Managing data through the lens of an ontology

Maurizio Lenzerini
Dipartimento di Ingegneria Informatica
Automatica e Gestionale Antonio Ruberti

3rd Int. Workshop on Big Data and Computational Intelligence
Beijing, China, July 29 – 31, 2016
Information system architecture enabled by DBMS

Pre-DBMS architecture (need of a unified data storage):

Application

Data sources

“Ideal information system architecture” with DBMS (‘70s):

Application

Database
Today in many organizations ...

- Distributed, redundant, application-dependent, and mutually incoherent data
- Desperate need of a coherent, conceptual, unified view of data

Maurizio Lenzerini
Ontology-based Data Management
BDCI 2016 (3/76)
... even with just one data source

Fragment of a relational table in a Bank Information system:

<table>
<thead>
<tr>
<th>CUC</th>
<th>TS_START</th>
<th>TS_END</th>
<th>ID_GRUP</th>
<th>FLAG_CP</th>
<th>FLAG_CF</th>
<th>FATTURATO</th>
<th>FLAG_FATT</th>
</tr>
</thead>
<tbody>
<tr>
<td>124589</td>
<td>30-lug-2004</td>
<td>1-gen-9999</td>
<td>92736</td>
<td>S</td>
<td>N</td>
<td>195000,00</td>
<td>N</td>
</tr>
<tr>
<td>140904</td>
<td>15-mag-2001</td>
<td>15-giu-2005</td>
<td>35060</td>
<td>N</td>
<td>N</td>
<td>230600,00</td>
<td>N</td>
</tr>
<tr>
<td>124589</td>
<td>5-mag-2001</td>
<td>30-lug-2004</td>
<td>92736</td>
<td>N</td>
<td>S</td>
<td>195000,00</td>
<td>S</td>
</tr>
<tr>
<td>-452901</td>
<td>13-mag-2001</td>
<td>27-lug-2004</td>
<td>92770</td>
<td>S</td>
<td>N</td>
<td>392000,00</td>
<td>N</td>
</tr>
<tr>
<td>129008</td>
<td>10-mag-2001</td>
<td>1-gen-9999</td>
<td>62010</td>
<td>N</td>
<td>S</td>
<td>247000,00</td>
<td>S</td>
</tr>
<tr>
<td>-472900</td>
<td>10-mag-2001</td>
<td>1-gen-9999</td>
<td>62010</td>
<td>S</td>
<td>N</td>
<td>0 00</td>
<td>N</td>
</tr>
<tr>
<td>130976</td>
<td>7-mag-2001</td>
<td>9-lug-2003</td>
<td>75680</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
... even with just one data source

Negative value denotes a holding

<table>
<thead>
<tr>
<th>CUC</th>
<th>TS_START</th>
<th>TS_END</th>
<th>ID_GRUP</th>
<th>FLAG_CP</th>
<th>FLAG_CF</th>
<th>FATTURATO</th>
<th>FLAG_FATT</th>
</tr>
</thead>
<tbody>
<tr>
<td>124589</td>
<td>30-lug-2004</td>
<td>1-gen-9999</td>
<td>92736</td>
<td>S</td>
<td>N</td>
<td>195000,00</td>
<td>N</td>
</tr>
<tr>
<td>140904</td>
<td>15-mag-2001</td>
<td>15-giu-2005</td>
<td>35060</td>
<td>N</td>
<td>N</td>
<td>230600,00</td>
<td>N</td>
</tr>
<tr>
<td>124589</td>
<td>5-mag-2001</td>
<td>30-lug-2004</td>
<td>92736</td>
<td>N</td>
<td>S</td>
<td>195000,00</td>
<td>S</td>
</tr>
<tr>
<td>-452901</td>
<td>13-mag-2001</td>
<td>27-lug-2004</td>
<td>92770</td>
<td>S</td>
<td>N</td>
<td>392000,00</td>
<td>N</td>
</tr>
<tr>
<td>129008</td>
<td>10-mag-2001</td>
<td>1-gen-9999</td>
<td>62010</td>
<td>N</td>
<td>S</td>
<td>247000,00</td>
<td>S</td>
</tr>
<tr>
<td>-472900</td>
<td>10-mag-2001</td>
<td>1-gen-9999</td>
<td>62010</td>
<td>S</td>
<td>N</td>
<td>0 00</td>
<td>N</td>
</tr>
<tr>
<td>130976</td>
<td>7-mag-2001</td>
<td>9-lug-2003</td>
<td>75680</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
... even with just one data source

<table>
<thead>
<tr>
<th>CUC</th>
<th>TS_START</th>
<th>TS_END</th>
<th>ID_GRUP</th>
<th>FLAG_CP</th>
<th>FLAG_CF</th>
<th>FATTURATO</th>
<th>FLAG_FATT</th>
</tr>
</thead>
<tbody>
<tr>
<td>124589</td>
<td>30-lug-2004</td>
<td>1-gen-9999</td>
<td>92736</td>
<td>S</td>
<td>N</td>
<td>195000,00</td>
<td>N</td>
</tr>
<tr>
<td>140904</td>
<td>15-mag-2001</td>
<td>15-giu-2005</td>
<td>35060</td>
<td>N</td>
<td>N</td>
<td>230600,00</td>
<td>N</td>
</tr>
<tr>
<td>124589</td>
<td>5-mag-2001</td>
<td>30-lug-2004</td>
<td>92736</td>
<td>N</td>
<td>S</td>
<td>195000,00</td>
<td>S</td>
</tr>
<tr>
<td>-452901</td>
<td>13-mag-2001</td>
<td>27-lug-2004</td>
<td>92770</td>
<td>S</td>
<td>N</td>
<td>392000,00</td>
<td>N</td>
</tr>
<tr>
<td>129008</td>
<td>10-mag-2001</td>
<td>1-gen-9999</td>
<td>62010</td>
<td>N</td>
<td>S</td>
<td>247000,00</td>
<td>S</td>
</tr>
<tr>
<td>-472900</td>
<td>10-mag-2001</td>
<td>1-gen-9999</td>
<td>62010</td>
<td>S</td>
<td>N</td>
<td>0 00</td>
<td>N</td>
</tr>
<tr>
<td>130976</td>
<td>7-mag-2001</td>
<td>9-lug-2003</td>
<td>75680</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*S means that the customer is the leader of the group it belongs to*

*S means that the customer is the head of the group it belongs to*
... even with just one data source

\[ N \text{ means that the FATTURATO field is not valid} \]

<table>
<thead>
<tr>
<th>CUC</th>
<th>TS_START</th>
<th>TS_END</th>
<th>ID_GRUP</th>
<th>FLAG_CP</th>
<th>FLAG_CF</th>
<th>FATTURATO</th>
<th>FLAG_FATT</th>
</tr>
</thead>
<tbody>
<tr>
<td>124589</td>
<td>30-lug-2004</td>
<td>1-gen-9999</td>
<td>92736</td>
<td>S</td>
<td>N</td>
<td>195000,00</td>
<td>N</td>
</tr>
<tr>
<td>140904</td>
<td>15-mag-2001</td>
<td>15-giu-2005</td>
<td>35060</td>
<td>N</td>
<td>N</td>
<td>230600,00</td>
<td>N</td>
</tr>
<tr>
<td>124589</td>
<td>5-mag-2001</td>
<td>30-lug-2004</td>
<td>92736</td>
<td>N</td>
<td>S</td>
<td>195000,00</td>
<td>S</td>
</tr>
<tr>
<td>-452901</td>
<td>13-mag-2001</td>
<td>27-lug-2004</td>
<td>92770</td>
<td>S</td>
<td>N</td>
<td>392000,00</td>
<td>N</td>
</tr>
<tr>
<td>129008</td>
<td>10-mag-2001</td>
<td>1-gen-9999</td>
<td>62010</td>
<td>N</td>
<td>S</td>
<td>247000,00</td>
<td>S</td>
</tr>
<tr>
<td>-472900</td>
<td>10-mag-2001</td>
<td>1-gen-9999</td>
<td>62010</td>
<td>S</td>
<td>N</td>
<td>00</td>
<td>N</td>
</tr>
<tr>
<td>130976</td>
<td>7-mag-2001</td>
<td>9-lug-2003</td>
<td>75680</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Large enterprises spend a great deal of time and money on data preparation and information integration (~40% of information-technology shops’ budget).

