Ontology-based Data Access
A Tutorial on Query Reformulation and Optimization

Diego Calvanese

KRDB Research Centre for Knowledge and Data
Free University of Bozen-Bolzano, Italy

Seminar on Ontology Research in Brazil (ONTOBRAS)

Sãu Paulo, Brazil, 1–3 October 2018

Outline

Outline

Query answering via query reformulation

To compute the certain answers to a SPARQL query q over an OBDA instance $\mathcal{O} = \langle \mathcal{P}, \mathcal{D} \rangle$, with $\mathcal{P} = \langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$:

1. Compute the perfect rewriting of q w.r.t. $\mathcal{T}$.
2. Unfold the perfect rewriting wrt the mapping $\mathcal{M}$.
3. Optimize the unfolded query, using database constraints.
4. Evaluate the resulting SQL query over $\mathcal{D}$.

Steps 1 – 3 are collectively called **query reformulation**.

The rewriting Step 1 deals with the objects that are existentially implied by the axioms of the ontology.

unibz

Example of existential reasoning

Suppose that every graduate student is supervised by some professor, i.e.

$$\mathit{GraduateStudent} \sqsubseteq \exists \mathit{isSupervisedBy}.\mathit{Professor}$$

and john is a graduate student: $\mathit{GraduateStudent}(\mathtt{john})$.

What is the answer to the following query?

$$q(x) \leftarrow \mathit{isSupervisedBy}(x, y),\ \mathit{Professor}(y)$$

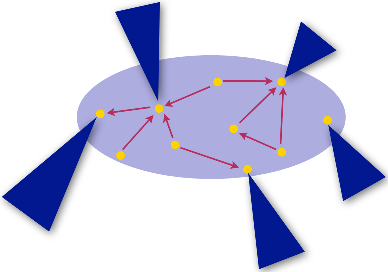
The answer should be `john`, even though we don't know who is John's supervisor (under existential reasoning).

unibz

Existential reasoning and query rewriting

Canonical model

Every consistent *DL-Lite* KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ has a **canonical model** $\mathcal{I}_{\mathcal{K}}$, which **gives the right answers to all CQs**, i.e., $\text{cert}(q, \mathcal{K}) = \text{ans}(q, \mathcal{I}_{\mathcal{K}})$



- The core part can be handled by **saturating the mapping**.
- The anonymous part can be handled by **Tree-witness rewriting**.

Example of existential reasoning (continued)

Using the (tree witness) rewriting algorithm, the query

$$q(x) \leftarrow \mathit{isSupervisedBy}(x, y),\ \mathit{Professor}(y)$$

is rewritten to a union of two conjunctive queries (or a SPARQL union query):

$$q(x) \leftarrow \mathit{isSupervisedBy}(x, y),\ \mathit{Professor}(y)$$
$$q(x) \leftarrow \mathit{GraduateStudent}(x)$$

Therefore, over the Abox $\mathit{GraduateStudent}(\mathtt{john})$, the rewritten query returns `john` as an answer.

Note: In *Ontop*, if one wants to answer queries by performing existential reasoning, the tree-witness rewriting algorithm needs to be switched on explicitly.

unibz

The *PerfectRef* algorithm for query rewriting

To illustrate Step ❶ of the query reformulation algorithm, we briefly describe *PerfectRef*, a simple query rewriting algorithm that requires to iterate over:

- rewriting steps that involve TBox inclusion assertions, and
- unification of query atoms.

The perfect rewriting of q is still a SPARQL query involving `UNION`.

Note: disjointness assertions play a role in ontology satisfiability, but can be ignored during query rewriting (i.e., we have **separability**).

unibz

Query rewriting step: Basic idea

Intuition: an inclusion assertion corresponds to a logic programming rule.

Basic rewriting step:

When an atom in the query unifies with the **head** of the rule, generate a new query by substituting the atom with the **body** of the rule.

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

Example

The inclusion assertion $\mathit{FullProf} \sqsubseteq \mathit{Prof}$
corresponds to the logic programming rule $\mathit{Prof}(z) \leftarrow \mathit{FullProf}(z)$.

Consider the query $q(x) \leftarrow \mathit{Prof}(x)$.

By applying the inclusion assertion to the atom $\mathit{Prof}(x)$, we generate:

$$q(x) \leftarrow \mathit{FullProf}(x).$$

This query is added to the input query, and contributes to the perfect rewriting.

Query rewriting (cont'd)

Example

Consider the query $q(x) \leftarrow \mathit{teaches}(x, y), \mathit{Course}(y)$

and the inclusion assertion $\exists \mathit{teaches}^- \sqsubseteq \mathit{Course}$
as a logic programming rule: $\mathit{Course}(z_2) \leftarrow \mathit{teaches}(z_1, z_2)$.

