Query answering in description logics: *DL-Lite*$_\mathcal{A}$

Giuseppe De Giacomo

Dipartimento di Informatica e Sistemistica
Sapienza Università di Roma

Outline

Outline

Ontologies and data

- The best current DL reasoning systems can deal with moderately large ABoxes. $\rightsquigarrow 10^4$ individuals *(and this is a big achievement of the last years)!*
- But data of interests in typical information systems are much **larger** $\rightsquigarrow 10^6 - 10^9$ individuals
- The best technology to deal with large amounts of data are **relational databases**.

Question:
How can we use ontologies together with large amounts of data?

Ontologies and data

- The best current DL reasoning systems can deal with moderately large ABoxes. $\leadsto 10^4$ individuals *(and this is a big achievement of the last years)!*
- But data of interests in typical information systems are much **larger** $\leadsto 10^6 - 10^9$ individuals
- The best technology to deal with large amounts of data are **relational databases**.

Question:
How can we use ontologies together with large amounts of data?

Ontologies and data

- The best current DL reasoning systems can deal with moderately large ABoxes. $\leadsto 10^4$ individuals *(and this is a big achievement of the last years)!*
- But data of interests in typical information systems are much **larger** $\leadsto 10^6 - 10^9$ individuals
- The best technology to deal with large amounts of data are **relational databases**.

Question:
How can we use ontologies together with large amounts of data?

Challenges when integrating data into ontologies

Deal with well-known tradeoff between expressive power of the ontology language and complexity of dealing with (i.e., performing inference over) ontologies in that language.

Requirements come from the specific setting:

- We have to fully take into account the ontology.
  $\rightsquigarrow$ **inference**
- We have to deal very large amounts of data.
  $\rightsquigarrow$ **relational databases**
- We want flexibility in querying the data.
  $\rightsquigarrow$ **expressive query language**
- We want to keep the data in the sources, and not move it around.
  $\rightsquigarrow$ **map** data sourses to the ontology (cf. Data Integration)

Challenges when integrating data into ontologies

Deal with well-known tradeoff between expressive power of the ontology language and complexity of dealing with (i.e., performing inference over) ontologies in that language.

Requirements come from the specific setting:

- We have to fully take into account the ontology.
  $\rightsquigarrow$ **inference**
- We have to deal very large amounts of data.
  $\rightsquigarrow$ **relational databases**
- We want flexibility in querying the data.
  $\rightsquigarrow$ **expressive query language**
- We want to keep the data in the sources, and not move it around.
  $\rightsquigarrow$ **map** data sourses to the ontology (cf. Data Integration)

Challenges when integrating data into ontologies

Deal with well-known tradeoff between expressive power of the ontology language and complexity of dealing with (i.e., performing inference over) ontologies in that language.

Requirements come from the specific setting:

- We have to fully take into account the ontology.
  $\leadsto$ **inference**
- We have to deal very large amounts of data.
  $\leadsto$ **relational databases**
- We want flexibility in querying the data.
  $\leadsto$ **expressive query language**
- We want to keep the data in the sources, and not move it around.
  $\leadsto$ **map** data sourses to the ontology (cf. Data Integration)

Challenges when integrating data into ontologies

Deal with well-known tradeoff between expressive power of the ontology language and complexity of dealing with (i.e., performing inference over) ontologies in that language.

Requirements come from the specific setting:

- We have to fully take into account the ontology.
  $\leadsto$ **inference**
- We have to deal very large amounts of data.
  $\leadsto$ **relational databases**
- We want flexibility in querying the data.
  $\leadsto$ **expressive query language**
- We want to keep the data in the sources, and not move it around.
  $\leadsto$ **map** data sourses to the ontology (cf. Data Integration)

Questions addressed in this part of the tutorial

1. Which is the “right” **query language**?
2. Which is the “right” **ontology language**?
3. How can we bridge the **semantic mismatch** between the ontology and the data sources?
4. How can **tools for ontology-based data access and integration** fully take into account all these issues?

Outline

Ontology languages vs. query languages

Which query language to use?

Two extreme cases:

1. Just classes and properties of the ontology $\rightsquigarrow$ instance checking
   - Ontology languages are tailored for capturing intensional relationships.
   - They are quite **poor as query languages**:
     Cannot refer to same object via multiple navigation paths in the ontology, i.e., allow only for a limited form of JOIN, namely chaining.
2. Full SQL (or equivalently, first-order logic)
   - Problem: in the presence of incomplete information, query answering becomes **undecidable** (FOL validity).

Ontology languages vs. query languages

Which query language to use?

Two extreme cases:

1. Just classes and properties of the ontology $\rightsquigarrow$ instance checking
   - Ontology languages are tailored for capturing intensional relationships.
   - They are quite **poor as query languages**:
     Cannot refer to same object via multiple navigation paths in the ontology, i.e., allow only for a limited form of JOIN, namely chaining.
2. Full SQL (or equivalently, first-order logic)
   - Problem: in the presence of incomplete information, query answering becomes **undecidable** (FOL validity).

Ontology languages vs. query languages

Which query language to use?

Two extreme cases:

1. Just classes and properties of the ontology $\rightsquigarrow$ instance checking
   - Ontology languages are tailored for capturing intensional relationships.
   - They are quite **poor as query languages**:
     Cannot refer to same object via multiple navigation paths in the ontology, i.e., allow only for a limited form of JOIN, namely chaining.
2. Full SQL (or equivalently, first-order logic)
   - Problem: in the presence of incomplete information, query answering becomes **undecidable** (FOL validity).

Conjunctive queries (CQs)

A **conjunctive query (CQ)** is a first-order query of the form

$$q(\vec{x}) \leftarrow \exists\vec{y}.R_1(\vec{x},\vec{y}) \wedge \cdots \wedge R_k(\vec{x},\vec{y})$$

where each $R_i(\vec{x},\vec{y})$ is an atom using (some of) the free variables $\vec{x}$, the existentially quantified variables $\vec{y}$, and possibly constants.

We will also use the simpler Datalog notation:

$$q(\vec{x}) \leftarrow R_1(\vec{x},\vec{y}),\ldots,R_k(\vec{x},\vec{y})$$

Note:

- CQs contain no disjunction, no negation, no universal quantification.
- Correspond to SQL/relational algebra **select-project-join (SPJ) queries** – the most frequently asked queries.
- They can also be written as **SPARQL** queries.
- A Boolean CQ is a CQ without free variables $\Rightarrow$ $q() \leftarrow \exists\vec{y}.R_1(\vec{y}) \wedge \cdots \wedge R_k(\vec{y})$.

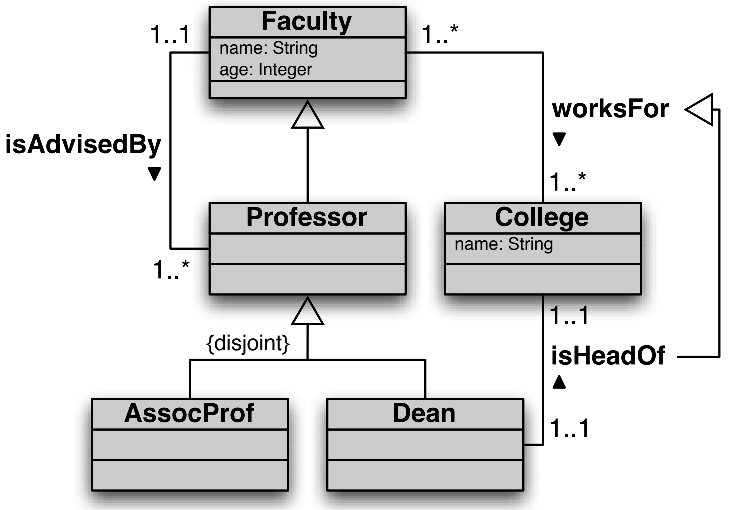
Example of conjunctive query

$$\begin{array}{rcl}
\text{Professor} & \sqsubseteq & \text{Faculty} \\
\text{AssocProf} & \sqsubseteq & \text{Professor} \\
\text{Dean} & \sqsubseteq & \text{Professor} \\
\text{AssocProf} & \sqsubseteq & \neg\text{Dean} \\
\text{Faculty} & \sqsubseteq & \exists\text{age} \\
\exists\text{age}^- & \sqsubseteq & \text{Integer} \\
\exists\text{worksFor} & \sqsubseteq & \text{Faculty} \\
\exists\text{worksFor}^- & \sqsubseteq & \text{College} \\
\text{Faculty} & \sqsubseteq & \exists\text{worksFor} \\
\text{College} & \sqsubseteq & \exists\text{worksFor}^- \\
 & \vdots &
\end{array}$$

$$\begin{array}{l}
q(nf, af, nd) \leftarrow \exists f, c, d, ad. \\
\quad \text{worksFor}(f, c) \land \text{isHeadOf}(d, c) \land \text{name}(f, nf) \land \text{name}(d, nd) \land \\
\quad \text{age}(f, af) \land \text{age}(d, ad) \land af = ad
\end{array}$$

Example of conjunctive query

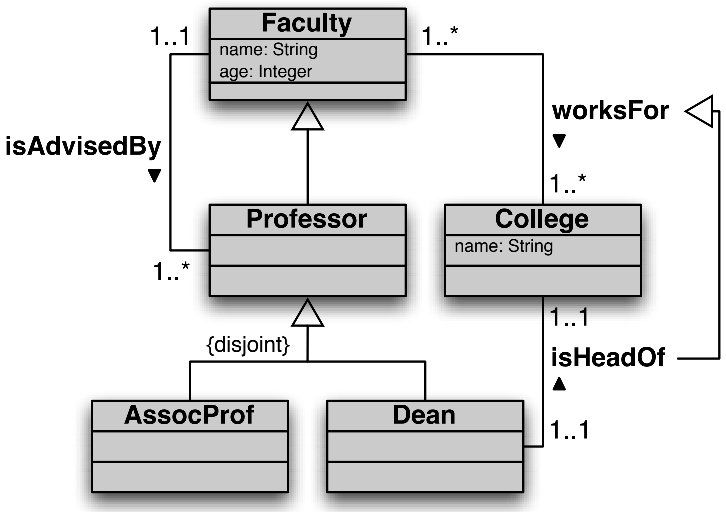
$$\begin{array}{rcl}
\text{Professor} & \sqsubseteq & \text{Faculty} \\
\text{AssocProf} & \sqsubseteq & \text{Professor} \\
\text{Dean} & \sqsubseteq & \text{Professor} \\
\text{AssocProf} & \sqsubseteq & \neg\text{Dean} \\
\text{Faculty} & \sqsubseteq & \exists\text{age} \\
\exists\text{age}^- & \sqsubseteq & \text{Integer} \\
\exists\text{worksFor} & \sqsubseteq & \text{Faculty} \\
\exists\text{worksFor}^- & \sqsubseteq & \text{College} \\
\text{Faculty} & \sqsubseteq & \exists\text{worksFor} \\
\text{College} & \sqsubseteq & \exists\text{worksFor}^- \\
 & \vdots &
\end{array}$$



$$\begin{array}{l}
q(nf, af, nd) \leftarrow \exists f, c, d, ad. \\
\quad \text{worksFor}(f, c) \land \text{isHeadOf}(d, c) \land \text{name}(f, nf) \land \text{name}(d, nd) \land \\
\quad \text{age}(f, af) \land \text{age}(d, ad) \land af = ad
\end{array}$$

Example of conjunctive query

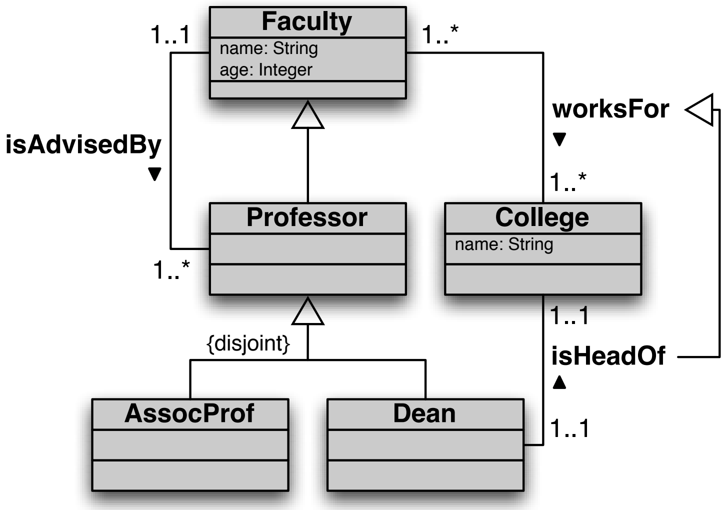
$$
\begin{array}{rcl}
\text{Professor} & \sqsubseteq & \text{Faculty} \\
\text{AssocProf} & \sqsubseteq & \text{Professor} \\
\text{Dean} & \sqsubseteq & \text{Professor} \\
\text{AssocProf} & \sqsubseteq & \neg\text{Dean} \\
\text{Faculty} & \sqsubseteq & \exists\text{age} \\
\exists\text{age}^{-} & \sqsubseteq & \text{Integer} \\
\exists\text{worksFor} & \sqsubseteq & \text{Faculty} \\
\exists\text{worksFor}^{-} & \sqsubseteq & \text{College} \\
\text{Faculty} & \sqsubseteq & \exists\text{worksFor} \\
\text{College} & \sqsubseteq & \exists\text{worksFor}^{-} \\
 & \vdots &
\end{array}
$$



$$
\begin{aligned}
q(nf, af, nd) \;\leftarrow\; & \exists f, c, d, ad. \\
& \text{worksFor}(f, c) \wedge \text{isHeadOf}(d, c) \wedge \text{name}(f, nf) \wedge \text{name}(d, nd) \wedge \\
& \text{age}(f, af) \wedge \text{age}(d, ad) \wedge af = ad
\end{aligned}
$$

Example of conjunctive query

$$
\begin{array}{rcl}
\text{Professor} & \sqsubseteq & \text{Faculty} \\
\text{AssocProf} & \sqsubseteq & \text{Professor} \\
\text{Dean} & \sqsubseteq & \text{Professor} \\
\text{AssocProf} & \sqsubseteq & \neg\text{Dean} \\
\text{Faculty} & \sqsubseteq & \exists\text{age} \\
\exists\text{age}^{-} & \sqsubseteq & \text{Integer} \\
\exists\text{worksFor} & \sqsubseteq & \text{Faculty} \\
\exists\text{worksFor}^{-} & \sqsubseteq & \text{College} \\
\text{Faculty} & \sqsubseteq & \exists\text{worksFor} \\
\text{College} & \sqsubseteq & \exists\text{worksFor}^{-} \\
 & \vdots &
\end{array}
$$



$$
\begin{aligned}
q(nf, af, nd) \;\leftarrow\; & \exists f, c, d, ad. \\
& \text{worksFor}(f, c) \wedge \text{isHeadOf}(d, c) \wedge \text{name}(f, nf) \wedge \text{name}(d, nd) \wedge \\
& \text{age}(f, af) \wedge \text{age}(d, ad) \wedge af = ad
\end{aligned}
$$

Conjunctive queries and SQL – Example

Relational alphabet:
$\mathsf{worksFor(fac, coll)}$, $\mathsf{isHeadOf(dean, coll)}$, $\mathsf{name(p, n)}$, $\mathsf{age(p, a)}$

Query: return name, age, and name of dean of all faculty that have the same age as their dean.

