Ontology-based access to data sources

Maurizio Lenzerini
Sapienza Università di Roma

(joint work with D. Calvanese, G. De Giacomo, D. Lembo, A. Poggi, R. Rosati)

Work carried out in the context of EU FP6 project TONES

AT&T Labs, USA, August 16, 2007
Motivation

Achieve logical transparency in access to data, by

- hiding to the user where and how data are stored
- presenting to the user a conceptual view of the data
- using a semantically rich formalism for the conceptual view

Similar to Data Integration, but with a rich conceptual description as the global view.

Relevant for: Enterprise Application Integration, Data Warehousing, Semantic Web, ...
Ontology-based data access

Ontology as a conceptual view over data sources

Fundamental task: query answering
Challenges

Choose appropriate

- **ontology language**
- **mapping language**
- **query language**
- **semantics**

so as to obtain a system that is

- **correct** (i.e., sound and complete with respect to the semantics)
- **expressive**
- **efficient**
Our solution

- **Ontology language**
  - The *DL-Lite* family of Description Logic

- **Mapping language**
  - Global-As-View (GAV) mappings, with suitable mechanisms for addressing the “impedance mismatch” problem

- **Query language**
  - (Union of) Conjunctive queries

- **Semantics**
  - Based on first-order logic

- **Essentially optimal solution** (in terms of complexity of query answering)
This talk

- First part
  - “Stand-alone” ontology
  - The data layer stores instances of the ontology
  - No need of complex mappings to data sources
- Second part
  - The data layer is independent from the ontology
  - The mapping is now crucial
- Third part
  - Current and future work
We start with an alphabet of unary predicate symbols (concepts), binary predicate symbols (roles), and constants (objects).

A **DL ontology** $\mathcal{O}$ is characterized by a pair $\langle \mathcal{T}, \mathcal{A} \rangle$ such that:

- the **TBox** $\mathcal{T}$ represents the **intensional** level of the ontology, i.e. it consists of a set of universal assertions (called inclusion assertions) on concepts and roles.

- the **ABox** $\mathcal{A}$ represents the **extensional** level of the ontology, i.e. it consists of a set of membership assertions, stating that a given object (or pair of objects) is an instance of an atomic concept (or an atomic role).

A **conjunctive query** $q$ is a conjunction of atoms over basic concepts and roles of $\mathcal{T}$

$$q = \{ \vec{x} \mid \exists \vec{y}. \text{conj} (\vec{x}, \vec{y}) \}$$

**Example:**

$$\{ x \mid \exists y. \text{Manager} (x) \land \text{Member} (x, y) \land \exists \text{Director} (x) \}$$
Description Logic Ontologies – Semantics

Pure first-order logic semantics for $O = \langle \mathcal{T}, \mathcal{A} \rangle$.

An interpretation is a pair $I = (\Delta^I, \cdot^I)$, where

- $\Delta^I$ is a non-empty set, called the domain of $I$
- $\cdot^I$ is the interpretation function of $I$, i.e., a function that assigns a different element of $\Delta^I$ to each constant (unique name assumption), a subset of $\Delta^I$ to each concept, and a subset of $\Delta^I \times \Delta^I$ to each role

An interpretation is a model of $O$ if it satisfies all inclusion assertions in $\mathcal{T}$ and all membership assertions in $\mathcal{A}$.

$\leadsto$ Open World Assumption: no negative fact is assumed by default.
Stand-alone ontologies

In our approach to a stand-alone ontology $\mathcal{O} = \langle T, A \rangle$:

- **Intensional level**
  - Represented as a Description Logic $T$Box $T$
  - Constrains the possible models

- **Extensional level**
  - The $A$Box $A$ (over atomic concepts and roles) is stored in a relational database (see later)
  - The $A$Box can be very large
Reasoning services

- Knowledge base satisfiability: check whether $\mathcal{O} = \langle \mathcal{I}, \mathcal{A} \rangle$ has a model.

- Query answering amounts to computing certain answers to $q$ wrt $\mathcal{O}$:

$$\text{cert}(q, \mathcal{O}) = \{ \vec{c} \mid \mathcal{O} \models q(\vec{c}) \}$$

i.e., the tuples of constants that are answers to the query $q$ in every model of $\mathcal{O}$.

