Overview of Part 2: Ontology-based access to information

1. Introduction to ontology-based access to information
   - Introduction to ontologies
   - Ontology languages

2. Description Logics and the DL-Lite family
   - An introduction to DLs
   - DLs as a formal language to specify ontologies
   - Queries in Description Logics
   - The DL-Lite family of tractable DLs

3. Linking ontologies to relational data
   - The impedance mismatch problem
   - OBDA systems
   - Query answering in OBDA systems

4. Reasoning in the DL-Lite family
   - TBox reasoning
   - TBox & ABox reasoning
   - Complexity of reasoning in Description Logics
   - Reasoning in the Description Logic DL-Lite_A
Chapter I

Introduction to ontology-based access to information

Outline

1 Introduction to ontologies

2 Ontology languages
Different meanings of “Semantics”

- Part of linguistics that studies the meaning of words and phrases.
- **Meaning** of a set of symbols in some representation scheme.
  Provides a means to specify and communicate the intended meaning of a set of “syntactic” objects.
- **Formal semantics of a language** (e.g., an artificial language).
  (Meta-mathematical) mechanism to associate to each sentence in a language an element of a symbolic domain that is “outside the language”.

In information systems, meanings 2 and 3 are the relevant ones:
- In order to talk about semantics we need a representation scheme, i.e., an ontology.
- . . . but 2 makes no sense without 3.
**Def.: Ontology**

is a representation scheme that describes a **formal conceptualization** of a domain of interest.

The specification of an ontology comprises several levels:

- **Meta-level**: specifies a set of **modeling categories**.
- **Intensional level**: specifies a set of **conceptual elements** (instances of categories) and of rules to describe the conceptual structures of the domain.
- **Extensional level**: specifies a set of **instances** of the conceptual elements described at the intensional level.
Ontologies at the core of information systems

The usage of all system resources (data and services) is done through the domain conceptualization.

Ontology mediated access to data

Desiderata: achieve logical transparency in access to data:

- **Hide** to the user where and how data are stored.
- Present to the user a **conceptual view** of the data.
- Use a **semantically rich formalism** for the conceptual view.

_We will see that this setting is similar to the one of Data Integration. The difference is that here the ontology provides a rich conceptual description as the information managed by the system._
Ontologies at the core of cooperation

The cooperation between systems is done at the level of the conceptualization.

Three novel challenges

- Languages
- Methodologies
- Tools

... for specifying, building, and managing ontologies to be used in information systems.
Challenge 1: Ontology languages

- Several proposals for ontology languages have been made.
- **Tradeoff** between expressive power of the language and computational complexity of dealing with (i.e., performing inference over) ontologies specified in that language.
- Usability needs to be addressed.

In this course:
- We discuss variants of ontology languages suited for managing **ontologies in information systems**.
- We study the above mentioned **tradeoff** ...
- ... paying particular attention to the aspects related to data management.

Challenge 2: Methodologies

- Developing and dealing with ontologies is a complex and challenging task.
- Developing **good ontologies** is even more challenging.
- It requires to master the technologies based on semantics, which in turn requires good knowledge about the languages, their semantics, and the implications it has w.r.t. reasoning over the ontology.

In this course:
- We study in depth the **semantics of ontologies**, with an emphasis on their relationship to data in information sources.
- We thus lay the **foundations for the development of methodologies**, though we do not discuss specific ontology-development methodologies here.
Challenge 3: Tools

- According to the principle that “there is no meaning without a language with a formal semantics”, the formal semantics becomes the solid basis for dealing with ontologies.
- Hence every kind of access to an ontology (to extract information, to modify it, etc.), requires to fully take into account its semantics.
- We need to resort to tools that provide capabilities to perform automated reasoning over the ontology, and the kind of reasoning should be sound and complete w.r.t. the formal semantics.

In this course:

- We discuss the requirements for such ontology management tools.
- We will work with a tool that has been specifically designed for optimized access to information sources through ontologies.

A challenge across the three challenges: Scalability

When we want to use ontologies to access information sources, we have to address the three challenges of languages, methodologies, and tools by taking into account scalability w.r.t.:

- the size of (the intensional level of) the ontology
- the number of ontologies
- the size of the information sources that are accessed through the ontology/ontologies.

In this course we pay particular attention to the third aspect, since we work under the realistic assumption that the extensional level (i.e., the data) largely dominates in size the intensional level of an ontology.
Outline

1 Introduction to ontologies

2 Ontology languages
   • Elements of an ontology language
   • Intensional level of an ontology language
   • Extensional level of an ontology language
   • Ontologies and other formalisms
   • Queries

Elements of an ontology language

• Syntax
   • Alphabet
   • Languages constructs
   • Sentences to assert knowledge

• Semantics
   • Formal meaning

• Pragmatics
   • Intended meaning
   • Usage
Static vs. dynamic aspects

The aspects of the domain of interest that can be modeled by an ontology language can be classified into:

- **Static aspects**
  - Are related to the structuring of the domain of interest.
  - Supported by virtually all languages.

- **Dynamic aspects**
  - Are related to how the elements of the domain of interest evolve over time.
  - Supported only by some languages, and only partially (cf. services).

Before delving into the dynamic aspects, we need a good understanding of the static ones.

In this course we concentrate essentially on the static aspects.

Intensional level of an ontology language

An ontology language for expressing the intensional level usually includes:

- Concepts
- Properties of concepts
- Relationships between concepts, and their properties
- Axioms
- Queries

Ontologies are typically *rendered as diagrams* (e.g., Semantic Networks, Entity-Relationship schemas, UML Class Diagrams).
### Concepts

**Def.:** **Concept**

Is an element of an ontology that denotes a collection of instances (e.g., the set of “employees”).

We distinguish between:

- **Intensional definition:** specification of name, properties, relations, . . .
- **Extensional definition:** specification of the instances

Concepts are also called **classes, entity types, frames**.
Properties

Def.: Property

Is an element of an ontology that qualifies another element (e.g., a concept or a relationship).

Property definition (intensional and extensional):

- **Name**
- **Type:** may be either
  - atomic (integer, real, string, enumerated, ...), or
  - e.g., eye-color → { blu, brown, green, grey }
  - structured (date, set, list, ...)
  - e.g., date → day/month/year
- The definition may also specify a default value.

Properties are also called attributes, features, slots.

Relationships

Def.: Relationship

Is an element of an ontology that expresses an association among concepts.

We distinguish between:

- **Intensional definition:**
  - specification of involved concepts
  - e.g., worksFor is defined on Employee and Project
- **Extensional definition:**
  - specification of the instances of the relationship, called facts
  - e.g., worksFor(domenico, tones)

Relationships are also called associations, relationship types, roles.
**Axioms**

**Def.: Axiom**

Is a logical formula that expresses at the intensional level a condition that must be satisfied by the elements at the extensional level.

