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?

Query answering in description logics: *DL-Lit* (3/56)

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)

Query answering in description logics: *DL-Lit* (4/56)

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.

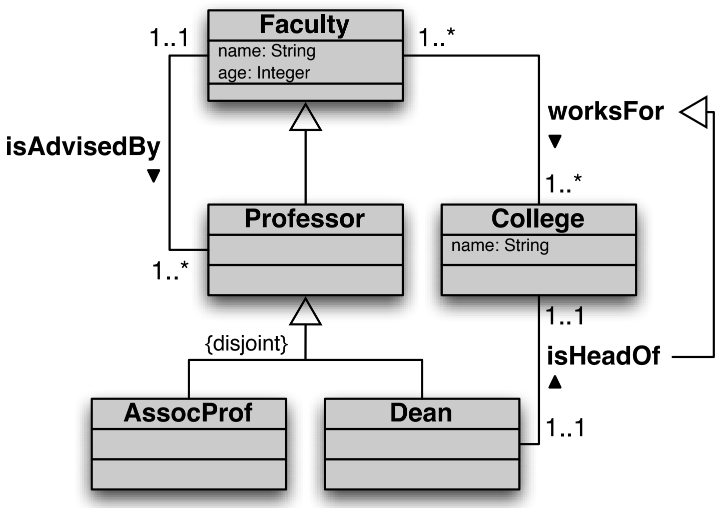
Example of conjunctive query

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

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

Example of conjunctive query

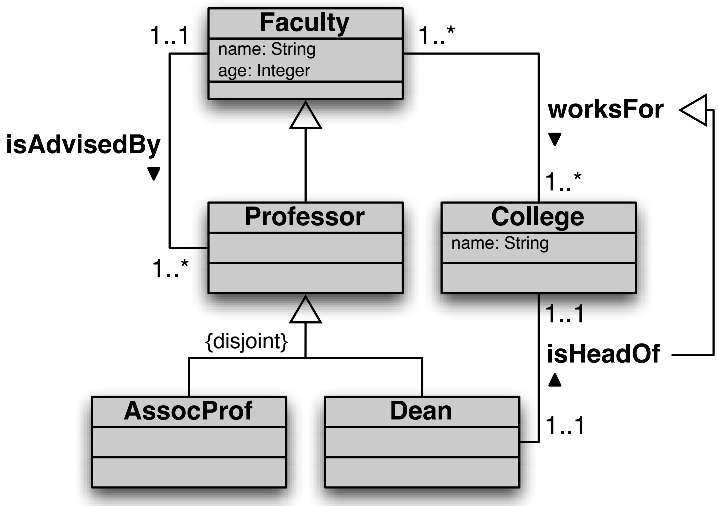
$$
\begin{array}{rcl}
\text{Professor} & \sqsubseteq & \text{Faculty} \\
\text{AssocProf} & \sqsubseteq & \text{Professor} \\
\text{Dean} & \sqsubseteq & \text{Professor} \\
\text{AssocProf} & \sqsubseteq & \neg\text{Dean} \\
\text{Faculty} & \sqsubseteq & \exists\text{age} \\
\exists\text{age}^- & \sqsubseteq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



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

Example of conjunctive query

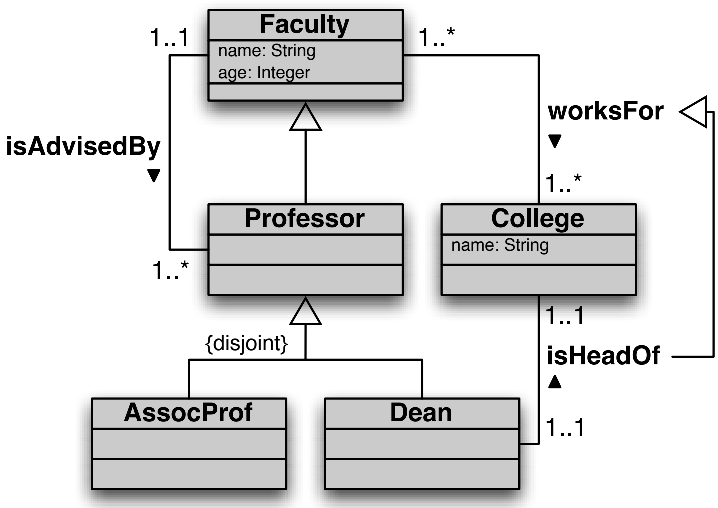
$$
\begin{array}{rcl}
\text{Professor} & \sqsubseteq & \text{Faculty} \\
\text{AssocProf} & \sqsubseteq & \text{Professor} \\
\text{Dean} & \sqsubseteq & \text{Professor} \\
\text{AssocProf} & \sqsubseteq & \neg\text{Dean} \\
\text{Faculty} & \sqsubseteq & \exists\text{age} \\
\exists\text{age}^{-} &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



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

Example of conjunctive query

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



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

Conjunctive queries and SQL – Example

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

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

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**

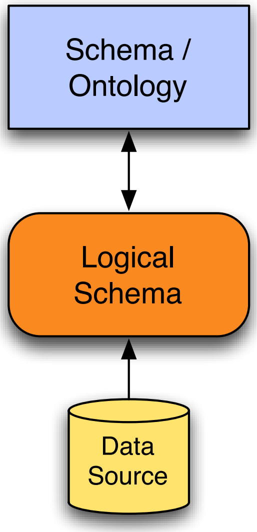
Note: for the moment we assume to deal with an ordinary ABox, which however may be very large and thus is stored in a database.

Query answering under the database assumption

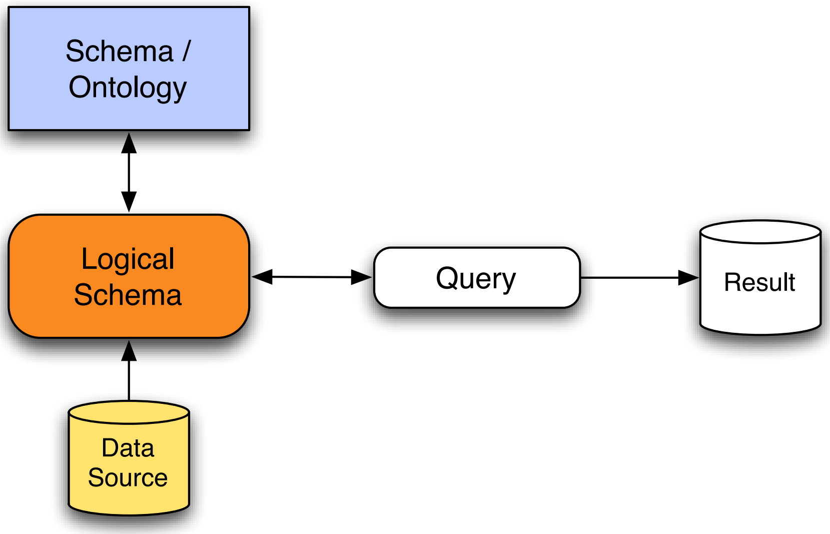
- Data are completely specified (CWA), and typically large.
- Schema/intensional information used in the design phase.
- At **runtime**, the data is assumed to satisfy the schema, and therefore the **schema is not used**.
- Queries allow for complex navigation paths in the data (cf. SQL).