Market for information integration software estimated to grow to $3.4 billion by 2019 [Gartner, 2015]

Data integration is a large and growing part of software development, computer science, and specific applications settings, such as scientific computing, semantic web, etc..

Data preparation and integration is also crucial for “big data” processing (to make sense of big data!)

Basing the integrated view of data on a clean, rich and abstract conceptual representation of the data has always been both a goal and a challenge [Mylopoulos et al 1984]
Ontology-based Data Management is a new paradigm, rooted on the idea of using Database Theory fundamentals and Knowledge Representation and Reasoning techniques for a new way of managing data, and characterized by the following principles:

- Data may reside where they are (no need to move data)
- Build a conceptual specification of the domain of interest, in terms of knowledge structures
- Map such knowledge structures to concrete data sources
- Express all services over the knowledge structures
- Automatically translate knowledge services to data services
Based on three main components:

- **Ontology**, a declarative, logic-based specification of the domain of interest, used as a unified, conceptual view for clients.

- **Data sources**, representing external, independent, heterogeneous, storage (or, more generally, computational) structures.

- **Mappings**, used to semantically link data at the sources to the ontology.
Outline

1. Ontology-based data management: The framework
2. Query answering
3. Inconsistency tolerance
4. Metamodeling and metaquerying
5. Conclusion
Outline

1. Ontology-based data management: The framework
2. Query answering
3. Inconsistency tolerance
4. Metamodeling and metaquerying
5. Conclusion
An ontology-based data management system is a triple $\langle \mathcal{O}, S, M \rangle$, where

- $\mathcal{O}$ is the ontology, expressed as a logical theory (TBox in a Description Logic)
- $S$ is a relational database the data sources (note that federation tools are able to present a set of heterogeneous data sources as a single relational database)
- $M$ is a set of mapping assertions, each one of the form

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

where

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

Let $\mathcal{I} = (\Delta, \cdot)$ be an interpretation for the ontology $\mathcal{O}$, where $\Delta$ is the domain and $\cdot$ is the interpretation function.

**Def.: Semantics**

$I = (\Delta, \cdot)$ is a model of $\langle O, S, M \rangle$ if:
- $I$ is a model of $O$;
- $I$ satisfies $M$ wrt $S$, i.e., satisfies every assertion in $M$ wrt $S$.

**Def.: Mapping satisfaction (sound mappings)**

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

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

is true in $I \cup S$.

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

$$\text{cert}(q, \mathcal{K}) = \{ \vec{c} \mid \vec{c}^\mathcal{I} \in q^\mathcal{I} \text{ for every model } \mathcal{I} \text{ of } \mathcal{K} \}$$
Which languages?

- Which language for expressing the ontology?
  - We use Description Logics (OWL), but which one?

- Which language for expressing the mappings?
  - We should deal with the impedance mismatch problem

- Which language for expressing queries over the ontology?

**Challenge:** optimal compromise between expressive power and data complexity.
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.

〜 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!*
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 can be created from her $SSN$: $\text{pers}(SSN)$
- A project can be created from its $Name$: $\text{proj}(PrName)$

$\text{pers}$ and $\text{proj}$ are Skolem functions.

If $VRD56B25$ is a $SSN$, then $\text{pers}(VRD56B25)$ is an object term denoting a person.
Actual data is stored in a DB:

\[ D_1[SSN: String, PrName: String] \]
Employees and Projects they work for

\[ D_2[Code: String, Salary: Int] \]
Employee's Code with salary

\[ D_3[Code: String, SSN: String] \]
Employee's Code with SSN

...
Ontology with mappings – Example

TBox $\mathcal{T}$ (UML)

![TBox diagram showing relationships between Employee and Project]

federated schema of the DB $S$

- $D_1[\text{SSN: String, PrName: String}]$
  Employees and Projects they work for
- $D_2[\text{Code: String, Salary: Int}]$
  Employee’s Code with salary
- $D_3[\text{Code: String, SSN: String}]$
  Employee’s Code with SSN

Mapping $\mathcal{M}$

$M_1$: SELECT SSN, PrName
FROM $D_1$

$\Rightarrow$ Employee($\text{pers(SSN)}$),
Project($\text{proj(PrName)}$),
projectName($\text{proj(PrName), PrName}$),
workFor($\text{pers(SSN), proj(PrName)}$)

$M_2$: SELECT SSN, Salary
FROM $D_2$, $D_3$

$\Rightarrow$ Employee($\text{pers(SSN)}$),
salary($\text{pers(SSN), Salary}$)
Ontology-based data management (OBDM): topics

- *Ontology-based [ data access | query answering ]* (OBDA | OBQA)
  - query answering under classical semantics
  - inconsistency tolerant query answering
  - meta-querying
- *Ontology-based data quality (OBDQ)*
- *Ontology-based data governance (OBDG)*
- *Ontology-based data restructuring (OBDR)*
- *Ontology-based business intelligence (OBBI)*
- *Ontology-based data exchange and coordination (OBDE)*
- *Ontology-based data update (OBDU)*
- *Ontology-based service and process management (OBDS)*

General requirements:
- large data collections
- efficiency with respect to size of data (data complexity)
Outline

1. Ontology-based data management: The framework
2. Query answering
3. Inconsistency tolerance
4. Metamodeling and metaquerying
5. Conclusion
In this talk, we mostly abstract from the mapping: we assume that $\mathcal{M}$ is a GAV mapping, and we denote by $\mathcal{M}(S)$ the database obtained by “transferring” the data from the sources to the alphabet of the ontology.

$\mathcal{M}(S)$ can be seen as a set of facts built on the alphabet of $\mathcal{O}$ (i.e., a set of ground atomic formulas in logic, or simply, an ABox, in DL terminology). In other words, formally, we can consider our system as constituted by the pair $\langle \mathcal{O}, \mathcal{A} \rangle$, where $\mathcal{A}$ is an ABox.

In principle, to obtain a query over $S$ from a query over $\mathcal{M}(S)$, we can unfold the query based on $\mathcal{M}$. 
Mostly, we consider conjunctive queries (CQ), i.e., queries of the form (Datalog notation)

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

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

- CQs contain no disjunction, no negation, no universal quantification
- Correspond to SQL/relational algebra select-project-join (SPJ) queries – the most frequently asked queries
- They can also be written as SPARQL queries
- A Union of CQs (UCQ) is a set of CQs with the same head predicate
Example of query

\[ q(x) \leftarrow \text{supervisedBy}(x, y), \text{ComputerScientist}(y), \text{hates}(y, z), \text{ComputerEngineering}(z) \]
Is ontology-based query answering essentially the same problem as query answering in databases?
ComputerProfessor is partitioned into ComputerScientist and ComputerEngineer. (*) [Andrea Schaerf 1993]
To determine this answer, we need to resort to reasoning by cases on the instances.
To determine this answer, we need to resort to reasoning by cases on the instances.
Answering FOL queries is **undecidable**, even if the ontology is empty, and the set of mappings is empty.