Different kinds of axioms/conditions:

- subclass relationships, e.g., Manager $\sqsubseteq$ Employee
- equivalences, e.g., Manager $\equiv$ AreaManager $\sqcup$ TopManager
- disjointness, e.g., AreaManager $\sqcap$ TopManager $\equiv$ ⊥
- (cardinality) restrictions, e.g., each Employee worksFor at least 3 Project

Axioms are also called assertions. A special kind of axioms are definitions.

---

**Extensional level of an ontology language**

At the extensional level we have individuals and facts:

- An **instance** represents an individual (or object) in the extension of a concept. e.g., domenico is an instance of Employee
- A **fact** represents a relationship holding between instances. e.g., worksFor(domenico, tones)
The three levels of an ontology

Comparison with other formalisms

- **Ontology languages vs. knowledge representation languages:**
  Ontologies are knowledge representation schemas.

- **Ontology vs. logic:**
  Logic is the tool for assigning semantics to ontology languages.

- **Ontology languages vs. conceptual data models:**
  Conceptual schemas are special ontologies, suited for conceptualizing a single logical model (database).

- **Ontology languages vs. programming languages:**
  Class definitions are special ontologies, suited for conceptualizing a single structure for computation.
Classification of ontology languages

- Graph-based
  - Semantic networks
  - Conceptual graphs
  - UML class diagrams, Entity-Relationship schemas

- Frame based
  - Frame Systems
  - OKBC, XOL

- Logic based
  - Description Logics (e.g., $SHOIQ$, $DLR$, $DL$-Lite, OWL, . . . )
  - Rules (e.g., RuleML, LP/Prolog, F-Logic)
  - First Order Logic (e.g., KIF)
  - Non-classical logics (e.g., non-monotonic, probabilistic)

Queries

Queries may be posed over an ontology.

**Def.: Query**

Is an expression at the intensional level denoting a (possibly structured) collection of individuals satisfying a given condition.

**Def.: Meta-Query**

Is an expression at the meta level denoting a collection of ontology elements satisfying a given condition.

**Note:** One may also conceive queries that span across levels (object-meta queries), cf. [RDF], [CK06]
### Ontology languages vs. query languages

**Ontology languages:**
- Tailored for capturing intensional relationships.
- 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.

Instead, **when querying** a data source (either directly, or via the ontology), to retrieve the data of interest, general forms of **joins** are required.

*It follows that the constructs for queries may be quite different from the constructs used in the ontology to form concepts and relationships.*

---

#### Example of query

![Ontology diagram with examples of queries](image)

\[
q(ce, cm, se, sm) \leftarrow \exists e, p, m. \\
\text{worksFor}(e, p) \land \text{manages}(m, p) \land \text{boss}(m, e) \land \text{empCode}(e, ce) \land \\
\text{empCode}(m, cm) \land \text{salary}(e, se) \land \text{salary}(m, sm) \land se \geq sm
\]
Query answering under different assumptions

Depending on the setting, query answering may have different meanings:

- Traditional databases $\leadsto$ complete information
- Ontologies (or knowledge bases) $\leadsto$ incomplete information

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).

$\leadsto$ Query answering amounts to query evaluation, which is computationally easy.
Query answering in traditional databases (cont’d)

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:**
\[ q(x) \leftarrow \exists p. \text{Manager}(x), \text{Project}(p), \text{worksFor}(x,p) \]

**Answer:**
\{ john \}
Query answering over 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.

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 over ontologies – Example

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

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

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

Answer:  \{ john, nick, mary \}

Query answering over ontologies – Example 2

Each person has a father, who is a person.

DB:  Person ⊇ { john, nick, toni }
     hasFather ⊇ { (john,nick), (nick,toni) }

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

Answers:  
\[
to \ q_1: \ (john,nick), (nick,toni) \\
to \ q_2: \ (john, nick, toni) \\
to \ q_3: \ (john, nick, toni) \\
to \ q_4: \ \}
\]
QA over ontologies – Andrea’s Example (cont’d)

Manager is partitioned into AreaManager and TopManager.

Employee ⊇ \{ andrea, nick, mary, john \}
Manager ⊇ \{ andrea, nick, mary \}
AreaManager ⊇ \{ nick \}
TopManager ⊇ \{ mary \}
supervisedBy ⊇ \{(john, andrea), (john, mary)\}
officeMate ⊇ \{(mary, andrea), (andrea, nick)\}

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

Answer: \{ john \}

To determine this answer, we need to resort to reasoning by cases.
Query answering in Ontology-Based Data Access

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

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

Researchers are starting only now to tackle this difficult and challenging problem. In this course we will study state-of-the-art technology in this area.

Chapter II

Description Logics and the DL-Lite family
Outline

3. A gentle introduction to Description Logics
   - Ingredients of Description Logics
   - Description language
   - Description Logics ontologies
   - Reasoning in Description Logics

4. DLs as a formal language to specify ontologies

5. Queries in Description Logics

6. The DL-Lite family of tractable Description Logics
What are Description Logics?

Description Logics (DLs) \cite{BCM03} are logics specifically designed to represent and reason on structured knowledge.

The domain of interest is composed of objects and is structured into:

- **concepts**, which correspond to classes, and denote sets of objects
- **roles**, which correspond to (binary) relationships, and denote binary relations on objects

The knowledge is asserted through so-called assertions, i.e., logical axioms.

Origins of Description Logics

DLs stem from early days Knowledge Representation formalisms (late '70s, early '80s):

- Semantic Networks: graph-based formalism, used to represent the meaning of sentences
- Frame Systems: frames used to represent prototypical situations, antecedents of object-oriented formalisms

Problems: no clear semantics, reasoning not well understood

**Description Logics** (a.k.a. Concept Languages, Terminological Languages) developed starting in the mid '80s, with the aim of providing semantics and inference techniques to knowledge representation systems.
### Current applications of Description Logics

DLs have evolved from being used “just” in KR.

**Novel applications of DLs:**
- **Databases:**
  - schema design, schema evolution
  - query optimization
  - integration of heterogeneous data sources, data warehousing
- **Conceptual modeling**
- **Foundation for the Semantic Web** (variants of OWL correspond to specific DLs)
- ...
A description language provides the means for defining:

- **concepts**, corresponding to classes: interpreted as sets of objects;
- **roles**, corresponding to relationships: interpreted as binary relations on objects.

To define concepts and roles:

- We start from a (finite) alphabet of **atomic concepts** and **atomic roles**, i.e., simply names for concept and roles.
- Then, by applying specific **constructors**, we can build **complex concepts** and **roles**, starting from the atomic ones.

