Knowledge Representation and Ontologies

Part 4: Ontology Based Data Access

Diego Calvanese, Luciano Serafini

Faculty of Computer Science
Master of Science in Computer Science

A.Y. 2010/2011

Freie Universität Bozen
Libera Università di Bolzano
Free University of Bozen · Bolzano

Overview of the Course

1. Modeling information through ontologies
   1. Introduction to ontologies
   2. Ontology languages
   3. UML class diagrams as FOL ontologies
2. Using logic for knowledge representation
   1. Main components of a logic
   2. Reasoning methods in logics
   3. Exercises on analyzing logics
3. Description Logics
   1. Introduction to DLs
   2. Reasoning in simple DLs
   3. More expressive DLs
   4. Fuzzy DLs
   5. Ontology modularization, integration, and contextualization
4. Ontology based data access
   1. Description Logics for data access
   2. Query answering over databases and ontologies
   3. Linking ontologies to relational data
   4. Reasoning in the *DL-Lite* family
5. Conclusions and references

unibz.it

Part 4.1

Description Logics for data access

Outline of Part 4.1

Outline of Part 4.1

1. Description Logics and UML Class Diagrams
   - UML Class Diagrams as ontology formalisms
   - Reducing reasoning in UML to reasoning in DLs
   - Reducing reasoning in DLs to reasoning in UML
   - Reasoning on UML Class Diagrams

2. The *DL-Lite* family of tractable Description Logics

Outline of Part 4.1

Reasoning on UML Class Diagrams

We have seen that UML class diagrams are in tight correspondence with ontology languages (in fact, they can be viewed as an ontology language).
Let's consider again the two questions we asked before:

1. Can we develop sound, complete, and **terminating** procedures for reasoning on UML Class Diagrams?

- We can exploit the formalization of UML Class Diagrams in **Description Logics**.
- We will see that reasoning on UML Class Diagrams can be done in ExpTime in general (and actually, it can be carried out by current DLs-based systems such as FACT++, PELLET, or RACER-PRO).

2. How hard is it to reason on UML Class Diagrams in general?

- We will see that is is ExpTime-hard in general.
- However, we can single out **interesting fragments** on which to reason efficiently.

DLs vs. UML Class Diagrams

There is a tight correspondence between variants of DLs and UML Class Diagrams [Berardi *et al.*, 2005; Artale *et al.*, 2007].

- We can devise two transformations:
  - one that associates to each UML Class Diagram $\mathcal{D}$ a DL TBox $\mathcal{T}_{\mathcal{D}}$.
  - one that associates to each DL TBox $\mathcal{T}$ a UML Class Diagram $\mathcal{D}_{\mathcal{T}}$.
- The transformations are not model-preserving, but are based on a correspondence between instantiations of the Class Diagram and models of the associated TBox.
- The transformations are **satisfiability-preserving**, i.e., a class C is consistent in $\mathcal{D}$ iff the corresponding concept is satisfiable in $\mathcal{T}$.

Outline of Part 4.1

1. Description Logics and UML Class Diagrams
   - UML Class Diagrams as ontology formalisms
   - Reducing reasoning in UML to reasoning in DLs
   - Reducing reasoning in DLs to reasoning in UML
   - Reasoning on UML Class Diagrams

2. The *DL-Lite* family of tractable Description Logics

Encoding UML Class Diagrams in DLs

The ideas behind the encoding of a UML Class Diagram $\mathcal{D}$ in terms of a DL TBox $\mathcal{T}_{\mathcal{D}}$ are quite natural:

- Each class is represented by an atomic concept.
- Each attribute is represented by a role.
- Each binary association is represented by a role.
- Each non-binary association is reified, i.e., represented as a concept connected to its components by roles.
- Each part of the diagram is encoded by suitable assertions.

Encoding of classes and attributes

- A **UML class** C is represented by an **atomic concept** C
- Each **attribute** a of type T for C is represented by an **atomic role** a.
  - To encode the **typing** of a:

$$\exists a \sqsubseteq C \qquad\qquad \exists a^- \sqsubseteq T$$

  - To encode the **multiplicity** $[m..n]$ of a:

$$C \sqsubseteq (\geq m\, a) \sqcap (\leq n\, a)$$

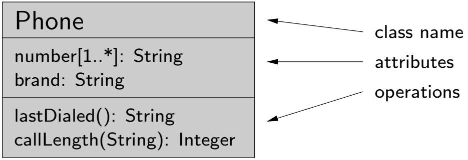
    - When m is 0, we omit the first conjunct.
    - When n is $*$, we omit the second conjunct.
    - When the multiplicity is $[0..*]$ we omit the whole assertion.
    - When the multiplicity is missing (i.e., $[1..1]$), the assertion becomes:

$$C \sqsubseteq \exists a \sqcap (\leq 1\, a)$$

Note: We have assumed that different classes don't share attributes.

- The encoding can be extended also to operations of classes.

Encoding of classes and attributes – Example



- To encode the class Phone, we introduce a concept Phone.
- Encoding of the attributes number and brand:

$$\begin{array}{rcl} \exists \mathsf{number} & \sqsubseteq & \mathsf{Phone} \\ \exists \mathsf{brand} & \sqsubseteq & \mathsf{Phone} \end{array} \qquad \begin{array}{rcl} \exists \mathsf{number}^- & \sqsubseteq & \mathsf{String} \\ \exists \mathsf{brand}^- & \sqsubseteq & \mathsf{String} \end{array}$$

- Encoding of the multiplicities of the attributes number and brand:

$$\begin{array}{rcl} \mathsf{Phone} & \sqsubseteq & \exists \mathsf{number} \\ \mathsf{Phone} & \sqsubseteq & \exists \mathsf{brand} \sqcap (\leq 1\, \mathsf{brand}) \end{array}$$

- We do not consider the encoding of the operations: lastDialed() and callLength(String).

Encoding of associations

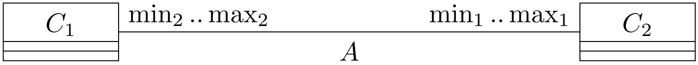
The encoding of associations depends on:

- the presence/absence of an association class;
- the arity of the association.

Arity	Without association class	With association class
Binary	via a DL role	via reification
Non-binary	via reification	via reification

Note: an **aggregation** *is just a particular kind of binary association without association class and is encoded via a DL role.*

Encoding of binary associations without association class



- An association A between C_1 and C_2 is represented by a DL role A, with:

$$\exists A \sqsubseteq C_1 \qquad\qquad \exists A^- \sqsubseteq C_2$$

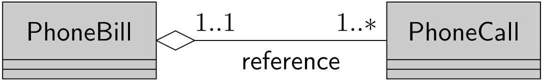
- To encode the multiplicities of A:
  - each instance of class C_1 is connected through association A to at least $\min_1$ and at most $\max_1$ instances of C_2:

$$C_1 \quad \sqsubseteq \quad (\geq \min_1\ A) \sqcap (\leq \max_1\ A)$$

  - each instance of class C_2 is connected through association A^- to at least $\min_2$ and at most $\max_2$ instances of C_1:

$$C_2 \quad \sqsubseteq \quad (\geq \min_2\ A^-) \sqcap (\leq \max_2\ A^-)$$

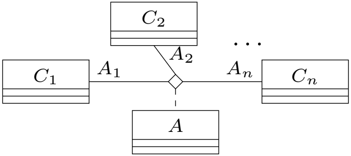
Binary associations without association class – Example

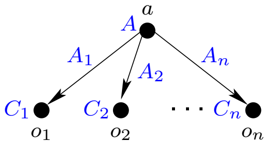


$$
\begin{array}{rcl}
\exists\mathsf{reference} & \sqsubseteq & \mathsf{PhoneBill}\\
\exists\mathsf{reference}^- & \sqsubseteq & \mathsf{PhoneCall}\\
\mathsf{PhoneBill} & \sqsubseteq & (\geq 1\,\mathsf{reference})\\
\mathsf{PhoneCall} & \sqsubseteq & (\geq 1\,\mathsf{reference}^-) \sqcap (\leq 1\,\mathsf{reference}^-)
\end{array}
$$

Note: an aggregation is just a particular kind of binary association without association class.

Encoding of associations via reification





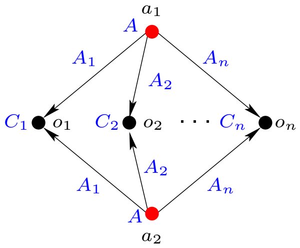
- An association A is represented by a concept A.
- Each instance a of A represents an instance $(o_1, \ldots, o_n)$ of the association.
- n (binary) roles $A_1, \ldots, A_n$ are used to connect an object a representing a tuple to objects $o_1, \ldots, o_n$ representing the components of the tuple.
- To ensure that the instances of A correctly represent tuples:

$$
\begin{array}{rcll}
\exists A_i & \sqsubseteq & A, & \text{for } i \in \{1, \ldots, n\} \\
\exists A_i^- & \sqsubseteq & C_i, & \text{for } i \in \{1, \ldots, n\} \\
A & \sqsubseteq & \exists A_1 \sqcap \cdots \sqcap \exists A_n \sqcap (\leq 1\, A_1) \sqcap \cdots \sqcap (\leq 1\, A_n) &
\end{array}
$$

Note: when the roles of A are explicitly named in the class diagram, we can use such role names instead of $A_1, \ldots, A_n$.

Encoding of associations via reification

We have not ruled out the existence of two instances a_1, a_2 of concept A representing the same instance $(o_1, \ldots, o_n)$ of association A:



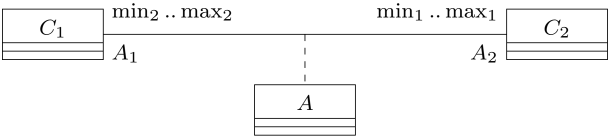
To rule out such a situation we could add an identification assertion (see later):

$(\textbf{id}\ A\ A_1, \ldots, A_n)$

Note: in a **tree-model** *the above situation cannot occur.*

$\leadsto$ By the tree-model property of DLs, when reasoning on a KB, we can restrict the attention to tree-models.
Hence we **can ignore the identification assertions**.

Multiplicities of binary associations with association class



We can encode the multiplicities of association A by means of number restrictions on the inverses of roles A_1 and A_2:

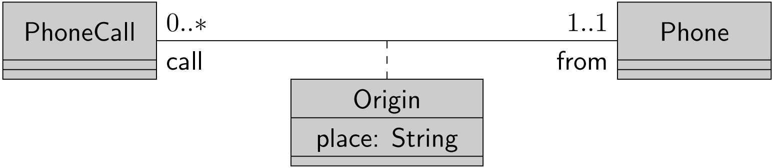
- each instance of class C_1 is connected through association A to at least min_1 and at most max_1 instances of C_2:

$$C_1 \sqsubseteq (\geq \mathrm{min}_1\ A_1^-) \sqcap (\leq \mathrm{max}_1\ A_1^-)$$

- each instance of class C_2 is connected through association A^- to at least min_2 and at most max_2 instances of C_1:

$$C_2 \sqsubseteq (\geq \mathrm{min}_2\ A_2^-) \sqcap (\leq \mathrm{max}_2\ A_2^-)$$

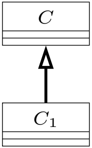
Associations with association class – Example



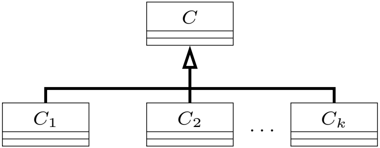
$$
\begin{array}{rcl}
\exists \mathsf{place} & \sqsubseteq & \mathsf{Origin} \\
\mathsf{Origin} & \sqsubseteq & \exists \mathsf{place} \sqcap (\leq 1\,\mathsf{place}) \\
\exists \mathsf{call} & \sqsubseteq & \mathsf{Origin} \\
\exists \mathsf{from} & \sqsubseteq & \mathsf{Origin} \\
\mathsf{Origin} & \sqsubseteq & \exists \mathsf{call} \sqcap (\leq 1\,\mathsf{call}) \sqcap \\
 & & \exists \mathsf{from} \sqcap (\leq 1\,\mathsf{from}) \\
\mathsf{PhoneCall} & \sqsubseteq & (\geq 1\,\mathsf{call}^-) \sqcap (\leq 1\,\mathsf{call}^-)
\end{array}
$$

$$
\begin{array}{rcl}
\exists \mathsf{place}^- & \sqsubseteq & \mathsf{String} \\
\exists \mathsf{call}^- & \sqsubseteq & \mathsf{PhoneCall} \\
\exists \mathsf{from}^- & \sqsubseteq & \mathsf{Phone}
\end{array}
$$

Encoding of ISA and generalization



$$C_1 \sqsubseteq C$$



$$\begin{array}{ccc} C_1 & \sqsubseteq & C \\ & \vdots & \\ C_k & \sqsubseteq & C \end{array}$$

- When the generalization is **disjoint**:

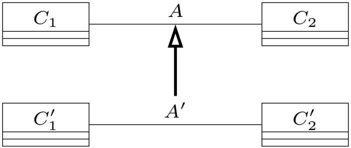
$$C_i \sqsubseteq \neg C_j \qquad \text{for } 1 \leq i < j \leq k$$

- When the generalization is **complete**:

$$C \sqsubseteq C_1 \sqcup \cdots \sqcup C_k$$

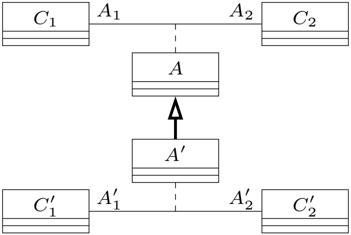
Encoding of ISA between associations

- Without reification:



Role inclusion assertion: $A' \sqsubseteq A$

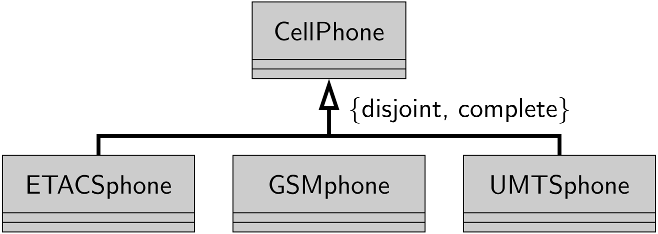
- With reification:



Concept inclusion assert.: $A' \sqsubseteq A$

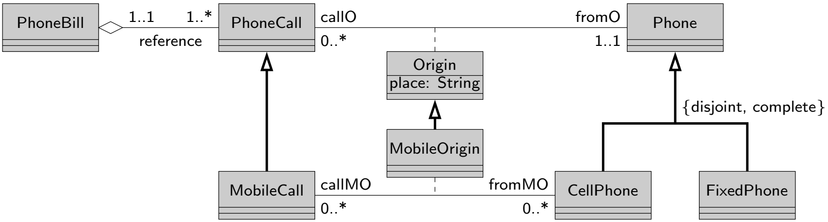
Role inclusion assertions: $A'_1 \sqsubseteq A_1$
$A'_2 \sqsubseteq A_2$

ISA and generalization – Example



$$
\begin{array}{rcl@{\qquad}rcl}
\mathsf{ETACSphone} & \sqsubseteq & \mathsf{CellPhone} & \mathsf{ETACSphone} & \sqsubseteq & \neg\mathsf{GSMPhone} \\
\mathsf{GSMSphone} & \sqsubseteq & \mathsf{CellPhone} & \mathsf{ETACSphone} & \sqsubseteq & \neg\mathsf{UMTSPhone} \\
\mathsf{UMTSSphone} & \sqsubseteq & \mathsf{CellPhone} & \mathsf{GSMphone} & \sqsubseteq & \neg\mathsf{UMTSPhone} \\
\mathsf{CellPhone} & \sqsubseteq & \multicolumn{4}{l}{\mathsf{ETACSphone} \sqcup \mathsf{GSMphone} \sqcup \mathsf{UMTSPhone}}
\end{array}
$$

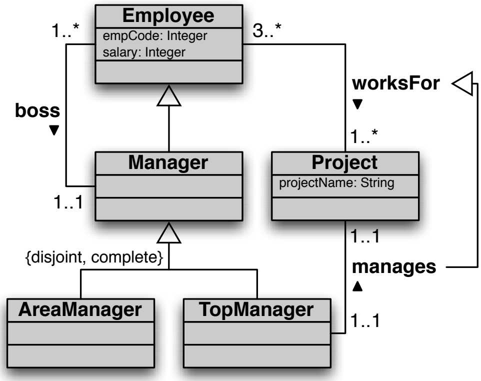
Encoding UML Class Diagrams in DLs – Example



$$
\begin{array}{rcl}
\exists \text{reference} & \sqsubseteq & \text{PhoneBill} \\
\exists \text{reference}^- & \sqsubseteq & \text{PhoneCall} \\
\text{PhoneBill} & \sqsubseteq & (\geq 1\, \text{reference}) \\
\text{PhoneCall} & \sqsubseteq & (\geq 1\, \text{reference}^-) \sqcap \\
 & & (\leq 1\, \text{reference}^-) \\
\exists \text{place} & \sqsubseteq & \text{Origin} \\
\exists \text{place}^- & \sqsubseteq & \text{String} \\
\text{Origin} & \sqsubseteq & \exists \text{place} \sqcap (\leq 1\, \text{place}) \\
\exists \text{callO} & \sqsubseteq & \text{Origin} \\
\exists \text{callO}^- & \sqsubseteq & \text{PhoneCall} \\
\exists \text{fromO} & \sqsubseteq & \text{Origin} \\
\exists \text{fromO}^- & \sqsubseteq & \text{Phone} \\
\text{Origin} & \sqsubseteq & \exists \text{callO} \sqcap (\leq 1\, \text{callO}) \sqcap \\
 & & \exists \text{fromO} \sqcap (\leq 1\, \text{fromO})
\end{array}
$$

$$
\begin{array}{rcl}
\text{PhoneCall} & \sqsubseteq & (\geq 1\, \text{callO}^-) \sqcap (\leq 1\, \text{callO}^-) \\
\exists \text{callMO} & \sqsubseteq & \text{MobileOrigin} \\
\exists \text{callMO}^- & \sqsubseteq & \text{MobileCall} \\
\exists \text{fromMO} & \sqsubseteq & \text{MobileOrigin} \\
\exists \text{fromMO}^- & \sqsubseteq & \text{CellPhone} \\
\text{MobileOrigin} & \sqsubseteq & \exists \text{callMO} \sqcap (\leq 1\, \text{callMO}) \sqcap \\
 & & \exists \text{fromMO} \sqcap (\leq 1\, \text{fromMO}) \\
\text{MobileOrigin} & \sqsubseteq & \text{Origin} \\
\text{callMO} & \sqsubseteq & \text{callO} \\
\text{fromMO} & \sqsubseteq & \text{fromO} \\
\text{MobileCall} & \sqsubseteq & \text{PhoneCall} \\
\text{CellPhone} & \sqsubseteq & \text{Phone} \\
\text{FixedPhone} & \sqsubseteq & \text{Phone} \sqcap \neg\text{CellPhone} \\
\text{Phone} & \sqsubseteq & \text{CellPhone} \sqcup \text{FixedPhone}
\end{array}
$$

Encoding UML Class Diagrams in DLs – Example 2



$$
\begin{array}{rcl}
\mathsf{Manager} & \sqsubseteq & \mathsf{Employee} \\
\mathsf{AreaManager} & \sqsubseteq & \mathsf{Manager} \\
\mathsf{TopManager} & \sqsubseteq & \mathsf{Manager} \\
\mathsf{AreaManager} & \sqsubseteq & \neg\mathsf{TopManager} \\
\mathsf{Manager} & \sqsubseteq & \mathsf{AreaManager} \sqcup \\
 & & \mathsf{TopManager} \\
\exists\mathsf{salary}^- & \sqsubseteq & \mathsf{Integer} \\
\exists\mathsf{salary} & \sqsubseteq & \mathsf{Employee} \\
\mathsf{Employee} & \sqsubseteq & \exists\mathsf{salary} \sqcap (\leq 1\,\mathsf{salary}) \\
\exists\mathsf{worksFor} & \sqsubseteq & \mathsf{Employee} \\
\exists\mathsf{worksFor}^- & \sqsubseteq & \mathsf{Project} \\
\mathsf{Employee} & \sqsubseteq & \exists\mathsf{worksFor} \\
\mathsf{Project} & \sqsubseteq & (\geq 3\,\mathsf{worksFor}^-) \\
\mathsf{manages} & \sqsubseteq & \mathsf{worksFor} \\
 & \ldots &
\end{array}
$$

Outline of Part 4.1

1. Description Logics and UML Class Diagrams
   - UML Class Diagrams as ontology formalisms
   - Reducing reasoning in UML to reasoning in DLs
   - Reducing reasoning in DLs to reasoning in UML
   - Reasoning on UML Class Diagrams
2. The *DL-Lite* family of tractable Description Logics

Reducing reasoning in $\mathcal{ALC}$ to reasoning in UML

We show how to reduce reasoning over $\mathcal{ALC}$ TBoxes to reasoning on UML Class Diagrams:

- We restrict the attention to so-called **primitive $\mathcal{ALC}^-$ TBoxes**, where the concept inclusion assertions have a simplified form:
  - there is a single atomic concept on the left-hand side;
  - there is a single concept constructor on the right-hand side.
- Given a primitive $\mathcal{ALC}^-$ TBox $\mathcal{T}$, we construct a UML Class Diagram $\mathcal{D}_\mathcal{T}$ such that:

  an atomic concept A in $\mathcal{T}$ is satisfiable
  iff
  the corresponding class A in $\mathcal{D}_\mathcal{T}$ is satisfiable.

