Semantic Technologies for Data Access and Integration

Part 3: Query Processing and Optimization

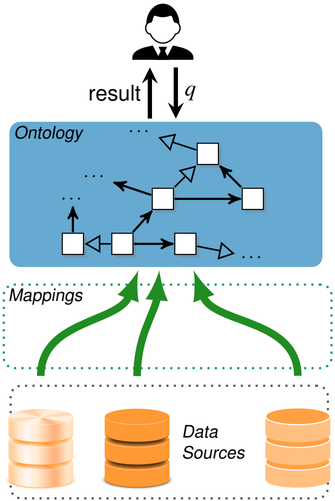
Diego Calvanese, Guohui Xiao

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

27th ACM International Conference on Information and Knowledge Management (CIKM)

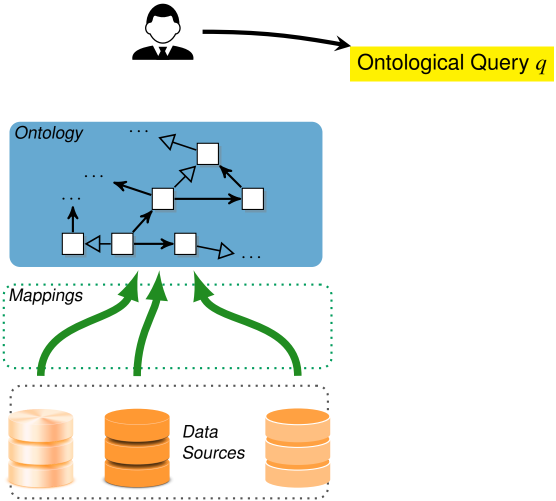
Torino, Italy, 22–26 October 2018

Query answering by query rewriting



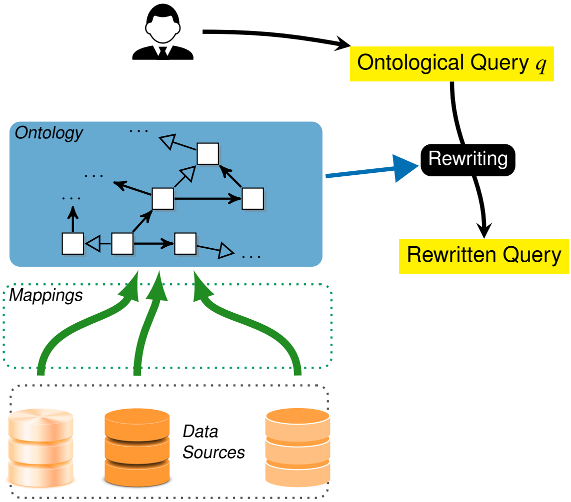
unibz

Query answering by query rewriting

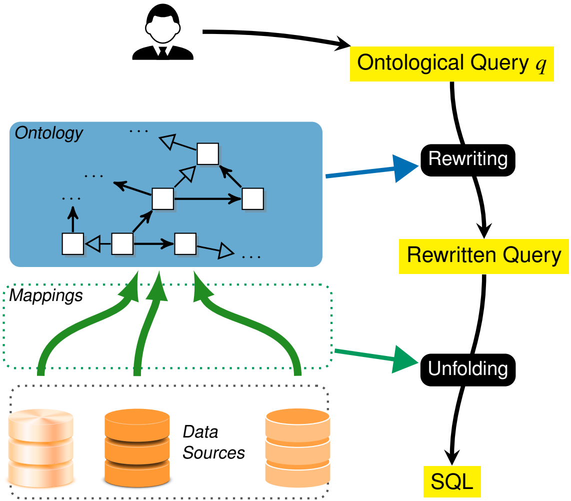


unibz

Query answering by query rewriting

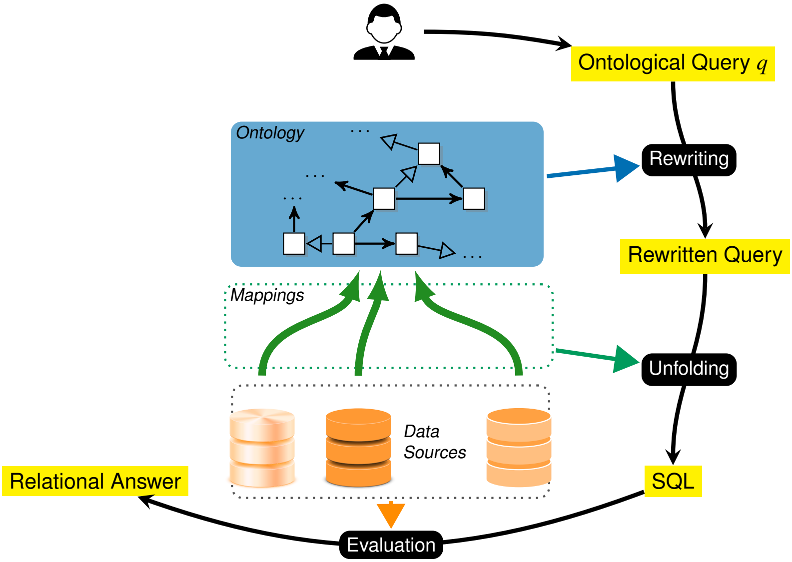