A **description language** is characterized by the set of constructs that are available for that.
The formal semantics of DLs is given in terms of interpretations.

**Def.:** An interpretation $\mathcal{I} = (\Delta^I, \cdot^I)$ consists of:

- a nonempty set $\Delta^I$, the domain of $\mathcal{I}$
- an interpretation function $\cdot^I$, which maps
  - each individual $a$ to an element $a^I$ of $\Delta^I$
  - each atomic concept $A$ to a subset $A^I$ of $\Delta^I$
  - each atomic role $P$ to a subset $P^I$ of $\Delta^I \times \Delta^I$

**Note:** A DL interpretation is analogous to a FOL interpretation, except that, by tradition, it is specified in terms of a function $\cdot^I$ rather than a set of (unary and binary) relations.

The interpretation function is extended to complex concepts and roles according to their syntactic structure.

<table>
<thead>
<tr>
<th>Construct</th>
<th>Syntax</th>
<th>Example</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>atomic concept</td>
<td>$A$</td>
<td>Doctor</td>
<td>$A^I \subseteq \Delta^I$</td>
</tr>
<tr>
<td>atomic role</td>
<td>$P$</td>
<td>hasChild</td>
<td>$P^I \subseteq \Delta^I \times \Delta^I$</td>
</tr>
<tr>
<td>atomic negation</td>
<td>$\neg A$</td>
<td>$\neg$Doctor</td>
<td>$\Delta^I \setminus A^I$</td>
</tr>
<tr>
<td>conjunction</td>
<td>$C \sqcap D$</td>
<td>Hum $\sqcap$ Male</td>
<td>$C^I \cap D^I$</td>
</tr>
<tr>
<td>(unqual.) exist. res.</td>
<td>$\exists R$</td>
<td>$\exists$hasChild</td>
<td>${ a \mid \exists b. (a, b) \in R^I }$</td>
</tr>
<tr>
<td>value restriction</td>
<td>$\forall R.C$</td>
<td>$\forall$hasChild.Male</td>
<td>${ a \mid \forall b. (a, b) \in R^I \rightarrow b \in C^I }$</td>
</tr>
<tr>
<td>bottom</td>
<td>$\bot$</td>
<td></td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

($C$, $D$ denote arbitrary concepts and $R$ an arbitrary role)

The above constructs form the basic language $\mathcal{AL}$ of the family of $\mathcal{AL}$ languages.
### Additional concept and role constructors

<table>
<thead>
<tr>
<th>Construct</th>
<th>$\mathcal{AL}$</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>disjunction</td>
<td>$U$</td>
<td>$\mathcal{C} \sqcup \mathcal{D}$</td>
<td>$\mathcal{C}^I \sqcup \mathcal{D}^I$</td>
</tr>
<tr>
<td>top</td>
<td>$\top$</td>
<td></td>
<td>$\Delta^I$</td>
</tr>
<tr>
<td>qual. exist. res.</td>
<td>$\mathcal{E}$</td>
<td>$\exists R. \mathcal{C}$</td>
<td>${ a \mid \exists b. (a, b) \in \mathcal{R}^I \land b \in \mathcal{C}^I }$</td>
</tr>
<tr>
<td>(full) negation</td>
<td>$\mathcal{C}$</td>
<td>$\neg \mathcal{C}$</td>
<td>$\Delta^I \setminus \mathcal{C}^I$</td>
</tr>
<tr>
<td>number restrictions</td>
<td>$\mathcal{N}$</td>
<td>$(\geq k \mathcal{R})$</td>
<td>${ a \mid #{b \mid (a, b) \in \mathcal{R}^I } \geq k }$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(\leq k \mathcal{R})$</td>
<td>${ a \mid #{b \mid (a, b) \in \mathcal{R}^I } \leq k }$</td>
</tr>
<tr>
<td>qual. number restrictions</td>
<td>$\mathcal{Q}$</td>
<td>$(\geq k \mathcal{R}, \mathcal{C})$</td>
<td>${ a \mid #{b \mid (a, b) \in \mathcal{R}^I \land b \in \mathcal{C}^I } \geq k }$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(\leq k \mathcal{R}, \mathcal{C})$</td>
<td>${ a \mid #{b \mid (a, b) \in \mathcal{R}^I \land b \in \mathcal{C}^I } \leq k }$</td>
</tr>
<tr>
<td>inverse role</td>
<td>$\mathcal{I}$</td>
<td>$\mathcal{R}^{-}$</td>
<td>${ (a, b) \mid (b, a) \in \mathcal{R}^I }$</td>
</tr>
<tr>
<td>role closure</td>
<td>$\mathcal{T}$</td>
<td>$\mathcal{R}^{*}$</td>
<td>$(\mathcal{R}^I)^*$</td>
</tr>
</tbody>
</table>

Many different DL constructs and their combinations have been investigated.

### Further examples of DL constructs

- **Disjunction:** $\forall \text{hasChild}.(\text{Doctor} \sqcup \text{Lawyer})$
- **Qualified existential restriction:** $\exists \text{hasChild}.\text{Doctor}$
- **Full negation:** $\neg(\text{Doctor} \sqcup \text{Lawyer})$
- **Number restrictions:** $(\geq 2 \text{hasChild}) \sqcap (\leq 1 \text{ sibling})$
- **Qualified number restrictions:** $(\geq 2 \text{hasChild}. \text{Doctor})$
- **Inverse role:** $\forall \text{hasChild}^{-}.\text{Doctor}$
- **Reflexive-transitive role closure:** $\exists \text{hasChild}^{*}.\text{Doctor}$
Reasoning on concept expressions

An interpretation $I$ is a **model** of a concept $C$ if $C^I \neq \emptyset$.

**Basic reasoning tasks:**

- **Concept satisfiability**: does $C$ admit a model?
- **Concept subsumption** $C \sqsubseteq D$: does $C^I \subseteq D^I$ hold for all interpretations $I$?

Subsumption is used to build the concept hierarchy:

```
  Human
  Man
  Woman
  Father
  HappyFather
```

*Note: (1) and (2) are mutually reducible if DL is propositionally closed.*

### Complexity of reasoning on concept expressions

**Complexity of concept satisfiability:** [DLNN97]

<table>
<thead>
<tr>
<th>Description language</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{AL}$, $\mathcal{ALN}$</td>
<td>PTIME</td>
</tr>
<tr>
<td>$\mathcal{ALC\Upsilon}$, $\mathcal{ALC\Upsilon N}$</td>
<td>NP-complete</td>
</tr>
<tr>
<td>$\mathcal{ALE}$</td>
<td>coNP-complete</td>
</tr>
<tr>
<td>$\mathcal{ALC}$, $\mathcal{ALC\Upsilon N}$, $\mathcal{ALC\Upsilon I}$, $\mathcal{ALC\Upsilon QI}$</td>
<td>PSPACE-complete</td>
</tr>
</tbody>
</table>

**Observations:**

- Two sources of complexity:
  - union ($\cup$) of type NP,
  - existential quantification ($\exists$) of type coNP.

