Virtual Knowledge Graphs: Query Processing

Guohui Xiao

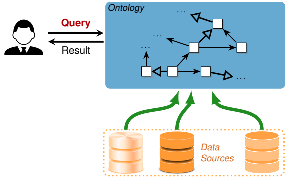
University of Bergen, Norway

January 25, 2024

Outline

Outline

VKGs: Formalization



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

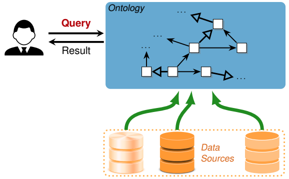
A **VKG specification** is a triple $\mathcal{P} = \langle \mathcal{O}, \mathcal{M}, \mathcal{S} \rangle$, where:

- $\mathcal{O}$ is an ontology (expressed in OWL 2 QL),
- $\mathcal{S}$ is a (possibly federated) relational database schema for the data sources, possibly with integrity constraints,
- $\mathcal{M}$ is a set of (R2RML) mapping assertions between $\mathcal{O}$ and $\mathcal{S}$.

A **VKG instance** is a pair $\langle \mathcal{P}, \mathcal{D} \rangle$, where

- $\mathcal{P} = \langle \mathcal{O}, \mathcal{M}, \mathcal{S} \rangle$ is a VKG specification, and
- $\mathcal{D}$ is a (possibly federated) relational database compliant with $\mathcal{S}$.

VKGs: Formalization



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

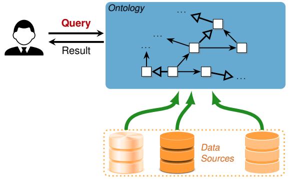
A **VKG specification** is a triple $\mathcal{P} = \langle \mathcal{O}, \mathcal{M}, \mathcal{S} \rangle$, where:

- $\mathcal{O}$ is an ontology (expressed in OWL 2 QL),
- $\mathcal{S}$ is a (possibly federated) relational database schema for the data sources, possibly with integrity constraints,
- $\mathcal{M}$ is a set of (R2RML) mapping assertions between $\mathcal{O}$ and $\mathcal{S}$.

A **VKG instance** is a pair $\langle \mathcal{P}, \mathcal{D} \rangle$, where

- $\mathcal{P} = \langle \mathcal{O}, \mathcal{M}, \mathcal{S} \rangle$ is a VKG specification, and
- $\mathcal{D}$ is a (possibly federated) relational database compliant with $\mathcal{S}$.

VKGs: Formalization



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

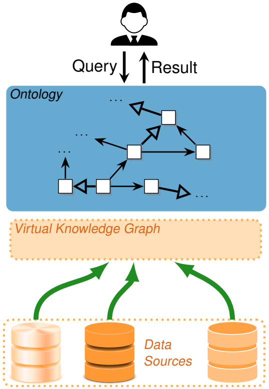
A **VKG specification** is a triple $\mathcal{P} = \langle \mathcal{O}, \mathcal{M}, \mathcal{S} \rangle$, where:

- $\mathcal{O}$ is an ontology (expressed in OWL 2 QL),
- $\mathcal{S}$ is a (possibly federated) relational database schema for the data sources, possibly with integrity constraints,
- $\mathcal{M}$ is a set of (R2RML) mapping assertions between $\mathcal{O}$ and $\mathcal{S}$.

A **VKG instance** is a pair $\langle \mathcal{P}, \mathcal{D} \rangle$, where

- $\mathcal{P} = \langle \mathcal{O}, \mathcal{M}, \mathcal{S} \rangle$ is a VKG specification, and
- $\mathcal{D}$ is a (possibly federated) relational database compliant with $\mathcal{S}$.

Semantics of VKGs



Remember:

- The mapping $\mathcal{M}$ generates from the data $\mathcal{D}$ in the sources a **virtual knowledge graph** $\mathcal{V} = \mathcal{M}(\mathcal{D})$.
- The set of constants that can appear in $\mathcal{V}$ consists of:
  - values obtained directly from the database, and
  - IRIs, which are constructed by applying the **iri** function to string constants and database values.

  We use $C_{\mathcal{V}}$, i.e., $C_{\mathcal{M}(\mathcal{D})}$, to denote such set of constants.

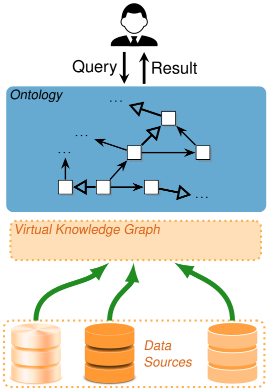
A first-order interpretation $\mathcal{I}$ of the ontology predicates and the constants in $C_{\mathcal{M}(\mathcal{D})}$ is a **model** of $\langle \mathcal{P}, \mathcal{D} \rangle$ if

- it satisfies all axioms in $\mathcal{O}$, and
- contains all facts in $\mathcal{M}(\mathcal{D})$, i.e., retrieved from $\mathcal{D}$ through $\mathcal{M}$.

Note:

- In general, $\langle \mathcal{P}, \mathcal{D} \rangle$ has infinitely many models, and some of these might be infinite.
- However, for query answering, we do not need to compute such models.

Semantics of VKGs



Remember:

- The mapping $\mathcal{M}$ generates from the data $\mathcal{D}$ in the sources a **virtual knowledge graph** $\mathcal{V} = \mathcal{M}(\mathcal{D})$.
- The set of constants that can appear in $\mathcal{V}$ consists of:
  - values obtained directly from the database, and
  - IRIs, which are constructed by applying the **iri** function to string constants and database values.

  We use $C_{\mathcal{V}}$, i.e., $C_{\mathcal{M}(\mathcal{D})}$, to denote such set of constants.

A first-order interpretation $\mathcal{I}$ of the ontology predicates and the constants in $C_{\mathcal{M}(\mathcal{D})}$ is a **model** of $\langle \mathcal{P}, \mathcal{D} \rangle$ if

- it satisfies all axioms in $\mathcal{O}$, and
- contains all facts in $\mathcal{M}(\mathcal{D})$, i.e., retrieved from $\mathcal{D}$ through $\mathcal{M}$.

Note:

- In general, $\langle \mathcal{P}, \mathcal{D} \rangle$ has infinitely many models, and some of these might be infinite.
- However, for query answering, we do not need to compute such models.

Query answering in VKGs – Certain answers

In VKGs, we want to answer queries formulated over the ontology, by using the data provided by the data sources through the mapping.

Consider our formalization of VKGs and a VKG instance $\mathcal{J}$.

Certain answers $\text{cert}(q, \mathcal{J})$ – Intuition

Given a VKG instance $\mathcal{J}$ and a query q over $\mathcal{J}$, the certain answers $\text{cert}(q, \mathcal{J})$ to q over $\mathcal{J}$ are those answers to q that hold in every model of $\mathcal{J}$.

Certain answers $\text{cert}(q, \mathcal{J})$ – Formal definition

Given a VKG instance $\mathcal{J} = \langle \mathcal{P}, \mathcal{D} \rangle$ and a query q over $\mathcal{J}$, a tuple $\vec{c}$ of constants in $C_{\mathcal{M}(\mathcal{D})}$ is a **certain answer** to q over $\mathcal{J}$, i.e., $\vec{c} \in \text{cert}(q, \mathcal{J})$, if for every model $\mathcal{I}$ of $\mathcal{J}$ we have that $\vec{c} \in q(\mathcal{I})$.

Note: Each certain answer $\vec{c}$ is a tuple of constants in $C_{\mathcal{M}(\mathcal{D})}$, but when we evaluate q over an interpretation $\mathcal{I}$, it returns tuples of elements of $\Delta^{\mathcal{I}}$. Therefore, we should actually require that $\vec{c}^{\mathcal{I}} \in q(\mathcal{I})$, and not that $\vec{c} \in q(\mathcal{I})$. However, due to the standard names assumption, we have that $\vec{c}^{\mathcal{I}} = \vec{c}$, so the two conditions are equivalent.

Query answering in VKGs – Certain answers

In VKGs, we want to answer queries formulated over the ontology, by using the data provided by the data sources through the mapping.

Consider our formalization of VKGs and a VKG instance $\mathcal{J}$.

Certain answers cert$(\boldsymbol{q}, \mathcal{J})$ – Intuition

Given a VKG instance $\mathcal{J}$ and a query $\boldsymbol{q}$ over $\mathcal{J}$, the certain answers cert$(\boldsymbol{q}, \mathcal{J})$ to $\boldsymbol{q}$ over $\mathcal{J}$ are those answers to $\boldsymbol{q}$ that hold in every model of $\mathcal{J}$.

Certain answers cert$(\boldsymbol{q}, \mathcal{J})$ – Formal definition

Given a VKG instance $\mathcal{J} = \langle \mathcal{P}, \mathcal{D} \rangle$ and a query $\boldsymbol{q}$ over $\mathcal{J}$, a tuple $\vec{c}$ of constants in $C_{\mathcal{M}(\mathcal{D})}$ is a **certain answer** to $\boldsymbol{q}$ over $\mathcal{J}$, i.e., $\vec{c} \in$ cert$(\boldsymbol{q}, \mathcal{J})$, if for every model $\mathcal{I}$ of $\mathcal{J}$ we have that $\vec{c} \in \boldsymbol{q}(\mathcal{I})$.

Note: Each certain answer $\vec{c}$ is a tuple of constants in $C_{\mathcal{M}(\mathcal{D})}$, but when we evaluate $\boldsymbol{q}$ over an interpretation $\mathcal{I}$, it returns tuples of elements of $\Delta^{\mathcal{I}}$. Therefore, we should actually require that $\vec{c}^{\mathcal{I}} \in \boldsymbol{q}(\mathcal{I})$, and not that $\vec{c} \in \boldsymbol{q}(\mathcal{I})$. However, due to the standard names assumption, we have that $\vec{c}^{\mathcal{I}} = \vec{c}$, so the two conditions are equivalent.

Query answering in VKGs – Certain answers

In VKGs, we want to answer queries formulated over the ontology, by using the data provided by the data sources through the mapping.

Consider our formalization of VKGs and a VKG instance $\mathcal{J}$.

Certain answers $\mathsf{cert}(\boldsymbol{q}, \mathcal{J})$ – Intuition

Given a VKG instance $\mathcal{J}$ and a query $\boldsymbol{q}$ over $\mathcal{J}$, the certain answers $\mathsf{cert}(\boldsymbol{q}, \mathcal{J})$ to $\boldsymbol{q}$ over $\mathcal{J}$ are those answers to $\boldsymbol{q}$ that hold in every model of $\mathcal{J}$.

Certain answers $\mathsf{cert}(\boldsymbol{q}, \mathcal{J})$ – Formal definition

Given a VKG instance $\mathcal{J} = \langle \mathcal{P}, \mathcal{D} \rangle$ and a query $\boldsymbol{q}$ over $\mathcal{J}$, a tuple $\vec{c}$ of constants in $C_{\mathcal{M}(\mathcal{D})}$ is a **certain answer** to $\boldsymbol{q}$ over $\mathcal{J}$, i.e., $\vec{c} \in \mathsf{cert}(\boldsymbol{q}, \mathcal{J})$, if for every model $\mathcal{I}$ of $\mathcal{J}$ we have that $\vec{c} \in \boldsymbol{q}(\mathcal{I})$.

