Answering Queries in Description Logics: Theory and Applications to Data Management

Diego Calvanese[1], Michael Zakharyaschev[2]

[1] KRDB Research Centre
Free University of Bozen-Bolzano

[2] Birbeck College, London

ESSLLI 2010, August 14–20, 2010
Copenhagen, Denmark

Overview of the Course

Lecture III

Query answering in the *DL-Lite* family

Outline of Lecture 3

Outline of Lecture 3

Outline of Lecture 3

Queries over Description Logics ontologies

Traditionally, simple concept (or role) expressions have been considered as queries over DL ontologies.

We have seen that we need more complex forms of queries, such as those used in databases.

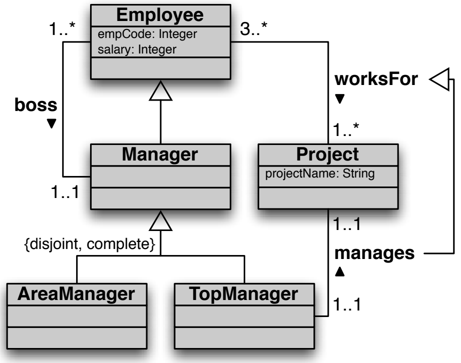
Def.: A **conjunctive query** $q(\vec{x})$ over an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$

is a conjunctive query $\exists \vec{y}.\, conj(\vec{x}, \vec{y})$

- whose **predicate symbols are atomic concept and roles** of $\mathcal{T}$, and
- that may contain constants that are individuals of $\mathcal{A}$.

Remember: a CQ corresponds to a select-project-join SQL query.

Queries over Description Logics ontologies – Example



Conjunctive query over the above ontology:

$$q(x, y) \leftarrow \exists p.\ \mathsf{Employee}(x), \mathsf{Employee}(y), \mathsf{Project}(p), \mathsf{boss}(x, y), \mathsf{worksFor}(x, p), \mathsf{worksFor}(y, p)$$

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 $q(\vec{x}) = \exists \vec{y}.\, conj(\vec{x}, \vec{y})$ 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 $\mathcal{O}$.

Query answering in ontologies

Def.: **Query answering** over an ontology $\mathcal{O}$

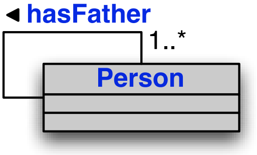
Is the problem of **computing the certain answers** to a query over $\mathcal{O}$.

Computing certain answers is a form of **logical implication**:

$$\vec{c} \in cert(q, \mathcal{O}) \qquad \text{iff} \qquad \mathcal{O} \models q(\vec{c})$$

Note: A special case of query answering is **instance checking**: it amounts to answering the boolean query $q() \leftarrow A(c)$ (resp., $q() \leftarrow P(c_1, c_2)$) over $\mathcal{O}$ (in this case $\vec{c}$ is the empty tuple).

Query answering in ontologies – Example



TBox $\mathcal{T}$:
$$\begin{array}{rcl} \exists \text{hasFather} & \sqsubseteq & \text{Person} \\ \exists \text{hasFather}^- & \sqsubseteq & \text{Person} \\ \text{Person} & \sqsubseteq & \exists \text{hasFather} \end{array}$$

ABox $\mathcal{A}$: Person(`john`), Person(`nick`), Person(`toni`)
hasFather(`john`,`nick`), hasFather(`nick`,`toni`)

Queries:

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

Certain answers:
$cert(q_1, \langle \mathcal{T}, \mathcal{A} \rangle) = \{$ (`john`,`nick`), (`nick`,`toni`) $\}$
$cert(q_2, \langle \mathcal{T}, \mathcal{A} \rangle) = \{$ `john`, `nick`, `toni` $\}$
$cert(q_3, \langle \mathcal{T}, \mathcal{A} \rangle) = \{$ `john`, `nick`, `toni` $\}$
$cert(q_4, \langle \mathcal{T}, \mathcal{A} \rangle) = \{ \}$

Unions of conjunctive queries

We consider also unions of CQs over an ontology.

A **union of conjunctive queries** (UCQ) has the form:

$$\exists \vec{y_1}.\, conj(\vec{x},\vec{y_1}) \lor \cdots \lor \exists \vec{y_k}.\, conj(\vec{x},\vec{y_k})$$

where each $\exists \vec{y_i}.\, conj(\vec{x},\vec{y_i})$ is a CQ.

The (certain) answers to a UCQ are defined analogously to those for CQs.

Example

$$\begin{aligned} q(x) \leftarrow\ & (\mathsf{Manager}(x) \land \mathsf{worksFor}(x, \mathtt{tones})) \ \lor \\ & (\exists y.\, \mathsf{boss}(x,y) \land \mathsf{worksFor}(y, \mathtt{tones})) \end{aligned}$$

In datalog notation:

$$\begin{aligned} q(x) &\leftarrow \mathsf{Manager}(x), \mathsf{worksFor}(x, \mathtt{tones}) \\ q(x) &\leftarrow \exists y.\, \mathsf{boss}(x,y) \land \mathsf{worksFor}(y, \mathtt{tones}) \end{aligned}$$

Outline of Lecture 3

Data and combined complexity

When measuring the complexity of answering a query $q(\vec{x})$ over an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, various parameters are of importance.

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.
- **Query complexity**: only the size of the query matters.
  TBox and ABox 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 OBDA setting, **the size of the data largely dominates** the size of the conceptual layer (and of the query).
$\leadsto$ **Data complexity** is the relevant complexity measure.

Data complexity of query answering

When studying the complexity of query answering, we need to consider the associated decision problem:

Def.: **Recognition problem** for query answering

Given an ontology $\mathcal{O}$, a query q over $\mathcal{O}$, and a tuple $\vec{c}$ of constants, **check whether** $\vec{c} \in cert(q, \mathcal{O})$.

We look mainly at the **data complexity** of query answering, i.e., complexity of the recognition problem computed **w.r.t. the size of the ABox only**.

Complexity of query answering in DLs

Query answering has been studied extensively for (unions of) CQs and various ontology languages:

	Combined complexity	Data complexity
Plain databases	NP-complete	in AC^0 [(1)]
OWL 2 (and less)	2ExpTime-complete [(3)]	coNP-hard [(2)]

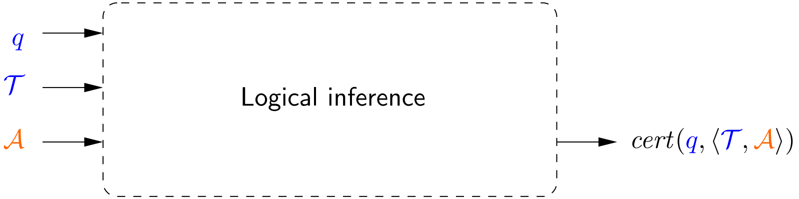
(1) This is what we need to scale with the data.
(2) Already for a TBox with a single disjunction [Donini *et al.*, 1994; Calvanese *et al.*, 2006].
But coNP-complete for very expressive DLs [Levy and Rousset, 1998; Ortiz *et al.*, 2006; Glimm *et al.*, 2007].
(3) [Calvanese *et al.*, 1998; Lutz, 2007]

Questions

- Can we find interesting (description) logics for which query answering can be done efficiently (i.e., in AC^0)?
- If yes, can we leverage relational database technology for query answering?