Query answering by query rewriting



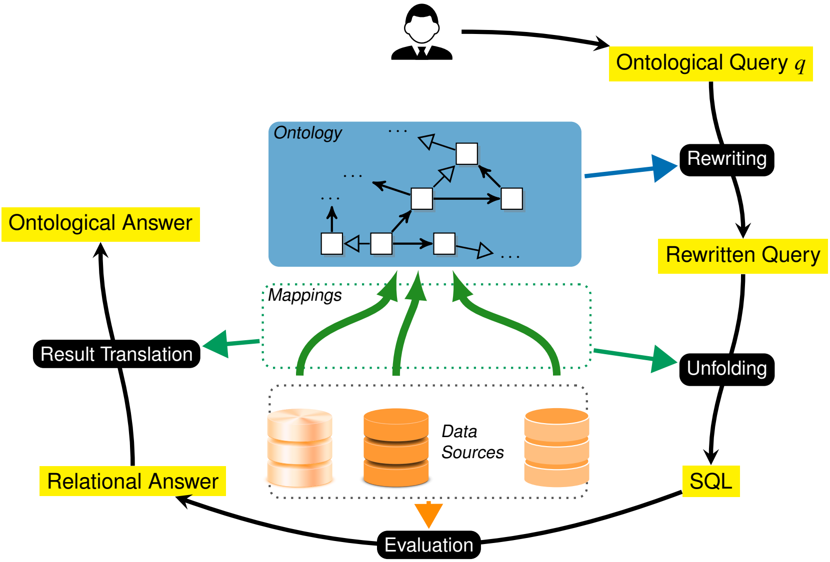
unibz

Query answering by query rewriting

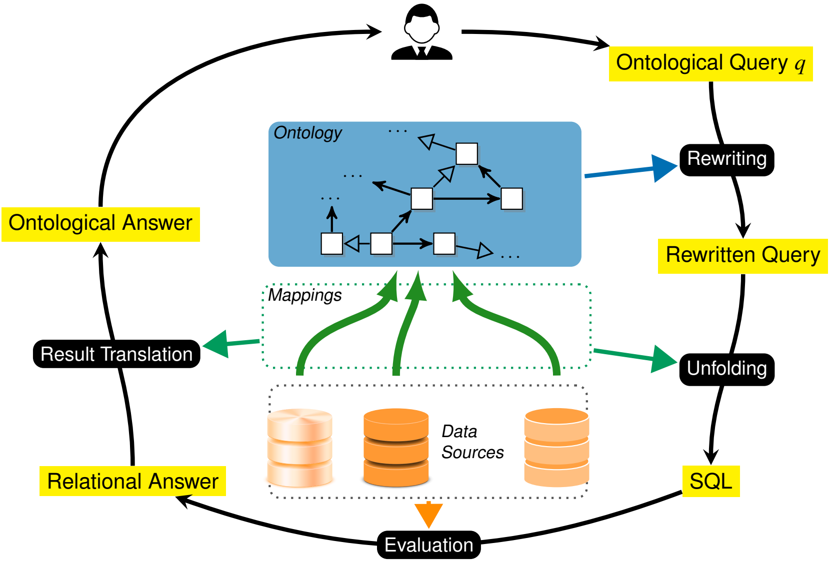


unibz

Query answering by query rewriting



Query answering by query rewriting



unibz

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**.

unibz

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.

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: $\mathit{GraduateStudent}(\mathtt{john})$.

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?

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

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?