Note: We preserve satisfiability, but do not have a direct correspondence between models of $\mathcal{T}$ and instantiations of $\mathcal{D}_\mathcal{T}$.

Encoding DL TBoxes in UML Class Diagrams

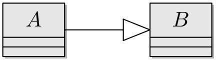
Given a primitive $\mathcal{ALC}^-$ TBox $\mathcal{T}$, we construct $\mathcal{D}_\mathcal{T}$ as follows:

- For each atomic concept A in $\mathcal{T}$, we introduce in $\mathcal{D}_\mathcal{T}$ a class A.
- We introduce in $\mathcal{D}_\mathcal{T}$ an additional class O that generalizes all the classes corresponding to atomic concepts.
- For each atomic role P, we introduce in $\mathcal{D}_\mathcal{T}$:
  - a class C_P (that reifies P);
  - two functional associations P_1, P_2, representing the two components of P.
- For each inclusion assertion in $\mathcal{T}$, we introduce suitable parts of $\mathcal{D}_\mathcal{T}$, as shown in the following.
  We need to encode the following kinds of inclusion assertions:

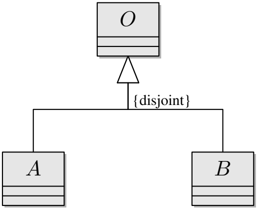
$$\begin{array}{rcl} A & \sqsubseteq & B \\ A & \sqsubseteq & \neg B \\ A & \sqsubseteq & B_1 \sqcup B_2 \end{array} \qquad \begin{array}{rcl} A & \sqsubseteq & \exists P.B \\ A & \sqsubseteq & \forall P.B \end{array}$$

Encoding of inclusion and of disjointness

For each assertion $A \sqsubseteq B$ of $\mathcal{T}$, add the following to $\mathcal{D}_\mathcal{T}$:

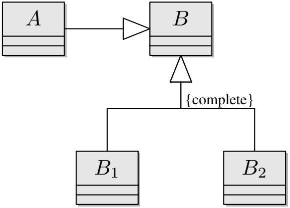


For each assertion $A \sqsubseteq \neg B$ of $\mathcal{T}$, add the following to $\mathcal{D}_\mathcal{T}$:



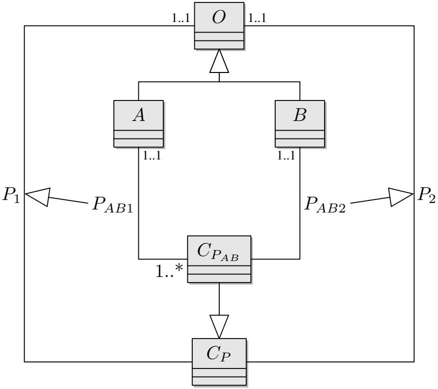
Encoding of union

For each assertion $A \sqsubseteq B_1 \sqcup B_2$ of $\mathcal{T}$, introduce an *auxiliary* class B, and add the following to $\mathcal{D}_\mathcal{T}$:



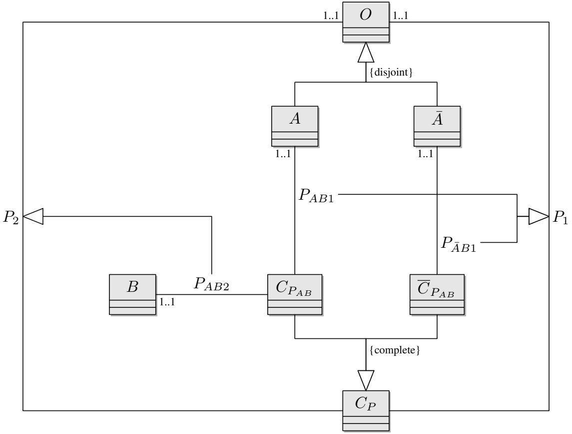
Encoding of existential quantification

For each assertion $A \sqsubseteq \exists P.B$ of $\mathcal{T}$, introduce the auxiliary class $C_{P_{AB}}$ and the associations P_{AB1} and P_{AB2}, and add the following to $\mathcal{D}_{\mathcal{T}}$:



Encoding of universal quantification

For each assertion $A \sqsubseteq \forall P.B$ of $\mathcal{T}$, introduce the auxiliary classes $\bar{A}$, $C_{P_{AB}}$, and $\overline{C}_{P_{AB}}$, and the associations P_{AB1}, $P_{\bar{A}B1}$, and P_{AB2}, and add the following to $\mathcal{D}_{\mathcal{T}}$:



Complexity of reasoning on UML Class Diagrams

Lemma

An atomic concept A in a primitive $\mathcal{ALC}^-$ TBox $\mathcal{T}$ is satisfiable if and only if the class A is satisfiable in the UML Class Diagram $\mathcal{D}_\mathcal{T}$.

Reasoning over primitive $\mathcal{ALC}^-$ TBoxes is ExpTime-hard.
From this, we obtain:

Theorem

Reasoning over UML Class Diagrams is **ExpTime-hard**.

Outline of Part 4.1

Reasoning on UML Class Diagrams using DLs

- The two encodings show that DL TBoxes and UML Class Diagrams essentially have the **same computational properties**.
- Hence, reasoning over UML Class Diagrams has the same complexity as reasoning over ontologies in expressive DLs, i.e., EXPTIME-complete.
- This is somewhat surprising, since UML Class Diagrams are so widely used and yet reasoning on them (and hence fully understanding the implication they may give rise to), in general is a computationally very hard task.
- The high complexity is caused by:
  1. the possibility to use disjunction (covering constraints)
  2. the interaction between role inclusions and functionality constraints (maximum 1 cardinality – see encoding of universal and existential quantification)

Note: Without (1) and restricting (2), reasoning becomes simpler [Artale *et al.*, 2007]:

- NLOGSPACE-complete in combined complexity
- in LOGSPACE in data complexity (see later)

Efficient reasoning on UML Class Diagrams

We are interested in using UML Class Diagrams to specify ontologies in the context of ontology-based data access.

Questions

- Which is the right combination of constructs to allow in UML Class Diagrams to be used for OBDA?
- Are there techniques for query answering in this case that can be derived from Description Logics?
- Can query answering be done efficiently in the size of the data?
- If yes, can we leverage relational database technology for query answering?

Outline of Part 4.1

Outline of Part 4.1

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 data 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.
- Captures conceptual modeling formalism.

The *DL-Lite* family provides new foundations for Ontology-Based Data Access.

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

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

- Takes into account the distinction between **objects** and **values**:
  - Objects are elements of an abstract interpretation domain.
  - Values are elements of concrete data types, such as integers, strings, ecc.
  - Values are connected to objects through **attributes** (rather than roles).
- Is equipped with identification assertions.
- Captures most of UML class diagrams and Extended ER diagrams.
- Enjoys nice computational properties, both w.r.t. the traditional reasoning tasks, and w.r.t. query answering (see later).

Outline of Part 4.1

Syntax of the *DL-Lite*$_{\mathcal{A}}$ description language

- Concept expressions: atomic concept A

$$\begin{array}{rcl} B & \longrightarrow & A \mid \exists Q \mid \delta(U) \\ C & \longrightarrow & \top_C \mid B \mid \neg B \end{array}$$

- Role expressions: atomic role P

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

- Value-domain expressions: each T_i is one of the RDF datatypes

$$\begin{array}{rcl} E & \longrightarrow & \rho(U) \\ F & \longrightarrow & \top_D \mid T_1 \mid \cdots \mid T_n \end{array}$$

- Attribute expressions: atomic attribute U

$$V \longrightarrow U \mid \neg U$$

Semantics of *DL-Lite*$_{\mathcal{A}}$ – Objects vs. values

	Objects	Values
Interpretation domain $\Delta^{\mathcal{I}}$	Domain of objects $\Delta_O^{\mathcal{I}}$	Domain of values $\Delta_V^{\mathcal{I}}$
Alphabet Γ of constants	Object constants Γ_O $c^{\mathcal{I}} \in \Delta_O^{\mathcal{I}}$	Value constants Γ_V $d^{\mathcal{I}} = \mathit{val}(d)$ given a priori
Unary predicates	Concept C $C^{\mathcal{I}} \subseteq \Delta_O^{\mathcal{I}}$	RDF datatype T_i $T_i^{\mathcal{I}} \subseteq \Delta_V^{\mathcal{I}}$ given a priori
Binary predicates	Role R $R^{\mathcal{I}} \subseteq \Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}}$	Attribute V $V^{\mathcal{I}} \subseteq \Delta_O^{\mathcal{I}} \times \Delta_V^{\mathcal{I}}$

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

Construct	Syntax	Example	Semantics
top concept	$\top_C$		$\top_C^\mathcal{I} = \Delta_O^\mathcal{I}$
atomic concept	A	Doctor	$A^\mathcal{I} \subseteq \Delta_O^\mathcal{I}$
existential restriction	$\exists Q$	$\exists$child$^-$	$\{o \mid \exists o'.\, (o,o') \in Q^\mathcal{I}\}$
concept negation	$\neg B$	$\neg\exists$child	$\Delta^\mathcal{I} \setminus B^\mathcal{I}$
attribute domain	$\delta(U)$	δ(salary)	$\{o \mid \exists v.\, (o,v) \in U^\mathcal{I}\}$
atomic role	P	child	$P^\mathcal{I} \subseteq \Delta_O^\mathcal{I} \times \Delta_O^\mathcal{I}$
inverse role	P^-	child$^-$	$\{(o,o') \mid (o',o) \in P^\mathcal{I}\}$
role negation	$\neg Q$	$\neg$manages	$(\Delta_O^\mathcal{I} \times \Delta_O^\mathcal{I}) \setminus Q^\mathcal{I}$
top domain	$\top_D$		$\top_D^\mathcal{I} = \Delta_V^\mathcal{I}$
datatype	T_i	`xsd:int`	$\mathit{val}(T_i) \subseteq \Delta_V^\mathcal{I}$
attribute range	$\rho(U)$	ρ(salary)	$\{v \mid \exists o.\, (o,v) \in U^\mathcal{I}\}$
atomic attribute	U	salary	$U^\mathcal{I} \subseteq \Delta_O^\mathcal{I} \times \Delta_V^\mathcal{I}$
attribute negation	$\neg U$	$\neg$salary	$(\Delta_O^\mathcal{I} \times \Delta_V^\mathcal{I}) \setminus U^\mathcal{I}$
object constant	c	`john`	$c^\mathcal{I} \in \Delta_O^\mathcal{I}$
value constant	d	`'john'`	$\mathit{val}(d) \in \Delta_V^\mathcal{I}$

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

TBox assertions can have the following forms:

- Inclusion assertions:

$B \sqsubseteq C$	concept inclusion	$E \sqsubseteq F$	value-domain inclusion
$Q \sqsubseteq R$	role inclusion	$U \sqsubseteq V$	attribute inclusion

- Functionality assertions:

$(\textbf{funct}\ Q)$	role functionality	$(\textbf{funct}\ U)$	attribute functionality

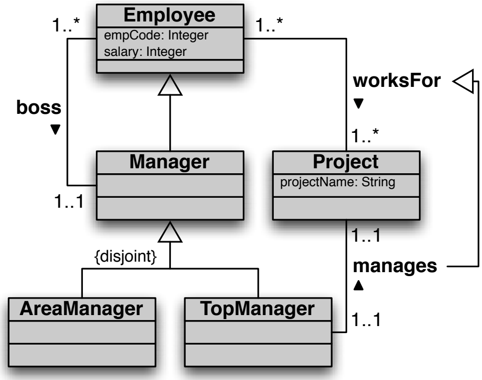
- Identification assertions: $(\textbf{id}\ B\ I_1, \ldots, I_n)$
  where each I_j is a role, an inverse role, or an attribute

ABox assertions: $A(c)$, $P(c, c')$, $U(c, d)$,
where c, c' are object constants and d is a value constant

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

Assertion	Syntax	Example	Semantics
conc. incl.	$B \sqsubseteq C$	Father $\sqsubseteq$ $\exists$child	$B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$
role 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`	$E^{\mathcal{I}} \subseteq F^{\mathcal{I}}$
attr. incl.	$U \sqsubseteq V$	offPhone $\sqsubseteq$ phone	$U^{\mathcal{I}} \subseteq V^{\mathcal{I}}$
role funct.	(**funct** Q)	(**funct** father)	$\forall o, o_1, o_2.(o, o_1) \in Q^{\mathcal{I}} \land (o, o_2) \in Q^{\mathcal{I}} \rightarrow o_1 = o_2$
att. funct.	(**funct** U)	(**funct** ssn)	$\forall o, v, v'.(o, v) \in U^{\mathcal{I}} \land (o, v') \in U^{\mathcal{I}} \rightarrow v = v'$
id const.	(**id** $B\ I_1, \ldots, I_n$)	(**id** Person name, dob)	$I_1, \ldots, I_n$ identify instances of B
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')	$(c^{\mathcal{I}}, \mathit{val}(d)) \in U^{\mathcal{I}}$

DL-Lite$_{\mathcal{A}}$ – Example

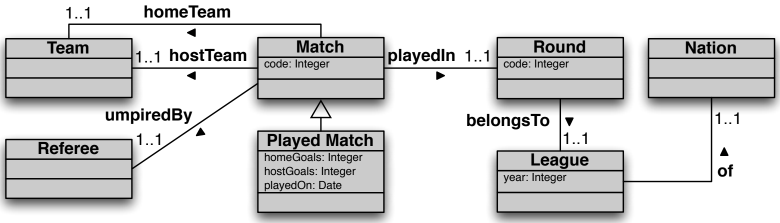


$$
\begin{array}{rcl}
\text{Manager} & \sqsubseteq & \text{Employee} \\
\text{AreaManager} & \sqsubseteq & \text{Manager} \\
\text{TopManager} & \sqsubseteq & \text{Manager} \\
\text{AreaManager} & \sqsubseteq & \neg\text{TopManager} \\
\text{Employee} & \sqsubseteq & \delta(\text{empCode}) \\
\delta(\text{empCode}) & \sqsubseteq & \text{Employee} \\
\rho(\text{empCode}) & \sqsubseteq & \texttt{xsd:int} \\
\multicolumn{3}{c}{(\textbf{funct}\ \text{empCode})} \\
\multicolumn{3}{c}{(\textbf{id}\ \text{Employee empCode})} \\
\exists\text{worksFor} & \sqsubseteq & \text{Employee} \\
\exists\text{worksFor}^- & \sqsubseteq & \text{Project} \\
\text{Employee} & \sqsubseteq & \exists\text{worksFor} \\
\text{Project} & \sqsubseteq & \exists\text{worksFor}^- \\
\multicolumn{3}{c}{(\textbf{funct}\ \text{manages})} \\
\multicolumn{3}{c}{(\textbf{funct}\ \text{manages}^-)} \\
\text{manages} & \sqsubseteq & \text{worksFor} \\
 & \vdots &
\end{array}
$$

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

Outline of Part 4.1

Identification assertions – Example



What we would like to additionally capture:

1. No two leagues with the same year and the same nation exist
2. Within a certain league, the code associated to a round is unique
3. Every match is identified by its code within its round
4. Every referee can umpire at most one match in the same round
5. No team can be the home team of more than one match per round
6. No team can be the host team of more than one match per round

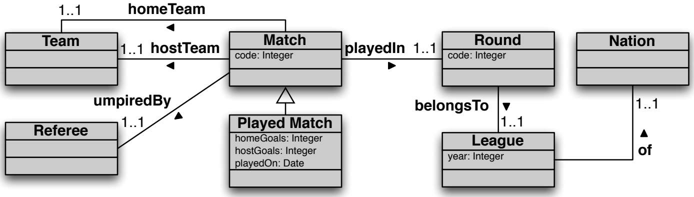
Identification assertions – Example (cont'd)

$\mathsf{League} \sqsubseteq \exists \mathsf{of}$
$\exists \mathsf{of} \sqsubseteq \mathsf{League}$
$\exists \mathsf{of}^- \sqsubseteq \mathsf{Nation}$
$\mathsf{Round} \sqsubseteq \exists \mathsf{belongsTo}$
$\exists \mathsf{belongsTo} \sqsubseteq \mathsf{Round}$
$\exists \mathsf{belongsTo}^- \sqsubseteq \mathsf{League}$
$\mathsf{Match} \sqsubseteq \exists \mathsf{playedIn}$
...

$\mathsf{PlayedMatch} \sqsubseteq \mathsf{Match}$
$\mathsf{Match} \sqsubseteq \delta(\mathsf{code})$
$\mathsf{Round} \sqsubseteq \delta(\mathsf{code})$
$\mathsf{PlayedMatch} \sqsubseteq \delta(\mathsf{playedOn})$
...
$\rho(\mathsf{playedOn}) \sqsubseteq \mathtt{xsd{:}date}$
$\rho(\mathsf{code}) \sqsubseteq \mathtt{xsd{:}int}$
...

(**funct** of)
(**funct** belongsTo)
(**funct** playedIn)
(**funct** homeTeam)
(**funct** hostTeam)
(**funct** umpiredBy)
(**funct** code)
(**funct** year)
(**funct** homeGoals)
(**funct** hostGoals)
(**funct** playedOn)

Identification assertions – Example (cont'd)



1. No two leagues with the same year and the same nation exist
2. Within a certain league, the code associated to a round is unique
3. Every match is identified by its code within its round
4. Every referee can umpire at most one match in the same round
5. No team can be the home team of more than one match per round
6. No team can be the host team of more than one match per round

(**id** League of, year)
(**id** Round belongsTo, code)
(**id** Match playedIn, code)
(**id** Match umpiredBy, playedIn)
(**id** Match homeTeam, playedIn)
(**id** Match hostTeam, playedIn)

Semantics of identification assertions

Let $(\textbf{id}\ B\ I_1, \ldots, I_n)$ be an identification assertion in a *DL-Lite*$_{\mathcal{A}}$ TBox.

An interpretation $\mathcal{I}$ satisfies such an assertion if for all $o_1, o_2 \in B^{\mathcal{I}}$ and for all objects or values $u_1, \ldots, u_n$, we have that

$$(o_1, u_j) \in I_j^{\mathcal{I}} \text{ and } (o_2, u_j) \in I_j^{\mathcal{I}}, \text{ for } j \in \{1, \ldots, n\}, \text{ implies that } o_1 = o_2.$$

In other words, the instance o_i of B is identified by the tuple $(u_1, \ldots, u_n)$ of objects or values to which it is connected via $I_1, \ldots, I_n$, respectively.

Note: the roles or attributes I_j are not required to be functional or mandatory.