Outline of Lecture 3

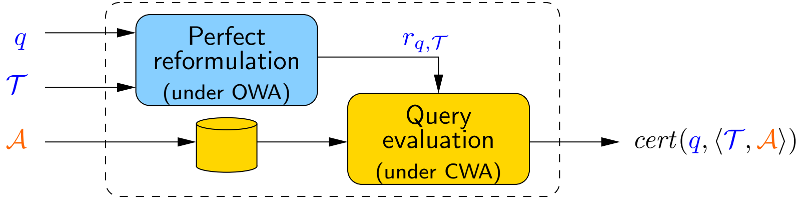
Inference in query answering



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 reformulation (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 $r_{q,\mathcal{T}}$.

$\mathcal{Q}$-rewritability

Let Q be a query language and $\mathcal{L}$ an ontology language.

Def.: **$\mathcal{Q}$-rewritability**

For an ontology language $\mathcal{L}$, query answering is **$\mathcal{Q}$-rewritable** if for every TBox $\mathcal{T}$ of $\mathcal{L}$ and for every query q, the perfect reformulation $r_{q,\mathcal{T}}$ of q w.r.t. $\mathcal{T}$ can be expressed in the query language $\mathcal{Q}$.

Notice that the complexity of computing $r_{q,\mathcal{T}}$ or the size of $r_{q,\mathcal{T}}$ do **not** affect data complexity.

Hence, $\mathcal{Q}$-rewritability is tightly related to **data complexity**, i.e.:

- complexity of computing $cert(q, \langle\mathcal{T}, \mathcal{A}\rangle)$ measured in the size of the ABox $\mathcal{A}$ only,
- which corresponds to the **complexity of evaluating** $r_{q,\mathcal{T}}$ **over** $\mathcal{A}$.

Language of the rewriting

The **expressiveness of the ontology language affects the rewriting language**, i.e., the language into which we are able to rewrite UCQs:

- When we can rewrite into FOL/SQL (i.e., the ontology language enjoys FOL-rewritability).
  $\rightsquigarrow$ Query evaluation can be done in SQL, i.e., via an RDBMS
  (*Note:* FOL is in AC^0).
- 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 *DL-Lite* and extensions

In the following, we study reasoning and query answering in *DL-Lite* and variants:

- We are interested in the boundary of languages that are "efficiently tractable" in the size of the data.
- We will show that:
  - *DL-Lite* and variants enjoy nice computational properties.
  - Extending *DL-Lite* with natural constructs makes it lose such properties.

Outline of Lecture 3

Outline of Lecture 3

DL-Lite$_{horn}^{(\mathcal{NH})}$

We start from *DL-Lite*$_{horn}^{(\mathcal{NH})}$, the maximal *DL-Lite* variant for which query answering is FOL-rewritable.

Concept and role language:

- Concept expressions: $B \longrightarrow \bot \mid A \mid \geq q\, R$ (A an atomic concept)
- Role expressions: $R \longrightarrow P \mid P^-$ (P an atomic role)

TBox:

- Concept inclusion assertions: $B_1 \sqcap \cdots \sqcap B_n \sqsubseteq B$
- Role inclusion assertions: $R_1 \sqsubseteq R_2$
- Role disjointness assertions: $R_1 \sqsubseteq \neg R_2$

With the **restriction** that if a role R has a proper subrole, then there are no negative occurrences of $\geq q\, R$ or $\geq q\, R^-$ with $q \geq 2$.

We consider also:

- Role functionality assertions: $\geq 2\, Q \sqsubseteq \bot$, or equivalently: (**funct** Q)

Beyond *DL-Lite*

We consider now DL languages containing **DL constructs beyond those of** $\mathit{DL\text{-}Lite}_{\mathbf{horn}}^{(\mathcal{NH})}$, or combinations of constructs that are not legal in $\mathit{DL\text{-}Lite}_{horn}^{(\mathcal{NH})}$.

We show that (essentially) all such extensions of *DL-Lite* make it lose its nice computational properties.

Specifically, we consider the following DL constructs:

Construct	Syntax	Example	Semantics
disjunction	$C_1 \sqcup C_2$	Doctor $\sqcup$ Lawyer	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
qual. exist. restr.	$\exists Q.C$	$\exists$child.Male	$\{a \mid \exists b.\, (a,b) \in Q^{\mathcal{I}} \wedge b \in C^{\mathcal{I}} \}$
qual. univ. restr.	$\forall Q.C$	$\forall$child.Male	$\{a \mid \forall b.\, (a,b) \in Q^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}} \}$

Beyond $\mathit{DL\text{-}Lite}_{\mathit{horn}}^{(\mathcal{NH})}$: results on data complexity

	Lhs	Rhs	Funct. assert.	Role incl.	Data complexity of query answering
0	$\mathit{DL\text{-}Lite}_{\mathit{horn}}^{(\mathcal{NH})}$		$\surd^*$	$\surd^*$	in AC^0
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 roles.
- 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**.

Observations

Data complexity of query answering:

- RDFS is a subset of *DL-Lite*$^{\mathcal{H}}_{core}$ $\leadsto$ is FOL-rewritable, hence AC^0.
- Horn-$\mathcal{SHIQ}$ [Hustadt *et al.*, 2005] is PTIME-**hard** even for instance checking (line 8).
- DLP [Grosof *et al.*, 2003] is PTIME-**hard** (line 4)
- $\mathcal{EL}$ [Baader *et al.*, 2005] is PTIME-**hard** (line 4).
- Although used in ER and UML, no hope of including **covering constraints**, since we get coNP-hardness for otherwise trivial DLs (line 10)

Outline of Lecture 3

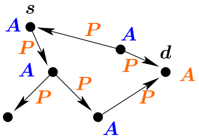
Qualified existential quantification in the lhs of inclusions

Adding **qualified existentials in the lhs** of inclusions makes instance checking (and hence query answering) NLogSpace-hard:

	Lhs	Rhs	$\mathcal{F}$	$\mathcal{H}$	Data complexity
1	$A \mid \exists P.A$	A	–	–	NLogSpace-hard

Hardness proof is by a reduction from reachability in directed graphs:

- TBox $\mathcal{T}$: a single inclusion assertion $\exists P.A \sqsubseteq A$
- ABox $\mathcal{A}$: encodes graph using P and asserts $A(d)$



Result:
$\langle \mathcal{T}, \mathcal{A} \rangle \models A(s)$ iff d is reachable from s in the graph.

Note: Since the reduction has to show hardness in data complexity, the graph must be encoded in the ABox (while the TBox has to be fixed).

NLogSpace-hard cases

Instance checking (and hence query answering) is NLogSpace-hard in data complexity for:

	Lhs	Rhs	$\mathcal{F}$	$\mathcal{H}$	Data complexity
1	$A \mid \exists P.A$	A	–	–	NLogSpace-hard

By reduction from reachability in directed graphs.

2	A	$A \mid \forall P.A$	–	–	NLogSpace-hard

Follows from 1 by replacing $\exists P.A_1 \sqsubseteq A_2$ with $A_1 \sqsubseteq \forall P^-.A_2$, and by replacing each occurrence of P^- with P', for a new role P'.

3	A	$A \mid \exists P.A$	$\surd$	–	NLogSpace-hard

Proved by simulating in the reduction $\exists P.A_1 \sqsubseteq A_2$
via $A_1 \sqsubseteq \exists P^-.A_2$ and (**funct** P^-),
and by replacing again each occurrence of P^- with P', for a new role P'.