The inclusion applies to $\mathit{Course}(y)$, and we add to the rewriting the query

$$q(x) \leftarrow \mathit{teaches}(x, y), \mathit{teaches}(z_1, y).$$

Example

Consider now the query $q(x) \leftarrow \mathit{teaches}(x, y)$

and the inclusion assertion $\mathit{FullProf} \sqsubseteq \exists \mathit{teaches}$
as a logic programming rule: $\mathit{teaches}(z, f(z)) \leftarrow \mathit{FullProf}(z)$.

The inclusion applies to $\mathit{teaches}(x, y)$, and we add to the rewriting the query

$$q(x) \leftarrow \mathit{FullProf}(x).$$

Query rewriting – Constants

Example

Conversely, for the query $q(x) \leftarrow \mathit{teaches}(x, \mathtt{databases})$

and the same inclusion assertion as before $\mathit{FullProf} \sqsubseteq \exists \mathit{teaches}$
as a logic programming rule: $\mathit{teaches}(z, f(z)) \leftarrow \mathit{FullProf}(z)$

$\mathit{teaches}(x, \mathtt{databases})$ does not unify with $\mathit{teaches}(z, f(z))$, since the **skolem term** $f(z)$ in the head of the rule **does not unify** with the constant $\mathtt{databases}$.
Remember: We adopt the **unique name assumption**.

We say that the inclusion does **not** apply to the atom $\mathit{teaches}(x, \mathtt{databases})$.

Example

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

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

Query rewriting – Join variables

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

Example

Consider the query $q(x) \leftarrow \mathit{teaches}(x, y), \mathit{Course}(y)$

and the inclusion assertion $\mathit{FullProf} \sqsubseteq \exists \mathit{teaches}$
as a logic programming rule: $\mathit{teaches}(z, f(z)) \leftarrow \mathit{FullProf}(z)$.

The inclusion assertion above does **not** apply to the atom $\mathit{teaches}(x, y)$.

unibz

Query rewriting – Reduce step

Example

Consider now the query $q(x) \leftarrow \mathit{teaches}(x, y), \mathit{teaches}(z, y)$

and the inclusion assertion $\mathit{FullProf} \sqsubseteq \exists\mathit{teaches}$
as a logic rule: $\mathit{teaches}(z, f(z)) \leftarrow \mathit{FullProf}(z)$.

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

Example

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

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

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

$$q(x) \leftarrow \mathit{FullProf}(x).$$

Query rewriting – Summary

To compute the perfect rewriting of a query q, start from q, iteratively get a CQ q' to be processed, and do one of the following:

- Apply to some atom of q' an inclusion assertion in $\mathcal{T}$ as follows:

$$
\begin{array}{rccc}
A_1 \sqsubseteq A_2 & \ldots, A_2(x), \ldots & \rightsquigarrow & \ldots, A_1(x), \ldots \\
\exists P \sqsubseteq A & \ldots, A(x), \ldots & \rightsquigarrow & \ldots, P(x, \_), \ldots \\
\exists P^- \sqsubseteq A & \ldots, A(x), \ldots & \rightsquigarrow & \ldots, P(\_, x), \ldots \\
A \sqsubseteq \exists P & \ldots, P(x, \_), \ldots & \rightsquigarrow & \ldots, A(x), \ldots \\
A \sqsubseteq \exists P^- & \ldots, P(\_, x), \ldots & \rightsquigarrow & \ldots, A(x), \ldots \\
\exists P_1 \sqsubseteq \exists P_2 & \ldots, P_2(x, \_), \ldots & \rightsquigarrow & \ldots, P_1(x, \_), \ldots \\
P_1 \sqsubseteq P_2 & \ldots, P_2(x, y), \ldots & \rightsquigarrow & \ldots, P_1(x, y), \ldots \\
P_1 \sqsubseteq P_2^- & \ldots, P_2(x, y), \ldots & \rightsquigarrow & \ldots, P_1(y, x), \ldots \\
\ldots & & &
\end{array}
$$

  ('_' denotes a variable that appears only once)

- Choose two atoms of q' that unify, and apply the unifier to q'.

Each time, the result of the above step is added to the queries to be processed.

Note: Unifying atoms can make rules applicable that were not so before, and is required for completeness of the method [C. et al. 2007].

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

unibz

Query rewriting algorithm

Algorithm *PerfectRef*($Q, \mathcal{T}_P$)
Input: union of conjunctive queries Q, set $\mathcal{T}_P$ of *DL-Lite* inclusion assertions
Output: union of conjunctive queries PR
$PR := Q$;
repeat
  $PR' := PR$;
  for each $q \in PR'$ **do**
    for each g in q **do**
      for each inclusion assertion I in $\mathcal{T}_P$ **do**
        if I is applicable to g **then** $PR := PR \cup \{\, \textit{ApplyPI}(q, g, I) \,\}$;
    for each g_1, g_2 in q **do**
      if g_1 and g_2 unify **then** $PR := PR \cup \{\tau(\textit{Reduce}(q, g_1, g_2))\}$;
until $PR' = PR$;
return PR