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

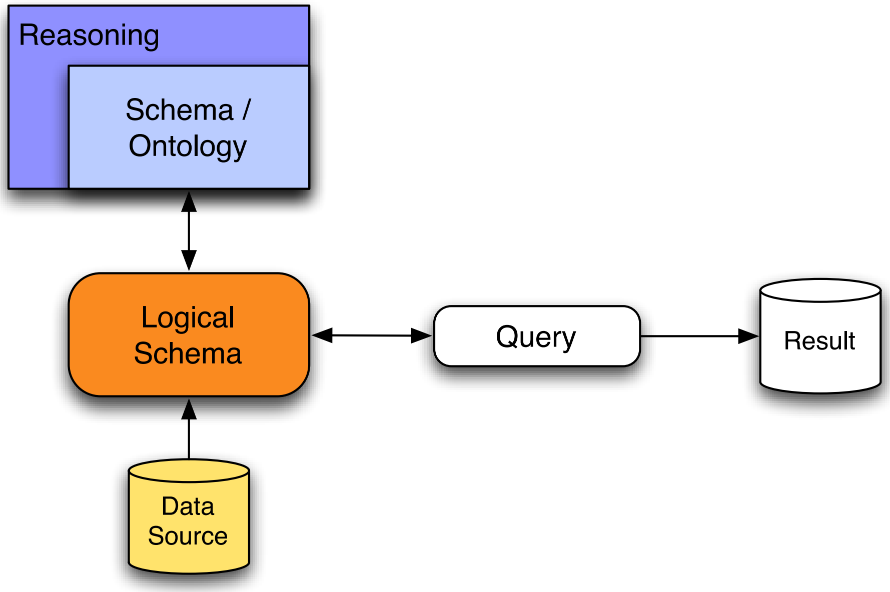
Query answering under the database assumption (cont'd)



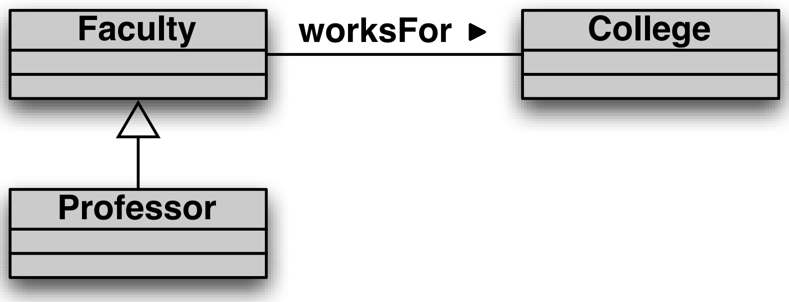
Query answering under the database assumption (cont'd)



Query answering under the database assumption (cont'd)



Query answering under the database assumption – Example



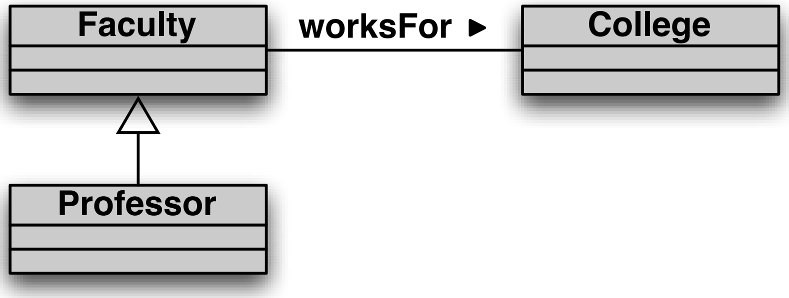
For each class/property we have a (complete) table in the database.

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

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

Answer: ???

Query answering under the database assumption – Example



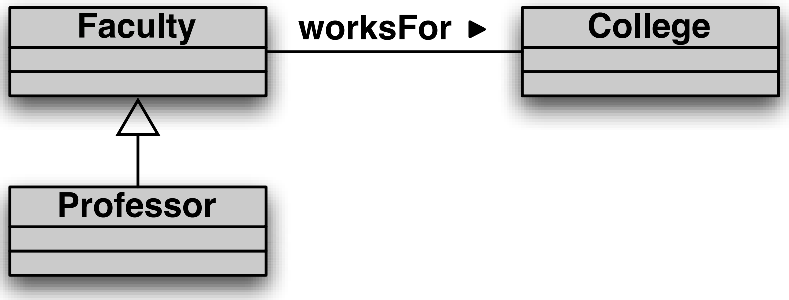
For each class/property we have a (complete) table in the database.

DB: Faculty = { john, mary, nick }
Professor = { john, nick }
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, nick }
Professor = { john, nick }
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

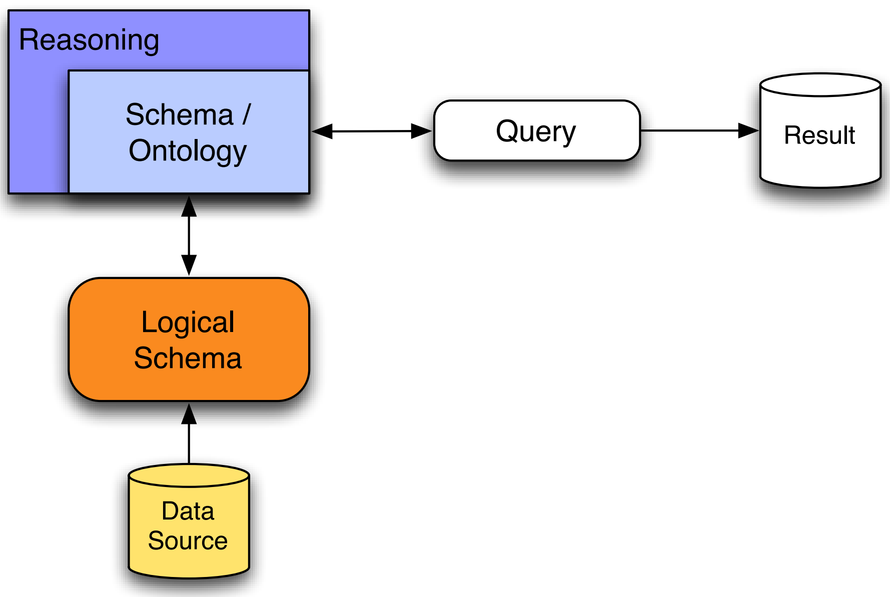
- An ontology imposes constraints on the data.
- Actual data may be incomplete or inconsistent w.r.t. such constraints.
- The system has to take into account the constraints during query answering, and overcome incompleteness or inconsistency.

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

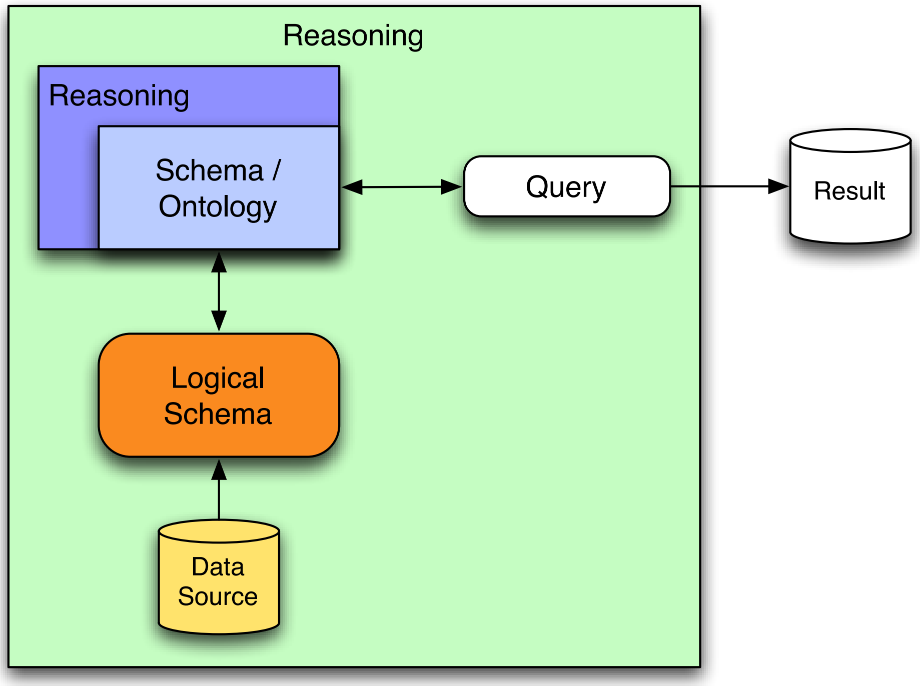
Note:

- 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.

