Mapping Management and Expressive Ontologies in Ontology-Based Data Access

3. Theoretical Foundations of OBDA

Diego Calvanese, Benjamin Cogrel, Guohui Xiao

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

unibz
Freie Universität Bozen
Libera Università di Bolzano
Free University of Bozen-Bolzano

20th Int. Conf. on Knowledge Engineering and Knowledge Management (EKAW)
Bologna, 19 November 2016

Outline

unibz

Outline

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

unibz

Query answering via query rewriting

Query answering can be done via **query rewriting**

Given a (U)CQ q and an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$:

1. Compute the perfect rewriting of q w.r.t. $\mathcal{T}$, which is a FOL query.
2. Evaluate the perfect rewriting over $\mathcal{A}$. (We are ignoring the mapping.)

unibz

Query answering via query rewriting

Query answering can be done via query rewriting

Given a (U)CQ q and an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$:

1. Compute the perfect rewriting of q w.r.t. $\mathcal{T}$, which is a FOL query.
2. Evaluate the perfect rewriting over $\mathcal{A}$. (We are ignoring the mapping.)

I briefly describe *PerfectRef*, a simple algorithm for Step 1 that requires to iterate over:

- rewriting steps that involve inclusion assertions, and
- unification steps.

Query answering via query rewriting

Query answering can be done via query rewriting

Given a (U)CQ q and an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$:

1. Compute the perfect rewriting of q w.r.t. $\mathcal{T}$, which is a FOL query.
2. Evaluate the perfect rewriting over $\mathcal{A}$. (We are ignoring the mapping.)

I briefly describe *PerfectRef*, a simple algorithm for Step 1 that requires to iterate over:

- rewriting steps that involve inclusion assertions, and
- unification steps.

Note: disjointness assertions and functionalities 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.

Example

The inclusion assertion $\mathsf{MovieActor} \sqsubseteq \mathsf{Actor}$
corresponds to the logic programming rule $\mathsf{Actor}(z) \leftarrow \mathsf{MovieActor}(z)$.

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 $\mathsf{MovieActor} \sqsubseteq \mathsf{Actor}$

corresponds to the logic programming rule $\mathsf{Actor}(z) \leftarrow \mathsf{MovieActor}(z)$.

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 $\mathsf{MovieActor} \sqsubseteq \mathsf{Actor}$

corresponds to the logic programming rule $\mathsf{Actor}(z) \leftarrow \mathsf{MovieActor}(z)$.

Consider the query $\mathsf{q}(x) \leftarrow \mathsf{Actor}(x)$.

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 $\mathsf{MovieActor} \sqsubseteq \mathsf{Actor}$
corresponds to the logic programming rule $\mathsf{Actor}(z) \leftarrow \mathsf{MovieActor}(z)$.

Consider the query $\mathsf{q}(x) \leftarrow \mathsf{Actor}(x)$.

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

$$\mathsf{q}(x) \leftarrow \mathsf{MovieActor}(x).$$

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

Query rewriting (cont'd)

Example

Consider the query $\mathsf{q}(x) \ \leftarrow \ \mathsf{playsIn}(x,y), \mathsf{Movie}(y)$

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

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

$$\mathsf{q}(x) \ \leftarrow \ \mathsf{playsIn}(x,y), \mathsf{playsIn}(z_1,y).$$

Query rewriting (cont'd)

Example

Consider the query $\mathsf{q}(x) \leftarrow \mathsf{playsIn}(x, y), \mathsf{Movie}(y)$

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

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

$$\mathsf{q}(x) \leftarrow \mathsf{playsIn}(x, y), \mathsf{playsIn}(z_1, y).$$

Example

Consider now the query $\mathsf{q}(x) \leftarrow \mathsf{playsIn}(x, y)$

and the inclusion assertion $\mathsf{MovieActor} \sqsubseteq \exists\mathsf{playsIn}$
as a logic programming rule: $\mathsf{playsIn}(z, f(z)) \leftarrow \mathsf{MovieActor}(z)$.

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

$$\mathsf{q}(x) \leftarrow \mathsf{MovieActor}(x).$$

Query rewriting – Constants

Example

Conversely, for the query $\mathsf{q}(x) \leftarrow \mathsf{playsIn}(x, \mathtt{matrix})$

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

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

Query rewriting – Constants

Example

Conversely, for the query $\mathsf{q}(x) \leftarrow \mathsf{playsIn}(x, \mathtt{matrix})$

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

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

We say that the inclusion does **not** apply to the atom $\mathsf{playsIn}(x, \mathtt{matrix})$.

unibz

Query rewriting – Constants

Example

Conversely, for the query $\mathsf{q}(x) \leftarrow \mathsf{playsIn}(x, \mathtt{matrix})$

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

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

We say that the inclusion does **not** apply to the atom $\mathsf{playsIn}(x, \mathtt{matrix})$.

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:

$$\mathsf{q}(x, y) \leftarrow \mathsf{playsIn}(x, y).$$

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 $\quad \mathsf{q}(x) \leftarrow \mathsf{playsIn}(x, y), \mathsf{Movie}(y)$

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

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

Query rewriting – Reduce step

Example

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

and the inclusion assertion $\mathsf{MovieActor} \sqsubseteq \exists\mathsf{playsIn}$
as a logic rule: $\mathsf{playsIn}(z, f(z)) \leftarrow \mathsf{MovieActor}(z)$.

This inclusion assertion does not apply to $\mathsf{playsIn}(x, y)$ or $\mathsf{playsIn}(z, y)$, 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 $\mathsf{q}(x) \leftarrow \mathsf{playsIn}(x, y), \mathsf{playsIn}(z, y)$

and the inclusion assertion $\mathsf{MovieActor} \sqsubseteq \exists\mathsf{playsIn}$
as a logic rule: $\mathsf{playsIn}(z, f(z)) \leftarrow \mathsf{MovieActor}(z)$.

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

$$\mathsf{q}(x) \leftarrow \mathsf{playsIn}(x, y).$$

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

$$\mathsf{q}(x) \leftarrow \mathsf{MovieActor}(x).$$

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:

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

  ('$\_$' 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.

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 **[CDLLR07]**.

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:

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

  ('$\_$' 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 [**CDLLR07**].