**Unions of conjunctive queries** (UCQs) do not suffer from this problem.

We can go beyond unions of conjunctive queries without falling into undecidability, but we get intractability in data complexity very soon.
## Complexity of conjunctive query answering in DLs

<table>
<thead>
<tr>
<th></th>
<th>Combined complexity</th>
<th>Data complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain databases</td>
<td>NP-complete</td>
<td>in \textit{LogSpace} \textsuperscript{(1)}</td>
</tr>
<tr>
<td>OWL 2</td>
<td>?</td>
<td>\textit{coNP-hard} \textsuperscript{(2)}</td>
</tr>
</tbody>
</table>

\textsuperscript{(1)} Going beyond probably means not scaling with the data.

\textsuperscript{(2)} Already for a TBox with a single disjunction (see example above).

### Questions

- Can we find interesting DLs for which the query answering problem can be solved efficiently (in \textit{LogSpace} wrt data complexity)?
- If yes, can we leverage relational database technology for query answering in OBDM?
### Complexity of conjunctive query answering in DLs

<table>
<thead>
<tr>
<th></th>
<th>Combined complexity</th>
<th>Data complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain databases</td>
<td>NP-complete</td>
<td>in LogSpace</td>
</tr>
<tr>
<td>OWL 2</td>
<td>?</td>
<td>coNP-hard</td>
</tr>
</tbody>
</table>

(1) Going beyond probably means not scaling with the data.

(2) Already for a TBox with a single disjunction (see example above).

### Questions

- Can we find interesting DLs for which the query answering problem can be solved efficiently (in LogSpace wrt data complexity)?
- If yes, can we leverage relational database technology for query answering in OBDM?
A very popular logic: $DL$-$Lite_{A,id}$

$DL$-$Lite_{A,id}$ is the most expressive logic in the $DL$-$Lite$ family

Expressions in $DL$-$Lite_{A,id}$:

\[
\begin{align*}
B & \rightarrow A \mid \exists Q \mid \delta(U) \\
Q & \rightarrow P \mid P^- \\
T & \rightarrow \top_D \mid T_1 \mid \cdots \mid T_n
\end{align*}
\]

\[
\begin{align*}
E & \rightarrow \rho(U) \\
V & \rightarrow U \mid \neg U \\
R & \rightarrow Q \mid \neg Q
\end{align*}
\]

 Assertions in $DL$-$Lite_{A,id}$:

\[
\begin{align*}
B & \sqsubseteq C \quad (\text{concept inclusion}) & E & \sqsubseteq T \quad (\text{value-domain inclusion}) \\
Q & \sqsubseteq R \quad (\text{role inclusion}) & U & \sqsubseteq V \quad (\text{attribute inclusion}) \\
(idB \; \pi_1, \ldots, \pi_n) & \quad (\text{identification assertions}) & \text{funt} \; Q & \quad (\text{role functionality}) \\
\text{funt} \; U & \quad (\text{attribute functionality})
\end{align*}
\]

In identification and functional assertions, roles and attributes cannot specialized, and each $\pi_i$ denotes a path (with at least one path with length 1), which is an expression built according to the following syntax rule:

\[
\pi \rightarrow S \mid B? \mid \pi_1 \circ \pi_2
\]
Semantics of $DL$-$Lite^{A,id}_A$ (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 $\text{DL-Lite}_{A,id}$

<table>
<thead>
<tr>
<th>Concept</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>Domain and range of properties</td>
<td>$\exists P \sqsubseteq A_1 \quad \exists P^- \sqsubseteq A_2$</td>
</tr>
<tr>
<td>Mandatory participation ($\text{min card} = 1$)</td>
<td>$A_1 \sqsubseteq \exists P \quad A_2 \sqsubseteq \exists P^-$</td>
</tr>
<tr>
<td>Functionality of relations ($\text{max card} = 1$)</td>
<td>$(\text{funct } P) \quad (\text{funct } P^-)$</td>
</tr>
<tr>
<td>ISA between properties</td>
<td>$Q_1 \sqsubseteq Q_2$</td>
</tr>
<tr>
<td>Disjointness between properties</td>
<td>$Q_1 \sqsubseteq \neg Q_2$</td>
</tr>
</tbody>
</table>

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

**Note 2:** $\text{DL-Lite}_{A,id}$ can be extended to capture also min cardinality constraints ($A \sqsubseteq \leq n Q$), max cardinality constraints ($A \sqsubseteq \geq n Q$) [Artale et al, JAIR 2009], $n$-ary relations, and denial assertions (not considered here for simplicity).
Example of $DL$-Lite$^{A,id}$ ontology

- Faculty
- Professor
- AssocProf
- Dean

- Professor ⊑ Faculty
- AssocProf ⊑ Professor
- Dean ⊑ Professor
- AssocProf ⊑ ¬Dean

- College ⊑ Faculty
- Faculty ⊑ worksFor
- worksFor ⊑ College

- Dean ⊑ College
- Dean ⊑ isHeadOf
- College ⊑ isHeadOf

- isHeadOf ⊑ worksFor

- name: String
- age: Integer

- worksFor (funct age)

- isHeadOf (funct isHeadOf)
- isHeadOf (funct isHeadOf¬)
Possible approaches:

- the chase (used in database theory for reasoning about data dependencies [Maier 1983], and in data exchange for computing universal solutions [Fagin 2013])
- resolution-based methods
- ...

None of the existing approaches directly works for our purpose.

→ So, we designed our own algorithm, called PerfectRef, implemented in our OBDM tool, MASTRO
Remark

We call positive inclusions (PIs) assertions of the form

\[ B_1 \sqsubseteq B_2, \quad Q_1 \sqsubseteq Q_2 \]

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

\[ B_1 \sqsubseteq \neg B_2, \quad Q_1 \sqsubseteq \neg Q_2 \]

Theorem

Let \( q \) be a boolean UCQs and \( \mathcal{O} = \mathcal{O}_{PI} \cup \mathcal{O}_{NI} \cup \mathcal{O}_{id} \) be a TBox s.t.

- \( \mathcal{O}_{PI} \) is a set of PIs
- \( \mathcal{O}_{NI} \) is a set of NIs
- \( \mathcal{O}_{id} \) is a set of identification assertions.

For each \( S \) such that \( \langle \mathcal{O}, S, \mathcal{M} \rangle \) is satisfiable, we have that

\[ \langle \mathcal{O}, S, \mathcal{M} \rangle \models q \iff \langle \mathcal{O}_{PI}, S, \mathcal{M} \rangle \models q. \]

In other words, we have that \( \text{cert}(q, \langle \mathcal{O}, S, \mathcal{M} \rangle) = \text{cert}(q, \langle \mathcal{O}_{PI}, S, \mathcal{M} \rangle) \).
Query answering in $DL{-}Lite_{A,id}$: Query rewriting (cont’d)

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

\[
q(x) \leftarrow \text{Professor}(x)
\]

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

as a logic rule:

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

**Basic rewriting step:**

- *when* the atom unifies with the **head** of the rule (with mgu $\sigma$).
- *substitute* the atom with the **body** of the rule (to which $\sigma$ is applied).