Conjunctive queries and SQL – Example

Relational alphabet:
$\mathsf{worksFor(fac, coll)}$, $\mathsf{isHeadOf(dean, coll)}$, $\mathsf{name(p, n)}$, $\mathsf{age(p, a)}$

Query: return name, age, and name of dean of all faculty that have the same age as their dean.

Expressed in SQL:

```
SELECT NF.name, AF.age, ND.name
FROM worksFor W, isHeadOf H, name NF, name ND, age AF, age AD
WHERE W.fac = NF.p   AND  W.fac = AF.p  AND
      H.dean = ND.p  AND  H.dean = AD.p  AND
      W.coll = H.coll  AND  AF.a = AD.a
```

Conjunctive queries and SQL – Example

Relational alphabet:

worksFor(fac, coll), isHeadOf(dean, coll), name(p, n), age(p, a)

Query: return name, age, and name of dean of all faculty that have the same age as their dean.

Expressed in SQL:

```
SELECT NF.name, AF.age, ND.name
FROM worksFor W, isHeadOf H, name NF, name ND, age AF, age AD
WHERE W.fac = NF.p   AND  W.fac = AF.p  AND
      H.dean = ND.p  AND  H.dean = AD.p  AND
      W.coll = H.coll  AND  AF.a = AD.a
```

Expressed as a CQ:

$$\begin{aligned} q(nf, af, nd) \leftarrow\ & \mathsf{worksFor}(f1, c1),\ \mathsf{isHeadOf}(d1, c2), \\ & \mathsf{name}(f2, nf),\ \mathsf{name}(d2, nd),\ \mathsf{age}(f3, af),\ \mathsf{age}(d3, ad), \\ & f1 = f2,\ f1 = f3,\ d1 = d2,\ d1 = d3,\ c1 = c2,\ af = ad \end{aligned}$$

Query answering under different assumptions

There are fundamentally different assumptions when addressing query answering in different settings:

- **traditional database assumption**
- **knowledge representation assumption**

Note: for the moment we assume to deal with an ordinary ABox, which however may be very large and thus is stored in a database.

Query answering under different assumptions

There are fundamentally different assumptions when addressing query answering in different settings:

- **traditional database assumption**
- **knowledge representation assumption**

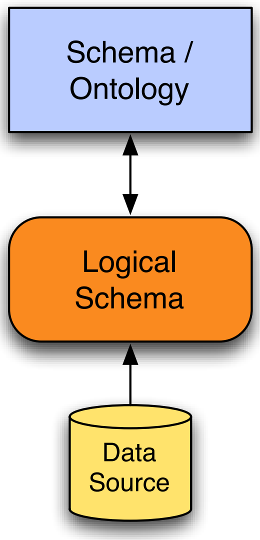
Note: for the moment we assume to deal with an ordinary ABox, which however may be very large and thus is stored in a database.

Query answering under the database assumption

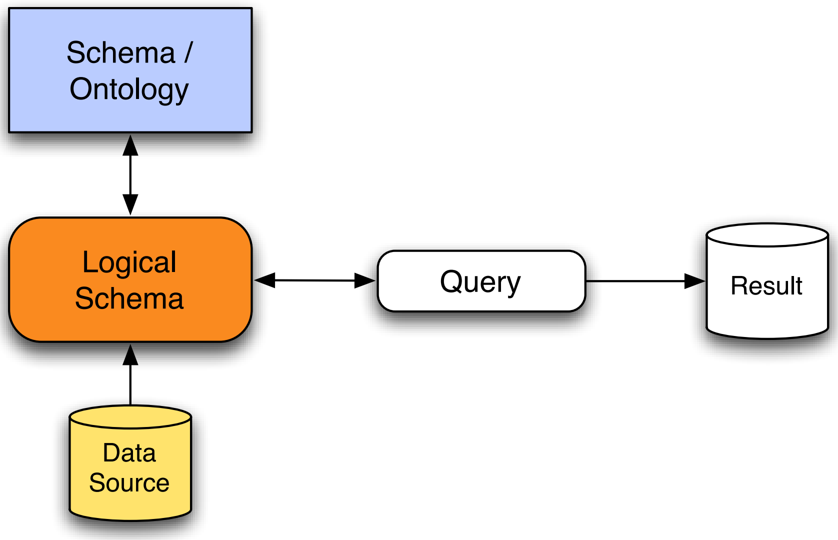
- Data are completely specified (CWA), and typically large.
- Schema/intensional information used in the design phase.
- At **runtime**, the data is assumed to satisfy the schema, and therefore the **schema is not used**.
- Queries allow for complex navigation paths in the data (cf. SQL).

$\leadsto$ Query answering amounts to **query evaluation**, which is computationally easy.

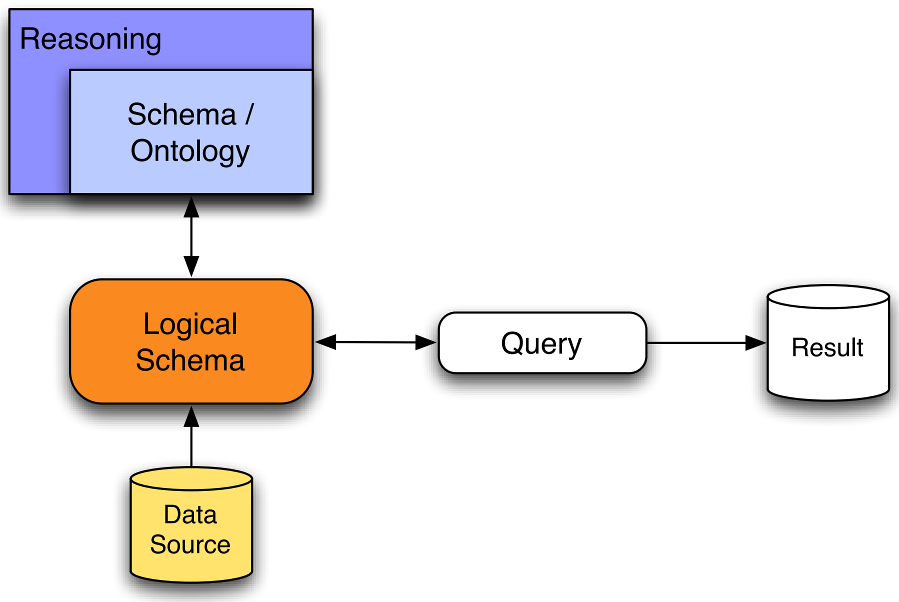
Query answering under the database assumption (cont'd)



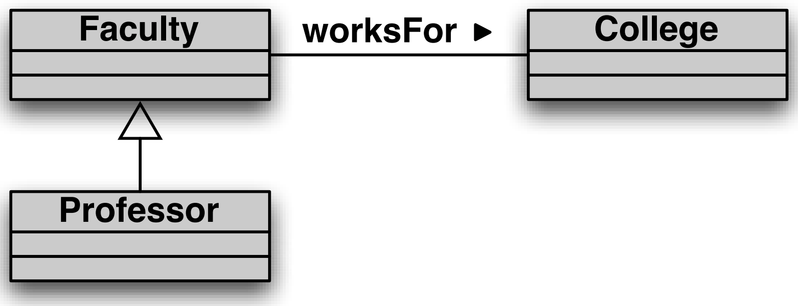
Query answering under the database assumption (cont'd)



Query answering under the database assumption (cont'd)



Query answering under the database assumption – Example



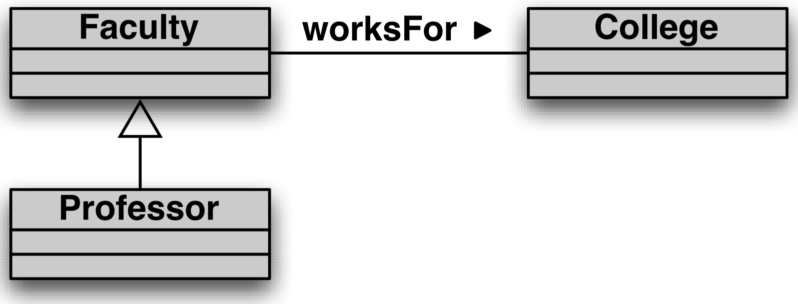
For each class/property we have a (complete) table in the database.

DB: Faculty = { `john`, `mary`, `paul` }
Professor = { `john`, `paul` }
College = { `collA`, `collB` }
worksFor = { (`john`,`collA`), (`mary`,`collB`) }

Query: $q(x) \leftarrow \exists c.\, \text{Professor}(x), \text{College}(c), \text{worksFor}(x, c)$

Answer: ???

Query answering under the database assumption – Example



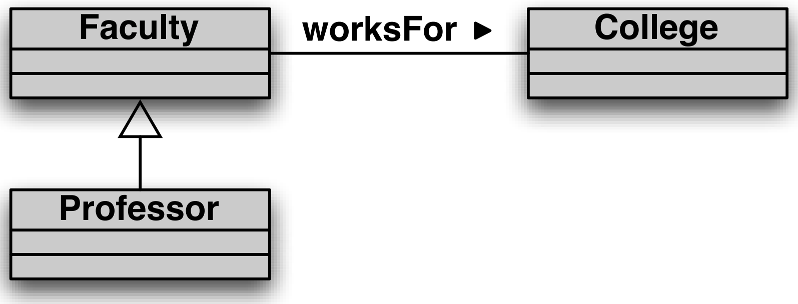
For each class/property we have a (complete) table in the database.

DB: Faculty = { john, mary, paul }
Professor = { john, paul }
College = { collA, collB }
worksFor = { (john,collA), (mary,collB) }

Query: $q(x) \leftarrow \exists c.\, \text{Professor}(x), \text{College}(c), \text{worksFor}(x, c)$

Answer: ???

Query answering under the database assumption – Example



For each class/property we have a (complete) table in the database.

DB: Faculty = { john, mary, paul }
Professor = { john, paul }
College = { collA, collB }
worksFor = { (john,collA), (mary,collB) }

Query: $q(x) \leftarrow \exists c.\, \text{Professor}(x), \text{College}(c), \text{worksFor}(x, c)$

Answer: { john }

Query answering under the KR assumption

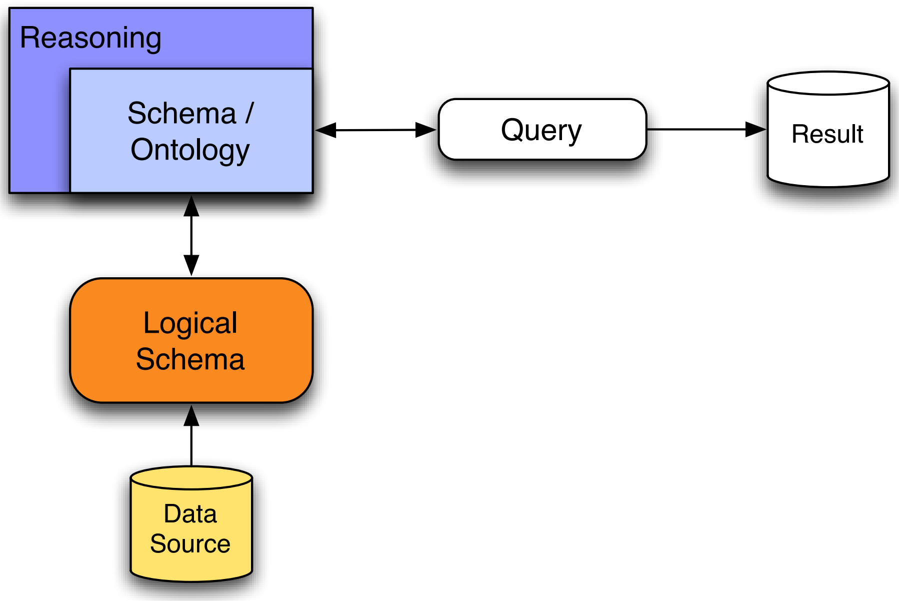
- An ontology imposes constraints on the data.
- Actual data may be incomplete or inconsistent w.r.t. such constraints.
- The system has to take into account the constraints during query answering, and overcome incompleteness or inconsistency.

$\rightsquigarrow$ Query answering amounts to **logical inference**, which is computationally more costly.

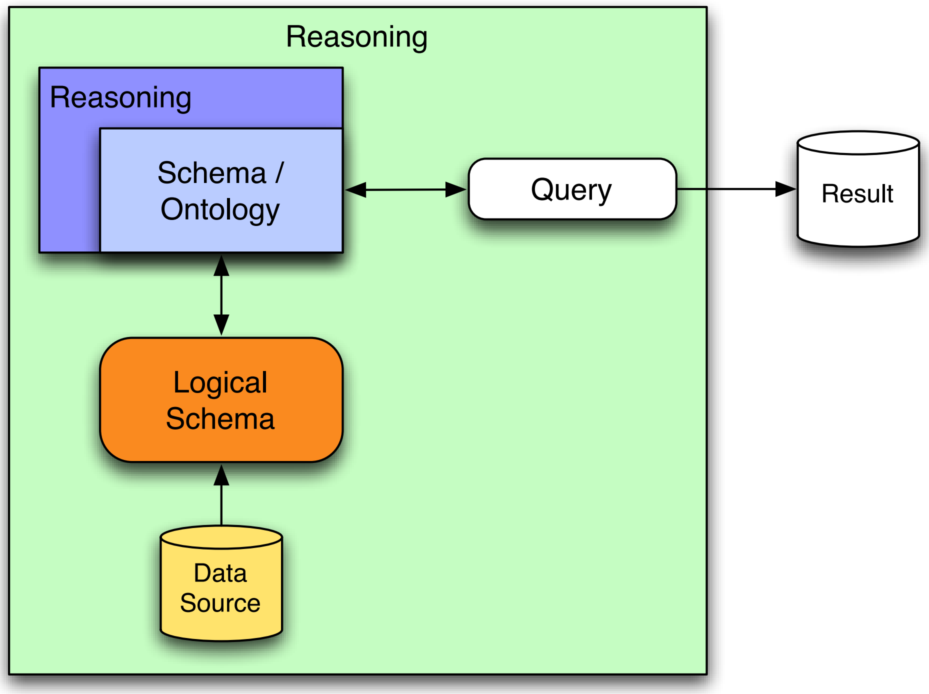
Note:

- Size of the data is not considered critical (comparable to the size of the intensional information).
- Queries are typically simple, i.e., atomic (a class name), and query answering amounts to instance checking.

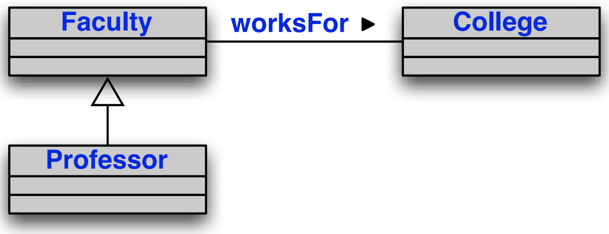
Query answering under the KR assumption (cont'd)



Query answering under the KR assumption (cont'd)



Query answering under the KR assumption – Example



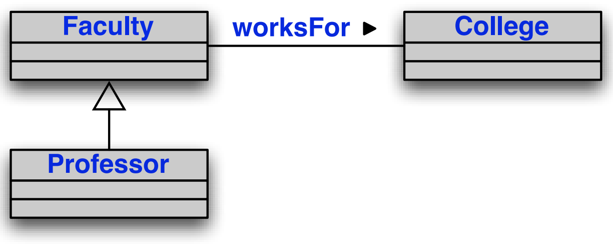
The tables in the database may be **incompletely specified**, or even missing for some classes/properties.