Note: Each certain answer $\vec{c}$ is a tuple of constants in $C_{\mathcal{M}(\mathcal{D})}$, but when we evaluate $\boldsymbol{q}$ over an interpretation $\mathcal{I}$, it returns tuples of elements of $\Delta^{\mathcal{I}}$. Therefore, we should actually require that $\vec{c}^{\mathcal{I}} \in \boldsymbol{q}(\mathcal{I})$, and not that $\vec{c} \in \boldsymbol{q}(\mathcal{I})$. However, due to the standard names assumption, we have that $\vec{c}^{\mathcal{I}} = \vec{c}$, so the two conditions are equivalent.

Query answering in VKGs – Certain answers

In VKGs, we want to answer queries formulated over the ontology, by using the data provided by the data sources through the mapping.

Consider our formalization of VKGs and a VKG instance $\mathcal{J}$.

Certain answers cert$(\boldsymbol{q}, \mathcal{J})$ – Intuition

Given a VKG instance $\mathcal{J}$ and a query $\boldsymbol{q}$ over $\mathcal{J}$, the certain answers cert$(\boldsymbol{q}, \mathcal{J})$ to $\boldsymbol{q}$ over $\mathcal{J}$ are those answers to $\boldsymbol{q}$ that hold in every model of $\mathcal{J}$.

Certain answers cert$(\boldsymbol{q}, \mathcal{J})$ – Formal definition

Given a VKG instance $\mathcal{J} = \langle \mathcal{P}, \mathcal{D} \rangle$ and a query $\boldsymbol{q}$ over $\mathcal{J}$, a tuple $\vec{c}$ of constants in $C_{\mathcal{M}(\mathcal{D})}$ is a **certain answer** to $\boldsymbol{q}$ over $\mathcal{J}$, i.e., $\vec{c} \in$ cert$(\boldsymbol{q}, \mathcal{J})$, if for every model $\mathcal{I}$ of $\mathcal{J}$ we have that $\vec{c} \in \boldsymbol{q}(\mathcal{I})$.

Note: Each certain answer $\vec{c}$ is a tuple of constants in $C_{\mathcal{M}(\mathcal{D})}$, but when we evaluate $\boldsymbol{q}$ over an interpretation $\mathcal{I}$, it returns tuples of elements of $\Delta^{\mathcal{I}}$. Therefore, we should actually require that $\vec{c}^{\mathcal{I}} \in \boldsymbol{q}(\mathcal{I})$, and not that $\vec{c} \in \boldsymbol{q}(\mathcal{I})$. However, due to the standard names assumption, we have that $\vec{c}^{\mathcal{I}} = \vec{c}$, so the two conditions are equivalent.

First-order rewritability

To make computing certain answers viable in practice, the VKG setting relies on reducing it to evaluating SQL (i.e., first-order logic) queries over the data.

Consider a VKG specification $\mathcal{P} = \langle \mathcal{O}, \mathcal{M}, \mathcal{S} \rangle$.

First-order rewritability

A query $r(\vec{x})$ is a **first-order rewriting** of a query $q(\vec{x})$ with respect to $\mathcal{P}$ if, for every source DB $\mathcal{D}$,

certain answers to $q(\vec{x})$ over $\langle \mathcal{P}, \mathcal{D} \rangle$ $=$ answers to $r(\vec{x})$ over $\mathcal{D}$.

For OWL 2 QL ontologies and R2RML mappings,
(core) SPARQL queries are first-order rewritable.

In other words, **in VKGs, we can compute the certain answers to a SPARQL query by evaluating over the sources its rewriting, which is a SQL query.**

First-order rewritability

To make computing certain answers viable in practice, the VKG setting relies on reducing it to evaluating SQL (i.e., first-order logic) queries over the data.

Consider a VKG specification $\mathcal{P} = \langle \mathcal{O}, \mathcal{M}, \mathcal{S} \rangle$.

First-order rewritability

A query $r(\vec{x})$ is a **first-order rewriting** of a query $q(\vec{x})$ with respect to $\mathcal{P}$ if, for every source DB $\mathcal{D}$,

certain answers to $q(\vec{x})$ over $\langle \mathcal{P}, \mathcal{D} \rangle$ $=$ answers to $r(\vec{x})$ over $\mathcal{D}$.

For OWL 2 QL ontologies and R2RML mappings, (core) SPARQL queries are first-order rewritable.

In other words, **in VKGs, we can compute the certain answers to a SPARQL query by evaluating over the sources its rewriting, which is a SQL query.**

First-order rewritability

To make computing certain answers viable in practice, the VKG setting relies on reducing it to evaluating SQL (i.e., first-order logic) queries over the data.

Consider a VKG specification $\mathcal{P} = \langle \mathcal{O}, \mathcal{M}, \mathcal{S} \rangle$.

First-order rewritability

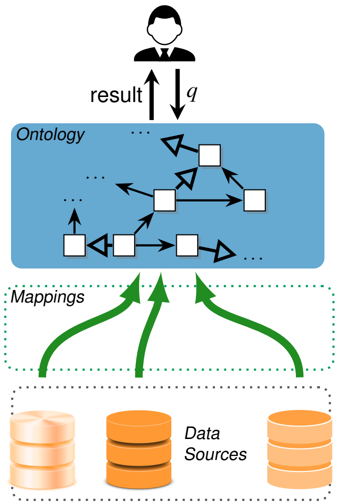
A query $r(\vec{x})$ is a **first-order rewriting** of a query $q(\vec{x})$ with respect to $\mathcal{P}$ if, for every source DB $\mathcal{D}$, certain answers to $q(\vec{x})$ over $\langle \mathcal{P}, \mathcal{D} \rangle$ = answers to $r(\vec{x})$ over $\mathcal{D}$.

For OWL 2 QL ontologies and R2RML mappings, (core) SPARQL queries are first-order rewritable.

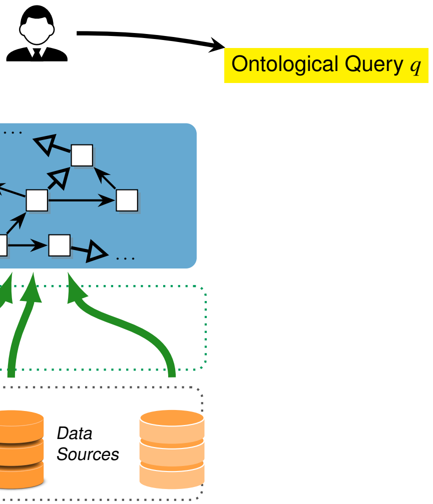
In other words, **in VKGs, we can compute the certain answers to a SPARQL query by evaluating over the sources its rewriting, which is a SQL query.**

Outline

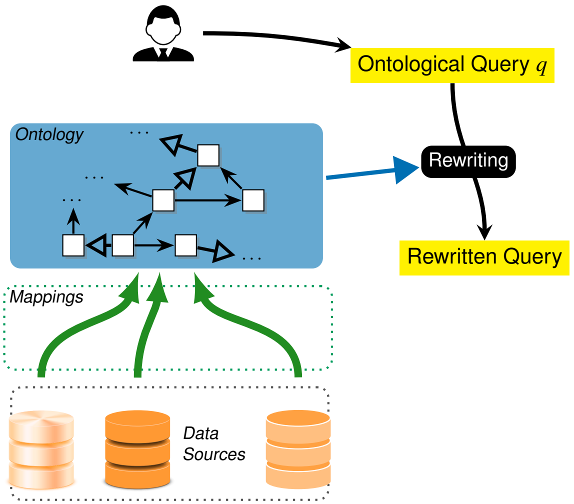
Query answering via query reformulation – Conceptual framework



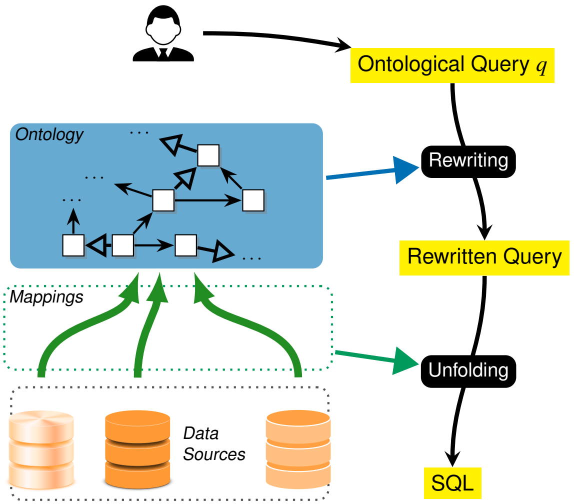
Query answering via query reformulation – Conceptual framework



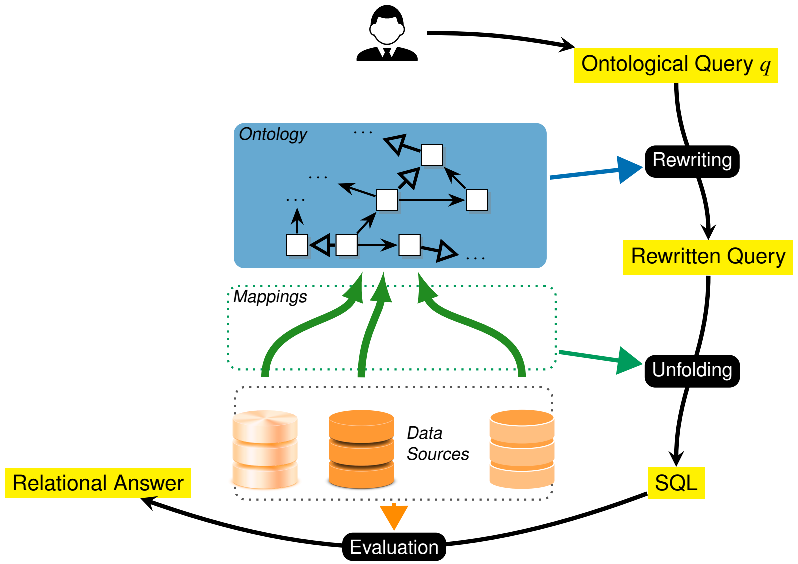
Query answering via query reformulation – Conceptual framework



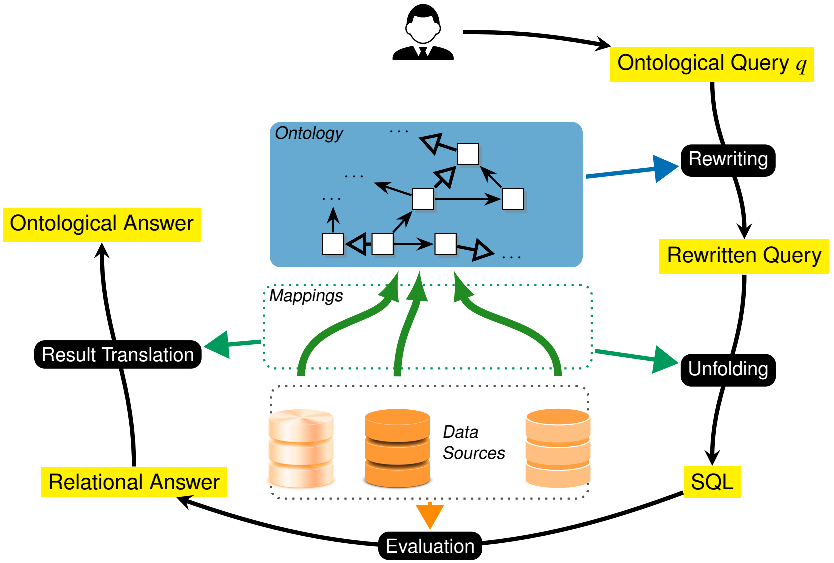
Query answering via query reformulation – Conceptual framework