  When they are combined, the complexity jumps to PSPACE.

- Number restrictions ($\mathcal{N}$) do not add to the complexity.
Structural properties vs. asserted properties

We have seen how to build complex concept and roles expressions, which allow one to denote classes with a complex structure.

However, in order to represent real world domains, one needs the ability to assert properties of classes and relationships between them (e.g., as done in UML class diagrams).

The assertion of properties is done in DLs by means of an ontology (or knowledge base).

Description Logics ontology (or knowledge base)

Is a pair $O = (T, A)$, where $T$ is a TBox and $A$ is an ABox:

**Def.: Description Logics TBox**

Consists of a set of assertions on concepts and roles:
- Inclusion assertions on concepts: $C_1 \subseteq C_2$
- Inclusion assertions on roles: $R_1 \subseteq R_2$
- Property assertions on (atomic) roles:
  - (transitive $P$)
  - (symmetric $P$)
  - (domain $P$ $C$)
  - (functional $P$)
  - (reflexive $P$)
  - (range $P$ $C$)  

**Def.: Description Logics ABox**

Consists of a set of membership assertions on individuals:
- for concepts: $A(c)$
- for roles: $P(c_1, c_2)$ (we use $c_i$ to denote individuals)
Description Logics knowledge base – Example

Note: We use $C_1 \equiv C_2$ as an abbreviation for $C_1 \subseteq C_2$, $C_2 \subseteq C_1$.

TBox assertions:
- Inclusion assertions on concepts:
  - $\text{Father} \equiv \text{Human} \cap \text{Male} \cap \exists \text{hasChild}$
  - $\text{HappyFather} \sqsubseteq \text{Father} \cap \forall \text{hasChild}.(\text{Doctor} \sqcup \text{Lawyer} \sqcup \text{HappyPerson})$
  - $\text{HappyAnc} \sqsubseteq \forall \text{descendant}.\text{HappyFather}$
  - $\text{Teacher} \sqsubseteq \neg \text{Doctor} \sqcap \neg \text{Lawyer}$
- Inclusion assertions on roles:
  - $\text{hasChild} \sqsubseteq \text{descendant}$
  - $\text{hasFather} \sqsubseteq \text{hasChild}^\neg$
- Property assertions on roles:
  - (transitive descendant), (reflexive descendant), (functional hasFather)

ABox membership assertions:
- $\text{Teacher}(\text{mary})$, $\text{hasFather}(\text{mary}, \text{john})$, $\text{HappyAnc}(\text{john})$

Semantics of a Description Logics knowledge base

The semantics is given by specifying when an interpretation $\mathcal{I}$ satisfies an assertion:
- $C_1 \subseteq C_2$ is satisfied by $\mathcal{I}$ if $C_1^\mathcal{I} \subseteq C_2^\mathcal{I}$.
- $R_1 \subseteq R_2$ is satisfied by $\mathcal{I}$ if $R_1^\mathcal{I} \subseteq R_2^\mathcal{I}$.
- A property assertion $(\text{prop } P)$ is satisfied by $\mathcal{I}$ if $P^\mathcal{I}$ is a relation that has the property $\text{prop}$.

(Note: domain and range assertions can be expressed by means of concept inclusion assertions.)
- $A(c)$ is satisfied by $\mathcal{I}$ if $c^\mathcal{I} \in A^\mathcal{I}$.
- $P(c_1, c_2)$ is satisfied by $\mathcal{I}$ if $(c_1^\mathcal{I}, c_2^\mathcal{I}) \in P^\mathcal{I}$.

We adopt the unique name assumption, i.e., $c_1^\mathcal{I} \neq c_2^\mathcal{I}$, for $c_1 \neq c_2$. 
Models of a Description Logics ontology

Def.: **Model** of a DL knowledge base

An interpretation $\mathcal{I}$ is a model of $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ if it satisfies all assertions in $\mathcal{T}$ and all assertions in $\mathcal{A}$.

$\mathcal{O}$ is said to be **satisfiable** if it admits a model.

The fundamental reasoning service from which all other ones can be easily derived is . . .

Def.: **Logical implication**

$\mathcal{O}$ logically implies an assertion $\alpha$, written $\mathcal{O} \models \alpha$, if $\alpha$ is satisfied by all models of $\mathcal{O}$.

TBox reasoning

- **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., $\mathcal{I} \models C_1 \subseteq C_2$.
- **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., $\mathcal{T} \models C_1 \cap C_2 \equiv \bot$.
- **Functionality implication**: A functionality assertion (funct $R$) is logically implied by $\mathcal{T}$ if for every model $\mathcal{I}$ of $\mathcal{T}$, we have that $(o, o_1) \in R^\mathcal{I}$ and $(o, o_2) \in R^\mathcal{I}$ implies $o_1 = o_2$, i.e., $\mathcal{T} \models \text{(funct } R\text{)}$.

Analogous definitions hold for role satisfiability, subsumption, equivalence, and disjointness.
Reasoning over an ontology

- **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 $\mathcal{O}$, 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 $R$ in $\mathcal{O}$, i.e., whether $\mathcal{O} \models R(c_1, c_2)$.

- **Query Answering**: see later . . .

Reasoning in Description Logics – Example

**TBox**:
- Inclusion assertions on concepts:
  - Father $\equiv$ Human $\sqcap$ Male $\sqcap$ $\exists\text{hasChild}$
  - HappyFather $\sqsubseteq$ Father $\sqcap$ $\forall\text{hasChild}.(\text{Doctor} \sqcup \text{Lawyer} \sqcup \text{HappyPerson})$
  - HappyAnc $\sqsubseteq$ $\forall\text{descendant}.\text{HappyFather}$
  - Teacher $\sqsubseteq$ $\neg\text{Doctor} \sqcap \neg\text{Lawyer}$
- Inclusion assertions on roles:
  - hasChild $\sqsubseteq$ descendant
  - hasFather $\sqsubseteq$ hasChild$^\neg$
- Property assertions on roles:
  - (transitive descendant), (reflexive descendant), (functional hasFather)

The above TBox logically implies: HappyAncestor $\sqsubseteq$ Father.

- Membership assertions:
  - Teacher(mary), hasFather(mary, john), HappyAnc(john)

The above TBox and ABox logically imply: HappyPerson(mary)
Complexity of reasoning over DL ontologies

Reasoning over DL ontologies is much more complex than reasoning over concept expressions:

**Bad news:**
- without restrictions on the form of TBox assertions, reasoning over DL ontologies is already *ExpTime-hard*, even for very simple DLs (see, e.g., [Don03]).

