrow \text{Professor}(z) \)

teaches\((x, f1)\) does not unify with teaches\((z, f(z))\), since the skolem term \( f(z) \) in the head of the rule does not unify with the constant \( f1 \).

Remember: We adopt the unique name assumption.

In this case, we say that the PI does not apply to the atom \( \text{teaches}(x, f1) \).

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

\( q(x, y) \leftarrow \text{teaches}(x, y) \)
Query rewriting – Join variables

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

Consider the query  \( q(x) \leftarrow \text{teaches}(x, y), \text{Course}(y) \)  and the PI  \( \text{Professor} \sqsubseteq \exists \text{teaches} \)  as a logic rule:  \( \text{teaches}(z, f(z)) \leftarrow \text{Professor}(z) \)  The PI above does not apply to the atom  \( \text{teaches}(x, y) \).
Consider now the query $q(x) \leftarrow \text{teaches}(x, y), \text{teaches}(z, y)$ and the PI

\[ \text{Professor} \sqsubseteq \exists \text{teaches} \]

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

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

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

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

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

$$ q(x) \leftarrow \text{Professor}(x) $$
Query rewriting – Summary

Reformulate the CQ $q$ into a set of queries:

- Apply to $q$ and the computed queries in all possible ways the PIs in $\mathcal{T}$:

$$
A_1 \sqsubseteq A_2 \quad \ldots, A_2(x), \ldots \Rightarrow \ldots, A_1(x), \ldots \\
\exists P \sqsubseteq A \quad \ldots, A(x), \ldots \Rightarrow \ldots, P(x, \_), \ldots \\
\exists P^- \sqsubseteq A \quad \ldots, A(x), \ldots \Rightarrow \ldots, P(\_, x), \ldots \\
A \sqsubseteq \exists P \quad \ldots, P(x, \_), \ldots \Rightarrow \ldots, A(x), \ldots \\
A \sqsubseteq \exists P^- \quad \ldots, P(\_, x), \ldots \Rightarrow \ldots, A(x), \ldots \\
\exists P_1 \sqsubseteq \exists P_2 \quad \ldots, P_2(x, \_), \ldots \Rightarrow \ldots, P_1(x, \_), \ldots \\
P_1 \sqsubseteq P_2 \quad \ldots, P_2(x, y), \ldots \Rightarrow \ldots, P_1(x, y), \ldots \\
$$

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

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

- Apply in all possible ways unification between atoms in a query.

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

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

Algorithm \textit{PerfectRef}(Q, T_P)

\textbf{Input:} union of conjunctive queries \( Q \), set of DL-Lite \( \mathcal{A} \) PIs \( T_P \)

\textbf{Output:} union of conjunctive queries \( PR \)

\( PR := Q; \)

\textbf{repeat}

\( PR' := PR; \)

\textbf{for each} \( q \in PR' \) \textbf{do}

\textbf{for each} \( g \) \textbf{in} \( q \) \textbf{do}

\textbf{for each} PI \( I \) \textbf{in} \( T_P \) \textbf{do}

\hspace{1cm} \textbf{if} \( I \) \textbf{is applicable to} \( g \) \textbf{then} \( PR := PR \cup \{ \text{ApplyPI}(q, g, I) \} \);

\textbf{for each} \( g_1, g_2 \) \textbf{in} \( q \) \textbf{do}

\hspace{1cm} \textbf{if} \( g_1 \) \textbf{and} \( g_2 \) \textbf{unify} \textbf{then} \( PR := PR \cup \{ \tau(\text{Reduce}(q, g_1, g_2)) \} \);

\textbf{until} \( PR' = PR; \)

\textbf{return} \( PR \)

\textbf{Observations:}

- Termination follows from having only finitely many different rewritings.
- NIs or functionalities do not play any role in the rewriting of the query.
Query answering in *DL-Lite* – Example

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

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

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

**ABox:**  
$\text{teaches}(\text{john}, \text{fl})$  
$\text{Professor}(\text{mary})$

It is easy to see that evaluating the perfect rewriting over the ABox viewed as a database produces as answer $\{\text{john}, \text{mary}\}$.
Query answering in \textit{DL-Lite} – An interesting example

**TBox:** \begin{align*}
\text{Person} & \sqsubseteq \exists \text{hasFather} \\
\exists \text{hasFather}^- & \sqsubseteq \text{Person}
\end{align*}

**ABox:** \text{Person(mary)}

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

\begin{align*}
q(x) & \leftarrow \text{Person}(x), \text{hasFather}(x, y_1), \text{hasFather}(y_1, y_2), \text{hasFather}(y_2, y_3) \\
& \quad \Downarrow \text{Apply } \text{Person} \sqsubseteq \exists \text{hasFather} \text{ to the atom } \text{hasFather}(y_2, y_3)
\end{align*}

\begin{align*}
q(x) & \leftarrow \text{Person}(x), \text{hasFather}(x, y_1), \text{hasFather}(y_1, y_2), \text{Person}(y_2) \\
& \quad \Downarrow \text{Apply } \exists \text{hasFather}^- \sqsubseteq \text{Person} \text{ to the atom } \text{Person}(y_2)
\end{align*}

\begin{align*}
q(x) & \leftarrow \text{Person}(x), \text{hasFather}(x, y_1), \text{hasFather}(y_1, y_2), \text{hasFather}(\_ , y_2) \\
& \quad \Downarrow \text{Unify} \text{ atoms } \text{hasFather}(y_1, y_2) \text{ and } \text{hasFather}(\_ , y_2)
\end{align*}

\begin{align*}
q(x) & \leftarrow \text{Person}(x), \text{hasFather}(x, y_1), \text{hasFather}(y_1, y_2) \\
& \quad \Downarrow \ldots
\end{align*}

\begin{align*}
q(x) & \leftarrow \text{Person}(x), \text{hasFather}(x, \_ ) \\
& \quad \Downarrow \text{Apply } \text{Person} \sqsubseteq \exists \text{hasFather} \text{ to the atom } \text{hasFather}(x, \_ )
\end{align*}

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

\[ \text{unibz} \]
Query answering over satisfiable *DL-Lite* ontologies

For an ABox $A$ and a query $q$ over $A$, let $Eval_{CWA}(q, A)$ denote the evaluation of $q$ over $A$ considered as a database (i.e., considered under the CWA).