Note the difference wrt query evaluation in databases!

Basic Question: For which ontology languages (i.e., DLs) can we rephrase query answering over an ontology into query answering over a relational database?
The DL-Lite family

- Is a family of DLs optimized according to the tradeoff between expressive power and data complexity.
- Two maximal languages that enjoy nice computational properties: $DL\text{-}\text{Lite}\mathcal{F}$, $DL\text{-}\text{Lite}\mathcal{R}$
  (we use simply $DL\text{-}\text{Lite}$ to refer to both languages)
- With minimal additions to $DL\text{-}\text{Lite}\mathcal{F}$ or $DL\text{-}\text{Lite}\mathcal{R}$, such nice properties are lost.

M. Lenzerini

Ontology-based access to data sources
**DL-Lite**

**TBox Language** (\( A \) atomic concept, \( P \) atomic role):

- **Concept inclusion assertions:** \( C_l \sqsubseteq C_r \), with:

\[
\begin{align*}
    C_l & \rightarrow A \mid \exists R \mid C_l \sqcap C_2 \mid C_l \sqcup C_2 \\
    C_r & \rightarrow A \mid \exists R \mid C_r \sqcap C_2 \mid \bot \mid \top \\
    R & \rightarrow P \mid P^- 
\end{align*}
\]

- **Functionality assertions:** \((\text{funct } R)\)

**Observations:**

- Captures the basic constructs of all ontology languages, including Entity Relationship Diagrams and UML Class Diagrams
- Notable exception: covering constraints in generalizations – if we add them, query answering becomes coNP-hard in data complexity
**DL-Lite^R**

TBox Language ($A$ atomic concept, $P$ atomic role):

- **Concept inclusion assertions:** $Cl \sqsubseteq Cr$, with:

  
  \[
  Cl \rightarrow A \mid \exists R \mid Cl_1 \sqcap Cl_2 \mid Cl_1 \sqcup Cl_2 \\
  Cr \rightarrow A \mid \exists R.Cr \mid Cr_1 \sqcap Cr_2 \mid \bot \mid \top \\
  R \rightarrow P \mid P^- 
  \]

- **Role inclusion assertions:** $R_1 \sqsubseteq R_2$

**Properties:**
- Drops functional restrictions in favor of ISA between roles
- Extends (the DL fragment of) RDFS
# Semantics of DL-Lite

<table>
<thead>
<tr>
<th>Construct</th>
<th>Syntax</th>
<th>Example</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>bottom, top</td>
<td>( \bot, \top )</td>
<td></td>
<td>( \emptyset, \Delta^I )</td>
</tr>
<tr>
<td>atom. conc.</td>
<td>( A )</td>
<td>Doctor</td>
<td>( A^I \subseteq \Delta^I )</td>
</tr>
<tr>
<td>atom. role</td>
<td>( P )</td>
<td>child</td>
<td>( P^I \subseteq \Delta^I \times \Delta^I )</td>
</tr>
<tr>
<td>exist. res.</td>
<td>( \exists P )</td>
<td>( \exists \text{child} )</td>
<td>( { d \mid \exists e. (d, e) \in P^I } )</td>
</tr>
<tr>
<td>exist. res.</td>
<td>( \exists P^- )</td>
<td>( \exists \text{child}^- )</td>
<td>( { e \mid \exists d. (d, e) \in P^I } )</td>
</tr>
<tr>
<td>negation</td>
<td>( \neg \text{Cl} )</td>
<td>( \neg \text{Doctor} )</td>
<td>( \Delta^I \setminus \text{Cl}^I )</td>
</tr>
<tr>
<td>incl. asser.</td>
<td>( \text{Cl} \subseteq \text{Cr} )</td>
<td>( \text{Father} \subseteq \exists \text{child} )</td>
<td>( \text{Cl}^I \subseteq \text{Cr}^I )</td>
</tr>
<tr>
<td>funct. asser.</td>
<td>(funct ( P ))</td>
<td>(funct ( \text{succ} ))</td>
<td>( \forall d, e, e'. (d, e) \in P^I \land (d, e') \in P^I \supset e = e' )</td>
</tr>
<tr>
<td>funct. asser.</td>
<td>(funct ( P^- ))</td>
<td>(funct ( \text{child}^- ))</td>
<td>( \forall e, e', d. (e, d) \in P^I \land (e', d) \in P^I \supset e = e' )</td>
</tr>
<tr>
<td>mem. asser.</td>
<td>( \text{Cl}(a) )</td>
<td>( \text{Father}(\text{bob}) )</td>
<td>( a^I \in A^I )</td>
</tr>
<tr>
<td>mem. asser.</td>
<td>( P(a, b) )</td>
<td>( \text{child}(\text{bob, ann}) )</td>
<td>( (a^I, b^I) \in P^I )</td>
</tr>
</tbody>
</table>