**Good news:**
- We can add a lot of expressivity (i.e., essentially all DL constructs seen so far), while still staying within the *ExpTime* upper bound.
- There are DL reasoners that perform reasonably well in practice for such DLs (e.g., Racer, Pellet, Fact++, . . .) [MH03].

Outline

1. A gentle introduction to Description Logics
2. DLs as a formal language to specify ontologies
   - DLs to specify ontologies
   - DLs vs. OWL
   - DLs vs. UML Class Diagrams
3. Queries in Description Logics
4. The *DL-Lite* family of tractable Description Logics
Relationship between DLs and ontology formalisms

- DLs are nowadays advocated to provide the foundations for ontology languages.
- Different versions of the W3C standard **Web Ontology Language (OWL)** have been defined as syntactic variants of certain DLs.
- DLs are also ideally suited to capture the fundamental features of conceptual modeling formalisms used in information systems design:
  - **Entity-Relationship diagrams**, used in database conceptual modeling
  - **UML Class Diagrams**, used in the design phase of software applications

We briefly overview these correspondences, highlighting essential DL constructs, also in light of the tradeoff between expressive power and computational complexity of reasoning.

**DLs vs. OWL**

The Web Ontology Language (OWL) comes in different variants:

- **OWL1 Lite** is a variant of the DL $SHI^F(D)$, where:
  - $S$ stands for $\mathcal{ALC}$ extended with **transitive roles**, 
  - $\mathcal{H}$ stands for **role hierarchies** (i.e., role inclusion assertions),
  - $\mathcal{I}$ stands for **inverse roles**,
  - $\mathcal{F}$ stands for functionality of roles,
  - $(D)$ stand for **data types**, which are necessary in any practical knowledge representation language.

- **OWL1 DL** is a variant of $SHOIN(D)$, where:
  - $O$ stands for **nominals**, which means the possibility of using individuals in the TBox (i.e., the intensional part of the ontology),
  - $\mathcal{N}$ stands for (unqualified) **number restrictions**,
A new version of OWL, **OWL2**, is currently being standardized:

- **OWL2 DL** is a variant of $SROIQ(D)$, which adds to OWL1 DL several constructs, while still preserving satisfiability of reasoning.
  - $Q$ stands for qualified number restrictions.
  - $R$ stands for regular role hierarchies, where role chaining might be used in the left-hand side of role inclusion assertions, with suitable acyclicity conditions.

- OWL2 defines also three **profiles**: OWL2 QL, OWL2 EL, OWL2 EL.
  - 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 (see later).

**Note:** all constructs come also in the Data... instead of Object... variant.
DL axioms vs. OWL axioms

<table>
<thead>
<tr>
<th>OWL axiom</th>
<th>DL syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>SubClassOf</td>
<td>$C_1 \sqsubseteq C_2$</td>
<td>Human $\sqsubseteq$ Animal $\sqcap$ Biped</td>
</tr>
<tr>
<td>EquivalentClasses</td>
<td>$C_1 \equiv C_2$</td>
<td>Man $\equiv$ Human $\sqcap$ Male</td>
</tr>
<tr>
<td>DisjointClasseses</td>
<td>$C_1 \sqsubsetneq C_2$</td>
<td>Man $\sqsubsetneq$ Male</td>
</tr>
<tr>
<td>SameIndividual</td>
<td>${a_1} \equiv {a_2}$</td>
<td>${\text{presBush}} \equiv {\text{G.W.Bush}}$</td>
</tr>
<tr>
<td>DifferentIndividuals</td>
<td>${a_1} \subsetneq \neg{a_2}$</td>
<td>${\text{john}} \subsetneq \neg{\text{peter}}$</td>
</tr>
<tr>
<td>SubObjectPropertyOf</td>
<td>$P_1 \sqsubseteq P_2$</td>
<td>hasDaughter $\sqsubseteq$ hasChild</td>
</tr>
<tr>
<td>EquivalentObjectProperties</td>
<td>$P_1 \equiv P_2$</td>
<td>hasCost $\equiv$ hasPrice</td>
</tr>
<tr>
<td>InverseObjectProperties</td>
<td>$P_1 \sqsubseteq P_2^\bot$</td>
<td>hasChild $\equiv$ hasParent $^\bot$</td>
</tr>
<tr>
<td>TransitiveObjectProperty</td>
<td>$P^+ \sqsubseteq P$</td>
<td>ancestor $^+$ $\sqsubseteq$ ancestor</td>
</tr>
<tr>
<td>FunctionalObjectProperty</td>
<td>$\top \sqsubseteq (\leq 1 P)$</td>
<td>$\top \sqsubseteq (\leq 1 \text{ hasFather})$</td>
</tr>
</tbody>
</table>

There is a tight correspondence between variants of DLs and UML Class Diagrams [BCDG05].

- We can devise two transformations:
  - one that associates to each UML Class Diagram $D$ a DL TBox $T_D$.
  - one that associates to each DL TBox $T$ a UML Class Diagram $D_T$.

- The transformations are not model-preserving, but are based on a correspondence between instantiations of the Class Diagram and models of the associated ontology.

- The transformations are **satisfiability-preserving**, i.e., a class $C'$ is consistent in $D$ iff the corresponding concept is satisfiable in $T$. 
A gentle introduction to DLs

DLs to specify ontologies

Queries in DLs

The DL-Lite family

Chap. 2: Description Logics and the DL-Lite family

Encoding UML Class Diagrams in DLs

The ideas behind the encoding of a UML Class Diagram $D$ in terms of a DL TBox $T_D$ 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.

We illustrate the encoding by means of an example.

Note: Domain and range of associations are expressed by means of concept inclusions.
Encoding DL TBoxes in UML Class Diagrams

The encoding of an $\mathcal{ALC}$ TBox $T$ in terms of a UML Class Diagram $T_D$ is based on the following observations:

- We can restrict the attention to $\mathcal{ALC}$ TBoxes, that are constituted by concept inclusion assertions of a simplified form (single atomic concept on the left, and a single concept constructor on the right).
- For each such inclusion assertion, the encoding introduces a portion of UML Class Diagram, that may refer to some common classes.

Reasoning in the encoded $\mathcal{ALC}$-fragment is already ExpTime-hard. From this, we obtain:

**Theorem**

Reasoning over UML Class Diagrams is ExpTime-hard.