**Theorem**

Let $T$ be a *DL-Lite* TBox, $T_P$ the set of PIs in $T$, and $q$ a CQ over $T$. Then, for each ABox $A$ such that $⟨T, A⟩$ is satisfiable, we have that

$$cert(q, ⟨T, A⟩) = Eval_{CWA}(PerfectRef(q, T_P), A).$$

As a consequence, query answering over a satisfiable *DL-Lite* ontology is FOL-rewritable.

Notice that we did not use NIs or functionality assertions of $T$ in computing $cert(q, ⟨T, A⟩)$. Indeed, when the ontology is satisfiable, we can ignore NIs and functionality assertions for query answering.
The proof of the previous result exploits a fundamental property of *DL-Lite*, that relies on the following notion.

**Def.: Canonical model**

Let $\mathcal{O} = \langle T, A \rangle$ be a *DL-Lite* ontology. A model $\mathcal{I}_\mathcal{O}$ of $\mathcal{O}$ is called **canonical** if for every model $\mathcal{I}$ of $\mathcal{O}$ there is a homomorphism from $\mathcal{I}_\mathcal{O}$ to $\mathcal{I}$.

**Theorem**

Every satisfiable *DL-Lite* ontology has a **canonical model**.

Properties of the canonical models of a *DL-Lite* ontology:

- A canonical model is in general infinite.
- All canonical models are homomorphically equivalent, hence we can do as if there was a single canonical model.
From the definition of canonical model, and since homomorphisms are closed under composition, we get that:

To compute the certain answer to a query $q$ over an ontology $\mathcal{O}$, one could in principle evaluate $q$ over a canonical model $\mathcal{I}_\mathcal{O}$ of $\mathcal{O}$.

- This does not give us directly an algorithm for query answering over an ontology $\mathcal{O} = \langle T, A \rangle$, since $\mathcal{I}_\mathcal{O}$ may be infinite.
- However, one can show that evaluating $q$ over $\mathcal{I}_\mathcal{O}$ amounts to evaluating the perfect rewriting $r_{q,T}$ over $A$. 
<table>
<thead>
<tr>
<th>The DL-Lite family</th>
<th>Reasoning in DL-Lite</th>
<th>Linking ontologies to relational data</th>
<th>Conclusions and further work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ontology satisfiability</td>
<td>Query answering over satisfiable ontologies</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ontology satisfiability</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Complexity of reasoning in DL-Lite</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Linking ontologies to relational data</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Conclusions and further work</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Outline of Part 3

1. The *DL-Lite* family of tractable Description Logics
2. Reasoning in *DL-Lite*
   - Query answering over satisfiable ontologies
   - Ontology satisfiability
   - Complexity of reasoning in *DL-Lite*
3. Linking ontologies to relational data
4. Conclusions and further work
Let us now consider the problem of establishing whether an ontology is satisfiable.

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

**Theorem**

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

Unsatisfiability in $DL-Lite_A$ ontologies can be caused by NIs or by functionality assertions.

**Example**

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

ABox $\mathcal{A}$:  
Student($john$)  
teaches($john$, fl)  
teaches($michael$, fl)
Checking satisfiability of $DL$-$Lite_A$ ontologies

Satisfiability of a $DL$-$Lite_A$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is reduced to evaluating over $DB(\mathcal{A})$ a UCQ that asks for the existence of objects violating the NI and functionality assertions.

Let $\mathcal{T}_P$ the set of PIs in $\mathcal{T}$.

We deal with NIs and functionality assertions differently.

For each NI $N \in \mathcal{T}$:

1. we construct a boolean CQ $q_N()$ such that
   $$\langle \mathcal{T}_P, \mathcal{A} \rangle \models q_N() \iff \langle \mathcal{T}_P \cup \{N\}, \mathcal{A} \rangle \text{ is unsatisfiable}$$

2. We check whether $\langle \mathcal{T}_P, \mathcal{A} \rangle \models q_N()$ using PerfectRef, i.e., we compute $\text{PerfectRef}(q_N, \mathcal{T}_P)$, and evaluate it over $DB(\mathcal{A})$.

For each functionality assertion $F \in \mathcal{T}$:

1. we construct a boolean CQ $q_F()$ such that
   $$\mathcal{A} \models q_F() \iff \langle \{F\}, \mathcal{A} \rangle \text{ is unsatisfiable}.$$  

2. We check whether $\mathcal{A} \models q_F()$, by simply evaluating $q_F$ over $DB(\mathcal{A})$. 

© D. Calvanese (FUB)
Checking violations of negative inclusions

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

$$A_1 \sqsubseteq \neg A_2 \quad \leadsto \quad q_N() \leftarrow A_1(x), A_2(x)$$

$$\exists P \sqsubseteq \neg A \quad \text{or} \quad A \sqsubseteq \neg \exists P \quad \leadsto \quad q_N() \leftarrow P(x, y), A(x)$$

$$\exists P^- \sqsubseteq \neg A \quad \text{or} \quad A \sqsubseteq \neg \exists P^- \quad \leadsto \quad q_N() \leftarrow P(y, x), A(x)$$

$$\exists P_1 \sqsubseteq \neg \exists P_2 \quad \leadsto \quad q_N() \leftarrow P_1(x, y), P_2(x, z)$$

$$\exists P_1 \sqsubseteq \neg \exists P_2^- \quad \leadsto \quad q_N() \leftarrow P_1(x, y), P_2(z, x)$$

$$\exists P_1^- \sqsubseteq \neg \exists P_2 \quad \leadsto \quad q_N() \leftarrow P_1(x, y), P_2(y, z)$$

$$\exists P_1^- \sqsubseteq \neg \exists P_2^- \quad \leadsto \quad q_N() \leftarrow P_1(x, y), P_2(z, y)$$

$$P_1 \sqsubseteq \neg P_2 \quad \text{or} \quad P_1^- \sqsubseteq \neg P_2^- \quad \leadsto \quad q_N() \leftarrow P_1(x, y), P_2(x, y)$$

$$P_1^- \sqsubseteq \neg P_2 \quad \text{or} \quad P_1 \sqsubseteq \neg P_2^- \quad \leadsto \quad q_N() \leftarrow P_1(x, y), P_2(y, x)$$
Checking violations of negative inclusions – Example

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

Nls $\mathcal{N}$ : $\text{Professor} \sqsubseteq \neg \text{Student}$

Query $q_{\mathcal{N}}$: $q_{\mathcal{N}}(x) \leftarrow \text{Student}(x), \text{Professor}(x)$

Perfect Rewriting:

$q_{\mathcal{N}}(x) \leftarrow \text{Student}(x), \text{Professor}(x)$
$q_{\mathcal{N}}(x) \leftarrow \text{Student}(x), \text{teaches}(x, \_)$

ABox $\mathcal{A}$: $\text{teaches(john, fl)}$

$\text{Student(john)}$

It is easy to see that $\langle \mathcal{T}_P, \mathcal{A} \rangle \models q_{\mathcal{N}}(x)$, and that the ontology $\langle \mathcal{T}_P \cup \{\text{Professor} \sqsubseteq \neg \text{Student}\}, \mathcal{A} \rangle$ is unsatisfiable.
Boolean queries vs. non-boolean queries for NIs

To ensure correctness of the method, the queries used to check for the violation of a NI need to be boolean.

