COMP718: Ontologies and Knowledge Bases

Lecture 9: Ontology/Conceptual Model based Data Access

Maria Keet

email: keet@ukzn.ac.za

home: http://www.meteck.org

School of Mathematics, Statistics, and Computer Science
University of KwaZulu-Natal, South Africa

10 April 2012

Outline

1 OBDA Options

2 Some technical details
- Introduction
- The ontology language
- The mapping layer
  - 'Impedance' mismatch
  - Mapping assertions
- Query answering

An ontology with a very large ABox (intro last week)

- Scaling up to realistic size knowledge base handling large amounts of data
- To realise this, we need
  - A language of relatively low computational complexity
  - A way to store large amounts of data
  - Some mechanism to link up the previous two ingredients
  - Query (and reason over) the combination of the previous three
- Use the "Ontology-Based Data Access" (OBDA) approach
  - with the "ontology" in OBDA just a DL knowledge base
  - Most examples and use cases: the 'ontology' is a DL-formalised conceptual data model
- Example application with the WONDER system

An ontology with a very large ABox (this week)

⇒ What are the options to link an ontology to large amounts of data?

- Two principal options (in KR view): query rewriting and data completion
- Several implementation infrastructures; 'external ABox' most popular (realised with RDBMS or RDF Triple store)

⇒ What is there behind the scenes for the non-graphical OBDA-part in WONDER and the OBDA systems you set up in the lab?

Outline

1 OBDA Options

2 Some technical details
- Introduction
- The ontology language
- The mapping layer
  - 'Impedance' mismatch
  - Mapping assertions
- Query answering

OBDA options

- KR perspective (with OWA): query rewriting vs data completion
- DB perspective (with CWA): we probably won't cover this in the lecture
- See slides `obda-slides2012TomanCOMP718ukzn.pdf`

Outline

1. OBDA Options
2. Some technical details
   - Introduction
   - The ontology language
   - The mapping layer
     - 'Impedance' mismatch
     - Mapping assertions
   - Query answering

Linking ontologies to relational data[1]

- Ontology-Based Data Access systems (static components)
  - An ontology language
  - A mapping language
  - The data
- Query answering in Ontology-Based Data Access systems
  - Reasoning over the TBox
  - Query rewriting
  - Query unfolding
  - Relational database technology

These slides are based on Calvanese's MOSS'09 slides, which also will be made available

[1] More precisely: "Option I, v1.0" mentioned in David Toman's slides.

An OBDA system

Definition (Ontology-Based Data Access system)

An OBDA 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 $\mathcal{D}$

Note: this is for the current system, but one could conceive of a system that has an RDF triple store as $\mathcal{D}$

$\mathcal{D}$ as ABox

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

- 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
- Other ways to manage the ABox from an implementation point of view:
  - Description Logics reasoners maintain the ABox is main-memory data structures (recollect the 4 GB HGT-DB)
  - Hence, when an ABox becomes large, managing it in secondary storage may be required,

    but this is again handled directly by the reasoner

$\mathcal{D}$ as ABox

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

- 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
- Other ways to manage the ABox from an implementation point of view:
  - Description Logics reasoners maintain the ABox is main-memory data structures (recollect the 4 GB HGT-DB)
  - Hence, when an ABox becomes large, managing it in secondary storage may be required,

    but this is again handled directly by the reasoner

$\mathcal{D}$ = Relational database as ABox

- In addition to ABox scalability, there are other reasons to realise the ABox with $\mathcal{D}$:
    - 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
- 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
- New problems:
    - The so-called impedance mismatch between values in the relational database and the objects that the ABox expects
    - How to link the TBox to the "ABox" that is realised as a $\mathcal{D}$?

$\mathcal{D}$ = Relational database as ABox

- In addition to ABox scalability, there are other reasons to realise the ABox with $\mathcal{D}$:
  - 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
- 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
- New problems:
  - The so-called impedance mismatch between values in the relational database and the objects that the ABox expects
  - How to link the TBox to the "ABox" that is realised as a $\mathcal{D}$?