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

\[
q(x) \leftarrow \text{AssocProfessor}(x)
\]

We say that the PI $\text{AssocProfessor} \sqsubseteq \text{Professor}$ applies to the atom $\text{Professor}(x)$. 
Consider now the query

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

as a logic rule:

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

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

\[ q(x) \leftarrow \text{Professor}(x) \]
Conversely, for the following query with join variables

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

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

as a logic rule:

\[
\text{teaches}(z_1, z_2) \leftarrow \text{Professor}(z_1)
\]

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

Conversely, the PI

\[
\exists \text{teaches}^- \sqsubseteq \text{Course}
\]

as a logic rule:

\[
\text{Course}(z_2) \leftarrow \text{teaches}(z_1, z_2)
\]

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

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

\[
q(x) \leftarrow \text{teaches}(x, y), \text{teaches}(z_1, y)
\]
We now have the query

\[ q(x) \leftarrow teaches(x, y), teaches(z, y) \]

The PI

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

as a logic rule:

\[ teaches(z_1, z_2) \leftarrow \text{Professor}(z_1) \]

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

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

\[ q(x) \leftarrow teaches(x, y) \]

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

\[ q(x) \leftarrow \text{Professor}(x) \]
Algorithm PerfectRef($q$, $\mathcal{O}_P$)

Input: conjunctive query $q$, set of $DL-Lite_{A,id}$ PLs $\mathcal{O}_P$

Output: union of conjunctive queries $PR$

$PR := \{ q \}$;

repeat

$PR' := PR$;

for each $q \in PR'$ do

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

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

if $I$ is applicable to $g$

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

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

if $g_1$ and $g_2$ unify

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

until $PR' = PR$;

return $PR$
Answering by rewriting in $DL$-$Lite_{A,id}$: The algorithm

1. Rewrite the CQ $q$ into a UCQs: apply to $q$ in all possible ways the PIs in the TBox $\mathcal{O}$.
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.

**Theorem (Calvanese et al, JAR 2007)**

The query resulting from the above process is a UCQ, and is the perfect rewriting $r_{q,\mathcal{O}}$, i.e., evaluating $r_{q,\mathcal{O}}$ over $\mathcal{M}(\mathcal{S})$ computes the certain answers to $q$ wrt $\langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$.

Note that the same algorithm can be used to check satisfiability of $\langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$. 
Query answering in $DL-Lite_{A,id}$: Example

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

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

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

$M(S)$: teaches(John, databases)  
Professor(Mary)

It is easy to see that the evaluation of $r_{q,\emptyset}$ over $M(S)$ in this case produces the set $\{John, Mary\}$. 
Complexity

\( n \) : query size
\( m \) : number of predicate symbols in \( \mathcal{O} \) or query \( q \)

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

Query answering for CQs and UCQs is:
- \textbf{PTime} in the size of \( \mathcal{TBox} \).
- \textbf{AC}^0 in the size of the \( \mathcal{M}(S) \).
- Exponential in the size of the \textit{query}.

Can we go beyond \textit{DL-Lite}_{A,id} and remain in \textbf{AC}^0?

By adding essentially any other DL construct (without limitations) we lose these computational properties.
Beyond $\text{DL-Lite}_{A,\text{id}}$: results on data complexity

<table>
<thead>
<tr>
<th></th>
<th>Lhs</th>
<th>Rhs</th>
<th>Funct.</th>
<th>Prop. incl.</th>
<th>Data complexity of query answering</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\text{DL-Lite}_{A,\text{id}}$</td>
<td>$A$</td>
<td>$-$</td>
<td>$\checkmark$</td>
<td>in $\text{AC}^0$</td>
</tr>
<tr>
<td>1</td>
<td>$A \mid \exists P . A$</td>
<td>$A$</td>
<td>$-$</td>
<td>$-$</td>
<td>$\text{NLogSpace-hard}$</td>
</tr>
<tr>
<td>2</td>
<td>$A$</td>
<td>$A \mid \forall P . A$</td>
<td>$-$</td>
<td>$-$</td>
<td>$\text{NLogSpace-hard}$</td>
</tr>
<tr>
<td>3</td>
<td>$A$</td>
<td>$A \mid \exists P . A$</td>
<td>$\checkmark$</td>
<td>$-$</td>
<td>$\text{NLogSpace-hard}$</td>
</tr>
<tr>
<td>4</td>
<td>$A \mid \exists P . A \mid A_1 \cap A_2$</td>
<td>$A$</td>
<td>$-$</td>
<td>$-$</td>
<td>$\text{PTime-hard}$</td>
</tr>
<tr>
<td>5</td>
<td>$A \mid A_1 \cap A_2$</td>
<td>$A \mid \forall P . A$</td>
<td>$-$</td>
<td>$-$</td>
<td>$\text{PTime-hard}$</td>
</tr>
<tr>
<td>6</td>
<td>$A \mid A_1 \cap A_2$</td>
<td>$A \mid \exists P . A$</td>
<td>$\checkmark$</td>
<td>$-$</td>
<td>$\text{PTime-hard}$</td>
</tr>
<tr>
<td>7</td>
<td>$A \mid \exists P . A \mid \exists P^- . A$</td>
<td>$A \mid \exists P$</td>
<td>$-$</td>
<td>$-$</td>
<td>$\text{PTime-hard}$</td>
</tr>
<tr>
<td>8</td>
<td>$A \mid \exists P \mid \exists P^-$. A</td>
<td>$A \mid \exists P \mid \exists P^-$</td>
<td>$\checkmark$</td>
<td>$\checkmark$</td>
<td>$\text{PTime-hard}$</td>
</tr>
<tr>
<td>9</td>
<td>$A \mid \neg A$</td>
<td>$A$</td>
<td>$-$</td>
<td>$-$</td>
<td>$\text{coNP-hard}$</td>
</tr>
<tr>
<td>10</td>
<td>$A$</td>
<td>$A \mid A_1 \sqcup A_2$</td>
<td>$-$</td>
<td>$-$</td>
<td>$\text{coNP-hard}$</td>
</tr>
<tr>
<td>11</td>
<td>$A \mid \forall P . A$</td>
<td>$A$</td>
<td>$-$</td>
<td>$-$</td>
<td>$\text{coNP-hard}$</td>
</tr>
</tbody>
</table>

- $\text{DL-Lite}_{A,\text{id}}$ is the most expressive DL of the $\text{DL-Lite}$ family
- $\text{NLogSpace}$ and $\text{PTime}$ hardness holds already for instance checking.
- For $\text{coNP}$-hardness in line 10, a TBox with a single assertion $A_L \sqsubseteq A_T \sqcup A_F$ suffices! $\leadsto$ No hope of including covering constraints.
A portion of an ontology for the Italian Public Debt:
Sources of complexity

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

Two sources of complexity wrt query:

- **conjunctive query evaluation is NP-complete** – complexity comes from the need of matching the query and the data
  \[ \sim \text{unavoidable!} \]

- The rewritten query has exponential size wrt the original query – complexity comes from the need of “expanding” the query w.r.t. the ontology
  \[ \sim \text{avoidable?} \]

**Example**

```
TBox \( \mathcal{T} \):  

\begin{align*}
q(x) & \leftarrow A(x), P(x, y), A(y), P(y, z), A(z) \\
\text{UCQ rewriting of } q \text{ w.r.t. } \mathcal{T} & \text{ contains 729 CQs} \\
i.e., \text{ it is a UNION of 729 SPJ SQL queries} 
\end{align*}
```
Sources of complexity

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

Two sources of complexity wrt query:

- conjunctive query evaluation is **NP-complete** – complexity comes from the need of matching the query and the data
  \( \Rightarrow \) unavoidable!

- the rewritten query has exponential size wrt the original query – complexity comes from the need of “expanding” the query w.r.t. the ontology
  \( \Rightarrow \) avoidable?

