Answering Queries in Description Logics: Theory and Applications to Data Management

Diego Calvanese[1], Michael Zakharyaschev[2]

[1] KRDB Research Centre
Free University of Bozen-Bolzano

[2] Birbeck College, London

ESSLLI 2010, August 14–20, 2010
Copenhagen, Denmark

Overview of the Course

1. Introduction and background
    1. Ontology-based data management
    2. Brief introduction to computational complexity
    3. Query answering in databases
    4. Querying databases and ontologies
2. Lightweight description logics
    5. Introduction to description logics
    6. DLs for conceptual data modeling: the *DL-Lite* family
    7. The $\mathcal{EL}$ family of tractable description logics
3. Query answering in the *DL-Lite* family
    8. Query answering in description logics
    9. Lower bounds for description logics beyond *DL-Lite*
    10. Reasoning and query answering by rewriting
4. The combined approach to query answering
    11. Query answering in *DL-Lite*: data completion
    12. Query rewriting in $\mathcal{EL}$
5. Linking ontologies to relational data
    13. The impedance mismatch problem
    14. Query answering in Ontology-Based Data Access systems
6. Conclusions and references

Lecture V

Linking ontologies to relational data

Outline of Lecture 5

Outline of Lecture 5

1. The impedance mismatch problem
2. Query answering in ontology-based data access systems
3. References

Managing ABoxes

In the traditional DL setting, it is assumed that the data is maintained in the ABox of the ontology:

- The ABox is perfectly compatible with the TBox:
  - the vocabulary of concepts, roles, and attributes is the one used in the TBox.
  - The ABox "stores" abstract objects, and these objects and their properties are those returned by queries over the ontology.
- There may be different ways to manage the ABox from a physical point of view:
  - Description Logics reasoners maintain the ABox is main-memory data structures.
  - When an ABox becomes large, managing it in secondary storage may be required, but this is again handled directly by the reasoner.

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**.

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.

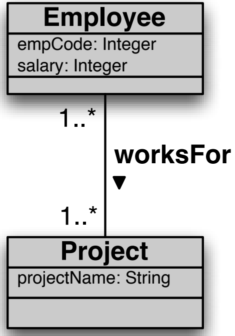
Note: the **ABox** is only **virtual**, and the objects are not materialized.

Solution to the impedance mismatch problem

We define a **mapping language** that allows for specifying how to transform data into abstract objects: [Poggi *et al.*, 2008]

- 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.

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

$D_2[Code\colon \text{String}, Salary\colon \text{Int}]$
Employee's code with salary

$D_3[Code\colon \text{String}, SSN\colon \text{String}]$
Employee's Code with SSN

...

Intuitively:

- An employee should be created from her SSN: **pers**(SSN)
- A project should be created from its name: **proj**($PrName$)

Creating object identifiers

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

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

Example

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

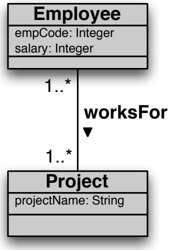
Def.: A **mapping assertion** between a database $\mathcal{D}$ and a TBox $\mathcal{T}$ has the form

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

where

- Φ is an arbitrary SQL query of arity $n > 0$ over $\mathcal{D}$;
- Ψ is a conjunctive query over $\mathcal{T}$ of arity $n' > 0$ **without non-distinguished variables**;
- $\vec{x}$, $\vec{y}$ are variables, with $\vec{y} \subseteq \vec{x}$;
- $\vec{t}$ are variable terms of the form $\mathbf{f}(\vec{z})$, with $\mathbf{f} \in \Lambda$ and $\vec{z} \subseteq \vec{x}$.

Mapping assertions – Example



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

...

m_1: SELECT SSN, PrName FROM D_1 $\rightsquigarrow$ Employee(**pers**(*SSN*)), Project(**proj**(*PrName*)), projectName(**proj**(*PrName*), *PrName*), worksFor(**pers**(*SSN*), **proj**(*PrName*))

m_2: SELECT SSN, Salary FROM D_2, D_3 WHERE D_2.Code = D_3.Code $\rightsquigarrow$ Employee(**pers**(*SSN*)), salary(**pers**(*SSN*), *Salary*)

Outline of Lecture 5

Ontology-Based Data Access System

The mapping assertions are a crucial part of an Ontology-Based Data Access System.

Def.: Ontology-Based Data Access System

is a triple $\mathcal{O} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$, where

- $\mathcal{T}$ is a TBox.
- $\mathcal{D}$ is a relational database.
- $\mathcal{M}$ is a set of mapping assertions between $\mathcal{T}$ and $\mathcal{D}$.

Semantics of mappings

To define the semantics of an OBDA system $\mathcal{O} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$, we first need to define the semantics of mappings.

Def.: Satisfaction of a mapping assertion with respect to a database

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

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

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

Note: $\mathit{Eval}(\Phi, \mathcal{D})$ denotes the result of evaluating Φ over the database $\mathcal{D}$.
$\Psi[\vec{x}/\vec{v}]$ denotes Ψ where each x_i has been substituted with v_i.

Semantics of an OBDA system

Def.: Model of an OBDA system

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

- $\mathcal{I}$ is a model of $\mathcal{T}$;
- $\mathcal{I}$ 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.

Answering queries over an OBDA system

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

- 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 $\mathcal{D}$.

Two approaches for query answering over $\mathcal{O}$:

1. Bottom-up approach:
   - Explicitly construct an ABox $\mathcal{A}_{\mathcal{M},\mathcal{D}}$ using $\mathcal{D}$ and $\mathcal{M}$, and compute the certain answers over $\langle \mathcal{T}, \mathcal{A}_{\mathcal{M},\mathcal{D}} \rangle$.
   - Is conceptually simpler, but less efficient (PTIME in the data).
2. Top-down approach:
   - Unfold the query w.r.t. $\mathcal{M}$ and generate a query over $\mathcal{D}$.
   - Is more sophisticated, but also more efficient.

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* OBDA system $\mathcal{O} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$ is

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

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.

Outline of Lecture 5

References I

[Poggi *et al.*, 2008] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati.

Linking data to ontologies.

J. on Data Semantics, X:133–173, 2008.