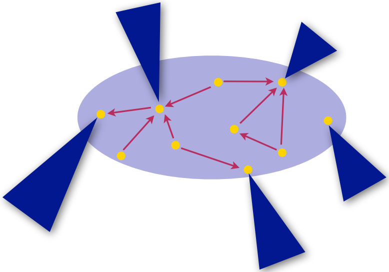
$$q(x) \leftarrow \mathit{isSupervisedBy}(x, y),\ \mathit{Professor}(y)$$

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

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., cert$(q, \mathcal{K})$ = ans$(q, \mathcal{I}_{\mathcal{K}})$

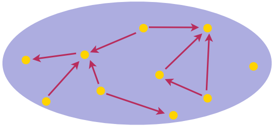


Existential reasoning and query rewriting

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., cert$(q, \mathcal{K})$ = ans$(q, \mathcal{I}_{\mathcal{K}})$

Core
individuals
from $\mathcal{A}$



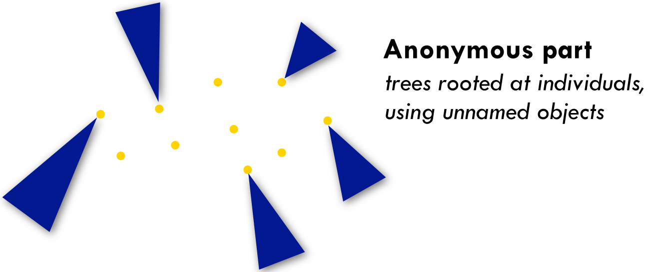
- The core part can be handled by **saturating the mapping**.

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., cert$(q, \mathcal{K})$ = ans$(q, \mathcal{I}_\mathcal{K})$

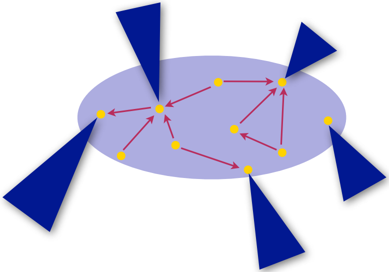


- The core part can be handled by **saturating the mapping**.
- The anonymous part can be handled by **Tree-witness rewriting**.

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., cert$(q, \mathcal{K})$ = ans$(q, \mathcal{I}_{\mathcal{K}})$



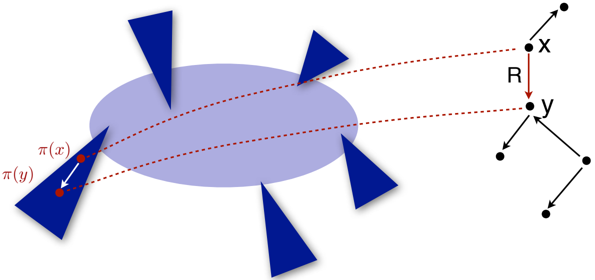
- The core part can be handled by **saturating the mapping**.
- The anonymous part can be handled by **Tree-witness rewriting**.

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., cert$(q, \mathcal{K})$ = ans$(q, \mathcal{I}_{\mathcal{K}})$



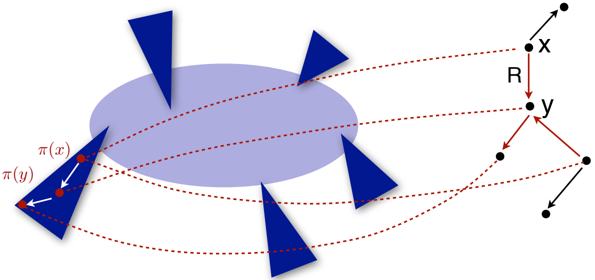
- The core part can be handled by **saturating the mapping**.
- The anonymous part can be handled by **Tree-witness rewriting**.

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., cert$(q, \mathcal{K})$ = 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**.

unibz

Example of existential reasoning (continued)

Using the (tree witness) rewriting algorithm, the query

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

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):

$$\begin{array}{lcl} q(x) & \leftarrow & \mathit{isSupervisedBy}(x, y),\ \mathit{Professor}(y) \\ q(x) & \leftarrow & \mathit{GraduateStudent}(x) \end{array}$$

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

unibz

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):

$$\begin{aligned} q(x) &\leftarrow \mathit{isSupervisedBy}(x, y),\ \mathit{Professor}(y) \\ q(x) &\leftarrow \mathit{GraduateStudent}(x) \end{aligned}$$

Therefore, over the Abox $\mathit{GraduateStudent}(\texttt{john})$, 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.

The *PerfectRef* algorithm for query rewriting

To illustrate Step 1 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`.

unibz

The *PerfectRef* algorithm for query rewriting

To illustrate Step 1 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**).

Query rewriting step: Basic idea

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

Example

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

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 $\mathit{Prof}(z) \leftarrow \mathit{FullProf}(z)$.