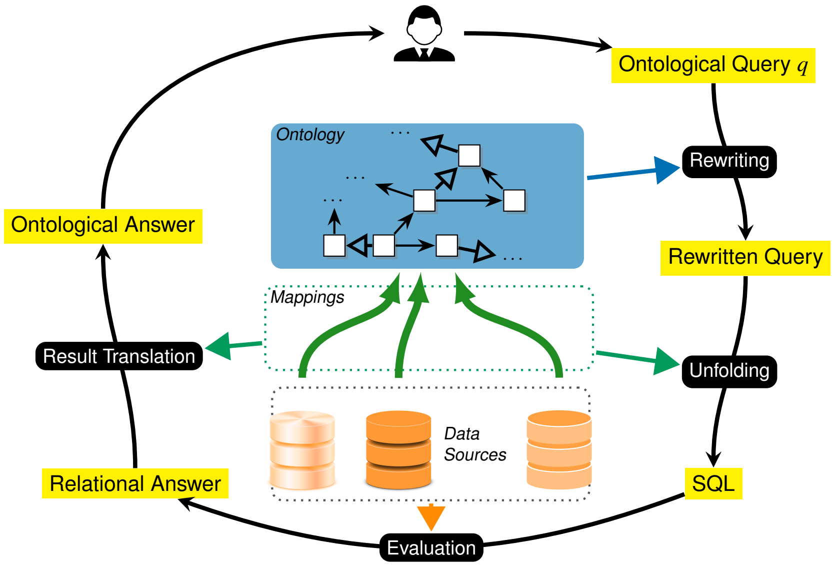
Query answering via query reformulation – Conceptual framework



Query answering via query reformulation – Conceptual framework



Query answering via query reformulation – Conceptual framework



Query answering via query reformulation – Optimizations needed

The above conceptual framework is realized as follows.

Computing certain answers to a SPARQL query q over a VKG instance $\langle \mathcal{P}, \mathcal{D} \rangle$, with $\mathcal{P} = \langle \mathcal{O}, \mathcal{M}, \mathcal{S} \rangle$:

1. Compute the perfect rewriting of q w.r.t. $\mathcal{O}$.
2. Unfold the perfect rewriting w.r.t. 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**.

We analyze now these steps more in detail.

Query answering via query reformulation – Optimizations needed

The above conceptual framework is realized as follows.

Computing certain answers to a SPARQL query q over a VKG instance $\langle \mathcal{P}, \mathcal{D} \rangle$, with $\mathcal{P} = \langle \mathcal{O}, \mathcal{M}, \mathcal{S} \rangle$:

1. Compute the perfect rewriting of q w.r.t. $\mathcal{O}$.
2. Unfold the perfect rewriting w.r.t. 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**.

We analyze now these steps more in detail.

Outline

Rewriting step

The rewriting Step ❶ deals with the knowledge encoded by the axioms of the ontology:

- hierarchies of classes and of properties;
- objects that are existentially implied by such axioms: existential reasoning.

We illustrate the need for dealing with these two aspects with two examples.

Dealing with hierarchies

Suppose that every graduate student is a student, i.e.,

$$\text{GraduateStudent} \sqsubseteq \text{Student}$$

and john is a graduate student: GraduateStudent(`john`).

What is the answer to the following query, asking for all students?

$$\text{q}(x) \leftarrow \text{Student}(x)$$

In SPARQL: `SELECT ?x WHERE { ?x a Student . }`
The answer should be `john`, since being a graduate student, he is also a student.

Rewritten Query:

```
SELECT ?x WHERE { { ?x a Student . } UNION { ?x a GraduateStudent . } }
```

Dealing with hierarchies

Suppose that every graduate student is a student, i.e.,

$$\text{GraduateStudent} \sqsubseteq \text{Student}$$

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

What is the answer to the following query, asking for all students?

$$\text{q}(x) \leftarrow \text{Student}(x)$$

In SPARQL: `SELECT ?x WHERE { ?x a Student . }`

The answer should be `john`, since being a graduate student, he is also a student.

Rewritten Query:

```
SELECT ?x WHERE { { ?x a Student . } UNION { ?x a GraduateStudent . } }
```

Dealing with hierarchies

Suppose that every graduate student is a student, i.e.,

$$\text{GraduateStudent} \sqsubseteq \text{Student}$$

and john is a graduate student: $\text{GraduateStudent}(\texttt{john})$.

What is the answer to the following query, asking for all students?

$$\text{q}(x) \leftarrow \text{Student}(x)$$

In SPARQL: `SELECT ?x WHERE { ?x a Student . }`
The answer should be `john`, since being a graduate student, he is also a student.

Rewritten Query:

```
SELECT ?x WHERE { { ?x a Student . } UNION { ?x a GraduateStudent . } }
```

Dealing with existential reasoning

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

$$\text{Student} \sqsubseteq \exists \text{isSupervisedBy}.\text{Professor}$$

and john is a student: $\text{Student}(\texttt{john})$.

What is the answer to the following query, asking for all individuals supervised by some professor?

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

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

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

Dealing with existential reasoning

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

$$\text{Student} \sqsubseteq \exists\text{isSupervisedBy.Professor}$$

and john is a student: $\text{Student}(\texttt{john})$.

What is the answer to the following query, asking for all individuals supervised by some professor?

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

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

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

Dealing with existential reasoning

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

$$\text{Student} \sqsubseteq \exists \text{isSupervisedBy}.\text{Professor}$$

and john is a student: $\text{Student}(\texttt{john})$.

What is the answer to the following query, asking for all individuals supervised by some professor?

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

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

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

The query rewriting algorithm

The **query rewriting** algorithm takes into account hierarchies and existential reasoning, by "compiling" the axioms of the ontology into the query.

Example

Consider the ontology axioms:

$$\begin{aligned}\text{Student} &\sqsubseteq \exists\text{isSupervisedBy}.\text{Professor}\\ \text{GraduateStudent} &\sqsubseteq \text{Student}\end{aligned}$$

Using these axioms, the rewriting algorithm rewrites the query

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

into a union of conjunctive queries (or a SPARQL union query):

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

Therefore, over the data Student(`john`), the rewritten query returns `john` as an answer.

Note: In *Ontop*, existential reasoning needs to be switched on explicitly, since it affects performance.

The query rewriting algorithm

The **query rewriting** algorithm takes into account hierarchies and existential reasoning, by “compiling” the axioms of the ontology into the query.

Example

Consider the ontology axioms:

$$\begin{aligned}\text{Student} &\sqsubseteq \exists\text{isSupervisedBy}.\text{Professor}\\ \text{GraduateStudent} &\sqsubseteq \text{Student}\end{aligned}$$

Using these axioms, the rewriting algorithm rewrites the query

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

into a union of conjunctive queries (or a SPARQL union query):

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

Therefore, over the data Student($\texttt{john}$), the rewritten query returns $\texttt{john}$ as an answer.

Note: In *Ontop*, existential reasoning needs to be switched on explicitly, since it affects performance.

Outline

Query unfolding

We consider now Step ② of reformulation, i.e., the unfolding w.r.t. the mapping $\mathcal{M}$.

In principle, we have two approaches to exploit the mapping:

- bottom-up approach: simpler, but typically less efficient
- top-down approach: more sophisticated, but also more efficient

Both approaches require to first **split** the set of atoms in the target queries of the mapping assertions into the constituent atoms.

Note: In the following, to make notation more compact, we represent an IRI-template of the form

$$\texttt{:xxx/}\{v_1\}\texttt{/}\{v_2\}\texttt{/}\cdots\texttt{/}\{v_n\}$$

more compactly as

$$\mathbf{xxx}(v_1, \ldots, v_n).$$

Query unfolding

We consider now Step ❷ of reformulation, i.e., the unfolding w.r.t. the mapping $\mathcal{M}$.

In principle, we have two approaches to exploit the mapping:

- bottom-up approach: simpler, but typically less efficient
- top-down approach: more sophisticated, but also more efficient

Both approaches require to first **split** the set of atoms in the target queries of the mapping assertions into the constituent atoms.

Note: In the following, to make notation more compact, we represent an IRI-template of the form

$$\texttt{:xxx/}\{v_1\}\texttt{/}\{v_2\}\texttt{/}\cdots\texttt{/}\{v_n\}$$

more compactly as

$$\mathbf{xxx}(v_1, \ldots, v_n).$$

Splitting of mappings

A mapping assertion $\Phi \rightsquigarrow \Psi$, where the target query Ψ is constituted by the atoms $X_1, \ldots, X_k$, can be split into k mapping assertions:

$$\Phi \rightsquigarrow X_1 \qquad \cdots \qquad \Phi \rightsquigarrow X_k$$

This is possible, since Ψ does not contain non-distinguished variables.

Example

m_1: `SELECT pcode, acode, aname FROM ACTOR` $\rightsquigarrow$ Play(**pl**($pcode$)),
Actor(**act**($acode$)),
name(**act**($acode$), $aname$),
actsIn(**act**($acode$), **pl**($pcode$))

is split into

m_1^1: `SELECT pcode, acode, aname FROM ACTOR` $\rightsquigarrow$ Play(**pl**($pcode$))
m_1^2: `SELECT pcode, acode, aname FROM ACTOR` $\rightsquigarrow$ Actor(**act**($acode$))
m_1^3: `SELECT pcode, acode, aname FROM ACTOR` $\rightsquigarrow$ name(**act**($acode$), $aname$)
m_1^4: `SELECT pcode, acode, aname FROM ACTOR` $\rightsquigarrow$ actsIn(**act**($acode$), **pl**($pcode$))

Bottom-up approach to deal with mappings: Materialization

Consists in a straightforward application of the mappings to the data:

1. Propagate the data from $\mathcal{D}$ through $\mathcal{M}$, **materializing** the RDF graph $\mathcal{V} = \mathcal{M}(\mathcal{D})$ (the constants in such an RDF graph are values and object terms obtained from the database values).
2. Apply to $\mathcal{V}$ and to the ontology $\mathcal{O}$, the query answering algorithm (based on query rewriting) developed for *DL-Lite* / OWL 2 QL.

This approach has several drawbacks:

- The technique is no more AC^0 in the size of the data, since the RDF graph $\mathcal{V}$ to materialize is in general polynomial in the size of the data.
- $\mathcal{V}$ may be very large, and thus it may be infeasible to actually materialize it.
- Freshness of $\mathcal{V}$ with respect to the underlying data source(s) may be an issue, and one would need to propagate source updates (cf. Data Warehousing).