The above definition of semantics implies that, in the case where an instance $o \in B^{\mathcal{I}}$ is connected by means of $I_j^{\mathcal{I}}$ to a set $u_j^1, \ldots, u_j^k$ of objects (or values), it is each single u_j^h that contributes to the identification of o, and not the whole set $\{u_j^1, \ldots, u_j^k\}$.

Outline of Part 4.1

1. Description Logics and UML Class Diagrams

2. The *DL-Lite* family of tractable Description Logics
   - Basic features of *DL-Lite*
   - Syntax and semantics of *DL-Lite*
   - Identification assertions in *DL-Lite*
   - Members of the *DL-Lite* family
   - Properties of *DL-Lite*

Restriction on TBox assertions in *DL-Lite*$_{\mathcal{A}}$ ontologies

We will see that, to ensure the good computational properties that we aim at, we have to impose a **restriction** on the use of functionality and role/attribute inclusions.

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

No functional or identifying role or attribute can be specialized by using it in the right-hand side of a role or attribute inclusion assertion.

Formally:

- If (**funct** P), (**funct** P^-), (**id** $B\ \ldots, P, \ldots$), or (**id** $B\ \ldots, P^-, \ldots$) is in $\mathcal{T}$, then $Q \sqsubseteq P$ and $Q \sqsubseteq P^-$ are **not in** $\mathcal{T}$.
- If (**funct** U) or (**id** $B\ \ldots, U, \ldots$) is in $\mathcal{T}$, then $U' \sqsubseteq U$ is **not in** $\mathcal{T}$.

DL-Lite$_{\mathcal{F}}$ and *DL-Lite*$_{\mathcal{R}}$

We consider also two sub-languages of *DL-Lite*$_{\mathcal{A}}$ (that trivially obey the previous restriction):

- *DL-Lite*$_{\mathcal{F}}$: Allows for functionality assertions, but does not allow for role inclusion assertions.
- *DL-Lite*$_{\mathcal{R}}$: Allows for role inclusion assertions, but does not allow for functionality assertions.

In both *DL-Lite*$_{\mathcal{F}}$ and *DL-Lite*$_{\mathcal{R}}$ we do not consider data values (and hence drop value domains and attributes).

Note: We simply use *DL-Lite* to refer to any of the logics of the *DL-Lite* family.

Outline of Part 4.1

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

ISA between classes	$A_1 \sqsubseteq A_2$
Disjointness between classes	$A_1 \sqsubseteq \neg A_2$
Mandatory participation to relations	$A_1 \sqsubseteq \exists P \quad A_2 \sqsubseteq \exists P^-$
Domain and range of relations	$\exists P \sqsubseteq A_1 \quad \exists P^- \sqsubseteq A_2$
Functionality of relations	$(\textbf{funct}\ P) \quad (\textbf{funct}\ P^-)$
ISA between relations	$Q_1 \sqsubseteq Q_2$
Disjointness between relations	$Q_1 \sqsubseteq \neg Q_2$
Domain and range of attributes	$\delta(U) \sqsubseteq A \quad \rho(U) \sqsubseteq T_i$
Mandatory and functional attributes	$A \sqsubseteq \delta(U) \quad (\textbf{funct}\ U)$
Identification constraints	$(\textbf{id}\ A\ P, \ldots, P'^-, \ldots, U, \ldots)$

Properties of *DL-Lite*

- The TBox may contain **cyclic dependencies** (which typically increase the computational complexity of reasoning).

  Example: $A \sqsubseteq \exists P$, $\exists P^- \sqsubseteq A$

- In the syntax, we have not included $\sqcap$ on the right hand-side of inclusion assertions, but it can trivially be added, since

$$B \sqsubseteq C_1 \sqcap C_2 \quad \text{is equivalent to} \quad \begin{array}{lll} B & \sqsubseteq & C_1 \\ B & \sqsubseteq & C_2 \end{array}$$

- A domain assertion on role P has the form: $\exists P \sqsubseteq A_1$
  A range assertion on role P has the form: $\exists P^- \sqsubseteq A_2$

Properties of *DL-Lite*$_{\mathcal{F}}$

DL-Lite$_{\mathcal{F}}$ does **not** enjoy the **finite model property**.

Example

TBox $\mathcal{T}$: $\text{Nat} \sqsubseteq \exists\text{succ}$ $\qquad \exists\text{succ}^- \sqsubseteq \text{Nat}$

$\text{Zero} \sqsubseteq \text{Nat} \sqcap \neg\exists\text{succ}^-$ $\qquad (\textbf{funct}\ \text{succ}^-)$

ABox $\mathcal{A}$: $\text{Zero}(0)$

$\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ admits only infinite models.
Hence, it is satisfiable, but **not finitely satisfiable**.

Hence, reasoning w.r.t. arbitrary models is different from reasoning w.r.t. finite models only.

Properties of *DL-Lite*$_{\mathcal{R}}$

- *DL-Lite*$_{\mathcal{R}}$ **does enjoy the finite model property**. Hence, reasoning w.r.t. finite models is the same as reasoning w.r.t. arbitrary models.
- With role inclusion assertions, we can simulate **qualified existential quantification** in the rhs of an inclusion assertion $A_1 \sqsubseteq \exists Q.A_2$.

  To do so, we introduce a new role Q_{A_2} and:
  - the role inclusion assertion $Q_{A_2} \sqsubseteq Q$
  - the concept inclusion assertions: $$\begin{array}{rcl} A_1 & \sqsubseteq & \exists Q_{A_2} \\ \exists Q_{A_2}^- & \sqsubseteq & A_2 \end{array}$$

  In this way, we can consider $\exists Q.A$ in the right-hand side of an inclusion assertion as an abbreviation.

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.
- Extends (the DL fragment of) the ontology language **RDFS**.
- Is completely symmetric w.r.t. **direct and inverse properties**.
- Is at the basis of the **OWL2 QL** profile of OWL2.

The OWL2 QL Profile

OWL2 defines three **profiles**: OWL2 QL, OWL2 EL, OWL2 RL.

- Each profile corresponds to a syntactic fragment (i.e., a sub-language) of OWL2 DL that is targeted towards a specific use.
- The restrictions in each profile guarantee better computational properties than those of OWL2 DL.

The **OWL2 QL** profile is derived from the DLs of the *DL-Lite* family:

- "[It] includes most of the main features of conceptual models such as UML class diagrams and ER diagrams."
- "[It] is aimed at applications that use very large volumes of instance data, and where query answering is the most important reasoning task. In OWL2 QL, conjunctive query answering can be implemented using conventional relational database systems."

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

1. We have seen that *DL-Lite*$_\mathcal{A}$ can capture the essential features of prominent conceptual modeling formalisms.
2. In the following, we will analyze reasoning in *DL-Lite*, and establish the following characterization of its computational properties:
   - Ontology satisfiability and all classical DL reasoning tasks are:
     - Efficiently tractable in the size of the TBox (i.e., PTime).
     - Very efficiently tractable in the size of the ABox (i.e., AC^0).
   - Query answering for CQs and UCQs is:
     - PTime in the size of the TBox.
     - AC^0 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.
3. We will also see that *DL-Lite* is essentially the maximal DL enjoying these nice computational properties.

From (1), (2), and (3) we get that:

DL-Lite is a representation formalism that is very well suited to underlie ontology-based data management systems.

Part 4.2

Query answering over databases and ontologies

Outline of Part 4.2

3. Query answering in databases
   - First-order logic queries
   - Query evaluation problem
   - Conjunctive queries and homomorphisms
   - Unions of conjunctive queries

4. Querying databases and ontologies
   - Query answering in traditional databases
   - Query answering in ontologies
   - Query answering in ontology-based data access

5. Query answering in Description Logics
   - Queries over Description Logics ontologies
   - Certain answers
   - Complexity of query answering

Outline of Part 4.2

Outline of Part 4.2

Queries

- A **query** is a mechanism to extract new information from given information stored in some form. The extracted information is called the **answer** to the query.
- In the most general sense, a query is an arbitrary (computable) function, from some input to some output.
- Typically, one is interested in queries expressed in some (restricted) query language that provides guarantees on the computational properties of computing answers to queries.
- Here we consider queries that:
  - are expressed over a relational alphabet, and
  - return as result a relation, i.e., a set of tuples of objects satisfying a certain condition.

A very prominent query language of this form is **first-order logic**.

First-order logic

We consider now first-order logic with equality (FOL) as a mechanism to express queries.

- FOL is the logic to speak about **objects**, which constitute the domain of discourse (or universe).
- FOL is concerned about **properties** of these objects and **relations** over objects (corresponding to unary and n-ary **predicates**, respectively).
- FOL also has **functions**, including **constants**, that denote objects.

FOL syntax – Terms

We first introduce:

- A set $Vars = \{x_1, \ldots, x_n\}$ of **individual variables** (i.e., variables that denote single objects).
- A set of **functions symbols**, each of given arity ≥ 0.
  Functions of arity 0 are called constants.

Def.: The set of $Terms$ is defined inductively as follows:

- Each variable is a term, i.e., $Vars \subseteq Terms$;
- If $t_1, \ldots, t_k \in Terms$ and f^k is a k-ary function symbol, then $f^k(t_1, \ldots, t_k) \in Terms$;
- Nothing else is in $Terms$.

FOL syntax – Formulas

Def.: The set of *Formulas* is defined inductively as follows:

- If $t_1, \ldots, t_k \in \mathit{Terms}$ and P^k is a k-ary predicate, then $P^k(t_1, \ldots, t_k) \in \mathit{Formulas}$ (atomic formulas).
- If $t_1, t_2 \in \mathit{Terms}$, then $t_1 = t_2 \in \mathit{Formulas}$.
- If $\varphi \in \mathit{Formulas}$ and $\psi \in \mathit{Formulas}$ then
  - $\neg\varphi \in \mathit{Formulas}$
  - $\varphi \land \psi \in \mathit{Formulas}$
  - $\varphi \lor \psi \in \mathit{Formulas}$
  - $\varphi \rightarrow \psi \in \mathit{Formulas}$
- If $\varphi \in \mathit{Formulas}$ and $x \in \mathit{Vars}$ then
  - $\exists x.\varphi \in \mathit{Formulas}$
  - $\forall x.\varphi \in \mathit{Formulas}$
- Nothing else is in $\mathit{Formulas}$.

Note: a predicate of arity 0 is a proposition (as in propositional logic).

Interpretations

Given an **alphabet** of predicates $P_1, P_2, \ldots$ and function symbols $f_1, f_2, \ldots$, each with an associated arity, a FOL **interpretation** is:

$$\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$$

where:

- $\Delta^\mathcal{I}$ is the interpretation domain (a set of objects);
- $\cdot^\mathcal{I}$ is the interpretation function that interprets predicates and function symbols as follows:
  - if P_i is a k-ary predicate, then $P_i^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \cdots \times \Delta^\mathcal{I}$ (k times)
  - if f_i is a k-ary function, $k \geq 1$, then $f_i^\mathcal{I} : \Delta^\mathcal{I} \times \cdots \times \Delta^\mathcal{I} \longrightarrow \Delta^\mathcal{I}$
  - if f_i is a constant (i.e., a 0-ary function), then $f_i^\mathcal{I} : () \longrightarrow \Delta^\mathcal{I}$ (i.e., f_i denotes exactly one object of the domain)

Assignment

Let $Vars$ be a set of (individual) variables.

Def.: Given an interpretation $\mathcal{I}$, an **assignment** is a function

$$\alpha : Vars \longrightarrow \Delta^{\mathcal{I}}$$

that assigns to each variable $x \in Vars$ an object $\alpha(x) \in \Delta^{\mathcal{I}}$.

It is convenient to extend the notion of assignment to terms. We can do so by defining a function $\hat{\alpha} : Terms \longrightarrow \Delta^{\mathcal{I}}$ inductively as follows:

- $\hat{\alpha}(x) = \alpha(x)$, if $x \in Vars$
- $\hat{\alpha}(f(t_1, \ldots, t_k)) = f^{\mathcal{I}}(\hat{\alpha}(t_1), \ldots, \hat{\alpha}(t_k))$

Note: for constants $\hat{\alpha}(c) = c^{\mathcal{I}}$.

Truth in an interpretation wrt an assignment

We define when a FOL formula φ is **true** in an interpretation $\mathcal{I}$ wrt an assignment α, written $\mathcal{I}, \alpha \models \varphi$:

$$
\begin{array}{ll}
\mathcal{I}, \alpha \models P(t_1, \ldots, t_k), & \text{if } (\hat{\alpha}(t_1), \ldots, \hat{\alpha}(t_k)) \in P^{\mathcal{I}} \\
\mathcal{I}, \alpha \models t_1 = t_2, & \text{if } \hat{\alpha}(t_1) = \hat{\alpha}(t_2) \\
\mathcal{I}, \alpha \models \neg\varphi, & \text{if } \mathcal{I}, \alpha \not\models \varphi \\
\mathcal{I}, \alpha \models \varphi \land \psi, & \text{if } \mathcal{I}, \alpha \models \varphi \text{ and } \mathcal{I}, \alpha \models \psi \\
\mathcal{I}, \alpha \models \varphi \lor \psi, & \text{if } \mathcal{I}, \alpha \models \varphi \text{ or } \mathcal{I}, \alpha \models \psi \\
\mathcal{I}, \alpha \models \varphi \rightarrow \psi, & \text{if } \mathcal{I}, \alpha \models \varphi \text{ implies } \mathcal{I}, \alpha \models \psi \\
\mathcal{I}, \alpha \models \exists x.\varphi, & \text{if for some } a \in \Delta^{\mathcal{I}} \text{ we have } \mathcal{I}, \alpha[x \mapsto a] \models \varphi \\
\mathcal{I}, \alpha \models \forall x.\varphi, & \text{if for every } a \in \Delta^{\mathcal{I}} \text{ we have } \mathcal{I}, \alpha[x \mapsto a] \models \varphi
\end{array}
$$

Here, $\alpha[x \mapsto a]$ stands for the new assignment obtained from α as follows:

$$
\begin{array}{ll}
\alpha[x \mapsto a](x) = a & \\
\alpha[x \mapsto a](y) = \alpha(y), & \text{for } y \neq x
\end{array}
$$

Note: we have assumed that variables are standardized apart.

Open vs. closed formulas

Definitions

- A variable x in a formula φ is **free** if x does not occur in the scope of any quantifier, otherwise it is **bound**.
- An **open formula** is a formula that has some free variable.
- A **closed formula**, also called **sentence**, is a formula that has no free variables.

For **closed formulas** (but not for open formulas) we can define what it means to be **true in an interpretation**, written $\mathcal{I} \models \varphi$, without mentioning the assignment, since the assignment α does not play any role in verifying $\mathcal{I}, \alpha \models \varphi$.

Instead, open formulas are strongly related to **queries** — cf. relational databases.

FOL queries

Def.: A **FOL query** is an (open) FOL formula.

When φ is a FOL query with free variables $(x_1, \ldots, x_k)$, then we sometimes write it as $\varphi(x_1, \ldots, x_k)$, and say that φ has **arity** k.

Given an interpretation $\mathcal{I}$, we are interested in those assignments that map the variables $x_1, \ldots, x_k$ (and only those).
We write an assignment α s.t. $\alpha(x_i) = a_i$, for $i = 1, \ldots, k$, as $\langle a_1, \ldots, a_k \rangle$.

Def.: Given an interpretation $\mathcal{I}$, the **answer to a query** $\varphi(x_1, \ldots, x_k)$ is

$$\varphi(x_1, \ldots, x_k)^{\mathcal{I}} = \{(a_1, \ldots, a_k) \mid \mathcal{I}, \langle a_1, \ldots, a_k \rangle \models \varphi(x_1, \ldots, x_k)\}$$

Note: We will also use the notation $\varphi^{\mathcal{I}}$, which keeps the free variables implicit, and $\varphi(\mathcal{I})$ making apparent that φ becomes a functions from interpretations to set of tuples.

FOL boolean queries

Def.: A **FOL boolean query** is a FOL query without free variables.

Hence, the answer to a boolean query $\varphi()$ is defined as follows:

$$\varphi()^{\mathcal{I}} = \{() \mid \mathcal{I}, \langle\rangle \models \varphi()\}$$

Such an answer is

- the empty tuple $()$, if $\mathcal{I} \models \varphi$
- the empty set $\emptyset$, if $\mathcal{I} \not\models \varphi$.

As an obvious convention we read $()$ as "true" and $\emptyset$ as "false".

FOL formulas: logical tasks

Definitions

- Validity: φ is **valid** iff for all $\mathcal{I}$ and α we have that $\mathcal{I}, \alpha \models \varphi$.
- Satisfiability: φ is **satisfiable** iff there exists an $\mathcal{I}$ and α such that $\mathcal{I}, \alpha \models \varphi$, and unsatisfiable otherwise.
- Logical implication: φ **logically implies** ψ, written $\varphi \models \psi$ iff for all $\mathcal{I}$ and α, if $\mathcal{I}, \alpha \models \varphi$ then $\mathcal{I}, \alpha \models \psi$.
- Logical equivalence: φ is **logically equivalent** to ψ, iff for all $\mathcal{I}$ and α, we have that $\mathcal{I}, \alpha \models \varphi$ iff $\mathcal{I}, \alpha \models \psi$ (i.e., $\varphi \models \psi$ and $\psi \models \varphi$).

FOL queries – Logical tasks

- Validity: if φ is valid, then $\varphi^{\mathcal{I}} = \Delta^{\mathcal{I}} \times \cdots \times \Delta^{\mathcal{I}}$ for all $\mathcal{I}$, i.e., the query always returns all the tuples of $\mathcal{I}$.
- Satisfiability: if φ is satisfiable, then $\varphi^{\mathcal{I}} \neq \emptyset$ for some $\mathcal{I}$, i.e., the query returns at least one tuple.
- Logical implication: if φ logically implies ψ, then $\varphi^{\mathcal{I}} \subseteq \psi^{\mathcal{I}}$ for all $\mathcal{I}$, written $\varphi \subseteq \psi$, i.e., the answer to φ is contained in that of ψ in every interpretation. This is called **query containment**.
- Logical equivalence: if φ is logically equivalent to ψ, then $\varphi^{\mathcal{I}} = \psi^{\mathcal{I}}$ for all $\mathcal{I}$, written $\varphi \equiv \psi$, i.e., the answer to the two queries is the same in every interpretation. This is called **query equivalence** and corresponds to query containment in both directions.

Note: These definitions can be extended to the case where we have **axioms**, i.e., **constraints** on the admissible interpretations.

Outline of Part 4.2

3. Query answering in databases
   - First-order logic queries
   - Query evaluation problem
   - Conjunctive queries and homomorphisms
   - Unions of conjunctive queries
4. Querying databases and ontologies
5. Query answering in Description Logics

Query evaluation

Let us consider a **finite interpretation** $\mathcal{I}$, i.e., an interpretation (over the finite alphabet) for which $\Delta^{\mathcal{I}}$ is finite.

Note: whenever we have to evaluate a query, we are only interested in the interpretation of the relation and function symbols that appear in the query, which are **finitely many**.

Then we can consider query evaluation as an algorithmic problem, and study its computational properties.

Note: To study the **computational complexity** of the problem, we need to define a corresponding decision problem.

Query evaluation problem

Definitions

- **Query answering problem**: given a finite interpretation $\mathcal{I}$ and a FOL query $\varphi(x_1, \ldots, x_k)$, compute

$$\varphi^{\mathcal{I}} = \{(a_1, \ldots, a_k) \mid \mathcal{I}, \langle a_1, \ldots, a_k \rangle \models \varphi(x_1, \ldots, x_k)\}$$

- **Recognition problem (for query answering)**: given a finite interpretation $\mathcal{I}$, a FOL query $\varphi(x_1, \ldots, x_k)$, and a tuple $(a_1, \ldots, a_k)$, with $a_i \in \Delta^{\mathcal{I}}$, check whether $(a_1, \ldots, a_k) \in \varphi^{\mathcal{I}}$, i.e., whether

$$\mathcal{I}, \langle a_1, \ldots, a_k \rangle \models \varphi(x_1, \ldots, x_k)$$

Note: The recognition problem for query answering is the decision problem corresponding to the query answering problem.