Example

TBox $\mathcal{T}$:  
\[
A_1 \sqsubseteq \neg A_0 \\
A_1 \sqsubseteq A_0 \\
A_1 \sqsubseteq A_0 \\
\exists P \sqsubseteq A_1 \\
A_2 \sqsubseteq \exists P^-
\]

ABox $\mathcal{A}$:  
\[
A_2(c)
\]

Since $A_1$, $P$, and $A_2$ are unsatisfiable, also $\langle \mathcal{T}, \mathcal{A} \rangle$ is unsatisfiable.

Consider the query corresponding to the NI $A_1 \sqsubseteq \neg A_0$.

\[
q_N() \leftarrow A_1(x), A_0(x)
\]

Then $\text{PerfectRef}(q_N, \mathcal{T}_P)$ is:

\[
q_N() \leftarrow A_1(x), A_0(x) \\
q_N() \leftarrow A_1(x) \\
q_N() \leftarrow P(x, -) \\
q_N() \leftarrow A_2(-)
\]

We have that $\langle \mathcal{T}_P, \mathcal{A} \rangle \models q_N()$.

\[
q'_{N}(x) \leftarrow A_1(x), A_0(x)
\]

Then $\text{PerfectRef}(q'_{N}, \mathcal{T}_P)$ is

\[
q'_{N}(x) \leftarrow A_1(x), A_0(x) \\
q'_{N}(x) \leftarrow A_1(x) \\
q'_{N}(x) \leftarrow P(x, -)
\]

$\text{cert}(q'_{N}, \langle \mathcal{T}_P, \mathcal{A} \rangle) = \emptyset$, hence $q'_{N}(x)$ does not detect unsatisfiability.
Checking violations of functionality assertions

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

\[
\begin{align*}
(funct\ P) & \implies q_F() \leftarrow P(x, y), P(x, z), y \neq z \\
(funct\ P^-) & \implies q_F() \leftarrow P(x, y), P(z, y), x \neq z
\end{align*}
\]

**Example**

**Functionality** \( F: \ (funct\ teaches^-) \)

**Query** \( q_F: \ q_F() \leftarrow teaches(x, y), teaches(z, y), x \neq z \)

**ABox** \( A: \)

\begin{align*}
teaches(john, fl) \\
teaches(michael, fl)
\end{align*}

It is easy to see that \( A \models q_F() \), and that \( \langle \{(funct\ teaches^-)\}, A \rangle \), is **unsatisfiable**.
From satisfiability to query answering in $DL$-$Lite_\mathcal{A}$

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

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

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

(b) There exists a functionality assertion $F \in \mathcal{T}$ such that $\mathcal{A} \models q_F()$.

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

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

Notably, to check ontology satisfiability, each NI and each functionality assertion can be processed individually.
From the previous lemma and the theorem on query answering for satisfiable $DL\text{-}Lite_{\mathcal{A}}$ ontologies, we get the following result.

**Theorem**

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

(a) There exists a NI $N \in \mathcal{T}$ s.t. $Eval_{\text{cwa}}(\text{PerfectRef}(q_N, \mathcal{T}_P), \mathcal{A})$ returns true.

(b) There exists a func. assertion $F \in \mathcal{T}$ s.t. $Eval_{\text{cwa}}(q_F, \mathcal{A})$ returns true.

**Note:** All the queries $q_N()$ and $q_F()$ can be combined into a single UCQ. Hence, satisfiability of a $DL\text{-}Lite_{\mathcal{A}}$ ontology is reduced to evaluating a FOL-query over an ontology whose TBox consists of positive inclusions only (and hence is satisfiable).
Outline of Part 3

1. The DL-Lite family of tractable Description Logics

2. Reasoning in DL-Lite
   - Query answering over satisfiable ontologies
   - Ontology satisfiability
   - Complexity of reasoning in DL-Lite

3. Linking ontologies to relational data

4. Conclusions and further work
The DL-Lite family

Reasoning in DL-Lite

Linking ontologies to relational data

Conclusions and further work

Part 3: Ontology Based Data Access

Complexity results for DL-Lite

TBox reasoning

- \( \text{PTIME} \) in the size of the TBox (schema complexity)

Ontology satisfiability

- \( \text{PTIME} \) in the size of the ontology (combined complexity)
- \( \text{AC}^0 \) in the size of the ABox (data complexity)

Query answering

- NP-complete in the size of query and ontology (combined complexity)
- \( \text{PTIME} \) in the size of the ontology (schema+data complexity)
- \( \text{AC}^0 \) in the size of the ABox (data complexity)

Can we further extend these results to more expressive ontology languages?

**Essentially NO!**

(Unless we take particular care)
Beyond $DL-Lite_\mathcal{A}$: results on data complexity

Essentially all extensions of $DL-Lite$ with additional DL constructs, or with combinations of constructs that are not legal in $DL-Lite$, make it lose its nice computational properties [Calvanese et al., 2013b].