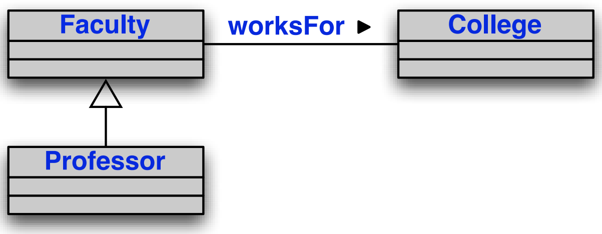
Query answering under the KR assumption (cont'd)



Query answering under the KR assumption (cont'd)



Query answering under the KR assumption – Example



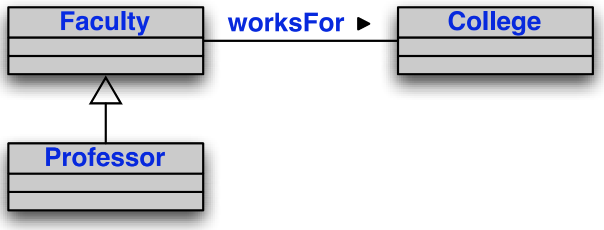
The tables in the database may be **incompletely specified**, or even missing for some classes/properties.

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

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

Answer: ???

Query answering under the KR assumption – Example



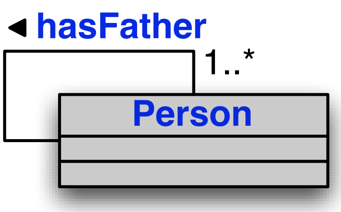
The tables in the database may be **incompletely specified**, or even missing for some classes/properties.

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

Query: $q(x) \leftarrow \text{Faculty}(x)$

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

Query answering under the KR assumption – Example 2



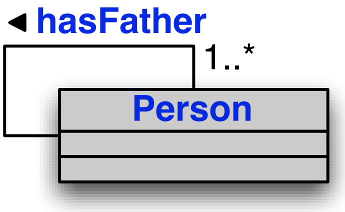
Each person has a father, who is a person.

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

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

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

Query answering under the KR assumption – Example 2



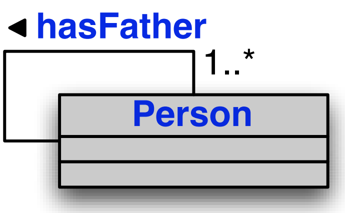
Each person has a father, who is a person.

DB: Person $\supseteq$ { `john`, `nick`, `toni` }
hasFather $\supseteq$ { (`john`,`nick`), (`nick`,`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`,`nick`), (`nick`,`toni`) }
to q_2: ???
to q_3: ???
to q_4: ???

Query answering under the KR assumption – Example 2



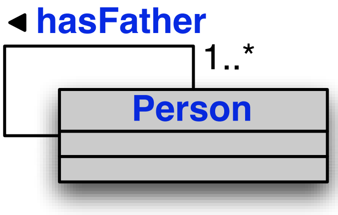
Each person has a father, who is a person.

DB: Person $\supseteq$ { `john`, `nick`, `toni` }
hasFather $\supseteq$ { (`john`,`nick`), (`nick`,`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`,`nick`), (`nick`,`toni`) }
to q_2: { `john`, `nick`, `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, nick, toni }
hasFather $\supseteq$ { (john,nick), (nick,toni) }

Queries: $q_1(x, y) \leftarrow \text{hasFather}(x, y)$

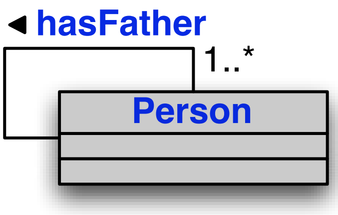
$q_2(x) \leftarrow \exists y.\, \text{hasFather}(x, y)$

$q_3(x) \leftarrow \exists y_1, y_2, y_3.\, \text{hasFather}(x, y_1), \text{hasFather}(y_1, y_2), \text{hasFather}(y_2, y_3)$

$q_4(x, y_3) \leftarrow \exists y_1, y_2.\, \text{hasFather}(x, y_1), \text{hasFather}(y_1, y_2), \text{hasFather}(y_2, y_3)$

Answers: to q_1: { (john,nick), (nick,toni) }
to q_2: { john, nick, toni }
to q_3: { john, nick, 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, nick, toni }
hasFather $\supseteq$ { (john,nick), (nick,toni) }

Queries: $q_1(x, y) \leftarrow \text{hasFather}(x, y)$

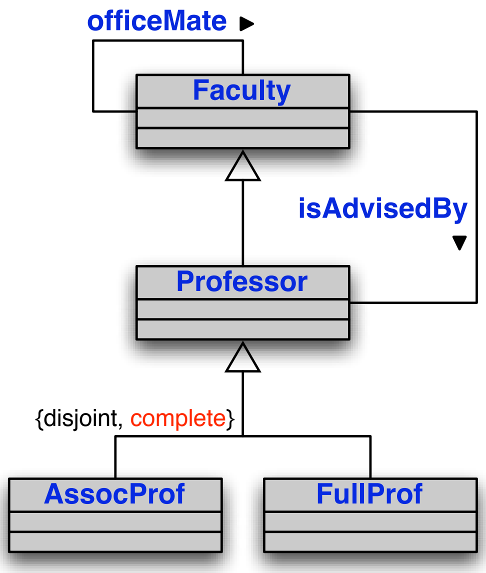
$q_2(x) \leftarrow \exists y.\, \text{hasFather}(x, y)$

$q_3(x) \leftarrow \exists y_1, y_2, y_3.\, \text{hasFather}(x, y_1), \text{hasFather}(y_1, y_2), \text{hasFather}(y_2, y_3)$

$q_4(x, y_3) \leftarrow \exists y_1, y_2.\, \text{hasFather}(x, y_1), \text{hasFather}(y_1, y_2), \text{hasFather}(y_2, y_3)$

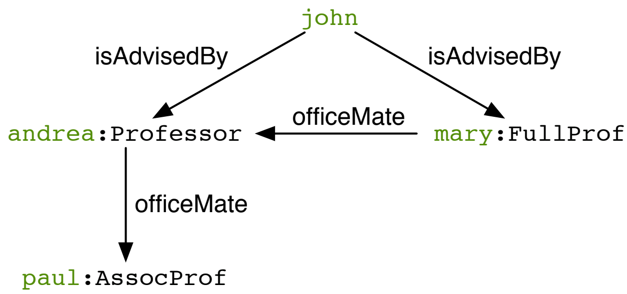
Answers: to q_1: { (john,nick), (nick,toni) }
to q_2: { john, nick, toni }
to q_3: { john, nick, toni }
to q_4: { }