---

Example

**Example**

TBox \( \mathcal{T} \):

\[
\begin{align*}
A & \leftarrow B, C \\
B & \leftarrow D, E \\
C & \leftarrow F, G \\
D & \\
E & \\
F & \\
G & \\
H & \\
I & \\
\end{align*}
\]

\[
q(x) \leftarrow A(x), P(x, y), A(y), P(y, z), A(z)
\]

UCQ rewriting of \( q \) w.r.t. \( \mathcal{T} \) contains 729 CQs

i.e., it is a UNION of 729 SPJ SQL queries
Eliminating redundant TBox assertions

TBox optimization is based on a characterization of assertions in a TBox $\mathcal{T}$ that are redundant wrt a set $\Sigma$ of ABox dependencies.

**Example (Direct redundancy)**

Let $\mathcal{T}$ be:

$\exists \text{hasFather} \text{Person}$

Let $\Sigma$ be:

$\exists \text{hasFather} \text{Human}$

Note: $\Sigma$ enforces e.g., that

$\text{hasFather}(\text{luisa, franz}) \in \mathcal{A}$ implies $\text{Human}(\text{luisa}) \in \mathcal{A}$.

Then $\text{Person} \sqsubseteq \text{Human}$ is redundant in $\mathcal{T}$.

The overall characterization of redundant TBox assertions is more involved (see [?]).
Eliminating redundant TBox assertions

TBox optimization is based on a characterization of assertions in a TBox $\mathcal{T}$ that are redundant wrt a set $\Sigma$ of ABox dependencies.

Example (Direct redundancy)

Let $\mathcal{T}$ be:
- $\exists \text{hasFather} \text{Person}$
- $\text{Human}$

Let $\Sigma$ be:
- $\exists \text{hasFather} \text{Person}$
- $\text{Human}$

Note: $\Sigma$ enforces e.g., that $\text{hasFather}(\text{luisa}, \text{franz}) \in \mathcal{A}$ implies $\text{Human}(\text{luisa}) \in \mathcal{A}$.

Then $\text{Person} \sqsubseteq \text{Human}$ is redundant in $\mathcal{T}$.

The overall characterization of redundant TBox assertions is more involved (see [?]).
Computing an optimized TBox

Given a TBox $\mathcal{T}$ and a set $\Sigma$ of ABox dependencies:

1. Compute the deductive closure $\mathcal{T}_{cl}$ of $\mathcal{T}$ (at most quadratic in size of $\mathcal{T}$).
2. Compute the deductive closure $\Sigma_{cl}$ of $\Sigma$ (at most quadratic in size of $\Sigma$).
3. Eliminate from $\mathcal{T}_{cl}$ all TBox assertions redundant wrt $\Sigma_{cl}$, obtaining $\mathcal{T}_{opt}$.

Notes:

- $\mathcal{T}_{opt}$ can be computed in polynomial time in the size of $\mathcal{T}$ and $\Sigma$.
- $\mathcal{T}_{opt}$ might be much smaller than $\mathcal{T}$.

Theorem

For every (virtual) ABox $\mathcal{A}$ satisfying $\Sigma$ and for every UCQ $q$, we have that

$$\text{cert}(q, \langle \mathcal{T}, \mathcal{A} \rangle) = \text{cert}(q, \langle \mathcal{T}_{opt}, \mathcal{A} \rangle).$$

Hence, $\mathcal{T}_{opt}$ can be used instead of $\mathcal{T}$ independently of the adopted query rewriting method (provided the ABox satisfies $\Sigma$).
Computing an optimized TBox

Given a TBox \( \mathcal{T} \) and a set \( \Sigma \) of ABox dependencies:

1. Compute the deductive closure \( \mathcal{T}_{cl} \) of \( \mathcal{T} \) (at most quadratic in size of \( \mathcal{T} \)).
2. Compute the deductive closure \( \Sigma_{cl} \) of \( \Sigma \) (at most quadratic in size of \( \Sigma \)).
3. Eliminate from \( \mathcal{T}_{cl} \) all TBox assertions redundant wrt \( \Sigma_{cl} \), obtaining \( \mathcal{T}_{opt} \).

**Notes:**

- \( \mathcal{T}_{opt} \) can be computed in polynomial time in the size of \( \mathcal{T} \) and \( \Sigma \).
- \( \mathcal{T}_{opt} \) might be much smaller than \( \mathcal{T} \).

**Theorem**

For every (virtual) ABox \( \mathcal{A} \) satisfying \( \Sigma \) and for every UCQ \( q \), we have that

\[
\text{cert}(q, \langle \mathcal{T}, \mathcal{A} \rangle) = \text{cert}(q, \langle \mathcal{T}_{opt}, \mathcal{A} \rangle).
\]

Hence, \( \mathcal{T}_{opt} \) can be used instead of \( \mathcal{T} \) independently of the adopted query rewriting method (provided the ABox satisfies \( \Sigma \)).
Computing an optimized TBox

Given a TBox $\mathcal{T}$ and a set $\Sigma$ of ABox dependencies:

1. Compute the deductive closure $\mathcal{T}_{cl}$ of $\mathcal{T}$ (at most quadratic in size of $\mathcal{T}$).
2. Compute the deductive closure $\Sigma_{cl}$ of $\Sigma$ (at most quadratic in size of $\Sigma$).
3. Eliminate from $\mathcal{T}_{cl}$ all TBox assertions redundant wrt $\Sigma_{cl}$, obtaining $\mathcal{T}_{opt}$.

**Notes:**

- $\mathcal{T}_{opt}$ can be computed in polynomial time in the size of $\mathcal{T}$ and $\Sigma$.
- $\mathcal{T}_{opt}$ might be much smaller than $\mathcal{T}$.

**Theorem**

For every (virtual) ABox $\mathcal{A}$ satisfying $\Sigma$ and for every UCQ $q$, we have that

$$\text{cert}(q, \langle \mathcal{T}, \mathcal{A} \rangle) = \text{cert}(q, \langle \mathcal{T}_{opt}, \mathcal{A} \rangle).$$

Hence, $\mathcal{T}_{opt}$ can be used instead of $\mathcal{T}$ independently of the adopted query rewriting method (provided the ABox satisfies $\Sigma$).
Outline

1. Ontology-based data management: The framework
2. Query answering
3. Inconsistency tolerance
4. Metamodeling and metaquerying
5. Conclusion
Up to now, we have implicitly assumed to deal with satisfiable OBDM systems, but in practice the OBDM system can be unsatisfiable.

**Problem**

Query answering based on classical logic becomes meaningless in the presence of inconsistency (*ex falso quodlibet*).
Example: an inconsistent *DL-Lite* ontology

\[ O \]

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RedWine ⊑ Wine</td>
<td>WhiteWine ⊑ Wine</td>
</tr>
<tr>
<td>RedWine ⊑ ¬ WhiteWine</td>
<td>Wine ⊑ ¬ Beer</td>
</tr>
<tr>
<td>Wine ⊑ ∃ producedBy</td>
<td>∃ producedBy ⊑ Wine</td>
</tr>
<tr>
<td>Wine ⊑ ¬ Winery</td>
<td>Beer ⊑ ¬ Winery</td>
</tr>
<tr>
<td>∃ producedBy ⊑ Winery</td>
<td>(funct producedBy)</td>
</tr>
</tbody>
</table>

\[ M \]

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1(x,y,‘white’) ↦ WhiteWine(x)</td>
<td>R1(x,y,‘red’) ↦ RedWine(x)</td>
</tr>
<tr>
<td>R2(x,y) ↦ Beer(x)</td>
<td>R1(x,y,z) ∨ R2(x,y) ↦ producedBy(x,y)</td>
</tr>
</tbody>
</table>

\[ S \]

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1(grechetto,p1,‘white’)</td>
<td>R1(grechetto,p1,‘red’)</td>
</tr>
<tr>
<td>R2(guinness,p2)</td>
<td>R1(falanghina,p1,‘white’)</td>
</tr>
</tbody>
</table>
Inconsistent-tolerant semantics

**Problem**
To handle classically-inconsistent OBDM systems in a more meaningful way, one needs to change the semantics.

The semantics proposed in [Lembo et al, RR 2010] for inconsistent OBDM systems is based on the following principles:

- We assume that $O$ and $M$ are always consistent (this is true if $O$ is expressed in $DL-Lite_{A,id}$), so that inconsistencies are caused by the interaction between the data at $S$ and the other components of the system, i.e., between $M(S)$ and $O$.