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

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

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

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 $\quad q(x) \leftarrow \mathit{teaches}(x, \mathtt{databases})$

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

$\mathit{teaches}(x, \mathtt{databases})$ 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**.

unibz

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 $\mathit{teaches}(x, \mathtt{databases})$.

unibz

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

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

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

unibz

Query rewriting – Reduce step

Example

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

and the inclusion assertion $\quad \mathit{FullProf} \sqsubseteq \exists\mathit{teaches}$

as a logic rule: $\quad \mathit{teaches}(z, f(z)) \leftarrow \mathit{FullProf}(z).$

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.

Query rewriting – Reduce step

Example

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

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

This inclusion assertion does not apply to $teaches(x, y)$ or $teaches(z, y)$, 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 $teaches(x, y)$ and $teaches(z, y)$.
This rewriting step is called **reduce**, and produces the query

$$q(x) \leftarrow teaches(x, y).$$

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

$$q(x) \leftarrow FullProf(x).$$

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:

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

  ('_' 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.

unibz

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:

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

  ('_' 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].

unibz

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:

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

  ('_' 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** $PR := PR \cup \{$ *ApplyPI*$(q, g, I) \}$;
    for each g_1, g_2 in q **do**
      if g_1 and g_2 unify **then** $PR := PR \cup \{\tau($*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:

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

Corresponding rules:

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

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

unibz

Query answering in *DL-Lite* – Example

TBox:

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

Corresponding rules:

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

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

Perfect rewriting: $q(x) \leftarrow \mathit{teaches}(x, y), \mathit{Course}(y)$

Query answering in *DL-Lite* – Example

TBox:

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

Corresponding rules:

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

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

Perfect rewriting: $q(x) \leftarrow \mathit{teaches}(x, y), \mathit{Course}(y)$
$q(x) \leftarrow \mathit{teaches}(x, y), \mathit{teaches}(\_, y)$

unibz

Query answering in *DL-Lite* – Example

TBox:

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

Corresponding rules:

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

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

Perfect rewriting: $q(x) \leftarrow \mathit{teaches}(x,y), \mathit{Course}(y)$

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

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

unibz

Query answering in *DL-Lite* – Example

TBox:

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

Corresponding rules:

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

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

Perfect rewriting: $q(x) \leftarrow \mathit{teaches}(x,y), \mathit{Course}(y)$

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

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

$q(x) \leftarrow \mathit{Prof}(x)$

unibz

Query answering in *DL-Lite* – Example

TBox:

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

Corresponding rules:

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

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

Perfect rewriting:

$$q(x) \leftarrow \mathit{teaches}(x, y), \mathit{Course}(y)$$
$$q(x) \leftarrow \mathit{teaches}(x, y), \mathit{teaches}(\_, y)$$
$$q(x) \leftarrow \mathit{teaches}(x, \_)$$
$$q(x) \leftarrow \mathit{Prof}(x)$$
$$q(x) \leftarrow \mathit{FullProf}(x)$$

unibz

Query answering in *DL-Lite* – Example

TBox:

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

Corresponding rules:

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

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

Perfect rewriting:

$$q(x) \leftarrow \mathit{teaches}(x, y), \mathit{Course}(y)$$
$$q(x) \leftarrow \mathit{teaches}(x, y), \mathit{teaches}(\_, y)$$
$$q(x) \leftarrow \mathit{teaches}(x, \_)$$
$$q(x) \leftarrow \mathit{Prof}(x)$$
$$q(x) \leftarrow \mathit{FullProf}(x)$$

ABox: $\mathit{teaches}(\mathtt{jim}, \mathtt{databases})$ $\quad \mathit{FullProf}(\mathtt{jim})$
$\mathit{teaches}(\mathtt{julia}, \mathtt{security})$ $\quad \mathit{FullProf}(\mathtt{nicole})$

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

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

TBox: $\mathit{Person} \sqsubseteq \exists \mathit{hasFather}$
$\exists \mathit{hasFather}^- \sqsubseteq \mathit{Person}$

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

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

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

TBox: $\mathit{Person} \sqsubseteq \exists \mathit{hasFather}$
$\exists \mathit{hasFather}^- \sqsubseteq \mathit{Person}$

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

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

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

unibz

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

TBox: $\mathit{Person} \sqsubseteq \exists \mathit{hasFather}$
$\exists \mathit{hasFather}^{-} \sqsubseteq \mathit{Person}$

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

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

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

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

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

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

TBox: $\mathit{Person} \sqsubseteq \exists \mathit{hasFather}$
$\exists \mathit{hasFather}^{-} \sqsubseteq \mathit{Person}$

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

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

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

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

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

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

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

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

TBox: $\mathit{Person} \sqsubseteq \exists \mathit{hasFather}$
$\exists \mathit{hasFather}^{-} \sqsubseteq \mathit{Person}$

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

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

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

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

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

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

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

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

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

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

TBox: $\mathit{Person} \sqsubseteq \exists \mathit{hasFather}$
$\exists \mathit{hasFather}^- \sqsubseteq \mathit{Person}$

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

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

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

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

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

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

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

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

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

$\Downarrow$

...