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:

$$\begin{aligned}\text{MovieActor} &\sqsubseteq \text{Actor}\\ \text{Actor} &\sqsubseteq \exists\text{playsIn}\\ \exists\text{playsIn}^- &\sqsubseteq \text{Movie}\end{aligned}$$

Corresponding rules:

$$\begin{aligned}\text{Actor}(x) &\leftarrow \text{MovieActor}(x)\\ \exists y(\text{playsIn}(x,y)) &\leftarrow \text{Actor}(x)\\ \text{Movie}(x) &\leftarrow \text{playsIn}(y,x)\end{aligned}$$

Query: $\text{q}(x) \leftarrow \text{playsIn}(x,y), \text{Movie}(y)$

unibz

Query answering in *DL-Lite* – Example

TBox:

$$\begin{aligned} \mathsf{MovieActor} &\sqsubseteq \mathsf{Actor} \\ \mathsf{Actor} &\sqsubseteq \exists \mathsf{playsIn} \\ \exists \mathsf{playsIn}^- &\sqsubseteq \mathsf{Movie} \end{aligned}$$

Corresponding rules:

$$\begin{aligned} \mathsf{Actor}(x) &\leftarrow \mathsf{MovieActor}(x) \\ \exists y(\mathsf{playsIn}(x, y)) &\leftarrow \mathsf{Actor}(x) \\ \mathsf{Movie}(x) &\leftarrow \mathsf{playsIn}(y, x) \end{aligned}$$

Query: $\mathsf{q}(x) \leftarrow \mathsf{playsIn}(x, y), \mathsf{Movie}(y)$

Perfect rewriting: $\mathsf{q}(x) \leftarrow \mathsf{playsIn}(x, y), \mathsf{Movie}(y)$

unibz

Query answering in *DL-Lite* – Example

TBox:

$$\begin{aligned} \mathsf{MovieActor} &\sqsubseteq \mathsf{Actor} \\ \mathsf{Actor} &\sqsubseteq \exists\mathsf{playsIn} \\ \exists\mathsf{playsIn}^- &\sqsubseteq \mathsf{Movie} \end{aligned}$$

Corresponding rules:

$$\begin{aligned} \mathsf{Actor}(x) &\leftarrow \mathsf{MovieActor}(x) \\ \exists y(\mathsf{playsIn}(x,y)) &\leftarrow \mathsf{Actor}(x) \\ \mathsf{Movie}(x) &\leftarrow \mathsf{playsIn}(y,x) \end{aligned}$$

Query: $\mathsf{q}(x) \leftarrow \mathsf{playsIn}(x,y), \mathsf{Movie}(y)$

Perfect rewriting:

$$\begin{aligned} \mathsf{q}(x) &\leftarrow \mathsf{playsIn}(x,y), \mathsf{Movie}(y) \\ \mathsf{q}(x) &\leftarrow \mathsf{playsIn}(x,y), \mathsf{playsIn}(\_,y) \end{aligned}$$

unibz

Query answering in *DL-Lite* – Example

TBox:

$$\begin{aligned} \text{MovieActor} &\sqsubseteq \text{Actor} \\ \text{Actor} &\sqsubseteq \exists\text{playsIn} \\ \exists\text{playsIn}^- &\sqsubseteq \text{Movie} \end{aligned}$$

Corresponding rules:

$$\begin{aligned} \text{Actor}(x) &\leftarrow \text{MovieActor}(x) \\ \exists y(\text{playsIn}(x, y)) &\leftarrow \text{Actor}(x) \\ \text{Movie}(x) &\leftarrow \text{playsIn}(y, x) \end{aligned}$$

Query: $\text{q}(x) \leftarrow \text{playsIn}(x, y), \text{Movie}(y)$

Perfect rewriting:

$$\begin{aligned} \text{q}(x) &\leftarrow \text{playsIn}(x, y), \text{Movie}(y) \\ \text{q}(x) &\leftarrow \text{playsIn}(x, y), \text{playsIn}(\_, y) \\ \text{q}(x) &\leftarrow \text{playsIn}(x, \_) \end{aligned}$$

Query answering in *DL-Lite* – Example

TBox:

$$\mathsf{MovieActor} \sqsubseteq \mathsf{Actor}$$
$$\mathsf{Actor} \sqsubseteq \exists \mathsf{playsIn}$$
$$\exists \mathsf{playsIn}^- \sqsubseteq \mathsf{Movie}$$

Corresponding rules:

$$\mathsf{Actor}(x) \leftarrow \mathsf{MovieActor}(x)$$
$$\exists y(\mathsf{playsIn}(x,y)) \leftarrow \mathsf{Actor}(x)$$
$$\mathsf{Movie}(x) \leftarrow \mathsf{playsIn}(y,x)$$

Query: $\mathsf{q}(x) \leftarrow \mathsf{playsIn}(x,y), \mathsf{Movie}(y)$

Perfect rewriting:

$$\mathsf{q}(x) \leftarrow \mathsf{playsIn}(x,y), \mathsf{Movie}(y)$$
$$\mathsf{q}(x) \leftarrow \mathsf{playsIn}(x,y), \mathsf{playsIn}(\_,y)$$
$$\mathsf{q}(x) \leftarrow \mathsf{playsIn}(x,\_)$$
$$\mathsf{q}(x) \leftarrow \mathsf{Actor}(x)$$

unibz

Query answering in *DL-Lite* – Example

TBox:

$$\begin{aligned} \text{MovieActor} &\sqsubseteq \text{Actor} \\ \text{Actor} &\sqsubseteq \exists\text{playsIn} \\ \exists\text{playsIn}^- &\sqsubseteq \text{Movie} \end{aligned}$$

Corresponding rules:

$$\begin{aligned} \text{Actor}(x) &\leftarrow \text{MovieActor}(x) \\ \exists y(\text{playsIn}(x,y)) &\leftarrow \text{Actor}(x) \\ \text{Movie}(x) &\leftarrow \text{playsIn}(y,x) \end{aligned}$$

Query: $\text{q}(x) \leftarrow \text{playsIn}(x,y), \text{Movie}(y)$

Perfect rewriting:

$$\begin{aligned} \text{q}(x) &\leftarrow \text{playsIn}(x,y), \text{Movie}(y) \\ \text{q}(x) &\leftarrow \text{playsIn}(x,y), \text{playsIn}(\_,y) \\ \text{q}(x) &\leftarrow \text{playsIn}(x,\_) \\ \text{q}(x) &\leftarrow \text{Actor}(x) \\ \text{q}(x) &\leftarrow \text{MovieActor}(x) \end{aligned}$$

unibz

Query answering in *DL-Lite* – Example

TBox:

$$\begin{aligned}\text{MovieActor} &\sqsubseteq \text{Actor}\\ \text{Actor} &\sqsubseteq \exists\text{playsIn}\\ \exists\text{playsIn}^- &\sqsubseteq \text{Movie}\end{aligned}$$