DB: Professor $\supseteq$ { `john`, `paul` }
College $\supseteq$ { `collA`, `collB` }
worksFor $\supseteq$ { (`john`,`collA`), (`mary`,`collB`) }

Query: $q(x) \leftarrow$ Faculty(x)

Answer: ???

Query answering under the KR assumption – Example



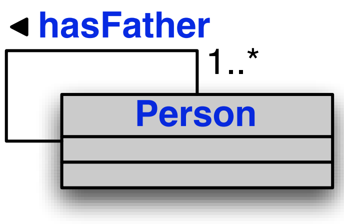
The tables in the database may be **incompletely specified**, or even missing for some classes/properties.

DB: Professor $\supseteq$ { `john`, `paul` }
College $\supseteq$ { `collA`, `collB` }
worksFor $\supseteq$ { (`john`,`collA`), (`mary`,`collB`) }

Query: $q(x) \leftarrow$ Faculty(x)

Answer: { `john`, `paul`, `mary` }

Query answering under the KR assumption – Example 2



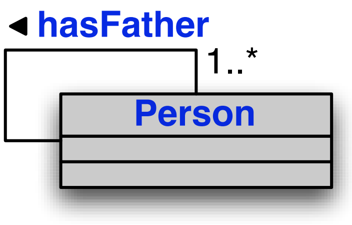
Each person has a father, who is a person.

DB: Person $\supseteq$ { `john`, `paul`, `toni` }
hasFather $\supseteq$ { (`john`,`paul`), (`paul`,`toni`) }

Queries: $q_1(x, y) \leftarrow$ hasFather(x, y)
$q_2(x) \leftarrow \exists y.$ hasFather(x, y)
$q_3(x) \leftarrow \exists y_1, y_2, y_3.$ hasFather(x, y_1), hasFather(y_1, y_2), hasFather(y_2, y_3)
$q_4(x, y_3) \leftarrow \exists y_1, y_2.$ hasFather(x, y_1), hasFather(y_1, y_2), hasFather(y_2, y_3)

Answers: to q_1: ???
to q_2: ???
to q_3: ???
to q_4: ???

Query answering under the KR assumption – Example 2



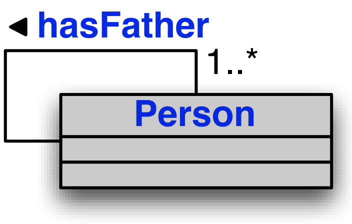
Each person has a father, who is a person.

DB: Person $\supseteq$ { john, paul, toni }
hasFather $\supseteq$ { (john,paul), (paul,toni) }

Queries: $q_1(x,y) \leftarrow$ hasFather(x,y)
$q_2(x) \leftarrow \exists y.$ hasFather(x,y)
$q_3(x) \leftarrow \exists y_1,y_2,y_3.$ hasFather(x,y_1), hasFather(y_1,y_2), hasFather(y_2,y_3)
$q_4(x,y_3) \leftarrow \exists y_1,y_2.$ hasFather(x,y_1), hasFather(y_1,y_2), hasFather(y_2,y_3)

Answers: to q_1: { (john,paul), (paul,toni) }
to q_2: ???
to q_3: ???
to q_4: ???

Query answering under the KR assumption – Example 2



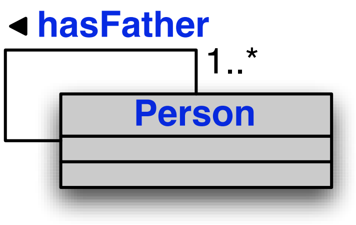
Each person has a father, who is a person.

DB: Person $\supseteq$ { john, paul, toni }
hasFather $\supseteq$ { (john,paul), (paul,toni) }

Queries: $q_1(x,y) \leftarrow$ hasFather(x,y)
$q_2(x) \leftarrow \exists y.$ hasFather(x,y)
$q_3(x) \leftarrow \exists y_1,y_2,y_3.$ hasFather(x,y_1), hasFather(y_1,y_2), hasFather(y_2,y_3)
$q_4(x,y_3) \leftarrow \exists y_1,y_2.$ hasFather(x,y_1), hasFather(y_1,y_2), hasFather(y_2,y_3)

Answers: to q_1: { (john,paul), (paul,toni) }
to q_2: { john, paul, toni }
to q_3: ???
to q_4: ???

Query answering under the KR assumption – Example 2



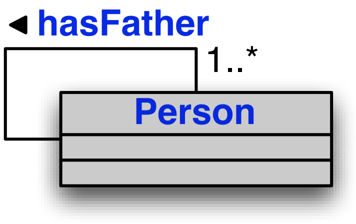
Each person has a father, who is a person.

DB: Person $\supseteq$ { `john`, `paul`, `toni` }
hasFather $\supseteq$ { (`john`,`paul`), (`paul`,`toni`) }

Queries: $q_1(x, y) \leftarrow$ hasFather(x, y)
$q_2(x) \leftarrow \exists y.$ hasFather(x, y)
$q_3(x) \leftarrow \exists y_1, y_2, y_3.$ hasFather(x, y_1), hasFather(y_1, y_2), hasFather(y_2, y_3)
$q_4(x, y_3) \leftarrow \exists y_1, y_2.$ hasFather(x, y_1), hasFather(y_1, y_2), hasFather(y_2, y_3)

Answers: to q_1: { (`john`,`paul`), (`paul`,`toni`) }
to q_2: { `john`, `paul`, `toni` }
to q_3: { `john`, `paul`, `toni` }
to q_4: ???

Query answering under the KR assumption – Example 2



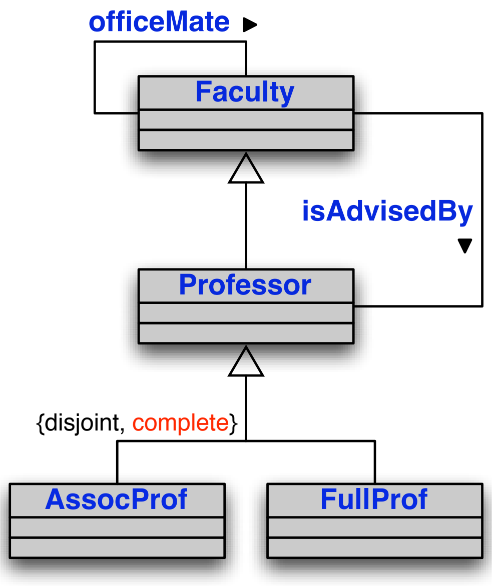
Each person has a father, who is a person.

DB: Person $\supseteq$ { `john`, `paul`, `toni` }
hasFather $\supseteq$ { (`john`,`paul`), (`paul`,`toni`) }

Queries: $q_1(x, y) \leftarrow$ hasFather(x, y)
$q_2(x) \leftarrow \exists y.$ hasFather(x, y)
$q_3(x) \leftarrow \exists y_1, y_2, y_3.$ hasFather(x, y_1), hasFather(y_1, y_2), hasFather(y_2, y_3)
$q_4(x, y_3) \leftarrow \exists y_1, y_2.$ hasFather(x, y_1), hasFather(y_1, y_2), hasFather(y_2, y_3)

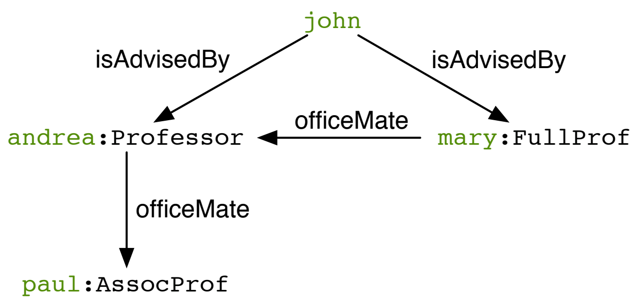
Answers: to q_1: { (`john`,`paul`), (`paul`,`toni`) }
to q_2: { `john`, `paul`, `toni` }
to q_3: { `john`, `paul`, `toni` }
to q_4: { }

QA under the KR assumption – Andrea's Example

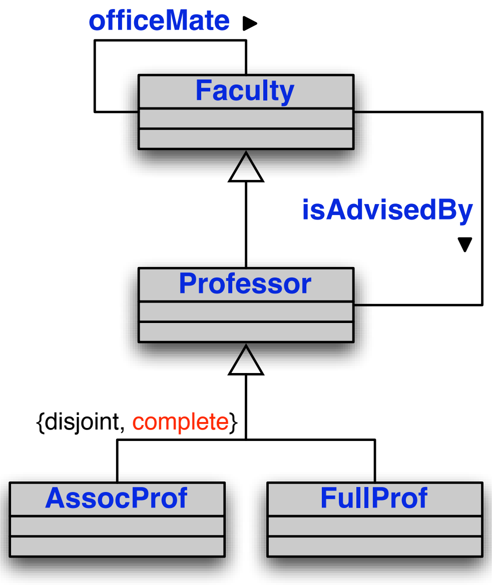


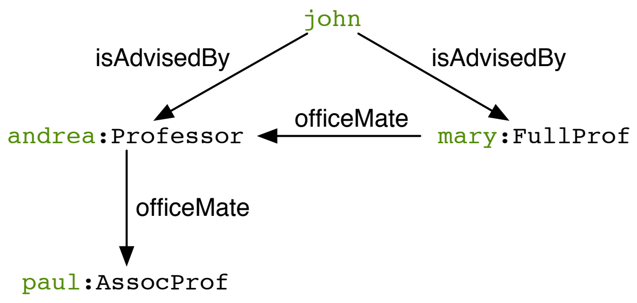
Professor $\equiv$ AssocProf $\sqcup$ FullProf

Faculty $\supseteq$ { andrea, paul, mary, john }
Professor $\supseteq$ { andrea, paul, mary }
AssocProf $\supseteq$ { paul }
FullProf $\supseteq$ { mary }
isAdvisedBy $\supseteq$ { (john,andrea), (john,mary) }
officeMate $\supseteq$ { (mary,andrea), (andrea,paul) }



QA under the KR assumption – Andrea's Example (cont'd)

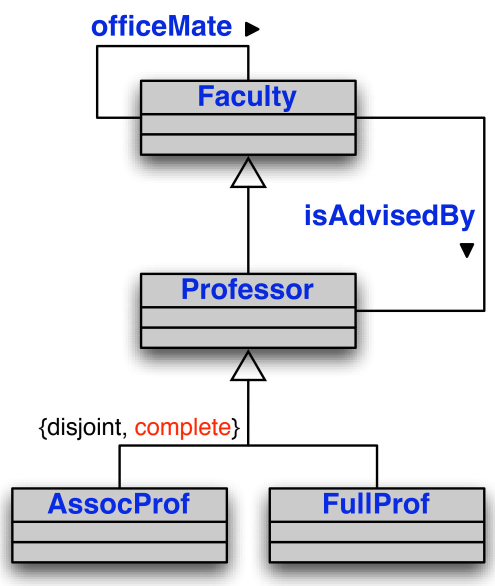


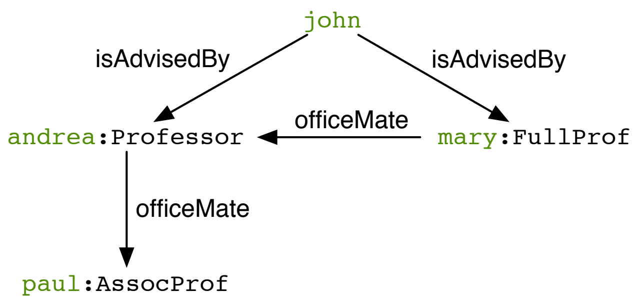


$q() \leftarrow \exists y, z.$
isAdvisedBy(john, y), FullProf(y),
officeMate(y, z), AssocProf(z)

Answer: yes or no?

QA under the KR assumption – Andrea's Example (cont'd)

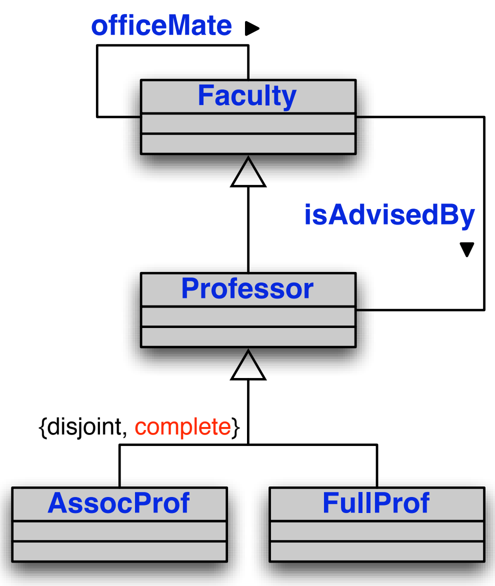


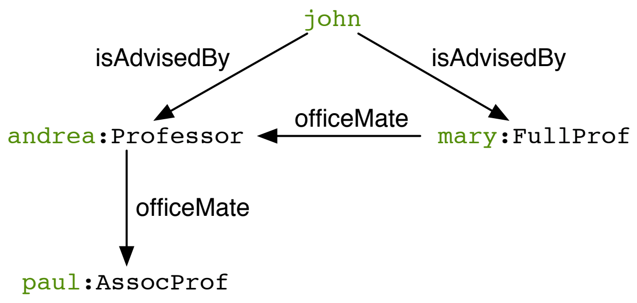


$$q() \leftarrow \exists y, z.\ \mathsf{isAdvisedBy}(\mathtt{john}, y),\ \mathsf{FullProf}(y),\ \mathsf{officeMate}(y, z),\ \mathsf{AssocProf}(z)$$

Answer: yes or no?

QA under the KR assumption – Andrea's Example (cont'd)





$$q() \leftarrow \exists y, z.\ \mathsf{isAdvisedBy}(\mathtt{john}, y),\ \mathsf{FullProf}(y),\ \mathsf{officeMate}(y, z),\ \mathsf{AssocProf}(z)$$

Answer: **yes!**

To determine this answer, we need to resort to **reasoning by cases**.

Query answering when accessing data through ontologies

We have to face the difficulties of both DB and KB assumptions:

- The actual **data** is stored in external information sources (i.e., databases), and thus its size is typically **very large**.
- The ontology introduces **incompleteness** of information, and we have to do logical inference, rather than query evaluation.
- We want to take into account at **runtime** the **constraints** expressed in the ontology.
- We want to answer **complex database-like queries**.
- We may have to deal with multiple information sources, and thus face also the problems that are typical of data integration.

Certain answers to a query

Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an ontology, $\mathcal{I}$ an interpretation for $\mathcal{O}$, and $q(\vec{x}) \leftarrow \exists \vec{y}.\, conj(\vec{x}, \vec{y})$ a CQ.