Query evaluation algorithm

We define now an algorithm that computes the function $\texttt{Truth}(\mathcal{I}, \alpha, \varphi)$ in such a way that $\texttt{Truth}(\mathcal{I}, \alpha, \varphi) = \texttt{true}$ iff $\mathcal{I}, \alpha \models \varphi$.

We make use of an auxiliary function $\texttt{TermEval}(\mathcal{I}, \alpha, t)$ that, given an interpretation $\mathcal{I}$ and an assignment α, evaluates a term t returning an object $o \in \Delta^{\mathcal{I}}$:

```
Δ^I TermEval(I,α,t) {
   if (t is x ∈ Vars)
      return α(x);
   if (t is f(t_1,...,t_k))
      return f^I(TermEval(I,α,t_1),...,TermEval(I,α,t_k));
}
```

Note: constants are considered as function symbols of arity 0

Then, $\texttt{Truth}(\mathcal{I}, \alpha, \varphi)$ can be defined by structural recursion on φ.

Query evaluation algorithm (cont'd)

```
boolean Truth(𝓘,α,φ) {
  if (φ is t_1 = t_2)
    return TermEval(𝓘,α,t_1) = TermEval(𝓘,α,t_2);
  if (φ is P(t_1,...,t_k))
    return P^𝓘(TermEval(𝓘,α,t_1),...,TermEval(𝓘,α,t_k));
  if (φ is ¬ψ)
    return ¬Truth(𝓘,α,ψ);
  if (φ is ψ ∘ ψ')
    return Truth(𝓘,α,ψ) ∘ Truth(𝓘,α,ψ');
  if (φ is ∃x.ψ) {
    boolean b = false;
    for all (a ∈ Δ^𝓘)
       b = b ∨ Truth(𝓘,α[x ↦ a],ψ);
    return b;
  }
  if (φ is ∀x.ψ) {
    boolean b = true;
    for all (a ∈ Δ^𝓘)
       b = b ∧ Truth(𝓘,α[x ↦ a],ψ);
    return b;
  }
}
```

Query evaluation – Results

Theorem (Termination of $\texttt{Truth}(\mathcal{I}, \alpha, \varphi)$)

The algorithm Truth terminates.

Proof. Immediate. □

Theorem (Correctness)

The algorithm Truth is sound and complete, i.e., $\mathcal{I}, \alpha \models \varphi$ if and only if $\texttt{Truth}(\mathcal{I}, \alpha, \varphi) = \texttt{true}$.

Proof. Easy, since the structure of the algorithm directly reflects the inductive definition of $\mathcal{I}, \alpha \models \varphi$. □

Query evaluation – Time complexity I

Theorem (Time complexity of $\texttt{Truth}(\mathcal{I}, \alpha, \varphi)$)

The time complexity of $\texttt{Truth}(\mathcal{I}, \alpha, \varphi)$ is $(|\mathcal{I}| + |\alpha| + |\varphi|)^{|\varphi|}$, i.e., polynomial in the size of $\mathcal{I}$ and exponential in the size of φ.

Proof.

- Each $f^{\mathcal{I}}$ (of arity k) can be represented as a k-dimensional array, hence accessing the required element can be done in time linear in $|\mathcal{I}|$.
- $\texttt{TermEval}(\ldots)$ visits the term, so it generates a polynomial number of recursive calls, hence runs in time polynomial in $(|\mathcal{I}| + |\alpha| + |\varphi|)$.
- Each $P^{\mathcal{I}}$ (of arity k) can be represented as a k-dimensional boolean array, hence accessing the required element can be done in time linear in $|\mathcal{I}|$.

Query evaluation – Time complexity II

- $\texttt{Truth}(\ldots)$ for the boolean cases simply visits the formula, so generates either one or two recursive calls.
- $\texttt{Truth}(\ldots)$ for the quantified cases $\exists x.\varphi$ and $\forall x.\psi$ involves looping for all elements in $\Delta^{\mathcal{I}}$ and testing the resulting assignments.
- The total number of such tests is $O(|\mathcal{I}|^{\sharp Vars})$.

Hence the claim holds. $\square$

Query evaluation – Space complexity I

Theorem (Space complexity of $\texttt{Truth}(\mathcal{I}, \alpha, \varphi)$)

The space complexity of $\texttt{Truth}(\mathcal{I}, \alpha, \varphi)$ is $|\varphi| \cdot (|\varphi| \cdot \log |\mathcal{I}|)$, i.e., logarithmic in the size of $\mathcal{I}$ and polynomial in the size of φ.

Proof.

- Each $f^{\mathcal{I}}(\ldots)$ can be represented as a k-dimensional array, hence accessing the required element requires $O(\log |\mathcal{I}|)$ space.
- $\texttt{TermEval}(\ldots)$ simply visits the term, so it generates a polynomial number of recursive calls. Each activation record has $O(\log |\mathcal{I}|)$ size, and we need $O(|\varphi|)$ activation records.
- Each $P^{\mathcal{I}}(\ldots)$ can be represented as a k-dimensional boolean array, hence accessing the required element requires $O(\log |\mathcal{I}|)$ space.

Query evaluation – Space complexity II

- Truth(…) for the boolean cases simply visits the formula, so generates either one or two recursive calls, each requiring constant space.
- Truth(…) for the quantified cases $\exists x.\varphi$ and $\forall x.\psi$ involves looping for all elements in $\Delta^{\mathcal{I}}$ and testing the resulting assignments.
- The total number of activation records that need to be at the same time on the stack is $O(\sharp Vars) \leq O(|\varphi|)$.

Hence the claim holds. □

Note: the worst case form for the formula is

$$Q_1 x_1 . Q_2 x_2 . \cdots Q_n x_n . P(x_1, x_2, \ldots, x_{n-1}, x_n).$$

where each Q_i is one of $\forall$ or $\exists$.

Query evaluation – Complexity measures [Vardi, 1982]

Definition (Combined complexity)

The **combined complexity** is the complexity of $\{\langle \mathcal{I}, \alpha, \varphi \rangle \mid \mathcal{I}, \alpha \models \varphi\}$, i.e., interpretation, tuple, and query are all considered part of the input.

Definition (Data complexity)

The **data complexity** is the complexity of $\{\langle \mathcal{I}, \alpha \rangle \mid \mathcal{I}, \alpha \models \varphi\}$, i.e., the query φ is fixed (and hence not considered part of the input).

Definition (Query complexity)

The **query complexity** is the complexity of $\{\langle \alpha, \varphi \rangle \mid \mathcal{I}, \alpha \models \varphi\}$, i.e., the interpretation $\mathcal{I}$ is fixed (and hence not considered part of the input).

Query evaluation – Combined, data, query complexity

Theorem (Combined complexity of query evaluation)

The complexity of $\{\langle \mathcal{I}, \alpha, \varphi \rangle \mid \mathcal{I}, \alpha \models \varphi\}$ is:

- time: exponential
- space: $\textsc{PSpace}$-complete — see [Vardi, 1982] for hardness

Theorem (Data complexity of query evaluation)

The complexity of $\{\langle \mathcal{I}, \alpha \rangle \mid \mathcal{I}, \alpha \models \varphi\}$ is:

- time: polynomial
- space: in $\textsc{LogSpace}$

Theorem (Query complexity of query evaluation)

The complexity of $\{\langle \alpha, \varphi \rangle \mid \mathcal{I}, \alpha \models \varphi\}$ is:

- time: exponential
- space: $\textsc{PSpace}$-complete — see [Vardi, 1982] for hardness

Outline of Part 4.2

3. Query answering in databases
   - First-order logic queries
   - Query evaluation problem
   - Conjunctive queries and homomorphisms
   - Unions of conjunctive queries
4. Querying databases and ontologies
5. Query answering in Description Logics

(Union of) Conjunctive queries – (U)CQs

(Unions of) **conjunctive queries** are an important class of queries:

- A (U)CQ is a FOL query using only conjunction, existential quantification (and disjunction).
- Hence, UCQs contain no negation, no universal quantification, and no function symbols besides constants.
- Correspond to SQL/relational algebra **(union) select-project-join (SPJ) queries** – the most frequently asked queries.
- (U)CQs exhibit nice computational and semantic properties, and have been studied extensively in database theory.
- They are important in practice, since relational database engines are specifically optimized for CQs.

Definition of conjunctive queries (CQs)

Def.: A **conjunctive query (CQ)** is a FOL query of the form

$$\exists \vec{y}. conj(\vec{x}, \vec{y})$$

where $conj(\vec{x}, \vec{y})$ is a conjunction of atoms and equalities, over the free variables $\vec{x}$, the existentially quantified variables $\vec{y}$, and possibly constants.

Note:

- CQs contain no disjunction, no negation, no universal quantification, and no function symbols besides constants.
- Hence, they correspond to relational algebra **select-project-join (SPJ) queries**.
- CQs are the most frequently asked queries.

Conjunctive queries and SQL – Example

Relational alphabet:

Person(name, age), Lives(person, city), Manages(boss, employee)

Query: return name and age of all persons that live in the same city as their boss.

Expressed in SQL:

```
SELECT P.name, P.age
FROM Person P, Manages M, Lives L1, Lives L2
WHERE P.name = L1.person  AND  P.name = M.employee  AND
      M.boss = L2.person  AND  L1.city = L2.city
```

Expressed as a CQ: (the distinguished variables are the blue ones)

$$\exists b, e, p_1, c_1, p_2, c_2.\mathsf{Person}(n, a) \land \mathsf{Manages}(b, e) \land \mathsf{Lives}(p1, c1) \land \mathsf{Lives}(p2, c2) \land$$
$$n = p1 \ \land \ n = e \ \land \ b = p2 \ \land \ c1 = c2$$

Or simpler: $\exists b, c.\mathsf{Person}(n, a) \land \mathsf{Manages}(b, n) \land \mathsf{Lives}(n, c) \land \mathsf{Lives}(b, c)$

Datalog notation for CQs

A CQ $q = \exists\vec{y}.conj(\vec{x},\vec{y})$ can also be written using **datalog notation** as

$$q(\vec{x}_1) \leftarrow conj'(\vec{x}_1,\vec{y}_1)$$

where $conj'(\vec{x}_1,\vec{y}_1)$ is the list of atoms in $conj(\vec{x},\vec{y})$ obtained by equating the variables $\vec{x}$, $\vec{y}$ according to the equalities in $conj(\vec{x},\vec{y})$.

As a result of such an equality elimination, we have that $\vec{x}_1$ and $\vec{y}_1$ can contain constants and multiple occurrences of the same variable.

Def.: In the above query q, we call:

- $q(\vec{x}_1)$ the **head**;
- $conj'(\vec{x}_1,\vec{y}_1)$ the **body**;
- the variables in $\vec{x}_1$ the **distinguished variables**;
- the variables in $\vec{y}_1$ the **non-distinguished variables**.

Conjunctive queries – Example

- Consider the alphabet $\Sigma = \{E/2\}$ and an **interpretation** $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$. Note that $E^{\mathcal{I}}$ is a binary relation, i.e., $\mathcal{I}$ is a directed graph.
- The following **CQ** q returns all nodes that participate to a triangle in the graph:

$$\exists y, z. E(x, y) \land E(y, z) \land E(z, x)$$

- The query q in **datalog notation** becomes:

$$q(x) \leftarrow E(x, y), E(y, z), E(z, x)$$

- The query q in **SQL** is (we use `Edge(f,s)` for $E(x, y)$):

```
SELECT E1.f
FROM Edge E1, Edge E2, Edge E3
WHERE E1.s = E2.f AND E2.s = E3.f AND E3.s = E1.f
```

Nondeterministic evaluation of CQs

Since a CQ contains only existential quantifications, we can evaluate it by:

1. **guessing a variable assignment** for the non-distinguished variables;
2. **evaluating** the resulting formula (that has no quantifications).

We define a boolean function for CQ evaluation:

```
boolean ConjTruth(𝓘,α,∃y⃗.conj(x⃗,y⃗)) {
  GUESS assignment α[y⃗ ↦ a⃗] {
     return Truth(𝓘,α[y⃗ ↦ a⃗],conj(x⃗,y⃗));
}
```

where Truth$(\mathcal{I}, \alpha, \varphi)$ is defined as for FOL queries, considering only the required cases.

Nondeterministic CQ evaluation algorithm

Specifically, for CQs, Truth$(\mathcal{I}, \alpha, \varphi)$ is defined as follows:

```
boolean Truth(𝓘,α,φ) {
  if (φ is t_1 = t_2)
    return TermEval(𝓘,α,t_1) = TermEval(𝓘,α,t_2);
  if (φ is P(t_1,...,t_k))
    return P^𝓘(TermEval(𝓘,α,t_1),...,TermEval(𝓘,α,t_k));
  if (φ is ψ ∧ ψ')
    return Truth(𝓘,α,ψ) ∧ Truth(𝓘,α,ψ');
}

Δ^𝓘 TermEval(𝓘,α,t) {
   if (t is a variable x) return α(x);
   if (t is a constant c) return c^𝓘;
}
```

CQ evaluation – Combined, data, and query complexity

Theorem (Combined complexity of CQ evaluation)

$\{\langle\mathcal{I},\alpha,q\rangle \mid \mathcal{I},\alpha \models q\}$ is **NP-complete** — see below for hardness.

- time: exponential
- space: polynomial

Theorem (Data complexity of CQ evaluation)

$\{\langle\mathcal{I},\alpha\rangle \mid \mathcal{I},\alpha \models q\}$ is **in LogSpace**

- time: polynomial
- space: logarithmic

Theorem (Query complexity of CQ evaluation)

$\{\langle\alpha,q\rangle \mid \mathcal{I},\alpha \models q\}$ is **NP-complete** — see below for hardness.

- time: exponential
- space: polynomial

3-colorability

An undirected graph is k-**colorable** if it is possible to assign to each node one of k colors in such a way that every two nodes connected by an edge have different colors.

Def.: 3-colorability is the following decision problem

Given an undirected graph $G = (V, E)$, is it 3-colorable?

Theorem

3-colorability is NP-complete.

We exploit 3-colorability to show NP-hardness of conjunctive query evaluation.

Reduction from 3-colorability to CQ evaluation

Let $G = (V, E)$ be an undirected graph (without edges connecting a node to itself). We consider a relational alphabet consisting of a single binary relation Edge and define:

- An **Interpretation**: $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where:
  - $\Delta^{\mathcal{I}} = \{\mathsf{r}, \mathsf{g}, \mathsf{b}\}$
  - $\mathsf{Edge}^{\mathcal{I}} = \{(\mathsf{r}, \mathsf{g}), (\mathsf{g}, \mathsf{r}), (\mathsf{r}, \mathsf{b}), (\mathsf{b}, \mathsf{r}), (\mathsf{g}, \mathsf{b}), (\mathsf{b}, \mathsf{g})\}$
- A **conjunctive query:** Let $V = \{v_1, \ldots, v_n\}$, then consider the boolean conjunctive query defined as:

$$q_G \;=\; \exists x_1, \ldots, x_n . \bigwedge_{\{v_i, v_j\} \in E} \mathsf{Edge}(x_i, x_j) \wedge \mathsf{Edge}(x_j, x_i)$$

Theorem

G is 3-colorable iff $\mathcal{I} \models q_G$.

NP-hardness of CQ evaluation

The previous reduction immediately gives us the hardness for combined complexity.

Theorem

CQ evaluation is NP-hard in combined complexity.

Note: in the previous reduction, the interpretation does not depend on the actual graph. Hence, the reduction provides also the lower-bound for query complexity.

Theorem

CQ evaluation is NP-hard in query (and combined) complexity.

Homomorphism

Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ and $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ be two interpretations over the same alphabet (for simplicity, we consider only constants as functions).

Def.: A homomorphism from $\mathcal{I}$ to $\mathcal{J}$

is a mapping $h : \Delta^{\mathcal{I}} \to \Delta^{\mathcal{J}}$ that preserves constants and relations, i.e., such that:

- $h(c^{\mathcal{I}}) \ = \ c^{\mathcal{J}}$
- if $(a_1, \ldots, a_k) \in P^{\mathcal{I}}$ then $(h(a_1), \ldots, h(a_k)) \in P^{\mathcal{J}}$

Note: An isomorphism is a homomorphism that is one-to-one and onto.

Theorem

FOL is unable to distinguish between interpretations that are isomorphic.

Proof. See any standard book on logic. □

Recognition problem and boolean query evaluation

Consider the recognition problem associated to the evaluation of a query q of arity k. Then

$$\mathcal{I}, \alpha \models q(x_1, \ldots, x_k) \qquad \text{iff} \qquad \mathcal{I}_{\alpha,\vec{c}} \models q(c_1, \ldots, c_k)$$

where $\mathcal{I}_{\alpha,\vec{c}}$ is identical to $\mathcal{I}$ but includes new constants $c_1, \ldots, c_k$ that are interpreted as $c_i^{\mathcal{I}_{\alpha,\vec{c}}} = \alpha(x_i)$.

That is, we can **reduce the recognition problem to the evaluation of a boolean query**.

Canonical interpretation of a (boolean) CQ

Let q be a boolean conjunctive query $\exists x_1, \ldots, x_n.conj$

Def.: The canonical interpretation $\mathcal{I}_q$ associated with q

is the interpretation $\mathcal{I}_q = (\Delta^{\mathcal{I}_q}, \cdot^{\mathcal{I}_q})$, where

- $\Delta^{\mathcal{I}_q} = \{x_1, \ldots, x_n\} \cup \{c \mid c$ constant occurring in $q\}$, i.e., all the variables and constants in q;
- $c^{\mathcal{I}_q} = c$, for each constant c in q;
- $(t_1, \ldots, t_k) \in P^{\mathcal{I}_q}$ iff the atom $P(t_1, \ldots, t_k)$ occurs in q.

Sometimes the procedure for obtaining the canonical interpretation is called **freezing** of q.

Canonical interpretation of a (boolean) CQ – Example

Consider the boolean query q

$$q(c) \leftarrow E(c, y), E(y, z), E(z, c)$$

Then, the canonical interpretation $\mathcal{I}_q$ is defined as

$$\mathcal{I}_q \;=\; (\Delta^{\mathcal{I}_q}, \cdot^{\mathcal{I}_q})$$

where

- $\Delta^{\mathcal{I}_q} = \{y, z, c\}$
- $E^{\mathcal{I}_q} = \{(c, y), (y, z), (z, c)\}$
- $c^{\mathcal{I}_q} = c$

Canonical interpretation and (boolean) CQ evaluation

Theorem ([Chandra and Merlin, 1977])

For boolean CQs, $\mathcal{I} \models q$ iff there exists a homomorphism from $\mathcal{I}_q$ to $\mathcal{I}$.

Proof.
"$\Rightarrow$" Let $\mathcal{I} \models q$, let α be an assignment to the existential variables that makes q true in $\mathcal{I}$, and let $\hat{\alpha}$ be its extension to constants. Then $\hat{\alpha}$ is a homomorphism from $\mathcal{I}_q$ to $\mathcal{I}$.

"$\Leftarrow$" Let h be a homomorphism from $\mathcal{I}_q$ to $\mathcal{I}$. Then restricting h to the variables only we obtain an assignment to the existential variables that makes q true in $\mathcal{I}$. $\square$

Canonical interpretation and CQ evaluation – Example

Consider the boolean query $q() \leftarrow R_1(x,y), R_2(y,z), R_1(x,z)$.

The canonical interpretation of q is $\mathcal{I}_q = (\Delta^{\mathcal{I}_q}, \cdot^{\mathcal{I}_q})$, where

$$\Delta^{\mathcal{I}_q} = \{x,y,z\}, \qquad R_1^{\mathcal{I}_q} = \{(x,y),(x,z)\} \qquad R_2^{\mathcal{I}_q} = \{(y,z)\}$$

Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, with

$$\Delta^{\mathcal{I}} = \{a,b\}, \qquad R_1^{\mathcal{I}} = \{(a,b)\} \qquad R_2^{\mathcal{I}} = \{(b,b)\}$$

Then h defined as follows is a homomorphism from $\mathcal{I}_q$ to $\mathcal{I}$:

$$h(x) = a, \qquad h(y) = b, \qquad h(z) = b$$

This shows that $\mathcal{I} \models q$.