$\mathcal{D}$ = Relational database as ABox

- In addition to ABox scalability, there are other reasons to realise the ABox with $\mathcal{D}$:
  - 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
- 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
- New problems:
  - The so-called impedance mismatch between values in the relational database and the objects that the ABox expects
  - How to link the TBox to the "ABox" that is realised as a $\mathcal{D}$?

The DL-Lite family

- A family of DLs optimized according to the tradeoff between expressive power and complexity of query answering, with emphasis on data
- Carefully designed to have nice computational properties for answering UCQs (i.e., computing certain answers):
  - The same complexity as relational databases
  - Query answering can be delegated to a relational DB engine
  - The DLs of the *DL-Lite* family are essentially the maximally expressive ontology languages enjoying these nice computational properties
- Introduction of $\textit{DL-Lite}_{\mathcal{R}}$, a member of the *DL-Lite* family, essentially corresponds to OWL2 QL[2]

[2] Actually, the current OBDA implementation can handle $\textit{DL-Lite}_{\mathcal{A}}$, and all *DL-Lite* languages adhere to the UNA

DL-Lite$_{\mathcal{R}}$ (compacter DL notation of OWL 2 QL)

TBox assertions:

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

$$\begin{array}{lcl} Cl & \longrightarrow & A \mid \exists Q \\ Cr & \longrightarrow & A \mid \exists Q \mid \neg A \mid \neg \exists Q \\ Q & \longrightarrow & P \mid P^{-} \end{array}$$

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

$$R \longrightarrow Q \mid \neg Q$$

ABox assertions: $A(c)$, $P(c_1, c_2)$, with c_1, c_2 constants

Note: *DL-Lite*$_{\mathcal{R}}$ can be straightforwardly adapted to distinguish also between object and data properties (attributes).

$DL\text{-}Lite_{\mathcal{R}}$ (compacter DL notation of OWL 2 QL)

Construct	Syntax	Example	Semantics
atomic conc.	A	Doctor	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
exist. restr.	$\exists Q$	$\exists$child$^-$	$\{d \mid \exists e.(d,e) \in Q^{\mathcal{I}}\}$
at. conc. neg.	$\neg A$	$\neg$Doctor	$\Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$
conc. neg.	$\neg \exists Q$	$\neg \exists$child	$\Delta^{\mathcal{I}} \setminus (\exists Q)^{\mathcal{I}}$
atomic role	P	child	$P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
inverse role	P^-	child$^-$	$\{(o,o') \mid (o',o) \in P^{\mathcal{I}}\}$
role negation	$\neg Q$	$\neg$manages	$(\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}) \setminus Q^{\mathcal{I}}$
conc. incl.	$Cl \sqsubseteq Cr$	Father $\sqsubseteq \exists$child	$Cl^{\mathcal{I}} \subseteq Cr^{\mathcal{I}}$
role incl.	$Q \sqsubseteq R$	hasFather $\sqsubseteq$ child$^-$	$Q^{\mathcal{I}} \subseteq R^{\mathcal{I}}$
mem. asser.	$A(c)$	Father(bob)	$c^{\mathcal{I}} \in A^{\mathcal{I}}$
mem. asser.	$P(c_1, c_2)$	child(bob, ann)	$(c_1^{\mathcal{I}}, c_2^{\mathcal{I}}) \in P^{\mathcal{I}}$

DL-Lite$_{\mathcal{R}}$ (compacter DL notation of OWL 2 QL)

ISA between classes	$A_1 \sqsubseteq A_2$
Disjointness between classes	$A_1 \sqsubseteq \neg A_2$
Domain and range of properties	$\exists P \sqsubseteq A_1 \quad \exists P^- \sqsubseteq A_2$
Mandatory participation *(min card = 1)*	$A_1 \sqsubseteq \exists P \quad A_2 \sqsubseteq \exists P^-$
ISA between properties	$Q_1 \sqsubseteq Q_2$
Disjointness between properties	$Q_1 \sqsubseteq \neg Q_2$