- We resort to the notion of repair [Arenas et al, PODS 1999]. Intuitively, a repair for $⟨O, S, M⟩$ is an ontology $⟨O, A⟩$ that is consistent, and “minimally” differs from $⟨O, S, M⟩$.

What does it mean for $A$ to be “minimally different” from $\langle O, S, M \rangle$? We base this concept on the notion of symmetric difference.

We write $S_1 \oplus S_2$ to denote the symmetric difference between $S_1$ and $S_2$, i.e.,

$$S_1 \oplus S_2 = (S_1 \setminus S_2) \cup (S_2 \setminus S_1)$$

**Definition (Repair)**

Let $\mathcal{K} = \langle O, S, M \rangle$ be an OBDM system. A repair of $\mathcal{K}$ is an ABox $A$ such that:

1. $\text{Mod}(\langle O, A \rangle) \neq \emptyset$,
2. no set of facts $A'$ exists such that
   - $\text{Mod}(\langle O, A' \rangle) \neq \emptyset$,
   - $A' \oplus M(S) \subset A \oplus M(S)$
Example: Repairs

$Rep_1$

\{WhiteWine(grechetto), Beer(guinnes), WhiteWine(falanghina)\}

$Rep_2$

\{RedWine(grechetto), Beer(guinnes), WhiteWine(falanghina)\}

$Rep_3$

\{WhiteWine(grechetto), producedBy(guinnes, p2), WhiteWine(falanghina)\}

$Rep_4$

\{RedWine(grechetto), producedBy(guinnes, p2), WhiteWine(falanghina)\}
Reasoning with all repairs: the AR semantics

Problems:
- Many repairs in general
- What is the complexity of reasoning about all such repairs?

Theorem
Let $\mathcal{K} = \langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$ be an OBDM system, and let $\alpha$ be a ground atom. Deciding whether $\alpha$ is logically implied by every repair of $\mathcal{K}$ is coNP-complete with respect to data complexity.
coNP hardness of the AR-semantics

Ontology $\mathcal{O}$

- $\exists R \sqsubseteq \text{Unsat}$
- $\exists R \sqsubseteq \neg \exists \text{LT}_1$
- $\exists R \sqsubseteq \neg \exists \text{LF}_1$
- $\exists R \sqsubseteq \neg \exists \text{LT}_2$
- $\exists R \sqsubseteq \neg \exists \text{LF}_2$
- $\exists R \sqsubseteq \neg \exists \text{LT}_3$
- $\exists R \sqsubseteq \neg \exists \text{LF}_3$
- $\exists \text{LT}_1 \sqsubseteq \neg \exists \text{LF}_1$
- $\exists \text{LT}_1 \sqsubseteq \neg \exists \text{LF}_2$
- $\exists \text{LT}_1 \sqsubseteq \neg \exists \text{LF}_3$
- $\exists \text{LF}_1 \sqsubseteq \neg \exists \text{LT}_2$
- $\exists \text{LF}_1 \sqsubseteq \neg \exists \text{LT}_3$
- $\exists \text{LT}_2 \sqsubseteq \neg \exists \text{LF}_2$
- $\exists \text{LT}_2 \sqsubseteq \neg \exists \text{LF}_3$
- $\exists \text{LF}_2 \sqsubseteq \neg \exists \text{LT}_3$
- $\exists \text{LT}_3 \sqsubseteq \neg \exists \text{LF}_3$

3-CNFFormula $\phi$: $(a_1 \lor \neg a_2 \lor \neg a_3) \land (\neg a_3 \lor a_4 \lor \neg a_1)$

ABox $\mathcal{A}$ corresponding to $\phi$

\[ \phi \text{ satisfiable iff } \langle \mathcal{O}, \mathcal{A} \rangle \not\models_{AR} \text{Unsat}(a) \]
Other intractability results of the AR semantics, even for simpler languages (e.g., [Bienvenu et al 2012-2015])

Idea: The IAR semantics

We consider the “intersection of all repairs”, and take the set of models of such intersection as the semantics of the system (When in Doubt, Throw It Out).

Note that the IAR semantics is an approximation of the AR semantics
Inconsistent-tolerant query answering

Two possible methods for answering queries posed to $\mathcal{K} = \langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$ according to the inconsistency-tolerant semantics:

- Compute the intersection $\mathcal{A}$ of all repairs of $\mathcal{K}$, and then compute $\bar{t}$ such that $\langle \mathcal{O}, \mathcal{A} \rangle \models q(\bar{t})$
- Rewrite the query $q$ into $q'$ in such a way that, for all $\bar{t}$, we have that $\mathcal{K} \models_{IAR} q(\bar{t})$ is equivalent to $\bar{t} \in q'(\mathcal{M}(\mathcal{S}))$. Then, evaluate $q'$ over $\mathcal{M}(\mathcal{S})$.

We have devised a rewriting technique which encodes a UCQ $q$ into a FOL query $q'$ which, evaluated against the original $\mathcal{M}(\mathcal{S})$ retrieves only the certain answers of $q$ w.r.t the IAR semantics [Lembo et al, JSW 2015].
Rewriting technique

We provide a rewriting technique which encodes a UCQ $Q$ into a FOL query $Q'$ which evaluated against the original $S$ retrieves only the certain answers of $Q$ w.r.t the IR semantics

Rewriting technique

Given a UCQ $Q = q_1 \lor q_2 \lor \ldots \lor q_n$ over $\langle O, S, M \rangle$

- we compute $\text{PerfectRef}_{IAR}(Q, O, M)$ as $\text{MapRewriting}_{M}(\text{IncRewritingUCQ}_{IAR}(\text{PerfectRef}(Q, O), O))$
- we evaluate $\text{PerfectRef}_{IAR}(Q, O, M)$ over $S$

where

- $\text{PerfectRef}(Q, O)$ rewrites $Q$ taking care of $O$
- $\text{IncRewritingUCQ}_{IAR}(Q, O) = \bigvee_{i=1}^{n} \text{IncRewriting}(q_i, O)$ rewrites $Q$ taking care of inconsistencies
- $\text{MapRewriting}_{M}(Q)$ rewrites $Q$ taking care of $M$
Let us consider the CQ

$$q = \exists x. \text{RedWine}(x)$$

We have that $\text{IncRewriting}_{IAR}(q, \mathcal{O})$ is

$$\exists x. \text{RedWine}(x) \land \neg \text{WhiteWine}(x) \land \neg \text{Beer}(x) \land \neg \text{Winery}(x) \land$$

$$\neg(\exists y. \text{producedBy}(x, y) \land x \neq y)$$
**Theorem**

Let $Q$ be a UCQ over $\langle \mathcal{O}, S, M \rangle$. Deciding whether $\vec{t} \in \text{cert}_{IAR}(Q, \langle \mathcal{O}, S, M \rangle)$ is in $AC^0$ in data complexity.