Corresponding rules:

$$\begin{aligned}\text{Actor}(x) &\leftarrow \text{MovieActor}(x)\\ \exists y(\text{playsIn}(x,y)) &\leftarrow \text{Actor}(x)\\ \text{Movie}(x) &\leftarrow \text{playsIn}(y,x)\end{aligned}$$

Query: $\text{q}(x) \leftarrow \text{playsIn}(x,y), \text{Movie}(y)$

Perfect rewriting:

$$\begin{aligned}\text{q}(x) &\leftarrow \text{playsIn}(x,y), \text{Movie}(y)\\ \text{q}(x) &\leftarrow \text{playsIn}(x,y), \text{playsIn}(\_,y)\\ \text{q}(x) &\leftarrow \text{playsIn}(x,\_)\\ \text{q}(x) &\leftarrow \text{Actor}(x)\\ \text{q}(x) &\leftarrow \text{MovieActor}(x)\end{aligned}$$

ABox: $\text{playsIn}(\texttt{keanu}, \texttt{matrix})$ $\quad$ $\text{MovieActor}(\texttt{keanu})$
$\text{playsIn}(\texttt{sigourney}, \texttt{alien})$ $\quad$ $\text{MovieActor}(\texttt{nicole})$

Evaluating the perfect rewriting over the ABox (seen as a DB) produces as answer $\{\texttt{keanu}, \texttt{sigourney}, \texttt{nicole}\}$.

unibz

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

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

ABox: $\mathsf{Person}(\texttt{john})$

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

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

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

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

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

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

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

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

ABox: $\text{Person}(\texttt{john})$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

unibz

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

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

ABox: $\text{Person}(\texttt{john})$

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

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

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

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

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

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

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

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

$\Downarrow$

...

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

unibz

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

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

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

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

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

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

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

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

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

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

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

$\Downarrow$

...

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

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

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

Complexity of query answering in *DL-Lite*

Query answering for CQs and UCQs 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 CQs and UCQs 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 constructor we lose these nice computational properties.

unibz

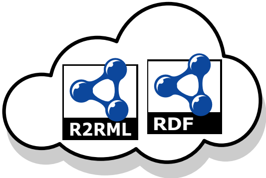
Outline

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

Querying the OBDA system

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

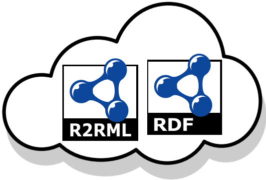
- *DL-Lite*$_\mathcal{R}$ TBox $\mathcal{T}$
- RDF graph $\mathcal{G}$ obtained from the mapping $\mathcal{M}$ and the data sources $\mathcal{D}$
- $\mathcal{G}$ can be viewed as the ABox



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

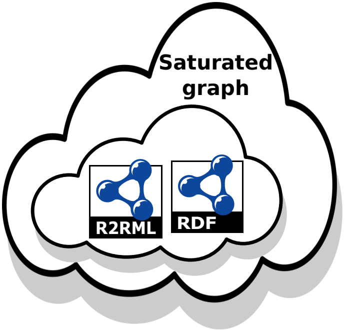


Query answering

- SPARQL query $\boldsymbol{q}$ 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 $\boldsymbol{q}$ over $\mathcal{K}$

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

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

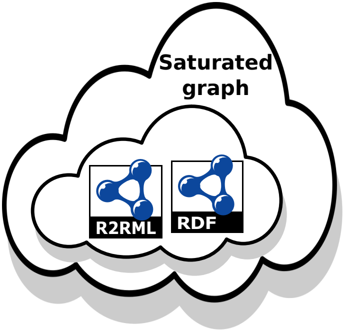
Querying the OBDA system

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



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

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

unibz

How to handle the RDF graph $\mathcal{G}_{\mathsf{sat}}$ 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 $\mathcal{M}_{\mathsf{sat}}$
- Typical profile: OWL 2 QL

unibz

H-complete ABox

[RoKZ13; rweb-KontchakovZ14]

ABox saturation

- H-complete ABox: contains all the inferable ABox assertions

H-complete ABox

[RoKZ13; rweb-KontchakovZ14]

ABox saturation

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

$$\mathcal{K} \models \alpha \quad \text{iff} \quad \alpha \in \mathcal{K}'$$

unibz

H-complete ABox

[RoKZ13; rweb-KontchakovZ14]

ABox saturation

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

$$\mathcal{K} \models \alpha \quad \text{iff} \quad \alpha \in \mathcal{K}'$$

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

- Composition of the mapping $\mathcal{M}$ and the *DL-Lite*$_{\mathcal{R}}$ TBox $\mathcal{T}$
- $\mathcal{M}_{\text{sat}} + \mathcal{D} \rightarrow \mathcal{G}_{\text{sat}}$ (H-complete ABox)

unibz

H-complete ABox
[RoKZ13; rweb-KontchakovZ14]

ABox saturation

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

$$\mathcal{K} \models \alpha \quad \text{iff} \quad \alpha \in \mathcal{K}'$$

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

- Composition of the mapping $\mathcal{M}$ and the *DL-Lite*$_{\mathcal{R}}$ TBox $\mathcal{T}$
- $\mathcal{M}_{\text{sat}} + \mathcal{D} \to \mathcal{G}_{\text{sat}}$ (H-complete ABox)
- Independent of the SPARQL query $\boldsymbol{q}$ (can be pre-computed)
- Can be optimized (query containment)