Bottom-up approach to deal with mappings: Materialization

Consists in a straightforward application of the mappings to the data:

1. Propagate the data from $\mathcal{D}$ through $\mathcal{M}$, **materializing** the RDF graph $\mathcal{V} = \mathcal{M}(\mathcal{D})$ (the constants in such an RDF graph are values and object terms obtained from the database values).
2. Apply to $\mathcal{V}$ and to the ontology $\mathcal{O}$, the query answering algorithm (based on query rewriting) developed for *DL-Lite* / OWL 2 QL.

This approach has several drawbacks:

- The technique is no more AC^0 in the size of the data, since the RDF graph $\mathcal{V}$ to materialize is in general polynomial in the size of the data.
- $\mathcal{V}$ may be very large, and thus it may be infeasible to actually materialize it.
- Freshness of $\mathcal{V}$ with respect to the underlying data source(s) may be an issue, and one would need to propagate source updates (cf. Data Warehousing).

Top-down approach to deal with mappings: Unfolding

The top-down approach is realized by computing from the (rewritten) query q_{r} a new query q_{unf}, by **unfolding** q_{r} using (the split version of) the mappings $\mathcal{M}$.

Consider the mapping assertions $\Phi_i \rightsquigarrow \Psi_i$.

- Essentially, each atom in q_{r} that unifies with an atom in some Ψ_i is substituted with the corresponding query Φ_i over the database.
- The unfolded query q_{unf} is such that for each database $\mathcal{D}$ we have that:

$$q_{\mathsf{unf}}(\mathcal{D}) \; = \; \mathit{Eval}_{\mathsf{CWA}}(q_{\mathsf{r}}, \mathcal{M}(\mathcal{D})).$$

Unfolding

To unfold a query q_r with respect to a set $\mathcal{M}$ of mapping assertions:

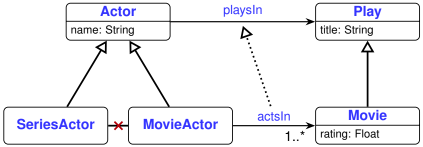
1. For each non-split mapping assertion $\Phi_i(\vec{x}) \rightsquigarrow \Psi_i(\vec{t}, \vec{y})$:
   1. Introduce a **view symbol** Aux_i of arity equal to that of Φ_i.
   2. Add a **view definition** $\mathsf{Aux}_i(\vec{x}) \leftarrow \Phi_i(\vec{x})$.

2. For each split version $\Phi_i(\vec{x}) \rightsquigarrow X_i^j(\vec{t}, \vec{y})$ of a mapping assertion, introduce a **clause** $X_i^j(\vec{t}, \vec{y}) \leftarrow \mathsf{Aux}_i(\vec{x})$.

3. Obtain from q_r in all possible ways queries q_{aux} defined over the view symbols Aux_i as follows:
   1. Find a most general unifier ϑ that unifies each atom $X(\vec{z})$ in the body of q_r with the head of a clause $X(\vec{t}, \vec{y}) \leftarrow \mathsf{Aux}_i(\vec{x})$.
   2. Substitute each atom $X(\vec{z})$ with $\vartheta(\mathsf{Aux}_i(\vec{x}))$, i.e., with the body the unified clause to which the unifier ϑ is applied.

4. The unfolded query q_{unf} is the **union** of all queries q_{aux}, together with the view definitions for the predicates Aux_i appearing in q_{aux}.

Unfolding – Example



m_1:
```
SELECT pcode, acode, aname
FROM ACTOR
```
$\rightsquigarrow$ Play(**pl**($pcode$)), Actor(**act**($acode$)), name(**act**($acode$), $aname$), actsIn(**act**($acode$), **pl**($pcode$))

m_2:
```
SELECT mcode, acode, mtitle
FROM MOVIE M, ACTOR A
WHERE M.mcode = A.pcode
  AND M.type = "m"
```
$\rightsquigarrow$ Movie(**pl**($mcode$)), playsIn(**act**($acode$), **pl**($mcode$)), title(**pl**($mcode$), $mtitle$)

We define a view Aux$_i$ for the source query of each mapping m_i.

For each (split) mapping assertion, we introduce a clause:

$$\begin{array}{rcl}
\text{Play}(\textbf{pl}(pcode)) & \leftarrow & \text{Aux}_1(pcode, \_, \_) \\
\text{Actor}(\textbf{act}(acode)) & \leftarrow & \text{Aux}_1(\_, acode, \_) \\
\text{name}(\textbf{act}(acode), aname) & \leftarrow & \text{Aux}_1(\_, acode, aname) \\
\text{actsIn}(\textbf{act}(acode), \textbf{pl}(pcode)) & \leftarrow & \text{Aux}_1(pcode, acode, \_) \\
\text{Movie}(\textbf{pl}(mcode)) & \leftarrow & \text{Aux}_2(mcode, \_, \_) \\
\text{playsIn}(\textbf{act}(acode), \textbf{pl}(mcode)) & \leftarrow & \text{Aux}_2(mcode, acode, \_) \\
\text{title}(\textbf{pl}(mcode), mtitle) & \leftarrow & \text{Aux}_2(mcode, \_, mtitle)
\end{array}$$

Unfolding – Example (cont'd)

Query over the ontology: Actors with their name who act in a movie whose title is `"The Matrix"`:

$\boldsymbol{q}(a, n) \leftarrow \mathsf{Actor}(a), \mathsf{name}(a, n), \mathsf{actsIn}(a, p), \mathsf{Movie}(p), \mathsf{title}(p, \texttt{"The Matrix"})$

$$\begin{aligned}
\mathsf{Actor}(\mathbf{act}(acode)) &\leftarrow \mathsf{Aux}_1(\_, acode, \_)\\
\mathsf{name}(\mathbf{act}(acode), aname) &\leftarrow \mathsf{Aux}_1(\_, acode, aname)\\
\mathsf{actsIn}(\mathbf{act}(acode), \mathbf{pl}(pcode)) &\leftarrow \mathsf{Aux}_1(pcode, acode, \_)\\
\mathsf{Movie}(\mathbf{pl}(mcode)) &\leftarrow \mathsf{Aux}_2(mcode, \_, \_)\\
\mathsf{title}(\mathbf{pl}(mcode), mtitle) &\leftarrow \mathsf{Aux}_2(mcode, \_, mtitle)
\end{aligned}$$

A unifier ϑ between the atoms in $\boldsymbol{q}$ and the clause heads is:

$\vartheta(a) = \mathbf{act}(acode) \qquad \vartheta(n) = aname$

$\vartheta(p) = \mathbf{pl}(pcode) \qquad \vartheta(mcode) = pcode \qquad \vartheta(mtitle) = \texttt{"The Matrix"}$

After applying ϑ to $\boldsymbol{q}$, we obtain:

$$\begin{aligned}
\boldsymbol{q}(\mathbf{act}(acode), aname) \leftarrow\ & \mathsf{Actor}(\mathbf{act}(acode)),\ \mathsf{name}(\mathbf{act}(acode), aname),\ \mathsf{actsIn}(\mathbf{act}(acode), \mathbf{pl}(pcode)),\\
& \mathsf{Movie}(\mathbf{pl}(pcode)),\ \mathsf{title}(\mathbf{pl}(pcode), \texttt{"The Matrix"})
\end{aligned}$$

Substituting the atoms with the bodies of the clauses (after having applied the unifier), we obtain:

$$\begin{aligned}
\boldsymbol{q}(\mathbf{act}(acode), aname) \leftarrow\ & \mathsf{Aux}_1(\_, acode, \_),\ \mathsf{Aux}_1(\_, acode, aname),\ \mathsf{Aux}_1(pcode, acode, \_),\\
& \mathsf{Aux}_2(pcode, \_, \_),\ \mathsf{Aux}_2(pcode, \_, \texttt{"The Matrix"})
\end{aligned}$$

Unfolding – Example (cont'd)

Query over the ontology: Actors with their name who act in a movie whose title is `"The Matrix"`:

$\boldsymbol{q}(a,n) \leftarrow \mathsf{Actor}(a), \mathsf{name}(a,n), \mathsf{actsIn}(a,p), \mathsf{Movie}(p), \mathsf{title}(p, \texttt{"The Matrix"})$

$$
\begin{aligned}
\mathsf{Actor}(\mathbf{act}(acode)) &\leftarrow \mathsf{Aux}_1(\_, acode, \_)\\
\mathsf{name}(\mathbf{act}(acode), aname) &\leftarrow \mathsf{Aux}_1(\_, acode, aname)\\
\mathsf{actsIn}(\mathbf{act}(acode), \mathbf{pl}(pcode)) &\leftarrow \mathsf{Aux}_1(pcode, acode, \_)\\
\mathsf{Movie}(\mathbf{pl}(mcode)) &\leftarrow \mathsf{Aux}_2(mcode, \_, \_)\\
\mathsf{title}(\mathbf{pl}(mcode), mtitle) &\leftarrow \mathsf{Aux}_2(mcode, \_, mtitle)
\end{aligned}
$$

A unifier ϑ between the atoms in $\boldsymbol{q}$ and the clause heads is:

$\vartheta(a) = \mathbf{act}(acode) \qquad \vartheta(n) = aname$

$\vartheta(p) = \mathbf{pl}(pcode) \qquad \vartheta(mcode) = pcode \qquad \vartheta(mtitle) = \texttt{"The Matrix"}$

After applying ϑ to $\boldsymbol{q}$, we obtain:

$\boldsymbol{q}(\mathbf{act}(acode), aname) \leftarrow \mathsf{Actor}(\mathbf{act}(acode)), \ \mathsf{name}(\mathbf{act}(acode), aname), \ \mathsf{actsIn}(\mathbf{act}(acode), \mathbf{pl}(pcode)),$
$\mathsf{Movie}(\mathbf{pl}(pcode)), \ \mathsf{title}(\mathbf{pl}(pcode), \texttt{"The Matrix"})$

Substituting the atoms with the bodies of the clauses (after having applied the unifier), we obtain:

$\boldsymbol{q}(\mathbf{act}(acode), aname) \leftarrow \mathsf{Aux}_1(\_, acode, \_), \ \mathsf{Aux}_1(\_, acode, aname), \ \mathsf{Aux}_1(pcode, acode, \_),$
$\mathsf{Aux}_2(pcode, \_, \_), \ \mathsf{Aux}_2(pcode, \_, \texttt{"The Matrix"})$

Unfolding – Example (cont'd)

Query over the ontology: Actors with their name who act in a movie whose title is "`The Matrix`":

$\boldsymbol{q}(a,n) \leftarrow \mathsf{Actor}(a), \mathsf{name}(a,n), \mathsf{actsIn}(a,p), \mathsf{Movie}(p), \mathsf{title}(p, \texttt{"The Matrix"})$

$$\begin{array}{rcl} \mathsf{Actor}(\textbf{act}(acode)) & \leftarrow & \mathsf{Aux}_1(\_, acode, \_) \\ \mathsf{name}(\textbf{act}(acode), aname) & \leftarrow & \mathsf{Aux}_1(\_, acode, aname) \\ \mathsf{actsIn}(\textbf{act}(acode), \textbf{pl}(pcode)) & \leftarrow & \mathsf{Aux}_1(pcode, acode, \_) \\ \mathsf{Movie}(\textbf{pl}(mcode)) & \leftarrow & \mathsf{Aux}_2(mcode, \_, \_) \\ \mathsf{title}(\textbf{pl}(mcode), mtitle) & \leftarrow & \mathsf{Aux}_2(mcode, \_, mtitle) \end{array}$$