<table>
<thead>
<tr>
<th>Lhs of inclusions</th>
<th>Rhs of inclusions</th>
<th>Funct.</th>
<th>Role incl.</th>
<th>Data complexity of query answering</th>
</tr>
</thead>
<tbody>
<tr>
<td>$DL-Lite_\mathcal{A}$</td>
<td>$A$</td>
<td>$\sqrt{*}$</td>
<td>$\sqrt{*}$</td>
<td>in $AC^0$</td>
</tr>
<tr>
<td>$A \sqsubseteq \exists P . A$</td>
<td>$A$</td>
<td>$-$</td>
<td>$-$</td>
<td>NLogSpace-hard</td>
</tr>
<tr>
<td>$A$</td>
<td>$A \sqsubseteq \forall P . A$</td>
<td>$-$</td>
<td>$-$</td>
<td>NLogSpace-hard</td>
</tr>
<tr>
<td>$A$</td>
<td>$A \sqsubseteq \exists P . A$</td>
<td>$\sqrt{*}$</td>
<td>$-$</td>
<td>NLogSpace-hard</td>
</tr>
<tr>
<td>$A \sqsubseteq \exists P . A \sqcap A_1 \sqcap A_2$</td>
<td>$A$</td>
<td>$-$</td>
<td>$-$</td>
<td>PTime-hard</td>
</tr>
<tr>
<td>$A \sqsubseteq A_1 \sqcap A_2$</td>
<td>$A \sqsubseteq \forall P . A$</td>
<td>$-$</td>
<td>$-$</td>
<td>PTime-hard</td>
</tr>
<tr>
<td>$A \sqsubseteq A_1 \sqcap A_2$</td>
<td>$A \sqsubseteq \exists P . A$</td>
<td>$\sqrt{*}$</td>
<td>$-$</td>
<td>PTime-hard</td>
</tr>
<tr>
<td>$A \sqsubseteq \exists P . A \sqcap \exists P^-. A$</td>
<td>$A \sqsubseteq \exists P$</td>
<td>$-$</td>
<td>$-$</td>
<td>PTime-hard</td>
</tr>
<tr>
<td>$A \sqsubseteq \exists P \sqcap \exists P^-$</td>
<td>$A \sqsubseteq \exists P \sqcap \exists P^-$</td>
<td>$\sqrt{*}$</td>
<td>$\sqrt{*}$</td>
<td>PTime-hard</td>
</tr>
<tr>
<td>$A \sqsubseteq \neg A$</td>
<td>$A$</td>
<td>$-$</td>
<td>$-$</td>
<td>coNP-hard</td>
</tr>
<tr>
<td>$A$</td>
<td>$A \sqsubseteq A_1 \sqcup A_2$</td>
<td>$-$</td>
<td>$-$</td>
<td>coNP-hard</td>
</tr>
<tr>
<td>$A \sqsubseteq \forall P . A$</td>
<td>$A$</td>
<td>$-$</td>
<td>$-$</td>
<td>coNP-hard</td>
</tr>
</tbody>
</table>

- 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! $\sim$ No hope of including covering constraints.
Combining functionalities and role inclusions

One can show that, by including in DL-Lite both functionality of roles and role inclusions \textit{without restrictions on their interaction} [Artale \textit{et al.}, 2009]:

- query answering becomes \( \text{PTime} \)-hard in data complexity;
- the complexity of TBox reasoning jumps from \( \text{NLogSpace} \) to \( \text{ExpTime} \)-complete.

\textit{Recall that, when the data complexity of query answering is above } \( \text{AC}^0 \), \textit{the DL does not enjoy FOL-rewritability.}

\textbf{As a consequence of these results, we get:}

The restriction on the interaction of functionality and role inclusions of \( DL\text{-Lite}_A \) is necessary:

- to preserve FOL-rewritability of query answering and ontology satisfiability;
- to guarantee efficient reasoning on the TBox (i.e., at the schema level).
Outline of Part 3

1. The DL-Lite family of tractable Description Logics

2. Reasoning in DL-Lite

3. Linking ontologies to relational data
   - The impedance mismatch problem
   - Ontology-Based Data Access systems
   - Query answering in Ontology-Based Data Access systems
   - Optimizing OBDA in Ontop

4. Conclusions and further work
Outline of Part 3

1. The DL-Lite family of tractable Description Logics

2. Reasoning in DL-Lite

3. Linking ontologies to relational data
   - The impedance mismatch problem
   - Ontology-Based Data Access systems
   - Query answering in Ontology-Based Data Access systems
   - Optimizing OBDA in Ontop

4. Conclusions and further work
Data in external sources

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

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

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

The architecture of an OBDA system is based on three main components:

- **Ontology**: provides a unified, conceptual view of the managed information.
- **Data source(s)**: are external and independent (possibly multiple and heterogeneous).
- **Mappings**: semantically link data at the sources with the ontology.
The impedance mismatch problem

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

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

**Solution:**

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

**Note:** the ABox is only virtual, and the objects are not materialized.
Solution to the impedance mismatch problem

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

- Each mapping assertion maps:
  - a query that retrieves values from a data source to...
  - a set of atoms specified over the ontology.

- Basic idea: use **Skolem functions** in the atoms over the ontology to "generate" the objects from the data values.

- Semantics of mappings:
  - Objects are denoted by terms (of exactly one level of nesting).
  - Different terms denote different objects (i.e., we make the unique name assumption on terms).
Impedance mismatch – Example

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

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

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

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

Intuitively:

- An employee should be created from her SSN: \texttt{pers}(SSN)
- A project should be created from its name: \texttt{proj}(PrName)
Creating object identifiers

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

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

**Example**

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

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

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

A **mapping assertion** between a database with schema \( S \) and a TBox \( \mathcal{T} \) has the form

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

where

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

D₁[SSN: String, PrName: String]
Employees and Projects they work for

D₂[Code: String, Salary: Int]
Employee’s code with salary

D₃[Code: String, SSN: String]
Employee’s code with SSN

\[ m₁: \text{SELECT SSN, PrName} \quad \sim \quad \text{Employee}(\text{pers}(\text{SSN})), \]
\[ \text{Project}(\text{proj}(\text{PrName})), \text{projName}(\text{proj}(\text{PrName}), \text{PrName}), \]
\[ \text{worksFor}(\text{pers}(\text{SSN}), \text{proj}(\text{PrName})) \]

\[ m₂: \text{SELECT SSN, Salary} \quad \sim \quad \text{Employee}(\text{pers}(\text{SSN})), \]
\[ \text{salary}(\text{pers}(\text{SSN}), \text{Salary}) \]

\[ \text{WHERE } D₂.\text{Code} = D₃.\text{Code} \]
Concrete mapping languages

Several proposals for concrete languages to map a relational DB to an ontology:

- They assume that the ontology is populated in terms of RDF triples.
- Some template mechanism is used to specify the triples to instantiate.

Examples: D2RQ\(^1\), SML\(^2\), Ontop\(^3\)

**R2RML**

- Most popular RDB to RDF mapping language
- W3C Recommendation 27 Sep. 2012, [http://www.w3.org/TR/r2rml/](http://www.w3.org/TR/r2rml/)
- R2RML mappings are themselves expressed as RDF graphs and written in Turtle syntax.