- inclusion assertions \(\rightarrow\) inclusion dependencies or disjointness constraints
- functionality assertions \(\rightarrow\) functional dependencies
- membership assertions \(\rightarrow\) tuples on an incomplete database
### Capturing basic ontology constructs in *DL-Lite*

<table>
<thead>
<tr>
<th>Ontology Construct</th>
<th>DL-Lite Formulation</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 relations</td>
<td>$\exists P \sqsubseteq A_1$, $\exists P^- \sqsubseteq A_2$</td>
</tr>
<tr>
<td>mandatory participation</td>
<td>$A_1 \sqsubseteq \exists P$, $A_2 \sqsubseteq \exists P^-$</td>
</tr>
<tr>
<td>functionality of relations (in $DL-Lite_\mathcal{F}$)</td>
<td>$(funct\ P)$, $(funct\ P^-)$</td>
</tr>
<tr>
<td>ISA between relations (in $DL-Lite_\mathcal{R}$)</td>
<td>$R_1 \sqsubseteq R_2$</td>
</tr>
</tbody>
</table>
**DL-Lite – Example**

- **Manager** ⊑ **Employee**
- **AreaManager** ⊑ **Manager**
- **TopManager** ⊑ **Manager**
- **AreaManager** ⊑ ¬**TopManager**
- ∃**WorksFor** ⊑ **Employee**
- ∃**WorksFor** ⊑ **Project**
- Project ⊑ ∃**WorksFor**⁻
  (funct **WorksFor**)  
  (funct **WorksFor**⁻)

**Note:** in *DL-Lite* we cannot capture completeness of the hierarchy
Query answering: computing certain answers

Logical inference

\[ q \rightarrow cert(q, T \cup A) \]

\[ T \rightarrow \]

\[ A \rightarrow \]
Query answering can always be thought as done in two phases:

1. **Perfect reformulation**: producing the query \( r_q, T \), namely the function \( \text{cert}[q, T](\cdot) \)

2. **Query evaluation**: evaluating \( r_q, T \) over the ABox \( A \) seen as a database, and forgetting about the TBox \( T \) – produces \( \text{cert}(q, \emptyset) \)

For a query language \( Q \) and for a DL \( \mathcal{L} \), query answering is \( Q \)-reducible if, for each \( q \) and each \( T \) expressed in \( \mathcal{L} \), \( r_q, T \) is in \( Q \).

Special case of interest: FOL-reducibility
**Q-reducibility and data complexity**

Q-reducibility is tightly related to data complexity, i.e., complexity of evaluating \( r_{q,T} \) measured in the size of the ABox \( \mathcal{A} \).