A unifier ϑ between the atoms in $\boldsymbol{q}$ and the clause heads is:

$\vartheta(a) = \textbf{act}(acode)$ $\quad \vartheta(n) = aname$

$\vartheta(p) = \textbf{pl}(pcode)$ $\quad \vartheta(mcode) = pcode$ $\quad \vartheta(mtitle) = \texttt{"The Matrix"}$

After applying ϑ to $\boldsymbol{q}$, we obtain:

$\boldsymbol{q}(\textbf{act}(acode), aname) \leftarrow \mathsf{Actor}(\textbf{act}(acode)), \; \mathsf{name}(\textbf{act}(acode), aname), \; \mathsf{actsIn}(\textbf{act}(acode), \textbf{pl}(pcode)),$
$\mathsf{Movie}(\textbf{pl}(pcode)), \; \mathsf{title}(\textbf{pl}(pcode), \texttt{"The Matrix"})$

Substituting the atoms with the bodies of the clauses (after having applied the unifier), we obtain:

$\boldsymbol{q}(\textbf{act}(acode), aname) \leftarrow \mathsf{Aux}_1(\_, acode, \_), \; \mathsf{Aux}_1(\_, acode, aname), \; \mathsf{Aux}_1(pcode, acode, \_),$
$\mathsf{Aux}_2(pcode, \_, \_), \; \mathsf{Aux}_2(pcode, \_, \texttt{"The Matrix"})$

Unfolding – Example (cont'd)

Query over the ontology: Actors with their name who act in a movie whose title is `"The Matrix"`:

$q(a,n) \leftarrow \mathsf{Actor}(a), \mathsf{name}(a,n), \mathsf{actsIn}(a,p), \mathsf{Movie}(p), \mathsf{title}(p, \texttt{"The Matrix"})$

$$\begin{aligned}
\mathsf{Actor}(\mathbf{act}(acode)) &\leftarrow \mathsf{Aux}_1(\_, acode, \_)\\
\mathsf{name}(\mathbf{act}(acode), aname) &\leftarrow \mathsf{Aux}_1(\_, acode, aname)\\
\mathsf{actsIn}(\mathbf{act}(acode), \mathbf{pl}(pcode)) &\leftarrow \mathsf{Aux}_1(pcode, acode, \_)\\
\mathsf{Movie}(\mathbf{pl}(mcode)) &\leftarrow \mathsf{Aux}_2(mcode, \_, \_)\\
\mathsf{title}(\mathbf{pl}(mcode), mtitle) &\leftarrow \mathsf{Aux}_2(mcode, \_, mtitle)
\end{aligned}$$

A unifier ϑ between the atoms in q and the clause heads is:

$\vartheta(a) = \mathbf{act}(acode) \qquad \vartheta(n) = aname$

$\vartheta(p) = \mathbf{pl}(pcode) \qquad \vartheta(mcode) = pcode \qquad \vartheta(mtitle) = \texttt{"The Matrix"}$

After applying ϑ to q, we obtain:

$$\begin{aligned}
q(\mathbf{act}(acode), aname) \leftarrow\ & \mathsf{Actor}(\mathbf{act}(acode)),\ \mathsf{name}(\mathbf{act}(acode), aname),\ \mathsf{actsIn}(\mathbf{act}(acode), \mathbf{pl}(pcode)),\\
& \mathsf{Movie}(\mathbf{pl}(pcode)),\ \mathsf{title}(\mathbf{pl}(pcode), \texttt{"The Matrix"})
\end{aligned}$$

Substituting the atoms with the bodies of the clauses (after having applied the unifier), we obtain:

$$\begin{aligned}
q(\mathbf{act}(acode), aname) \leftarrow\ & \mathsf{Aux}_1(\_, acode, \_),\ \mathsf{Aux}_1(\_, acode, aname),\ \mathsf{Aux}_1(pcode, acode, \_),\\
& \mathsf{Aux}_2(pcode, \_, \_),\ \mathsf{Aux}_2(pcode, \_, \texttt{"The Matrix"})
\end{aligned}$$

Exponential blowup in the unfolding

When there are multiple mapping assertions for each atom, the unfolded query may be exponential in the original one.

Consider a query: $q(y) \leftarrow C_1(y), C_2(y), \ldots, C_n(y)$

and the mappings: $m_i^1 \colon \Phi_i^1(x) \leadsto C_i(\textbf{iri}(x))$ (for $i \in \{1, \ldots, n\}$)
$m_i^2 \colon \Phi_i^2(x) \leadsto C_i(\textbf{iri}(x))$

We add the view definitions: $\mathsf{Aux}_i^j(x) \leftarrow \Phi_i^j(x)$
and introduce the clauses: $C_i(\textbf{iri}(x)) \leftarrow \mathsf{Aux}_i^j(x)$ (for $i \in \{1, \ldots, n\}, j \in \{1, 2\}$).

There is a single unifier, namely $\vartheta(y) = \textbf{iri}(x)$, but each atom $C_i(y)$ in the query unifies with the head of two clauses.

Hence, we obtain one unfolded query

$$q(\textbf{iri}(x)) \leftarrow \mathsf{Aux}_1^{j_1}(x), \mathsf{Aux}_2^{j_2}(x), \ldots, \mathsf{Aux}_n^{j_n}(x)$$

for each possible combination of $j_i \in \{1, 2\}$, for $i \in \{1, \ldots, n\}$.
Hence, we obtain 2^n **unfolded queries**.

Exponential blowup in the unfolding

When there are multiple mapping assertions for each atom, the unfolded query may be exponential in the original one.

Consider a query: $\boldsymbol{q}(y) \leftarrow C_1(y), C_2(y), \ldots, C_n(y)$

and the mappings: $m_i^1 \colon \Phi_i^1(x) \rightsquigarrow C_i(\textbf{iri}(x))$ (for $i \in \{1, \ldots, n\}$)
$m_i^2 \colon \Phi_i^2(x) \rightsquigarrow C_i(\textbf{iri}(x))$

We add the view definitions: $\mathsf{Aux}_i^j(x) \leftarrow \Phi_i^j(x)$
and introduce the clauses: $C_i(\textbf{iri}(x)) \leftarrow \mathsf{Aux}_i^j(x)$ (for $i \in \{1, \ldots, n\}, j \in \{1, 2\}$).

There is a single unifier, namely $\vartheta(y) = \textbf{iri}(x)$, but each atom $C_i(y)$ in the query unifies with the head of two clauses.

Hence, we obtain one unfolded query

$$\boldsymbol{q}(\textbf{iri}(x)) \leftarrow \mathsf{Aux}_1^{j_1}(x), \mathsf{Aux}_2^{j_2}(x), \ldots, \mathsf{Aux}_n^{j_n}(x)$$

for each possible combination of $j_i \in \{1, 2\}$, for $i \in \{1, \ldots, n\}$.
Hence, we obtain 2^n **unfolded queries**.

Implementation of top-down approach to query answering

To implement the top-down approach, we need to generate an SQL query.

We can follow different strategies:

1. Substitute each view predicate in the unfolded queries with the corresponding SQL query over the source:
   - + joins are performed on the DB attributes, hence can be done efficiently, e.g., by exploiting indexes;
   - + does not generate doubly nested queries;
   - – the number of unfolded queries may be exponential.

2. Construct for each atom in the original query a new view. This view takes the union of all SQL queries corresponding to the view predicates, and constructs also the IRIs based on the IRI templates:
   - + avoids exponential blow-up of the resulting query, since the union (of the queries coming from multiple mappings) is done before the joins;
   - – joins are performed on IRIs, i.e., on terms built using string concatenation, hence are highly inefficient;
   - – generates doubly nested queries, which per se the database has difficulty in optimizing.

Which method is better, depends on various parameters, and there is no definitive answer.
In general, one needs a mixed approach that applies different strategies to different parts of the query.

Outline

Contributions of rewriting and unfolding

- We are interested in computing certain answers to SPARQL queries over a VKG instance $\langle \mathcal{P}, \mathcal{D} \rangle$, with $\mathcal{P} = \langle \mathcal{O}, \mathcal{M}, \mathcal{S} \rangle$.
- In practice, by computing the rewriting q_r of q w.r.t. $\mathcal{O}$ and its unfolding w.r.t. $\mathcal{M}$, the resulting query q_unf might become very large, and costly to execute over $\mathcal{D}$.

Let us consider the contributions of rewriting and unfolding to the query answers:

- In principle, evaluating the unfolding q_unf (of q_r w.r.t. $\mathcal{M}$) over $\mathcal{D}$, gives the same result as evaluating q_r over the RDF graph $\mathcal{V} = \mathcal{M}(\mathcal{D})$ extracted through the mapping $\mathcal{M}$ from the data $\mathcal{D}$.
- Instead, the impact of the rewriting on the query answers consists of two components:
  1. the rewriting w.r.t. class and property hierarchies (including domain and range assertions), i.e., $C_1 \sqsubseteq C_2, \quad P_1 \sqsubseteq P_2, \quad \exists P \sqsubseteq C, \quad \exists P^- \sqsubseteq C$;
  2. the rewriting taking into account existential reasoning, i.e., $C \sqsubseteq \exists R, \quad C_1 \sqsubseteq \exists R.C_2$.

Note: Component 1 corresponds to computing the saturation $\mathcal{V}_\text{sat}$ of $\mathcal{V}$ w.r.t. class and property hierarchies, while component 2 can be handled only through rewriting.

Contributions of rewriting and unfolding

- We are interested in computing certain answers to SPARQL queries over a VKG instance $\langle \mathcal{P}, \mathcal{D} \rangle$, with $\mathcal{P} = \langle \mathcal{O}, \mathcal{M}, \mathcal{S} \rangle$.
- In practice, by computing the rewriting q_r of q w.r.t. $\mathcal{O}$ and its unfolding w.r.t. $\mathcal{M}$, the resulting query q_{unf} might become very large, and costly to execute over $\mathcal{D}$.

Let us consider the contributions of rewriting and unfolding to the query answers:

- In principle, evaluating the unfolding q_{unf} (of q_r w.r.t. $\mathcal{M}$) over $\mathcal{D}$, gives the same result as evaluating q_r over the RDF graph $\mathcal{V} = \mathcal{M}(\mathcal{D})$ extracted through the mapping $\mathcal{M}$ from the data $\mathcal{D}$.
- Instead, the impact of the rewriting on the query answers consists of two components:
  1. the rewriting w.r.t. class and property hierarchies (including domain and range assertions), i.e., $C_1 \sqsubseteq C_2$, $P_1 \sqsubseteq P_2$, $\exists P \sqsubseteq C$, $\exists P^- \sqsubseteq C$;
  2. the rewriting taking into account existential reasoning, i.e., $C \sqsubseteq \exists R$, $C_1 \sqsubseteq \exists R.C_2$.