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

Note2: *DL-Lite*$_{\mathcal{R}}$ cannot capture **functionality** on roles (*max card = 1*)

Relational database as ABox

- Sources store data, which is constituted by values taken from concrete domains, such as strings, integers, codes, ...
- Instances of concepts and relations in an ontology are (abstract) objects
- Solution:
  - Specify how to construct from the data values in the relational sources the (abstract) objects that populate the ABox of the ontology
  - Embed this specification in the mappings between the data sources and the ontology
- Use a *virtual ABox*, where the objects are not materialized

Relational database as ABox

- Sources store data, which is constituted by values taken from concrete domains, such as strings, integers, codes, ...
- Instances of concepts and relations in an ontology are (abstract) objects
- Solution:
  - Specify how to construct from the data values in the relational sources the (abstract) objects that populate the ABox of the ontology
  - Embed this specification in the mappings between the data sources and the ontology
- Use a *virtual ABox*, where the objects are not materialized

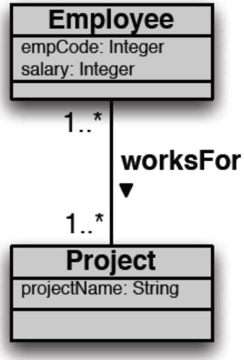
Relational database as ABox

- Sources store data, which is constituted by values taken from concrete domains, such as strings, integers, codes, ...
- Instances of concepts and relations in an ontology are (abstract) objects
- Solution:
  - Specify how to construct from the data values in the relational sources the (abstract) objects that populate the ABox of the ontology
  - Embed this specification in the mappings between the data sources and the ontology
- Use a *virtual ABox*, where the objects are not materialized

Solution to the impedance mismatch

- Define a mapping language that allows for specifying how to transform data into abstract objects, where
  - 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)

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*: 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

...

Intuitively:

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

Associate objects in the ontology to data in the tables

- Introduce an alphabet Λ of function symbols, each with an associated arity
- Use value constants from an alphabet Γ_V to denote values
- Use object terms instead of object constants to denote objects: and object term has the form $f(d_1, \ldots, d_n)$ with $f \in \Lambda$, and each d_i is a value constant in Γ_V

Example

- If a person is identified by her *SSN*, we can introduce a function symbol pers/1. If `NRM18JUL18` is a *SSN*, then pers(`NRM18JUL18`) denotes a person.
- If a person is identified by her *name* and *dateOfBirth*, we can introduce a function symbol pers/2. Then pers(`Mandela`, `18/07/18`) denotes a person.

Associate objects in the ontology to data in the tables

- Introduce an alphabet Λ of function symbols, each with an associated arity
- Use value constants from an alphabet Γ_V to denote values
- Use object terms instead of object constants to denote objects: and object term has the form $f(d_1, \ldots, d_n)$ with $f \in \Lambda$, and each d_i is a value constant in Γ_V

Example

- If a person is identified by her *SSN*, we can introduce a function symbol pers/1. If `NRM18JUL18` is a *SSN*, then pers(`NRM18JUL18`) denotes a person.
- If a person is identified by her *name* and *dateOfBirth*, we can introduce a function symbol pers/2. Then pers(`Mandela`, `18/07/18`) denotes a person.

Mapping assertions, formally

- Mapping assertions are used to extract the data from the DB to populate the ontology
- Use of variable terms, which are like object terms, but with variables instead of values as arguments of the functions

Definition (Mapping assertion between a database and a TBox)

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

$$\Phi \rightsquigarrow \Psi$$

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, possibly involving variable terms.

Mapping assertions, formally

- Mapping assertions are used to extract the data from the DB to populate the ontology
- Use of variable terms, which are like object terms, but with variables instead of values as arguments of the functions

Definition (Mapping assertion between a database and a TBox)

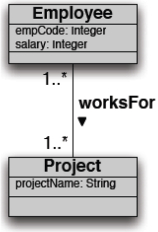
A mapping assertion between a database $\mathcal{D}$ and a TBox $\mathcal{T}$ has the form

$$\Phi \leadsto \Psi$$

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, possibly involving variable terms.