Special cases of interest:

- **Q** is FOL – the DL enjoys FOL-reducibility
  - \( \sim \) Query evaluation via RDBMS
  - \( \sim \) **Q** is in LOGSPACE

- **Q** is NLOGSPACE-hard \( \sim \) Query evaluation requires linear recursion

- **Q** is PTIME-hard \( \sim \) Query evaluation requires recursion (e.g., Datalog)

- **Q** is coNP-hard \( \sim \) Query evaluation requires power of Disjunctive Datalog
Previous work on data complexity for DL query answering

Much of the previous work deals with atomic queries only (instance checking in DLs):

[Donini & al. JLC’94] Data and combined complexity for DLs up to $\mathcal{ALC}$

[Hustadt & al. IJCAI’05] Data complexity for very expressive DLs via reduction to Disjunctive Datalog. Identify also polynomial cases (Horn-$\mathcal{SHIQ}$)

Complexity of answering conjunctive queries has been addressed in:

[Levy & Rousset AIJ’98] coNP upper bound for $\mathcal{ALCNR}$ knowledge bases (CARIN setting)

[— & al. AAAI’00] EXPTIME upper bound for $\mathcal{DLR}$ knowledge bases (via reduction to PDL)

[— & al. AAAI’05] Polynomial upper bound for $\mathcal{DL-Lite}$ knowledge base (using techniques drawn from databases with constraints)
Previous work on query answering under dependencies

Query answering (and query containment) under dependencies has been studied extensively in databases:

[Johnson & Klug JCSS’84] query containment under inclusion dependencies

[Calì & Lembo & Rosati ’03] query answering under keys and non-key-conflicting inclusion dependencies

[Fagin & al.’03] recent work on data exchange
Our basic question, and the *DL-Lite* family

- Both $DL-Lite_F$, and $DL-Lite_R$ enjoy FOL-reducibility

- With minimal additions to $DL-Lite_F$ or $DL-Lite_R$, data complexity jumps to \textsc{NLogSpace} or above

$\leadsto$ We lose FOL-reducibility

Provides an answer to our basic question: *For which DLs can we rephrase query answering over an ontology into query answering over a relational database?*
Query answering in *DL-Lite*

Given a CQ $q$ and $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, we compute $\text{cert}(q, \mathcal{O})$ as follows:

1. Store ABox $\mathcal{A}$ in a relational database
2. Close TBox $\mathcal{T}$, and check for satisfiability wrt $\mathcal{A}$
3. Using $\mathcal{T}$, reformulate CQ $q$ as a union $r_{q,\mathcal{T}}$ of CQs
4. Evaluate $r_{q,\mathcal{T}}$ directly over $\mathcal{A}$ using RDBMS technology

Correctness of this algorithm shows FOL-reducibility of query answering in *DL-Lite*.

$\leadsto$ Query answering over *DL-Lite* ontologies can be done using RDBMS technology.

$\leadsto$ Prototype system implemented: QuOnto
Query answering: 1. ABox storage

ABox $\mathcal{A}$ stored as a relational database in a standard DBMS as follows:

- For each atomic concept $A$ used in ABox:
  - define a unary relational table $\text{tab}_A$
  - populate $\text{tab}_A$ with every $\langle c \rangle$ such that $A(c)$ is in ABox

- For each role $P$ used in ABox,
  - define a binary relational table $\text{tab}_P$
  - populate $\text{tab}_P$ with every $\langle a, b \rangle$ such that $P(a, b)$ is in ABox
Query answering: 2. KB satisfiability

To check if \( \mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle \) is satisfiable (this works for DL-Lite\(\mathcal{F}\), slightly more complicated for DL-Lite\(\mathcal{R}\)):

2.1 Close the TBox \( \mathcal{T} \) by computing all disjointness assertions that are implied according to the rule:

- if \( \text{Cl}_1 \sqsubseteq B \) and \( B \sqcap \text{Cl}_2 \sqsubseteq \bot \), then add \( \text{Cl}_1 \sqcap \text{Cl}_2 \sqsubseteq \bot \).

2.2 Verify that the ABox \( \mathcal{A} \) does not explicitly violate any disjointness or functionality assertion of the closed TBox.