TBox, user-defined mapping assertions and foreign key

Student $\sqcup$ PostDoc $\sqcup$ AssociateProfessor $\sqcup$ $\exists$teaches $\sqsubseteq$ Person

$$\mathsf{Student}(\mathrm{URI}_1(p)) \leftarrow \texttt{uni1-student}(p, f, l) \tag{1}$$

$$\mathsf{PostDoc}(\mathrm{URI}_2(a)) \leftarrow \texttt{uni1-academic}(a, f, l, s), s = 9 \tag{2}$$

$$\mathsf{AssociateProfessor}(\mathrm{URI}_2(a)) \leftarrow \texttt{uni1-academic}(a, f, l, s), s = 2 \tag{3}$$

$$\mathsf{FacultyMember}(\mathrm{URI}_2(a)) \leftarrow \texttt{uni1-academic}(a, f, l, s) \tag{4}$$

$$\mathsf{teaches}(\mathrm{URI}_2(a), \mathrm{URI}_3(c)) \leftarrow \texttt{uni1-teaching}(c, a) \tag{5}$$

FK: $\exists y_1.\texttt{uni1-teaching}(y_1, x) \rightarrow \exists y_2 y_3 y_4.\texttt{uni1-academic}(x, y_2, y_3, y_4)$

TBox, user-defined mapping assertions and foreign key

$\mathsf{Student} \sqcup \mathsf{PostDoc} \sqcup \mathsf{AssociateProfessor} \sqcup \exists \mathsf{teaches} \sqsubseteq \mathsf{Person}$

$$\mathsf{Student}(\mathrm{URI}_1(p)) \leftarrow \texttt{uni1-student}(p, f, l) \tag{1}$$

$$\mathsf{PostDoc}(\mathrm{URI}_2(a)) \leftarrow \texttt{uni1-academic}(a, f, l, s), s = 9 \tag{2}$$

$$\mathsf{AssociateProfessor}(\mathrm{URI}_2(a)) \leftarrow \texttt{uni1-academic}(a, f, l, s), s = 2 \tag{3}$$

$$\mathsf{FacultyMember}(\mathrm{URI}_2(a)) \leftarrow \texttt{uni1-academic}(a, f, l, s) \tag{4}$$

$$\mathsf{teaches}(\mathrm{URI}_2(a), \mathrm{URI}_3(c)) \leftarrow \texttt{uni1-teaching}(c, a) \tag{5}$$

FK: $\exists y_1.\texttt{uni1-teaching}(y_1, x) \rightarrow \exists y_2 y_3 y_4.\texttt{uni1-academic}(x, y_2, y_3, y_4)$

Non-optimized saturated mapping assertions for Person

$$\mathsf{Person}(\mathrm{URI}_1(p)) \leftarrow \texttt{uni1-student}(p, f, l) \tag{6}$$

$$\mathsf{Person}(\mathrm{URI}_2(a)) \leftarrow \texttt{uni1-academic}(a, f, l, s), s = 9 \tag{7}$$

$$\mathsf{Person}(\mathrm{URI}_2(a)) \leftarrow \texttt{uni1-academic}(a, f, l, s), s = 2 \tag{8}$$

$$\mathsf{Person}(\mathrm{URI}_2(a)) \leftarrow \texttt{uni1-academic}(a, f, l, s) \tag{9}$$

$$\mathsf{Person}(\mathrm{URI}_2(a)) \leftarrow \texttt{uni1-teaching}(c, a) \tag{10}$$

TBox, user-defined mapping assertions and foreign key

$\mathsf{Student} \sqcup \mathsf{PostDoc} \sqcup \mathsf{AssociateProfessor} \sqcup \exists \mathsf{teaches} \sqsubseteq \mathsf{Person}$

$$\begin{aligned}
\mathsf{Student}(\mathrm{URI}_1(p)) &\leftarrow \texttt{uni1-student}(p, f, l) && (1)\\
\mathsf{PostDoc}(\mathrm{URI}_2(a)) &\leftarrow \texttt{uni1-academic}(a, f, l, s), s = 9 && (2)\\
\mathsf{AssociateProfessor}(\mathrm{URI}_2(a)) &\leftarrow \texttt{uni1-academic}(a, f, l, s), s = 2 && (3)\\
\mathsf{FacultyMember}(\mathrm{URI}_2(a)) &\leftarrow \texttt{uni1-academic}(a, f, l, s) && (4)\\
\mathsf{teaches}(\mathrm{URI}_2(a), \mathrm{URI}_3(c)) &\leftarrow \texttt{uni1-teaching}(c, a) && (5)
\end{aligned}$$

FK: $\exists y_1.\texttt{uni1-teaching}(y_1, x) \rightarrow \exists y_2 y_3 y_4.\texttt{uni1-academic}(x, y_2, y_3, y_4)$

Non-optimized saturated mapping assertions for Person

$$\begin{aligned}
\mathsf{Person}(\mathrm{URI}_1(p)) &\leftarrow \texttt{uni1-student}(p, f, l) && (6)\\
\mathsf{Person}(\mathrm{URI}_2(a)) &\leftarrow \texttt{uni1-academic}(a, f, l, s), s = 9 && (7)\\
\mathsf{Person}(\mathrm{URI}_2(a)) &\leftarrow \texttt{uni1-academic}(a, f, l, s), s = 2 && (8)\\
\mathsf{Person}(\mathrm{URI}_2(a)) &\leftarrow \texttt{uni1-academic}(a, f, l, s) && (9)\\
\mathsf{Person}(\mathrm{URI}_2(a)) &\leftarrow \texttt{uni1-teaching}(c, a) && (10)
\end{aligned}$$

Mapping assertions for Person after optimization (query containment)

$$\begin{aligned}
\mathsf{Person}(\mathrm{URI}_1(p)) &\leftarrow \texttt{uni1-student}(p, f, l) && (11)\\
\mathsf{Person}(\mathrm{URI}_2(p)) &\leftarrow \texttt{uni1-academic}(p, f, l, s) && (12)
\end{aligned}$$

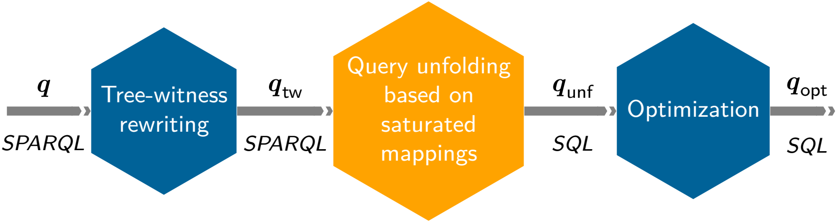
Outline

1. Query rewriting wrt an OWL 2 QL ontology
2. Saturation and optimization of the mapping
3. Query reformulation
   - Tree-witness rewriting
   - SQL query optimization

unibz

Query reformulation

Implemented by Ontop



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

Example: Existential reasoning (I)

Example

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

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

and john is a graduate student:

$$\texttt{GraduateStudent(john)}$$

Example: Existential reasoning (I)

Example

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

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

and john is a graduate student:

$$\mathtt{GraduateStudent(john)}$$

What is the answer to the following query?

```
SELECT ?x WHERE { ?x isSupervisedBy [ a Professor ] .}
```

Example: Existential reasoning (I)

Example

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

$$\texttt{GraduateStudent} \sqsubseteq \exists\, \texttt{isSupervisedBy.Professor}$$

and john is a graduate student:

$$\texttt{GraduateStudent(john)}$$

What is the answer to the following query?