---

\(^1\)[http://d2rq.org/d2rq-language](http://d2rq.org/d2rq-language)

\(^2\)[http://sparqlify.org/wiki/Sparqlification_mapping_language](http://sparqlify.org/wiki/Sparqlification_mapping_language)

Outline of Part 3

1. The DL-Lite family of tractable Description Logics

2. Reasoning in DL-Lite

3. Linking ontologies to relational data
   - The impedance mismatch problem
   - Ontology-Based Data Access systems
     - Query answering in Ontology-Based Data Access systems
     - Optimizing OBDA in Ontop

4. Conclusions and further work
To formalize OBDA, we distinguish between the intensional and the extensional level information.

An **OBDA specification** is a triple $\mathcal{P} = \langle T, S, M \rangle$, where:

- $T$ is a DL TBox providing the intensional level of an ontology.
- $S$ is a (possibly federated) relational database schema for the data sources, possibly with constraints;
- $M$ is a set of mapping assertions between $T$ and $S$.

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

- $\mathcal{P} = \langle T, S, M \rangle$ is an OBDA specification, and
- $D$ is a relational database compliant with $S$. 
Semantics of an OBDA system: Intuition

In an OBDA system, the mapping $\mathcal{M}$ encodes how the data $\mathcal{D}$ in the source(s) $\mathcal{S}$ should be used to populate the elements of the TBox $\mathcal{T}$.

The data $\mathcal{D}$ and the mapping $\mathcal{M}$ define a virtual data layer $\mathcal{V}$, which behaves like a (virtual) ABox.

- Queries are answered w.r.t. $\mathcal{T}$ and $\mathcal{V}$.
- One aim is to avoid materializing the data of $\mathcal{V}$.
- Instead, the intensional information in $\mathcal{T}$ and $\mathcal{M}$ is used to translate queries over $\mathcal{T}$ into queries formulated over $\mathcal{S}$.

OBDA vs. Ontology Based Query Answering (OBQA)

OBDA relies on OBQA to process queries w.r.t. the TBox $\mathcal{T}$, but in addition is concerned with efficiently dealing with the mapping $\mathcal{M}$.

**OBDA should not be confused with OBQA.**
Semantics of mappings

To formally define the semantics of an OBDA system $O = \langle P, D \rangle$, where $P = \langle T, S, M \rangle$, we first need to define the semantics of mappings.

Satisfaction of a mapping assertion with respect to a database

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

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

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

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

The following interpretation $\mathcal{I}$ satisfies the mapping assertions $m_1$ and $m_2$ with respect to the above database:

$m_1$: \[
\text{SELECT SSN, PrName FROM } D_1
\]
$\rightsquigarrow$

$m_2$: \[
\text{SELECT SSN, Salary FROM } D_2, D_3
\]
$\rightsquigarrow$

The following interpretation $\mathcal{I}$ satisfies the mapping assertions $m_1$ and $m_2$ with respect to the above database:

$\mathcal{I}: \Delta^O_{\mathcal{I}} = \{\text{pers}(23AB), \text{proj}(\text{optique}), \ldots\}$, $\Delta^V_{\mathcal{I}} = \{\text{optique}, 1500, \ldots\}$
Employee$^\mathcal{I} = \{\text{pers}(23AB), \ldots\}$, Project$^\mathcal{I} = \{\text{proj}(\text{optique}), \ldots\}$,
projName$^\mathcal{I} = \{(\text{proj}(\text{optique}), \text{optique}), \ldots\}$,
worksFor$^\mathcal{I} = \{(\text{pers}(23AB), \text{proj}(\text{optique})), \ldots\}$,
salary$^\mathcal{I} = \{(\text{pers}(23AB), 1500), \ldots\}$
Semantics of an OBDA system

**Model** of an OBDA system

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

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

An OBDA system $\mathcal{O}$ is **satisfiable** if it admits at least one model.
Outline of Part 3

1. The DL-Lite family of tractable Description Logics
2. Reasoning in DL-Lite
3. Linking ontologies to relational data
   - The impedance mismatch problem
   - Ontology-Based Data Access systems
   - Query answering in Ontology-Based Data Access systems
   - Optimizing OBDA in Ontop
4. Conclusions and further work
Answering queries over an OBDA system

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

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

Two approaches to exploit the mapping:

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

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

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

$$\Phi \rightsquigarrow X_1 \quad \ldots \quad \Phi \rightsquigarrow X_k$$

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

Example

$m_1$: SELECT SSN, PrName FROM D$_1$ $\rightsquigarrow$ Employee(pers(SSN)), Project(proj(PrName)), projectName(proj(PrName), PrName), worksFor(pers(SSN), proj(PrName))

is split into

$m_1^1$: SELECT SSN, PrName FROM D$_1$ $\rightsquigarrow$ Employee(pers(SSN))

$m_1^2$: SELECT SSN, PrName FROM D$_1$ $\rightsquigarrow$ Project(proj(PrName))

$m_1^3$: SELECT SSN, PrName FROM D$_1$ $\rightsquigarrow$ projectName(proj(PrName), PrName)

$m_1^4$: SELECT SSN, PrName FROM D$_1$ $\rightsquigarrow$ worksFor(pers(SSN), proj(PrName))
Bottom-up approach to query answering

Consists in a straightforward application of the mappings:

1. Propagate the data from $D$ through $M$, materializing an ABox $A_{M,D}$ (the constants in such an ABox are values and object terms).

2. Apply to $A_{M,D}$ and to the TBox $T$, the satisfiability and query answering algorithms developed for $DL-Lite_A$.

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

- The technique is no more $AC^0$ in the data, since the ABox $A_{M,D}$ to materialize is in general polynomial in the size of the data.
- $A_{M,D}$ may be very large, and thus it may be infeasible to actually materialize it.
- Freshness of $A_{M,D}$ with respect to the underlying data source(s) may be an issue, and one would need to propagate source updates (cf. Data Warehousing).
### Top-down approach to query answering

Consists of three steps:

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

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

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

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

1. For each non-split mapping assertion $\Phi_i(x) \sim \Psi_i(t, y)$:
   1. Introduce a view symbol $\text{Aux}_i$ of arity equal to that of $\Phi_i$.
   2. Add a view definition $\text{Aux}_i(x) \leftarrow \Phi_i(x)$.

2. For each split version $\Phi_i(x) \sim X_j(t, y)$ of a mapping assertion, introduce a clause $X_j(t, y) \leftarrow \text{Aux}_i(x)$.