Outline of Lecture 3

Path System Accessibility

To show PTIME-hardness, we use a reduction from a PTIME-complete problem. We use Path System Accessibility.

Instance of Path System Accessibility: $PS = (N, E, S, t)$ with

- N a set of nodes
- $E \subseteq N \times N \times N$ an accessibility relation
- $S \subseteq N$ a set of source nodes
- $t \in N$ a terminal node

Accessibility of nodes is defined inductively:

- each $n \in S$ is accessible
- if $(n, n_1, n_2) \in E$ and n_1, n_2 are accessible, then also n is accessible

Given an instance PS of Path System Accessibility, deciding whether t is accessible, is **PTIME-complete**.

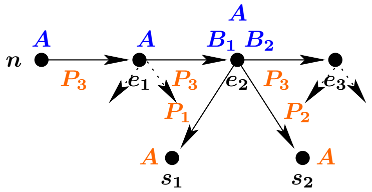
Reduction from Path System Accessibility

- We construct a TBox $\mathcal{T}$ consisting of the inclusion assertions:

$$\begin{array}{rcl} \exists P_1.A & \sqsubseteq & B_1 \\ \exists P_2.A & \sqsubseteq & B_2 \end{array} \qquad \begin{array}{rcl} B_1 \sqcap B_2 & \sqsubseteq & A \\ \exists P_3.A & \sqsubseteq & A \end{array}$$

- Given an instance $PS = (N, E, S, t)$, we construct an ABox $\mathcal{A}$ that:
  - encodes the accessibility relation using P_1, P_2, and P_3, and
  - asserts $A(s)$ for each source node $s \in S$.

$$\begin{array}{l} e_1 = (n,\ .\ ,\ .\ ) \\ e_2 = (n, s_1, s_2) \\ e_3 = (n,\ .\ ,\ .\ ) \end{array}$$



Result:

$\langle \mathcal{T}, \mathcal{A} \rangle \models A(t)$ iff t is accessible in PS.

Outline of Lecture 3

coNP-hard cases

Are obtained when we can use in the query **two concepts that cover another concept**. This forces **reasoning by cases** on the data.

Query answering is coNP-hard in data complexity for:

	Cl	Cr	$\mathcal{F}$	$\mathcal{H}$	Data complexity
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

All three cases are proved by adapting the proof of coNP-hardness of instance checking for $\mathcal{ALE}$ by [Donini *et al.*, 1994].

2+2-SAT

2+2-SAT: satisfiability of a 2+2-CNF formula, i.e., a CNF formula where each clause has exactly 2 positive and 2 negative literals.

Example: $\varphi = c_1 \land c_2 \land c_3$, with

$$\begin{aligned} c_1 &= v_1 \lor v_2 \lor \neg v_3 \lor \neg v_4 \\ c_2 &= \textit{false} \lor \textit{false} \lor \neg v_1 \lor \neg v_4 \\ c_3 &= \textit{false} \lor v_4 \lor \neg \textit{true} \lor \neg v_2 \end{aligned}$$

2+2-SAT is NP-complete [Donini *et al.*, 1994].

Reduction from 2+2-SAT

We construct a TBox $\mathcal{T}$ and a query $q()$ over concepts L, T, F and roles P_1, P_2, N_1, N_2.

- TBox $\mathcal{T} = \{\ L \sqsubseteq T \sqcup F\ \}$
- $q() \leftarrow P_1(c, v_1), P_2(c, v_2), N_1(c, v_3), N_2(c, v_4),$
  $F(v_1), F(v_2), T(v_3), T(v_4)$

Given a 2+2-CNF formula $\varphi = c_1 \wedge \cdots \wedge c_k$ over vars $v_1, \ldots, v_n$, *true*, *false*, we construct an ABox $\mathcal{A}_\varphi$ using individuals $\mathtt{c}_1, \ldots \mathtt{c}_k$, $\mathtt{v}_1, \ldots, \mathtt{v}_n$, $\mathtt{true}$, $\mathtt{false}$:

- for each propositional variable v_i: $L(\mathtt{v}_i)$
- for each clause $c_j = v_{j_1} \vee v_{j_2} \vee \neg v_{j_3} \vee \neg v_{j_4}$:
  $P_1(\mathtt{c}_j, \mathtt{v}_{j_1})$, $P_2(\mathtt{c}_j, \mathtt{v}_{j_2})$, $N_1(\mathtt{c}_j, \mathtt{v}_{j_3})$, $N_2(\mathtt{c}_j, \mathtt{v}_{j_4})$
- $T(\mathtt{true})$, $F(\mathtt{false})$

Note: the TBox $\mathcal{T}$ and the query q do not depend on φ, hence this reduction works for data complexity.

Reduction from 2+2-SAT (cont'd)

Lemma

$\langle \mathcal{T}, A_\varphi \rangle \not\models q()$ iff φ is satisfiable.

Proof (sketch).

"$\Rightarrow$" If $\langle \mathcal{T}, A_\varphi \rangle \not\models q()$, then there is a model $\mathcal{I}$ of $\langle \mathcal{T}, A_\varphi \rangle$ s.t. $\mathcal{I} \not\models q()$. We define a truth assignment $\alpha_\mathcal{I}$ by setting $\alpha_\mathcal{I}(v_i) =$ *true* iff $\mathrm{v}_i^\mathcal{I} \in T^\mathcal{I}$. Notice that, since $L \sqsubseteq T \sqcup F$, if $\mathrm{v}_i^\mathcal{I} \notin T^\mathcal{I}$, then $\mathrm{v}_i^\mathcal{I} \in F^\mathcal{I}$.
It is easy to see that, since $q()$ asks for a false clause and $\mathcal{I} \not\models q()$, for each clause c_j, one of the literals in c_j evaluates to *true* in $\alpha_\mathcal{I}$.
"$\Leftarrow$" From a truth assignment α that satisfies φ, we construct an interpretation $\mathcal{I}_\alpha$ with $\Delta^{\mathcal{I}_\alpha} = \{c_1, \ldots, c_k, v_1, \ldots, v_n, t, f\}$, and:

- $\mathtt{c}_j^{\mathcal{I}_\alpha} = c_j$, $\mathtt{v}_i^{\mathcal{I}_\alpha} = v_i$, $\mathtt{true}^{\mathcal{I}_\alpha} = t$, $\mathtt{false}^{\mathcal{I}_\alpha} = f$
- $T^{\mathcal{I}_\alpha} = \{v_i \mid \alpha(v_i) = \textit{true}\} \cup \{t\}$, $F^{\mathcal{I}_\alpha} = \{v_i \mid \alpha(v_i) = \textit{false}\} \cup \{f\}$

It is easy to see that $\mathcal{I}_\alpha$ is a model of $\langle \mathcal{T}, A_\varphi \rangle$ and that $\mathcal{I}_\alpha \not\models q()$. $\square$

Outline of Lecture 3

Combining functionalities and role inclusions

Consider now $\textit{DL-Lite}^{\mathcal{FH}}_{\textit{core}}$, which is the union of $\textit{DL-Lite}^{\mathcal{F}}_{\textit{core}}$ and $\textit{DL-Lite}^{\mathcal{H}}_{\textit{core}}$, i.e., the *DL-Lite* logic that allows for using both role functionality and role inclusions without any restrictions.