$q(x) \leftarrow \mathit{Person}(x), \mathit{hasFather}(x, \_)$

unibz

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

TBox: $\mathit{Person} \sqsubseteq \exists \mathit{hasFather}$
$\exists \mathit{hasFather}^{-} \sqsubseteq \mathit{Person}$

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

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

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

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

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

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

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

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

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

$\Downarrow$

...

$q(x) \leftarrow \mathit{Person}(x), \mathit{hasFather}(x, \_)$

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

$q(x) \leftarrow \mathit{Person}(x)$

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.

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.

unibz

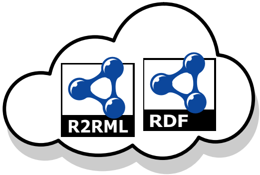
Outline

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



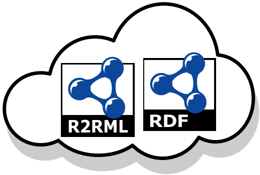
Querying the OBDA system

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 q over $\mathcal{K}$



unibz

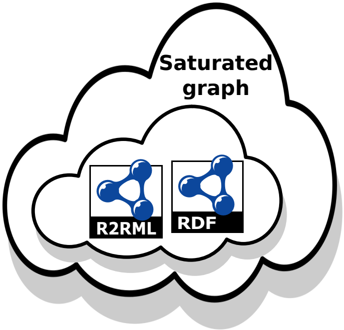
Querying the OBDA system

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 q over $\mathcal{K}$



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

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

unibz

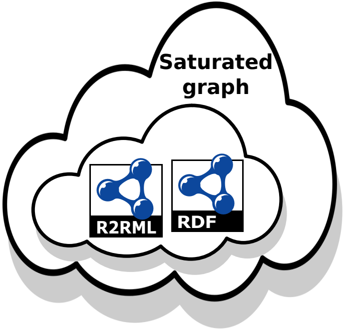
Querying the OBDA system

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 q over $\mathcal{K}$
- If there is no existential restriction $B \sqsubseteq \exists R.C$ in $\mathcal{T}$, 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

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

By materializing it

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

unibz

How to handle the RDF graph $\mathcal{G}_{\text{sat}}$ 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 $\mathcal{M}_{\text{sat}}$
- 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

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:

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

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).

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

$\mathit{Student} \sqsubseteq \mathit{Person}$	$\mathit{Student}(\textbf{iri1}(scode)) \leftsquigarrow \texttt{student}(scode, fn, ln)$	(1)
$\mathit{PostDoc} \sqsubseteq \mathit{Person}$	$\mathit{PostDoc}(\textbf{iri2}(acode)) \leftsquigarrow \texttt{academic}(acode, fn, ln, pos),\ pos = 9$	(2)
$\mathit{AssociateProfessor} \sqsubseteq \mathit{Person}$	$\mathit{AssociateProfessor}(\textbf{iri2}(acode)) \leftsquigarrow \texttt{academic}(acode, fn, ln, pos),\ pos = 2$	(3)
$\exists \mathit{teaches} \sqsubseteq \mathit{Person}$	$\mathit{FacultyMember}(\textbf{iri2}(acode)) \leftsquigarrow \texttt{academic}(acode, fn, ln, pos)$	(4)
	$\mathit{teaches}(\textbf{iri2}(acode), \textbf{iri3}(course)) \leftsquigarrow \texttt{teaching}(course, acode)$	(5)

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

TBox, user-defined mapping assertions, and foreign key

$\mathit{Student} \sqsubseteq \mathit{Person}$	$\mathit{Student}(\textbf{iri1}(scode)) \rightsquigarrow \texttt{student}(scode, fn, ln)$	(1)
$\mathit{PostDoc} \sqsubseteq \mathit{Person}$	$\mathit{PostDoc}(\textbf{iri2}(acode)) \rightsquigarrow \texttt{academic}(acode, fn, ln, pos),\ pos = 9$	(2)
$\mathit{AssociateProfessor} \sqsubseteq \mathit{Person}$	$\mathit{AssociateProfessor}(\textbf{iri2}(acode)) \rightsquigarrow \texttt{academic}(acode, fn, ln, pos),\ pos = 2$	(3)
$\exists \mathit{teaches} \sqsubseteq \mathit{Person}$	$\mathit{FacultyMember}(\textbf{iri2}(acode)) \rightsquigarrow \texttt{academic}(acode, fn, ln, pos)$	(4)
	$\mathit{teaches}(\textbf{iri2}(acode), \textbf{iri3}(course)) \rightsquigarrow \texttt{teaching}(course, acode)$	(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*