3. Obtain from $q_{pr}$ in all possible ways queries $q_{aux}$ defined over the view symbols $\text{Aux}_i$ as follows:
   1. Find a most general unifier $\vartheta$ that unifies each atom $X(z)$ in the body of $q_{pr}$ with the head of a clause $X(t, y) \leftarrow \text{Aux}_i(x)$.
   2. Substitute each atom $X(z)$ with $\vartheta(\text{Aux}_i(x))$, i.e., with the body the unified clause to which the unifier $\vartheta$ is applied.

4. The unfolded query $q_{unf}$ is the union of all queries $q_{aux}$, together with the view definitions for the predicates $\text{Aux}_i$ appearing in $q_{aux}$.

© D. Calvanese (FUB)
Unfolding – Example

\[
m_1: \text{SELECT SSN, PrName} \quad \sim \quad \text{Employee}(\text{pers}(SSN)),
\]
\[
\text{Project}(\text{proj}(PrName)),
\]
\[
\text{projectName}(\text{proj}(PrName), PrName),
\]
\[
\text{worksFor}(\text{pers}(SSN), \text{proj}(PrName))
\]

\[
m_2: \text{SELECT SSN, Salary} \quad \sim \quad \text{Employee}(\text{pers}(SSN)),
\]
\[
\text{salary}(\text{pers}(SSN), Salary)
\]

We define a view \( Aux_i \) for the source query of each mapping \( m_i \).

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

\[
\text{Employee}(\text{pers}(SSN)) \leftarrow \text{Aux}_1(SSN, PrName)
\]
\[
\text{proj}(\text{proj}(PrName)) \leftarrow \text{Aux}_1(SSN, PrName)
\]
\[
\text{proj}(\text{proj}(PrName)) \leftarrow \text{Aux}_1(SSN, PrName)
\]
\[
\text{worksFor}(\text{pers}(SSN), \text{proj}(PrName)) \leftarrow \text{Aux}_1(SSN, PrName)
\]
\[
\text{Employee}(\text{pers}(SSN)) \leftarrow \text{Aux}_2(SSN, Salary)
\]
\[
\text{salary}(\text{pers}(SSN), Salary) \leftarrow \text{Aux}_2(SSN, Salary)
\]
Unfolding – Example (cont’d)

Query over ontology: employees who work for optique and their salary:

\[ q(e,s) \leftarrow \text{Employee}(e), \text{salary}(e,s), \text{worksFor}(e,p), \text{projectName}(p, \text{optique}) \]

A unifier between the atoms in \( q \) and the clause heads is:

\[ \vartheta(e) = \text{pers}(SSN) \quad \vartheta(s) = \text{Salary} \]
\[ \vartheta(\text{PrName}) = \text{optique} \quad \vartheta(p) = \text{proj}(\text{optique}) \]

After applying \( \vartheta \) to \( q \), we obtain:

\[ q(\text{pers}(SSN), \text{Salary}) \leftarrow \text{Employee}(\text{pers}(SSN)), \text{salary}(\text{pers}(SSN), \text{Salary}), \]
\[ \text{worksFor}(\text{pers}(SSN), \text{proj}(\text{optique})), \]
\[ \text{projectName}(\text{proj}(\text{optique}), \text{optique}) \]

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

\[ q(\text{pers}(SSN), \text{Salary}) \leftarrow \text{Aux}_1(SSN, \text{optique}), \text{Aux}_2(SSN, \text{Salary}), \]
\[ \text{Aux}_1(SSN, \text{optique}), \text{Aux}_1(SSN, \text{optique}) \]
Exponential blowup in the unfolding

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

Consider a query: \( q(y) \leftarrow A_1(y), A_2(y), \ldots, A_n(y) \)
and the mappings:
\[ m_i^1: \Phi_i^1(x) \leadsto A_i(f(x)) \quad \text{(for } i \in \{1, \ldots, n\}\) 
\[ m_i^2: \Phi_i^2(x) \leadsto A_i(f(x)) \]

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

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

Hence, we obtain one unfolded query
\[ q(f(x)) \leftarrow \text{Aux}_1^{j_1}(x), \text{Aux}_2^{j_2}(x), \ldots, \text{Aux}_n^{j_n}(x) \]

for each possible combination of \( j_i \in \{1, 2\}, \text{ for } i \in \{1, \ldots, n\}\)
Hence, we obtain \( 2^n \) unfolded queries.
Computational complexity of query answering

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

**Theorem**

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

- **1. NP-complete** in the size of the query.
- **2. PTIME** in the size of the TBox $T$ and the mappings $M$.
- **3. AC$^0$** in the size of the database $D$.

*Note:* The AC$^0$ result is a consequence of the fact that query answering in such a setting can be reduced to evaluating an SQL query over the relational database.
Implementation of top-down approach to query answering

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

We can follow different strategies:

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

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

Which method is better, depends on various parameters.
We have seen that answering full SQL (i.e., FOL) queries is undecidable.

However, we can treat the answers to an UCQ, as “knowledge”, and perform further computations on that knowledge.

This corresponds to applying a knowledge operator to UCQs that are embedded into an arbitrary SQL query (EQL queries) [Calvanese et al., 2007b]

- The UCQs are answered according to the certain answer semantics.
- The SQL query is evaluated on the facts returned by the UCQs.

The approach can be implemented by rewriting the UCQs and embedding the rewritten UCQs into SQL.

The user “sees” arbitrary SQL queries, but these SQL queries are evaluated according to a weakened semantics.
Outline of Part 3

1. The DL-Lite family of tractable Description Logics

2. Reasoning in DL-Lite

3. Linking ontologies to relational data
   - The impedance mismatch problem
   - Ontology-Based Data Access systems
   - Query answering in Ontology-Based Data Access systems
   - Optimizing OBDA in Ontop

4. Conclusions and further work
Experimentations and experiences

Several experimentations (in rough chronological order):
- Monte dei Paschi di Siena (led by Sapienza Univ. of Rome)
- Selex: world leading radar producer
- National Accessibility Portal of South Africa
- Horizontal Gene Transfer data and ontology
- Stanford’s “Resource Index” comprising 200 ontologies from BioPortal
- Norwegian Petroleum Directorate (NPD) FactPages (within Optique)
- Benchmarking on (partially) artificial data ongoing

Observations:
- Approach highly effective for bridging impedance mismatch between data sources and ontology.
- Rewriting technique effective against incompleteness in the data.

However, performance is a major issue that still prevents large-scale deployment of this technology.
Query processing in a traditional OBDA system

- **SPARQL query** ($q$) is rewritten using the ontology $\mathcal{T}$.
- The rewritten UCQ query ($q'$) is unfolded using the mapping $\mathcal{M}$.
- The unfolded SQL query ($q''$) is used to access the data $D$.