Canonical interpretation and (boolean) CQ evaluation

The previous result can be rephrased as follows:

(The recognition problem associated to) **query evaluation can be reduced to finding a homomorphism**.

Finding a homomorphism between two interpretations (i.e., relational structures) is also known as solving a **Constraint Satisfaction Problem** (CSP), a problem well-studied in AI – see also [Kolaitis and Vardi, 1998].

Query containment

Def.: Query containment

Given two FOL queries φ and ψ of the same arity, φ **is contained in** ψ, denoted $\varphi \subseteq \psi$, if for all interpretations $\mathcal{I}$ and all assignments α we have that

$$\mathcal{I}, \alpha \models \varphi \quad \text{implies} \quad \mathcal{I}, \alpha \models \psi$$

(In logical terms: $\varphi \models \psi$.)

Note: Query containment is of special interest in query optimization.

Theorem

For FOL queries, query containment is undecidable.

Proof.: Reduction from FOL logical implication. $\square$

Query containment for CQs

For CQs, query containment $q_1(\vec{x}) \subseteq q_2(\vec{x})$ can be reduced to query evaluation.

1. **Freeze the free variables**, i.e., consider them as constants.
   This is possible, since $q_1(\vec{x}) \subseteq q_2(\vec{x})$ iff
   - $\mathcal{I}, \alpha \models q_1(\vec{x})$ implies $\mathcal{I}, \alpha \models q_2(\vec{x})$, for all $\mathcal{I}$ and α; or equivalently
   - $\mathcal{I}_{\alpha,\vec{c}} \models q_1(\vec{c})$ implies $\mathcal{I}_{\alpha,\vec{c}} \models q_2(\vec{c})$, for all $\mathcal{I}_{\alpha,\vec{c}}$, where $\vec{c}$ are new constants, and $\mathcal{I}_{\alpha,\vec{c}}$ extends $\mathcal{I}$ to the new constants with $c^{\mathcal{I}_{\alpha,\vec{c}}} = \alpha(x)$.

2. **Construct the canonical interpretation $\mathcal{I}_{q_1(\vec{c})}$ of the CQ $q_1(\vec{c})$ on the left hand side** ...

3. ... and **evaluate on $\mathcal{I}_{q_1(\vec{c})}$ the CQ $q_2(\vec{c})$ on the right hand side**, i.e., check whether $\mathcal{I}_{q_1(\vec{c})} \models q_2(\vec{c})$.

Reducing containment of CQs to CQ evaluation

Theorem ([Chandra and Merlin, 1977])

For CQs, $q_1(\vec{x}) \subseteq q_2(\vec{x})$ iff $\mathcal{I}_{q_1(\vec{c})} \models q_2(\vec{c})$, where $\vec{c}$ are new constants.

Proof.

"⇒" Assume that $q_1(\vec{x}) \subseteq q_2(\vec{x})$.

- Since $\mathcal{I}_{q_1(\vec{c})} \models q_1(\vec{c})$, it follows that $\mathcal{I}_{q_1(\vec{c})} \models q_2(\vec{c})$.

"⇐" Assume that $\mathcal{I}_{q_1(\vec{c})} \models q_2(\vec{c})$.

- By [Chandra and Merlin, 1977] on hom., for every $\mathcal{I}$ such that $\mathcal{I} \models q_1(\vec{c})$ there exists a homomorphism h from $\mathcal{I}_{q_1(\vec{c})}$ to $\mathcal{I}$.
- On the other hand, since $\mathcal{I}_{q_1(\vec{c})} \models q_2(\vec{c})$, again by [Chandra and Merlin, 1977] on hom., there exists a homomorphism h' from $\mathcal{I}_{q_2(\vec{c})}$ to $\mathcal{I}_{q_1(\vec{c})}$.
- The mapping $h \circ h'$ (obtained by composing h and h') is a homomorphism from $\mathcal{I}_{q_2(\vec{c})}$ to $\mathcal{I}$. Hence, once again by [Chandra and Merlin, 1977] on hom., $\mathcal{I} \models q_2(\vec{c})$.

So we can conclude that $q_1(\vec{c}) \subseteq q_2(\vec{c})$, and hence $q_1(\vec{x}) \subseteq q_2(\vec{x})$. $\square$

Query containment for CQs

For CQs, we also have that (boolean) query evaluation $\mathcal{I} \models q$ can be reduced to query containment.

Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$.
We construct the (boolean) CQ $q_{\mathcal{I}}$ as follows:

- $q_{\mathcal{I}}$ has no existential variables (hence no variables at all);
- the constants in $q_{\mathcal{I}}$ are the elements of $\Delta^{\mathcal{I}}$;
- for each relation P interpreted in $\mathcal{I}$ and for each fact $(a_1, \ldots, a_k) \in P^{\mathcal{I}}$, $q_{\mathcal{I}}$ contains one atom $P(a_1, \ldots, a_k)$ (note that each $a_i \in \Delta^{\mathcal{I}}$ is a constant in $q_{\mathcal{I}}$).

Theorem

For CQs, $\mathcal{I} \models q$ iff $q_{\mathcal{I}} \subseteq q$.

Query containment for CQs – Complexity

From the previous results and NP-completenss of combined complexity of CQ evaluation, we immediately get:

Theorem

Containment of CQs is NP-complete.

Since CQ evaluation is NP-complete even in query complexity, the above result can be strengthened:

Theorem

Containment $q_1(\vec{x}) \subseteq q_2(\vec{x})$ of CQs is NP-complete, even when q_1 is considered fixed.

Outline of Part 4.2

Union of conjunctive queries (UCQs)

Def.: A **union of conjunctive queries (UCQ)** is a FOL query of the form

$$\bigvee_{i=1,\ldots,n} \exists \vec{y}_i . conj_i(\vec{x}, \vec{y}_i)$$

where each $\exists \vec{y}_i . conj_i(\vec{x}, \vec{y}_i)$ is a conjunctive query (note that all CQs in a UCQ have the same set of distinguished variables).

Note: Obviously, each conjunctive query is also a union of conjunctive queries.

Datalog notation for UCQs

A union of conjunctive queries

$$q \;=\; \bigvee_{i=1,\ldots,n} \exists \vec{y}_i.\, conj_i(\vec{x}, \vec{y}_i)$$

is written in **datalog notation** as

$$\begin{array}{llll} \{ & q(\vec{x}) & \leftarrow & conj'_1(\vec{x}, \vec{y_1}') \\ & & \vdots & \\ & q(\vec{x}) & \leftarrow & conj'_n(\vec{x}, \vec{y_n}') \quad \} \end{array}$$

where each element of the set is the datalog expression corresponding to the conjunctive query $q_i = \exists \vec{y}_i.\, conj_i(\vec{x}, \vec{y}_i)$.

Note: normally, we omit the set brackets.

Evaluation of UCQs

From the definition of FOL query we have that:

$$\mathcal{I}, \alpha \models \bigvee_{i=1,\ldots,n} \exists \vec{y}_i.\, conj_i(\vec{x}, \vec{y}_i)$$

if and only if

$$\mathcal{I}, \alpha \models \exists \vec{y}_i.\, conj_i(\vec{x}, \vec{y}_i), \qquad \text{for some } i \in \{1, \ldots, n\}.$$

Hence to evaluate a UCQ q, we simply evaluate a number (linear in the size of q) of conjunctive queries in isolation.

Hence, **evaluating UCQs has the same complexity as evaluating CQs**.

UCQ evaluation – Combined, data, and query complexity

Theorem (Combined complexity of UCQ evaluation)

$\{\langle \mathcal{I}, \alpha, q\rangle \mid \mathcal{I}, \alpha \models q\}$ is NP-**complete**.

- time: exponential
- space: polynomial

Theorem (Data complexity of UCQ evaluation)

$\{\langle \mathcal{I}, q\rangle \mid \mathcal{I}, \alpha \models q\}$ is **in** $\textsc{LogSpace}$ (query q fixed).

- time: polynomial
- space: logarithmic

Theorem (Query complexity of UCQ evaluation)

$\{\langle \alpha, q\rangle \mid \mathcal{I}, \alpha \models q\}$ is NP-**complete** (interpretation $\mathcal{I}$ fixed).

- time: exponential
- space: polynomial

Query containment for UCQs

Theorem

For UCQs, the following holds:
$\{q_1, \ldots, q_k\} \subseteq \{q'_1, \ldots, q'_n\}$ iff for each q_i there is a q'_j such that $q_i \subseteq q'_j$.

Proof.
"$\Leftarrow$" Obvious.

"$\Rightarrow$" If the containment holds, then we have
$\{q_1(\vec{c}), \ldots, q_k(\vec{c})\} \subseteq \{q'_1(\vec{c}), \ldots, q'_n(\vec{c})\}$, where $\vec{c}$ are new constants:

- Now consider $\mathcal{I}_{q_i(\vec{c})}$. We have $\mathcal{I}_{q_i(\vec{c})} \models q_i(\vec{c})$, and hence $\mathcal{I}_{q_i(\vec{c})} \models \{q_1(\vec{c}), \ldots, q_k(\vec{c})\}$.
- By the containment, we have that $\mathcal{I}_{q_i(\vec{c})} \models \{q'_1(\vec{c}), \ldots, q'_n(\vec{c})\}$. I.e., there exists a $q'_j(\vec{c})$ such that $\mathcal{I}_{q_i(\vec{c})} \models q'_j(\vec{c})$.
- Hence, by [Chandra and Merlin, 1977] on containment of CQs, we have that $q_i \subseteq q'_j$. $\square$

Query containment for UCQs – Complexity

From the previous result, we have that we can check $\{q_1, \ldots, q_k\} \subseteq \{q'_1, \ldots, q'_n\}$ by at most $k \cdot n$ CQ containment checks.

We immediately get:

Theorem

Containment of UCQs is NP-complete.

Outline of Part 4.2

3. Query answering in databases

4. Querying databases and ontologies
   - Query answering in traditional databases
   - Query answering in ontologies
   - Query answering in ontology-based data access

5. Query answering in Description Logics

Query answering

In ontology-based data access we are interested in a reasoning service that is not typical in ontologies (or in a FOL theory, or in UML class diagrams, or in a knowledge base) but it is very common in databases: **query answering**.

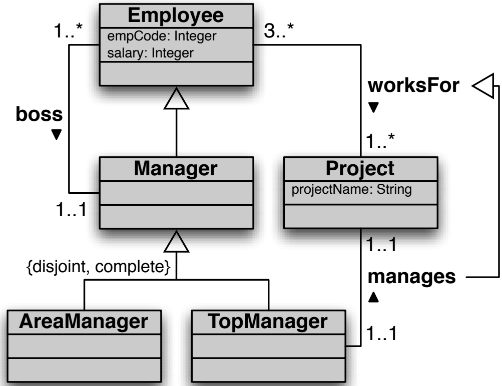
Def.: Query

Is an expression at the intensional level denoting a set of tuples of individuals satisfying a given condition.

Def.: Query Answering

Is the reasoning service that actually computes the answer to a query.

Example of query over an ontology



$$
\begin{aligned}
q(ce, cm, sa) \ \leftarrow\ & \exists e, p, m. \\
& \text{worksFor}(e, p) \land \text{manages}(m, p) \land \text{boss}(m, e) \land \text{empCode}(e, ce) \land \\
& \text{empCode}(m, cm) \land \text{salary}(e, sa) \land \text{salary}(m, sa)
\end{aligned}
$$

Query answering under different assumptions

There are two fundamentally different assumptions when addressing query answering:

- **Complete information** on the data, as in traditional databases.
- **Incomplete information** on the data, as in ontologies (aka knowledge bases), but also information integration in databases.

Outline of Part 4.2

3 Query answering in databases

4 Querying databases and ontologies
- Query answering in traditional databases
- Query answering in ontologies
- Query answering in ontology-based data access

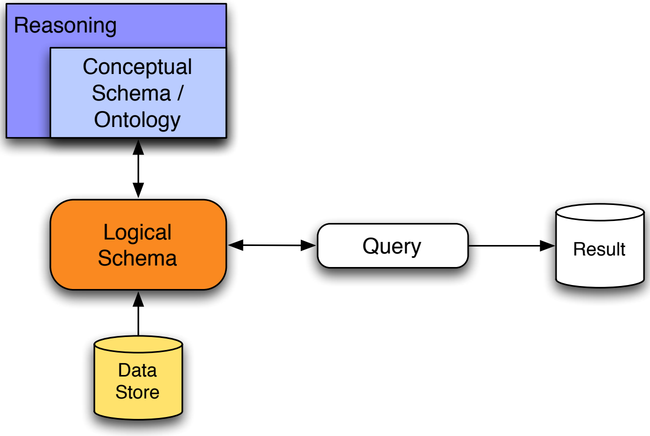
5 Query answering in Description Logics

Query answering in traditional databases

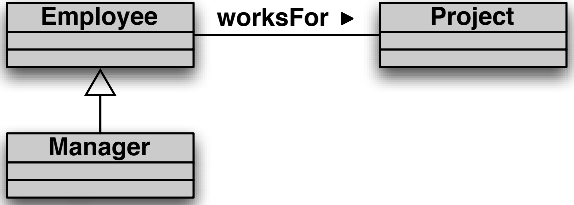
- Data are completely specified (CWA), and typically large.
- Schema/intensional information used in the design phase.
- At **runtime**, the data is assumed to satisfy the schema, and therefore the **schema is not used**.
- Queries allow for complex navigation paths in the data (cf. SQL).

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

Query answering in traditional databases (cont'd)



Query answering in traditional databases – Example



For each concept/relationship we have a (complete) table in the DB.

DB: Employee = { john, mary, nick }
Manager = { john, nick }
Project = { prA, prB }
worksFor = { (john,prA), (mary,prB) }

Query: $q(x) \leftarrow \exists p.\, \text{Manager}(x) \land \text{Project}(p) \land \text{worksFor}(x, p)$

Answer: { john }

Outline of Part 4.2

Query answering in ontologies

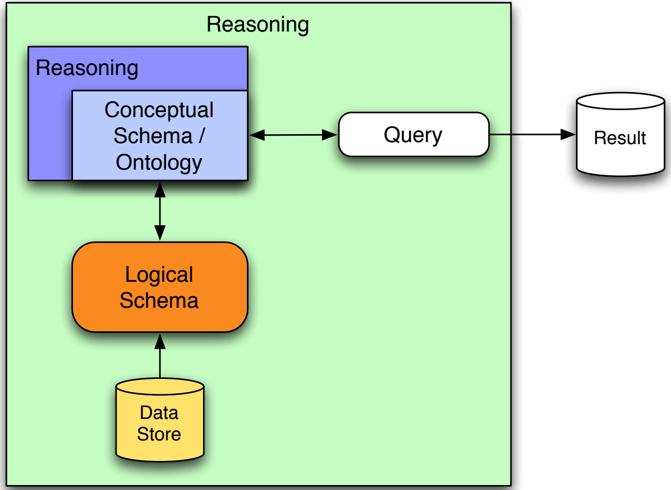
- An ontology (or conceptual schema, or knowledge base) imposes constraints on the data.
- Actual data may be incomplete or inconsistent w.r.t. such constraints.
- The system has to take into account the constraints during query answering, and overcome incompleteness or inconsistency.

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

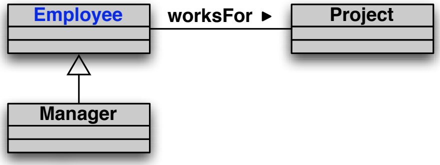
Note:

- The 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 in ontologies (cont'd)



Query answering in ontologies – Example



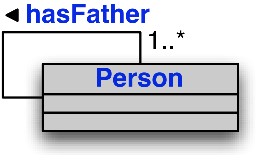
The tables in the database may be **incompletely specified**, or even missing for some classes/properties.

DB: $\text{Manager} \supseteq \{\ \texttt{john}, \texttt{nick}\ \}$
$\text{Project} \supseteq \{\ \texttt{prA}, \texttt{prB}\ \}$
$\text{worksFor} \supseteq \{\ (\texttt{john},\texttt{prA}), (\texttt{mary},\texttt{prB})\ \}$

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

Answer: $\{\ \texttt{john}, \texttt{nick}, \texttt{mary}\ \}$

Query answering in ontologies – Example 2



Each person has a father, who is a person.

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

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

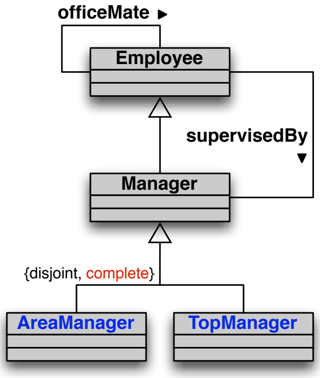
$$q_2(x) \leftarrow \exists y.\,\text{hasFather}(x, y)$$

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

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

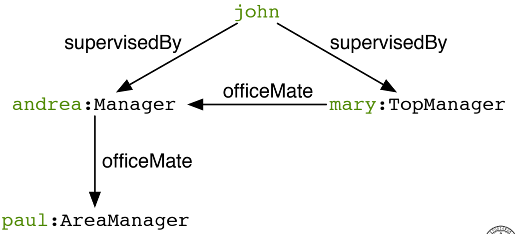
Answers: to q_1: { (john,nick), (nick,toni) }
to q_2: { john, nick, toni }
to q_3: { john, nick, toni }
to q_4: { }

QA in ontologies – Andrea's Example$^{(*)}$



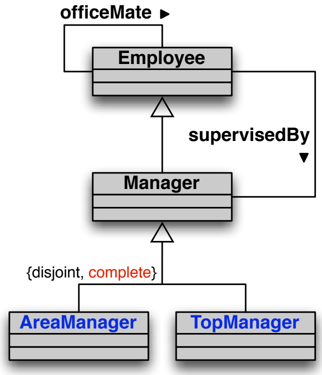
Manager is **partitioned into** AreaManager and TopManager.

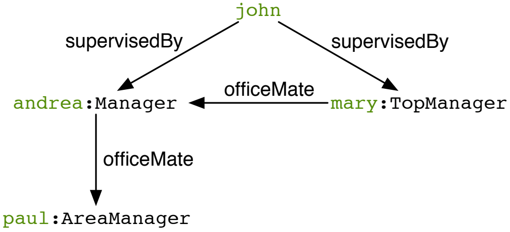
$$
\begin{aligned}
\text{Employee} &\supseteq \{\ \texttt{andrea}, \texttt{paul}, \texttt{mary}, \texttt{john}\ \}\\
\text{Manager} &\supseteq \{\ \texttt{andrea}, \texttt{paul}, \texttt{mary}\ \}\\
\text{AreaManager} &\supseteq \{\ \texttt{paul}\ \}\\
\text{TopManager} &\supseteq \{\ \texttt{mary}\ \}\\
\text{supervisedBy} &\supseteq \{\ (\texttt{john},\texttt{andrea}), (\texttt{john},\texttt{mary})\ \}\\
\text{officeMate} &\supseteq \{\ (\texttt{mary},\texttt{andrea}), (\texttt{andrea},\texttt{paul})\ \}
\end{aligned}
$$



(∗) Due to Andrea Schaerf [Schaerf, 1993].

QA in ontologies – Andrea's Example (cont'd)





$$q(x) \leftarrow \exists y, z.\, \mathsf{supervisedBy}(x, y) \land \mathsf{TopManager}(y) \land \mathsf{officeMate}(y, z) \land \mathsf{AreaManager}(z)$$

Answer: { john }

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

Outline of Part 4.2

3. Query answering in databases
4. Querying databases and ontologies
   - Query answering in traditional databases
   - Query answering in ontologies
   - Query answering in ontology-based data access
5. Query answering in Description Logics

Query answering in ontology-based data access

In OBDA, we have to face the difficulties of both settings:

- 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.

Questions that need to be addressed

In the context of ontology-based data access:

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** take into account these issues?

Which language to use for querying ontologies?

Two borderline 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).