This can be done by issuing suitable conjunctive queries over the database tables storing \( \mathcal{A} \), e.g.:

- \( \mathcal{A} \) violates \( A_1 \sqcap A_2 \sqsubseteq \bot \) iff \( q(\mathcal{A}) \neq \emptyset \), where
  \[
  q = \{ \langle \rangle \mid A_1(x), A_2(x) \}
  \]

- \( \mathcal{A} \) violates (funct \( P \)) iff \( q(\mathcal{A}) \neq \emptyset \), where
  \[
  q = \{ \langle \rangle \mid P(x, y), P(x, z), y \neq z \} \]
Query answering: 3 and 4

3. Query reformulation
Reformulate the CQ $q$ into a set of queries – apply to $q$ in all possible ways the inclusion assertions in the TBox (and unify atoms, if possible), so as to obtain new disjuncts that could contribute to the answer:

\[ A_1 \sqsubseteq A_2 \quad \ldots, A_2(x), \ldots \quad \leadsto \quad \ldots, A_1(x), \ldots \]
\[ \exists P \sqsubseteq A \quad \ldots, A(x), \ldots \quad \leadsto \quad \ldots, P(x, _), \ldots \]
\[ \exists P^- \sqsubseteq A \quad \ldots, A(x), \ldots \quad \leadsto \quad \ldots, P(_ , x), \ldots \]
\[ A \sqsubseteq \exists P \quad \ldots, P(x, _), \ldots \quad \leadsto \quad \ldots, A(x), \ldots \]
\[ A \sqsubseteq \exists P^- \quad \ldots, P(_ , x), \ldots \quad \leadsto \quad \ldots, A(x), \ldots \]
\[ \exists P_1 \sqsubseteq \exists P_2 \quad \ldots, P_2(x, _), \ldots \quad \leadsto \quad \ldots, P_1(x, _), \ldots \]

4. Evaluation of reformulated query
The resulting union of CQs is evaluated over the ABox stored as relational database.
Query answering in *DL-Lite* – Observations

Our technique is based on rewriting (i.e., inverse chase), rather than chasing the database.

What if we wanted to chase the database?

- We are in a case where the chase would be infinite in general (no weakly acyclic tgds).
- Note that *DL-Lite* does not even have the finite model property.
- Could we find a bound on the size of the chase that guarantees correctness of query answering?
  - No! For any bound we fix for the chase, can give a query that, when evaluated on the chase does not provide the certain answers.
  - We could find a bound that depends on the size of the query.
**DL-Lite: complexity results**

- KB satisfiability is
  - **polynomial** in the size of TBox and of ABox (in fact LOGSPACE in the ABox)

- Query answering is
  - **exponential** in the size of the query (NP-complete)
  - **polynomial** in the size of TBox and of ABox (in fact LOGSPACE in the ABox)

Can we further extend these results to more expressive ontology languages / DLs?
## Summary of results on data complexity

<table>
<thead>
<tr>
<th></th>
<th>$Cl$</th>
<th>$Cr$</th>
<th>$\mathcal{F}$</th>
<th>$\mathcal{R}$</th>
<th>Data complexity of query answering</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$DL$-$Lite_\mathcal{F}$</td>
<td>$\top$</td>
<td>$\top$</td>
<td>$\bot$</td>
<td>in LOGSPACE</td>
</tr>
<tr>
<td>2</td>
<td>$DL$-$Lite_\mathcal{R}$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>in LOGSPACE</td>
</tr>
<tr>
<td>3</td>
<td>$DLR$-$Lite_\mathcal{F}$</td>
<td>$\top$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>in LOGSPACE</td>
</tr>
<tr>
<td>4</td>
<td>$DLR$-$Lite_\mathcal{R}$</td>
<td>$\bot$</td>
<td>$\top$</td>
<td>$\bot$</td>
<td>in LOGSPACE</td>
</tr>
<tr>
<td>5</td>
<td>$A \mid \exists P.A$</td>
<td>$A$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>NLOGSPACE-hard</td>
</tr>
<tr>
<td>6</td>
<td>$A$</td>
<td>$A \mid \forall P.A$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>NLOGSPACE-hard</td>
</tr>
<tr>
<td>7</td>
<td>$A$</td>
<td>$A \mid \exists P.A$</td>
<td>$\top$</td>
<td>$\bot$</td>
<td>NLOGSPACE-hard</td>
</tr>
<tr>
<td>8</td>
<td>$A \mid \exists P.A \mid A_1 \sqcap A_2$</td>
<td>$A$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>PTIME-hard</td>
</tr>
<tr>
<td>9</td>
<td>$A \mid A_1 \sqcap A_2$</td>
<td>$A \mid \forall P.A$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>PTIME-hard</td>
</tr>
<tr>
<td>10</td>
<td>$A \mid A_1 \sqcap A_2$</td>
<td>$A \mid \exists P.A$</td>
<td>$\top$</td>
<td>$\bot$</td>
<td>PTIME-hard</td>
</tr>
<tr>
<td>11</td>
<td>$A \mid \exists P.A \mid \exists P^-\cdot A$</td>
<td>$A \mid \exists P$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>PTIME-hard</td>
</tr>
<tr>
<td>12</td>
<td>$A$</td>
<td>$A \mid \exists P.A \mid \exists P^-\cdot A$</td>
<td>$\top$</td>
<td>$\bot$</td>
<td>PTIME-hard</td>
</tr>
<tr>
<td>13</td>
<td>$A \mid \exists P.A$</td>
<td>$A \mid \exists P.A$</td>
<td>$\top$</td>
<td>$\bot$</td>
<td>PTIME-hard</td>
</tr>
<tr>
<td>14</td>
<td>$A \mid \lnot A$</td>
<td>$A$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>coNP-hard</td>
</tr>
<tr>
<td>15</td>
<td>$A$</td>
<td>$A \mid A_1 \sqcup A_2$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>coNP-hard</td>
</tr>
<tr>
<td>16</td>
<td>$A \mid \forall P.A$</td>
<td>$A$</td>
<td>$\bot$</td>
<td>$\bot$</td>
<td>coNP-hard</td>
</tr>
</tbody>
</table>