- Virtual ABox $\mathcal{V}$ is created through ABox virtualisation.

This process involves rewriting and unfolding steps to translate the SPARQL query into an equivalent SQL query that can access relational data.
What makes the resulting SQL query grow exponentially?

Three main factors affect the size of the resulting query $q''$:

**Existentials:** Sub-queries of $q$ with *existentially quantified variables* might lead in general to exponentially large rewritings.

**Hierarchies:** Concepts / roles occurring in the query $q$ can have many *subconcepts / subroles* according to $T$, which all have to be included in the rewriting $q'$.

**Mappings:** The mapping $M$ can provide *multiple definitions of the concepts and roles in the ontology*, which may result in a further exponential blowup in the unfolding step of $q'$ to $q''$. 
Impact of hierarchies – Example

Example

TBox $\mathcal{T}$:

$q(x) \leftarrow A(x), P(x,y), A(y), P(y,z), A(z)$

UCQ rewriting of $q$ w.r.t. $\mathcal{T}$ contains 729 CQs
i.e., a UNION of 729 SPJ SQL queries

The size of UCQ rewritings may become very large

- In the worst case, it may be $O((|\mathcal{T}| \cdot |q|)^{|q|})$, i.e., exponential in $|q|$.
- Unfortunately, this blowup occurs also in practice.
Taming the size of the rewriting

Note: It is not possible to avoid rewriting altogether, since this would require in general to materialize an infinite database [Calvanese et al., 2007c].

Several techniques have been proposed recently to limit the size of the rewriting:

- Alternative rewriting techniques [Pérez-Urbina et al., 2010]: more efficient algorithm based on resolution, but produces still an exponential UCQ.
- Combined approach [Kontchakov et al., 2010]: combines partial materialization with rewriting:
  - When $T$ contains no role inclusions rewriting is polynomial.
  - But in general rewriting is exponential.
  - Materialization requires control over the data sources and might not be applicable in an OBDA setting.
- Rewriting into non-recursive Datalog:
  - Presto system [Rosati and Almatelli, 2010]: still worst-case exponential.
  - Polynomial rewriting for Datalog$^\pm$ [Gottlob and Schwentick, 2012]: rewriting uses polynomially many new existential variables and “guesses” a relevant portion of the canonical model for the TBox.

See [Kikot et al., 2012; Gottlob et al., 2014] for discussion and further results.
A holistic approach to optimization

Recall our main objective

Given an OBDA specification \( \mathcal{P} = \langle T, M, S \rangle \), a database \( D \), and a set of queries, compute the certain answers of such queries w.r.t. \( \mathcal{O} = \langle \mathcal{P}, D \rangle \) as efficiently as possible.

Observe:

- The size of the rewriting is only one coordinate in the problem space.
- Optimizing rewriting is necessary but not sufficient, since the more compact rewritings are in general much more difficult to evaluate.
- In fact, the efficiency of the query evaluation by the DBMS is the crucial factor.

Hence, a holistic approach is required, that considers all components of an OBDA system, i.e.:

- the TBox \( T \),
- the mappings \( M \),
- the data sources \( D \) with their dependencies in \( S \), and
- the query load.
Optimizations in Ontop [Rodriguez-Muro et al., 2013]

1. Tree-witness rewriting over H-complete ABoxes.
2. $T$-mappings incorporating $T$ into $M$.
4. Optimized unfolding.
The Ontop OBDA framework

Developed at the Free Univ. of Bozen-Bolzano: http://ontop.inf.unibz.it/

“Stay on top of your data with semantics”

Features of Ontop

- Query language: support for SPARQL 1.0 (and part of 1.1)
- Mapping languages:
  - Intuitive Ontop mapping language
  - Support for R2RML W3C standard
- Database: Support for free and commercial DBMSs
  - PostgreSQL, MySQL, H2, DB2, ORACLE, MS SQL SERVER, TEIID, ADP
- Java library/providers for Sesame and OWL-API
  - Sesame: a de-facto standard framework for processing RDF data
  - OWL-API: Java API and reference implementation for OWL Ontologies
- Integrated with Protege 4.x
- Provides a SPARQL end-point (via Sesame Workbench)
- Apache open source license
Outline of Part 3

1. The DL-Lite family of tractable Description Logics
2. Reasoning in DL-Lite
3. Linking ontologies to relational data
4. Conclusions and further work
Main publications

Most of the results presented in this course have been published:

- **Reasoning and query answering in DL-Lite:** [Calvanese et al., 2005; Calvanese et al., 2006b; Calvanese et al., 2007c; Calvanese et al., 2007a; Artale et al., 2009; Calvanese et al., 2013b]

- **Mapping to data sources and OBDA:** [Calvanese et al., 2006a; Calvanese et al., 2008a; Poggi et al., 2008a]

- **Connection between description logics and conceptual modeling formalisms:** [Calvanese et al., 1998; Berardi et al., 2005; Artale et al., 2007; Calvanese et al., 2009b]

- **Tool descriptions:** [Poggi et al., 2008b; Rodríguez-Muro and Calvanese, 2008; Calvanese et al., 2011; Rodriguez-Muro and Calvanese, 2012]

- **Optimization techniques:** [Rodriguez-Muro et al., 2013; Kontchakov et al., 2014]

- **Case studies:** [Keet et al., 2008; Savo et al., 2010; Calvanese et al., 2013a]

A summary of many of the presented results and techniques, with detailed proofs is given in [Calvanese et al., 2009a].
Query rewriting for more expressive ontology languages

These results have then been extended to more expressive ontology languages, using different techniques:

- In [Artale et al., 2009] various DL-Lite extensions are considered, providing a comprehensive treatment of the expressiveness/complexity trade-off for the DL-Lite family and related logics:
  - number restrictions besides functionality;
  - conjunction on the left-hand side of inclusions (horn logics);
  - boolean constructs;
  - constraints on roles, such as (ir)reflexivity, (a)symmetry, transitivity;
  - presence and absence of the unique name assumption.

- Alternative query rewriting techniques based on resolution, and applicable also to more expressive logics (leading to recursive rewritings) [Pérez-Urbina et al., 2010].