Observations:

- Termination follows from having only finitely many different rewritings.
- Disjointness assertions and functionalities do not play any role in the rewriting of the query.

unibz

Query answering in *DL-Lite* – Example

TBox:

$FullProf \sqsubseteq Prof$
$Prof \sqsubseteq \exists teaches$
$\exists teaches^{-} \sqsubseteq Course$

Corresponding rules:

$Prof(x) \leftarrow FullProf(x)$
$\exists y(teaches(x,y)) \leftarrow Prof(x)$
$Course(x) \leftarrow teaches(y,x)$

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

Perfect rewriting:
$q(x) \leftarrow teaches(x,y), Course(y)$
$q(x) \leftarrow teaches(x,y), teaches(\_,y)$
$q(x) \leftarrow teaches(x,\_)$
$q(x) \leftarrow Prof(x)$
$q(x) \leftarrow FullProf(x)$

ABox:
teaches(jim, databases)
teaches(julia, security)
FullProf(jim)
FullProf(nicole)

Evaluating the perfect rewriting over the ABox (seen as a DB) produces as answer {**jim**, **julia**, **nicole**}.

Query answering in *DL-Lite* – An interesting example

TBox: $Person \sqsubseteq \exists hasFather$
$\exists hasFather^- \sqsubseteq Person$

ABox: $Person(\mathtt{john})$

Query: $q(x) \leftarrow Person(x), hasFather(x, y_1), hasFather(y_1, y_2), hasFather(y_2, y_3)$

$q(x) \leftarrow Person(x), hasFather(x, y_1), hasFather(y_1, y_2), hasFather(y_2, \_)$

$\Downarrow$ **Apply** $Person \sqsubseteq \exists hasFather$ to the atom $hasFather(y_2, \_)$

$q(x) \leftarrow Person(x), hasFather(x, y_1), hasFather(y_1, y_2), Person(y_2)$

$\Downarrow$ **Apply** $\exists hasFather^- \sqsubseteq Person$ to the atom $Person(y_2)$

$q(x) \leftarrow Person(x), hasFather(x, y_1), hasFather(y_1, y_2), hasFather(\_, y_2)$

$\Downarrow$ **Unify** atoms $hasFather(y_1, y_2)$ and $hasFather(\_, y_2)$

$q(x) \leftarrow Person(x), hasFather(x, y_1), hasFather(y_1, y_2)$

$\Downarrow$

...

$q(x) \leftarrow Person(x), hasFather(x, \_)$

$\Downarrow$ **Apply** $Person \sqsubseteq \exists hasFather$ to the atom $hasFather(x, \_)$

$q(x) \leftarrow Person(x)$

unibz

Complexity of query answering in *DL-Lite*

Query answering for UCQs / SPARQL queries is:

- Efficiently tractable in the size of the TBox, i.e., PTIME.
- Very efficiently tractable in the size of the ABox, i.e., AC^0.
- Exponential in the size of the **query**, more precisely NP-complete.

  In theory this is not bad, since this is precisely the complexity of evaluating CQs in plain relational DBs.

Can we go beyond *DL-Lite*?

Essentially no! By adding essentially any additional DL constructor we lose first-order rewritability and hence these nice computational properties.

Outline

Impedance mismatch

We need to address the impedance mismatch problem

- In relational databases, information is represented as tuples of values.
- In ontologies, information is represented using both objects and values ...
  - ... with objects playing the main role, ...
  - ... and values playing a subsidiary role as fillers of object attributes.

Proposed solution:

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

Note: the **data layer** (typically) is only **virtual**, since the objects are not materialized at the level of the ontology.

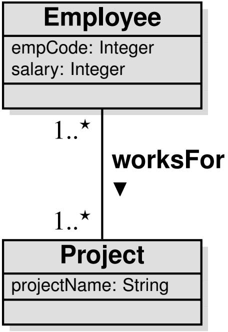
unibz

Solution to the impedance mismatch problem

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

- Each mapping assertion maps:
  - a query that retrieves values from a data source to . . .
  - a set of atoms specified over the ontology.
- Basic idea: use **Skolem functions** (or more concretely, **pattern templates**) 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*: *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*)

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.
  - Concretely, the object terms are obtained by instantiating the patterns with values from the database.