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

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

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

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

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

unibz

TBox, user-defined mapping assertions, and foreign key

$\mathit{Student} \sqsubseteq \mathit{Person}$	$\mathit{Student}(\mathbf{iri1}(scode)) \rightsquigarrow \texttt{student}(scode, fn, ln)$	(1)
$\mathit{PostDoc} \sqsubseteq \mathit{Person}$	$\mathit{PostDoc}(\mathbf{iri2}(acode)) \rightsquigarrow \texttt{academic}(acode, fn, ln, pos),\ pos = 9$	(2)
$\mathit{AssociateProfessor} \sqsubseteq \mathit{Person}$	$\mathit{AssociateProfessor}(\mathbf{iri2}(acode)) \rightsquigarrow \texttt{academic}(acode, fn, ln, pos),\ pos = 2$	(3)
$\exists \mathit{teaches} \sqsubseteq \mathit{Person}$	$\mathit{FacultyMember}(\mathbf{iri2}(acode)) \rightsquigarrow \texttt{academic}(acode, fn, ln, pos)$	(4)
	$\mathit{teaches}(\mathbf{iri2}(acode), \mathbf{iri3}(course)) \rightsquigarrow \texttt{teaching}(course, acode)$	(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*

$$\mathit{Person}(\mathbf{iri1}(scode)) \rightsquigarrow \texttt{student}(scode, fn, ln) \quad (6)$$

$$\mathit{Person}(\mathbf{iri2}(acode)) \rightsquigarrow \texttt{academic}(acode, fn, ln, pos),\ pos = 9 \quad (7)$$

$$\mathit{Person}(\mathbf{iri2}(acode)) \rightsquigarrow \texttt{academic}(acode, fn, ln, pos),\ pos = 2 \quad (8)$$

$$\mathit{Person}(\mathbf{iri2}(acode)) \rightsquigarrow \texttt{academic}(acode, fn, ln, pos) \quad (9)$$

$$\mathit{Person}(\mathbf{iri2}(acode)) \rightsquigarrow \texttt{teaching}(course, acode) \quad (10)$$

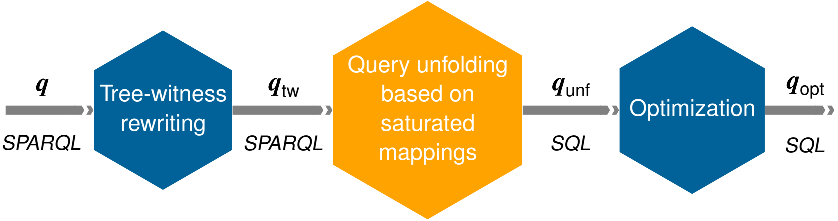
By **optimizing the mapping** using query containment and the FK, we can remove mapping assertions 7, 8, and 10

$$\mathit{Person}(\mathbf{iri1}(scode)) \rightsquigarrow \texttt{student}(scode, fn, ln) \quad (6)$$

$$\mathit{Person}(\mathbf{iri2}(acode)) \rightsquigarrow \texttt{academic}(acode, fn, ln, pos) \quad (9)$$

Outline

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

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

unibz

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

unibz

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!

Reformulation example – 1. Unfolding

Saturated mapping

$$\mathit{Teacher}(\mathbf{iri2}(acode)) \leadsto \texttt{academic}(acode, fn, ln, pos),\ pos \in [1..8]$$

$$\mathit{Teacher}(\mathbf{iri2}(acode)) \leadsto \texttt{teaching}(course, acode)$$

$$\mathit{firstName}(\mathbf{iri1}(scode), fn) \leadsto \texttt{student}(scode, fn, ln)$$

$$\mathit{firstName}(\mathbf{iri2}(acode), fn) \leadsto \texttt{academic}(acode, fn, ln, pos)$$

$$\mathit{lastName}(\mathbf{iri1}(scode), ln) \leadsto \texttt{student}(scode, fn, ln)$$

$$\mathit{lastName}(\mathbf{iri2}(acode), ln) \leadsto \texttt{academic}(acode, fn, ln, pos)$$