**Complexity**

<table>
<thead>
<tr>
<th>problem</th>
<th>$AR$-semantics</th>
<th>$IAR$-semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance checking</td>
<td>coNP-complete</td>
<td>in $AC_0$</td>
</tr>
<tr>
<td>UCQ answering</td>
<td>coNP-complete</td>
<td>in $AC_0$</td>
</tr>
</tbody>
</table>
Outline

1. Ontology-based data management: The framework
2. Query answering
3. Inconsistency tolerance
4. Metamodeling and metaquerying
5. Conclusion
Up to now, we have assumed that the TBox and the ABox were first-order.

- **Metamodelling**: specifying
  - *metaclasses* (classes whose instances can be themselves classes), and
  - *metaproperties* (relationships between metaclasses)

- **Metaquerying**: expressing queries with
  - variables both in predicate and object position, and
  - TBox atoms
Enriching the mapping languages: mapping intensional knowledge

Source $S$: 

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Coupé</td>
</tr>
<tr>
<td>T2</td>
<td>SUV</td>
</tr>
<tr>
<td>T3</td>
<td>Sedan</td>
</tr>
<tr>
<td>T4</td>
<td>Estate</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CarCode</th>
<th>CarType</th>
<th>EngineSize</th>
<th>BreakPower</th>
<th>Color</th>
<th>TopSpeed</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB111</td>
<td>T1</td>
<td>2000</td>
<td>200</td>
<td>Silver</td>
<td>260</td>
</tr>
<tr>
<td>AF333</td>
<td>T2</td>
<td>3000</td>
<td>300</td>
<td>Black</td>
<td>200</td>
</tr>
<tr>
<td>BR444</td>
<td>T2</td>
<td>4000</td>
<td>400</td>
<td>Grey</td>
<td>220</td>
</tr>
<tr>
<td>AC222</td>
<td>T4</td>
<td>2000</td>
<td>125</td>
<td>Dark Blue</td>
<td>180</td>
</tr>
<tr>
<td>BN555</td>
<td>T3</td>
<td>1000</td>
<td>75</td>
<td>Light Blue</td>
<td>180</td>
</tr>
<tr>
<td>BP666</td>
<td>T1</td>
<td>3000</td>
<td>600</td>
<td>Red</td>
<td>240</td>
</tr>
</tbody>
</table>
Example

Ontology $\mathcal{O}$: Car $\sqsubseteq$ Vehicle

Source $\mathcal{S}$:

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Coupé</td>
</tr>
<tr>
<td>T2</td>
<td>SUV</td>
</tr>
<tr>
<td>T3</td>
<td>Sedan</td>
</tr>
<tr>
<td>T4</td>
<td>Estate</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CarCode</th>
<th>CarType</th>
<th>EngineSize</th>
<th>BreakPower</th>
<th>Color</th>
<th>TopSpeed</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB111</td>
<td>T1</td>
<td>2000</td>
<td>200</td>
<td>Silver</td>
<td>260</td>
</tr>
<tr>
<td>AF333</td>
<td>T2</td>
<td>3000</td>
<td>300</td>
<td>Black</td>
<td>200</td>
</tr>
<tr>
<td>BR444</td>
<td>T2</td>
<td>4000</td>
<td>400</td>
<td>Grey</td>
<td>220</td>
</tr>
<tr>
<td>AC222</td>
<td>T4</td>
<td>2000</td>
<td>125</td>
<td>Dark Blue</td>
<td>180</td>
</tr>
<tr>
<td>BN555</td>
<td>T3</td>
<td>1000</td>
<td>75</td>
<td>Light Blue</td>
<td>180</td>
</tr>
<tr>
<td>BP666</td>
<td>T1</td>
<td>3000</td>
<td>600</td>
<td>Red</td>
<td>240</td>
</tr>
</tbody>
</table>

Mapping $\mathcal{M}$:

- $\{ y \mid T\text{-CarTypes}(x, y) \} \leadsto TypeOfCar(x)$
- $\{ y \mid T\text{-CarTypes}(x, y) \} \leadsto y \sqsubseteq Car$
- $\{ (x, v, z) \mid T\text{-Cars}(x, y, t, u, v, q) \land T\text{-CarTypes}(y, z) \} \leadsto z(x)$
- $\{ (x, y) \mid T\text{-CarTypes}(z_1, x) \land T\text{-CarTypes}(z_2, y) \land x \neq y \} \leadsto x \sqsubseteq \neg y$

The ontology $\mathcal{O}$ is enriched through $\mathcal{M}$ and $\mathcal{S}$.
With metaclasses and metaproperties in the ontology, metaqueries become natural, e.g.:

**Example**

Interesting queries that can be posed to $\langle S, M \rangle$ exploit the higher-order nature of the system:

- Return all the instances of Car, each one with its own type:
  
  $$q(x, y) \leftarrow y(x), \text{Car}(x), \text{TypeOfCar}(y)$$

- Return all the concepts of which car $AB111$ is an instance:
  
  $$q(x) \leftarrow x(AB111)$$
Example of metaquerying

Consider querying an ontology about the “pizza” domain, including

**Classes:** margherita, ortolana, vegetarian
**Object properties:** ate, liked, dislike

\[
\{ (x) \mid \text{ate}(x,y), \text{liked}(x,y), \text{margherita}(y) \} \\
\{ (x) \mid \text{ate}(x,y), \text{liked}(x,y), \text{margherita}(y), \text{dislike}(x,\text{margherita}) \} \\
\{ (x,z) \mid \text{ate}(x,y), \text{liked}(x,y), z(y), \text{dislike}(x,z) \} \\
\{ (x,z) \mid \text{ate}(x,y), \text{liked}(x,y), z(y), \text{dislike}(x,z), z \sqsubseteq \text{vegetarian} \} 
\]
Example of metaquerying

Consider querying an ontology about the “pizza” domain, including

Classes: margherita, ortolana, vegetarian
Object properties: ate, liked, dislike

\{ (x) | \text{ate}(x,y), \text{liked}(x,y), \text{margherita}(y) \} \\
\{ (x) | \text{ate}(x,y), \text{liked}(x,y), \text{margherita}(y), \text{dislike}(x,\text{margherita}) \} \\
\{ (x,z) | \text{ate}(x,y), \text{liked}(x,y), z(y), \text{dislike}(x,z) \} \\
\{ (x,z) | \text{ate}(x,y), \text{liked}(x,y), z(y), \text{dislike}(x,z), z \sqsubseteq \text{vegetarian} \}
Example of metaquerying

Consider querying an ontology about the “pizza” domain, including

Classes:       margherita, ortolana, vegeterian
Object properties: ate, liked, dislike

\{ (x) \mid \text{ate}(x,y), \text{liked}(x,y), \text{margherita}(y) \} \\
\{ (x) \mid \text{ate}(x,y), \text{liked}(x,y), \text{margherita}(y), \text{dislike}(x,\text{margherita}) \} \\
\{ (x,z) \mid \text{ate}(x,y), \text{liked}(x,y), z(y), \text{dislike}(x,z) \} \\
\{ (x,z) \mid \text{ate}(x,y), \text{liked}(x,y), z(y), \text{dislike}(x,z), z \sqsubseteq \text{vegeterian} \}
Example of metaquerying

Consider querying an ontology about the “pizza” domain, including

Classes: margherita, ortolana, vegeterian
Object properties: ate, liked, dislike

\{ (x) \mid \text{ate}(x,y), \text{liked}(x,y), \text{margherita}(y) \} \\
\{ (x) \mid \text{ate}(x,y), \text{liked}(x,y), \text{margherita}(y), \text{dislike}(x,\text{margherita}) \} \\
\{ (x,z) \mid \text{ate}(x,y), \text{liked}(x,y), z(y), \text{dislike}(x,z) \} \\
\{ (x,z) \mid \text{ate}(x,y), \text{liked}(x,y), z(y), \text{dislike}(x,z), z \sqsubseteq \text{vegeterian} \}
Example of metaquerying

Consider querying an ontology about the “pizza” domain, including

Classes: margherita, ortolana, vegetarian
Object properties: ate, liked, dislike

\{
    (x) | ate(x, y), liked(x, y), margherita(y)
\}

\{
    (x) | ate(x, y), liked(x, y), margherita(y), dislike(x, margherita)
\}

\{
    (x, z) | ate(x, y), liked(x, y), z(y), dislike(x, z)
\}

\{
    (x, z) | ate(x, y), liked(x, y), z(y), dislike(x, z), z ⊑ vegetarian
\}
Note: differently from similar semantics, an object is not forced to be an individual object, or to have a class or object property extension (e.g., see $\beta$)
The “metagrounding” technique

Let $Q$ be a query over an ontology $O$.