A good tradeoff is to use (unions of) **conjunctive queries**.

Outline of Part 4.2

3 Query answering in databases

4 Querying databases and ontologies

5 Query answering in Description Logics
- Queries over Description Logics ontologies
- Certain answers
- Complexity of query answering

Outline of Part 4.2

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.

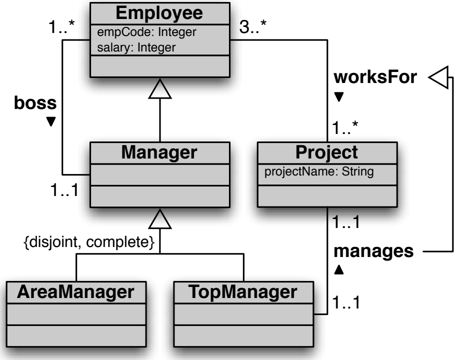
Def.: A **conjunctive query** $q(\vec{x})$ over an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$

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:

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

Outline of Part 4.2

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 $\mathcal{O}$

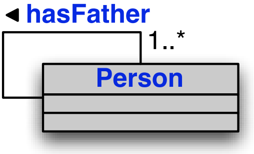
Is the problem of **computing the certain answers** to a query over $\mathcal{O}$.

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 $\mathcal{T}$:

$$\begin{array}{rcl} \exists \mathsf{hasFather} & \sqsubseteq & \mathsf{Person} \\ \exists \mathsf{hasFather}^- & \sqsubseteq & \mathsf{Person} \\ \mathsf{Person} & \sqsubseteq & \exists \mathsf{hasFather} \end{array}$$

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

Queries:

$$\begin{array}{lll} q_1(x,y) & \leftarrow & \mathsf{hasFather}(x,y) \\ q_2(x) & \leftarrow & \exists y.\, \mathsf{hasFather}(x,y) \\ q_3(x) & \leftarrow & \exists y_1,y_2,y_3.\, \mathsf{hasFather}(x,y_1) \land \mathsf{hasFather}(y_1,y_2) \land \mathsf{hasFather}(y_2,y_3) \\ q_4(x,y_3) & \leftarrow & \exists y_1,y_2.\, \mathsf{hasFather}(x,y_1) \land \mathsf{hasFather}(y_1,y_2) \land \mathsf{hasFather}(y_2,y_3) \end{array}$$

Certain answers:

$$\begin{array}{l} cert(q_1, \langle \mathcal{T}, \mathcal{A} \rangle) = \{\ (\mathtt{john},\mathtt{nick}),\ (\mathtt{nick},\mathtt{toni})\ \} \\ cert(q_2, \langle \mathcal{T}, \mathcal{A} \rangle) = \{\ \mathtt{john},\ \mathtt{nick},\ \mathtt{toni}\ \} \\ cert(q_3, \langle \mathcal{T}, \mathcal{A} \rangle) = \{\ \mathtt{john},\ \mathtt{nick},\ \mathtt{toni}\ \} \\ cert(q_4, \langle \mathcal{T}, \mathcal{A} \rangle) = \{\ \} \end{array}$$

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:

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

Outline of Part 4.2

Complexity measures for queries over ontologies

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

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

	Combined complexity	Data complexity
Plain databases	NP-complete	in AC^0 (1)
$\mathcal{ALCI}$, $\mathcal{SH}$, $\mathcal{SHIQ}$, ...	2ExpTime-complete (3)	coNP-complete (2)
OWL 2 (and less)	3ExpTime-hard	coNP-hard

(1) This is what we need to scale with the data.

(2) coNP-hard already for a TBox with a single disjunction [Donini *et al.*, 1994; Calvanese *et al.*, 2006b].
In coNP for very expressive DLs [Levy and Rousset, 1998; Ortiz *et al.*, 2006; Glimm *et al.*, 2007].

(3) [Calvanese *et al.*, 1998a; Lutz, 2007]

Questions

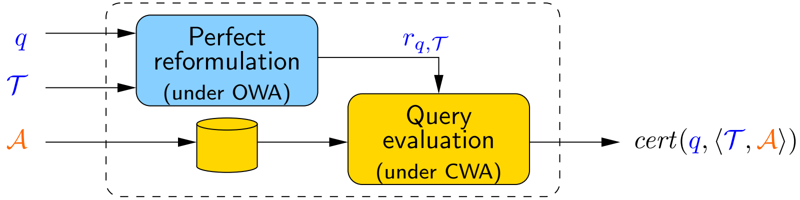
- Can we find interesting (description) logics for which query answering can be done efficiently (i.e., in AC^0)?
- If yes, can we leverage relational database technology for query answering?

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 $\mathcal{T}$.

$\leadsto$ 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 $r_{q,\mathcal{T}}$.

$\mathcal{Q}$-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.

Part 4.3

Linking ontologies to relational data

Outline of Part 4.3

Outline of Part 4.3

Managing ABoxes

In the traditional DL setting, it is assumed that the data is maintained in the ABox of the ontology:

- The ABox is perfectly compatible with the TBox:
  - the vocabulary of concepts, roles, and attributes is the one used in the TBox.
  - The ABox "stores" abstract objects, and these objects and their properties are those returned by queries over the ontology.
- There may be different ways to manage the ABox from a physical point of view:
  - Description Logics reasoners maintain the ABox is main-memory data structures.
  - When an ABox becomes large, managing it in secondary storage may be required, but this is again handled directly by the reasoner.

Data in external sources

There are several situations where the assumptions of having the data in an ABox managed directly by the ontology system (e.g., a Description Logics reasoner) is not feasible or realistic:

- When the ABox is very large, so that it requires relational database technology.
- When we have no direct control over the data since it belongs to some external organization, which controls the access to it.
- When multiple data sources need to be accessed, such as in Information Integration.

We would like to deal with such a situation by keeping the data in the external (relational) storage, and performing **query answering** by leveraging the capabilities of the **relational engine**.

The impedance mismatch problem

We have to deal with the **impedance mismatch problem**:

- Sources store data, which is constituted by values taken from concrete domains, such as strings, integers, codes, . . .
- Instead, instances of concepts and relations in an ontology are (abstract) objects.

Solution:

- We need to specify how to construct from the data values in the relational sources the (abstract) objects that populate the ABox of the ontology.
- This specification is embedded in the mappings between the data sources and the ontology.

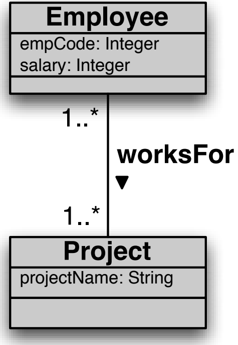
Note: the **ABox** is only **virtual**, and the objects are not materialized.

Solution to the impedance mismatch problem

We need to define a **mapping language** that allows for specifying how to transform data into abstract objects:

- Each mapping assertion maps:
  - a query that retrieves values from a data source to ...
  - a set of atoms specified over the ontology.
- Basic idea: use **Skolem functions** in the atoms over the ontology to "generate" the objects from the data values.
- Semantics of mappings:
  - Objects are denoted by terms (of exactly one level of nesting).
  - Different terms denote different objects (i.e., we make the unique name assumption on terms).

Impedance mismatch – Example



Actual data is stored in a DB:
– An employee is identified by her SSN.
– A project is identified by its name.

$D_1[SSN\colon \text{String}, PrName\colon \text{String}]$
Employees and projects they work for

$D_2[Code\colon \text{String}, Salary\colon \text{Int}]$
Employee's code with salary

$D_3[Code\colon \text{String}, SSN\colon \text{String}]$
Employee's Code with SSN

...

Intuitively:

- An employee should be created from her SSN: $\mathbf{pers}(SSN)$
- A project should be created from its name: $\mathbf{proj}(PrName)$

Creating object identifiers

We need to associate to the data in the tables objects in the ontology.

- We introduce an alphabet Λ of **function symbols**, each with an associated arity.
- To denote values, we use value constants from an alphabet Γ_V.
- To denote objects, we use **object terms** instead of object constants. An object term has the form $\mathbf{f}(d_1, \ldots, d_n)$, with $\mathbf{f} \in \Lambda$, and each d_i a value constant in Γ_V.

Example

- If a person is identified by her *SSN*, we can introduce a function symbol **pers**/1. If `VRD56B25` is a *SSN*, then **pers**(`VRD56B25`) denotes a person.
- If a person is identified by her *name* and *dateOfBirth*, we can introduce a function symbol **pers**/2. Then **pers**(`Vardi`, `25/2/56`) denotes a person.

Mapping assertions

Mapping assertions are used to extract the data from the DB to populate the ontology.

We make use of **variable terms**, which are like object terms, but with variables instead of values as arguments of the functions.

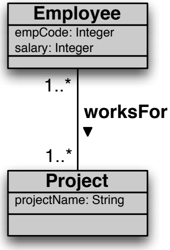
Def.: A **mapping assertion** between a database $\mathcal{D}$ and a TBox $\mathcal{T}$ has the form

$$\Phi(\vec{x}) \leadsto \Psi(\vec{t}, \vec{y})$$

where

- Φ is an arbitrary SQL query of arity $n > 0$ over $\mathcal{D}$;
- Ψ is a conjunctive query over $\mathcal{T}$ of arity $n' > 0$ **without non-distinguished variables**;
- $\vec{x}$, $\vec{y}$ are variables, with $\vec{y} \subseteq \vec{x}$;
- $\vec{t}$ are variable terms of the form $\mathbf{f}(\vec{z})$, with $\mathbf{f} \in \Lambda$ and $\vec{z} \subseteq \vec{x}$.

Mapping assertions – Example



D_1[*SSN*: String, *PrName*: String]
Employees and Projects they work for

D_2[*Code*: String, *Salary*: Int]
Employee's code with salary

D_3[*Code*: String, *SSN*: String]
Employee's code with SSN

...

m_1: `SELECT SSN, PrName FROM D1` $\rightsquigarrow$ Employee(**pers**(*SSN*)), Project(**proj**(*PrName*)), projectName(**proj**(*PrName*), *PrName*), worksFor(**pers**(*SSN*), **proj**(*PrName*))

m_2: `SELECT SSN, Salary FROM D2, D3 WHERE D2.Code = D3.Code` $\rightsquigarrow$ Employee(**pers**(*SSN*)), salary(**pers**(*SSN*), *Salary*)

Outline of Part 4.3

Ontology-Based Data Access System

The mapping assertions are a crucial part of an Ontology-Based Data Access System.

Def.: Ontology-Based Data Access System

is a triple $\mathcal{O} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$, where

- $\mathcal{T}$ is a TBox.
- $\mathcal{D}$ is a relational database.
- $\mathcal{M}$ is a set of mapping assertions between $\mathcal{T}$ and $\mathcal{D}$.

Semantics of mappings

To define the semantics of an OBDA system $\mathcal{O} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$, we first need to define the semantics of mappings.

Def.: Satisfaction of a mapping assertion with respect to a database

An interpretation $\mathcal{I}$ **satisfies** a mapping assertion $\Phi(\vec{x}) \rightsquigarrow \Psi(\vec{t}, \vec{y})$ in $\mathcal{M}$ **with respect to a database** $\mathcal{D}$, if for each tuple of values $\vec{v} \in \mathit{Eval}(\Phi, \mathcal{D})$, and for each ground atom in $\Psi[\vec{x}/\vec{v}]$, we have that:

- if the ground atom is $A(s)$, then $s^\mathcal{I} \in A^\mathcal{I}$.
- if the ground atom is $P(s_1, s_2)$, then $(s_1^\mathcal{I}, s_2^\mathcal{I}) \in P^\mathcal{I}$.

Intuitively, $\mathcal{I}$ **satisfies** $\Phi \rightsquigarrow \Psi$ w.r.t. $\mathcal{D}$ if all facts obtained by evaluating Φ over $\mathcal{D}$ and then propagating the answers to Ψ, hold in $\mathcal{I}$.

Note: $\mathit{Eval}(\Phi, \mathcal{D})$ denotes the result of evaluating Φ over the database $\mathcal{D}$.
$\Psi[\vec{x}/\vec{v}]$ denotes Ψ where each x_i has been substituted with v_i.

Semantics of an OBDA system

Def.: Model of an OBDA system

An interpretation $\mathcal{I}$ is a **model** of $\mathcal{O} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$ if:

- $\mathcal{I}$ is a model of $\mathcal{T}$;
- $\mathcal{I}$ satisfies $\mathcal{M}$ w.r.t. $\mathcal{D}$, i.e., $\mathcal{I}$ satisfies every assertion in $\mathcal{M}$ w.r.t. $\mathcal{D}$.

An OBDA system $\mathcal{O}$ is **satisfiable** if it admits at least one model.

Outline of Part 4.3

Answering queries over an OBDA system

In an OBDA system $\mathcal{O} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$

- Queries are posed over the TBox $\mathcal{T}$.
- The data needed to answer queries is stored in the database $\mathcal{D}$.
- The mapping $\mathcal{M}$ is used to bridge the gap between $\mathcal{T}$ and $\mathcal{D}$.

Two approaches to exploit the mapping:

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

Note: Both approaches require to first **split** the TBox queries in the mapping assertions into their constituent atoms.

Splitting of mappings

A mapping assertion $\Phi \leadsto \Psi$, where the TBox query Ψ is constituted by the atoms X_1,...,X_k, can be split into several mapping assertions:

$$\Phi \leadsto X_1 \qquad \cdots \qquad \Phi \leadsto X_k$$

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

Example

m_1: `SELECT SSN, PrName FROM D`$_1$ $\leadsto$ Employee(**pers**(*SSN*)),
Project(**proj**(*PrName*)),
projectName(**proj**(*PrName*), *PrName*),
worksFor(**pers**(*SSN*), **proj**(*PrName*))

is split into

m_1^1: `SELECT SSN, PrName FROM D`$_1$ $\leadsto$ Employee(**pers**(*SSN*))
m_1^2: `SELECT SSN, PrName FROM D`$_1$ $\leadsto$ Project(**proj**(*PrName*))
m_1^3: `SELECT SSN, PrName FROM D`$_1$ $\leadsto$ projectName(**proj**(*PrName*), *PrName*)
m_1^4: `SELECT SSN, PrName FROM D`$_1$ $\leadsto$ worksFor(**pers**(*SSN*), **proj**(*PrName*))

Bottom-up approach to query answering

Consists in a straightforward application of the mappings:

1. Propagate the data from $\mathcal{D}$ through $\mathcal{M}$, materializing an ABox $\mathcal{A}_{\mathcal{M},\mathcal{D}}$ (the constants in such an ABox are values and object terms).
2. Apply to $\mathcal{A}_{\mathcal{M},\mathcal{D}}$ and to the TBox $\mathcal{T}$, the satisfiability and query answering algorithms developed for *DL-Lite*$_{\mathcal{A}}$.

This approach has several drawbacks (hence is only theoretical):

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

Top-down approach to query answering

Consists of three steps:

1. **Reformulation:** Compute the perfect reformulation $q_{pr} = \mathit{PerfectRef}(q, \mathcal{T}_P)$ of the original query q, using the inclusion assertions of the TBox $\mathcal{T}$ (see later).

2. **Unfolding:** Compute from q_{pr} a new query q_{unf} by unfolding q_{pr} using (the split version of) the mappings $\mathcal{M}$.
   - Essentially, each atom in q_{pr} that unifies with an atom in Ψ is substituted with the corresponding query Φ over the database.
   - The unfolded query is such that $\mathit{Eval}(q_{unf}, \mathcal{D}) = \mathit{Eval}(q_{pr}, \mathcal{A}_{\mathcal{M},\mathcal{D}})$.

3. **Evaluation:** Delegate the evaluation of q_{unf} to the relational DBMS managing $\mathcal{D}$.

Unfolding

To unfold a query q_{pr} with respect to a set of mapping assertions:

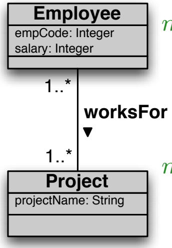
1. For each non-split mapping assertion $\Phi_i(\vec{x}) \rightsquigarrow \Psi_i(\vec{t}, \vec{y})$:
   1. Introduce a **view symbol** Aux$_i$ of arity equal to that of Φ_i.
   2. Add a **view definition** Aux$_i(\vec{x}) \leftarrow \Phi_i(\vec{x})$.

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

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

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

Unfolding – Example



m_1: `SELECT SSN, PrName FROM D`$_1$ $\leadsto$ Employee(**pers**(SSN)), Project(**proj**($PrName$)), projectName(**proj**($PrName$), $PrName$), worksFor(**pers**(SSN), **proj**($PrName$))

m_2: `SELECT SSN, Salary FROM D`$_2$`, D`$_3$ `WHERE D`$_2$`.Code = D`$_3$`.Code` $\leadsto$ Employee(**pers**(SSN)), salary(**pers**(SSN), $Salary$)

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

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

$$
\begin{array}{rcl}
\text{Employee}(\textbf{pers}(SSN)) & \leftarrow & \text{Aux}_1(SSN, PrName) \\
\text{projectName}(\textbf{proj}(PrName), PrName) & \leftarrow & \text{Aux}_1(SSN, PrName) \\
\text{Project}(\textbf{proj}(PrName)) & \leftarrow & \text{Aux}_1(SSN, PrName) \\
\text{worksFor}(\textbf{pers}(SSN), \textbf{proj}(PrName)) & \leftarrow & \text{Aux}_1(SSN, PrName) \\
\text{Employee}(\textbf{pers}(SSN)) & \leftarrow & \text{Aux}_2(SSN, Salary) \\
\text{salary}(\textbf{pers}(SSN), Salary) & \leftarrow & \text{Aux}_2(SSN, Salary)
\end{array}
$$

Unfolding – Example (cont'd)

Query over ontology: employees who work for tones and their salary:

$q(e,s) \leftarrow \mathsf{Employee}(e), \mathsf{salary}(e,s), \mathsf{worksFor}(e,p), \mathsf{projectName}(p, \mathtt{tones})$

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

$$\vartheta(e) = \textbf{pers}(\mathit{SSN}) \qquad \vartheta(s) = \mathit{Salary}$$
$$\vartheta(\mathit{PrName}) = \mathtt{tones} \qquad \vartheta(p) = \textbf{proj}(\mathtt{tones})$$

After applying ϑ to q, we obtain:

$$\begin{aligned} q(\textbf{pers}(\mathit{SSN}), \mathit{Salary}) \leftarrow\ & \mathsf{Employee}(\textbf{pers}(\mathit{SSN})),\ \mathsf{salary}(\textbf{pers}(\mathit{SSN}), \mathit{Salary}),\\ & \mathsf{worksFor}(\textbf{pers}(\mathit{SSN}), \textbf{proj}(\mathtt{tones})),\\ & \mathsf{projectName}(\textbf{proj}(\mathtt{tones}), \mathtt{tones}) \end{aligned}$$

Substituting the atoms with the bodies of the unified clauses, we obtain:

$$\begin{aligned} q(\textbf{pers}(\mathit{SSN}), \mathit{Salary}) \leftarrow\ & \mathsf{Aux}_1(\mathit{SSN}, \mathtt{tones}),\ \mathsf{Aux}_2(\mathit{SSN}, \mathit{Salary}),\\ & \mathsf{Aux}_1(\mathit{SSN}, \mathtt{tones}),\ \mathsf{Aux}_1(\mathit{SSN}, \mathtt{tones}) \end{aligned}$$

Exponential blowup in the unfolding

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

Consider a query: $q(y) \leftarrow A_1(y), A_2(y), \ldots, A_n(y)$

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

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

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

Hence, we obtain one unfolded query

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

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

Computational complexity of query answering

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

Theorem

Query answering in a *DL-Lite* OBDM system $\mathcal{O} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$ is

1. NP-**complete** in the size of the query.
2. PTime in the size of the **TBox** $\mathcal{T}$ and the **mappings** $\mathcal{M}$.
3. AC^0 in the size of the **database** $\mathcal{D}$.

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

Implementation of top-down approach to query answering

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

We can follow different strategies:

1. Substitute each view predicate in the unfolded queries with the corresponding SQL query over the source:
   - \+ joins are performed on the DB attributes;
   - \+ does not generate doubly nested queries;
   - − the number of unfolded queries may be exponential.