All NLOGSPACE and PTIME hardness results hold already for atomic queries.
An **NLOGSPACE-hard cases**

Adding **qualified existential** on the lhs of inclusions makes instance checking (and hence query answering) **NLOGSPACE-hard**:

\[
\begin{align*}
Cl & \rightarrow A \mid \exists P.A \\
Cr & \rightarrow A \\
R & \rightarrow P \\
\text{(funct } R\text{) is not allowed}
\end{align*}
\]

Hardness proof is by a reduction from reachability in directed graphs:

- **TBox** \( \mathcal{T} \) contains a single inclusion assertion \( \exists P.A \sqsubseteq A \)
- **ABox** \( \mathcal{A} \) encodes graph \( G \) using \( P \), and asserts \( A(d) \)

Result:

\( (\mathcal{T}, \mathcal{A}) \models A(s) \) iff \( d \) is reachable from \( s \) through \( P \) in \( G \)
Data layer independent from the ontology

We now come back to our original problem.

- No Abox – instances of concepts and roles derive from the sources
- The mapping language is now crucial
- Both the semantics, and query answering should be adapted to this case
Our framework

- The data sources are wrapped into a relational database $DB$ (constituted by the relational schema, and the extensions of the relations), so that we can query such data by using SQL.

- The database $DB$ is independent from the ontology; in other words, our aim is to link to the ontology a collection of data that exist autonomously, and have not been necessarily structured with the purpose of storing the ontology instances.

- $\text{ans}(\varphi, DB)$ denotes the set of tuples (of the arity of $\varphi$) of value constants returned as the result of the evaluation of the SQL query $\varphi$ over the database $DB$. 
Dealing with the impedance mismatch

Let $\Gamma_V$ be the alphabet of constants (values) appearing in the sources. We introduce a new alphabet $\Lambda$ of function symbols, where each function symbol has an associated arity, specifying the number of argument it accepts.