- a metagrounding of $Q$ is a query $Q'$ obtained from $Q$ by substituting the metavariables occurring in $Q$ in class, object property or data property positions with a class, object property and data property expression over $O$, respectively.

  - e.g., if $O_1$ contains the classes $A, B, C$ and the object property $R$, and $Q$ is the query
    \[
    Q_1() \leftarrow A \sqsubseteq \neg x, B(y), R(x, z), z(y)
    \]
    then a metagrounding of $Q$ is the query $Q'$ obtained by applying the substitution \(\{x \leftarrow C, z \leftarrow C\}\), i.e.,
    \[
    Q_1() \leftarrow A \sqsubseteq \neg C, B(y), R(C, C), C(y)
    \]

- Answering $Q$ through metagrounding resorts to compute the union of the answers to all metagroundings of $Q$. 
Does metagrounding work?

Example

- $O_1 : \{ B(F), C(F), A \sqsubseteq \neg C, R(C, A), R(B, C), A(E) \}$
- $Q_1() \leftarrow A \sqsubseteq \neg x, B(y), R(x, z), z(y)$

Although no metagrounding of $Q_1$ is true, one can show that $Q_1$ is indeed true, by partitioning the models of $O$ into

1. those for which $A$ and $B$ are disjoint ($x \leftarrow B, z \leftarrow C, y \leftarrow F$), and
2. those for which $A$ and $B$ are not disjoint ($x \leftarrow C, z \leftarrow A, y \leftarrow F$)

and showing that there exist two different metagroundings, one true in (1) and false in (2), and the other true in (2) and false in (1)

Metagrounding does not suffice

In general, answering metaqueries cannot be done through metagrounding. Note that in the above example, the “culprit” is the uncertainty of the axiom $A \sqsubseteq \neg B$.
An axiom $\alpha$ over the alphabet of $O$ is **certain** if either $O \models \alpha$, or $O \cup \{\alpha\}$ is unsatisfiable.

$O$ is **TBox-complete** if there exists no negative axiom that can be expressed over the alphabet of $O$ that is not certain.

TBox-completeness can be checked in quadratic time w.r.t. the size of the ontology alphabet.

A methodology can be devised to obtain a TBox-complete ontology from an ontology that is not TBox-complete, keeping the same “intended models”.

TBox-complete ontologies are common in practice since every ontology designed following the traditional methodology for designing ER schemas is a TBox-complete ontology.
Answering metaqueries over TBox-complete ontologies through metagrounding

It can be shown that given a TBox-complete ontology $\mathcal{O}$ and a query $Q$, $Q$ can be answered by applying the metagrounding technique, i.e. $Q$ is true if at least one of its metagrounding is true.

**Query answering algorithm**

```plaintext
input ontology $\mathcal{O}$, query $Q$
if there exists a metagrounding $Q'$ such that $\mathcal{O} \models int(Q')$ and $\mathcal{O} \models ext(Q')$, where
  - $int(Q')$ denotes the TBox atoms of $Q'$, and
  - $ext(Q')$ denotes the ABox atoms of $Q'$
then return true
else return false
```

$O \models int(Q')$ and $O \models ext(Q')$ can be checked by using any off-the-shelf OBDM inference and querying systems.
The general case

Let \( U^O \) be the set of negative assertions that can be expressed over the alphabet of \( O \) and are uncertain in \( O \).

**Definition**

- If \( \alpha \in U^O \), then a **violation set** of \( \alpha \) w.r.t. \( O \) is a minimal set of ABox axioms \( V_{\alpha,O} \) over the predicates of \( O \) and a set of individuals not in \( O \), such that \( \alpha \cup V_{\alpha,O} \) is unsatisfiable.
- If \( \sigma \subseteq U^O \), then the **\( \sigma \)-completion** of \( O \), denoted \( O^\sigma \), is the ontology \( O \cup \sigma \cup C^{U^O\setminus \sigma} \), where \( C^{U^O\setminus \sigma} \) is the union of the violation sets of axioms in \( U^O \) that are not in \( \sigma \).

**Note:** Intuitively, \( O^\sigma \) is obtained from \( O \) by adding all axioms in \( \sigma \) and suitable axioms in such a way that all axioms in \( U^O \) but not in \( \sigma \) are violated.

**Note:** \( O^\sigma \) is TBox-complete.
Example

For the following ontology $O_2$:

\{B(F), C(F), A \sqsubseteq \neg C, R(E, E), R(F, F), R(C, A), R(B, C), A(E)\}

We have $U^{O_2} = \{A \sqsubseteq \neg B\}$, and

- for $\sigma_1 = \{A \sqsubseteq \neg B\}$, we have $O^{\sigma_1} = O_2 \cup \{A \sqsubseteq \neg B\}$.
- for $\sigma_2 = \emptyset$, we have $O^{\sigma_2} = O_2 \cup \{A(s), B(s)\}$
Algorithm for answering metaqueries over general ontologies

Query answering algorithm

**input:** ontology $\mathcal{O}$, query $Q$

if there exists $\sigma \subseteq \mathcal{U}^\mathcal{O}$ such that $\mathcal{O}^\sigma \not\models Q$

then return *false*

else return *true*

### Complexity

<table>
<thead>
<tr>
<th></th>
<th>ABox complexity</th>
<th>TBox complexity</th>
<th>Combined complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TBox-complete ontologies</strong></td>
<td>$\text{AC}^0$</td>
<td>$\text{PTIME}$</td>
<td>NP-complete</td>
</tr>
<tr>
<td><strong>General ontologies</strong></td>
<td>$\text{AC}^0$</td>
<td>$\text{coNP}$-complete</td>
<td>$\Pi_2^P$-complete</td>
</tr>
</tbody>
</table>
Outline

1. Ontology-based data management: The framework
2. Query answering
3. Inconsistency tolerance
4. Metamodeling and metaquerying
5. Conclusion
Many challenges for OBDM

- Still a lot to do for **OBDM systems** (ONTOP, MASTRO, etc.)
  - More optimizations in query answering
  - More powerful ontology languages
  - Rewriting wrt mapping (even GAV mapping are problematic)
  - Preferences over repairs
  - Even more powerful metamodeling and metaquerying

- **Ontology-based data quality**
  - Instance level
  - Schema level
  - Explanation and provenance

- **Ontology-based update**
  - Semantics
  - Pushing the updates to the data sources
  - Updates in the presence of inconsistencies

- **Natural language interface** for querying

- **Ontology-based open data publishing**

- Desperate need of effective tools for modeling both the ontology and the mapping, and for supporting their evolution

- Experimenting OBDM in real applications