Due to the unrestricted interaction of functionality and role inclusions, $\textit{DL-Lite}^{\mathcal{FH}}_{\textit{core}}$ is significantly more complicated than the logics of the *DL-Lite* family:

- One can force the unification of existentially implied objects (i.e., separation does not hold anymore).
- Additional constructs besides those present in *DL-Lite* can be simulated.
- The computational complexity of reasoning increases significantly.

Unification of existentially implied objects – Example

TBox $\mathcal{T}$: $A \sqsubseteq \exists P$ $\quad P \sqsubseteq S$
$\exists P^- \sqsubseteq A$ $\quad$ (**funct** S)

ABox $\mathcal{A}$: $A(c_1),\ S(c_1,c_2),\ S(c_2,c_3),\ \ldots,\ S(c_{n-1},c_n)$

$$\begin{array}{rcl}
A(c_1),\quad A \sqsubseteq \exists P & \models & P(c_1,x),\ \text{for some } x\\
P(c_1,x),\quad P \sqsubseteq S & \models & S(c_1,x)\\
S(c_1,x),\quad S(c_1,c_2),\quad (\textbf{funct } S) & \models & x = c_2\\
P(c_1,c_2),\quad \exists P^- \sqsubseteq A & \models & A(c_2)\\
A(c_2),\quad A \sqsubseteq \exists P & \ldots & \\
 & \models & A(c_n)
\end{array}$$

Hence, we get:

- If we add $B(c_n)$ and $B \sqsubseteq \neg A$, the ontology becomes inconsistent.
- Similarly, the answer to the following query over $\langle \mathcal{T}, \mathcal{A} \rangle$ is *true*:

$$q() \leftarrow A(z_1), S(z_1,z_2), S(z_2,z_3), \ldots, S(z_{n-1},z_n), A(z_n)$$

Unification of existentially implied objects

Note: The number of unification steps above **depends on the data**. Hence this kind of deduction cannot be mimicked by a FOL (or SQL) query, since it requires a form of **recursion**. As a consequence, we get:

Combining functionality and role inclusions is problematic.

It breaks **separability**, i.e., functionality assertions may force existentially quantified objects to be unified with existing objects.

Note: the problems are caused by the **interaction** among:

- an inclusion $P \sqsubseteq S$ between roles,
- a functionality assertion $(\textbf{funct}\ S)$ on the super-role, and
- a cycle of concept inclusion assertions $A \sqsubseteq \exists P$ and $\exists P^- \sqsubseteq A$.

Simulation of constructs using funct. and role inclusions

In fact, by exploiting the interaction between functionality and role inclusions, we can simulate typical DL constructs not present in *DL-Lite*:

- Simulation of $A \sqsubseteq \exists R.C$: (*Note:* this does not require functionality)

$$A \sqsubseteq \exists R_C \qquad R_C \sqsubseteq R \qquad \exists R_C^- \sqsubseteq C$$

- Simulation of $A_1 \sqcap A_2 \sqsubseteq C$:

$$\begin{array}{lll} A_1 \sqsubseteq \exists R_1 & A_2 \sqsubseteq \exists R_2 & \\ R_1 \sqsubseteq R_{12} & R_2 \sqsubseteq R_{12} & (\textbf{funct } R_{12}) \\ \exists R_1^- \sqsubseteq \exists R_3^- & & \\ \exists R_3 \sqsubseteq C & & \\ R_3 \sqsubseteq R_{23} & R_2 \sqsubseteq R_{23} & (\textbf{funct } R_{23}^-) \end{array}$$

Simulation of constructs (cont'd)

Simulation of $A \sqsubseteq \forall R.C$:

We use **reification** of roles:

$$\begin{tikzcd} \bullet \arrow[r, "R"] & \bullet \end{tikzcd} \qquad \begin{tikzcd} \bullet & \bullet \arrow[l, "S_1"'] \arrow[r, "S_2"] & \bullet \end{tikzcd}$$

$S_{1,C} \sqsubseteq S_1$	$S_{1,\neg C} \sqsubseteq S_1$	(**funct** S_1)
$S_{2,C} \sqsubseteq S_2$	$S_{2,\neg C} \sqsubseteq S_2$	(**funct** S_2)
$\exists S_{1,C} \equiv \exists S_{2,C}$	$\exists S_{1,\neg C} \equiv \exists S_{2,\neg C}$	

$$\exists S_2 \sqsubseteq \exists S_{2,C} \sqcup \exists S_{2,\neg C}$$

$$\exists S_{2,C}^- \sqsubseteq C \qquad \exists S_{2,\neg C}^- \sqsubseteq \neg C$$

$$A \sqsubseteq \neg \exists S_{1,\neg C}^-$$

Complexity of *DL-Lite* with funct. and role inclusions

We can exploit the above constructions that simulate DL constructs to show lower bounds for reasoning with both functionality and role inclusions.

Theorem [Artale *et al.*, 2009]

For $DL\text{-}Lite_{core}^{\mathcal{FH}}$ ontologies:

- Checking satisfiability of the ontology is
  - ExpTime-**complete** in the size of the **ontology** (combined complexity).
  - PTime-**complete** in the size of the **ABox** (data complexity).
- TBox reasoning is ExpTime-**complete** in the size of the **TBox**.
- Query answering is
  - NP-**complete** in the size of the query and the ontology (comb. com.).
  - ExpTime-**complete** in the size of the **ontology**.
  - PTime-**complete** in the size of the **ABox** (data complexity).

Combining functionalities and role inclusions

We have seen that:

- By including in *DL-Lite* both functionality of roles and role inclusions without restrictions on their interaction, query answering becomes PTIME-hard.
- When the data complexity of query answering is NLOGSPACE or above, the DL does not enjoy FOL-rewritability.

As a consequence of these results, we get:

To preserve FOL-rewritability, the restriction on the interaction of functionality and role inclusions of $\textit{DL-Lite}_{\textit{core}}^{(\mathcal{FH})}$ is necessary.

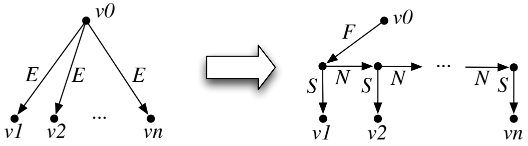
Outline of Lecture 3

Dropping the unique name assumption

Recall: the unique name assumption (UNA) states that different individuals must be interpreted as different domain objects.

We reconsider the complexity of query evaluation in $DL\text{-}Lite^{\mathcal{F}}_{core}$, and show that **without the UNA the data complexity increases**.

- We show how to reduce **reachability in directed graphs** to instance checking in $DL\text{-}Lite^{\mathcal{F}}_{core}$ without the UNA. This gives us an NLogSpace lower bound.
- We assume that the graph is represented through the first-child and next-sibling functional relations:



Dropping the unique name assumption (cont'd)

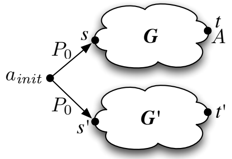
From G and two vertexes s and t of G, we define $\mathcal{O}_{una} = \langle \mathcal{T}_{una}, \mathcal{A}_G \rangle$:

- TBox uses an atomic concept A, and atomic roles P_0, P_F, P_N, P_S:

$$\mathcal{T}_{una} \;=\; \{(\textbf{funct}\ P_0)\} \cup \{(\textbf{funct}\ P_{\mathcal{R}}) \mid \mathcal{R} \in \{F, N, S\}\}.$$

- ABox is defined from G and the two vertexes s and t:

$$\begin{aligned}\mathcal{A}_G \;=\; & \{P_{\mathcal{R}}(a_1, a_2), P_{\mathcal{R}}(a'_1, a'_2) \mid (a_1, a_2) \in \mathcal{R},\ \text{for } \mathcal{R} \in \{F, N, S\}\} \cup \\ & \{A(t),\ P_0(a_{init}, s),\ P_0(a_{init}, s')\}\end{aligned}$$



This means that we encode in $\mathcal{A}_G$ two copies of G.