Def.: The **answer** to $q(\vec{x})$ over $\mathcal{I}$, denoted $q^{\mathcal{I}}$

. . . is the set of **tuples** $\vec{c}$ **of constants of** $\mathcal{A}$ such that the formula $\exists \vec{y}.\, conj(\vec{c}, \vec{y})$ evaluates to true in $\mathcal{I}$.

We are interested in finding those answers that hold in all models of an ontology.

Def.: The **certain answers** to $q(\vec{x})$ over $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, denoted $cert(q, \mathcal{O})$

. . . are the **tuples** $\vec{c}$ **of constants of** $\mathcal{A}$ such that $\vec{c} \in q^{\mathcal{I}}$, for every model $\mathcal{I}$ of $\mathcal{O}$.

Note: when q is boolean, we write $\mathcal{O} \models q$ iff q evaluates to true in every model $\mathcal{I}$ of $\mathcal{O}$, $\mathcal{O} \not\models q$ otherwise.

Certain answers to a query

Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an ontology, $\mathcal{I}$ an interpretation for $\mathcal{O}$, and $q(\vec{x}) \leftarrow \exists \vec{y}.\, conj(\vec{x}, \vec{y})$ a CQ.

Def.: The **answer** to $q(\vec{x})$ over $\mathcal{I}$, denoted $q^{\mathcal{I}}$

. . . is the set of **tuples $\vec{c}$ of constants of $\mathcal{A}$** such that the formula $\exists \vec{y}.\, conj(\vec{c}, \vec{y})$ evaluates to true in $\mathcal{I}$.

We are interested in finding those answers that hold in all models of an ontology.

Def.: The **certain answers** to $q(\vec{x})$ over $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, denoted $cert(q, \mathcal{O})$

. . . are the **tuples $\vec{c}$ of constants of $\mathcal{A}$** such that $\vec{c} \in q^{\mathcal{I}}$, for every model $\mathcal{I}$ of $\mathcal{O}$.

Note: when q is boolean, we write $\mathcal{O} \models q$ iff q evaluates to true in every model $\mathcal{I}$ of $\mathcal{O}$, $\mathcal{O} \not\models q$ otherwise.

Certain answers to a query

Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an ontology, $\mathcal{I}$ an interpretation for $\mathcal{O}$, and $q(\vec{x}) \leftarrow \exists \vec{y}.\, conj(\vec{x}, \vec{y})$ a CQ.

Def.: The **answer** to $q(\vec{x})$ over $\mathcal{I}$, denoted $q^{\mathcal{I}}$

. . . is the set of **tuples $\vec{c}$ of constants of $\mathcal{A}$** such that the formula $\exists \vec{y}.\, conj(\vec{c}, \vec{y})$ evaluates to true in $\mathcal{I}$.

We are interested in finding those answers that hold in all models of an ontology.

Def.: The **certain answers** to $q(\vec{x})$ over $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, denoted $cert(q, \mathcal{O})$

. . . are the **tuples $\vec{c}$ of constants of $\mathcal{A}$** such that $\vec{c} \in q^{\mathcal{I}}$, for every model $\mathcal{I}$ of $\mathcal{O}$.

Note: when q is boolean, we write $\mathcal{O} \models q$ iff q evaluates to true in every model $\mathcal{I}$ of $\mathcal{O}$, $\mathcal{O} \not\models q$ otherwise.

Data complexity

Various parameters affect the complexity of query answering over an ontology.

Depending on which parameters we consider, we get different complexity measures:

- **Data complexity**: only the size of the ABox (i.e., the data) matters.
  TBox and query are considered fixed.
- **Schema complexity**: only the size of the TBox (i.e., the schema) matters.
  ABox and query are considered fixed.
- **Combined complexity**: no parameter is considered fixed.

In the integration setting, **the size of the data largely dominates** the size of the conceptual layer (and of the query).
$\rightsquigarrow$ **Data complexity** is the relevant complexity measure.

Data complexity

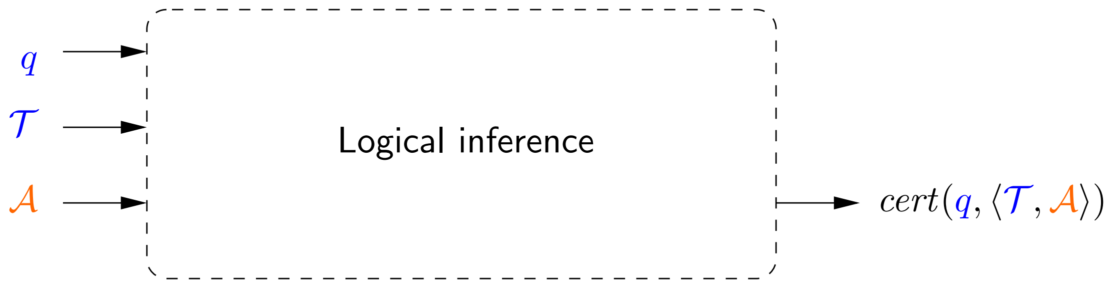
Various parameters affect the complexity of query answering over an ontology.

Depending on which parameters we consider, we get different complexity measures:

- **Data complexity**: only the size of the ABox (i.e., the data) matters.
  TBox and query are considered fixed.
- Schema complexity: only the size of the TBox (i.e., the schema) matters.
  ABox and query are considered fixed.
- Combined complexity: no parameter is considered fixed.

In the integration setting, **the size of the data largely dominates** the size of the conceptual layer (and of the query).
$\rightsquigarrow$ **Data complexity** is the relevant complexity measure.

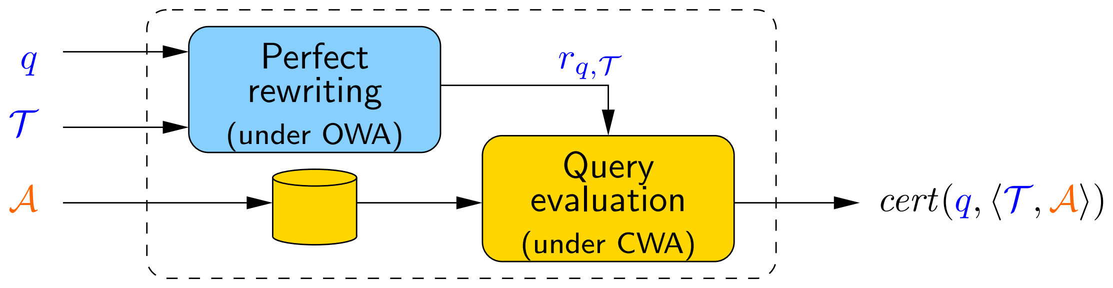
Inference in query answering



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

$\rightsquigarrow$ Query answering by **query rewriting**.

Query rewriting

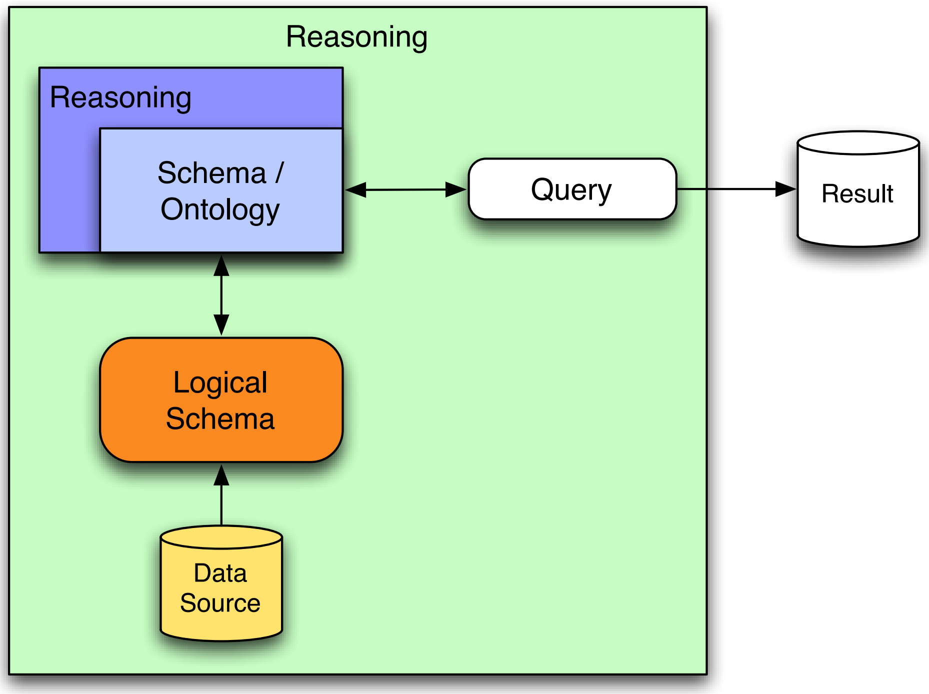

q
$\mathcal{T}$
$\mathcal{A}$
Perfect rewriting (under OWA)
$r_{q,\mathcal{T}}$
Query evaluation (under CWA)
$cert(q, \langle \mathcal{T}, \mathcal{A} \rangle)$

Query answering can **always** be thought as done in two phases:

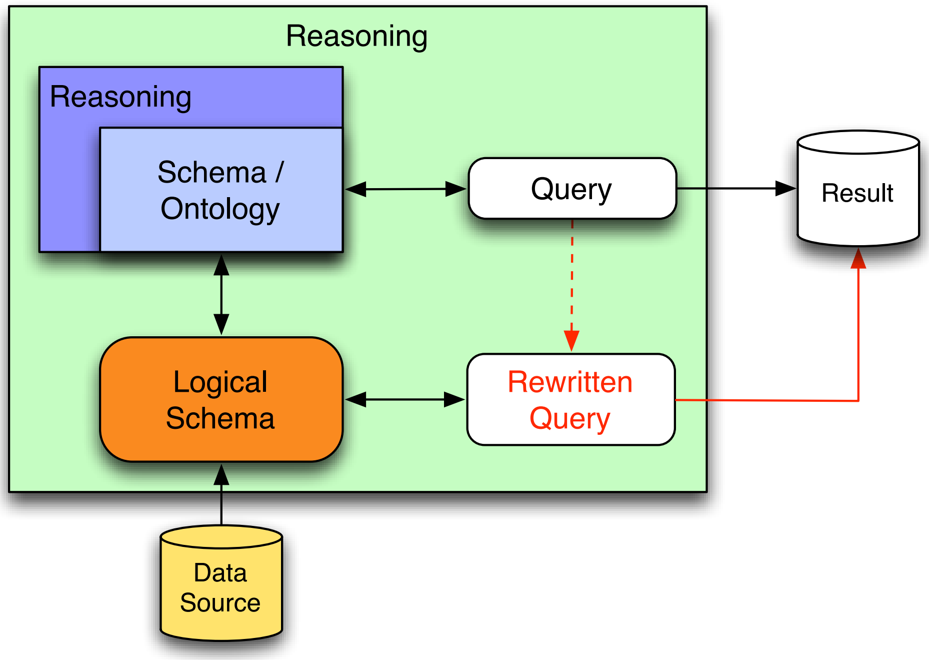
1. **Perfect rewriting**: produce from q and the TBox $\mathcal{T}$ a new query $r_{q,\mathcal{T}}$ (called the perfect rewriting of q w.r.t. $\mathcal{T}$).
2. **Query evaluation**: evaluate $r_{q,\mathcal{T}}$ over the ABox $\mathcal{A}$ seen as a complete database (and without considering the TBox $\mathcal{T}$).
   $\rightsquigarrow$ Produces $cert(q, \langle \mathcal{T}, \mathcal{A} \rangle)$.

Note: The "always" holds if we pose no restriction on the language in which to express the rewriting $r_{q,\mathcal{T}}$.

Query rewriting (cont'd)



Query rewriting (cont'd)



Language of the rewriting

The expressiveness of the ontology language affects the **query language into which we are able to rewrite CQs**:

- When we can rewrite into **FOL/SQL**.
  $\rightsquigarrow$ Query evaluation can be done in SQL, i.e., via an **RDBMS** (*Note:* FOL is in LogSpace).
- When we can rewrite into an NLogSpace-hard language.
  $\rightsquigarrow$ Query evaluation requires (at least) linear recursion.
- When we can rewrite into a PTime-hard language.
  $\rightsquigarrow$ Query evaluation requires full recursion (e.g., Datalog).
- When we can rewrite into a coNP-hard language.
  $\rightsquigarrow$ Query evaluation requires (at least) power of Disjunctive Datalog.

Language of the rewriting

The expressiveness of the ontology language affects the **query language into which we are able to rewrite CQs**:

- When we can rewrite into **FOL/SQL**.
  $\rightsquigarrow$ Query evaluation can be done in SQL, i.e., via an **RDBMS** (*Note:* FOL is in LogSpace).
- When we can rewrite into an NLogSpace-hard language.
  $\rightsquigarrow$ Query evaluation requires (at least) linear recursion.
- When we can rewrite into a PTime-hard language.
  $\rightsquigarrow$ Query evaluation requires full recursion (e.g., Datalog).
- When we can rewrite into a coNP-hard language.
  $\rightsquigarrow$ Query evaluation requires (at least) power of Disjunctive Datalog.

Language of the rewriting

The expressiveness of the ontology language affects the **query language into which we are able to rewrite CQs**:

- When we can rewrite into **FOL/SQL**.
  $\rightsquigarrow$ Query evaluation can be done in SQL, i.e., via an **RDBMS** (*Note:* FOL is in LogSpace).
- When we can rewrite into an NLogSpace-hard language.
  $\rightsquigarrow$ Query evaluation requires (at least) linear recursion.
- When we can rewrite into a PTime-hard language.
  $\rightsquigarrow$ Query evaluation requires full recursion (e.g., Datalog).
- When we can rewrite into a coNP-hard language.
  $\rightsquigarrow$ Query evaluation requires (at least) power of Disjunctive Datalog.

Language of the rewriting

The expressiveness of the ontology language affects the **query language into which we are able to rewrite CQs**:

- When we can rewrite into **FOL/SQL**.
  $\rightsquigarrow$ Query evaluation can be done in SQL, i.e., via an **RDBMS** (*Note:* FOL is in LogSpace).
- When we can rewrite into an NLogSpace-hard language.
  $\rightsquigarrow$ Query evaluation requires (at least) linear recursion.
- When we can rewrite into a PTime-hard language.
  $\rightsquigarrow$ Query evaluation requires full recursion (e.g., Datalog).
- When we can rewrite into a coNP-hard language.
  $\rightsquigarrow$ Query evaluation requires (at least) power of Disjunctive Datalog.

Complexity of query answering in DLs

Problem of rewriting is related to **complexity of query answering**.

Studied extensively for (unions of) CQs and various ontology languages:

	Combined complexity	Data complexity
Plain databases	NP-complete	in LOGSPACE [2]
OWL 2 (and less)	2EXPTIME-complete	coNP-hard [1]

[1] Already for a TBox with a single disjunction (see Andrea's example).
[2] This is what we need to scale with the data.

Questions

- Can we find interesting families of DLs for which the query answering problem can be solved efficiently (i.e., in LOGSPACE)?
- If yes, can we leverage relational database technology for query answering?

Complexity of query answering in DLs

Problem of rewriting is related to **complexity of query answering**.