2. Construct for each atom in the original query a new view. This view takes the union of all SQL queries corresponding to the view predicates, and constructs also the Skolem terms:
   - \+ avoids exponential blow-up of the resulting query, since the union (of the queries coming from multiple mappings) is done before the joins;
   - − joins are performed on Skolem terms;
   - − generates doubly nested queries.

Which method is better, depends on various parameters.
Experiments have shown that (1) behaves better in most cases.

Towards answering arbitrary SQL queries

- We have seen that answering full SQL (i.e., FOL) queries is undecidable.
- However, we can treat the answers to an UCQ, as "knowledge", and perform further computations on that knowledge.
- This corresponds to applying a knowledge operator to UCQs that are embedded into an arbitrary SQL query (EQL queries) [Calvanese *et al.*, 2007b]
  - The UCQs are answered according to the certain answer semantics.
  - The SQL query is evaluated on the facts returned by the UCQs.
- The approach can be implemented by rewriting the UCQs and embedding the rewritten UCQs into SQL.
- The user "sees" arbitrary SQL queries, but these SQL queries are evaluated according to a weakened semantics.

Outline of Part 4.3

The QuOnto system

- QuOnto is a tool for representing and reasoning over ontologies of the *DL-Lite* family.
- The basic functionalities it offers are:
  - Ontology representation
  - Ontology satisfiability check
  - Intensional reasoning services: concept/property subsumption and disjunction, concept/property satisfiability
  - Query Answering of UCQs
- Includes also support for:
  - Identification path constraints
  - Denial constraints
  - Epistemic queries (EQL-Lite on UCQs)
  - Epistemic constraints (EQL-Lite constraints)
- Reasoning services are highly optimized.
- Can be used with internal and external DBMS (include drivers for Oracle, DB2, IBM Information Integrator, SQL Server, MySQL, etc.).
- Implemented in Java – *APIs are available for selected projects upon request*.

Part 4.4

Reasoning in the *DL-Lite* family

Outline of Part 4.4

Outline of Part 4.4

Outline of Part 4.4

Remarks

In the following, we make some simplifying assumptions:

- We ignore the distinction between objects and values, since it is not relevant for reasoning. Hence we do not use value domains and attributes.
- We do not consider identification constraints.

Notation:

- When the distinction between $DL\text{-}Lite_{\mathcal{R}}$, $DL\text{-}Lite_{\mathcal{F}}$, or $DL\text{-}Lite_{\mathcal{A}}$ is not important, we use just *DL-Lite*.
- Q denotes a basic role, i.e., $Q \longrightarrow P \mid P^-$.
- R denotes a general role, i.e., $R \longrightarrow Q \mid \neg Q$.
- C denotes a general concept, i.e., $C \longrightarrow A \mid \neg A \mid \exists Q \mid \neg \exists Q$, where A is an atomic concept.

TBox Reasoning services

- **Concept Satisfiability:** C is satisfiable wrt $\mathcal{T}$, if there is a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}}$ is not empty, i.e., $\mathcal{T} \not\models C \equiv \bot$
- **Subsumption:** C_1 is subsumed by C_2 wrt $\mathcal{T}$, if for every model $\mathcal{I}$ of $\mathcal{T}$ we have $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$, i.e., $\mathcal{T} \models C_1 \sqsubseteq C_2$.
- **Equivalence:** C_1 and C_2 are equivalent wrt $\mathcal{T}$ if for every model $\mathcal{I}$ of $\mathcal{T}$ we have $C_1^{\mathcal{I}} = C_2^{\mathcal{I}}$, i.e., $\mathcal{T} \models C_1 \equiv C_2$.
- **Disjointness:** C_1 and C_2 are disjoint wrt $\mathcal{T}$ if for every model $\mathcal{I}$ of $\mathcal{T}$ we have $C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} = \emptyset$, i.e., $\mathcal{T} \models C_1 \sqcap C_2 \equiv \bot$
- **Functionality implication:** A functionality assertion $(\textbf{funct}\ Q)$ is logically implied by $\mathcal{T}$ if for every model $\mathcal{I}$ of $\mathcal{T}$, we have that $(o, o_1) \in Q^{\mathcal{I}}$ and $(o, o_2) \in Q^{\mathcal{I}}$ implies $o_1 = o_2$, i.e., $\mathcal{T} \models (\textbf{funct}\ Q)$.

Analogous definitions hold for role satisfiability, subsumption, equivalence, and disjointness.

From TBox reasoning to ontology (un)satisfiability

Basic reasoning service:

- **Ontology satisfiability**: Verify whether an ontology $\mathcal{O}$ is satisfiable, i.e., whether $\mathcal{O}$ admits at least one model.

In the following, we show how to reduce TBox reasoning to ontology unsatisfiability:

1. We show how to reduce TBox reasoning services to concept/role subsumption.
2. We provide reductions from concept/role subsumption to ontology unsatisfiability.

Outline of Part 4.4

Concept/role satisfiability, equivalence, and disjointness

Theorem

1. C is unsatisfiable wrt $\mathcal{T}$ iff $\mathcal{T} \models C \sqsubseteq \neg C$.
2. $\mathcal{T} \models C_1 \equiv C_2$ iff $\mathcal{T} \models C_1 \sqsubseteq C_2$ and $\mathcal{T} \models C_2 \sqsubseteq C_1$.
3. C_1 and C_2 are disjoint iff $\mathcal{T} \models C_1 \sqsubseteq \neg C_2$.

Proof (sketch).

1. "$\Leftarrow$" if $\mathcal{T} \models C \sqsubseteq \neg C$, then $C^\mathcal{I} \subseteq \Delta^\mathcal{I} \setminus C^\mathcal{I}$, for every model $\mathcal{I} = \langle \Delta^\mathcal{I}, \cdot^\mathcal{I} \rangle$ of $\mathcal{T}$; but this holds iff $C^\mathcal{I} = \emptyset$.

   "$\Rightarrow$" if C is unsatisfiable, then $C^\mathcal{I} = \emptyset$, for every model $\mathcal{I}$ of $\mathcal{T}$. Therefore $C^\mathcal{I} \subseteq (\neg C)^\mathcal{I}$.
2. Trivial.
3. Trivial. $\square$

Analogous reductions for role satisfiability, equivalence and disjointness.

From implication of functionalities to subsumption

Theorem

$\mathcal{T} \models (\textbf{funct } Q)$ iff either

- $(\textbf{funct } Q) \in \mathcal{T}$ (only for *DL-Lite*$_\mathcal{F}$ or *DL-Lite*$_\mathcal{A}$), or
- $\mathcal{T} \models Q \sqsubseteq \neg Q$.

Proof (sketch).

"$\Leftarrow$" The case in which $(\textbf{funct } Q) \in \mathcal{T}$ is trivial.
Instead, if $\mathcal{T} \models Q \sqsubseteq \neg Q$, then $Q^\mathcal{I} = \emptyset$ and hence $\mathcal{I} \models (\textbf{funct } Q)$, for every model $\mathcal{I}$ of $\mathcal{T}$.

"$\Rightarrow$" When neither $(\textbf{funct } Q) \in \mathcal{T}$ nor $\mathcal{T} \models Q \sqsubseteq \neg Q$, we can construct a model of $\mathcal{T}$ that is not a model of $(\textbf{funct } Q)$. □

The interesting part of this result is the "only-if" direction, telling us that in DL-Lite functionality is implied only in trivial ways.

Outline of Part 4.4

From concept subsumption to ontology unsatisfiability

Theorem

$\mathcal{T} \models C_1 \sqsubseteq C_2$ iff the ontology $\mathcal{O}_{C_1 \sqsubseteq C_2} = \langle \mathcal{T} \cup \{\hat{A} \sqsubseteq C_1, \hat{A} \sqsubseteq \neg C_2\},\ \{\hat{A}(c)\}\rangle$ is unsatisfiable, where $\hat{A}$ is an atomic concept not in $\mathcal{T}$, and c is a constant.

Intuitively, C_1 is subsumed by C_2 iff the smallest ontology containing $\mathcal{T}$ and implying both $C_1(c)$ and $\neg C_2(c)$ is unsatisfiable.

Proof (sketch).

"$\Leftarrow$" Let $\mathcal{O}_{C_1 \sqsubseteq C_2}$ be unsatisfiable, and suppose that $\mathcal{T} \not\models C_1 \sqsubseteq C_2$. Then there exists a model $\mathcal{I}$ of $\mathcal{T}$ such that $C_1^{\mathcal{I}} \not\subseteq C_2^{\mathcal{I}}$. Hence $C_1^{\mathcal{I}} \setminus C_2^{\mathcal{I}} \neq \emptyset$. We can extend $\mathcal{I}$ to a model of $\mathcal{O}_{C_1 \sqsubseteq C_2}$ by taking $c^{\mathcal{I}} = o$, for some $o \in C_1^{\mathcal{I}} \setminus C_2^{\mathcal{I}}$, and $\hat{A}^{\mathcal{I}} = \{c^{\mathcal{I}}\}$. This contradicts $\mathcal{O}_{C_1 \sqsubseteq C_2}$ being unsatisfiable.

"$\Rightarrow$" Let $\mathcal{T} \models C_1 \sqsubseteq C_2$, and suppose that $\mathcal{O}_{C_1 \sqsubseteq C_2}$ is satisfiable. Then there exists a model $\mathcal{I}$ be of $\mathcal{O}_{C_1 \sqsubseteq C_2}$. Then $\mathcal{I} \models \mathcal{T}$, and $\mathcal{I} \models C_1(c)$ and $\mathcal{I} \models \neg C_2(c)$, i.e., $\mathcal{I} \not\models C_1 \sqsubseteq C_2$. This contradicts $\mathcal{T} \models C_1 \sqsubseteq C_2$. $\square$

From role subsumption to ont. unsatisfiability for *DL-Lite*$_{\mathcal{R}}$

Theorem

Let $\mathcal{T}$ be a *DL-Lite*$_{\mathcal{R}}$ TBox and R_1, R_2 two general roles.
Then $\mathcal{T} \models R_1 \sqsubseteq R_2$ iff the ontology
$\mathcal{O}_{R_1 \sqsubseteq R_2} = \langle \mathcal{T} \cup \{\hat{P} \sqsubseteq R_1, \hat{P} \sqsubseteq \neg R_2\},\ \{\hat{P}(c_1, c_2)\}\rangle$ is unsatisfiable,
where $\hat{P}$ is an atomic role not in $\mathcal{T}$, and c_1, c_2 are two constants.

Intuitively, R_1 is subsumed by R_2 iff the smallest ontology containing $\mathcal{T}$ and implying both $R_1(c_1, c_2)$ and $\neg R_2(c_1, c_2)$ is unsatisfiable.

Proof (sketch).

Analogous to the one for concept subsumption. $\square$

Notice that *$\mathcal{O}_{R_1 \sqsubseteq R_2}$ is inherently a DL-Lite$_{\mathcal{R}}$ ontology.*

From role subsumption to ont. unsatisfiability for *DL-Lite*$_\mathcal{F}$

Theorem

Let $\mathcal{T}$ be a *DL-Lite*$_\mathcal{F}$ TBox, and Q_1, Q_2 two basic roles such that $Q_1 \neq Q_2$. Then,

1. $\mathcal{T} \models Q_1 \sqsubseteq Q_2$ iff Q_1 is unsatisfiable iff either $\exists Q_1$ or $\exists Q_1^-$ is unsatisfiable wrt $\mathcal{T}$, which can again be reduced to ontology unsatisfiability.
2. $\mathcal{T} \models \neg Q_1 \sqsubseteq Q_2$ iff $\mathcal{T}$ is unsatisfiable.
3. $\mathcal{T} \models Q_1 \sqsubseteq \neg Q_2$ iff either $\exists Q_1$ and $\exists Q_2$ are disjoint, or $\exists Q_1^-$ and $\exists Q_2^-$ are disjoint, iff either $\mathcal{T} \models \exists Q_1 \sqsubseteq \neg \exists Q_2$, or $\mathcal{T} \models \exists Q_1^- \sqsubseteq \neg \exists Q_2^-$, which can again be reduced to ontology unsatisfiability.

Notice that an inclusion of the form $\neg Q_1 \sqsubseteq \neg Q_2$ is equivalent to $Q_2 \sqsubseteq Q_1$, and therefore is considered in the first item.

From role subsumption to ont. unsatisfiability for $DL\text{-}Lite_{\mathcal{A}}$

Theorem

Let $\mathcal{T}$ be a $DL\text{-}Lite_{\mathcal{A}}$ TBox, and Q_1, Q_2 two basic roles such that $Q_1 \neq Q_2$. Then,

1. $\mathcal{T} \models Q_1 \sqsubseteq Q_2$ iff $\mathcal{O}_{Q_1 \sqsubseteq Q_2} = \langle \mathcal{T} \cup \{\hat{P} \sqsubseteq \neg Q_2\},\ \{Q_1(c_1, c_2), \hat{P}(c_1, c_2)\}\rangle$ is unsatisfiable, where $\hat{P}$ is an atomic role not in $\mathcal{T}$, and c_1, c_2 are two constants.
2. $\mathcal{T} \models \neg Q_1 \sqsubseteq Q_2$ iff $\mathcal{O}_{\neg Q_1 \sqsubseteq Q_2} = \langle \mathcal{T} \cup \{\hat{P} \sqsubseteq \neg Q_1, \hat{P} \sqsubseteq \neg Q_2\},\ \{\hat{P}(c_1, c_2)\}\rangle$ is unsatisfiable, where $\hat{P}$ is an atomic role not in $\mathcal{T}$, and c_1, c_2 are two constants.
3. $\mathcal{T} \models Q_1 \sqsubseteq \neg Q_2$ iff $\mathcal{O}_{Q_1 \sqsubseteq \neg Q_2} = \langle \mathcal{T},\ \{Q_1(c_1, c_2), Q_2(c_1, c_2)\}\rangle$ is unsatisfiable, where c_1, c_2 are two constants.

Notice that an inclusion of the form $\neg Q_1 \sqsubseteq \neg Q_2$ is equivalent to $Q_2 \sqsubseteq Q_1$, and therefore is considered in the first item.

Summary

- The results above tell us that we can support TBox reasoning services by relying on the ontology (un)satisfiability service.
- Ontology satisfiability is a form of reasoning over both the TBox and the ABox of the ontology.
- In the following, we first consider other TBox & ABox reasoning services, in particular **query answering**, and then turn back to ontology satisfiability.

Outline of Part 4.4

Outline of Part 4.4

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* 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 $\mathcal{O}$.
- $\mathcal{O} \models \neg P(c_1, c_2)$ iff $\mathcal{O}_{\neg P(c_1,c_2)} = \langle \mathcal{T},\ \mathcal{A} \cup \{P(c_1, c_2)\} \rangle$ is unsatisfiable.

Theorem

Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a *DL-Lite*$_\mathcal{F}$ ontology and P an atomic role.
Then $\mathcal{O} \models P(c_1, c_2)$ iff $\mathcal{O}$ is unsatisfiable or $P(c_1, c_2) \in \mathcal{A}$.

Theorem

Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a *DL-Lite*$_\mathcal{R}$ or *DL-Lite*$_\mathcal{A}$ ontology and P an atomic role.
Then $\mathcal{O} \models P(c_1, c_2)$ 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 $\mathcal{O}$.

Outline of Part 4.4

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 *DL-Lite*$_{\mathcal{A}}$ (and hence *DL-Lite*$_{\mathcal{F}}$ and *DL-Lite*$_{\mathcal{R}}$) enjoy FOL-rewritability of answering union 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

$$\begin{array}{rcl} Cl & \sqsubseteq & A \mid \exists Q \\ Q_1 & \sqsubseteq & Q_2 \end{array}$$

We call negative inclusions (NIs) assertions of the form

$$\begin{array}{rcl} Cl & \sqsubseteq & \neg A \mid \neg\exists Q \\ Q_1 & \sqsubseteq & \neg Q_2 \end{array}$$

Outline of Part 4.4

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*.

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

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

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

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

Query rewriting – Reduce step

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

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

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

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 $\mathcal{T}$:

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

  ('_' 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*$_\mathcal{A}$ 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** $PR := PR \cup \{$ *ApplyPI*$(q, g, I) \}$;
    for each g_1, g_2 in q **do**
      if g_1 and g_2 unify **then** $PR := PR \cup \{\tau($*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* – Example

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 $\mathsf{hasFather}(\_, y_2)$

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

$\Downarrow$

$\ldots$

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

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

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

Query answering over satisfiable *DL-Lite* ontologies

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

Theorem

Let $\mathcal{T}$ be a *DL-Lite* 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

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

As a consequence, query answering over a satisfiable *DL-Lite* 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**.

Canonical model of a *DL-Lite* ontology

The proof of the previous result exploits a fundamental property of *DL-Lite*, that relies on the following notion.

Def.: Canonical model

Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a *DL-Lite* ontology. A model $\mathcal{I}_{\mathcal{O}}$ of $\mathcal{O}$ is called **canonical** if for every model $\mathcal{I}$ of $\mathcal{O}$ there is a homomorphism from $\mathcal{I}_{\mathcal{O}}$ to $\mathcal{I}$.

Theorem

Every satisfiable *DL-Lite* ontology has a **canonical model**.

Properties of the canonical models of a *DL-Lite* ontology:

- A canonical model is in general infinite.
- All canonical models are homomorphically equivalent, hence we can do as if there was a single canonical model.

Query answering in *DL-Lite* – Canonical model

From the definition of canonical model, and since homomorphisms are closed under composition, we get that:

To compute the certain answer to a query q over an ontology $\mathcal{O}$, one could in principle evaluate q over a canonical model $\mathcal{I}_{\mathcal{O}}$ of $\mathcal{O}$.

- This does not give us directly an algorithm for query answering over an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, since $\mathcal{I}_{\mathcal{O}}$ may be infinite.
- However, one can show that evaluating q over $\mathcal{I}_{\mathcal{O}}$ amounts to evaluating the perfect rewriting $r_{q,\mathcal{T}}$ over $\mathcal{A}$.

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 *DL-Lite* ontologies can be done effectively using RDBMS technology:

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

Where:

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

Outline of Part 4.4

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 *DL-Lite* ontology where $\mathcal{T}$ contains **only PIs**.
Then, $\mathcal{O}$ is satisfiable.

Satisfiability of $DL\text{-}Lite_{\mathcal{A}}$ ontologies

Unsatisfiability in $DL\text{-}Lite_{\mathcal{A}}$ ontologies can be caused by **NIs** or by **functionality assertions**.

Example

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

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

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

Satisfiability of a *DL-Lite*$_{\mathcal{A}}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is reduced to 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 $\mathcal{A}$: $\text{teaches}(\texttt{john}, \texttt{fl})$
$\text{teaches}(\texttt{michael}, \texttt{fl})$

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

Lemma (Separation for *DL-Lite*$_\mathcal{A}$)

Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a *DL-Lite*$_\mathcal{A}$ 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 *DL-Lite*$_\mathcal{A}$ 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 $\textit{DL-Lite}_{\mathcal{A}}$

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

Theorem

Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a $\textit{DL-Lite}_{\mathcal{A}}$ 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. $\mathit{Eval}_{\mathrm{CWA}}(\mathit{PerfectRef}(q_N, \mathcal{T}_P), \mathcal{A})$ returns *true*.

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

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

Outline of Part 4.4

Complexity of query answering over satisfiable ontologies

Theorem

Query answering over *DL-Lite*$_{\mathcal{A}}$ 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. AC^0 is the data complexity of evaluating FOL queries over a DB. □

Complexity of ontology satisfiability

Theorem

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

1. $\mathrm{PT{\scriptsize IME}}$ 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

TBox reasoning over $\textit{DL-Lite}_{\mathcal{A}}$ 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. □

Outline of Part 4.4

10 TBox reasoning

11 TBox & ABox reasoning and query answering

12 Beyond *DL-Lite*
- Data complexity of query answering in DLs beyond *DL-Lite*
- NLogSpace-hard DLs
- PTime-hard DLs
- coNP-hard DLs
- Combining functionality and role inclusions
- Unique name assumption

Outline of Part 4.4

Beyond *DL-Lite*

We consider now DL languages that **extend DL-Lite with additional DL constructs** or with combinations of constructs that are not legal in *DL-Lite*.