Reasoning on UML Class Diagrams using DLs

- The two encodings show that DL TBoxes and UML Class Diagrams essentially have the same expressive power.
- Hence, reasoning over UML Class Diagrams has the same complexity as reasoning over ontologies in expressive DLs, i.e., ExpTime-complete.
- The high complexity is caused by:
  - the possibility to use disjunction (covering constraints)
  - the interaction between role inclusions and functionality constraints (maximum 1 cardinality)

Without (1) and restricting (2), reasoning becomes simpler [ACK+07]:

- $\text{NLogSpace}$-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

5 A gentle introduction to Description Logics

4 DLs as a formal language to specify ontologies

Queries in Description Logics

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

6 The DL-Lite family of tractable Description Logics
Traditionally, simple concept (or role) expressions have been considered as queries over DL ontologies.

We need more complex forms of queries, as those used in databases.

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

is a conjunctive query \( q(\bar{x}) \leftarrow \bar{y}. \text{conj}(\bar{x}, \bar{y}) \) where each atom in the body \( \text{conj}(\bar{x}, \bar{y}) \):

- has as predicate symbol an atomic concept or role of \( T \),
- may use variables in \( \bar{x} \) and \( \bar{y} \),
- may use constants that are individuals of \( A \).

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

Conjunctive query over the above ontology:

\[
q(x, y) \leftarrow \exists p. \text{Employee}(x), \text{Employee}(y), \text{Project}(p), \\
\text{boss}(x, y), \text{worksFor}(x, p), \text{worksFor}(y, p)
\]
Certain answers to a query

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

**Def.:** The **answer** to $q(\vec{x})$ over $I$, denoted $q^I$, is the set of **tuples** $\vec{c}$ of **constants** of $A$ such that the formula $\exists \vec{y}. \text{conj}(\vec{c}, \vec{y})$ evaluates to true in $I$.

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

**Def.:** The **certain answers** to $q(\vec{x})$ over $O = \langle T, A \rangle$, denoted $\text{cert}(q, O)$, are the **tuples** $\vec{c}$ of **constants** of $A$ such that $\vec{c} \in q^I$, for every model $I$ of $O$.

Query answering over ontologies

**Def.:** **Query answering** over an ontology $O$ is the problem of computing the certain answers to a query over $O$.

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

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

**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 $O$ (in this case $\vec{c}$ is the empty tuple).
A gentle introduction to DLs

DLs to specify ontologies

Queries in DLs

The DL-Lite family

Certain answers

Chap. 2: Description Logics and the DL-Lite family

Query answering over ontologies – Example

\[ \text{\texttt{TBox}}: \begin{align*}
\exists & \text{hasFather} \sqsubseteq \text{Person} \\
\exists & \text{hasFather} \sqsubseteq \text{Person} \\
\text{Person} & \sqsubseteq \exists \text{hasFather}
\end{align*} \]

\[ \text{\texttt{ABox}}: \begin{align*}
\text{Person}(\text{john}), & \text{Person}(\text{nick}), \text{Person}(\text{toni}) \\
\text{hasFather}(\text{john}, \text{nick}), & \text{hasFather}(\text{nick}, \text{toni})
\end{align*} \]

Queries:

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

Certain answers:

\[ \text{cert}(q_1, (T, A)) = \{ (\text{john}, \text{nick}), (\text{nick}, \text{toni}) \} \]
\[ \text{cert}(q_2, (T, A)) = \{ \text{john}, \text{nick}, \text{toni} \} \]
\[ \text{cert}(q_3, (T, A)) = \{ \text{john}, \text{nick}, \text{toni} \} \]
\[ \text{cert}(q_4, (T, A)) = \{ \} \]

Unions of conjunctive queries

We consider also unions of CQs over an ontology.

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

\[ q(\vec{x}) = \exists \vec{y}_1. \text{conj}(\vec{x}, \vec{y}_1) \lor \cdots \lor \exists \vec{y}_k. \text{conj}(\vec{x}, \vec{y}_k) \]

where each \( \exists \vec{y}_i. \text{conj}(\vec{x}, \vec{y}_i) \) is the body of a CQ.

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

**Example**

\[ q(x) \leftarrow (\text{Manager}(x) \land \text{worksFor}(x, \text{tones})) \lor \]
\[ (\exists y. \text{boss}(x, y) \land \text{worksFor}(y, \text{tones})) \]

We typically use the Datalog notation:

\[ q(x) \leftarrow \text{Manager}(x), \text{worksFor}(x, \text{tones}) \]
\[ q(x) \leftarrow \exists y. \text{boss}(x, y) \land \text{worksFor}(y, \text{tones}) \]
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**: TBox and query are considered fixed, and only the size of the ABox (i.e., the data) matters.
- **Query complexity**: TBox and ABox are considered fixed, and only the size of the query matters.
- **Schema complexity**: ABox and query are considered fixed, and only the size of the TBox (i.e., the schema) matters.
- **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).

$\sim \text{ Data complexity}$ is the relevant complexity measure.

Answering (U)CQs over DL ontologies has been studied extensively:

- **Combined complexity**:
  - NP-complete for plain databases (i.e., with an empty TBox)
  - ExpTime-complete for $\mathcal{ALC}$ [CDGL98, Lut07]
  - 2ExpTime-complete for very expressive DLs (with inverse roles) [CDGL98, Lut07]
- **Data complexity**:
  - in LogSpace for plain databases
  - coNP-hard with disjunction in the TBox [DLNS94, CDGL+06b]
  - coNP-complete for very expressive DLs [LR98, OCE06, GHLS07]

**Questions**

- Can we find interesting families of DLs for which the query answering problem can be solved efficiently?
- If yes, can we leverage relational database technology for query answering?
The **DL-Lite** family of tractable Description Logics

- The **DL-Lite** family
- Syntax of **DL-Lite**\(_F\) and **DL-Lite**\(_R\)
- Semantics of **DL-Lite**
- Properties of **DL-Lite**
- Syntax and Semantics of **DL-Lite**\(_A\)

**The **DL-Lite** family**

- Is a family of DLs optimized according to the tradeoff between expressive power and data complexity of query answering.
- We present first two incomparable languages of this family, **DL-Lite**\(_F\), **DL-Lite**\(_R\) (we use **DL-Lite** to refer to both).
- We will see that **DL-Lite** has nice computational properties:
  - \(\text{PTime}\) in the size of the TBox (schema complexity)
  - \(\text{LogSpace}\) in the size of the ABox (data complexity)
  - enjoys FOL-rewritability
- We will see that **DL-Lite**\(_F\) and **DL-Lite**\(_R\) are in some sense the maximal DLs with these nice computational properties, which are lost if the two logics are combined, and with minimal additions of constructs.
- We will see, however, that a restricted combination of **DL-Lite**\(_F\) and **DL-Lite**\(_R\) is possible, without losing the computational properties.