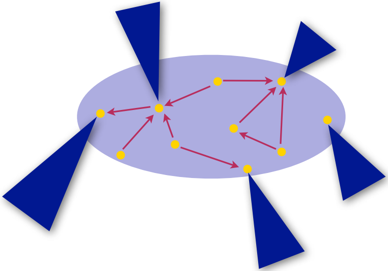
```
SELECT ?x WHERE { ?x isSupervisedBy [ a Professor ] .}
```

Yes. Even though we don't know who is john's supervisor.

unibz

Existential reasoning and tree-witness rewriting

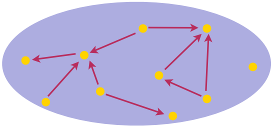
Fact: every consistent DL-Lite KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ has a **canonical model** $\mathcal{I}_{\mathcal{K}}$, which **gives the right answers to all CQs**, i.e., $\mathsf{cert}(q, \mathcal{K}) = \mathsf{ans}(q, \mathcal{I}_{\mathcal{K}})$



Existential reasoning and tree-witness rewriting

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

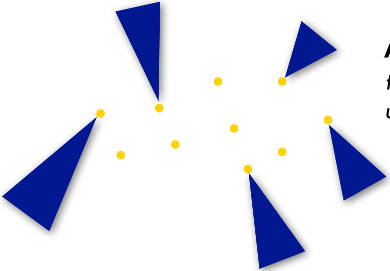
Core
individuals from $\mathcal{A}$



- The core part can be handled by the saturated mapping

Existential reasoning and tree-witness rewriting

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



Anonymous part

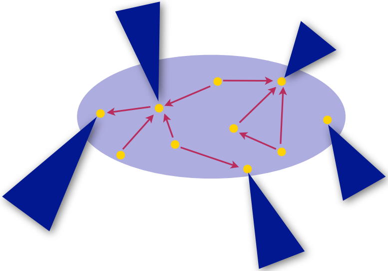
trees rooted at individuals, using unnamed objects

- The core part can be handled by the saturated mapping
- The anonymous part can be handled by Tree-witness rewriting

unibz

Existential reasoning and tree-witness rewriting

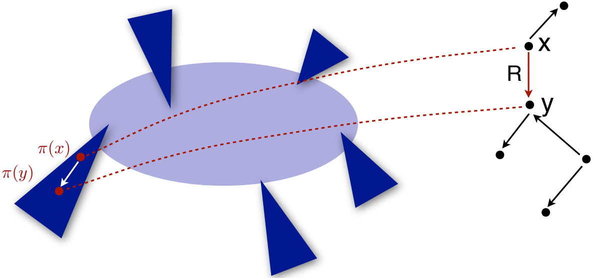
Fact: every consistent DL-Lite KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ has a **canonical model** $\mathcal{I}_{\mathcal{K}}$, which **gives the right answers to all CQs**, i.e., $\text{cert}(q, \mathcal{K}) = \text{ans}(q, \mathcal{I}_{\mathcal{K}})$



- The core part can be handled by the saturated mapping
- The anonymous part can be handled by Tree-witness rewriting

Existential reasoning and tree-witness rewriting

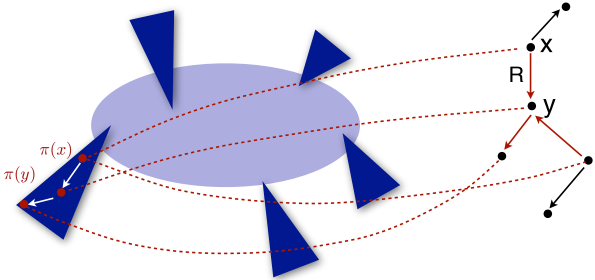
Fact: every consistent DL-Lite KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ has a **canonical model** $\mathcal{I}_{\mathcal{K}}$, which **gives the right answers to all CQs**, i.e., $\mathsf{cert}(q, \mathcal{K}) = \mathsf{ans}(q, \mathcal{I}_{\mathcal{K}})$



- The core part can be handled by the saturated mapping
- The anonymous part can be handled by Tree-witness rewriting

Existential reasoning and tree-witness rewriting

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



- The core part can be handled by the saturated mapping
- The anonymous part can be handled by Tree-witness rewriting

Example: Existential reasoning (II)

Using the tree witness rewriting algorithm, the query

```
SELECT ?x WHERE { ?x :isSupervisedBy [ a :Professor ] .}
```

Example: Existential reasoning (II)

Using the tree witness rewriting algorithm, the query

```
SELECT ?x WHERE { ?x :isSupervisedBy [ a :Professor ] .}
```

is rewritten to

```
SELECT ?x WHERE { ?x :isSupervisedBy [ a :Professor ] .}
UNION
SELECT ?x WHERE { ?x :GraduateStudent .}
```

unibz

Example: Existential reasoning (II)

Using the tree witness rewriting algorithm, the query

```
SELECT ?x WHERE { ?x :isSupervisedBy [ a :Professor ] .}
```

is rewritten to

```
SELECT ?x WHERE { ?x :isSupervisedBy [ a :Professor ] .}
UNION
SELECT ?x WHERE { ?x :GraduateStudent .}
```

Therefore, john is computed as an answer.