QA under the KR assumption – Andrea's Example

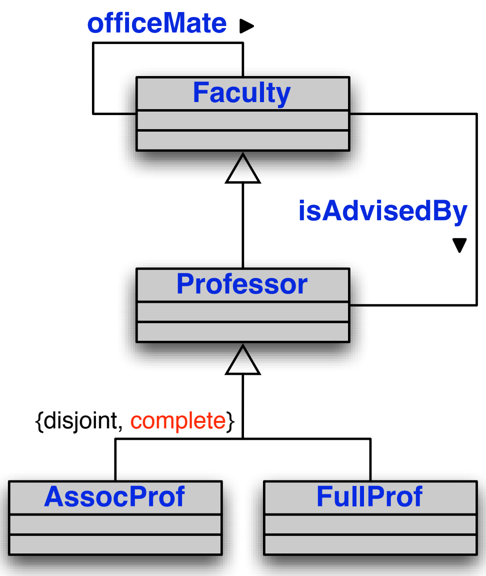


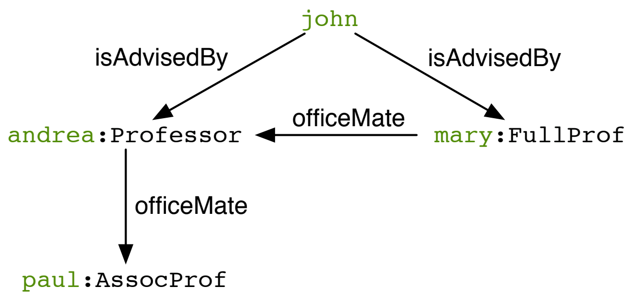
$$\text{Professor} \equiv \text{AssocProf} \sqcup \text{FullProf}$$

$$\begin{aligned}
\text{Faculty} &\supseteq \{\ \texttt{andrea}, \texttt{nick}, \texttt{mary}, \texttt{john}\ \} \\
\text{Professor} &\supseteq \{\ \texttt{andrea}, \texttt{nick}, \texttt{mary}\ \} \\
\text{AssocProf} &\supseteq \{\ \texttt{nick}\ \} \\
\text{FullProf} &\supseteq \{\ \texttt{mary}\ \} \\
\text{isAdvisedBy} &\supseteq \{\ (\texttt{john},\texttt{andrea}), (\texttt{john},\texttt{mary})\ \} \\
\text{officeMate} &\supseteq \{\ (\texttt{mary},\texttt{andrea}), (\texttt{andrea},\texttt{nick})\ \}
\end{aligned}$$



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

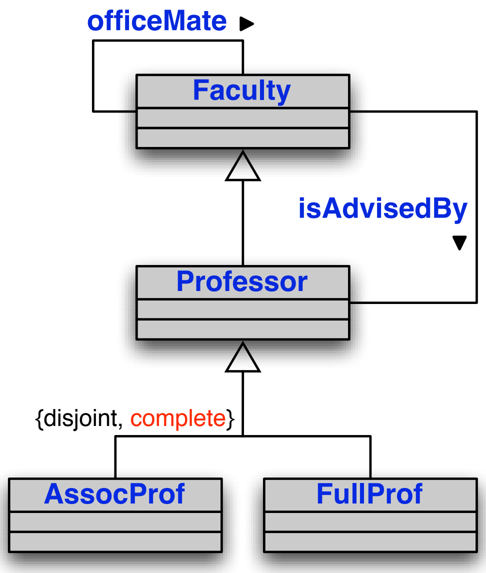


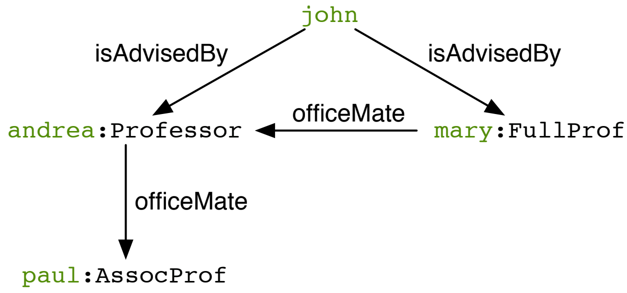


$$q() \leftarrow \exists y, z.\ \text{isAdvisedBy}(\texttt{john}, y),\ \text{FullProf}(y),\ \text{officeMate}(y, z),\ \text{AssocProf}(z)$$

Answer: yes or no?

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

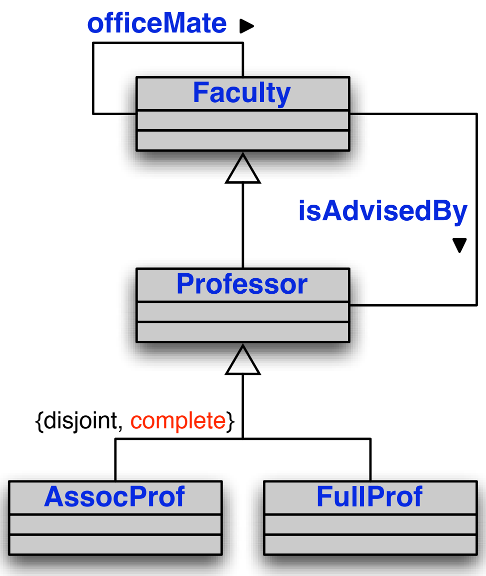


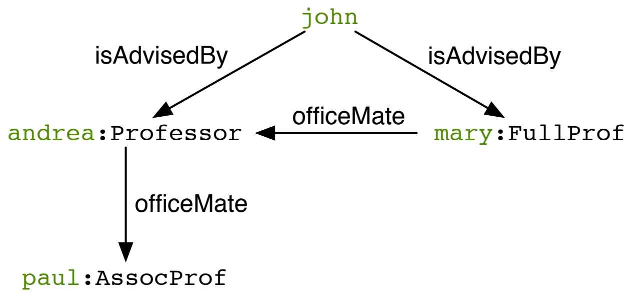


$$q() \leftarrow \exists y, z.\ \mathsf{isAdvisedBy}(\mathtt{john}, y),\ \mathsf{FullProf}(y),\ \mathsf{officeMate}(y, z),\ \mathsf{AssocProf}(z)$$

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 $\mathcal{O}$.

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 $cert(q, \mathcal{O})$

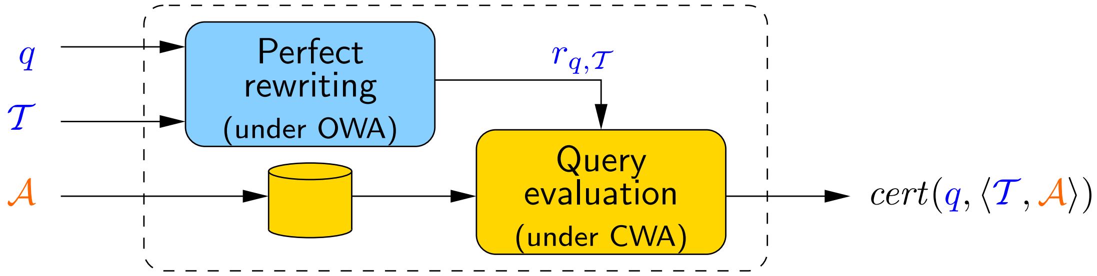
. . . are the **tuples $\vec{c}$ of constants of $\mathcal{A}$** such that $\vec{c} \in q^{\mathcal{I}}$, for every model $\mathcal{I}$ of $\mathcal{O}$.