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

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

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

$\leadsto$ No confusion between values stored in the database and the terms denoting objects.
Ontology with mappings to data sources

An ontology with mappings is characterized by a triple $\mathcal{O}_m = \langle \mathcal{T}, \mathcal{M}, DB \rangle$ such that:

- $\mathcal{T}$ is a TBox;
- $DB$ is a relational database;
- $\mathcal{M}$ is a set of mapping assertions, each one of the form $\Phi \rightsquigarrow \Psi$

where $\Phi$ is an arbitrary SQL query of arity $n > 0$ over $DB$, $\Psi$ is a conjunctive query over $\mathcal{T}$ of arity $n' > 0$ without non-distinguished variables, that possibly involves variable terms. A variable term is a term of the same form as the object terms introduced above, with the difference that variables appear as argument of the function. In other words, a variable term has the form $f(\bar{z})$, where $f$ is a function symbol in $\Lambda$ of arity $m$, and $\bar{z}$ denotes an $m$-tuple of variables.
Example

Let $DB$ be the database constituted by a set of relations with the following signature:

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

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

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

\[
\begin{align*}
&\text{tempEmp} \sqsubseteq \text{employee} & \text{project} \sqsubseteq \exists \text{ProjName} \\
&\text{manager} \sqsubseteq \text{employee} & \text{tempEmp} \sqsubseteq \exists \text{until} \\
&\text{employee} \sqsubseteq \text{person} & \exists \text{until} \sqsubseteq \exists \text{WORKS-FOR} \\
&\text{employee} \sqsubseteq \exists \text{WORKS-FOR} & \exists \text{WORKS-FOR} \sqsubseteq \text{project} \\
&\exists \text{WORKS-FOR} \sqsubseteq \text{project} & \exists \text{ProjName} \sqsubseteq \text{person} \\
&\text{person} \sqsubseteq \exists \text{PersName} & \text{manager} \sqsubseteq \neg \exists \text{until} \\
&(\text{funct} \text{ PersName}) & (\text{funct} \text{ until})
\end{align*}
\]

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

Mapping assertions $\mathcal{M}$:

$M_{m_1}$: $\text{SELECT SSN, PROJ, D FROM } D_1$

$M_{m_2}$: $\text{SELECT SSN, NAME FROM } D_2$

$M_{m_3}$: $\text{SELECT SSN, NAME FROM } D_3, D_4$
WHERE $D_3.\text{CODE}=D_4.\text{CODE}$

$M_{m_4}$: $\text{SELECT CODE, NAME FROM } D_3$
WHERE CODE NOT IN (SELECT CODE FROM $D_4$)

$\leadsto tempEmp(\text{pers}(\text{SSN})),$
$\text{WORKS-FOR}(\text{pers}(\text{SSN}), \text{proj}(\text{PROJ})),$
$\text{ProjName}(\text{proj}(\text{PROJ}), \text{str}(\text{PROJ})),$
until(\text{pers}(\text{SSN}), \text{date}(D))$

$\leadsto \text{employee}(\text{pers}(\text{SSN})),$
$\text{PersName}(\text{pers}(\text{SSN}), \text{str}(\text{NAME}))$

$\leadsto \text{manager}(\text{pers}(\text{SSN})),$
$\text{PersName}(\text{pers}(\text{SSN}), \text{str}(\text{NAME}))$

$\leadsto \text{manager}(\text{mgr}(\text{CODE})),$
$\text{PersName}(\text{mgr}(\text{CODE}), \text{str}(\text{NAME}))$
Let $\Psi(\vec{x})$ be a formula over a TBox with $n$ distinguished variables $\vec{x}$, and let $\vec{v}$ a tuple of constants of arity $n$. Then the ground instance $\Psi[\vec{x}/\vec{v}]$ of $\Psi(\vec{x})$ is the formula obtained by substituting every occurrence of $x_i$ with $v_i$ in $\Psi(\vec{x})$.

We say that $I$ satisfies $\Phi(\vec{x}) \models \Psi(\vec{t})$ wrt a database $DB$, if for every tuple of values $\vec{v}$ such that $\vec{v} \in \text{ans}(\Phi, DB)$, and for each ground atom $X$ in $\Psi[\vec{x}/\vec{v}]$, we have that:

- if $X$ has the form $A(s)$, then $s^I \in A^T$;
- if $X$ has the form $P(s_1, s_2)$, then $(s_1^I, s_2^I) \in P^T$.

An interpretation $I = (\Delta^I, \cdot^I)$ is a model of $O_m = \langle T, M, DB \rangle$ if:

- $I$ is a model of $T$;
- $I$ satisfies $M$ wrt $DB$, i.e., satisfies every assertion in $M$ wrt $DB$. 