Studied extensively for (unions of) CQs and various ontology languages:

	Combined complexity	Data complexity
Plain databases	NP-complete	in LOGSPACE [2]
OWL 2 (and less)	2EXPTIME-complete	coNP-hard [1]

[1] Already for a TBox with a single disjunction (see Andrea's example).
[2] This is what we need to scale with the data.

Questions

- Can we find interesting families of DLs for which the query answering problem can be solved efficiently (i.e., in LOGSPACE)?
- If yes, can we leverage relational database technology for query answering?

Complexity of query answering in DLs

Problem of rewriting is related to **complexity of query answering**.

Studied extensively for (unions of) CQs and various ontology languages:

	Combined complexity	Data complexity
Plain databases	NP-complete	in LOGSPACE [2]
OWL 2 (and less)	2EXPTIME-complete	coNP-hard [1]

[1] Already for a TBox with a single disjunction (see Andrea's example).
[2] This is what we need to scale with the data.

Questions

- Can we find interesting families of DLs for which the query answering problem can be solved efficiently (i.e., in LOGSPACE)?
- If yes, can we leverage relational database technology for query answering?

Complexity of query answering in DLs

Problem of rewriting is related to **complexity of query answering**.

Studied extensively for (unions of) CQs and various ontology languages:

	Combined complexity	Data complexity
Plain databases	NP-complete	in LOGSPACE [2]
OWL 2 (and less)	2EXPTIME-complete	coNP-hard [1]

[1] Already for a TBox with a single disjunction (see Andrea's example).
[2] This is what we need to scale with the data.

Questions

- Can we find interesting families of DLs for which the query answering problem can be solved efficiently (i.e., in LOGSPACE)?
- If yes, can we leverage relational database technology for query answering?

Outline

The *DL-Lite* family

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

DL-Lite$_{\mathcal{A}}$ provides robust foundations for Ontology-Based Data Access.

The *DL-Lite* family

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

DL-Lite$_\mathcal{A}$ provides robust foundations for Ontology-Based Data Access.

DL-Lite$_\mathcal{A}$ ontologies

TBox assertions:

- Class (concept) inclusion assertions: $B \sqsubseteq C$, with:

$$\begin{array}{lcl} B & \longrightarrow & A \mid \exists Q \\ C & \longrightarrow & B \mid \neg B \end{array}$$

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

$$\begin{array}{lcl} Q & \longrightarrow & P \mid P^- \\ R & \longrightarrow & Q \mid \neg Q \end{array}$$

- Functionality assertions: $(\textbf{funct}\ Q)$
- **Proviso:** functional properties cannot be specialized.

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

Note: *DL-Lite*$_\mathcal{A}$ distinguishes also between object and data properties (ignored here).

Semantics of *DL-Lite*$_\mathcal{A}$

Construct	Syntax	Example	Semantics
atomic conc.	A	Doctor	$A^\mathcal{I} \subseteq \Delta^\mathcal{I}$
exist. restr.	$\exists Q$	∃child$^-$	$\{d \mid \exists e. (d,e) \in Q^\mathcal{I}\}$
at. conc. neg.	$\neg A$	¬Doctor	$\Delta^\mathcal{I} \setminus A^\mathcal{I}$
conc. neg.	$\neg\exists Q$	¬∃child	$\Delta^\mathcal{I} \setminus (\exists Q)^\mathcal{I}$
atomic role	P	child	$P^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$
inverse role	P^-	child$^-$	$\{(o,o') \mid (o',o) \in P^\mathcal{I}\}$
role negation	$\neg Q$	¬manages	$(\Delta^\mathcal{I} \times \Delta^\mathcal{I}) \setminus Q^\mathcal{I}$
conc. incl.	$B \sqsubseteq C$	Father ⊑ ∃child	$B^\mathcal{I} \subseteq C^\mathcal{I}$
role incl.	$Q \sqsubseteq R$	hasFather ⊑ child$^-$	$Q^\mathcal{I} \subseteq R^\mathcal{I}$
funct. asser.	(**funct** Q)	(**funct** succ)	$\forall d,e,e'.(d,e) \in Q^\mathcal{I} \wedge (d,e') \in Q^\mathcal{I} \rightarrow e = e'$
mem. asser.	$A(c)$	Father(bob)	$c^\mathcal{I} \in A^\mathcal{I}$
mem. asser.	$P(c_1, c_2)$	child(bob, ann)	$(c_1^\mathcal{I}, c_2^\mathcal{I}) \in P^\mathcal{I}$

DL-Lite$_\mathcal{A}$ (as all DLs of the *DL-Lite* family) adopts the Unique Name Assumption (UNA), i.e., different individuals denote different objects.

Capturing basic ontology constructs in *DL-Lite*$_\mathcal{A}$

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

Note 1: *DL-Lite*$_\mathcal{A}$ cannot capture completeness of a hierarchy. This would require **disjunction** (i.e., **OR**).

Note2: *DL-Lite*$_\mathcal{A}$ can be extended to capture also **min cardinality constraints** ($A \sqsubseteq \leq nQ$) and **max cardinality constraints** ($A \sqsubseteq \geq nQ$) (not considered here for simplicity).

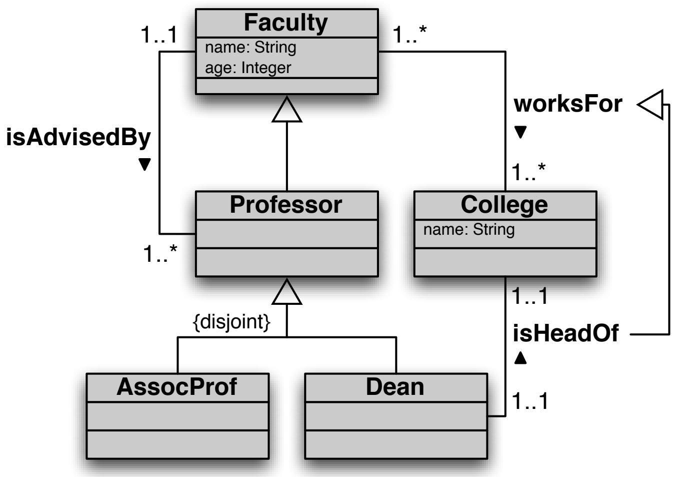
Capturing basic ontology constructs in *DL-Lite*$_{\mathcal{A}}$

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

Note 1: *DL-Lite*$_{\mathcal{A}}$ cannot capture completeness of a hierarchy. This would require **disjunction** (i.e., **OR**).

Note2: *DL-Lite*$_{\mathcal{A}}$ can be extended to capture also **min cardinality constraints** ($A \sqsubseteq \leq nQ$) and **max cardinality constraints** ($A \sqsubseteq \geq nQ$) (not considered here for simplicity).

Example



Professor	$\sqsubseteq$	Faculty
AssocProf	$\sqsubseteq$	Professor
Dean	$\sqsubseteq$	Professor
AssocProf	$\sqsubseteq$	¬Dean
Faculty	$\sqsubseteq$	$\exists$*age*
$\exists$*age*$^-$	$\sqsubseteq$	`xsd:integer`
	(**funct** *age*)	
∃worksFor	$\sqsubseteq$	Faculty
∃worksFor$^-$	$\sqsubseteq$	College
Faculty	$\sqsubseteq$	∃worksFor
College	$\sqsubseteq$	∃worksFor$^-$
∃isHeadOf	$\sqsubseteq$	Dean
∃isHeadOf$^-$	$\sqsubseteq$	College
Dean	$\sqsubseteq$	∃isHeadOf
College	$\sqsubseteq$	∃isHeadOf$^-$
isHeadOf	$\sqsubseteq$	worksFor
	(**funct** isHeadOf)	
	(**funct** isHeadOf$^-$)	
	⋮	

Observations on *DL-Lite*$_{\mathcal{A}}$

- Captures all the basic constructs of **UML Class Diagrams** and of the **ER Model** . . .
- . . . **except covering constraints** in generalizations.
- Is the logical underpinning of **OWL2 QL**, one of the OWL 2 Profiles.
- Extends (the DL fragment of) the ontology language **RDFS**.
- Is completely symmetric w.r.t. **direct and inverse properties**.
- Does **not** enjoy the **finite model property**, i.e., reasoning and query answering differ depending on whether we consider or not also infinite models.

Observations on *DL-Lite*$_{\mathcal{A}}$

- Captures all the basic constructs of **UML Class Diagrams** and of the **ER Model** . . .
- . . . **except covering constraints** in generalizations.
- Is the logical underpinning of **OWL2 QL**, one of the OWL 2 Profiles.
- Extends (the DL fragment of) the ontology language **RDFS**.
- Is completely symmetric w.r.t. **direct and inverse properties**.
- Does **not** enjoy the **finite model property**, i.e., reasoning and query answering differ depending on whether we consider or not also infinite models.

Query answering in *DL-Lite*$_\mathcal{A}$

- We study answering of UCQs over *DL-Lite*$_\mathcal{A}$ ontologies via query rewriting.
- We first consider query answering over **satisfiable ontologies**, i.e., that admit at least one model.
- Then, we show how to exploit query answering over satisfiable ontologies to establish ontology satisfiability.

Remark

we call **positive inclusions (PIs)** assertions of the form

$$\begin{array}{ccc} B_1 & \sqsubseteq & B_2 \\ Q_1 & \sqsubseteq & Q_2 \end{array}$$

whereas we call negative inclusions (NIs) assertions of the form

$$\begin{array}{ccc} B_1 & \sqsubseteq & \neg B_2 \\ Q_1 & \sqsubseteq & \neg Q_2 \end{array}$$

Query answering in *DL-Lite*$_\mathcal{A}$

- We study answering of UCQs over *DL-Lite*$_\mathcal{A}$ ontologies via query rewriting.
- We first consider query answering over **satisfiable ontologies**, i.e., that admit at least one model.
- Then, we show how to exploit query answering over satisfiable ontologies to establish ontology satisfiability.

Remark

we call **positive inclusions (PIs)** assertions of the form

$$\begin{array}{ccc} B_1 & \sqsubseteq & B_2 \\ Q_1 & \sqsubseteq & Q_2 \end{array}$$

whereas we call negative inclusions (NIs) assertions of the form

$$\begin{array}{ccc} B_1 & \sqsubseteq & \neg B_2 \\ Q_1 & \sqsubseteq & \neg Q_2 \end{array}$$

Query answering over satisfiable *DL-Lite*$_{\mathcal{A}}$ ontologies

Theorem

Let q be a boolean UCQs and $\mathcal{T} = \mathcal{T}_{\rm PI} \cup \mathcal{T}_{\rm NI} \cup \mathcal{T}_{\sf funct}$ be a TBox s.t.

- $\mathcal{T}_{\rm PI}$ is a set of PIs
- $\mathcal{T}_{\rm NI}$ is a set of NIs
- $\mathcal{T}_{\sf funct}$ is a set of functionalities.

For each ABox $\mathcal{A}$ such that $\langle \mathcal{T}, \mathcal{A} \rangle$ **is satisfiable**, we have that

$$\langle \mathcal{T}, \mathcal{A} \rangle \models q \textbf{ iff } \langle \mathcal{T}_{\sf PI}, \mathcal{A} \rangle \models q.$$

Proof [intuition]

q is a positive query, i.e., it does not contain atoms with negation nor inequality. $\mathcal{T}_{\rm NI}$ and $\mathcal{T}_{\sf funct}$ only contribute to infer new negative consequences, i.e, sentences involving negation.

If q is non-boolean, we have that $cert(q, \langle \mathcal{T}, \mathcal{A} \rangle) = cert(q, \langle \mathcal{T}_{\rm PI}, \mathcal{A} \rangle)$.

Query answering over satisfiable *DL-Lite*$_{\mathcal{A}}$ ontologies

Theorem

Let q be a boolean UCQs and $\mathcal{T} = \mathcal{T}_{\rm PI} \cup \mathcal{T}_{\rm NI} \cup \mathcal{T}_{\sf funct}$ be a TBox s.t.

- $\mathcal{T}_{\rm PI}$ is a set of PIs
- $\mathcal{T}_{\rm NI}$ is a set of NIs
- $\mathcal{T}_{\sf funct}$ is a set of functionalities.

For each ABox $\mathcal{A}$ such that $\langle \mathcal{T}, \mathcal{A} \rangle$ **is satisfiable**, we have that

$$\langle \mathcal{T}, \mathcal{A} \rangle \models q \textbf{ iff } \langle \mathcal{T}_{\sf PI}, \mathcal{A} \rangle \models q.$$

Proof [intuition]

q is a positive query, i.e., it does not contain atoms with negation nor inequality. $\mathcal{T}_{\rm NI}$ and $\mathcal{T}_{\sf funct}$ only contribute to infer new negative consequences, i.e, sentences involving negation.

If q is non-boolean, we have that $cert(q, \langle \mathcal{T}, \mathcal{A} \rangle) = cert(q, \langle \mathcal{T}_{\rm PI}, \mathcal{A} \rangle)$.

Query answering over satisfiable *DL-Lite*$_{\mathcal{A}}$ ontologies

Theorem

Let q be a boolean UCQs and $\mathcal{T} = \mathcal{T}_{\mathrm{PI}} \cup \mathcal{T}_{\mathrm{NI}} \cup \mathcal{T}_{\mathsf{funct}}$ be a TBox s.t.

- $\mathcal{T}_{\mathrm{PI}}$ is a set of PIs
- $\mathcal{T}_{\mathrm{NI}}$ is a set of NIs
- $\mathcal{T}_{\mathsf{funct}}$ is a set of functionalities.

For each ABox $\mathcal{A}$ such that $\langle \mathcal{T}, \mathcal{A} \rangle$ **is satisfiable**, we have that

$$\langle \mathcal{T}, \mathcal{A} \rangle \models q \text{ **iff** } \langle \mathcal{T}_{\mathsf{PI}}, \mathcal{A} \rangle \models q.$$

Proof [intuition]

q is a positive query, i.e., it does not contain atoms with negation nor inequality. $\mathcal{T}_{\mathrm{NI}}$ and $\mathcal{T}_{\mathsf{funct}}$ only contribute to infer new negative consequences, i.e, sentences involving negation.

If q is non-boolean, we have that $cert(q, \langle \mathcal{T}, \mathcal{A} \rangle) = cert(q, \langle \mathcal{T}_{\mathrm{PI}}, \mathcal{A} \rangle)$.

Satisfiability of *DL-Lite*$_{\mathcal{A}}$ ontologies

$\langle \mathcal{T}, \emptyset \rangle$ is always satisfiable. That is, inconsistency in *DL-Lite*$_{\mathcal{A}}$ may arise only when ABox assertions contradict the TBox.

$\langle \mathcal{T}_{\mathrm{PI}}, \mathcal{A} \rangle$, where $\mathcal{T}_{\mathrm{PI}}$ contains only PIs, is always satisfiable. That is, inconsistency in *DL-Lite*$_{\mathcal{A}}$ may arise only when ABox assertions violate functionalities or NIs.

Example: **TBox $\mathcal{T}$:** Professor $\sqsubseteq \neg$Student
$\exists$teaches $\sqsubseteq$ Professor
(**funct** teaches$^-$)

ABox $\mathcal{A}$: teaches($\mathtt{John}, \mathtt{databases}$)
Student($\mathtt{John}$)
teaches($\mathtt{Mark}, \mathtt{databases}$)

Violations of functionalities and of NIs can be checked separately!