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

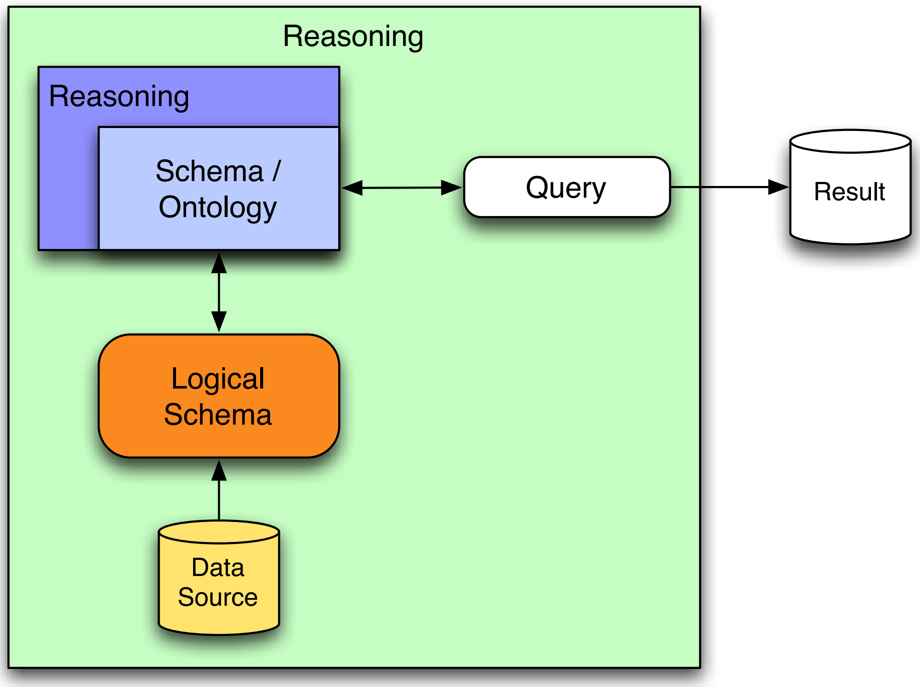


Query answering can **always** be thought as done in two phases:

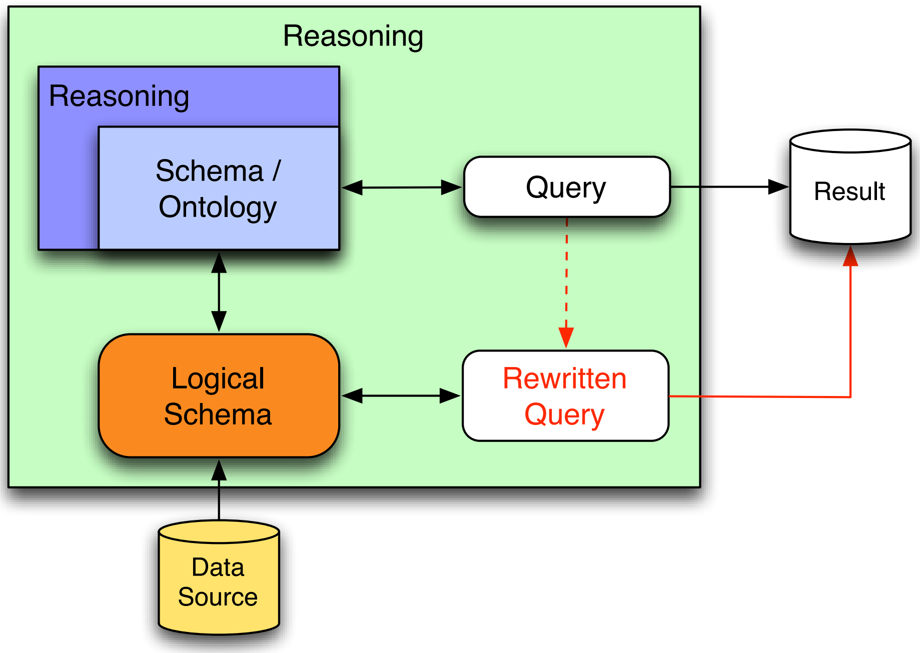
1. **Perfect rewriting**: produce from q and the TBox $\mathcal{T}$ a new query $r_{q,\mathcal{T}}$ (called the perfect rewriting of q w.r.t. $\mathcal{T}$).
2. **Query evaluation**: evaluate $r_{q,\mathcal{T}}$ over the ABox $\mathcal{A}$ seen as a complete database (and without considering the TBox $\mathcal{T}$).
   $\leadsto$ 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**.
  $\leadsto$ 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.
  $\leadsto$ Query evaluation requires (at least) linear recursion.
- When we can rewrite into a PTime-hard language.
  $\leadsto$ Query evaluation requires full recursion (e.g., Datalog).
- When we can rewrite into a coNP-hard language.
  $\leadsto$ 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.

Query answering in description logics: *DL-Lit* (26/56)

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**.
  ⇝ 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.
  ⇝ Query evaluation requires (at least) linear recursion.
- When we can rewrite into a PTime-hard language.
  ⇝ Query evaluation requires full recursion (e.g., Datalog).
- When we can rewrite into a coNP-hard language.
  ⇝ 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 $\textit{DL-Lite}_{\mathcal{A}}$, an expressive member of the *DL-Lite* family.

$\textit{DL-Lite}_{\mathcal{A}}$ 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 $\textit{DL-Lite}_{\mathcal{A}}$, an expressive member of the *DL-Lite* family.

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

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

TBox assertions:

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

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

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

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

- 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).

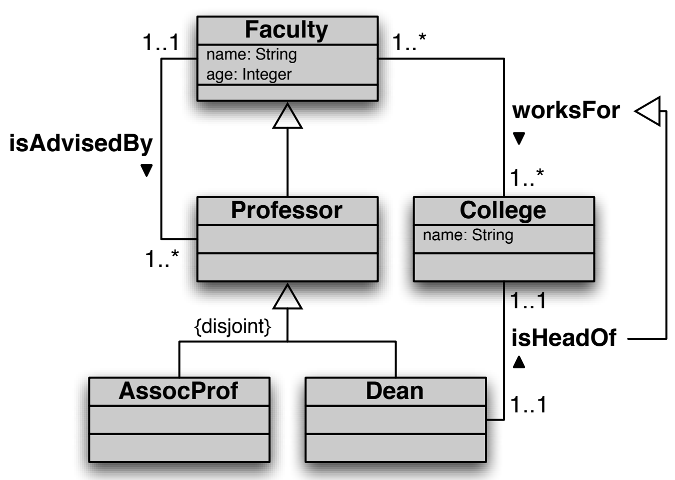
Semantics of the *DL-Lite*$_{\mathcal{A}}$ assertions

Assertion	Syntax	Example	Semantics
class incl.	$B \sqsubseteq C$	Father $\sqsubseteq \exists$child	$B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$
o-prop. incl.	$Q \sqsubseteq R$	father $\sqsubseteq$ anc	$Q^{\mathcal{I}} \subseteq R^{\mathcal{I}}$
v.dom. incl.	$E \sqsubseteq F$	ρ(age) $\sqsubseteq$ xsd:int	$E^{\mathcal{I}} \subseteq F^{\mathcal{I}}$
d-prop. incl.	$U \sqsubseteq V$	offPhone $\sqsubseteq$ phone	$U^{\mathcal{I}} \subseteq V^{\mathcal{I}}$
o-prop. funct.	$(\textbf{funct}\ Q)$	(**funct** father)	$\forall o, o, o''.(o, o') \in Q^{\mathcal{I}} \wedge (o, o'') \in Q^{\mathcal{I}} \rightarrow o' = o''$
d-prop. funct.	$(\textbf{funct}\ U)$	(**funct** ssn)	$\forall o, v, v'.(o, v) \in U^{\mathcal{I}} \wedge (o, v') \in U^{\mathcal{I}} \rightarrow v = v'$
mem. asser.	$A(c)$	Father(bob)	$c^{\mathcal{I}} \in A^{\mathcal{I}}$
mem. asser.	$P(c_1, c_2)$	child(bob, ann)	$(c_1^{\mathcal{I}}, c_2^{\mathcal{I}}) \in P^{\mathcal{I}}$
mem. asser.	$U(c, d)$	phone(bob, '2345')	$(c^{\mathcal{I}}, \mathit{val}(d)) \in U^{\mathcal{I}}$