Note: Component 1 corresponds to computing the saturation $\mathcal{V}_{\text{sat}}$ of $\mathcal{V}$ w.r.t. class and property hierarchies, while component 2 can be handled only through rewriting.

Tree-witness rewriting and saturated mapping

We want to avoid materializing $\mathcal{V}$ and $\mathcal{V}_{\text{sat}}$, but also want to avoid computing the query rewriting w.r.t. class and property hierarchies.

Therefore we proceed as follows:

1. We rewrite q only w.r.t. the inclusion assertions that cause existential reasoning (i.e., $C \sqsubseteq \exists R$, $C_1 \sqsubseteq \exists R.C_2$).
   $\rightsquigarrow$ **tree-witness rewriting** q_{tw} [Kikot et al. 2012]
2. We use instead class and property hierarchies (i.e., $C_1 \sqsubseteq C_2$, $P_1 \sqsubseteq P_2$) to enrich the mapping $\mathcal{M}$.
   $\rightsquigarrow$ **saturated mapping** $\mathcal{M}_{\text{sat}}$ [Rodriguez-Muro et al. 2013; Kontchakov, Rezk, et al. 2014]
3. We unfold the tree-witness rewriting q_{tw} w.r.t. the saturated mapping $\mathcal{M}_{\text{sat}}$.

It is possible to show that the resulting query is equivalent to the perfect rewriting q_{r} (as obtained, e.g., through ordinary rewriting w.r.t. $\mathcal{O}$ and unfolding w.r.t. $\mathcal{M}$).

For more details, we refer also to [Kontchakov & Zakharyaschev 2014].

Saturated mapping

Intuitively, the **saturated mapping** $\mathcal{M}_{\text{sat}}$ is obtained as the composition of $\mathcal{M}$ and the ontology $\mathcal{O}$.

For each mapping assertion in $\mathcal{M}$	and each TBox assertion in $\mathcal{O}$	we add a mapping assertion to $\mathcal{M}_{\text{sat}}$
$\Phi(x) \rightsquigarrow C_1(\textbf{iri}(x))$	$C_1 \sqsubseteq C_2$	$\Phi(x) \rightsquigarrow C_2(\textbf{iri}(x))$
$\Phi(x,y) \rightsquigarrow P(\textbf{iri}_1(x), \textbf{iri}_2(y))$	$\exists P \sqsubseteq C_1$	$\Phi(x,y) \rightsquigarrow C_1(\textbf{iri}_1(x))$
$\Phi(x,y) \rightsquigarrow P(\textbf{iri}_1(x), \textbf{iri}_2(y))$	$\exists P^- \sqsubseteq C_2$	$\Phi(x,y) \rightsquigarrow C_2(\textbf{iri}_2(y))$
$\Phi(x,y) \rightsquigarrow P_1(\textbf{iri}_1(x), \textbf{iri}_2(y))$	$P_1 \sqsubseteq P_2$	$\Phi(x,y) \rightsquigarrow P_2(\textbf{iri}_1(x), \textbf{iri}_2(y))$

Due to saturation, $\mathcal{M}_{\text{sat}}$ will contain at most $|\mathcal{O}| \cdot |\mathcal{M}|$ many mappings.

Note: The saturated mapping has also been called **T-mapping** in the literature.

Saturated mapping

Intuitively, the **saturated mapping** $\mathcal{M}_{\text{sat}}$ is obtained as the composition of $\mathcal{M}$ and the ontology $\mathcal{O}$.

For each mapping assertion in $\mathcal{M}$	and each TBox assertion in $\mathcal{O}$	we add a mapping assertion to $\mathcal{M}_{\text{sat}}$
$\Phi(x) \rightsquigarrow C_1(\mathbf{iri}(x))$	$C_1 \sqsubseteq C_2$	$\Phi(x) \rightsquigarrow C_2(\mathbf{iri}(x))$
$\Phi(x,y) \rightsquigarrow P(\mathbf{iri}_1(x), \mathbf{iri}_2(y))$	$\exists P \sqsubseteq C_1$	$\Phi(x,y) \rightsquigarrow C_1(\mathbf{iri}_1(x))$
$\Phi(x,y) \rightsquigarrow P(\mathbf{iri}_1(x), \mathbf{iri}_2(y))$	$\exists P^- \sqsubseteq C_2$	$\Phi(x,y) \rightsquigarrow C_2(\mathbf{iri}_2(y))$
$\Phi(x,y) \rightsquigarrow P_1(\mathbf{iri}_1(x), \mathbf{iri}_2(y))$	$P_1 \sqsubseteq P_2$	$\Phi(x,y) \rightsquigarrow P_2(\mathbf{iri}_1(x), \mathbf{iri}_2(y))$

Due to saturation, $\mathcal{M}_{\text{sat}}$ will contain at most $|\mathcal{O}| \cdot |\mathcal{M}|$ many mappings.

Note: The saturated mapping has also been called **T-mapping** in the literature.

Saturated mapping

Intuitively, the **saturated mapping** $\mathcal{M}_{\text{sat}}$ is obtained as the composition of $\mathcal{M}$ and the ontology $\mathcal{O}$.

For each mapping assertion in $\mathcal{M}$	and each TBox assertion in $\mathcal{O}$	we add a mapping assertion to $\mathcal{M}_{\text{sat}}$
$\Phi(x) \rightsquigarrow C_1(\textbf{iri}(x))$	$C_1 \sqsubseteq C_2$	$\Phi(x) \rightsquigarrow C_2(\textbf{iri}(x))$
$\Phi(x,y) \rightsquigarrow P(\textbf{iri}_1(x), \textbf{iri}_2(y))$	$\exists P \sqsubseteq C_1$	$\Phi(x,y) \rightsquigarrow C_1(\textbf{iri}_1(x))$
$\Phi(x,y) \rightsquigarrow P(\textbf{iri}_1(x), \textbf{iri}_2(y))$	$\exists P^- \sqsubseteq C_2$	$\Phi(x,y) \rightsquigarrow C_2(\textbf{iri}_2(y))$
$\Phi(x,y) \rightsquigarrow P_1(\textbf{iri}_1(x), \textbf{iri}_2(y))$	$P_1 \sqsubseteq P_2$	$\Phi(x,y) \rightsquigarrow P_2(\textbf{iri}_1(x), \textbf{iri}_2(y))$

Due to saturation, $\mathcal{M}_{\text{sat}}$ will contain at most $|\mathcal{O}| \cdot |\mathcal{M}|$ many mappings.

Note: The saturated mapping has also been called **T-mapping** in the literature.

Saturated mapping – Exercise

Ontology $\mathcal{O}$

$$\begin{aligned}
\text{Student} &\sqsubseteq \text{Person}\\
\text{PostDoc} &\sqsubseteq \text{Faculty}\\
\text{Professor} &\sqsubseteq \text{Faculty}\\
\exists\text{teaches} &\sqsubseteq \text{Faculty}\\
\text{Faculty} &\sqsubseteq \text{Person}
\end{aligned}$$

User-defined mapping assertions $\mathcal{M}$

$$\begin{aligned}
&\texttt{student}(scode, fn, ln) && \rightsquigarrow \text{Student}(\mathbf{iri1}(scode)) && (1)\\
&\texttt{academic}(acode, fn, ln, pos),\ pos = 9 && \rightsquigarrow \text{PostDoc}(\mathbf{iri2}(acode)) && (2)\\
&\texttt{academic}(acode, fn, ln, pos),\ pos = 2 && \rightsquigarrow \text{Professor}(\mathbf{iri2}(acode)) && (3)\\
&\texttt{teaching}(course, acode) && \rightsquigarrow \text{teaches}(\mathbf{iri2}(acode), \mathbf{iri3}(course)) && (4)\\
&\texttt{academic}(acode, fn, ln, pos) && \rightsquigarrow \text{Faculty}(\mathbf{iri2}(acode)) && (5)
\end{aligned}$$

Saturated mapping – Exercise

Ontology $\mathcal{O}$

$$\begin{aligned}
\text{Student} &\sqsubseteq \text{Person}\\
\text{PostDoc} &\sqsubseteq \text{Faculty}\\
\text{Professor} &\sqsubseteq \text{Faculty}\\
\exists\text{teaches} &\sqsubseteq \text{Faculty}\\
\text{Faculty} &\sqsubseteq \text{Person}
\end{aligned}$$

User-defined mapping assertions $\mathcal{M}$

$$\begin{array}{llr}
\texttt{student}(scode, fn, ln) & \rightsquigarrow \text{Student}(\textbf{iri1}(scode)) & (1)\\
\texttt{academic}(acode, fn, ln, pos),\ pos = 9 & \rightsquigarrow \text{PostDoc}(\textbf{iri2}(acode)) & (2)\\
\texttt{academic}(acode, fn, ln, pos),\ pos = 2 & \rightsquigarrow \text{Professor}(\textbf{iri2}(acode)) & (3)\\
\texttt{teaching}(course, acode) & \rightsquigarrow \text{teaches}(\textbf{iri2}(acode), \textbf{iri3}(course)) & (4)\\
\texttt{academic}(acode, fn, ln, pos) & \rightsquigarrow \text{Faculty}(\textbf{iri2}(acode)) & (5)
\end{array}$$

By **saturating the mapping**, we obtain $\mathcal{M}_{\text{sat}}$, containing additional mapping assertions

Saturated mapping – Exercise

Ontology $\mathcal{O}$

$$\begin{aligned} \text{Student} &\sqsubseteq \text{Person} \\ \text{PostDoc} &\sqsubseteq \text{Faculty} \\ \text{Professor} &\sqsubseteq \text{Faculty} \\ \exists\text{teaches} &\sqsubseteq \text{Faculty} \\ \text{Faculty} &\sqsubseteq \text{Person} \end{aligned}$$

User-defined mapping assertions $\mathcal{M}$

$$\begin{array}{lcl r} \texttt{student}(scode, fn, ln) & \rightsquigarrow & \text{Student}(\textbf{iri1}(scode)) & (1) \\ \texttt{academic}(acode, fn, ln, pos),\ pos = 9 & \rightsquigarrow & \text{PostDoc}(\textbf{iri2}(acode)) & (2) \\ \texttt{academic}(acode, fn, ln, pos),\ pos = 2 & \rightsquigarrow & \text{Professor}(\textbf{iri2}(acode)) & (3) \\ \texttt{teaching}(course, acode) & \rightsquigarrow & \text{teaches}(\textbf{iri2}(acode), \textbf{iri3}(course)) & (4) \\ \texttt{academic}(acode, fn, ln, pos) & \rightsquigarrow & \text{Faculty}(\textbf{iri2}(acode)) & (5) \end{array}$$

By **saturating the mapping**, we obtain $\mathcal{M}_{\text{sat}}$, containing additional mapping assertions for the classes Faculty and Person.

$$\begin{array}{lcl r} \texttt{student}(scode, fn, ln) & \rightsquigarrow & \text{Person}(\textbf{iri1}(scode)) & (6) \\ \texttt{academic}(acode, fn, ln, pos),\ pos = 9 & \rightsquigarrow & \text{Faculty}(\textbf{iri2}(acode)) & (7) \\ \texttt{academic}(acode, fn, ln, pos),\ pos = 9 & \rightsquigarrow & \text{Person}(\textbf{iri2}(acode)) & (8) \\ \texttt{academic}(acode, fn, ln, pos),\ pos = 2 & \rightsquigarrow & \text{Faculty}(\textbf{iri2}(acode)) & (9) \\ \texttt{academic}(acode, fn, ln, pos),\ pos = 2 & \rightsquigarrow & \text{Person}(\textbf{iri2}(acode)) & (10) \\ \texttt{academic}(acode, fn, ln, pos) & \rightsquigarrow & \text{Person}(\textbf{iri2}(acode)) & (11) \\ \texttt{teaching}(course, acode) & \rightsquigarrow & \text{Faculty}(\textbf{iri2}(acode)) & (12) \\ \texttt{teaching}(course, acode) & \rightsquigarrow & \text{Person}(\textbf{iri2}(acode)) & (13) \end{array}$$

Properties of saturated mappings

H-complete RDF graph

An RDF graph $\mathcal{G}$ is **H-complete** w.r.t. an ontology $\mathcal{O}$, if, for every RDF triple (s, p, o), we have:

$$\langle \mathcal{O}, \mathcal{G} \rangle \models (s, p, o) \qquad \text{iff} \qquad (s, p, o) \in \mathcal{G}$$

The **saturation** $\mathcal{G}_{\text{sat}}$ of $\mathcal{G}$ w.r.t. $\mathcal{O}$ is the smallest RDF graph that contains $\mathcal{G}$ and is H-complete w.r.t. $\mathcal{O}$.