M. Lenzerini

Ontology-based access to data sources
Query answering

Given a CQ $q$ and $O_m = \langle T, M, DB \rangle$ (assumed to be satisfiable), we compute $cert(q, O_m)$ as follows:

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

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

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

$\leadsto$ Prototype system implemented: MASTRO
Unfolding

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

\[
\begin{align*}
tempEmp(pers(s)) & \leftarrow Aux_{11}(s) \\
WORKS-FOR(pers(s), proj(p)) & \leftarrow Aux_{12}(s, p) \\
ProjName(proj(p), p) & \leftarrow Aux_{13}(p) \\
until(pers(s), d) & \leftarrow Aux_{14}(s, d) \\
employee(pers(s)) & \leftarrow Aux_{21}(s) \\
PersName(pers(s), n) & \leftarrow Aux_{22}(s, n) \\
manager(pers(s)) & \leftarrow Aux_{31}(s) \\
PersName(pers(s), n) & \leftarrow Aux_{32}(s, n) \\
manager(mgr(c)) & \leftarrow Aux_{41}(c) \\
PersName(mgr(c), n) & \leftarrow Aux_{42}(c, n)
\end{align*}
\]

where \( Aux_{ij} \) is a predicate denoting the result of the evaluation over \( DB \) of the query \( \Phi_{mi_j} \) in the left-hand side of the mapping \( \mathcal{M}_{mi_j} \).
Unfolding

Consider \( q(x) \leftarrow WORKS-FOR(x, y) \), whose reformulation \( Q' = r_{q,T} \) is:

\[
\begin{align*}
Q'(x) & \leftarrow WORKS-FOR(x, y) \\
Q'(x) & \leftarrow \text{until}(x, y) \\
Q'(x) & \leftarrow \text{tempEmp}(x) \\
Q'(x) & \leftarrow \text{employee}(x) \\
Q'(x) & \leftarrow \text{manager}(x)
\end{align*}
\]

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

\[
\begin{align*}
q(\text{pers}(s)) & \leftarrow \text{Aux}_{12}(s, p) \\
q(\text{pers}(s)) & \leftarrow \text{Aux}_{14}(s, d) \\
q(\text{pers}(s)) & \leftarrow \text{Aux}_{11}(s) \\
q(\text{pers}(s)) & \leftarrow \text{Aux}_{21}(s) \\
q(\text{pers}(s)) & \leftarrow \text{Aux}_{31}(s, n) \\
q(\text{mgr}(c)) & \leftarrow \text{Aux}_{41}(c, n)
\end{align*}
\]
Unfolding

From the above formulation, it is now possible to derive the corresponding SQL query \( Q'' \) that can be directly issued over the database \( DB \):

\[
\begin{align*}
\text{SELECT } & \text{ CONCAT(CONCAT('pers (',SSN),')) } \\
\text{FROM } & D_1 \\
\text{UNION} & \\
\text{SELECT } & \text{ CONCAT(CONCAT('pers (',SSN),')) } \\
\text{FROM } & D_2 \\
\text{UNION} & \\
\text{SELECT } & \text{ CONCAT(CONCAT('pers (',SSN),')) } \\
\text{FROM } & D_3, D_4 \\
\text{WHERE} & \ D_3\.CODE=D_4\.CODE \\
\text{UNION} & \\
\text{SELECT } & \text{ CONCAT(CONCAT('mgr (',CODE),')) } \\
\text{FROM } & D_3 \\
\text{WHERE} & \text{ CODE NOT IN (SELECT CODE FROM } D_4 )
\end{align*}
\]
Current and future work

Almost done:

- Implementation of MASTRO (see the system web page at http://www.dis.uniroma1.it/~quonto)
- Can we move to LAV mappings?
- Extensions of $DL$-$Lite_F$ and $DL$-$Lite_R$ with additional constructs
- Update on stand-alone ontologies

Current and future work:

- Experiments
- Going beyond (unions) of conjunctive queries (weaker semantics wrt FOL)
- Write-also access: updating the data sources through an ontology
- What if we want to restrict the attention to finite models only?
Some references