Example

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

Mapping assertions

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

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

A **mapping assertion** between a database with schema $\mathcal{S}$ and an ontology $\mathcal{O}$ has the form

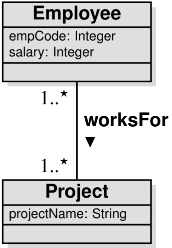
$$\Phi(\vec{x}) \rightsquigarrow \Psi(\vec{t}, \vec{y})$$

where

- Φ is an arbitrary SQL query of arity $n > 0$ over $\mathcal{S}$;
- Ψ is a conjunctive query over $\mathcal{O}$ of arity $n' > 0$ without existentially quantified 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 $\vec{z} \subseteq \vec{x}$.

unibz

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 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), $Salary$)

unibz

Concrete mapping languages

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

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

Examples: D2RQ[1], SML[2], Ontop[3]

R2RML

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

[1] http://d2rq.org/d2rq-language
[2] http://sparqlify.org/wiki/Sparqlification_mapping_language
[3] https://github.com/ontop/ontop/wiki/ontopOBDAModel#Mapping_axioms

Ontology-based data access: Formalization

To formalize OBDA, we distinguish between the intensional and the extensional level information.

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

- $\mathcal{O}$ is the (intensional level of an) ontology.
  We consider ontologies formalized in description logics (DLs), hence the intensional level is a DL TBox.
- $\mathcal{S}$ is a (possibly federated) relational database schema for the data source(s), possibly with constraints;
- $\mathcal{M}$ is a set of mapping assertions between $\mathcal{O}$ and $\mathcal{S}$.

An OBDA instance is a pair $\mathcal{J} = \langle \mathcal{P}, \mathcal{D} \rangle$, where

- $\mathcal{P} = \langle \mathcal{O}, \mathcal{M}, \mathcal{S} \rangle$ is an OBDA specification, and
- $\mathcal{D}$ is a relational database compliant with $\mathcal{S}$.

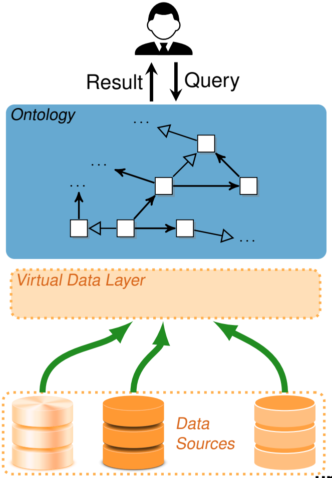
Semantics of OBDA: Intuition

In an OBDA instance $\mathcal{J} = \langle\langle\mathcal{O}, \mathcal{M}, \mathcal{S}\rangle, \mathcal{D}\rangle$, the **mapping** $\mathcal{M}$ encodes how the data $\mathcal{D}$ in the source(s) $\mathcal{S}$ should be used to populate the elements of $\mathcal{O}$.

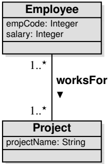
Virtual data layer

The data $\mathcal{D}$ and the mapping $\mathcal{M}$ define a virtual data layer $\mathcal{V} = \mathcal{M}(\mathcal{D})$

- Queries are answered w.r.t. $\mathcal{O}$ and $\mathcal{V}$.
- We do not really materialize the data of $\mathcal{V}$ (it is virtual!).
- Instead, the intensional information in $\mathcal{O}$ and $\mathcal{M}$ is used to translate queries over $\mathcal{O}$ into queries formulated over $\mathcal{S}$.



Virtual data layer – Example



D_1	
SSN	*PrName*
23AB	optique
...	...

D_2	
Code	*Salary*
e23	1500
...	...

D_3	
Code	*SSN*
e23	23AB
...	...

m_1: `SELECT SSN, PrName FROM D`$_1$ ⇝ *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` ⇝ *Employee*(**pers**(SSN)), *salary*(**pers**(SSN), $Salary$)

Applying m_1 and m_2 to the database, generates a virtual data layer:

Object terms: **pers**(23AB), **proj**(optique), ... Values: optique, 1500, ...

ABox assertions: *Employee*(**pers**(23AB)), ... *Project*({**proj**(optique)), ...
projectName(**proj**(optique), optique), ...
worksFor(**pers**(23AB), **proj**(optique)), ...
salary(**pers**(23AB), 1500), ...

unibz

Semantics of mappings

To formally define the semantics of an OBDA instance $\mathcal{J} = \langle \mathcal{P}, \mathcal{D} \rangle$, where $\mathcal{P} = \langle \mathcal{O}, \mathcal{M}, \mathcal{S} \rangle$, we first need to define the semantics of mappings.

Satisfaction of a mapping assertion with respect to a database

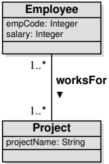
An interpretation $\mathcal{I}$ **satisfies** a mapping assertion $\Phi(\vec{x}) \rightsquigarrow \Psi(\vec{x})$ in $\mathcal{M}$ **with respect to a database** $\mathcal{D}$ **for** $\mathcal{S}$, if the following FOL sentence is true in $\mathcal{I} \cup \mathcal{D}$:

$$\forall \vec{x}. \Phi(\vec{x}) \rightarrow \Psi(\vec{x})$$

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

unibz

Semantics of mappings – Example



D_1

SSN	PrName
23AB	optique
...	...

D_2

Code	Salary
e23	1500
...	...