Satisfiability of *DL-Lite*$_{\mathcal{A}}$ ontologies

$\langle \mathcal{T}, \emptyset \rangle$ is always satisfiable. That is, inconsistency in *DL-Lite*$_{\mathcal{A}}$ may arise only when ABox assertions contradict the TBox.

$\langle \mathcal{T}_{\mathrm{PI}}, \mathcal{A} \rangle$, where $\mathcal{T}_{\mathrm{PI}}$ contains only PIs, is always satisfiable. That is, inconsistency in *DL-Lite*$_{\mathcal{A}}$ may arise only when ABox assertions violate functionalities or NIs.

Example: **TBox** $\mathcal{T}$**:** Professor $\sqsubseteq \neg$Student
$\exists$teaches $\sqsubseteq$ Professor
(**funct** teaches^{-})

ABox $\mathcal{A}$**:** teaches($\mathtt{John}, \mathtt{databases}$)
Student($\mathtt{John}$)
teaches($\mathtt{Mark}, \mathtt{databases}$)

Violations of functionalities and of NIs can be checked separately!

Satisfiability of *DL-Lite*$_{\mathcal{A}}$ ontologies

$\langle \mathcal{T}, \emptyset \rangle$ is always satisfiable. That is, inconsistency in *DL-Lite*$_{\mathcal{A}}$ may arise only when ABox assertions contradict the TBox.

$\langle \mathcal{T}_{\mathrm{PI}}, \mathcal{A} \rangle$, where $\mathcal{T}_{\mathrm{PI}}$ contains only PIs, is always satisfiable. That is, inconsistency in *DL-Lite*$_{\mathcal{A}}$ may arise only when ABox assertions violate functionalities or NIs.

Example: **TBox** $\mathcal{T}$**:** Professor $\sqsubseteq \neg$Student
$\exists$teaches $\sqsubseteq$ Professor
(**funct** teaches^{-})

ABox $\mathcal{A}$**:** teaches($\mathtt{John}, \mathtt{databases}$)
Student($\mathtt{John}$)
teaches($\mathtt{Mark}, \mathtt{databases}$)

Violations of functionalities and of NIs can be checked separately!

Satisfiability of *DL-Lite*$_{\mathcal{A}}$ ontologies

$\langle \mathcal{T}, \emptyset \rangle$ is always satisfiable. That is, inconsistency in *DL-Lite*$_{\mathcal{A}}$ may arise only when ABox assertions contradict the TBox.

$\langle \mathcal{T}_{\mathrm{PI}}, \mathcal{A} \rangle$, where $\mathcal{T}_{\mathrm{PI}}$ contains only PIs, is always satisfiable. That is, inconsistency in *DL-Lite*$_{\mathcal{A}}$ may arise only when ABox assertions violate functionalities or NIs.

Example: **TBox** $\mathcal{T}$**:** Professor $\sqsubseteq$ ¬Student
∃teaches $\sqsubseteq$ Professor
(**funct** teaches$^-$)

ABox $\mathcal{A}$**:** teaches(`John`, `databases`)
Student(`John`)
teaches(`Mark`, `databases`)

Violations of functionalities and of NIs can be checked separately!

Satisfiability of *DL-Lite*$_{\mathcal{A}}$ ontologies: Checking functs

Theorem

Let $\mathcal{T}_{\mathrm{PI}}$ be a TBox with only PIs, and (**funct** Q) a functionality assertion. Then, for any ABox $\mathcal{A}$,
$\langle \mathcal{T}_{\mathrm{PI}} \cup \{(\text{funct } Q)\}, \mathcal{A} \rangle$ is sat iff $\mathcal{A} \not\models \exists x, y, z. Q(x,y) \land Q(x,z) \land y \neq z$.

Proof [sketch]

$\langle \mathcal{T}_{\mathrm{PI}} \cup \{(\text{funct } Q)\}, \mathcal{A} \rangle$ is satisfiable iff $\langle \mathcal{T}_{\mathrm{PI}}, \mathcal{A} \rangle \not\models \neg(\text{funct } Q)$. This holds iff $\mathcal{A} \not\models \neg(\text{funct } Q)$ (separability property – sophisticated proof).
From separability, the claim easily follows, by noticing that (**funct** Q) corresponds to the FOL sentence $\forall x, y, z. Q(x,y) \land Q(x,z) \rightarrow y = z$.

For a set of functionalities, we take the union of sentences of the form above (which corresponds to a boolean FOL query).

Checking satisfiability wrt functionalities therefore amounts to evaluate a FOL query over the ABox.

Satisfiability of *DL-Lite*$_{\mathcal{A}}$ ontologies: Checking functs

Theorem

Let $\mathcal{T}_{\mathrm{PI}}$ be a TBox with only PIs, and (**funct** Q) a functionality assertion. Then, for any ABox $\mathcal{A}$,
$\langle \mathcal{T}_{\mathrm{PI}} \cup \{(\textbf{funct } Q)\}, \mathcal{A}\rangle$ **is sat iff** $\mathcal{A} \not\models \exists x, y, z. Q(x, y) \land Q(x, z) \land y \neq z$.

Proof [sketch]

$\langle \mathcal{T}_{\mathrm{PI}} \cup \{(\textbf{funct } Q)\}, \mathcal{A}\rangle$ is satisfiable iff $\langle \mathcal{T}_{\mathrm{PI}}, \mathcal{A}\rangle \not\models \neg(\textbf{funct } Q)$. This holds iff $\mathcal{A} \not\models \neg(\textbf{funct } Q)$ (separability property – sophisticated proof). From separability, the claim easily follows, by noticing that (**funct** Q) corresponds to the FOL sentence $\forall x, y, z. Q(x, y) \land Q(x, z) \rightarrow y = z$.

For a set of functionalities, we take the union of sentences of the form above (which corresponds to a boolean FOL query).

Checking satisfiability wrt functionalities therefore amounts to evaluate a FOL query over the ABox.

Satisfiability of *DL-Lite*$_{\mathcal{A}}$ ontologies: Checking functs

Theorem

Let $\mathcal{T}_{\mathrm{PI}}$ be a TBox with only PIs, and (**funct** Q) a functionality assertion. Then, for any ABox $\mathcal{A}$,
$\langle \mathcal{T}_{\mathrm{PI}} \cup \{(\textbf{funct } Q)\}, \mathcal{A}\rangle$ **is sat iff** $\mathcal{A} \not\models \exists x, y, z. Q(x, y) \land Q(x, z) \land y \neq z$.

Proof [sketch]

$\langle \mathcal{T}_{\mathrm{PI}} \cup \{(\textbf{funct } Q)\}, \mathcal{A}\rangle$ is satisfiable iff $\langle \mathcal{T}_{\mathrm{PI}}, \mathcal{A}\rangle \not\models \neg(\textbf{funct } Q)$. This holds iff $\mathcal{A} \not\models \neg(\textbf{funct } Q)$ (separability property – sophisticated proof). From separability, the claim easily follows, by noticing that (**funct** Q) corresponds to the FOL sentence $\forall x, y, z. Q(x, y) \land Q(x, z) \rightarrow y = z$.

For a set of functionalities, we take the union of sentences of the form above (which corresponds to a boolean FOL query).

Checking satisfiability wrt functionalities therefore amounts to evaluate a FOL query over the ABox.

Example

TBox $\mathcal{T}$: $\mathsf{Professor} \sqsubseteq \neg\mathsf{Student}$
$\exists\mathsf{teaches} \sqsubseteq \mathsf{Professor}$
$(\textbf{funct}\ \mathsf{teaches}^-)$

The query we associate to the functionality is:

$$q() \leftarrow \mathsf{teaches}(x, y), \mathsf{teaches}(x, z), y \neq z$$

which evaluated over the ABox

ABox $\mathcal{A}$: $\mathsf{teaches}(\mathtt{John}, \mathtt{databases})$
$\mathsf{Student}(\mathtt{John})$
$\mathsf{teaches}(\mathtt{Mark}, \mathtt{databases})$

returns true.

Satisfiability of *DL-Lite*$_\mathcal{A}$ ontologies: Checking NIs

Theorem

Let $\mathcal{T}_{\mathrm{PI}}$ be a TBox with only PIs, and $A_1 \sqsubseteq \neg A_2$ a NI. For any ABox $\mathcal{A}$, **$\langle\mathcal{T}_{\mathrm{PI}} \cup \{A_1 \sqsubseteq \neg A_2\}, \mathcal{A}\rangle$ is sat iff $\langle\mathcal{T}_{\mathrm{PI}}, \mathcal{A}\rangle \not\models \exists x.A_1(x) \land A_2(x)$.**

Proof [sketch]

$\langle\mathcal{T}_{\mathrm{PI}} \cup \{A_1 \sqsubseteq \neg A_2\}, \mathcal{A}\rangle$ is satisfiable iff $\langle\mathcal{T}_{\mathrm{PI}}, \mathcal{A}\rangle \not\models \neg(A_1 \sqsubseteq \neg A_2)$. The claim follows easily by noticing that $A_1 \sqsubseteq \neg A_2$ corresponds to the FOL sentence $\forall x.A_1(x) \to \neg A_2(x)$.

The property holds for all kinds of NIs ($A \sqsubseteq \exists Q$, $\exists Q_1 \sqsubseteq \exists Q_2$, etc.)

For a set of NIs, we take the union of sentences of the form above (which corresponds to a UCQ).

Checking satisfiability wrt NIs amounts to answering a UCQ over an ontology with only PIs (this can be reduced to evaluating a UCQ over the ABox – see later).

Satisfiability of *DL-Lite*$_\mathcal{A}$ ontologies: Checking NIs

Theorem

Let $\mathcal{T}_{\mathrm{PI}}$ be a TBox with only PIs, and $A_1 \sqsubseteq \neg A_2$ a NI. For any ABox $\mathcal{A}$, $\langle \mathcal{T}_{\mathbf{PI}} \cup \{A_1 \sqsubseteq \neg A_2\}, \mathcal{A}\rangle$ **is sat iff** $\langle \mathcal{T}_{\mathbf{PI}}, \mathcal{A}\rangle \not\models \exists x.A_1(x) \wedge A_2(x)$**.**

Proof [sketch]

$\langle \mathcal{T}_{\mathrm{PI}} \cup \{A_1 \sqsubseteq \neg A_2\}, \mathcal{A}\rangle$ is satisfiable iff $\langle \mathcal{T}_{\mathrm{PI}}, \mathcal{A}\rangle \not\models \neg(A_1 \sqsubseteq \neg A_2)$. The claim follows easily by noticing that $A_1 \sqsubseteq \neg A_2$ corresponds to the FOL sentence $\forall x.A_1(x) \rightarrow \neg A_2(x)$.

The property holds for all kinds of NIs ($A \sqsubseteq \exists Q$, $\exists Q_1 \sqsubseteq \exists Q_2$, etc.)

For a set of NIs, we take the union of sentences of the form above (which corresponds to a UCQ).

Checking satisfiability wrt NIs amounts to answering a UCQ over an ontology with only PIs (this can be reduced to evaluating a UCQ over the ABox – see later).

Satisfiability of *DL-Lite*$_\mathcal{A}$ ontologies: Checking NIs

Theorem

Let $\mathcal{T}_{\mathrm{PI}}$ be a TBox with only PIs, and $A_1 \sqsubseteq \neg A_2$ a NI. For any ABox $\mathcal{A}$, $\langle \mathcal{T}_{\mathbf{PI}} \cup \{A_1 \sqsubseteq \neg A_2\}, \mathcal{A}\rangle$ **is sat iff** $\langle \mathcal{T}_{\mathbf{PI}}, \mathcal{A}\rangle \not\models \exists x.A_1(x) \wedge A_2(x)$**.**

Proof [sketch]

$\langle \mathcal{T}_{\mathrm{PI}} \cup \{A_1 \sqsubseteq \neg A_2\}, \mathcal{A}\rangle$ is satisfiable iff $\langle \mathcal{T}_{\mathrm{PI}}, \mathcal{A}\rangle \not\models \neg(A_1 \sqsubseteq \neg A_2)$. The claim follows easily by noticing that $A_1 \sqsubseteq \neg A_2$ corresponds to the FOL sentence $\forall x.A_1(x) \rightarrow \neg A_2(x)$.

The property holds for all kinds of NIs ($A \sqsubseteq \exists Q$, $\exists Q_1 \sqsubseteq \exists Q_2$, etc.)

For a set of NIs, we take the union of sentences of the form above (which corresponds to a UCQ).

Checking satisfiability wrt NIs amounts to answering a UCQ over an ontology with only PIs (this can be reduced to evaluating a UCQ over the ABox – see later).

Example

TBox $\mathcal{T}$: $\mathsf{Professor} \sqsubseteq \neg\mathsf{Student}$
$\exists\mathsf{teaches} \sqsubseteq \mathsf{Professor}$
$(\textbf{funct}\ \mathsf{teaches}^-)$

The query we associate to the NI is:

$$q() \leftarrow \mathsf{Student}(x), \mathsf{Professor}(x)$$

whose answer over the ontology

$$\begin{array}{l} \exists\mathsf{teaches} \sqsubseteq \mathsf{Professor} \\ \mathsf{teaches}(\mathtt{John}, \mathtt{databases}) \\ \mathsf{Student}(\mathtt{John}) \\ \mathsf{teaches}(\mathtt{Mark}, \mathtt{databases}) \end{array}$$

is true.

Checking satisfiability of *DL-Lite*$_\mathcal{A}$ ontologies

Satisfiability of a *DL-Lite*$_\mathcal{A}$ ontology $\mathcal{O} = \langle\mathcal{T}, \mathcal{A}\rangle$ is reduced to evaluation of a first order query over $\mathcal{A}$, obtained by uniting

(a) the FOL query associated to functionalities in $\mathcal{T}$ to

(b) the UCQs produced by a rewriting procedure (depending only on the PIs in $\mathcal{T}$) applied to the query associated to NIs in $\mathcal{T}$.

$\rightsquigarrow$ Ontology satisfiability in *DL-Lite*$_\mathcal{A}$ can be done using RDMBS technology.

Query answering in *DL-Lite*$_{\mathcal{A}}$: Query rewriting

To the aim of answering queries, from now on we assume that $\mathcal{T}$ contains only PIs.

Given a CQ q and a satisfiable ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, we compute $cert(q, \mathcal{O})$ as follows

1. using $\mathcal{T}$, **reformulate** q as a **union** $r_{q,\mathcal{T}}$ **of CQs**.
2. Evaluate $r_{q,\mathcal{T}}$ directly over $\mathcal{A}$ managed in **secondary storage via a RDBMS**.

Correctness of this procedure shows FOL-rewritability of query answering in *DL-Lite*$_{\mathcal{A}}$
$\rightsquigarrow$ Query answering over *DL-Lite*$_{\mathcal{A}}$ ontologies can be done using RDMBS technology.

Query answering in *DL-Lite*$_{\mathcal{A}}$: Query rewriting

To the aim of answering queries, from now on we assume that $\mathcal{T}$ contains only PIs.