Note: $\mathcal{A}_G$ depends on G, but $\mathcal{T}_{una}$ does not.

We can show by induction on the length of paths from s that ...

t is reachable from s in G if and only if $\mathcal{O}_{una} \models A(t')$.

Dropping the unique name assumption – Complexity

The previous reduction shows that instance checking in $\textit{DL-Lite}^{\mathcal{F}}_{core}$ (and hence also $\textit{DL-Lite}^{(\mathcal{FH})}_{core}$) without the UNA is NLogSpace-hard.

With a more involved reduction, one can show an even stronger lower bound, that turns out to be tight.

Theorem [Artale *et al.*, 2009]

Instance checking in $\textit{DL-Lite}^{\mathcal{F}}_{core}$ and $\textit{DL-Lite}^{(\mathcal{FH})}_{core}$ without the UNA is PTime-complete in data complexity.

Outline of Lecture 3

Outline of Lecture 3

TBox and ABox reasoning services

- **Ontology Satisfiability:** Verify whether an ontology $\mathcal{O}$ is satisfiable, i.e., whether $\mathcal{O}$ admits at least one model.
- **Concept Instance Checking:** Verify whether an individual c is an instance of a concept C in an ontology $\mathcal{O}$, i.e., whether $\mathcal{O} \models C(c)$.
- **Role Instance Checking:** Verify whether a pair (c_1, c_2) of individuals is an instance of a role Q in an ontology $\mathcal{O}$, i.e., whether $\mathcal{O} \models Q(c_1, c_2)$.
- **Query Answering** Given a query q over an ontology $\mathcal{O}$, find all tuples $\vec{c}$ of constants such that $\mathcal{O} \models q(\vec{c})$.

Query answering and instance checking

For atomic concepts and roles, **instance checking is a special case of query answering**, in which the query is **boolean** and constituted by a single atom in the body.

- $\mathcal{O} \models A(c)$ iff $q() \leftarrow A(c)$ evaluated over $\mathcal{O}$ is true.
- $\mathcal{O} \models P(c_1, c_2)$ iff $q() \leftarrow P(c_1, c_2)$ evaluated over $\mathcal{O}$ is true.

From instance checking to ontology unsatisfiability

Theorem

Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a *DL-Lite*$_{core}^{(\mathcal{FH})}$ ontology, C a *DL-Lite* concept, and P an atomic role. Then:

- $\mathcal{O} \models C(c)$ iff $\mathcal{O}_{C(c)} = \langle \mathcal{T} \cup \{\hat{A} \sqsubseteq \neg C\},\ \mathcal{A} \cup \{\hat{A}(c)\} \rangle$ is unsatisfiable, where $\hat{A}$ is an atomic concept not in $\mathcal{O}$.
- $\mathcal{O} \models P(c_1, c_2)$ iff $\mathcal{O}_{P(c_1,c_2)} = \langle \mathcal{T} \cup \{\hat{P} \sqsubseteq \neg P\},\ \mathcal{A} \cup \{\hat{P}(c_1, c_2)\} \rangle$ is unsatisfiable, where $\hat{P}$ is an atomic role not in $\mathcal{O}$.
- $\mathcal{O} \models \neg P(c_1, c_2)$ iff $\mathcal{O}_{\neg P(c_1,c_2)} = \langle \mathcal{T},\ \mathcal{A} \cup \{P(c_1, c_2)\} \rangle$ is unsatisfiable.

Outline of Lecture 3

Certain answers

We recall that

Query answering over an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is a form of **logical implication**:

Find all tuples $\vec{c}$ of constants of $\mathcal{A}$ s.t. $\mathcal{O} \models q(\vec{c})$.

A.k.a. **certain answers** in databases, i.e., the tuples that are answers to q in **all** models of $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$:

$$cert(q, \mathcal{O}) \ = \ \{ \ \vec{c} \ \mid \ \vec{c} \in q^{\mathcal{I}}, \ \text{ for every model } \mathcal{I} \text{ of } \mathcal{O} \ \}$$

Note: We have assumed that the answer $q^{\mathcal{I}}$ to a query q over an interpretation $\mathcal{I}$ is constituted by a set of tuples of **constants** of $\mathcal{A}$, rather than objects in $\Delta^{\mathcal{I}}$.

$\mathcal{Q}$-rewritability for *DL-Lite*

- We now study rewritability of query answering over *DL-Lite* ontologies.
- In particular we will show that $\mathit{DL\text{-}Lite}_{\mathit{core}}^{(\mathcal{FH})}$ enjoys FOL-rewritability of answering unions of conjunctive queries.

Query answering vs. ontology satisfiability

- In the case in which an ontology is unsatisfiable, according to the "ex falso quod libet" principle, reasoning is trivialized.
- In particular, query answering is meaningless, since every tuple is in the answer to every query.
- We are not interested in encoding meaningless query answering into the perfect reformulation of the input query. Therefore, before query answering, we will always check ontology satisfiability to single out meaningful cases.

Thus, we proceed as follows:

1. We show how to do **query answering over satisfiable ontologies**.
2. We show how we can exploit the query answering algorithm also to check ontology satisfiability.

Remark

We call positive inclusions (PIs) assertions of the form

$$\begin{array}{rcl} Cl & \sqsubseteq & A \mid \exists R \\ R_1 & \sqsubseteq & R_2 \end{array}$$

We call negative inclusions (NIs) assertions of the form

$$\begin{array}{rcl} Cl & \sqsubseteq & \neg A \mid \neg\exists R \\ R_1 & \sqsubseteq & \neg R_2 \end{array}$$

Outline of Lecture 3

Query answering over satisfiable ontologies

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 $\mathcal{T}$, **rewrite** q into a UCQ $r_{q,\mathcal{T}}$ (the perfect rewriting of q w.r.t. $\mathcal{T}$).
2. **Evaluate** $r_{q,\mathcal{T}}$ over $\mathcal{A}$ (simply viewed as data), to return $cert(q, \mathcal{O})$.

Correctness of this procedure shows FOL-rewritability of query answering in *DL-Lite*$_{core}^{(\mathcal{FH})}$.

Query rewriting

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

Intuition: Use the PIs as basic rewriting rules:

$$\mathsf{AssistantProf} \sqsubseteq \mathsf{Professor}$$

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

Basic rewriting step:

when an atom in the query unifies with the **head** of the rule,

substitute the atom with the **body** of the rule.

We say that the PI inclusion **applies to** the atom.