D_3

Code	SSN
e23	23AB
...	...

m_1: `SELECT SSN, PrName FROM D1` $\rightsquigarrow$ *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` $\rightsquigarrow$ *Employee*(**pers**(SSN)), *salary*(**pers**(SSN), $Salary$)

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

$\Delta_O^{\mathcal{I}} = \{\textbf{pers}(\texttt{23AB}), \textbf{proj}(\texttt{optique}), \ldots\}$, $\Delta_V^{\mathcal{I}} = \{\texttt{optique}, \texttt{1500}, \ldots\}$

$\mathit{Employee}^{\mathcal{I}} = \{\textbf{pers}(\texttt{23AB}), \ldots\}$, $\mathit{Project}^{\mathcal{I}} = \{\textbf{proj}(\texttt{optique}), \ldots\}$,

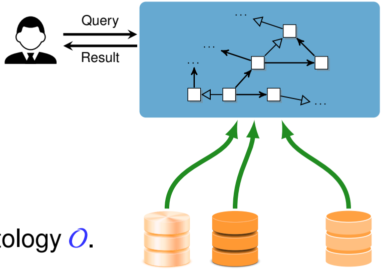
$\mathit{projectName}^{\mathcal{I}} = \{(\textbf{proj}(\texttt{optique}), \texttt{optique}), \ldots\}$,

$\mathit{worksFor}^{\mathcal{I}} = \{(\textbf{pers}(\texttt{23AB}), \textbf{proj}(\texttt{optique})), \ldots\}$,

$\mathit{salary}^{\mathcal{I}} = \{(\textbf{pers}(\texttt{23AB}), \texttt{1500}), \ldots\}$

unibz

Semantics of an OBDA instance



Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an interpretation of the ontology $\mathcal{O}$.

Model of an OBDA instance

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

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

An OBDA instance $\mathcal{J}$ is **satisfiable** if it admits at least one model.

Outline

1. Query rewriting wrt an OWL 2 QL ontology
2. Mapping specification
3. Saturation and optimization of the mapping
4. Query reformulation and optimization

unibz

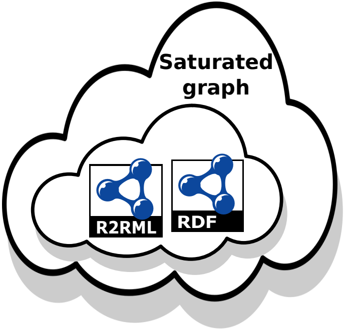
Querying the OBDA system

OBDA system $\mathcal{K} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$

- *DL-Lite*$_{\mathcal{R}}$ TBox $\mathcal{T}$
- RDF graph $\mathcal{G}$ obtained from the mapping $\mathcal{M}$ and the data sources $\mathcal{D}$
- $\mathcal{G}$ can be viewed as the ABox

Query answering

- SPARQL query $\boldsymbol{q}$ over $\mathcal{K}$
- If there is no existential restriction $B \sqsubseteq \exists R.C$ in $\mathcal{T}$, $\boldsymbol{q}$ can be directly evaluated over $\mathcal{G}_{\text{sat}}$



Saturated RDF graph $\mathcal{G}_{\text{sat}}$

- Saturation of $\mathcal{G}$ w.r.t. $\mathcal{T}$
- H-complete ABox

unibz

How to handle the RDF graph $\mathcal{G}_{sat}$ in practice?

By materializing it

- Materialization of $\mathcal{G}$ (ETL) + saturation
- – Large volume
- – Maintenance
- Typical profile: OWL 2 RL

By keeping it virtual

- Query rewriting
- + No materialization required
- Saturated mapping $\mathcal{M}_{sat}$
- Typical profile: OWL 2 QL

unibz

H-complete ABox

[Rodriguez-Muro, Kontchakov, and Zakharyaschev 2013; Kontchakov and Zakharyaschev 2014]

ABox saturation

- H-complete ABox: contains all the inferable ABox assertions
- Let $\mathcal{K}$ be a *DL-Lite*$_\mathcal{R}$ knowledge base, and let $\mathcal{K}_{\mathsf{sat}}$ be the result of saturating $\mathcal{K}$. Then, for every ABox assertion α, we have:

$$\mathcal{K} \models \alpha \qquad \text{iff} \qquad \alpha \in \mathcal{K}_{\mathsf{sat}}$$

Saturated mapping $\mathcal{M}_{\mathsf{sat}}$ (also called *T-mapping*)

- Composition of the mapping $\mathcal{M}$ and the *DL-Lite*$_\mathcal{R}$ TBox $\mathcal{T}$.
- $\mathcal{M}_{\mathsf{sat}}$ applied to $\mathcal{D}$ produces $\mathcal{G}_{\mathsf{sat}}$ (H-complete ABox).
- Does not depend of the SPARQL query $\boldsymbol{q}$ (can be pre-computed).
- Can be optimized (exploiting query containment).