Example: Existential reasoning (II)

Using the tree witness rewriting algorithm, the query

```
SELECT ?x WHERE { ?x :isSupervisedBy [ a :Professor ] .}
```

is rewritten to

```
SELECT ?x WHERE { ?x :isSupervisedBy [ a :Professor ] .}
UNION
SELECT ?x WHERE { ?x :GraduateStudent .}
```

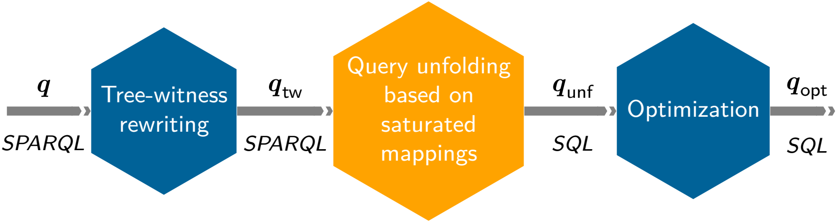
Therefore, john is computed as an answer.

Tree-witness rewriting option in *Ontop*

Note that if users want to enable existential reasoning, the option of tree-witness rewriting algorithm needs to be switched explicitly.

Query reformulation

Implemented by Ontop



Step	Input	Output
1. Tree-witness rewriting	q (SPARQL) and $\mathcal{T}$	q_{tw} (SPARQL)
2. Query unfolding	q_{tw} and $\mathcal{M}_{\text{sat}}$	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

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 mappings

$$\mathsf{firstName}(\mathrm{URI}_1(p), f) \leftarrow \texttt{uni1-student}(p, f, l)$$
$$\mathsf{firstName}(\mathrm{URI}_2(a), f) \leftarrow \texttt{uni1-academic}(a, f, l, s)$$
$$\mathsf{lastName}(\mathrm{URI}_1(p), l) \leftarrow \texttt{uni1-student}(p, f, l)$$
$$\mathsf{lastName}(\mathrm{URI}_2(a), l) \leftarrow \texttt{uni1-academic}(a, f, l, s)$$
$$\mathsf{Teacher}(\mathrm{URI}_2(a)) \leftarrow \texttt{uni1-academic}(a, f, l, s), s \in [1, 8]$$
$$\mathsf{Teacher}(\mathrm{URI}_2(a)) \leftarrow \texttt{uni1-teaching}(c, a)$$

Query

$$\boldsymbol{q}(x, y, z) \leftarrow \mathsf{Teacher}(x), \mathsf{firstName}(x, y), \mathsf{lastName}(x, z)$$
$$\boldsymbol{q}_{\mathsf{tw}} = \boldsymbol{q}$$

Reformulation example - 1. Unfolding

Saturated mappings

$$\text{firstName}(\mathrm{URI}_1(p), f) \leftarrow \texttt{uni1-student}(p, f, l)$$
$$\text{firstName}(\mathrm{URI}_2(a), f) \leftarrow \texttt{uni1-academic}(a, f, l, s)$$
$$\text{lastName}(\mathrm{URI}_1(p), l) \leftarrow \texttt{uni1-student}(p, f, l)$$
$$\text{lastName}(\mathrm{URI}_2(a), l) \leftarrow \texttt{uni1-academic}(a, f, l, s)$$
$$\text{Teacher}(\mathrm{URI}_2(a)) \leftarrow \texttt{uni1-academic}(a, f, l, s), s \in [1, 8]$$
$$\text{Teacher}(\mathrm{URI}_2(a)) \leftarrow \texttt{uni1-teaching}(c, a)$$

Query

$$\boldsymbol{q}(x, y, z) \leftarrow \text{Teacher}(x), \text{firstName}(x, y), \text{lastName}(x, z)$$
$$\boldsymbol{q}_{\text{tw}} = \boldsymbol{q}$$

Query unfolding

$$\boldsymbol{q}_{\text{unf}}(x, y, z) \leftarrow \boldsymbol{q}_{\text{unf1}}(x), \boldsymbol{q}_{\text{unf2}}(x, y), \boldsymbol{q}_{\text{unf3}}(x, z)$$
$$\boldsymbol{q}_{\text{unf1}}(\mathrm{URI}_2(a)) \leftarrow \texttt{uni1-academic}(a, f, l, s), s \in [1, 8]$$
$$\boldsymbol{q}_{\text{unf1}}(\mathrm{URI}_2(a)) \leftarrow \texttt{uni1-teaching}(c, a)$$
$$\boldsymbol{q}_{\text{unf2}}(\mathrm{URI}_1(p), f) \leftarrow \texttt{uni1-student}(p, f, l)$$
$$\boldsymbol{q}_{\text{unf2}}(\mathrm{URI}_2(a), f) \leftarrow \texttt{uni1-academic}(a, f, l, s)$$
$$\boldsymbol{q}_{\text{unf3}}(\mathrm{URI}_1(p), l) \leftarrow \texttt{uni1-student}(p, f, l)$$
$$\boldsymbol{q}_{\text{unf3}}(\mathrm{URI}_2(a), l) \leftarrow \texttt{uni1-academic}(a, f, l, s)$$

Reformulation example - 2. Structural optimization

Query unfolding

$$
\begin{aligned}
\boldsymbol{q}_{\mathsf{unf}}(x,y,z) &\leftarrow \boldsymbol{q}_{\mathsf{unf1}}(x), \boldsymbol{q}_{\mathsf{unf2}}(x,y), \boldsymbol{q}_{\mathsf{unf3}}(x,z)\\
\boldsymbol{q}_{\mathsf{unf1}}(\mathrm{URI}_2(a)) &\leftarrow \texttt{uni1-academic}(a,f,l,s), s \in [1,8]\\
\boldsymbol{q}_{\mathsf{unf1}}(\mathrm{URI}_2(a)) &\leftarrow \texttt{uni1-teaching}(c,a)\\
\boldsymbol{q}_{\mathsf{unf2}}(\mathrm{URI}_1(p),f) &\leftarrow \texttt{uni1-student}(p,f,l)\\
\boldsymbol{q}_{\mathsf{unf2}}(\mathrm{URI}_2(a),f) &\leftarrow \texttt{uni1-academic}(a,f,l,s)\\
\boldsymbol{q}_{\mathsf{unf3}}(\mathrm{URI}_1(p),l) &\leftarrow \texttt{uni1-student}(p,f,l)\\
\boldsymbol{q}_{\mathsf{unf3}}(\mathrm{URI}_2(a),l) &\leftarrow \texttt{uni1-academic}(a,f,l,s)
\end{aligned}
$$