In the example, the PI $\mathsf{AssistantProf} \sqsubseteq \mathsf{Professor}$ applies to the atom $\mathsf{Professor}(x)$. Towards the computation of the perfect rewriting, we add to the input query above, the query

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

Query rewriting (cont'd)

Consider the query $\mathsf{q}(x) \leftarrow \mathsf{teaches}(x,y), \mathsf{Course}(y)$

and the PI $\exists\mathsf{teaches}^- \sqsubseteq \mathsf{Course}$

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

The PI applies to the atom $\mathsf{Course}(y)$, and we add to the perfect rewriting the query

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

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

and the PI $\mathsf{Professor} \sqsubseteq \exists\mathsf{teaches}$

as a logic rule: $\mathsf{teaches}(z,f(z)) \leftarrow \mathsf{Professor}(z)$

The PI applies to the atom $\mathsf{teaches}(x,y)$, and we add to the perfect rewriting the query

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

Query rewriting – Constants

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

and the same PI as before $\mathsf{Professor} \sqsubseteq \exists\mathsf{teaches}$

as a logic rule: $\mathsf{teaches}(z, f(z)) \leftarrow \mathsf{Professor}(z)$

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

In this case, we say that the PI does not apply to the atom $\mathsf{teaches}(x, \mathtt{fl})$.

The same holds for the following query, where y is **distinguished**, since unifying $f(z)$ with y would correspond to returning a skolem term as answer to the query:

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

Query rewriting – Join variables

An analogous behavior to the one with constants and with distinguished variables holds when the atom contains **join variables** that would have to be unified with skolem terms.

Consider the query $\mathsf{q}(x) \leftarrow \mathsf{teaches}(x,y), \mathsf{Course}(y)$

and the PI $\mathsf{Professor} \sqsubseteq \exists \mathsf{teaches}$

as a logic rule: $\mathsf{teaches}(z, f(z)) \leftarrow \mathsf{Professor}(z)$

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

Query rewriting – Reduce step

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

and the PI $\mathsf{Professor} \sqsubseteq \exists \mathsf{teaches}$

as a logic rule: $\mathsf{teaches}(z, f(z)) \leftarrow \mathsf{Professor}(z)$

This PI does not apply to $\mathsf{teaches}(x,y)$ or $\mathsf{teaches}(z,y)$, since y is in join, and we would again introduce the skolem term in the rewritten query.

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

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

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

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

Query rewriting – Summary

Reformulate the CQ q into a set of queries:

- Apply to q and the computed queries in all possible ways the PIs in $\mathcal{T}$:

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

  ('_' denotes an **unbound** variable, i.e., a variable that appears only once)

  This corresponds to exploiting ISAs, role typing, and mandatory participation to obtain new queries that could contribute to the answer.

- Apply in all possible ways unification between atoms in a query.

  Unifying atoms can make rules applicable that were not so before, and is required for completeness of the method.

The UCQ resulting from this process is the **perfect rewriting** $r_{q,\mathcal{T}}$.

Query rewriting algorithm

```
**Algorithm** PerfectRef($Q, \mathcal{T}_P$)
**Input:** union of conjunctive queries $Q$, set of DL-Lite$_{core}^{(\mathcal{FH})}$ PIs $\mathcal{T}_P$
**Output:** union of conjunctive queries $PR$
$PR := Q$;
repeat
  $PR' := PR$;
  for each $q \in PR'$ do
    for each $g$ in $q$ do
      for each PI $I$ in $\mathcal{T}_P$ do
        if $I$ is applicable to $g$ then $PR := PR \cup \{ \mathit{ApplyPI}(q, g, I) \}$;
    for each $g_1, g_2$ in $q$ do
      if $g_1$ and $g_2$ unify then $PR := PR \cup \{\tau(\mathit{Reduce}(q, g_1, g_2))\}$;
until $PR' = PR$;
return $PR$
```

Observations:

- Termination follows from having only finitely many different rewritings.
- NIs or functionalities do not play any role in the rewriting of the query.

Query answering in *DL-Lite* – Exercise

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

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

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

ABox: $\mathsf{teaches}(\mathtt{john}, \mathtt{fl})$
$\mathsf{Professor}(\mathtt{mary})$

It is easy to see that evaluating the perfect rewriting over the ABox viewed as a database produces as answer $\{\mathtt{john}, \mathtt{mary}\}$.

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

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

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

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

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

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

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

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

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

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

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

$\Downarrow$

$\cdots$

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

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

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

Query answering over satisfiable $\textit{DL-Lite}_{\textit{core}}^{(\mathcal{FH})}$ ontologies

For an ABox $\mathcal{A}$ and a query q over $\mathcal{A}$, let $\textit{Eval}_{\text{CWA}}(q, \mathcal{A})$ denote the evaluation of q over $\mathcal{A}$ considered as a database (i.e., considered under the CWA).

Theorem

Let $\mathcal{T}$ be a $\textit{DL-Lite}_{\textit{core}}^{(\mathcal{FH})}$ TBox, $\mathcal{T}_P$ the set of PIs in $\mathcal{T}$, and q a CQ over $\mathcal{T}$. Then, for each ABox $\mathcal{A}$ such that $\langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable, we have that

$$cert(q, \langle \mathcal{T}, \mathcal{A} \rangle) = \textit{Eval}_{\text{CWA}}(\textit{PerfectRef}(q, \mathcal{T}_P), \mathcal{A}).$$

The proof exploits the canonical model, by relating the rewriting steps to the (chase) steps used for the construction of the canonical model.

As a consequence, query answering over a satisfiable $\textit{DL-Lite}_{\textit{core}}^{(\mathcal{FH})}$ ontology is FOL-rewritable.

Notice that we did not use NIs or functionality assertions of $\mathcal{T}$ in computing $cert(q, \langle \mathcal{T}, \mathcal{A} \rangle$. Indeed, **when the ontology is satisfiable, we can ignore NIs and functionality assertions for query answering**.

Using RDBMS technology for query answering

The **ABox** $\mathcal{A}$ can be stored as a **relational database** in a standard RDBMS:

- For each atomic concept A of the ontology:
  - define a unary relational table $\texttt{tab}_A$,
  - populate $\texttt{tab}_A$ with each $\langle c \rangle$ such that $A(c) \in \mathcal{A}$.
- For each atomic role P of the ontology,
  - define a binary relational table $\texttt{tab}_P$,
  - populate $\texttt{tab}_P$ with each $\langle c_1, c_2 \rangle$ such that $P(c_1, c_2) \in \mathcal{A}$.

We have that query answering over satisfiable $\textit{DL-Lite}_{\textit{core}}^{(\mathcal{FH})}$ ontologies can be done effectively using RDBMS technology:

$$cert(q, \langle \mathcal{T}, \mathcal{A} \rangle) = \textit{Eval}(\textit{SQL}(\textit{PerfectRef}(q, \mathcal{T}_P)), \textit{DB}(\mathcal{A}))$$

Where:

– $\textit{Eval}(q_s, \textit{DB})$ denotes the evaluation of an SQL query q_s over a database *DB*.
– $\textit{SQL}(q)$ denotes the SQL encoding of a UCQ q.
– $\textit{DB}(\mathcal{A})$ denotes the database obtained as above.

Outline of Lecture 3

Satisfiability of ontologies with only PIs

Let us now consider the problem of establishing whether an ontology is satisfiable.

A first notable result tells us that PIs alone cannot generate ontology unsatisfiability.

Theorem

Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a $\textit{DL-Lite}_{\mathit{core}}^{(\mathcal{FH})}$ ontology where $\mathcal{T}$ contains **only PIs**.
Then, $\mathcal{O}$ is satisfiable.

Satisfiability of $\textit{DL-Lite}_{\textit{core}}^{(\mathcal{FH})}$ ontologies

Unsatisfiability in $\textit{DL-Lite}_{\textit{core}}^{(\mathcal{FH})}$ ontologies can be caused by **NIs** or by **functionality assertions**.