TBox, user-defined mapping assertions, and foreign key

Student ⊔ *PostDoc* ⊔ *AssociateProfessor* ⊔ ∃*teaches* ⊑ *Person*

$$\textit{Student}(\textbf{iri1}(scode)) \leadsto \texttt{student}(scode, fn, ln) \quad (1)$$

$$\textit{PostDoc}(\textbf{iri2}(acode)) \leadsto \texttt{academic}(acode, fn, ln, pos),\ pos = 9 \quad (2)$$

$$\textit{AssociateProfessor}(\textbf{iri2}(acode)) \leadsto \texttt{academic}(acode, fn, ln, pos),\ pos = 2 \quad (3)$$

$$\textit{FacultyMember}(\textbf{iri2}(acode)) \leadsto \texttt{academic}(acode, fn, ln, pos) \quad (4)$$

$$\textit{teaches}(\textbf{iri2}(acode), \textbf{iri3}(course)) \leadsto \texttt{teaching}(course, acode) \quad (5)$$

FK: $\exists y_1.\texttt{teaching}(y_1, x) \rightarrow \exists y_2 y_3 y_4.\texttt{academic}(x, y_2, y_3, y_4)$

By **saturating the mapping**, we obtain mapping assertions for *Person*

$$\textit{Person}(\textbf{iri1}(scode)) \leadsto \texttt{student}(scode, fn, ln) \quad (6)$$

$$\textit{Person}(\textbf{iri2}(acode)) \leadsto \texttt{academic}(acode, fn, ln, pos),\ pos = 9 \quad (7)$$

$$\textit{Person}(\textbf{iri2}(acode)) \leadsto \texttt{academic}(acode, fn, ln, pos),\ pos = 2 \quad (8)$$

$$\textit{Person}(\textbf{iri2}(acode)) \leadsto \texttt{academic}(acode, fn, ln, pos) \quad (9)$$

$$\textit{Person}(\textbf{iri2}(acode)) \leadsto \texttt{teaching}(course, acode) \quad (10)$$

By **optimizing the mapping** using query containment and the FK, we can remove mapping assertions 7, 8, and 10

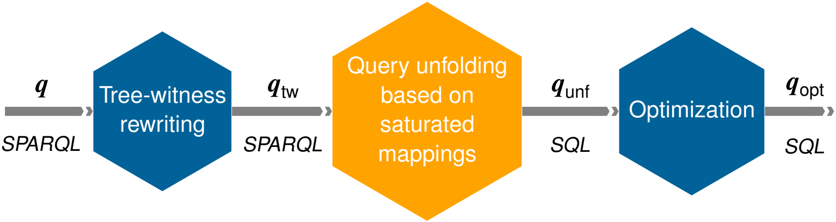
$$\textit{Person}(\textbf{iri1}(scode)) \leadsto \texttt{student}(scode, fn, ln) \quad (6)$$

$$\textit{Person}(\textbf{iri2}(acode)) \leadsto \texttt{academic}(acode, fn, ln, pos) \quad (9)$$

Outline

1. Query rewriting wrt an OWL 2 QL ontology
2. Mapping specification
3. Saturation and optimization of the mapping
4. Query reformulation and optimization

Query reformulation as implemented by Ontop



Step	Input	Output
1. Tree-witness rewriting	q (SPARQL) and $\mathcal{T}$	q_{tw} (SPARQL)
2. Query unfolding	q_{tw} and $\mathcal{M}_{sat}$	q_{unf} (SQL)
3. Query optimization	q_{unf}, primary and foreign keys	q_{opt} (SQL)

SQL query optimization

Objective : produce SQL queries that are . . .

- similar to manually written ones
- adapted to existing query planners

Structural optimization

- From join-of-unions to union-of-joins
- IRI decomposition to improve joining performance

Semantic optimization

- Redundant join elimination
- Redundant union elimination
- Using functional constraints

Integrity constraints

- Primary and foreign keys, unique constraints
- Sometimes implicit
- Vital for query reformulation!

unibz

Reformulation example – 1. Unfolding

Saturated mapping

$$\mathit{Teacher}(\mathbf{iri2}(acode)) \rightsquigarrow \texttt{academic}(acode, fn, ln, pos),\ pos \in [1..8]$$
$$\mathit{Teacher}(\mathbf{iri2}(acode)) \rightsquigarrow \texttt{teaching}(course, acode)$$
$$\mathit{firstName}(\mathbf{iri1}(scode), fn) \rightsquigarrow \texttt{student}(scode, fn, ln)$$
$$\mathit{firstName}(\mathbf{iri2}(acode), fn) \rightsquigarrow \texttt{academic}(acode, fn, ln, pos)$$
$$\mathit{lastName}(\mathbf{iri1}(scode), ln) \rightsquigarrow \texttt{student}(scode, fn, ln)$$
$$\mathit{lastName}(\mathbf{iri2}(acode), ln) \rightsquigarrow \texttt{academic}(acode, fn, ln, pos)$$