Hence, **DL-Lite** provides a positive answer to our basic questions, and sets the foundations for Ontology-Based Data Access.
DL-Lite\(_F\) ontologies

**TBox assertions:**

- Concept inclusion assertions: \( Cl \sqsubseteq Cr \), with:
  
  \[
  Cl \rightarrow A \mid \exists Q \\
  Cr \rightarrow A \mid \exists Q \mid \neg A \mid \neg \exists Q \\
  Q \rightarrow P \mid P^-
  \]

- Functionality assertions: \((\text{funct } Q)\)

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

**Observations:**

- Captures all the basic constructs of UML Class Diagrams and ER
- Notable exception: covering constraints in generalizations.

DL-Lite\(_R\) ontologies

**TBox assertions:**

- Concept inclusion assertions: \( Cl \sqsubseteq Cr \), with:
  
  \[
  Cl \rightarrow A \mid \exists Q \\
  Cr \rightarrow A \mid \exists Q \mid \neg A \mid \neg \exists Q \\
  Q \rightarrow R \mid \neg Q
  \]

- Role inclusion assertions: \( Q \sqsubseteq R \), with:
  
  \[
  Q \rightarrow P \mid P^- \\
  R \rightarrow Q \mid \neg Q
  \]

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

**Observations:**

- Drops functional restrictions in favor of ISA between roles.
- Extends (the DL fragment of) the ontology language RDFS.
## Semantics of DL-Lite

<table>
<thead>
<tr>
<th>Construct</th>
<th>Syntax</th>
<th>Example</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>atomic conc.</td>
<td>$A$</td>
<td>Doctor</td>
<td>$A^I \subseteq \Delta^I$</td>
</tr>
<tr>
<td>exist. restr.</td>
<td>$\exists Q$</td>
<td>$\exists \text{child}^-$</td>
<td>${d \mid \exists e. (d, e) \in Q^I}$</td>
</tr>
<tr>
<td>at. conc. neg.</td>
<td>$\neg A$</td>
<td>$\neg \text{Doctor}$</td>
<td>$\Delta^I \setminus A^I$</td>
</tr>
<tr>
<td>conc. neg.</td>
<td>$\neg \exists Q$</td>
<td>$\neg \exists \text{child}$</td>
<td>$\Delta^I \setminus (\exists Q)^I$</td>
</tr>
<tr>
<td>atomic role</td>
<td>$P$</td>
<td>child</td>
<td>$P^I \subseteq \Delta^I \times \Delta^I$</td>
</tr>
<tr>
<td>inverse role</td>
<td>$P^-$</td>
<td>$\text{child}^-$</td>
<td>${(o, o') \mid (o', o) \in P^I}$</td>
</tr>
<tr>
<td>role negation</td>
<td>$\neg Q$</td>
<td>$\neg \text{manages}$</td>
<td>$(\Delta^I_0 \times \Delta^I_0) \setminus Q^I$</td>
</tr>
<tr>
<td>conc. incl.</td>
<td>$Cl \subseteq Cr$</td>
<td>$\text{Father} \subseteq \exists \text{child}$</td>
<td>$Cl^I \subseteq Cr^I$</td>
</tr>
<tr>
<td>role incl.</td>
<td>$Q \subseteq R$</td>
<td>$\text{hasFather} \subseteq \text{child}^-$</td>
<td>$Q^I \subseteq R^I$</td>
</tr>
<tr>
<td>funct. asser.</td>
<td>$(\text{funct } Q)$</td>
<td>$(\text{funct succ})$</td>
<td>$\forall d, e, e'. (d, e) \in Q^I \land (d, e') \in Q^I \rightarrow e = e'$</td>
</tr>
<tr>
<td>mem. asser.</td>
<td>$A(c)$</td>
<td>$\text{Father(bob)}$</td>
<td>$c^I \in A^I$</td>
</tr>
<tr>
<td>mem. asser.</td>
<td>$P(c_1, c_2)$</td>
<td>$\text{child(bob, ann)}$</td>
<td>$(c^I_1, c^I_2) \in P^I$</td>
</tr>
</tbody>
</table>

### Capturing basic ontology constructs in DL-Lite

| ISA between classes | $A_1 \subseteq A_2$ |
| Disjointness between classes | $A_1 \subseteq \neg A_2$ |
| Domain and range of relations | $\exists P \subseteq A_1, \exists P^- \subseteq A_2$ |
| Mandatory participation | $A_1 \subseteq \exists P, A_2 \subseteq \exists P^-$ |
| Functionality of relations (in DL-Lite$_F$) | $(\text{funct } P), (\text{funct } P^-)$ |
| ISA between relations (in DL-Lite$_R$) | $Q_1 \subseteq Q_2$ |
| Disjointness between relations (in DL-Lite$_R$) | $Q_1 \subseteq \neg Q_2$ |
**DL-Lite – Example**

Additionally, in $\text{DL-Lite}_F$: $(\text{funct } \text{manages})$, $(\text{funct } \text{manages}^-)$, ... in $\text{DL-Lite}_R$: $\text{manages} \sqsubseteq \text{worksFor}$

*Note*: in $\text{DL-Lite}$ we cannot capture: – completeness of the hierarchy, – number restrictions

**Properties of $\text{DL-Lite}$**

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

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

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

  $$Cl \sqsubseteq Cr_1 \cap Cr_2 \quad \text{is equivalent to} \quad Cl \sqsubseteq Cr_1$$

- 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 $\text{DL-Lite}_F$

$\text{DL-Lite}_F$ does not enjoy the finite model property.

Example

TBox $T$: \[
\begin{align*}
\text{Nat} &\sqsubseteq \exists \text{succ} \\
\exists \text{succ}^- &\sqsubseteq \text{Nat} \\
\text{Zero} &\sqsubseteq \text{Nat} \sqcap \neg \exists \text{succ}^- \\
&\text{(funct succ$^-$)}
\end{align*}
\]

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

$\mathcal{O} = \langle T, A \rangle$ 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 $\text{DL-Lite}_R$

- The TBox may contain cyclic dependencies.
- $\text{DL-Lite}_R$ 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: $A_1 \sqsubseteq \exists Q_{A_2}$; $\exists Q_{A_2} \sqsubseteq A_2$

In this way, we can consider $\exists Q.A$ in the right-hand side of an inclusion assertion as an abbreviation.
What is missing in \( DL-Lite \) wrt popular data models?

Let us consider UML class diagrams that have the following features:

- functionality of associations (i.e., roles)
- inclusion (i.e., ISA) between associations
- attributes of concepts and associations, possibly functional
- covering constraints in hierarchies

What can we capture of these while maintaining FOL-rewritability?

1. We can forget about covering constraints, since they make query answering coNP-hard in data complexity (see Part 3).
2. Attributes of concepts are “syntactic sugar” (they could be modeled by means of roles), but their functionality is an issue.
3. We could also add attributes of roles (we won’t discuss this here).
4. Functionality and role inclusions are present separately (in \( DL-Lite_F \) and \( DL-Lite_R \)), but were not allowed to be used together.