Given a CQ q and a satisfiable ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, we compute $cert(q, \mathcal{O})$ as follows

1. using $\mathcal{T}$, **reformulate** q as a **union** $r_{q,\mathcal{T}}$ **of CQs**.
2. Evaluate $r_{q,\mathcal{T}}$ directly over $\mathcal{A}$ managed in **secondary storage via a RDBMS**.

Correctness of this procedure shows FOL-rewritability of query answering in *DL-Lite*$_{\mathcal{A}}$
$\rightsquigarrow$ Query answering over *DL-Lite*$_{\mathcal{A}}$ ontologies can be done using RDMBS technology.

Query answering in *DL-Lite*$_\mathcal{A}$: Query rewriting (cont'd)

Intuition: Use the PIs as basic rewriting rules

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

$$\mathsf{AssProfessor} \sqsubseteq \mathsf{Professor}$$

as a logic rule: $\mathsf{Professor}(z) \ \leftarrow\ \mathsf{AssProfessor}(z)$

Basic rewriting step:

when the atom unifies with the **head** of the rule (with mgu σ).
substitute the atom with the **body** of the rule (to which σ is applied).

Towards the computation of the perfect rewriting, we add to the input query above the following query ($\sigma = \{z/x\}$)

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

We say that the PI $\mathsf{AssProfessor} \sqsubseteq \mathsf{Professor}$ **applies** to the atom $\mathsf{Professor}(x)$.

Query answering in *DL-Lite*$_\mathcal{A}$: Query rewriting (cont'd)

Intuition: Use the PIs as basic rewriting rules

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

$$\mathsf{AssProfessor} \sqsubseteq \mathsf{Professor}$$

as a logic rule: $\mathsf{Professor}(z) \ \leftarrow\ \mathsf{AssProfessor}(z)$

Basic rewriting step:

when the atom unifies with the **head** of the rule (with mgu σ).
substitute the atom with the **body** of the rule (to which σ is applied).

Towards the computation of the perfect rewriting, we add to the input query above the following query ($\sigma = \{z/x\}$)

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

We say that the PI $\mathsf{AssProfessor} \sqsubseteq \mathsf{Professor}$ **applies** to the atom $\mathsf{Professor}(x)$.

Query answering in *DL-Lite*$_{\mathcal{A}}$: Query rewriting (cont'd)

Intuition: Use the PIs as basic rewriting rules

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

$$\mathsf{AssProfessor} \sqsubseteq \mathsf{Professor}$$

$$\text{as a logic rule:} \quad \mathsf{Professor}(z) \leftarrow \mathsf{AssProfessor}(z)$$

Basic rewriting step:

when the atom unifies with the **head** of the rule (with mgu σ).
substitute the atom with the **body** of the rule (to which σ is applied).

Towards the computation of the perfect rewriting, we add to the input query above the following query ($\sigma = \{z/x\}$)

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

We say that the PI $\mathsf{AssProfessor} \sqsubseteq \mathsf{Professor}$ **applies** to the atom $\mathsf{Professor}(x)$.

Query answering in *DL-Lite*$_{\mathcal{A}}$: Query rewriting (cont'd)

Consider now the query

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

$$\mathsf{Professor} \sqsubseteq \exists\mathsf{teaches}$$

$$\text{as a logic rule:} \quad \mathsf{teaches}(z_1, z_2) \leftarrow \mathsf{Professor}(z_1)$$

We add to the reformulation the query ($\sigma = \{z_1/x, z_2/y\}$)

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

Query answering in *DL-Lite*$_{\mathcal{A}}$: Query rewriting (cont'd)

Conversely, for the query

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

$$\mathsf{Professor} \sqsubseteq \exists \mathsf{teaches}$$

$$\text{as a logic rule:} \quad \mathsf{teaches}(z_1, z_2) \leftarrow \mathsf{Professor}(z_1)$$

$\mathsf{teaches}(x, \mathtt{databases})$ does not unify with $\mathsf{teaches}(z_1, z_2)$, since the **existentially quantified variable** z_2 in the head of the rule **does not unify** with the constant $\mathtt{databases}$**.**

In this case the PI **does not apply** to the atom $\mathsf{teaches}(x, \mathtt{databases})$.

The same holds for the following query, where y is **distinguished**

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

Query answering in *DL-Lite*$_{\mathcal{A}}$: Query rewriting (cont'd)

Conversely, for the query

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

$$\mathsf{Professor} \sqsubseteq \exists \mathsf{teaches}$$

$$\text{as a logic rule:} \quad \mathsf{teaches}(z_1, z_2) \leftarrow \mathsf{Professor}(z_1)$$

$\mathsf{teaches}(x, \mathtt{databases})$ does not unify with $\mathsf{teaches}(z_1, z_2)$, since the **existentially quantified variable** z_2 in the head of the rule **does not unify** with the constant $\mathtt{databases}$**.**

In this case the PI **does not apply** to the atom $\mathsf{teaches}(x, \mathtt{databases})$.

The same holds for the following query, where y is **distinguished**

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

Query answering in *DL-Lite*$_\mathcal{A}$: Query rewriting (cont'd)

Conversely, for the query

$$\mathsf{q}(x) \;\leftarrow\; \mathsf{teaches}(x, \mathtt{databases})$$

$$\mathsf{Professor} \sqsubseteq \exists\mathsf{teaches}$$
$$\text{as a logic rule:} \quad \mathsf{teaches}(z_1, z_2) \;\leftarrow\; \mathsf{Professor}(z_1)$$

$\mathsf{teaches}(x, \mathtt{databases})$ does not unify with $\mathsf{teaches}(z_1, z_2)$, since the **existentially quantified variable** z_2 in the head of the rule **does not unify** with the constant $\mathtt{databases}$**.**

In this case the PI **does not apply** to the atom $\mathsf{teaches}(x, \mathtt{databases})$.

The same holds for the following query, where y is **distinguished**

$$\mathsf{q}(x, y) \;\leftarrow\; \mathsf{teaches}(x, y)$$

Query answering in *DL-Lite*$_\mathcal{A}$: Query rewriting (cont'd)

An analogous behavior with join variables

$$\mathsf{q}(x) \;\leftarrow\; \mathsf{teaches}(x, y), \mathsf{Course}(y)$$

$$\mathsf{Professor} \sqsubseteq \exists\mathsf{teaches}$$
$$\text{as a logic rule:} \quad \mathsf{teaches}(z_1, z_2) \;\leftarrow\; \mathsf{Professor}(z_1)$$

The PI above does not apply to the atom $\mathsf{teaches}(x, y)$.

Conversely, the PI

$$\exists\mathsf{teaches}^- \sqsubseteq \mathsf{Course}$$
$$\text{as a logic rule:} \quad \mathsf{Course}(z_2) \;\leftarrow\; \mathsf{teaches}(z_1, z_2)$$

applies to the atom $\mathsf{Course}(y)$.

We add to the perfect rewriting the query $(\sigma = \{z_2/y\})$

$$\mathsf{q}(x) \;\leftarrow\; \mathsf{teaches}(x, y), \mathsf{teaches}(z_1, y)$$

Query answering in *DL-Lite*$_\mathcal{A}$: Query rewriting (cont'd)

An analogous behavior with join variables

$$\mathsf{q}(x) \;\leftarrow\; \mathsf{teaches}(x,y), \mathsf{Course}(y)$$

$$\begin{array}{rl} & \mathsf{Professor} \sqsubseteq \exists \mathsf{teaches} \\ \text{as a logic rule:} & \mathsf{teaches}(z_1,z_2) \;\leftarrow\; \mathsf{Professor}(z_1) \end{array}$$

The PI above does not apply to the atom $\mathsf{teaches}(x,y)$.

Conversely, the PI

$$\begin{array}{rl} & \exists \mathsf{teaches}^- \sqsubseteq \mathsf{Course} \\ \text{as a logic rule:} & \mathsf{Course}(z_2) \;\leftarrow\; \mathsf{teaches}(z_1,z_2) \end{array}$$

applies to the atom $\mathsf{Course}(y)$.

We add to the perfect rewriting the query ($\sigma = \{z_2/y\}$)

$$\mathsf{q}(x) \;\leftarrow\; \mathsf{teaches}(x,y), \mathsf{teaches}(z_1,y)$$

Query answering in *DL-Lite*$_\mathcal{A}$: Query rewriting (cont'd)

We now have the query

$$\mathsf{q}(x) \;\leftarrow\; \mathsf{teaches}(x,y), \mathsf{teaches}(z,y)$$

The PI

$$\begin{array}{rl} & \mathsf{Professor} \sqsubseteq \exists \mathsf{teaches} \\ \text{as a logic rule:} & \mathsf{teaches}(z_1,z_2) \;\leftarrow\; \mathsf{Professor}(z_1) \end{array}$$

does not apply to $\mathsf{teaches}(x,y)$ nor $\mathsf{teaches}(z,y)$, since y is a join variable.

However, we can transform the above query by unifying the atoms $\mathsf{teaches}(x,y)$, $\mathsf{teaches}(z_1,y)$. This rewriting step is called **reduce**, and produces the following query

$$\mathsf{q}(x) \;\leftarrow\; \mathsf{teaches}(x,y)$$

We can now apply the PI above ($sigma\{z_1/x, z_2/y\}$), and add to the reformulation the query

$$\mathsf{q}(x) \;\leftarrow\; \mathsf{Professor}(x)$$

Query answering in *DL-Lite*$_{\mathcal{A}}$: Query rewriting (cont'd)

We now have the query

$$\text{q}(x) \ \leftarrow\ \text{teaches}(x,y), \text{teaches}(z,y)$$

The PI $\quad$ Professor $\sqsubseteq \exists$teaches

as a logic rule: $\quad$ $\text{teaches}(z_1,z_2) \ \leftarrow\ \text{Professor}(z_1)$

does not apply to $\text{teaches}(x,y)$ nor $\text{teaches}(z,y)$, since y is a join variable.

However, we can transform the above query by unifying the atoms $\text{teaches}(x,y)$, $\text{teaches}(z_1,y)$. This rewriting step is called **reduce**, and produces the following query

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

We can now apply the PI above $(sigma\{z_1/x, z_2/y\})$, and add to the reformulation the query

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

Answering by rewriting in *DL-Lite*$_{\mathcal{A}}$: The algorithm

1. Rewrite the CQ q into a UCQs: apply to q in all possible ways the PIs in the TBox $\mathcal{T}$.
2. This corresponds to exploiting ISAs, role typings, and mandatory participations to obtain new queries that could contribute to the answer.
3. Unifying atoms can make applicable rules that could not be applied otherwise.
4. The UCQs resulting from this process is the **perfect rewriting** $r_{q,\mathcal{T}}$.
5. $r_{q,\mathcal{T}}$ is then **encoded into SQL** and evaluated over $\mathcal{A}$ managed in **secondary storage via a RDBMS**, to return the set $cert(q, \mathcal{O})$.

Query answering in $\textit{DL-Lite}_{\mathcal{A}}$: Example

TBox: $\text{Professor} \sqsubseteq \exists\text{teaches}$
$\exists\text{teaches}^- \sqsubseteq \text{Course}$

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

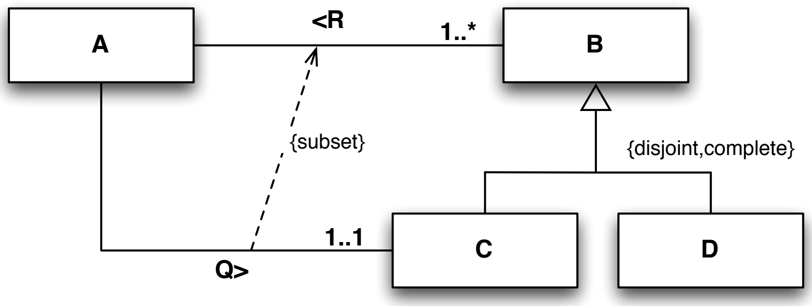
Perfect Rewriting: $\text{q}(x) \leftarrow \text{teaches}(x, y), \text{Course}(y)$
$\text{q}(x) \leftarrow \text{teaches}(x, y), \text{teaches}(z, y)$
$\text{q}(x) \leftarrow \text{teaches}(x, z)$
$\text{q}(x) \leftarrow \text{Professor}(x)$

ABox: $\text{teaches}(\texttt{John}, \texttt{databases})$
$\text{Professor}(\texttt{Mary})$

It is easy to see that the evaluation of $r_{q,\mathcal{T}}$ over $\mathcal{A}$ in this case produces the set $\{\texttt{John}, \texttt{Mary}\}$.

Example 1

Express in $\textit{DL-Lite}_{\mathcal{A}}$ the following ontology:



Considering the following ABox $\mathcal{A} = \{A(a)\}$ compute the answer to the following queries:

$$\begin{aligned} q(x) &\leftarrow Q(x, y), R(y, z). \\ q'() &\leftarrow B(x). \end{aligned}$$

Example 1 (solution)

Expansions:

$$
\begin{array}{lcll}
q(x) & \leftarrow & Q(x,y), R(y,z). & \\
q(x) & \leftarrow & Q(x,y), Q(z,y). & Q \sqsubseteq R^- \\
q(x) & \leftarrow & Q(x,y). & \text{unify: } z = x \\
q(x) & \leftarrow & A(x). & A \sqsubseteq \exists Q \\
 & & & \Longrightarrow \text{ answer } x = a
\end{array}
$$

$$
\begin{array}{lcll}
q'() & \leftarrow & B(x). & \\
q'() & \leftarrow & R(x,y). & \exists R. \sqsubseteq B \\
q'() & \leftarrow & A(y). & A \sqsubseteq \exists R^- \\
 & & & \Longrightarrow \text{ answer } \mathit{true} \text{ (by } y = a)
\end{array}
$$

Example 1 (solution)

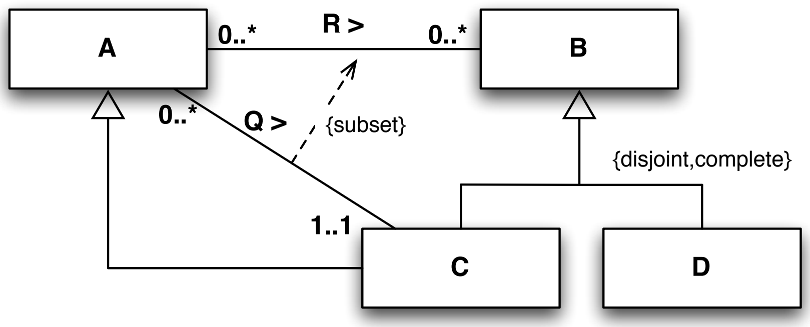
Expansions:

$$
\begin{array}{lcll}
q(x) & \leftarrow & Q(x,y), R(y,z). & \\
q(x) & \leftarrow & Q(x,y), Q(z,y). & Q \sqsubseteq R^- \\
q(x) & \leftarrow & Q(x,y). & \text{unify: } z = x \\
q(x) & \leftarrow & A(x). & A \sqsubseteq \exists Q \\
 & & & \Longrightarrow \text{ answer } x = a
\end{array}
$$

$$
\begin{array}{lcll}
q'() & \leftarrow & B(x). & \\
q'() & \leftarrow & R(x,y). & \exists R. \sqsubseteq B \\
q'() & \leftarrow & A(y). & A \sqsubseteq \exists R^- \\
 & & & \Longrightarrow \text{ answer } \mathit{true} \text{ (by } y = a)
\end{array}
$$