Example

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

ABox $\mathcal{A}$: $\text{Student}(\texttt{john})$
$\text{teaches}(\texttt{john}, \texttt{fl})$
$\text{teaches}(\texttt{michael}, \texttt{fl})$

Checking satisfiability of $\textit{DL-Lite}_{\textit{core}}^{(\mathcal{FH})}$ ontologies

Satisfiability of a $\textit{DL-Lite}_{\textit{core}}^{(\mathcal{FH})}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is reduced to evaluating over $DB(\mathcal{A})$ a UCQ that asks for the **existence of objects violating the NI and functionality assertions**.

Let $\mathcal{T}_P$ the set of PIs in $\mathcal{T}$.
We deal with NIs and functionality assertions differently.

For each NI $N \in \mathcal{T}$:

1. we construct a boolean CQ $q_N()$ such that
$$\langle \mathcal{T}_P, \mathcal{A} \rangle \models q_N() \quad \text{iff} \quad \langle \mathcal{T}_P \cup \{N\}, \mathcal{A} \rangle \text{ is unsatisfiable}$$
2. We check whether $\langle \mathcal{T}_P, \mathcal{A} \rangle \models q_N()$ using *PerfectRef*, i.e., we compute *PerfectRef*$(q_N, \mathcal{T}_P)$, and evaluate it over $DB(\mathcal{A})$.

For each functionality assertion $F \in \mathcal{T}$:

1. we construct a boolean CQ $q_F()$ such that
$$\mathcal{A} \models q_F() \quad \text{iff} \quad \langle \{F\}, \mathcal{A} \rangle \text{ is unsatisfiable.}$$
2. We check whether $\mathcal{A} \models q_F()$, by simply evaluating q_F over $DB(\mathcal{A})$.

Checking violations of negative inclusions

For each **NI** N in $\mathcal{T}$ we compute a boolean CQ $q_N()$ according to the following rules:

$$\begin{array}{rcl}
A_1 \sqsubseteq \neg A_2 & \rightsquigarrow & q_N() \leftarrow A_1(x), A_2(x) \\
\exists P \sqsubseteq \neg A \text{ or } A \sqsubseteq \neg \exists P & \rightsquigarrow & q_N() \leftarrow P(x,y), A(x) \\
\exists P^- \sqsubseteq \neg A \text{ or } A \sqsubseteq \neg \exists P^- & \rightsquigarrow & q_N() \leftarrow P(y,x), A(x) \\
\exists P_1 \sqsubseteq \neg \exists P_2 & \rightsquigarrow & q_N() \leftarrow P_1(x,y), P_2(x,z) \\
\exists P_1 \sqsubseteq \neg \exists P_2^- & \rightsquigarrow & q_N() \leftarrow P_1(x,y), P_2(z,x) \\
\exists P_1^- \sqsubseteq \neg \exists P_2 & \rightsquigarrow & q_N() \leftarrow P_1(x,y), P_2(y,z) \\
\exists P_1^- \sqsubseteq \neg \exists P_2^- & \rightsquigarrow & q_N() \leftarrow P_1(x,y), P_2(z,y) \\
P_1 \sqsubseteq \neg P_2 \text{ or } P_1^- \sqsubseteq \neg P_2^- & \rightsquigarrow & q_N() \leftarrow P_1(x,y), P_2(x,y) \\
P_1^- \sqsubseteq \neg P_2 \text{ or } P_1 \sqsubseteq \neg P_2^- & \rightsquigarrow & q_N() \leftarrow P_1(x,y), P_2(y,x)
\end{array}$$

Checking violations of negative inclusions – Example

PIs $\mathcal{T}_P$: $\exists\text{teaches} \sqsubseteq \text{Professor}$
NIs N : $\text{Professor} \sqsubseteq \neg\text{Student}$

Query q_N: $q_N() \leftarrow \text{Student}(x), \text{Professor}(x)$

Perfect Rewriting:
$q_N() \leftarrow \text{Student}(x), \text{Professor}(x)$
$q_N() \leftarrow \text{Student}(x), \text{teaches}(x, \_)$

ABox $\mathcal{A}$:
$\text{teaches}(\texttt{john}, \texttt{fl})$
$\text{Student}(\texttt{john})$

It is easy to see that $\langle \mathcal{T}_P, \mathcal{A} \rangle \models q_N()$, and that the ontology $\langle \mathcal{T}_P \cup \{\text{Professor} \sqsubseteq \neg\text{Student}\},\ \mathcal{A}\rangle$ **is unsatisfiable**.

Boolean queries vs. non-boolean queries for NIs

To ensure correctness of the method, the queries used to check for the violation of a NI need to be **boolean**.

Example

TBox $\mathcal{T}$: $A_1 \sqsubseteq \neg A_0$ $\quad \exists P \sqsubseteq A_1$
$A_1 \sqsubseteq A_0$ $\quad A_2 \sqsubseteq \exists P^-$

ABox $\mathcal{A}$: $A_2(c)$

Since A_1, P, and A_2 are unsatisfiable, also $\langle \mathcal{T}, \mathcal{A} \rangle$ is unsatisfiable.

Consider the query corresponding to the NI $A_1 \sqsubseteq \neg A_0$.

$q_N() \leftarrow A_1(x), A_0(x)$

Then *PerfectRef*($q_N, \mathcal{T}_P$) is:

$q_N() \leftarrow A_1(x), A_0(x)$
$q_N() \leftarrow A_1(x)$
$q_N() \leftarrow P(x, \_)$
$q_N() \leftarrow A_2(\_)$

We have that $\langle \mathcal{T}_P, \mathcal{A} \rangle \models q_N()$.

$q'_N(x) \leftarrow A_1(x), A_0(x)$

Then *PerfectRef*($q'_N, \mathcal{T}_P$) is

$q'_N(x) \leftarrow A_1(x), A_0(x)$
$q'_N(x) \leftarrow A_1(x)$
$q'_N(x) \leftarrow P(x, \_)$

$cert(q'_N, \langle \mathcal{T}_P, \mathcal{A} \rangle) = \emptyset$, hence $q'_N(x)$ does not detect unsatisfiability.

Checking violations of functionality assertions

For each **functionality assertion** F in $\mathcal{T}$ we compute a boolean FOL query $q_F()$ according to the following rules:

$$\begin{array}{lcl} (\textbf{funct } P) & \rightsquigarrow & q_F() \leftarrow P(x,y), P(x,z), y \neq z \\ (\textbf{funct } P^-) & \rightsquigarrow & q_F() \leftarrow P(x,y), P(z,y), x \neq z \end{array}$$

Example

Functionality F: $(\textbf{funct teaches}^-)$

Query q_F: $q_F() \leftarrow \textbf{teaches}(x,y), \textbf{teaches}(z,y), x \neq z$

ABox $\mathcal{A}$: $\mathtt{teaches}(\mathtt{john}, \mathtt{fl})$
$\mathtt{teaches}(\mathtt{michael}, \mathtt{fl})$

It is easy to see that $\mathcal{A} \models q_F()$, and that $\langle\{(\textbf{funct teaches}^-)\}, \mathcal{A}\rangle$, **is unsatisfiable**.

From satisfiability to query answering in $\textit{DL-Lite}_{\textit{core}}^{(\mathcal{FH})}$

Lemma (Separation for $\textit{DL-Lite}_{\textit{core}}^{(\mathcal{FH})}$)

Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a $\textit{DL-Lite}_{\textit{core}}^{(\mathcal{FH})}$ ontology, and $\mathcal{T}_P$ the set of PIs in $\mathcal{T}$. Then, $\mathcal{O}$ is unsatisfiable iff one of the following condition holds:

(a) There exists a NI $N \in \mathcal{T}$ such that $\langle \mathcal{T}_P, \mathcal{A} \rangle \models q_N()$.