Reformulation example - 2. Structural optimization

Query unfolding

$$
\begin{aligned}
\boldsymbol{q}_{\mathsf{unf}}(x, y, z) &\leftarrow \boldsymbol{q}_{\mathsf{unf1}}(x), \boldsymbol{q}_{\mathsf{unf2}}(x, y), \boldsymbol{q}_{\mathsf{unf3}}(x, z) \\
\boldsymbol{q}_{\mathsf{unf1}}(\mathrm{URI}_2(a)) &\leftarrow \texttt{uni1-academic}(a, f, l, s), s \in [1, 8] \\
\boldsymbol{q}_{\mathsf{unf1}}(\mathrm{URI}_2(a)) &\leftarrow \texttt{uni1-teaching}(c, a) \\
\boldsymbol{q}_{\mathsf{unf2}}(\mathrm{URI}_1(p), f) &\leftarrow \texttt{uni1-student}(p, f, l) \\
\boldsymbol{q}_{\mathsf{unf2}}(\mathrm{URI}_2(a), f) &\leftarrow \texttt{uni1-academic}(a, f, l, s) \\
\boldsymbol{q}_{\mathsf{unf3}}(\mathrm{URI}_1(p), l) &\leftarrow \texttt{uni1-student}(p, f, l) \\
\boldsymbol{q}_{\mathsf{unf3}}(\mathrm{URI}_2(a), l) &\leftarrow \texttt{uni1-academic}(a, f, l, s)
\end{aligned}
$$

Normalization: explicit joining conditions

$$
\begin{aligned}
\boldsymbol{q}_{\mathsf{exp}}(x, y, z) &\leftarrow \boldsymbol{q}_{\mathsf{unf1}}(x), \boldsymbol{q}_{\mathsf{unf2}}(x_1, y), \boldsymbol{q}_{\mathsf{unf3}}(x_2, z), x = x_1, x = x_2 \\
\boldsymbol{q}_{\mathsf{unf1}}(\mathrm{URI}_2(a)) &\leftarrow \texttt{uni1-academic}(a, f, l, s), s \in [1, 8] \\
\boldsymbol{q}_{\mathsf{unf1}}(\mathrm{URI}_2(a)) &\leftarrow \texttt{uni1-teaching}(c, a) \\
\boldsymbol{q}_{\mathsf{unf2}}(\mathrm{URI}_1(p), f) &\leftarrow \texttt{uni1-student}(p, f, l) \\
\boldsymbol{q}_{\mathsf{unf2}}(\mathrm{URI}_2(a), f) &\leftarrow \texttt{uni1-academic}(a, f, l, s) \\
\boldsymbol{q}_{\mathsf{unf3}}(\mathrm{URI}_1(p), l) &\leftarrow \texttt{uni1-student}(p, f, l) \\
\boldsymbol{q}_{\mathsf{unf3}}(\mathrm{URI}_2(a), l) &\leftarrow \texttt{uni1-academic}(a, f, l, s)
\end{aligned}
$$

Reformulation example - 2. Structural optimization

Normalization: explicit joining conditions

$$\begin{aligned}
\boldsymbol{q}_{\mathsf{exp}}(x,y,z) &\leftarrow \boldsymbol{q}_{\mathsf{unf1}}(x), \boldsymbol{q}_{\mathsf{unf2}}(x_1,y),\\
&\quad \boldsymbol{q}_{\mathsf{unf3}}(x_2,z),\\
&\quad x = x_1, x = x_2\\
\boldsymbol{q}_{\mathsf{unf1}}(\mathrm{URI}_2(a)) &\leftarrow \texttt{uni1-academic}(a,f,l,s),\\
&\quad s \in [1,8]\\
\boldsymbol{q}_{\mathsf{unf1}}(\mathrm{URI}_2(a)) &\leftarrow \texttt{uni1-teaching}(c,a)\\
\boldsymbol{q}_{\mathsf{unf2}}(\mathrm{URI}_1(p),f) &\leftarrow \texttt{uni1-student}(p,f,l)\\
\boldsymbol{q}_{\mathsf{unf2}}(\mathrm{URI}_2(a),f) &\leftarrow \texttt{uni1-academic}(a,f,l,s)\\
\boldsymbol{q}_{\mathsf{unf3}}(\mathrm{URI}_1(p),l) &\leftarrow \texttt{uni1-student}(p,f,l)\\
\boldsymbol{q}_{\mathsf{unf3}}(\mathrm{URI}_2(a),l) &\leftarrow \texttt{uni1-academic}(a,f,l,s)
\end{aligned}$$

Flattening (URI template lifting) - part 1/2

$$\begin{aligned}
\boldsymbol{q}_{\mathsf{lift}}(\mathrm{URI}_2(a),y,z) &\leftarrow \texttt{uni1-academic}(a,f_1,l_1,s_1),\\
&\quad \texttt{uni1-student}(p,f_2,l_2),\\
&\quad \texttt{uni1-student}(p_1,f_3,l_3),\\
&\quad \mathrm{URI}_2(a) = \mathrm{URI}_1(p),\\
&\quad \mathrm{URI}_2(a) = \mathrm{URI}_1(p_1),\\
&\quad s_1 \in [1,8]\\
\boldsymbol{q}_{\mathsf{lift}}(\mathrm{URI}_2(a),y,z) &\leftarrow \texttt{uni1-academic}(a,f_1,l_1,s_1),\\
&\quad \texttt{uni1-student}(p,f_2,l_2),\\
&\quad \texttt{uni1-academic}(a_2,f_3,z,s_3),\\
&\quad \mathrm{URI}_2(a) = \mathrm{URI}_1(p),\\
&\quad \mathrm{URI}_2(a) = \mathrm{URI}_2(a_2), \ldots
\end{aligned}$$

(One sub-query ignored)

$$\begin{aligned}
\boldsymbol{q}_{\mathsf{lift}}(\mathrm{URI}_2(a),y,z) &\leftarrow \texttt{uni1-academic}(a,f_1,l_1,s_1),\\
&\quad \texttt{uni1-academic}(a_1,y,l_2,s_2),\\
&\quad \texttt{uni1-academic}(a_2,f_3,z,s_3),\\
&\quad \mathrm{URI}_2(a) = \mathrm{URI}_2(a_1),\\
&\quad \mathrm{URI}_2(a) = \mathrm{URI}_2(a_2),\\
&\quad s_1 \in [1,8]
\end{aligned}$$