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	$\exists P \sqsubseteq A_1 \qquad \exists P^- \sqsubseteq A_2$
Mandatory participation *(min card = 1)*	$A_1 \sqsubseteq \exists P \qquad A_2 \sqsubseteq \exists P^-$
Functionality of relations *(max card = 1)*	$(\textbf{funct}\ P) \qquad (\textbf{funct}\ P^-)$
ISA between properties	$Q_1 \sqsubseteq Q_2$
Disjointness between properties	$Q_1 \sqsubseteq \neg Q_2$

Example



$$
\begin{array}{rcl}
\text{Professor} & \sqsubseteq & \text{Faculty} \\
\text{AssocProf} & \sqsubseteq & \text{Professor} \\
\text{Dean} & \sqsubseteq & \text{Professor} \\
\text{AssocProf} & \sqsubseteq & \neg\text{Dean} \\
\\
\text{Faculty} & \sqsubseteq & \exists \textit{age} \\
\exists \textit{age}^- & \sqsubseteq & \texttt{xsd:int} \\
& (\textbf{funct}\ \textit{age}) & \\
\\
\exists\text{worksFor} & \sqsubseteq & \text{Faculty} \\
\exists\text{worksFor}^- & \sqsubseteq & \text{College} \\
\text{Faculty} & \sqsubseteq & \exists\text{worksFor} \\
\text{College} & \sqsubseteq & \exists\text{worksFor}^- \\
\\
\exists\text{isHeadOf} & \sqsubseteq & \text{Dean} \\
\exists\text{isHeadOf}^- & \sqsubseteq & \text{College} \\
\text{Dean} & \sqsubseteq & \exists\text{isHeadOf} \\
\text{College} & \sqsubseteq & \exists\text{isHeadOf}^- \\
\text{isHeadOf} & \sqsubseteq & \text{worksFor} \\
& (\textbf{funct}\ \text{isHeadOf}) & \\
& (\textbf{funct}\ \text{isHeadOf}^-) & \\
& \vdots &
\end{array}
$$

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

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 **one of** the three candidate **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 **one of** the three candidate **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 description logics: *DL-Lit* (34/56)

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

Based on query reformulation: given an (U)CQ and an ontology:

1. **Compute its perfect rewriting**, which turns out to be a UCQ.
2. **Evaluate the perfect rewriting** on the ABox seen as a DB.

To compute the perfect rewriting, starting from the original (U)CQ, iteratively get a CQ to be processed and either:

- **expand** positive inclusions & **simplify** redundant atoms, or
- **unify** atoms in the CQ to obtain a more specific CQ to be further expanded.

Each result of the above steps is added to the queries to be processed.

Note: negative inclusions and functionalities play a role in ontology satisfiability, but not in query answering.

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

Based on query reformulation: given an (U)CQ and an ontology:

1. **Compute its perfect rewriting**, which turns out to be a UCQ.
2. **Evaluate the perfect rewriting** on the ABox seen as a DB.

To compute the perfect rewriting, starting from the original (U)CQ, iteratively get a CQ to be processed and either:

- **expand** positive inclusions & **simplify** redundant atoms, or
- **unify** atoms in the CQ to obtain a more specific CQ to be further expanded.

Each result of the above steps is added to the queries to be processed.

Note: negative inclusions and functionalities play a role in ontology satisfiability, but not in query answering.

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

Based on query reformulation: given an (U)CQ and an ontology:

1. **Compute its perfect rewriting**, which turns out to be a UCQ.
2. **Evaluate the perfect rewriting** on the ABox seen as a DB.

To compute the perfect rewriting, starting from the original (U)CQ, iteratively get a CQ to be processed and either:

- **expand** positive inclusions & **simplify** redundant atoms, or
- **unify** atoms in the CQ to obtain a more specific CQ to be further expanded.

Each result of the above steps is added to the queries to be processed.

Note: negative inclusions and functionalities play a role in ontology satisfiability, but not in query answering.

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

TBox: $\mathsf{Professor} \sqsubseteq \exists\mathsf{worksFor}$
$\exists\mathsf{worksFor}^- \sqsubseteq \mathsf{College}$

Query: $\mathsf{q}(x) \leftarrow \mathsf{worksFor}(x, y), \mathsf{College}(y)$

Perfect Reformulation: $\mathsf{q}(x) \leftarrow \mathsf{worksFor}(x, y), \mathsf{College}(y)$
$\mathsf{q}(x) \leftarrow \mathsf{worksFor}(x, y), \mathsf{worksFor}(\_, y)$
$\mathsf{q}(x) \leftarrow \mathsf{worksFor}(x, \_)$
$\mathsf{q}(x) \leftarrow \mathsf{Professor}(x)$

ABox: $\mathsf{worksFor}(\mathtt{john}, \mathtt{collA})$ $\mathsf{Professor}(\mathtt{john})$
$\mathsf{worksFor}(\mathtt{mary}, \mathtt{collB})$ $\mathsf{Professor}(\mathtt{nick})$

Evaluating the last two queries over the ABox (seen as a DB) produces as answer $\{\mathtt{john}, \mathtt{nick}, \mathtt{mary}\}$.

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

TBox: $\mathsf{Professor} \sqsubseteq \exists\mathsf{worksFor}$
$\exists\mathsf{worksFor}^- \sqsubseteq \mathsf{College}$

Query: $\mathsf{q}(x) \leftarrow \mathsf{worksFor}(x, y), \mathsf{College}(y)$

Perfect Reformulation: $\mathsf{q}(x) \leftarrow \mathsf{worksFor}(x, y), \mathsf{College}(y)$
$\mathsf{q}(x) \leftarrow \mathsf{worksFor}(x, y), \mathsf{worksFor}(\_, y)$
$\mathsf{q}(x) \leftarrow \mathsf{worksFor}(x, \_)$
$\mathsf{q}(x) \leftarrow \mathsf{Professor}(x)$

ABox: $\mathsf{worksFor}(\mathtt{john}, \mathtt{collA})$ $\mathsf{Professor}(\mathtt{john})$
$\mathsf{worksFor}(\mathtt{mary}, \mathtt{collB})$ $\mathsf{Professor}(\mathtt{nick})$

Evaluating the last two queries over the ABox (seen as a DB) produces as answer $\{\mathtt{john}, \mathtt{nick}, \mathtt{mary}\}$.