Example 2

Express in *DL-Lite*$_{\mathcal{A}}$ the following ontology:



Considering the following ABox $\mathcal{A} = \{Q(a,b), R(b,b), C(c)\}$ compute the answer to the following queries:

$$q(x) \quad \leftarrow \quad R(x,y), R(y,z), A(z).$$

Example 2 (solution)

Expansions:

```
q(x) :- R(x,y), R(y,z), A(z).
q(x) :- R(x,x), A(x).       --- unify
q(x) :- R(x,x), R(x,y).     --- Exists R ISA A
q(x) :- R(x,x).             --- unify

  answer x = b

......
```

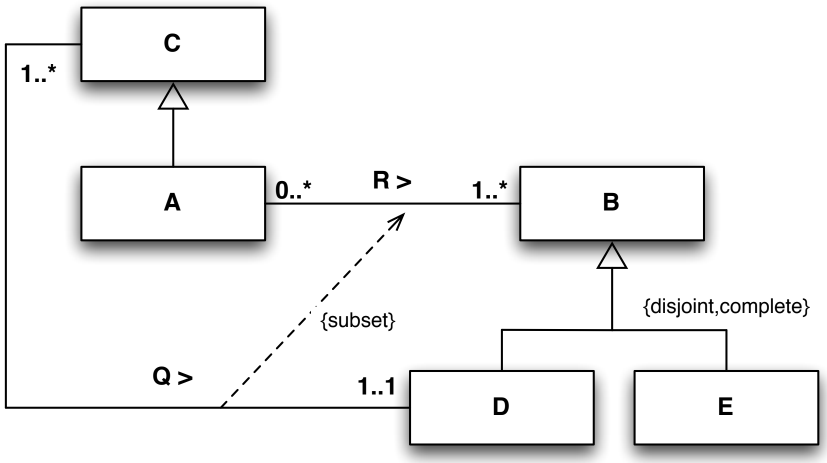
Example 2 (solution)

Expansions:

```
.....

q(x) :- R(x,y), R(y,z), A(z).
q(x) :- R(x,y), R(y,z), C(z).    --- C ISA A
q(x) :- R(x,y), R(y,z), Q(w,z). --- Exists Q- ISA C
q(x) :- R(x,y), Q(y,z), Q(w,z). --- Q ISA R
q(x) :- R(x,y), Q(y,z).          --- unify
q(x) :- R(x,y), A(y).            --- A ISA Exists Q
q(x) :- R(x,y), C(y).            --- C ISA A
q(x) :- R(x,y), Q(z,y).          --- Exists Q- ISA C
q(x) :- Q(x,y), Q(z,y).          --- Q ISA R
q(x) :- Q(x,y).                  --- unify

   answer x = a

q(x) :- A(x).                    --- A ISA Exists Q
q(x) :- C(x).                    --- C ISA A

   answer x = c
```

Example 3

Express in *DL-Lite*$_\mathcal{A}$ the following ontology:



Considering the following ABox $\mathcal{A} = \{C(a)\}$ compute the answer to the following queries:

$$\begin{aligned} q(x) &\leftarrow R(x,y), B(y). \\ q'(x) &\leftarrow A(x). \end{aligned}$$

Can we simplify the diagram?

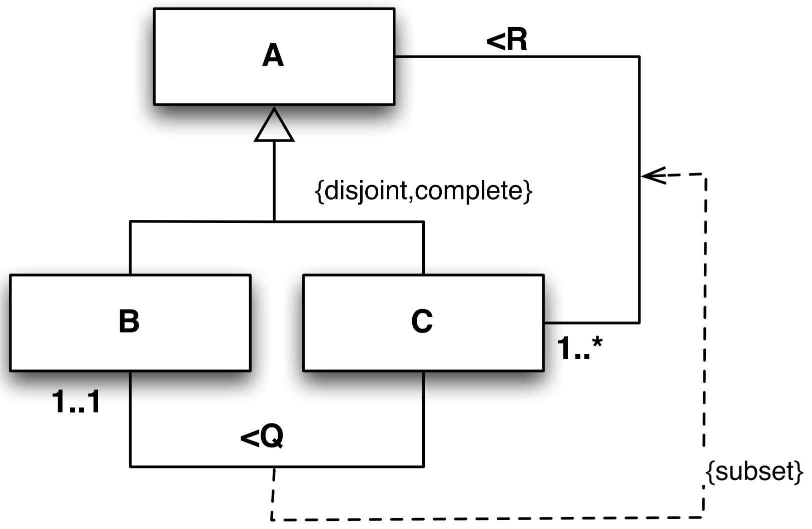
Example 3 (solution)

Expansions:

```
q(x) :- R(x,y), B(y).
q(x) :- R(x,y), D(y).   --- D ISA B
q(x) :- R(x,y), Q(z,y). --- Exists Q- ISA D
q(x) :- Q(x,y), Q(z,y). --- Q ISA R
q(x) :- Q(x,y).         --- unify
q(x) :- C(x).           --- C ISA Exists Q

   answer x = a


q'(x):- A(x).
q'(x):- R(x,y).   --- A ISA Exists R
q'(x):- Q(x,y).   --- Q ISA R
q'(x):- C(x).     --- C ISA Exists Q

   answer x = a
```

Example 4

Express in *DL-Lite*$_{\mathcal{A}}$ the following ontology:



Considering the following ABox $\mathcal{A} = \{B(b)\}$ compute the answer to the following queries:

$$\begin{aligned} q(z) &\leftarrow R(x,y), R(y,z). \\ q'() &\leftarrow C(x). \end{aligned}$$

Example 4 (solution)

Expansions:

```
q(z) :- R(x,y), R(y,z).
q(z) :- A(y), R(y,z).   --- A ISA Exists R-
q(z) :- C(y), R(y,z).   --- C ISA A
q(z) :- R(y,w), R(y,z). --- Exists R ISA C
q(z) :- R(y,z).         --- unify
q(z) :- A(z).           --- A ISA Exists R-
q(z) :- B(z).           --- B ISA A

   answer z = b


q'() :- C(x).
q'() :- R(x,y).   -- Exists R ISA C
q'() :- A(y).     -- A ISA Exists R-
q'() :- B(y).     -- B ISA A

   answer z = b
```

Complexity of reasoning in *DL-Lite*$_{\mathcal{A}}$

Ontology satisfiability and all classical DL reasoning tasks are:

- Efficiently tractable in the size of TBox (i.e., PTIME).
- Very efficiently tractable in the size of the ABox (i.e., LOGSPACE).

In fact, reasoning can be done by constructing suitable FOL/SQL queries and evaluating them over the ABox (**FOL-rewritability**).

Query answering for CQs and UCQs is:

- PTIME in the size of TBox.
- LOGSPACE in the size of the ABox.
- Exponential in the size of the query (NP-complete).
  Bad? ...not really, this is exactly as in relational DBs.

Can we go beyond *DL-Lite*$_{\mathcal{A}}$?

By adding essentially any other DL construct, e.g., union ($\sqcup$), value restriction ($\forall R.C$), etc., without some limitations we lose these nice computational properties (see later).

Complexity of reasoning in *DL-Lite*$_{\mathcal{A}}$

Ontology satisfiability and all classical DL reasoning tasks are:

- Efficiently tractable in the size of TBox (i.e., PTime).
- Very efficiently tractable in the size of the ABox (i.e., LogSpace).

In fact, reasoning can be done by constructing suitable FOL/SQL queries and evaluating them over the ABox (**FOL-rewritability**).

Query answering for CQs and UCQs is:

- PTime in the size of TBox.
- LogSpace in the size of the ABox.
- Exponential in the size of the query (NP-complete).
  Bad? ... not really, this is exactly as in relational DBs.

Can we go beyond *DL-Lite*$_{\mathcal{A}}$?

By adding essentially any other DL construct, e.g., union ($\sqcup$), value restriction ($\forall R.C$), etc., without some limitations we lose these nice computational properties (see later).

Complexity of reasoning in *DL-Lite*$_{\mathcal{A}}$

Ontology satisfiability and all classical DL reasoning tasks are:

- Efficiently tractable in the size of TBox (i.e., PTime).
- Very efficiently tractable in the size of the ABox (i.e., LogSpace).

In fact, reasoning can be done by constructing suitable FOL/SQL queries and evaluating them over the ABox (**FOL-rewritability**).

Query answering for CQs and UCQs is:

- PTime in the size of TBox.
- LogSpace in the size of the ABox.
- Exponential in the size of the query (NP-complete).
  Bad? ... not really, this is exactly as in relational DBs.

Can we go beyond *DL-Lite*$_{\mathcal{A}}$?

By adding essentially any other DL construct, e.g., union ($\sqcup$), value restriction ($\forall R.C$), etc., without some limitations we lose these nice computational properties (see later).

Complexity of reasoning in *DL-Lite*$_{\mathcal{A}}$

Ontology satisfiability and all classical DL reasoning tasks are:

- Efficiently tractable in the size of TBox (i.e., PTIME).
- Very efficiently tractable in the size of the ABox (i.e., LOGSPACE).

In fact, reasoning can be done by constructing suitable FOL/SQL queries and evaluating them over the ABox (**FOL-rewritability**).

Query answering for CQs and UCQs is:

- PTIME in the size of TBox.
- LOGSPACE in the size of the ABox.
- Exponential in the size of the query (NP-complete).
  Bad? . . . not really, this is exactly as in relational DBs.

Can we go beyond *DL-Lite*$_{\mathcal{A}}$?

By adding essentially any other DL construct, e.g., union ($\sqcup$), value restriction ($\forall R.C$), etc., without some limitations we lose these nice computational properties (see later).

Beyond *DL-Lite*$_{\mathcal{A}}$: results on data complexity

	lhs	rhs	funct.	Prop. incl.	Data complexity of query answering
0	*DL-Lite*$_{\mathcal{A}}$		$\surd$*	$\surd$*	in LOGSPACE
1	$A \mid \exists P.A$	A	–	–	NLOGSPACE-hard
2	A	$A \mid \forall P.A$	–	–	NLOGSPACE-hard
3	A	$A \mid \exists P.A$	$\surd$	–	NLOGSPACE-hard
4	$A \mid \exists P.A \mid A_1 \sqcap A_2$	A	–	–	PTIME-hard
5	$A \mid A_1 \sqcap A_2$	$A \mid \forall P.A$	–	–	PTIME-hard
6	$A \mid A_1 \sqcap A_2$	$A \mid \exists P.A$	$\surd$	–	PTIME-hard
7	$A \mid \exists P.A \mid \exists P^-.A$	$A \mid \exists P$	–	–	PTIME-hard
8	$A \mid \exists P \mid \exists P^-$	$A \mid \exists P \mid \exists P^-$	$\surd$	$\surd$	PTIME-hard
9	$A \mid \neg A$	A	–	–	**coNP-hard**
10	A	$A \mid A_1 \sqcup A_2$	–	–	**coNP-hard**
11	$A \mid \forall P.A$	A	–	–	**coNP-hard**

Notes:

- * with the "proviso" of not specializing functional properties.
- NLOGSPACE and PTIME hardness holds already for instance checking.
- For coNP-hardness in line 10, a TBox with a single assertion $A_L \sqsubseteq A_T \sqcup A_F$ suffices! $\leadsto$ **No** hope of including **covering constraints**.

Beyond union of conjunctive queries

Till now we have assumed that the client queries are UCQs (aka positive queries).
Can we go beyond UCQ? Can we go to full **FOL/SQL queries**?

- No! Answering FOL queries in presence of incomplete information is undecidable: Consider an empty source (no data), still a (boolean) FOL query may return *true* because it is valid! (FOL validity is undecidable)
- Yes! With some compromises:
  Query what the ontology **knows** about the domain, not what is **true** in the domain!
  On knowledge we have complete information, so evaluating FOL queries is LogSpace.

Beyond union of conjunctive queries

Till now we have assumed that the client queries are UCQs (aka positive queries).
Can we go beyond UCQ? Can we go to full **FOL/SQL queries**?

- No! Answering FOL queries in presence of incomplete information is undecidable: Consider an empty source (no data), still a (boolean) FOL query may return *true* because it is valid! (FOL validity is undecidable)
- Yes! With some compromises:
  Query what the ontology **knows** about the domain, not what is **true** in the domain!
  On knowledge we have complete information, so evaluating FOL queries is LogSpace.

Beyond union of conjunctive queries

Till now we have assumed that the client queries are UCQs (aka positive queries).
Can we go beyond UCQ? Can we go to full **FOL/SQL queries**?

- No! Answering FOL queries in presence of incomplete information is undecidable: Consider an empty source (no data), still a (boolean) FOL query may return *true* because it is valid! (FOL validity is undecidable)
- Yes! With some compromises:
  Query what the ontology **knows** about the domain, not what is **true** in the domain!
  On knowledge we have complete information, so evaluating FOL queries is LOGSPACE.

Beyond union of conjunctive queries

Till now we have assumed that the client queries are UCQs (aka positive queries).
Can we go beyond UCQ? Can we go to full **FOL/SQL queries**?

- No! Answering FOL queries in presence of incomplete information is undecidable: Consider an empty source (no data), still a (boolean) FOL query may return *true* because it is valid! (FOL validity is undecidable)
- Yes! With some compromises:
  Query what the ontology **knows** about the domain, not what is **true** in the domain!
  On knowledge we have complete information, so evaluating FOL queries is LOGSPACE.

SparSQL

Full **SQL**, but with relations in the FROM clause that are UCQs, expressed in **SPARQL**, over the ontology.

- **SPARQL** queries are used to query what is **true** in the domain.
- **SQL** is used to query what the ontology **knows** about the domain.

Example: negation

Return all known people that are neither known to be male nor known to be female.

```
SELECT persons.x
FROM SparqlTable(SELECT ?x
                 WHERE {?x rdf:type 'Person'}
                ) persons
EXCEPT  (
SELECT males.x
FROM SparqlTable(SELECT ?x
                 WHERE {?x rdf:type 'Male'}
                ) males
UNION
SELECT females.x
FROM SparqlTable(SELECT ?x
                 WHERE {?x rdf:type 'Female'}
                ) females
)
```

Example: aggregates

Return the people and the number of their known spouses, but only if they are known to be married to at least two people.

```
SELECT marriage.x, count(marriage.y)
FROM SparqlTable(SELECT ?x ?y
                 WHERE {?x :MarriedTo ?y}
                ) marriage
GROUP BY marriage.x
HAVING count(marriage.y) >= 2
```

SparSQL in *DL-Lite*$_{\mathcal{A}}$

Answering of SparSQL queries in *DL-Lite*$_{\mathcal{A}}$:

1. Expand and unfold the UCQs (in the SparqlTables) as usual in *DL-Lite*$_{\mathcal{A}}$ $\rightsquigarrow$ an SQL query over the ABox (seen as a database) for each SparqlTable in the FROM clauses.
2. Substitute SparqlTables with the new SQL queries. $\rightsquigarrow$ the result is again an SQL query over the ABox (seen as a database)!
3. Evaluate the resulting SQL query over the ABox (seen as a database)