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 D1` ⇝ Employee(**pers**(*SSN*)), Project(**proj**(*PrName*)), projectName(**proj**(*PrName*), *PrName*), worksFor(**pers**(*SSN*), **proj**(*PrName*))

m_2: `SELECT SSN, Salary FROM D2, D3 WHERE D2.Code = D3.Code` ⇝ Employee(**pers**(*SSN*)), salary(**pers**(*SSN*), *Salary*)

Mapping assertions in $\mathcal{M}$

Definition (Mapping assertion in $\mathcal{M}$ in an OBDA system)

A mapping assertion between a database $\mathcal{D}$ and a TBox $\mathcal{T}$ in $\mathcal{M}$ has the form

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

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 $f(\vec{z})$, with $f \in \Lambda$ and $\vec{z} \subseteq \vec{x}$.

Semantics of mappings

Intuitively: $\mathcal{I}$ satisfies $\Phi \rightsquigarrow \Psi$ with respect to $\mathcal{D}$ if all facts obtained by evaluating Φ over $\mathcal{D}$ and then propagating answers to Ψ, hold in $\mathcal{I}$.

Definition (Satisfaction of a mapping assertion with respect to a database)

An interpretation $\mathcal{I}$ satisfies a mapping assertion $\Phi(\vec{x}) \rightsquigarrow \Psi(\vec{t}, \vec{y})$ 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 $(s_1^{\mathcal{I}}, s_2^{\mathcal{I}}) \in P^{\mathcal{I}}$.

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

Semantics of an OBDA system

Definition (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 $\mathcal{M}$ with respect to $\mathcal{D}$, i.e., every assertion in $\mathcal{M}$ w.r.t. $\mathcal{D}$.

An OBDA system $\mathcal{O}$ is satisfiable if it admits at least one model

Semantics of an OBDA system

Definition (Model of an OBDA system)

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

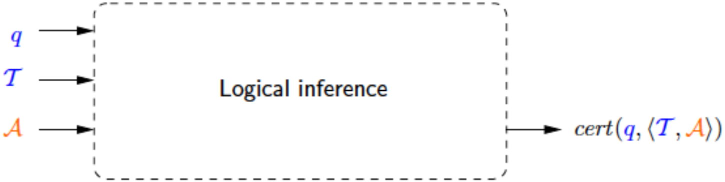
- $\mathcal{I}$ is a model of $\mathcal{T}$;
- $\mathcal{I}$ satisfies $\mathcal{M}$ with respect to $\mathcal{D}$, i.e., every assertion in $\mathcal{M}$ w.r.t. $\mathcal{D}$.

An OBDA system $\mathcal{O}$ is satisfiable if it admits at least one model

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

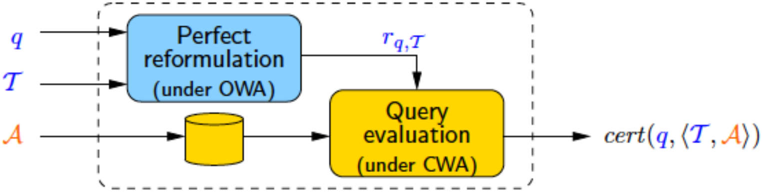
- Bottom-up approach:
  - Explicitly construct an ABox $\mathcal{A}_{\mathcal{M},\mathcal{D}}$ using $\mathcal{D}$ and $\mathcal{M}$, and compute the certain answers over $\langle \mathcal{T}, \mathcal{A}_{\mathcal{M},\mathcal{D}} \rangle$
  - Conceptually simpler, but less efficient (PTime in the data).
- 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
- OBDA with QuOnto/Quest uses the top-down approach