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

TBox: $\mathsf{Professor} \sqsubseteq \exists\mathsf{worksFor}$
$\exists\mathsf{worksFor}^- \sqsubseteq \mathsf{College}$

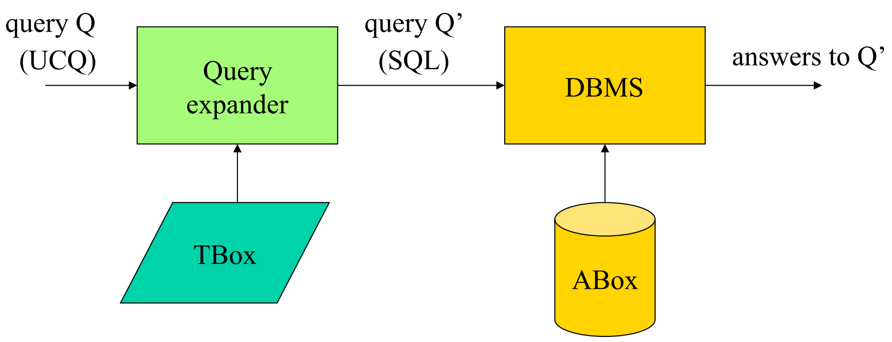
Query: $\mathsf{q}(x) \leftarrow \mathsf{worksFor}(x, y), \mathsf{College}(y)$

Perfect Reformulation: $\mathsf{q}(x) \leftarrow \mathsf{worksFor}(x, y), \mathsf{College}(y)$
$\mathsf{q}(x) \leftarrow \mathsf{worksFor}(x, y), \mathsf{worksFor}(\_, y)$
$\mathsf{q}(x) \leftarrow \mathsf{worksFor}(x, \_)$
$\mathsf{q}(x) \leftarrow \mathsf{Professor}(x)$

ABox: $\mathsf{worksFor}(\mathtt{john}, \mathtt{collA})$ $\mathsf{Professor}(\mathtt{john})$
$\mathsf{worksFor}(\mathtt{mary}, \mathtt{collB})$ $\mathsf{Professor}(\mathtt{nick})$

Evaluating the last two queries over the ABox (seen as a DB) produces as answer $\{\mathtt{john}, \mathtt{nick}, \mathtt{mary}\}$.

Query answering in DL-Lite



Example

TBox:

$\text{MALE} \sqsubseteq \text{PERSON}$ $\quad$ $\text{FEMALE} \sqsubseteq \text{PERSON}$

$\text{MALE} \sqsubseteq \neg\text{FEMALE}$

$\text{PERSON} \sqsubseteq \exists\text{hasFather}$ $\quad$ $\text{PERSON} \sqsubseteq \exists\text{hasMother}$

$\exists\text{hasFather}^{-} \sqsubseteq \text{MALE}$ $\quad$ $\exists\text{hasMother}^{-} \sqsubseteq \text{FEMALE}$

input query:

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

rewritten query:

$q'(x) \leftarrow \text{PERSON}(x) \lor$
$\text{FEMALE}(x) \lor$
$\text{MALE}(x) \lor$
$\text{hasFather}(y,x) \lor$
$\text{hasMother}(y,x)$

Example

rewritten query:

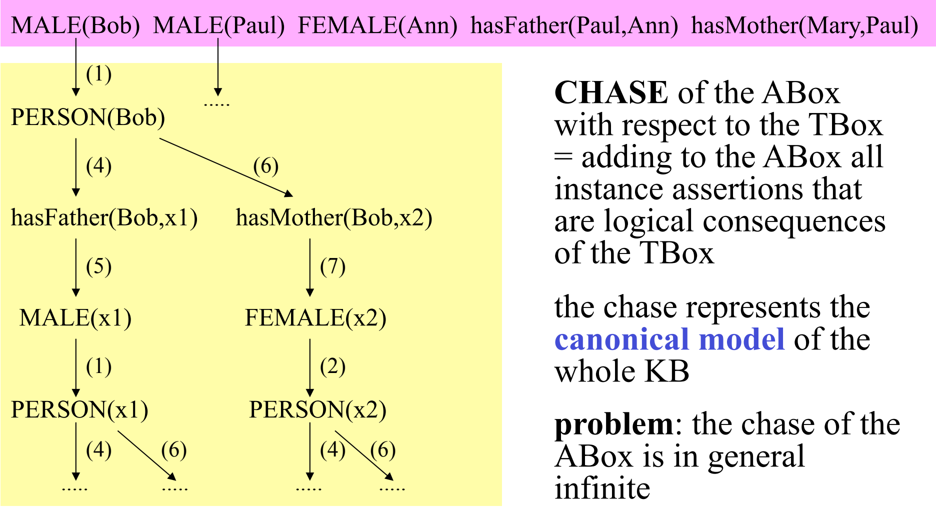
q'(x) ← PERSON(x) ∨
FEMALE(x) ∨
MALE(x) ∨
hasFather(y,x) ∨
hasMother(y,x)

ABox:

MALE(Bob)
MALE(Paul)
FEMALE(Ann)
hasFather(Paul,Ann)
hasMother(Mary,Paul)

answers to query:
{ Bob, Paul, Ann, Mary }

Answering queries: chasing the ABox



CHASE of the ABox with respect to the TBox = adding to the ABox all instance assertions that are logical consequences of the TBox

the chase represents the **canonical model** of the whole KB

problem: the chase of the ABox is in general infinite

Query rewriting algorithm for DL-Lite

$$\begin{array}{ccc} & q(x) \leftarrow PERSON(x) & \\ \swarrow & & \searrow \\ q(x) \leftarrow MALE(x) & & q(x) \leftarrow FEMALE(x) \\ \downarrow & & \downarrow \\ q(x) \leftarrow hasFather(y,x) & & q(x) \leftarrow hasMother(y,x) \end{array}$$

how to avoid the infinite chase of the ABox?

CHASE of the query:

- inclusions are applied “from right to left”
- this chase always terminates
- this chase is computed independently of the ABox

Query rewriting algorithm for DL-Lite

The rewriting algorithm iteratively applies two rewriting rules:

•**atom-rewrite**: takes an atom of the conjunctive query and rewrites it applying a TBox inclusion

- the inclusion is used as a rewriting rule (right-to-left)

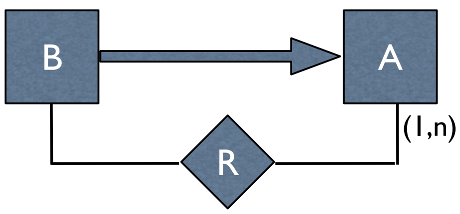
•**reduce**: takes two unifiable atoms of the conjunctive query and merges (unifies) them

Query rewriting algorithm for DL-Lite

```
Algorithm PerfectRef (q; 𝒯)
Input: conjunctive query q, DL-Lite TBox 𝒯
Output: union of conjunctive queries PR
PR := {q};
repeat
    PR0 := PR;
    for each q ∈ PR0 do
    (a) for each g in q do
          for each positive inclusion I  in 𝒯 do
            if I is applicable to g  then PR := PR ∪{q[g/gr(g,I)]};
    (b) for each g1, g2 in q  do
          if g1 and g2 unify then PR := PR ∪{f (reduce(q,g1,g2))}
until PR0 = PR;
return PR
```


KB

- TBOX:
  - A ISA SOME R
  - SOME R ISA A
  - SOME R⁻ ISA B
  - B ISA A



- ABOX:
  - B(c)
- QUERY:
  - q(x) :- R(x,y), R(y,z)

Query Answering

Expansion:

- q(x) :- R(x,y), R(y,z)
- q(x) :- R(x,y), R(y,_)
- q(x) :- R(x,y), A(y)
- q(x) :- R(x,y), B(y)
- q(x) :- R(x,y), R(_,y)
- q(x) :- R(x,y)
- q(x) :- R(x,_)
- q(x) :- A(x)
- q(x) :- B(x)

All queries empty except for the last!

Certain Answer: {c}

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*$_{\mathcal{A}}$?

No! By adding essentially any additional constructor we lose these nice computational properties.