Reformulation example - 2. Structural optimization

Normalization: explicit joining conditions

$$
\begin{aligned}
\boldsymbol{q}_{\mathsf{exp}}(x, y, z) &\leftarrow \boldsymbol{q}_{\mathsf{unf1}}(x), \boldsymbol{q}_{\mathsf{unf2}}(x_1, y),\\
&\quad \boldsymbol{q}_{\mathsf{unf3}}(x_2, z),\\
&\quad x = x_1, x = x_2\\
\boldsymbol{q}_{\mathsf{unf1}}(\mathrm{URI}_2(a)) &\leftarrow \texttt{uni1-academic}(a, f, l, s),\\
&\quad s \in [1, 8]\\
\boldsymbol{q}_{\mathsf{unf1}}(\mathrm{URI}_2(a)) &\leftarrow \texttt{uni1-teaching}(c, a)\\
\boldsymbol{q}_{\mathsf{unf2}}(\mathrm{URI}_1(p), f) &\leftarrow \texttt{uni1-student}(p, f, l)\\
\boldsymbol{q}_{\mathsf{unf2}}(\mathrm{URI}_2(a), f) &\leftarrow \texttt{uni1-academic}(a, f, l, s)\\
\boldsymbol{q}_{\mathsf{unf3}}(\mathrm{URI}_1(p), l) &\leftarrow \texttt{uni1-student}(p, f, l)\\
\boldsymbol{q}_{\mathsf{unf3}}(\mathrm{URI}_2(a), l) &\leftarrow \texttt{uni1-academic}(a, f, l, s)
\end{aligned}
$$

Flattening (URI template lifting) - part 2/2

$$
\begin{aligned}
\boldsymbol{q}_{\mathsf{lift}}(\mathrm{URI}_2(a), y, z) &\leftarrow \texttt{uni1-teaching}(c, a),\\
&\quad \texttt{uni1-student}(p, f_2, l_2),\\
&\quad \texttt{uni1-student}(p_1, f_3, l_3),\\
&\quad \mathrm{URI}_2(a) = \mathrm{URI}_1(p),\\
&\quad \mathrm{URI}_2(a) = \mathrm{URI}_1(p_1)
\end{aligned}
$$

(One sub-query ignored)

$$
\begin{aligned}
\boldsymbol{q}_{\mathsf{lift}}(\mathrm{URI}_2(a), y, z) &\leftarrow \texttt{uni1-teaching}(c, a),\\
&\quad \texttt{uni1-academic}(a_1, y, l_2, s_2),\\
&\quad \texttt{uni1-student}(p, f_3, l_3),\\
&\quad \mathrm{URI}_2(a) = \mathrm{URI}_2(a_1),\\
&\quad \mathrm{URI}_2(a) = \mathrm{URI}_1(p)\\
\boldsymbol{q}_{\mathsf{lift}}(\mathrm{URI}_2(a), y, z) &\leftarrow \texttt{uni1-teaching}(c, a),\\
&\quad \texttt{uni1-academic}(a_1, y, l_2, s_2),\\
&\quad \texttt{uni1-academic}(a_2, f_3, z, s_3),\\
&\quad \mathrm{URI}_2(a) = \mathrm{URI}_2(a_1),\\
&\quad \mathrm{URI}_2(a) = \mathrm{URI}_2(a_2)
\end{aligned}
$$

Reformulation example - 2. Structural optimization

Simplification and implicit equality normalization

$$
\begin{aligned}
\boldsymbol{q}_{\mathsf{struct}}(\mathrm{URI}_2(a), y, z) \leftarrow & \texttt{uni1-academic}(a, f_1, l_1, s_1),\\
& \texttt{uni1-academic}(a, y, l_2, s_2),\\
& \texttt{uni1-academic}(a, f_3, z, s_3), s_1 \in [1, 8]\\
\boldsymbol{q}_{\mathsf{struct}}(\mathrm{URI}_2(a), y, z) \leftarrow & \texttt{uni1-teaching}(c, a),\\
& \texttt{uni1-academic}(a, y, l_2, s_2),\\
& \texttt{uni1-academic}(a, f_3, z, s_3)
\end{aligned}
$$

Reformulation example - 2. Structural optimization

Simplification and implicit equality normalization

$$
\begin{aligned}
\boldsymbol{q}_{\mathsf{struct}}(\mathrm{URI}_2(a), y, z) \leftarrow & \texttt{uni1-academic}(a, f_1, l_1, s_1), \\
& \texttt{uni1-academic}(a, y, l_2, s_2), \\
& \texttt{uni1-academic}(a, f_3, z, s_3), s_1 \in [1, 8] \\
\boldsymbol{q}_{\mathsf{struct}}(\mathrm{URI}_2(a), y, z) \leftarrow & \texttt{uni1-teaching}(c, a), \\
& \texttt{uni1-academic}(a, y, l_2, s_2), \\
& \texttt{uni1-academic}(a, f_3, z, s_3)
\end{aligned}
$$

Remarks on the flattening step:

- Possible exponential blowup!
- Usually avoided thanks to incompatible URI templates.

Reformulation example - 3. Semantic optimization

Simplification and implicit equality normalization

$$\boldsymbol{q}_{\mathsf{struct}}(\mathrm{URI}_2(a), y, z) \leftarrow \texttt{uni1-academic}(a, f_1, l_1, s_1), \texttt{uni1-academic}(a, y, l_2, s_2), \texttt{uni1-academic}(a, f_3, z, s_3), s_1 \in [1, 8]$$

$$\boldsymbol{q}_{\mathsf{struct}}(\mathrm{URI}_2(a), y, z) \leftarrow \texttt{uni1-teaching}(c, a), \texttt{uni1-academic}(a, y, l_2, s_2), \texttt{uni1-academic}(a, f_3, z, s_3)$$

Self-join elimination (semantic optimization)

PK: $\texttt{uni1-academic}(a, b, c, d) \land \texttt{uni1-academic}(a, b', c', d') \rightarrow (b = b') \land (c = c') \land (d = d')$

$$\boldsymbol{q}_{\mathsf{opt}}(\mathrm{URI}_2(a), y, z) \leftarrow \texttt{uni1-academic}(a, y, z, s_1), s_1 \in [1, 8]$$

$$\boldsymbol{q}_{\mathsf{opt}}(\mathrm{URI}_2(a), y, z) \leftarrow \texttt{uni1-teaching}(c, a), \texttt{uni1-academic}(a, y, z, s_2)$$

unibz

References I