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

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

Reformulation example – 1. Unfolding

Saturated mapping

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

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

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

Query unfolding

$$\begin{aligned}
\boldsymbol{q}_{\mathsf{unf}}(x, y, z) &\leftarrow \boldsymbol{q}1_{\mathsf{unf}}(x),\ \boldsymbol{q}2_{\mathsf{unf}}(x, y),\\
&\quad\ \boldsymbol{q}3_{\mathsf{unf}}(x, z)\\
\boldsymbol{q}1_{\mathsf{unf}}(\mathbf{iri2}(acode)) &\leftarrow \mathtt{academic}(acode, fn, ln, pos),\\
&\quad\ pos \in [1..8]\\
\boldsymbol{q}1_{\mathsf{unf}}(\mathbf{iri2}(acode)) &\leftarrow \mathtt{teaching}(course, acode)\\
\boldsymbol{q}2_{\mathsf{unf}}(\mathbf{iri1}(scode), fn) &\leftarrow \mathtt{student}(scode, fn, ln)\\
\boldsymbol{q}2_{\mathsf{unf}}(\mathbf{iri2}(acode), fn) &\leftarrow \mathtt{academic}(acode, fn, ln, pos)\\
\boldsymbol{q}3_{\mathsf{unf}}(\mathbf{iri1}(scode), ln) &\leftarrow \mathtt{student}(scode, fn, ln)\\
\boldsymbol{q}3_{\mathsf{unf}}(\mathbf{iri2}(acode), ln) &\leftarrow \mathtt{academic}(acode, fn, ln, pos)
\end{aligned}$$

unibz

Reformulation example – 1. Unfolding

Saturated mapping

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

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

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

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

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

unibz

Reformulation example – 2. Structural optimization

Unfolded normalized query

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

unibz

Reformulation example – 2. Structural optimization

Unfolded normalized query

$$\begin{aligned}
\boldsymbol{q}_{\mathsf{norm}}(x,y,z) &\leftarrow \boldsymbol{q}1_{\mathsf{unf}}(x),\ \boldsymbol{q}2_{\mathsf{unf}}(x_1,y),\\
&\quad\ \boldsymbol{q}3_{\mathsf{unf}}(x_2,z),\\
&\quad\ x=x_1,\ x=x_2\\
\boldsymbol{q}1_{\mathsf{unf}}(\mathbf{iri2}(a)) &\leftarrow \mathtt{academic}(a,f,l,p),\\
&\quad\ p\in[1..8]\\
\boldsymbol{q}1_{\mathsf{unf}}(\mathbf{iri2}(a)) &\leftarrow \mathtt{teaching}(c,a)\\
\boldsymbol{q}2_{\mathsf{unf}}(\mathbf{iri1}(s),f) &\leftarrow \mathtt{student}(s,f,l)\\
\boldsymbol{q}2_{\mathsf{unf}}(\mathbf{iri2}(a),f) &\leftarrow \mathtt{academic}(a,f,l,p)\\
\boldsymbol{q}3_{\mathsf{unf}}(\mathbf{iri1}(s),l) &\leftarrow \mathtt{student}(s,f,l)\\
\boldsymbol{q}3_{\mathsf{unf}}(\mathbf{iri2}(a),l) &\leftarrow \mathtt{academic}(a,f,l,p)
\end{aligned}$$

Flattening (URI template lifting) – Part 1/2

$$\begin{aligned}
\boldsymbol{q}_{\mathsf{lift}}(\mathbf{iri2}(a),y,z) &\leftarrow \mathtt{academic}(a,f_1,l_1,p_1),\\
&\quad\ \mathtt{student}(s,f_2,l_2),\\
&\quad\ \mathtt{student}(s_1,f_3,l_3),\\
&\quad\ \mathbf{iri2}(a)=\mathbf{iri1}(s),\\
&\quad\ \mathbf{iri2}(a)=\mathbf{iri1}(s_1),\\
&\quad\ p_1\in[1..8]\\
\boldsymbol{q}_{\mathsf{lift}}(\mathbf{iri2}(a),y,z) &\leftarrow \mathtt{academic}(a,f_1,l_1,p_1),\\
&\quad\ \mathtt{student}(s,f_2,l_2),\\
&\quad\ \mathtt{academic}(a_2,f_3,z,p_3),\\
&\quad\ \mathbf{iri2}(a)=\mathbf{iri1}(s),\\
&\quad\ \mathbf{iri2}(a)=\mathbf{iri2}(a_2),\\
&\quad\ p_1\in[1..8]
\end{aligned}$$

(One sub-query not shown)

$$\begin{aligned}
\boldsymbol{q}_{\mathsf{lift}}(\mathbf{iri2}(a),y,z) &\leftarrow \mathtt{academic}(a,f_1,l_1,p_1),\\
&\quad\ \mathtt{academic}(a_1,y,l_2,p_2),\\
&\quad\ \mathtt{academic}(a_2,f_3,z,p_3),\\
&\quad\ \mathbf{iri2}(a)=\mathbf{iri2}(a_1),\\
&\quad\ \mathbf{iri2}(a)=\mathbf{iri2}(a_2),\\
&\quad\ p_1\in[1..8]
\end{aligned}$$