Query (we assume that the ontology is empty, hence $\boldsymbol{q}_{\text{tw}} = \boldsymbol{q}$)

$$\boldsymbol{q}(x, y, z) \leftarrow \mathit{Teacher}(x),\ \mathit{firstName}(x, y),\ \mathit{lastName}(x, z)$$

Query unfolding, and **normalization**, to make the join conditions explicit

$$\boldsymbol{q}_{\text{norm}}(x, y, z) \leftarrow \boldsymbol{q}1_{\text{unf}}(x),\ \boldsymbol{q}2_{\text{unf}}(x_1, y),\ \boldsymbol{q}3_{\text{unf}}(x_2, z),\ x = x_1,\ x = x_2$$
$$\boldsymbol{q}1_{\text{unf}}(\mathbf{iri2}(acode)) \leftarrow \texttt{academic}(acode, fn, ln, pos),\ pos \in [1..8]$$
$$\boldsymbol{q}1_{\text{unf}}(\mathbf{iri2}(acode)) \leftarrow \texttt{teaching}(course, acode)$$
$$\boldsymbol{q}2_{\text{unf}}(\mathbf{iri1}(scode), fn) \leftarrow \texttt{student}(scode, fn, ln)$$
$$\boldsymbol{q}2_{\text{unf}}(\mathbf{iri2}(acode), fn) \leftarrow \texttt{academic}(acode, fn, ln, pos)$$
$$\boldsymbol{q}3_{\text{unf}}(\mathbf{iri1}(scode), ln) \leftarrow \texttt{student}(scode, fn, ln)$$
$$\boldsymbol{q}3_{\text{unf}}(\mathbf{iri2}(acode), ln) \leftarrow \texttt{academic}(acode, fn, ln, pos)$$

Reformulation example – 2. Structural optimization

Unfolded normalized query

$$\begin{aligned}
q_{\text{norm}}(x,y,z) &\leftarrow q1_{\text{unf}}(x),\ q2_{\text{unf}}(x_1,y),\ q3_{\text{unf}}(x_2,z),\ x=x_1,\ x=x_2\\
q1_{\text{unf}}(\mathbf{iri2}(a)) &\leftarrow \texttt{academic}(a,f,l,p),\ p\in[1..8]\\
q1_{\text{unf}}(\mathbf{iri2}(a)) &\leftarrow \texttt{teaching}(c,a)\\
q2_{\text{unf}}(\mathbf{iri1}(s),f) &\leftarrow \texttt{student}(s,f,l)\\
q2_{\text{unf}}(\mathbf{iri2}(a),f) &\leftarrow \texttt{academic}(a,f,l,p)\\
q3_{\text{unf}}(\mathbf{iri1}(s),l) &\leftarrow \texttt{student}(s,f,l)\\
q3_{\text{unf}}(\mathbf{iri2}(a),l) &\leftarrow \texttt{academic}(a,f,l,p)
\end{aligned}$$

- While flattening, we can avoid to generate those queries that contain in their body an equality between two terms with incompatible IRI templates.
- This might avoid a potential exponential blowup.

Flattening (URI template lifting) – Part 1/2

$$q_{\text{lift}}(\mathbf{iri2}(a),y,z) \leftarrow \texttt{academic}(a,f_1,l_1,p_1),\ \texttt{student}(s,f_2,l_2),\ \texttt{student}(s_1,f_3,l_3),\ \mathbf{iri2}(a)=\mathbf{iri1}(s),\ \mathbf{iri2}(a)=\mathbf{iri1}(s_1),\ p_1\in[1..8]$$

$$q_{\text{lift}}(\mathbf{iri2}(a),y,z) \leftarrow \texttt{academic}(a,f_1,l_1,p_1),\ \texttt{student}(s,f_2,l_2),\ \texttt{academic}(a_2,f_3,z,p_3),\ \mathbf{iri2}(a)=\mathbf{iri1}(s),\ \mathbf{iri2}(a)=\mathbf{iri2}(a_2),\ p_1\in[1..8]$$

(One sub-query not shown)

$$q_{\text{lift}}(\mathbf{iri2}(a),y,z) \leftarrow \texttt{academic}(a,f_1,l_1,p_1),\ \texttt{academic}(a_1,y,l_2,p_2),\ \texttt{academic}(a_2,f_3,z,p_3),\ \mathbf{iri2}(a)=\mathbf{iri2}(a_1),\ \mathbf{iri2}(a)=\mathbf{iri2}(a_2),\ p_1\in[1..8]$$

unibz

Reformulation example – 2. Structural optimization

Unfolded normalized query

$$q_{\text{norm}}(x, y, z) \leftarrow q1_{\text{unf}}(x),\ q2_{\text{unf}}(x_1, y),\ q3_{\text{unf}}(x_2, z),\ x = x_1,\ x = x_2$$