(b) There exists a functionality assertion $F \in \mathcal{T}$ such that $\mathcal{A} \models q_F()$.

(a) relies on the properties that **NIs do not interact with each other**, and that **interaction between NIs and PIs** is captured **through** *PerfectRef*.

(b) exploits the property that **NIs and PIs do not interact with functionalities**: indeed, no functionality assertion is contradicted in a $\textit{DL-Lite}_{\textit{core}}^{(\mathcal{FH})}$ ontology $\mathcal{O}$, beyond those explicitly contradicted by the ABox.

Notably, to check ontology satisfiability, each NI and each functionality assertion can be processed individually.

FOL-rewritability of satisfiability in $\textit{DL-Lite}_{\textit{core}}^{(\mathcal{FH})}$

From the previous lemma and the theorem on query answering for satisfiable $\textit{DL-Lite}_{\textit{core}}^{(\mathcal{FH})}$ ontologies, we get the following result.

Theorem

Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a $\textit{DL-Lite}_{\textit{core}}^{(\mathcal{FH})}$ ontology, and $\mathcal{T}_P$ the set of PIs in $\mathcal{T}$.
Then, $\mathcal{O}$ is unsatisfiable iff one of the following condition holds:

(a) There exists a NI $N \in \mathcal{T}$ s.t. $\textit{Eval}_{\text{CWA}}(\textit{PerfectRef}(q_N, \mathcal{T}_P), \mathcal{A})$ returns *true*.

(b) There exists a func. assertion $F \in \mathcal{T}$ s.t. $\textit{Eval}_{\text{CWA}}(q_F, \mathcal{A})$ returns *true*.

Note: All the queries $q_N()$ and $q_F()$ can be combined into a single UCQ.
Hence, satisfiability of a $\textit{DL-Lite}_{\textit{core}}^{(\mathcal{FH})}$ ontology is reduced to evaluating a FOL-query over an ontology whose TBox consists of positive inclusions only (and hence is satisfiable).

Outline of Lecture 3

Complexity of query answering over satisfiable ontologies

Theorem [Calvanese *et al.*, 2007]

Query answering over $DL\text{-}Lite_{core}^{(\mathcal{FH})}$ ontologies is

1. NP-**complete** in the size of **query and ontology** (combined complexity).
2. PTime in the size of the **ontology** (schema+data complexity).
3. AC^0 in the size of the **ABox** (data complexity).

Proof (sketch).

1. **Guess** together the derivation of one of the CQs of the perfect rewriting, and an assignment to its existential variables. Checking the derivation and evaluating the guessed CQ over the ABox is then polynomial in combined complexity. NP-hardness follows from combined complexity of evaluating CQs over a database.
2. The number of CQs in the perfect rewriting is polynomial in the size of the TBox, and we can compute them in PTime.
3. Is the data complexity of evaluating FOL queries over a DB. □

Complexity of ontology satisfiability

Theorem [Calvanese *et al.*, 2007]

Checking **satisfiability** of *DL-Lite*$_{core}^{(\mathcal{FH})}$ ontologies is

1. $\textsc{PTime}$ in the size of the **ontology** (combined complexity).
2. AC^0 in the size of the **ABox** (data complexity).

Proof (sketch).

We observe that all the queries $q_N()$ and $q_F()$ checking for violations of negative inclusions N and functionality assertions F can be combined into a single UCQ whose size is linear in the TBOx, and does not depend on the ABox. Hence, the result follows directly from the complexity of query answering over satisfiable ontologies. □

Complexity of TBox reasoning

Theorem [Calvanese *et al.*, 2007]

TBox reasoning over $DL\text{-}Lite_{core}^{(\mathcal{FH})}$ ontologies is PTIME in the size of the **TBox** (schema complexity).

Proof (sketch).

Follows from the previous theorem, and from the fact that all TBox reasoning tasks can be reduced to ontology satisfiability.
Indeed, the size of the ontology constructed in the reduction is polynomial in the size of the input TBox. $\square$

Outline of Lecture 3

The QuOnto/Mastro system

`http://www.dis.uniroma1.it/~quonto/`

- QuOnto is an engine implementing reasoning and query answering over ontologies of the *DL-Lite* family.
- QuOnto is the core component of the Mastro system for ontology-based data access and integration (mapping management).
- Implements all basic TBox+ABox reasoning services, and UCQ answering.
- Includes also support for advanced features:
  - Constraints: identification path constraints, denial constraints, epistemic constraints (EQL-Lite constraints)
  - Epistemic queries (EQL-Lite on UCQs)
- Reasoning services are highly optimized.
- Can be used with internal and external DBMS (include drivers for Postgres, Oracle, DB2, SQL Server, MySQL).
- Implemented in Java
- QuOnto/Mastro is developed jointly at the Sapienza Univ. of Rome and at the Free Univ. of Bozen-Bolzano.

Outline of Lecture 3

References I

[Artale *et al.*, 2009] Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev.
The *DL-Lite* family and relations.
J. of Artificial Intelligence Research, 36:1–69, 2009.

[Baader *et al.*, 2005] Franz Baader, Sebastian Brandt, and Carsten Lutz.
Pushing the $\mathcal{EL}$ envelope.
In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 364–369, 2005.

[Calvanese *et al.*, 1998] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini.
On the decidability of query containment under constraints.
In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 149–158, 1998.

[Calvanese *et al.*, 2006] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo 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.

References II

[Calvanese *et al.*, 2007] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati.
Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family.
J. of Automated Reasoning, 39(3):385–429, 2007.

[Donini *et al.*, 1994] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf.
Deduction in concept languages: From subsumption to instance checking.
J. of Logic and Computation, 4(4):423–452, 1994.

[Glimm *et al.*, 2007] Birte Glimm, Ian Horrocks, Carsten Lutz, and Ulrike Sattler.
Conjunctive query answering for the description logic $\mathcal{SHIQ}$.
In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, pages 399–404, 2007.

[Grosof *et al.*, 2003] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker.
Description logic programs: Combining logic programs with description logic.
In *Proc. of the 12th Int. World Wide Web Conf. (WWW 2003)*, pages 48–57, 2003.

unibz.it

References III

[Hustadt *et al.*, 2005] Ullrich Hustadt, Boris Motik, and Ulrike Sattler.
Data complexity of reasoning in very expressive description logics.
In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 466–471, 2005.

[Levy and Rousset, 1998] Alon Y. Levy and Marie-Christine Rousset.
Combining Horn rules and description logics in CARIN.
Artificial Intelligence, 104(1–2):165–209, 1998.

[Lutz, 2007] Carsten Lutz.
Inverse roles make conjunctive queries hard.
In *Proc. of the 20th Int. Workshop on Description Logic (DL 2007)*, volume 250 of *CEUR Electronic Workshop Proceedings*, `http://ceur-ws.org/`, pages 100–111, 2007.

[Ortiz *et al.*, 2006] Maria Magdalena Ortiz, Diego Calvanese, and Thomas Eiter.
Characterizing data complexity for conjunctive query answering in expressive description logics.
In *Proc. of the 21st Nat. Conf. on Artificial Intelligence (AAAI 2006)*, pages 275–280, 2006.