Reformulation example – 2. Structural optimization

Unfolded normalized query

$$
\begin{aligned}
\boldsymbol{q}_{\text{norm}}(x,y,z) &\leftarrow \boldsymbol{q}1_{\text{unf}}(x),\ \boldsymbol{q}2_{\text{unf}}(x_1,y),\\
&\ \boldsymbol{q}3_{\text{unf}}(x_2,z),\\
&\ x = x_1,\ x = x_2\\
\boldsymbol{q}1_{\text{unf}}(\textbf{iri2}(a)) &\leftarrow \texttt{academic}(a,f,l,p),\\
&\ p \in [1..8]\\
\boldsymbol{q}1_{\text{unf}}(\textbf{iri2}(a)) &\leftarrow \texttt{teaching}(c,a)\\
\boldsymbol{q}2_{\text{unf}}(\textbf{iri1}(s),f) &\leftarrow \texttt{student}(s,f,l)\\
\boldsymbol{q}2_{\text{unf}}(\textbf{iri2}(a),f) &\leftarrow \texttt{academic}(a,f,l,p)\\
\boldsymbol{q}3_{\text{unf}}(\textbf{iri1}(s),l) &\leftarrow \texttt{student}(s,f,l)\\
\boldsymbol{q}3_{\text{unf}}(\textbf{iri2}(a),l) &\leftarrow \texttt{academic}(a,f,l,p)
\end{aligned}
$$

- 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 2/2

$$
\begin{aligned}
\boldsymbol{q}_{\text{lift}}(\textbf{iri2}(a),y,z) &\leftarrow \texttt{teaching}(c,a),\\
&\ \texttt{student}(s,f_2,l_2),\\
&\ \texttt{student}(s_1,f_3,l_3),\\
&\ \textbf{iri2}(a) = \textbf{iri1}(s),\\
&\ \textbf{iri2}(a) = \textbf{iri1}(s_1)
\end{aligned}
$$

$$
\begin{aligned}
\boldsymbol{q}_{\text{lift}}(\textbf{iri2}(a),y,z) &\leftarrow \texttt{teaching}(c,a),\\
&\ \texttt{student}(s,f_2,l_2),\\
&\ \texttt{academic}(a_2,f_3,z,p_3),\\
&\ \textbf{iri2}(a) = \textbf{iri1}(s),\\
&\ \textbf{iri2}(a) = \textbf{iri2}(a_2)
\end{aligned}
$$

(One sub-query not shown)

$$
\begin{aligned}
\boldsymbol{q}_{\text{lift}}(\textbf{iri2}(a),y,z) &\leftarrow \texttt{teaching}(c,a),\\
&\ \texttt{academic}(a_1,y,l_2,p_2),\\
&\ \texttt{academic}(a_2,f_3,z,p_3),\\
&\ \textbf{iri2}(a) = \textbf{iri2}(a_1),\\
&\ \textbf{iri2}(a) = \textbf{iri2}(a_2)
\end{aligned}
$$

unibz

Reformulation example – 3. Semantic optimization

We are left with just two queries

$$\begin{array}{rcl} \boldsymbol{q}_{\mathsf{lift}}(\mathbf{iri2}(a), y, z) & \leftarrow & \mathtt{academic}(a, f_1, l_1, p_1),\ \ p_1 \in [1..8], \\ & & \mathtt{academic}(a_1, y, l_2, p_2),\ \mathbf{iri2}(a) = \mathbf{iri2}(a_1), \\ & & \mathtt{academic}(a_2, f_3, z, p_3),\ \mathbf{iri2}(a) = \mathbf{iri2}(a_2) \\ \boldsymbol{q}_{\mathsf{lift}}(\mathbf{iri2}(a), y, z) & \leftarrow & \mathtt{teaching}(c, a), \\ & & \mathtt{academic}(a_1, y, l_2, p_2),\ \mathbf{iri2}(a) = \mathbf{iri2}(a_1), \\ & & \mathtt{academic}(a_2, f_3, z, p_3),\ \mathbf{iri2}(a) = \mathbf{iri2}(a_2) \end{array}$$

unibz

Reformulation example – 3. Semantic optimization

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

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

Reformulation example – 3. Semantic optimization

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

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

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

We can then exploit database constraints (e.g., primary keys) for semantic optimization of the query.

Self-join elimination (semantic optimization)

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

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 (e.g., 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