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**
conjunction	$C_1 \sqcap C_2$	Doctor $\sqcap$ Male	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
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	$\{a \mid \forall b.\, (a,b) \in Q^{\mathcal{I}} \to b \in C^{\mathcal{I}}\}$

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

	Lhs	Rhs	Funct.	Role incl.	Data complexity of query answering
0	*DL-Lite*$_\mathcal{A}$		$\surd^*$	$\surd^*$	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 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**.

Observations

- *DL-Lite*-family is FOL-rewritable, hence AC^0 – holds also with n-ary relations $\rightsquigarrow$ *DLR-Lite*$_{\mathcal{F}}$ and *DLR-Lite*$_{\mathcal{R}}$.
- RDFS is a subset of *DL-Lite*$_{\mathcal{R}}$ $\rightsquigarrow$ 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 trivial DLs (line 10)

Outline of Part 4.4

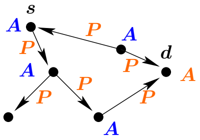
Qualified existential quantification in the lhs of inclusions

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

	Lhs	Rhs	$\mathcal{F}$	$\mathcal{R}$	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	$\mathcal{F}$	$\mathcal{R}$	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 $(\textbf{funct}\ P^-)$,
and by replacing again each occurrence of P^- with P', for a new role P'.

Outline of Part 4.4

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**.

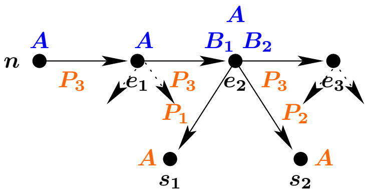
Reduction from Path System Accessibility

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

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

- 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 $s \in S$.

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



Result:

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

Outline of Part 4.4

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:

	Lhs	Rhs	$\mathcal{F}$	$\mathcal{R}$	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: $\varphi = c_1 \wedge c_2 \wedge c_3$, with

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

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 $\varphi = c_1 \land \cdots \land c_k$ 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 $c_j = v_{j_1} \lor v_{j_2} \lor \neg v_{j_3} \lor \neg v_{j_4}$:
  $P_1(\mathtt{c}_j, \mathtt{v}_{j_1})$, $P_2(\mathtt{c}_j, \mathtt{v}_{j_2})$, $N_1(\mathtt{c}_j, \mathtt{v}_{j_3})$, $N_2(\mathtt{c}_j, \mathtt{v}_{j_4})$
- $T(\mathtt{true})$, $F(\mathtt{false})$

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 $\alpha_\mathcal{I}(v_i) = \mathit{true}$ 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:

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

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

Outline of Part 4.4

Combining functionalities and role inclusions

Let $\textit{DL-Lite}_{\mathcal{FR}}$ be the DL that is the union of $\textit{DL-Lite}_{\mathcal{F}}$ and $\textit{DL-Lite}_{\mathcal{R}}$, 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 $\textit{DL-Lite}_{\mathcal{FR}}$ 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 $\mathcal{T}$: $A \sqsubseteq \exists P$ $\quad P \sqsubseteq S$
$\exists P^- \sqsubseteq A$ $\quad (\textbf{funct } S)$

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

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

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)

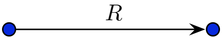
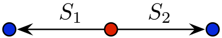
$$A \sqsubseteq \exists R_C \qquad R_C \sqsubseteq R \qquad \exists R_C^- \sqsubseteq C$$

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

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

Simulation of constructs (cont'd)

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

We use **reification** of roles:  

$$
\begin{array}{ccc}
S_{1,C} \sqsubseteq S_1 & S_{1,\neg C} \sqsubseteq S_1 & (\textbf{funct } S_1) \\
S_{2,C} \sqsubseteq S_2 & S_{2,\neg C} \sqsubseteq S_2 & (\textbf{funct } S_2) \\
\exists S_{1,C} \equiv \exists S_{2,C} & \exists S_{1,\neg C} \equiv \exists S_{2,\neg C} &
\end{array}
$$

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

$$
\begin{array}{cc}
\exists S_{2,C}^- \sqsubseteq C & \exists S_{2,\neg C}^- \sqsubseteq \neg C
\end{array}
$$

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

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 $DL\text{-}Lite_{\mathcal{FR}}$ 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 *DL-Lite*$_{\mathcal{A}}$ is necessary.

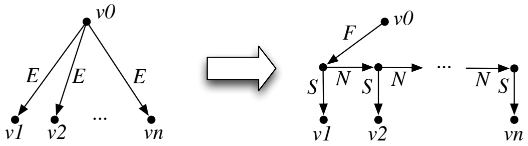
Outline of Part 4.4

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 $DL\text{-}Lite_{\mathcal{F}}$, and show that **without the UNA the data complexity increases**.

- We show how to reduce **reachability in directed graphs** to instance checking in $DL\text{-}Lite_{\mathcal{F}}$ 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)

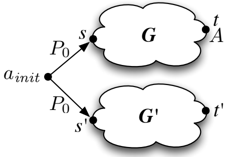
From G and two vertexes s and t of G, we define $\mathcal{O}_{una} = \langle \mathcal{T}_{una}, \mathcal{A}_G \rangle$:

- 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 *DL-Lite*$_\mathcal{F}$ (and hence also *DL-Lite*$_\mathcal{A}$) 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 *DL-Lite*$_\mathcal{F}$ and *DL-Lite*$_\mathcal{A}$ without the UNA is PTIME-complete in data complexity.

Part 4.5

Conclusions and references

Main publications

The results presented in Part 4 of the course have been published in the following papers:

- Reasoning and query answering in *DL-Lite*: [Calvanese *et al.*, 2005b; Calvanese *et al.*, 2006b; Calvanese *et al.*, 2007c; Calvanese *et al.*, 2007a; Artale *et al.*, 2009]
- Mapping to data sources and OBDA: [Calvanese *et al.*, 2006a; Calvanese *et al.*, 2008a; Poggi *et al.*, 2008a]
- Connection between description logics and conceptual modeling formalisms: [Calvanese *et al.*, 1998b; Berardi *et al.*, 2005; Artale *et al.*, 2007; Calvanese *et al.*, 2009b]
- Tool descriptions: [Acciarri *et al.*, 2005; Poggi *et al.*, 2008b; Rodríguez-Muro and Calvanese, 2008]
- Case studies: [Keet *et al.*, 2008; Amoroso *et al.*, 2008; Savo *et al.*, 2010]

A summary of most of the presented results and techniques, with detailed proofs is given in [Calvanese *et al.*, 2009a].

Query rewriting for more expressive ontology languages

The result presented in Part 4 of the course have recently been extended to more expressive ontology languages, using different techniques:

- In [Artale *et al.*, 2009] various *DL-Lite* extensions are considered, providing a comprehensive treatment of the expressiveness/complexity trade-off for the *DL-Lite* family and related logics:
  - number restrictions besides functionality;
  - conjunction on the left-hand side of inclusions (horn logics);
  - boolean constructs;
  - constraints on roles, such as (ir)reflexivity, (a)symmetry, transitivity;
  - presence and absence of the unique name assumption.
- Alternative query rewriting techniques based on resolution, and applicable also to more expressive logics (leading to recursive rewritings) [Pérez-Urbina *et al.*, 2009].
- Query rewriting techniques for database inspired constraint languages [Calì *et al.*, 2009a; Calì *et al.*, 2009b].

Further theoretical work

The results presented in this course have also inspired additional work relevant for ontology-based data access:

- We have considered mainly query answering. However, several other ontology-based services are of importance:
  - write-also access: updating a data source through an ontology [De Giacomo *et al.*, 2009; Calvanese *et al.*, 2010; Zheleznyakov *et al.*, 2010]
  - modularity and minimal module extraction [Kontchakov *et al.*, 2008; Kontchakov *et al.*, 2009]
  - privacy aware data access [Calvanese *et al.*, 2008b]
  - meta-level reasoning and query answering, a la RDFS [De Giacomo *et al.*, 2008]
  - provenance and explanation [Borgida *et al.*, 2008]
- Reasoning with respect to finite models only [Rosati, 2008].
- We have dealt only with the static aspects of information systems. However a crucial issue is how to deal with **dynamic aspects**. Preliminary results are in [Calvanese *et al.*, 2007d]. The general problem is largely unexplored.

Work on most of these issues is still ongoing.

Further practical and experimental work

The theoretical results indicate a good computational behaviour in the size of the data. However, performance is a critical issue in practice:

- The rewriting consists of a large number of CQs. Query containment can be used to prune the rewriting. This is already implemented in the QuOnto system, but requires further optimizations.
- The SQL queries generated by the mapping unfolding are not easy to process by the DBMS engine (e.g., they may contain complex joins on skolem terms computed on the fly).
  Different mapping unfolding strategies have a strong impact on computational complexity. Experimentation is ongoing to assess the tradeoff.
- Further extensive experimentations are ongoing:
  - on artificially generated data;
  - on real-world use cases.

Conclusions

- Ontology-based data access is **ready for prime time**.
- QuOnto provides serious proof of concept of this.
- We are successfully applying QuOnto in various **full-fledged case studies**.
- We are currently **looking for projects** where to apply such technology further!

Acknowledgements

Additional people involved in this work:

- Sapienza Università di Roma
  - Claudio Corona
  - Giuseppe De Giacomo
  - Domenico Lembo
  - Maurizio Lenzerini
  - Antonella Poggi
  - Riccardo Rosati
  - Marco Ruzzi
  - Domenico Fabio Savo
- Free University of Bozen-Bolzano
  - Mariano Rodriguez Muro
  - Manfred Gerstgasser
- Many students (thanks!)

References I

[Acciarri *et al.*, 2005] Andrea Acciarri, Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Mattia Palmieri, and Riccardo Rosati.
QuOnto: QUerying ONTOlogies.
In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, pages 1670–1671, 2005.

[Amoroso *et al.*, 2008] Alfonso Amoroso, Gennaro Esposito, Domenico Lembo, Paolo Urbano, and Raffaele Vertucci.
Ontology-based data integration with Mastro-i for configuration and data management at SELEX Sistemi Integrati.
In *Proc. of the 16th Ital. Conf. on Database Systems (SEBD 2008)*, pages 81–92, 2008.

[Artale *et al.*, 2007] Alessandro Artale, Diego Calvanese, Roman Kontchakov, Vladislav Ryzhikov, and Michael Zakharyaschev.
Reasoning over extended ER models.
In *Proc. of the 26th Int. Conf. on Conceptual Modeling (ER 2007)*, volume 4801 of *Lecture Notes in Computer Science*, pages 277–292. Springer, 2007.

References II

[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.*, 2003] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors.
The Description Logic Handbook: Theory, Implementation and Applications.
Cambridge University Press, 2003.

[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.

[Berardi *et al.*, 2005] Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo.
Reasoning on UML class diagrams.
Artificial Intelligence, 168(1–2):70–118, 2005.

References III

[Borgida *et al.*, 2008] Alexander Borgida, Diego Calvanese, and Mariano Rodríguez-Muro.
Explanation in the *DL-Lite* family of description logics.
In *Proc. of the 7th Int. Conf. on Ontologies, DataBases, and Applications of Semantics (ODBASE 2008)*, volume 5332 of *Lecture Notes in Computer Science*, pages 1440–1457. Springer, 2008.

[Calì and Kifer, 2006] Andrea Calì and Michael Kifer.
Containment of conjunctive object meta-queries.
In *Proc. of the 32nd Int. Conf. on Very Large Data Bases (VLDB 2006)*, pages 942–952, 2006.

[Calì *et al.*, 2009a] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz.
Datalog$^{\pm}$: a unified approach to ontologies and integrity constraints.
In *Proc. of the 12th Int. Conf. on Database Theory (ICDT 2009)*, pages 14–30, 2009.

[Calì *et al.*, 2009b] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz.
A general Datalog-based framework for tractable query answering over ontologies.
In *Proc. of the 28th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2009)*, pages 77–86, 2009.

References IV

[Calvanese *et al.*, 1998a] 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.*, 1998b] Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi.
Description logics for conceptual data modeling.
In Jan Chomicki and Günter Saake, editors, *Logics for Databases and Information Systems*, pages 229–264. Kluwer Academic Publishers, 1998.

[Calvanese *et al.*, 2005a] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo 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/`, 2005.

References V

[Calvanese *et al.*, 2005b] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati.

DL-Lite: Tractable description logics for ontologies.

In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, pages 602–607, 2005.

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

Linking data to ontologies: The description logic $\textit{DL-Lite}_{A}$.

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/`, 2006.

[Calvanese *et al.*, 2006b] 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 VI

[Calvanese *et al.*, 2007a] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati.
Can OWL model football leagues?
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/`, 2007.

[Calvanese *et al.*, 2007b] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati.
EQL-Lite: Effective first-order query processing in description logics.
In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, pages 274–279, 2007.

[Calvanese *et al.*, 2007c] 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.

References VII

[Calvanese *et al.*, 2007d] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati.

Actions and programs over description logic ontologies.

In *Proc. of the 20th Int. Workshop on Description Logic (DL 2007)*, volume 250 of *CEUR Electronic Workshop Proceedings*, http://ceur-ws.org/, pages 29–40, 2007.

[Calvanese *et al.*, 2008a] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Riccardo Rosati, and Marco Ruzzi.

Data integration through $\textit{DL-Lite}_{\mathcal{A}}$ ontologies.

In Klaus-Dieter Schewe and Bernhard Thalheim, editors, *Revised Selected Papers of the 3rd Int. Workshop on Semantics in Data and Knowledge Bases (SDKB 2008)*, volume 4925 of *Lecture Notes in Computer Science*, pages 26–47. Springer, 2008.

[Calvanese *et al.*, 2008b] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati.

View-based query answering over description logic ontologies.

In *Proc. of the 11th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2008)*, pages 242–251, 2008.

References VIII

[Calvanese *et al.*, 2009a] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodríguez-Muro, and Riccardo Rosati.
Ontologies and databases: The *DL-Lite* approach.
In Sergio Tessaris and Enrico Franconi, editors, *Semantic Technologies for Informations Systems – 5th Int. Reasoning Web Summer School (RW 2009)*, volume 5689 of *Lecture Notes in Computer Science*, pages 255–356. Springer, 2009.

[Calvanese *et al.*, 2009b] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati.
Conceptual modeling for data integration.
In Alex T. Borgida, Vinay Chaudhri, Paolo Giorgini, and Eric Yu, editors, *Conceptual Modeling: Foundations and Applications – Essays in Honor of John Mylopoulos*, volume 5600 of *Lecture Notes in Computer Science*, pages 173–197. Springer, 2009.

[Calvanese *et al.*, 2010] Diego Calvanese, Evgeny Kharlamov, Werner Nutt, and Dmitriy Zheleznyakov.
Updating ABoxes in *DL-Lite*.
In *Proc. of the 4th Alberto Mendelzon Int. Workshop on Foundations of Data Management (AMW 2010)*, volume 619 of *CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/`*, pages 3.1–3.12, 2010.

References IX

[Chandra and Merlin, 1977] Ashok K. Chandra and Philip M. Merlin.
Optimal implementation of conjunctive queries in relational data bases.
In *Proc. of the 9th ACM Symp. on Theory of Computing (STOC'77)*, pages 77–90, 1977.

[De Giacomo *et al.*, 2008] Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati.
Towards higher-order *DL-Lite*.
In *Proc. of the 21st Int. Workshop on Description Logic (DL 2008)*, volume 353 of *CEUR Electronic Workshop Proceedings*, http://ceur-ws.org/, 2008.

[De Giacomo *et al.*, 2009] Giuseppe De Giacomo, Maurizio Lenzerini, Antonella Poggi, and Riccardo Rosati.
On instance-level update and erasure in description logic ontologies.
J. of Logic and Computation, Special Issue on Ontology Dynamics, 19(5):745–770, 2009.

[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.

References X

[Donini, 2003] Francesco M. Donini.
Complexity of reasoning.
In Baader et al. [2003], chapter 3, pages 96–136.

[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.

[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.

References XI

[Keet *et al.*, 2008] C. Maria Keet, Ronell Alberts, Aurona Gerber, and Gibson Chimamiwa.
Enhancing web portals with Ontology-Based Data Access: the case study of South Africa's Accessibility Portal for people with disabilities.
In *Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2008)*, volume 432 of *CEUR Electronic Workshop Proceedings*, `http://ceur-ws.org/`, 2008.

[Kolaitis and Vardi, 1998] Phokion G. Kolaitis and Moshe Y. Vardi.
Conjunctive-query containment and constraint satisfaction.
In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 205–213, 1998.

[Kontchakov *et al.*, 2008] Roman Kontchakov, Frank Wolter, and Michael Zakharyaschev.
Can you tell the difference between *DL-Lite* ontologies?
In *Proc. of the 11th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2008)*, pages 285–295, 2008.

References XII

[Kontchakov *et al.*, 2009] R. Kontchakov, L. Pulina, U. Sattler, T. Schneider, P. Selmer, F. Wolter, and M. Zakharyaschev.
Minimal module extraction from DL-Lite ontologies using QBF solvers.
In *Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI 2009)*, pages 836–840, 2009.

[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.

[Möller and Haarslev, 2003] Ralf Möller and Volker Haarslev.
Description logic systems.
In Baader et al. [2003], chapter 8, pages 282–305.

References XIII

[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.

[Pérez-Urbina *et al.*, 2009] Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks.
A comparison of query rewriting techniques for *DL-lite*.
In *Proc. of the 22nd Int. Workshop on Description Logic (DL 2009)*, volume 477 of *CEUR Electronic Workshop Proceedings*, `http://ceur-ws.org/`, 2009.

[Poggi *et al.*, 2008a] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati.
Linking data to ontologies.
J. on Data Semantics, X:133–173, 2008.

[Poggi *et al.*, 2008b] Antonella Poggi, Mariano Rodríguez-Muro, and Marco Ruzzi.
Ontology-based database access with DIG-Mastro and the OBDA Plugin for Protégé.
In Kendall Clark and Peter F. Patel-Schneider, editors, *Proc. of the 4th Int. Workshop on OWL: Experiences and Directions (OWLED 2008 DC)*, 2008.

References XIV

[Rodríguez-Muro and Calvanese, 2008] Mariano Rodríguez-Muro and Diego Calvanese.
Towards an open framework for ontology based data access with Protégé and DIG 1.1.
In *Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2008)*, volume 432 of *CEUR Electronic Workshop Proceedings*, `http://ceur-ws.org/`, 2008.

[Rosati, 2008] Riccardo Rosati.
Finite model reasoning in *DL-Lite*.
In *Proc. of the 5th European Semantic Web Conf. (ESWC 2008)*, 2008.

[Savo *et al.*, 2010] Domenico Fabio Savo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodríguez-Muro, Vittorio Romagnoli, Marco Ruzzi, and Gabriele Stella.
Mastro at work: Experiences on ontology-based data access.
In *Proc. of the 23rd Int. Workshop on Description Logic (DL 2010)*, volume 573 of *CEUR Electronic Workshop Proceedings*, `http://ceur-ws.org/`, pages 20–31, 2010.

[Schaerf, 1993] Andrea Schaerf.
On the complexity of the instance checking problem in concept languages with existential quantification.
J. of Intelligent Information Systems, 2:265–278, 1993.

References XV

[Vardi, 1982] Moshe Y. Vardi.
The complexity of relational query languages.
In *Proc. of the 14th ACM SIGACT Symp. on Theory of Computing (STOC'82)*, pages 137–146, 1982.

[Zheleznyakov *et al.*, 2010] Dmitriy Zheleznyakov, Diego Calvanese, Evgeny Kharlamov, and Werner Nutt.
Updating TBoxes in *DL-Lite*.
In *Proc. of the 23rd Int. Workshop on Description Logic (DL 2010)*, volume 573 of *CEUR Electronic Workshop Proceedings*, `http://ceur-ws.org/`, pages 102–113, 2010.