Intuitively, $\mathcal{G}_{\text{sat}}$ is obtained from $\mathcal{G}$ by applying the class and property inclusions of $\mathcal{O}$, but without introducing new nodes.

Relationship between the saturated mapping $\mathcal{M}_{\text{sat}}$ and the saturation of $\mathcal{M}(\mathcal{D})$

- We have that $\mathcal{M}_{\text{sat}}(\mathcal{D}) = (\mathcal{M}(\mathcal{D}))_{\text{sat}}$ (hence, it is an H-complete RDF graph).
- $\mathcal{M}_{\text{sat}}$ does not depend on the SPARQL query q, hence it can be pre-computed.
- It can be optimized (by exploiting query containment).

Properties of saturated mappings

H-complete RDF graph

An RDF graph $\mathcal{G}$ is **H-complete** w.r.t. an ontology $\mathcal{O}$, if, for every RDF triple (s, p, o), we have:

$$\langle \mathcal{O}, \mathcal{G} \rangle \models (s, p, o) \qquad \text{iff} \qquad (s, p, o) \in \mathcal{G}$$

The **saturation** $\mathcal{G}_{\text{sat}}$ of $\mathcal{G}$ w.r.t. $\mathcal{O}$ is the smallest RDF graph that contains $\mathcal{G}$ and is H-complete w.r.t. $\mathcal{O}$.

Intuitively, $\mathcal{G}_{\text{sat}}$ is obtained from $\mathcal{G}$ by applying the class and property inclusions of $\mathcal{O}$, but without introducing new nodes.

Relationship between the saturated mapping $\mathcal{M}_{\text{sat}}$ and the saturation of $\mathcal{M}(\mathcal{D})$

- We have that $\mathcal{M}_{\text{sat}}(\mathcal{D}) = (\mathcal{M}(\mathcal{D}))_{\text{sat}}$ (hence, it is an H-complete RDF graph).
- $\mathcal{M}_{\text{sat}}$ does not depend on the SPARQL query q, hence it can be pre-computed.
- It can be optimized (by exploiting query containment).

Properties of saturated mappings

H-complete RDF graph

An RDF graph $\mathcal{G}$ is **H-complete** w.r.t. an ontology $\mathcal{O}$, if, for every RDF triple (s, p, o), we have:

$$\langle \mathcal{O}, \mathcal{G} \rangle \models (s, p, o) \qquad \text{iff} \qquad (s, p, o) \in \mathcal{G}$$

The **saturation** $\mathcal{G}_{\text{sat}}$ of $\mathcal{G}$ w.r.t. $\mathcal{O}$ is the smallest RDF graph that contains $\mathcal{G}$ and is H-complete w.r.t. $\mathcal{O}$.

Intuitively, $\mathcal{G}_{\text{sat}}$ is obtained from $\mathcal{G}$ by applying the class and property inclusions of $\mathcal{O}$, but without introducing new nodes.

Relationship between the saturated mapping $\mathcal{M}_{\text{sat}}$ and the saturation of $\mathcal{M}(\mathcal{D})$

- We have that $\mathcal{M}_{\text{sat}}(\mathcal{D}) = (\mathcal{M}(\mathcal{D}))_{\text{sat}}$ (hence, it is an H-complete RDF graph).
- $\mathcal{M}_{\text{sat}}$ does not depend on the SPARQL query q, hence it can be pre-computed.
- It can be optimized (by exploiting query containment).

Mapping optimization – Exercise

Saturated mapping assertions $\mathcal{M}_{\text{sat}}$

$$
\begin{array}{llr}
\ldots & & \\
\texttt{academic}(acode, fn, ln, pos) & \rightsquigarrow \text{Faculty}(\textbf{iri2}(acode)) & (5) \\
\texttt{student}(scode, fn, ln) & \rightsquigarrow \text{Person}(\textbf{iri1}(scode)) & (6) \\
\texttt{academic}(acode, fn, ln, pos),\ pos = 9 & \rightsquigarrow \text{Faculty}(\textbf{iri2}(acode)) & (7) \\
\texttt{academic}(acode, fn, ln, pos),\ pos = 9 & \rightsquigarrow \text{Person}(\textbf{iri2}(acode)) & (8) \\
\texttt{academic}(acode, fn, ln, pos),\ pos = 2 & \rightsquigarrow \text{Faculty}(\textbf{iri2}(acode)) & (9) \\
\texttt{academic}(acode, fn, ln, pos),\ pos = 2 & \rightsquigarrow \text{Person}(\textbf{iri2}(acode)) & (10) \\
\texttt{academic}(acode, fn, ln, pos) & \rightsquigarrow \text{Person}(\textbf{iri2}(acode)) & (11) \\
\texttt{teaching}(course, acode) & \rightsquigarrow \text{Faculty}(\textbf{iri2}(acode)) & (12) \\
\texttt{teaching}(course, acode) & \rightsquigarrow \text{Person}(\textbf{iri2}(acode)) & (13)
\end{array}
$$

Mapping optimization – Exercise

Saturated mapping assertions $\mathcal{M}_{\text{sat}}$

$$\begin{array}{llr}
\ldots & & \\
\texttt{academic}(acode, fn, ln, pos) & \rightsquigarrow \text{Faculty}(\textbf{iri2}(acode)) & (5) \\
\texttt{student}(scode, fn, ln) & \rightsquigarrow \text{Person}(\textbf{iri1}(scode)) & (6) \\
\texttt{academic}(acode, fn, ln, pos),\ pos = 9 & \rightsquigarrow \text{Faculty}(\textbf{iri2}(acode)) & (7) \\
\texttt{academic}(acode, fn, ln, pos),\ pos = 9 & \rightsquigarrow \text{Person}(\textbf{iri2}(acode)) & (8) \\
\texttt{academic}(acode, fn, ln, pos),\ pos = 2 & \rightsquigarrow \text{Faculty}(\textbf{iri2}(acode)) & (9) \\
\texttt{academic}(acode, fn, ln, pos),\ pos = 2 & \rightsquigarrow \text{Person}(\textbf{iri2}(acode)) & (10) \\
\texttt{academic}(acode, fn, ln, pos) & \rightsquigarrow \text{Person}(\textbf{iri2}(acode)) & (11) \\
\texttt{teaching}(course, acode) & \rightsquigarrow \text{Faculty}(\textbf{iri2}(acode)) & (12) \\
\texttt{teaching}(course, acode) & \rightsquigarrow \text{Person}(\textbf{iri2}(acode)) & (13)
\end{array}$$

Consider also a foreign key over the database relations

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

Mapping optimization – Exercise

Saturated mapping assertions $\mathcal{M}_{\text{sat}}$

$$\begin{array}{llr}
\dots & & \\
\texttt{academic}(acode, fn, ln, pos) & \rightsquigarrow \mathsf{Faculty}(\mathbf{iri2}(acode)) & (5) \\
\texttt{student}(scode, fn, ln) & \rightsquigarrow \mathsf{Person}(\mathbf{iri1}(scode)) & (6) \\
\texttt{academic}(acode, fn, ln, pos),\ pos = 9 & \rightsquigarrow \mathsf{Faculty}(\mathbf{iri2}(acode)) & (7) \\
\texttt{academic}(acode, fn, ln, pos),\ pos = 9 & \rightsquigarrow \mathsf{Person}(\mathbf{iri2}(acode)) & (8) \\
\texttt{academic}(acode, fn, ln, pos),\ pos = 2 & \rightsquigarrow \mathsf{Faculty}(\mathbf{iri2}(acode)) & (9) \\
\texttt{academic}(acode, fn, ln, pos),\ pos = 2 & \rightsquigarrow \mathsf{Person}(\mathbf{iri2}(acode)) & (10) \\
\texttt{academic}(acode, fn, ln, pos) & \rightsquigarrow \mathsf{Person}(\mathbf{iri2}(acode)) & (11) \\
\texttt{teaching}(course, acode) & \rightsquigarrow \mathsf{Faculty}(\mathbf{iri2}(acode)) & (12) \\
\texttt{teaching}(course, acode) & \rightsquigarrow \mathsf{Person}(\mathbf{iri2}(acode)) & (13)
\end{array}$$

Consider also a foreign key over the database relations

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

We can **optimize the mapping** using query containment and the FK.