- Query rewriting techniques for database inspired constraint languages [Calì et al., 2009a; Calì et al., 2009b; Calì et al., 2012; Gottlob et al., 2014].
Further theoretical work

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

- We have considered mainly query answering. However, several other ontology-based services are of importance:
  - write-also access: updating a data source through an ontology
    [De Giacomo et al., 2009; Calvanese et al., 2010; Zheleznyakov et al., 2010]
  - modularity and minimal module extraction
    [Kontchakov et al., 2008; Kontchakov et al., 2009]
  - privacy aware data access [Calvanese et al., 2008b]
  - meta-level reasoning and query answering, a la RDFS
    [De Giacomo et al., 2008]
  - provenance and explanation [Borgida et al., 2008]

- Reasoning with respect to finite models only [Rosati, 2008].

- We have dealt only with the static aspects of information systems. However a crucial issue is how to deal with dynamic aspects.
  See also work carried out in the EU project ACSI.

Work on most of these issues is still ongoing.
Further practical and experimental work

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

- The rewriting consists of a large number of CQs. Query containment can be used to prune the rewriting. This is already implemented in -ontop-system, but requires further optimizations.

- The SQL queries generated by the mapping unfolding are not easy to process by the DBMS engine (e.g., they may contain complex joins on skolem terms computed on the fly). Different mapping unfolding strategies have a strong impact on computational complexity. Experimentation is ongoing to assess the tradeoff.

- Further extensive experimentations are ongoing:
  - on artificially generated data;
  - on real-world use cases.

- An OBDA benchmarking suite is under development [Calvanese et al., 2014; Lanti et al., 2015].
Acknowledgements

Many people are/were involved in various aspects of this work:

- Alessandro Artale¹
- Benjamin Cogrel¹
- Giuseppe De Giacomo²
- Sarah Komla Ebri¹
- Roman Kontchakov³
- Davide Lanti¹
- Domenico Lembo²
- Maurizio Lenzerini²
- Antonella Poggi²
- Martin Rezk¹
- Mariano Rodriguez Muro¹,⁴
- Riccardo Rosati²
- Domenico Fabio Savo²
- Mindaugas Slusnys¹
- Guohui Xiao¹
- Michael Zakharyaschev³

Thanks go also to many developers!

¹ Free University of Bozen-Bolzano
² Sapienza Università di Roma
³ Birkbeck College London
⁴ IBM Research Watson

The development work on -ontop- and the experimentation are currently carried out within the EU Large Scale Integrating Project Optique (Scalable End-user Access to Big Data), grant n. FP7-318338, http://www.optique-project.eu/
References I


Reasoning over extended ER models.


The DL-Lite family and relations.

[Berardi et al., 2005] Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo.

Reasoning on UML class diagrams.

[Borgida et al., 2008] Alexander Borgida, Diego Calvanese, and Mariano Rodríguez-Muro.

Explanation in the DL-Lite family of description logics.
In Proc. of the 7th Int. Conf. on Ontologies, DataBases, and Applications of Semantics (ODBASE), volume 5332 of Lecture Notes in Computer Science, pages 1440–1457. Springer, 2008.
References II

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

[Calì et al., 2009b] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz.
A general Datalog-based framework for tractable query answering over ontologies.
In Proc. of the 28th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database

[Calì et al., 2012] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz.
A general Datalog-based framework for tractable query answering over ontologies.

[Calvanese et al., 1998] Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi.
Description logics for conceptual data modeling.
In Jan Chomicki and Günter Saake, editors, Logics for Databases and Information Systems,

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


Linking data to ontologies: The description logic *DL-Lite*$_\alpha$.


[Calvanese et al., 2006b] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati.

Data complexity of query answering in description logics.

References IV

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

Can OWL model football leagues?

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

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

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

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

Data integration through $DL$-Lite$A$ ontologies.
In Klaus-Dieter Schewe and Bernhard Thalheim, editors, Revised Selected Papers of the 3rd Int. Workshop on Semantics in Data and Knowledge Bases (SDKB 2008), volume 4925 of Lecture Notes in Computer Science, pages 26–47. Springer, 2008.

[Calvanese et al., 2008b] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati.
View-based query answering over description logic ontologies.

[Calvanese et al., 2009a] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodríguez-Muro, and Riccardo Rosati.
Ontologies and databases: The $DL$-Lite approach.
[Calvanese et al., 2009b] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati.

Conceptual modeling for data integration.


[Calvanese et al., 2010] Diego Calvanese, Evgeny Kharlamov, Werner Nutt, and Dmitriy Zheleznyakov.

Updating ABoxes in DL-Lite.


The Mastro system for ontology-based data access.

References VII


Optique: OBDA solution for big data.


Data complexity of query answering in description logics.


A scalable benchmark for OBDA systems: Preliminary report.

References VIII

Towards higher-order DL-Lite.

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

Rewriting ontological queries into small nonrecursive Datalog programs.

The price of query rewriting in ontology-based data access.
References IX


[Kikot et al., 2012] Stanislav Kikot, Roman Kontchakov, and Michael Zakharyaschev. Conjunctive query answering with OWL 2 QL.

[Kontchakov et al., 2008] Roman Kontchakov, Frank Wolter, and Michael Zakharyaschev. Can you tell the difference between DL-Lite ontologies?

References X


The combined approach to query answering in DL-Lite.


Answering SPARQL queries over databases under OWL 2 QL entailment regime.


[Lanti et al., 2015] Davide Lanti, Martin Rezk, Guohui Xiao, and Diego Calvanese.

The NPD benchmark: Reality check for OBDA systems.

In Proc. of the 18th Int. Conf. on Extending Database Technology (EDBT), 2015.
References XI

OWL Web Ontology Language profiles.
Available at http://www.w3.org/TR/owl-profiles/.

[Pérez-Urbina et al., 2010] Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks.
Tractable query answering and rewriting under description logic constraints.

Linking data to ontologies.

[Poggi et al., 2008b] Antonella Poggi, Mariano Rodríguez-Muro, and Marco Ruzzi.
Ontology-based database access with DIG-Mastro and the OBDA Plugin for Protégé.
In Kendall Clark and Peter F. Patel-Schneider, editors, Proc. of the 4th Int. Workshop on OWL: Experiences and Directions (OWLED DC), 2008.
References XII


Finite model reasoning in DL-Lite.

[Savo et al., 2010] Domenico Fabio Savo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodríguez-Muro, Vittorio Romagnoli, Marco Ruzzi, and Gabriele Stella.
Mastro at work: Experiences on ontology-based data access.

[Zheleznyakov et al., 2010] Dmitriy Zheleznyakov, Diego Calvanese, Evgeny Kharlamov, and Werner Nutt.
Updating TBoxes in DL-Lite.