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*$_{\mathcal{A}}$?

No! By adding essentially any additional constructor we lose these nice computational properties.

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*$_{\mathcal{A}}$?

No! By adding essentially any additional constructor we lose these nice computational properties.

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*$_\mathcal{A}$?

No! By adding essentially any additional constructor we lose these nice computational properties.

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

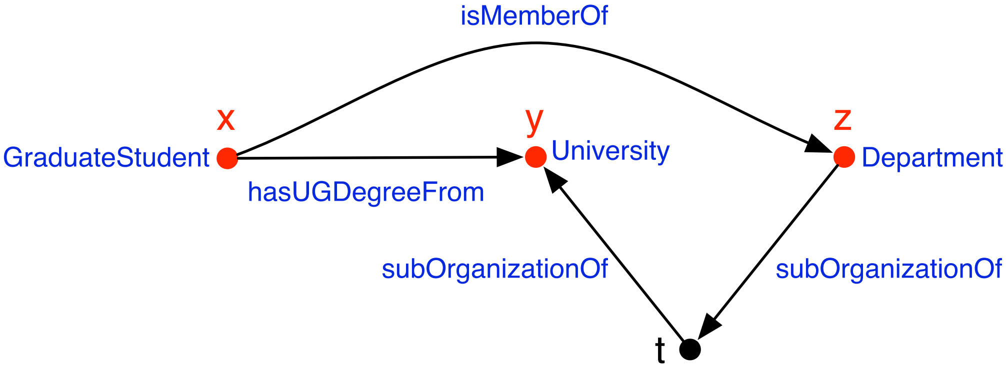
	lhs	rhs	funct.	Prop. incl.	Data complexity of query answering
0	*DL-Lite*$_\mathcal{A}$		$\surd^*$	$\surd^*$	in LOGSPACE
1	$A \mid \exists P.A$	A	–	–	NLOGSPACE-hard
2	A	$A \mid \forall P.A$	–	–	NLOGSPACE-hard
3	A	$A \mid \exists P.A$	$\surd$	–	NLOGSPACE-hard
4	$A \mid \exists P.A \mid A_1 \sqcap A_2$	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**.

Example of query

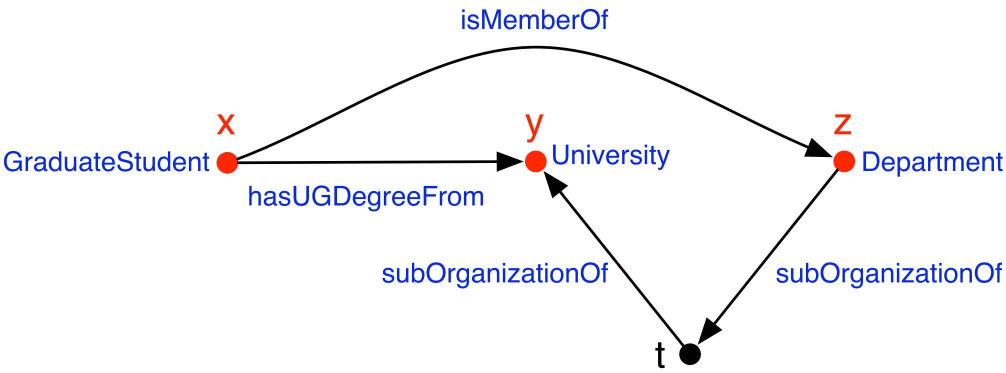
$$q(x,y,z) \leftarrow \text{GraduateStudent}(x),\ \text{University}(y),\ \text{Department}(z),\ \text{hasUndergraduateDegreeFrom}(x,y),\ \text{isMemberOf}(x,z),\ \text{subOrganizationOf}(z,t),\ \text{subOrganizationOf}(t,y)$$




```
SELECT ?X ?Y ?Z WHERE
  ?X rdf:type 'GraduateStudent'  .  ?Y rdf:type 'University' .
  ?Z rdf:type 'Department'  .
  ?X :hasUndergraduateDegreeFrom ?Y  .  ?X :isMemberOf ?Z  .
  ?Z subOrganizationOf ?T  .  ?T subOrganizationOf ?Y
```


Example of query

$$q(x,y,z) \leftarrow \text{GraduateStudent}(x),\ \text{University}(y),\ \text{Department}(z),\ \text{hasUndergraduateDegreeFrom}(x,y),\ \text{isMemberOf}(x,z),\ \text{subOrganizationOf}(z,t),\ \text{subOrganizationOf}(t,y)$$




```
SELECT ?X ?Y ?Z WHERE
  ?X rdf:type 'GraduateStudent'  .  ?Y rdf:type 'University' .
  ?Z rdf:type 'Department'  .
  ?X :hasUndergraduateDegreeFrom ?Y  .  ?X :isMemberOf ?Z  .
  ?Z subOrganizationOf ?T  .  ?T subOrganizationOf ?Y
```

Query answering in description logics: *DL-Lit* (48/56)

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 $DL\text{-}Lite_{\mathcal{A}}$

Answering of SparSQL queries in $DL\text{-}Lite_{\mathcal{A}}$:

1. Expand and unfold the UCQs (in the SparqlTables) as usual in $DL\text{-}Lite_{\mathcal{A}}$ $\leadsto$ 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. $\leadsto$ 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)

Outline

References I

[1] D. Berardi, D. Calvanese, and G. De Giacomo.
Reasoning on UML class diagrams.
Artificial Intelligence, 168(1–2):70–118, 2005.

[2] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, and R. Rosati.
Linking data to ontologies: The description logic *DL-Lite*$_{A}$.
In *Proc. of the 2nd Int. Workshop on OWL: Experiences and Directions (OWLED 2006)*, volume 216 of *CEUR Electronic Workshop Proceedings*, `http://ceur-ws.org/Vol-216/`, 2006.

[3] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati.
Tailoring OWL for data intensive ontologies.
In *Proc. of the 1st Int. Workshop on OWL: Experiences and Directions (OWLED 2005)*, volume 188 of *CEUR Electronic Workshop Proceedings*, `http://ceur-ws.org/Vol-188/`, 2005.

References II

[4] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati.
DL-Lite: Tractable description logics for ontologies.
In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, pages 602–607, 2005.

[5] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati.
Data complexity of query answering in description logics.
In *Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 260–270, 2006.

[6] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati.
Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family.
J. of Automated Reasoning, 39(3):385–429, 2007.

[7] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati.
Path-based identification constraints in description logics.
In *Proc. of the 11th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2008)*, pages 231–241, 2008.

References III

[8] D. Calvanese and M. Rodríguez.
An extension of DIG 2.0 for handling bulk data.
In *Proc. of the 3rd Int. Workshop on OWL: Experiences and Directions (OWLED 2007)*, volume 258 of *CEUR Electronic Workshop Proceedings*, `http://ceur-ws.org/Vol-258/`, 2007.

[9] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati.
Linking data to ontologies.
J. on Data Semantics, X:133–173, 2008.

[10] A. Poggi, M. Rodriguez, and M. Ruzzi.
Ontology-based database access with DIG-Mastro and the OBDA Plugin for Protégé.
In K. Clark and P. F. Patel-Schneider, editors, *Proc. of the OWL: Experiences and Directions 2008 (OWLED 2008 DC) Workshop*, 2008.

[11] M. Rodriguez-Muro, L. Lubyte, and D. Calvanese.
Realizing ontology based data access: A plug-in for Protégé.
In *Proc. of the 24th Int. Conf. on Data Engineering Workshops (ICDE 2008)*, pages 286–289, 2008.