Mapping optimization – Exercise

Saturated mapping assertions $\mathcal{M}_{\text{sat}}$

$$
\begin{array}{llr}
\ldots & & \\
\texttt{academic}(acode, fn, ln, pos) & \rightsquigarrow \text{Faculty}(\textbf{iri2}(acode)) & (5) \\
\texttt{student}(scode, fn, ln) & \rightsquigarrow \text{Person}(\textbf{iri1}(scode)) & (6) \\
\texttt{academic}(acode, fn, ln, pos),\ pos = 9 & \rightsquigarrow \text{Faculty}(\textbf{iri2}(acode)) & (7) \\
\texttt{academic}(acode, fn, ln, pos),\ pos = 9 & \rightsquigarrow \text{Person}(\textbf{iri2}(acode)) & (8) \\
\texttt{academic}(acode, fn, ln, pos),\ pos = 2 & \rightsquigarrow \text{Faculty}(\textbf{iri2}(acode)) & (9) \\
\texttt{academic}(acode, fn, ln, pos),\ pos = 2 & \rightsquigarrow \text{Person}(\textbf{iri2}(acode)) & (10) \\
\texttt{academic}(acode, fn, ln, pos) & \rightsquigarrow \text{Person}(\textbf{iri2}(acode)) & (11) \\
\texttt{teaching}(course, acode) & \rightsquigarrow \text{Faculty}(\textbf{iri2}(acode)) & (12) \\
\texttt{teaching}(course, acode) & \rightsquigarrow \text{Person}(\textbf{iri2}(acode)) & (13)
\end{array}
$$

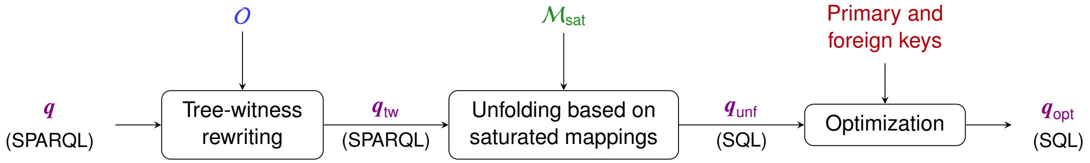
Consider also a foreign key over the database relations

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

We can **optimize the mapping** using query containment and the FK. This removes mapping assertions 7, 8, 9, 10, 12, and 13.

$$
\begin{array}{llr}
\ldots & & \\
\texttt{academic}(acode, fn, ln, pos) & \rightsquigarrow \text{Faculty}(\textbf{iri2}(acode)) & (5) \\
\texttt{student}(scode, fn, ln) & \rightsquigarrow \text{Person}(\textbf{iri1}(scode)) & (6) \\
\texttt{academic}(acode, fn, ln, pos) & \rightsquigarrow \text{Person}(\textbf{iri2}(acode)) & (11)
\end{array}
$$

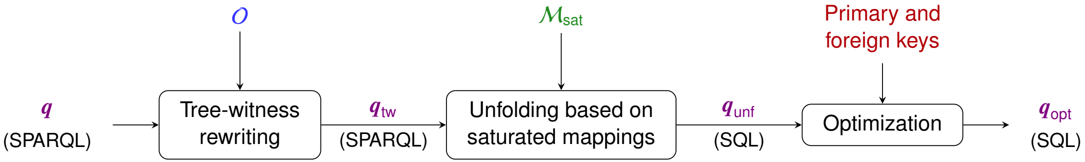
Query reformulation as implemented by the Ontop system



	Step	Input	Output
1.	Tree-witness rewriting	q and $\mathcal{O}$	q_{tw}
2.	Unfolding	q_{tw} and $\mathcal{M}_{\text{sat}}$	q_{unf}
3.	Optimization	q_{unf}, primary and foreign keys	q_{opt}

Let us now consider the optimization step.

Query reformulation as implemented by the Ontop system



	Step	Input	Output
1.	Tree-witness rewriting	q and $\mathcal{O}$	q_{tw}
2.	Unfolding	q_{tw} and $\mathcal{M}_{\text{sat}}$	q_{unf}
3.	Optimization	q_{unf}, primary and foreign keys	q_{opt}

Let us now consider the optimization step.

Outline

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 performance of joins

Semantic optimization

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

Integrity constraints

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

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 performance of joins

Semantic optimization

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

Integrity constraints

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

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 performance of joins

Semantic optimization

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

Integrity constraints

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

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 performance of joins

Semantic optimization

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

Integrity constraints

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

Reformulation example – 1. Unfolding

Saturated mapping

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

Query (we assume that the ontology is empty, hence $q_{\mathrm{r}} = q$)

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

We apply **query unfolding**, and then **normalization** to make the join conditions explicit.

$$
\begin{array}{rcl}
(x, y, z) & \leftarrow & \boldsymbol{q}1_{\text{unf}}(x), \\
 & & \\
q1_{\text{unf}}(\textbf{iri2}(acode)) & \leftarrow & \texttt{academic}(acode, fn, ln, pos), \\
 & & pos \in [1..8] \\
q1_{\text{unf}}(\textbf{iri2}(acode)) & \leftarrow & \texttt{teaching}(course, acode) \\
q2_{\text{unf}}(\textbf{iri1}(scode), fn) & \leftarrow & \texttt{student}(scode, fn, ln) \\
q2_{\text{unf}}(\textbf{iri2}(acode), fn) & \leftarrow & \texttt{academic}(acode, fn, ln, pos) \\
q3_{\text{unf}}(\textbf{iri1}(scode), ln) & \leftarrow & \texttt{student}(scode, fn, ln) \\
q3_{\text{unf}}(\textbf{iri2}(acode), ln) & \leftarrow & \texttt{academic}(acode, fn, ln, pos)
\end{array}
$$

Reformulation example – 1. Unfolding

Saturated mapping

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

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

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

We apply **query unfolding**, and then **normalization** to make the join conditions explicit.

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

Reformulation example – 1. Unfolding

Saturated mapping

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

$\texttt{teaching}(course, acode) \rightsquigarrow$ Teacher(**iri2**$(acode)$)

$\texttt{student}(scode, fn, ln) \rightsquigarrow$ firstName(**iri1**$(scode), fn)$

$\texttt{academic}(acode, fn, ln, pos) \rightsquigarrow$ firstName(**iri2**$(acode), fn)$

$\texttt{student}(scode, fn, ln) \rightsquigarrow$ lastName(**iri1**$(scode), ln)$

$\texttt{academic}(acode, fn, ln, pos) \rightsquigarrow$ lastName(**iri2**$(acode), ln)$

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

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

We apply **query unfolding**, and then **normalization** to make the join conditions explicit.

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

Reformulation example – 2. Structural optimization

Unfolded normalized query

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

- 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.

Reformulation example – 2. Structural optimization

Unfolded normalized query

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

- 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 1/2

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

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

(One sub-query not shown)

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

Reformulation example – 2. Structural optimization

Unfolded normalized query

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

- 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

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

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

(One sub-query not shown)

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

Reformulation example – 3. Semantic optimization

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

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

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

Self-join elimination (semantic optimization)

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

$$
\begin{aligned}
q_{\text{opt}}(\textbf{iri2}(a), y, z) &\leftarrow \texttt{academic}(a, y, z, p_1),\ p_1 \in [1..8]\\
q_{\text{opt}}(\textbf{iri2}(a), y, z) &\leftarrow \texttt{teaching}(c, a),\ \texttt{academic}(a, y, z, p_2)
\end{aligned}
$$

Reformulation example – 3. Semantic optimization

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

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

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

Self-join elimination (semantic optimization)

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

$$\begin{aligned} q_{\text{opt}}(\textbf{iri2}(a), y, z) &\leftarrow \texttt{academic}(a, y, z, p_1),\ p_1 \in [1..8] \\ q_{\text{opt}}(\textbf{iri2}(a), y, z) &\leftarrow \texttt{teaching}(c, a),\ \texttt{academic}(a, y, z, p_2) \end{aligned}$$

Reformulation example – 3. Semantic optimization

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

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

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

Self-join elimination (semantic optimization)

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

$$
\begin{aligned}
q_{\text{opt}}(\textbf{iri2}(a), y, z) &\leftarrow \texttt{academic}(a, y, z, p_1),\ p_1 \in [1..8]\\
q_{\text{opt}}(\textbf{iri2}(a), y, z) &\leftarrow \texttt{teaching}(c, a),\ \texttt{academic}(a, y, z, p_2)
\end{aligned}
$$

Reformulation example – 3. Semantic optimization

We are left with just two queries, which 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: $\texttt{academic}(acode, f, l, p) \land \texttt{academic}(acode, f', l', p') \rightarrow (f = f') \land (l = l') \land (p = p')$

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

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

Computational complexity of query answering

From the top-down approach to query answering, and the complexity results for *DL-Lite*, we obtain the following result.

Theorem

For OWL 2 QL (or *DL-Lite*) VKG instances $\langle \mathcal{P}, \mathcal{D} \rangle$, with $\mathcal{P} = \langle \mathcal{O}, \mathcal{M}, \mathcal{S} \rangle$, **query answering** for UCQs / SPARQL queries is:

1. Very efficiently tractable, i.e., in AC^0, in the size of the database $\mathcal{D}$.
2. Efficiently tractable, , i.e., in PTime, in the size of the ontology $\mathcal{O}$ and the mapping $\mathcal{M}$.
3. Exponential, more precisely NP-complete, in the size of the query.

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

Note: The AC^0 result is a consequence of the fact that query answering in such a setting can be reduced to evaluating a SQL query over the relational database $\mathcal{D}$.

Can we go beyond *DL-Lite* and maintain the same complexity results?

Essentially no! By adding essentially any additional constructs of OWL, we lose first-order rewritability and hence these nice computational properties.

Computational complexity of query answering

From the top-down approach to query answering, and the complexity results for *DL-Lite*, we obtain the following result.

Theorem

For OWL 2 QL (or *DL-Lite*) VKG instances $\langle \mathcal{P}, \mathcal{D} \rangle$, with $\mathcal{P} = \langle \mathcal{O}, \mathcal{M}, \mathcal{S} \rangle$, **query answering** for UCQs / SPARQL queries is:

1. Very efficiently tractable, i.e., in AC^0, in the size of the database $\mathcal{D}$.
2. Efficiently tractable, , i.e., in PTime, in the size of the ontology $\mathcal{O}$ and the mapping $\mathcal{M}$.
3. Exponential, more precisely NP-complete, in the size of the query.

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

Note: The AC^0 result is a consequence of the fact that query answering in such a setting can be reduced to evaluating a SQL query over the relational database $\mathcal{D}$.

Can we go beyond *DL-Lite* and maintain the same complexity results?

Essentially no! By adding essentially any additional constructs of OWL, we lose first-order rewritability and hence these nice computational properties.

References I

[1] Stanislav Kikot, Roman Kontchakov & Michael Zakharyaschev. “Conjunctive Query Answering with OWL 2 QL”. In: *Proc. of the 13th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR)*. 2012, pp. 275–285.

[2] Mariano Rodriguez-Muro, Roman Kontchakov & Michael Zakharyaschev. “Ontology-Based Data Access: Ontop of Databases”. In: *Proc. of the 12th Int. Semantic Web Conf. (ISWC)*. Vol. 8218. Lecture Notes in Computer Science. Springer, 2013, pp. 558–573. DOI: 10.1007/978-3-642-41335-3_35.

[3] Roman Kontchakov, Martin Rezk, Mariano Rodriguez-Muro, Guohui Xiao & Michael Zakharyaschev. “Answering SPARQL Queries over Databases under OWL 2 QL Entailment Regime”. In: *Proc. of the 13th Int. Semantic Web Conf. (ISWC)*. Vol. 8796. Lecture Notes in Computer Science. Springer, 2014, pp. 552–567. DOI: 10.1007/978-3-319-11964-9_35.

References II

[4] Roman Kontchakov & Michael Zakharyaschev. “An Introduction to Description Logics and Query Rewriting”. In: *Reasoning Web: Reasoning on the Web in the Big Data Era – 10th Int. Summer School Tutorial Lectures (RW)*. Vol. 8714. Lecture Notes in Computer Science. Springer, 2014, pp. 195–244. DOI: 10.1007/978-3-319-10587-1_5.

[5] Diego C., Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini & Riccardo Rosati. “Data Complexity of Query Answering in Description Logics”. In: *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR)*. 2006, pp. 260–270.

[6] Diego C., Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini & Riccardo Rosati. “Data Complexity of Query Answering in Description Logics”. In: *Artificial Intelligence* 195 (2013), pp. 335–360. DOI: 10.1016/j.artint.2012.10.003.