$$q1_{\text{unf}}(\mathbf{iri2}(a)) \leftarrow \texttt{academic}(a, f, l, p),\ p \in [1..8]$$

$$q1_{\text{unf}}(\mathbf{iri2}(a)) \leftarrow \texttt{teaching}(c, a)$$

$$q2_{\text{unf}}(\mathbf{iri1}(s), f) \leftarrow \texttt{student}(s, f, l)$$

$$q2_{\text{unf}}(\mathbf{iri2}(a), f) \leftarrow \texttt{academic}(a, f, l, p)$$

$$q3_{\text{unf}}(\mathbf{iri1}(s), l) \leftarrow \texttt{student}(s, f, l)$$

$$q3_{\text{unf}}(\mathbf{iri2}(a), l) \leftarrow \texttt{academic}(a, f, l, p)$$

- While flattening, we can avoid to generate those queries that contain in their body an equality between two terms with incompatible IRI templates.
- This might avoid a potential exponential blowup.

Flattening (URI template lifting) – Part 2/2

$$q_{\text{lift}}(\mathbf{iri2}(a), y, z) \leftarrow \texttt{teaching}(c, a),\ \texttt{student}(s, f_2, l_2),\ \texttt{student}(s_1, f_3, l_3),\ \mathbf{iri2}(a) = \mathbf{iri1}(s),\ \mathbf{iri2}(a) = \mathbf{iri1}(s_1)$$

$$q_{\text{lift}}(\mathbf{iri2}(a), y, z) \leftarrow \texttt{teaching}(c, a),\ \texttt{student}(s, f_2, l_2),\ \texttt{academic}(a_2, f_3, z, p_3),\ \mathbf{iri2}(a) = \mathbf{iri1}(s),\ \mathbf{iri2}(a) = \mathbf{iri2}(a_2)$$

$$q_{\text{lift}}(\mathbf{iri2}(a), y, z) \leftarrow \texttt{teaching}(c, a),\ \texttt{academic}(a_1, y, l_2, p_2),\ \texttt{student}(s, f_3, l_3),\ \mathbf{iri2}(a) = \mathbf{iri2}(a_1),\ \mathbf{iri2}(a) = \mathbf{iri1}(s)$$

$$q_{\text{lift}}(\mathbf{iri2}(a), y, z) \leftarrow \texttt{teaching}(c, a),\ \texttt{academic}(a_1, y, l_2, p_2),\ \texttt{academic}(a_2, f_3, z, p_3),\ \mathbf{iri2}(a) = \mathbf{iri2}(a_1),\ \mathbf{iri2}(a) = \mathbf{iri2}(a_2)$$

Reformulation example – 3. Semantic optimization

We are left with just two queries, that we can simplify by eliminating equalities

$$\begin{aligned} q_{\text{struct}}(\textbf{iri2}(a), y, z) \leftarrow\ & \texttt{academic}(a, f_1, l_1, p_1),\ p_1 \in [1..8], \\ & \texttt{academic}(a, y, l_2, p_2), \\ & \texttt{academic}(a, f_3, z, p_3) \end{aligned}$$

$$\begin{aligned} q_{\text{struct}}(\textbf{iri2}(a), y, z) \leftarrow\ & \texttt{teaching}(c, a), \\ & \texttt{academic}(a, y, l_2, p_2), \\ & \texttt{academic}(a, f_3, z, p_3) \end{aligned}$$

We can then exploit database constraints (such as primary keys) for semantic optimization of the query.

Self-join elimination (semantic optimization)

PK: $\texttt{academic}(acode, f, l, p) \land \texttt{academic}(acode, f', l', p') \rightarrow (f = f') \land (l = l') \land (p = p')$

$$q_{\text{opt}}(\textbf{iri2}(a), y, z) \leftarrow \texttt{academic}(a, y, z, p_1),\ p_1 \in [1..8]$$

$$q_{\text{opt}}(\textbf{iri2}(a), y, z) \leftarrow \texttt{teaching}(c, a),\ \texttt{academic}(a, y, z, p_2)$$

References I

[1] Diego C. et al. "Tractable Reasoning and Efficient Query Answering in Description Logics: The *DL-Lite* Family". In: *J. of Automated Reasoning* 39.3 (2007), pp. 385–429.

[2] Mariano Rodriguez-Muro, Roman Kontchakov, and Michael Zakharyaschev. "Ontology-Based Data Access: Ontop of Databases". In: *Proc. of ISWC*. Vol. 8218. LNCS. 2013, pp. 558–573. DOI: 10.1007/978-3-642-41335-3_35.

[3] Roman Kontchakov and Michael Zakharyaschev. "An Introduction to Description Logics and Query Rewriting". In: *RW 2014 Tutorial Lectures*. Vol. 8714. LNCS. Springer, 2014, pp. 195–244. DOI: 10.1007/978-3-319-10587-1_5.

unibz