Top-down approach to query answering, intuition



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

Top-down approach to query answering, intuition



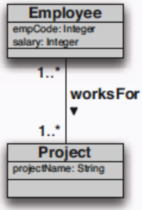
Top-down approach to query answering

- Reformulation: compute the perfect reformulation (rewriting), $q_{pr} = PerfectRef(q, \mathcal{T}_P)$, of the original query q using the inclusion assertions of the TBox $\mathcal{T}$ so that we have a UCQ.
- Unfolding: compute a new query q_{unf} from q_{pr} by using the (split version of) the mappings in $\mathcal{M}$
  - Each atom in q_{pr} that unifies with an atom in Ψ is substituted with the corresponding query Φ over the database
  - The unfolded query is such that $Eval(q_{unf}, \mathcal{D}) = Eval(q_{pr}, \mathcal{A}_{\mathcal{M},\mathcal{D}})$
- Evaluation: delegate the evaluation of q_{unf} to the relational DBMS managing $\mathcal{D}$

More examples, rewriting rules and algorithm are described on pp290-297 of the MOSS'09 slides, and more details on unfolding are on pp248-251 of the MOSS'09 slides.

Example

TBox $\mathcal{T}$ (UML)



TBox $\mathcal{T}$ (*DL-Lite*$_\mathcal{R}$)

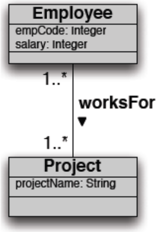
$$
\begin{array}{rcl}
\text{Employee} & \sqsubseteq & \exists\text{worksFor} \\
\exists\text{worksFor} & \sqsubseteq & \text{Employee} \\
\exists\text{worksFor}^- & \sqsubseteq & \text{Project} \\
\text{Project} & \sqsubseteq & \exists\text{worksFor}^- \\
 & \vdots &
\end{array}
$$

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

the perfect rewriting is

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

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 D1
```
⇝ Employee(**pers**(*SSN*)), Project(**proj**(*PrName*)), projectName(**proj**(*PrName*), *PrName*), worksFor(**pers**(*SSN*), **proj**(*PrName*))

m_2:
```
SELECT SSN, Salary
FROM D2, D3
WHERE D2.Code = D3.Code
```
⇝ Employee(**pers**(*SSN*)), salary(**pers**(*SSN*), *Salary*)

Example

To compute $unfold(r_{q,\mathcal{T}})$, we first **split** $\mathcal{M}$ as follows (always possible, since queries in the right-hand side of assertions in $\mathcal{M}$ are without non-distinguished variables):

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

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

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

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

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

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

Query unfolding, intuition

Then, we unify each atom of the query

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

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

$q(\mathbf{pers}(SSN)) \leftarrow$ SELECT SSN, PrName FROM D_1

$q(\mathbf{pers}(SSN)) \leftarrow$ SELECT SSN, Salary FROM D_2, D_3 WHERE D_2.CODE = D_3.CODE

The construction of object terms can be pushed into the SQL query, by resorting to SQL functions to manipulate strings (e.g., string concat).

Example

```
SELECT concat(concat('pers (',SSN),')')
FROM D1
UNION
SELECT concat(concat('pers (',SSN),')')
FROM D2, D3
WHERE D2.Code = D3.Code
```

Implementation of top-down approach to query answering

To generate an SQL query, one can follow different strategies:

- Substitute each view predicate in the unfolded queries with the corresponding SQL query over the source:
  - \+ joins are performed on the DB attributes
  - \+ does not generate doubly nested queries
  - – the number of unfolded queries may be exponential
- Construct for each atom in the original query a new view. This view takes the union of all SQL queries corresponding to the view predicates, and constructs also the Skolem terms
  - \+ avoids exponential blow-up of the resulting query, since the union (of the queries coming from multiple mappings) is done before the joins
  - – joins are performed on Skolem terms
  - – generates doubly nested queries

Which method is better, depends on various parameters

Summary

1. OBDA Options

2. Some technical details
   - Introduction
   - The ontology language
   - The mapping layer
     - 'Impedance' mismatch
     - Mapping assertions
   - Query answering