\( DL-Lite_A \) is a Description Logic designed to capture as much features as possible of conceptual data models, while preserving nice computational properties for query answering.

- Allows for both functionality assertions and role inclusion assertions, but restricts in a suitable way their interaction.
- 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, etc.
- Values are connected to objects through attributes, rather than roles (we consider here only concept attributes and not role attributes [CDGL+06a]).
- Enjoys FOL-rewritability, and hence is LogSpace in data complexity.
A gentle introduction to DLs

Syntax and Semantics of DL-Lite

Semantics of \( DL-Lite_A \) – Objects vs. values

We make use of an alphabet \( \Gamma \) of constants, partitioned into:

- an alphabet \( \Gamma_O \) of object constants.
- an alphabet \( \Gamma_V \) of value constants, in turn partitioned into alphabets \( \Gamma_{V_i} \), one for each RDF datatype \( T_i \).

The interpretation domain \( \Delta^I \) is partitioned into:

- a domain of objects \( \Delta^I_O \)
- a domain of values \( \Delta^I_V \)

The semantics of \( DL-Lite_A \) descriptions is determined as usual, considering the following:

- The interpretation \( C^I \) of a concept \( C \) is a subset of \( \Delta^I_O \).
- The interpretation \( R^I \) of a role \( R \) is a subset of \( \Delta^I_O \times \Delta^I_O \).
- The interpretation \( \text{val}(v) \) of each value constant \( v \) in \( \Gamma_V \) and RDF datatype \( T_i \) is given a priori (e.g., all strings for \( \text{xsd:string} \)).
- The interpretation \( V^I \) of an attribute \( V \) is a subset of \( \Delta^I_O \times \Delta^I_V \).
**Semantics of the \textit{DL-Lite}_A constructs**

<table>
<thead>
<tr>
<th>Construct</th>
<th>Syntax</th>
<th>Example</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>top concept</td>
<td>⊤</td>
<td>∃c</td>
<td>(\top_c = \Delta_I^c)</td>
</tr>
<tr>
<td>atomic concept</td>
<td>A</td>
<td>Doctor</td>
<td>(A \subseteq \Delta_I^A)</td>
</tr>
<tr>
<td>existential restriction</td>
<td>∃Q</td>
<td>∃\text{child}</td>
<td>({o \mid \exists o', (o, o') \in Q^4})</td>
</tr>
<tr>
<td>qualified exist. restriction</td>
<td>∃Q,C</td>
<td>∃\text{child.Male}</td>
<td>({o \mid \exists o', (o, o') \in Q^4 \land o' \in C^4})</td>
</tr>
<tr>
<td>concept negation</td>
<td>¬</td>
<td>¬∃\text{child}</td>
<td>(\Delta_I^c - B^4)</td>
</tr>
<tr>
<td>attribute domain</td>
<td>δ(U)</td>
<td>δ(salary)</td>
<td>({o \mid \exists v, (o, v) \in U^4})</td>
</tr>
<tr>
<td>atomic role</td>
<td>P</td>
<td>child</td>
<td>(P^4 \subseteq \Delta_I^P \times \Delta_I^c)</td>
</tr>
<tr>
<td>inverse role</td>
<td>P⁻</td>
<td>child⁻</td>
<td>({(o, o') \mid (o', o) \in P^4})</td>
</tr>
<tr>
<td>role negation</td>
<td>¬Q</td>
<td>¬\text{manages}</td>
<td>((\Delta_I^c \times \Delta_I^P) \setminus Q^4)</td>
</tr>
<tr>
<td>top domain</td>
<td>⊤_D</td>
<td></td>
<td>(\top_D = \Delta_I^D)</td>
</tr>
<tr>
<td>datatype</td>
<td>T_i</td>
<td>xsd:int</td>
<td>(\text{val}(T_i) \subseteq \Delta_I^{T_i})</td>
</tr>
<tr>
<td>attribute range</td>
<td>ρ(U)</td>
<td>ρ(salary)</td>
<td>({v \mid \exists o, (o, v) \in U^4})</td>
</tr>
<tr>
<td>atomic attribute</td>
<td>U</td>
<td>salary</td>
<td>(U^4 \subseteq \Delta_I^c \times \Delta_I^{T_i})</td>
</tr>
<tr>
<td>attribute negation</td>
<td>¬U</td>
<td>¬salary</td>
<td>((\Delta_I^c \times \Delta_I^{T_i}) \setminus U^4)</td>
</tr>
<tr>
<td>object constant</td>
<td>c</td>
<td>john</td>
<td>(c^k \in \Delta_I^c)</td>
</tr>
<tr>
<td>value constant</td>
<td>v</td>
<td>'john'</td>
<td>(\text{val}(v) \in \Delta_I^{T_i})</td>
</tr>
</tbody>
</table>

**\textit{DL-Lite}_A assertions**

TBox assertions can have the following forms:

- \(B \sqsubseteq C\) \quad \text{concept inclusion assertion}
- \(Q \sqsubseteq R\) \quad \text{role inclusion assertion}
- \(E \sqsubseteq F\) \quad \text{value-domain inclusion assertion}
- \(U \sqsubseteq V\) \quad \text{attribute inclusion assertion}
- \(\text{funct } Q\) \quad \text{role functionality assertion}
- \(\text{funct } U\) \quad \text{attribute functionality assertion}

ABox assertions: \(A(c), \ P(c, c'), \ U(c, d),\)

where \(c, c'\) are object constants

\(d\) is a value constant
Semantics of the $DL-Lite_{\mathcal{A}}$ assertions

<table>
<thead>
<tr>
<th>Assertion</th>
<th>Syntax</th>
<th>Example</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>conc. incl.</td>
<td>$B \sqsubseteq C$</td>
<td>Father $\sqsubseteq \exists \text{child}$</td>
<td>$B^T \sqsubseteq C^T$</td>
</tr>
<tr>
<td>role incl.</td>
<td>$Q \sqsubseteq R$</td>
<td>father $\sqsubseteq \text{anc}$</td>
<td>$Q^T \sqsubseteq R^T$</td>
</tr>
<tr>
<td>v.dom. incl.</td>
<td>$E \sqsubseteq F$</td>
<td>$\rho(\text{age}) \sqsubseteq \text{xsd:int}$</td>
<td>$E^T \sqsubseteq F^T$</td>
</tr>
<tr>
<td>attr. incl.</td>
<td>$U \sqsubseteq V$</td>
<td>offPhone $\sqsubseteq \text{phone}$</td>
<td>$U^T \sqsubseteq V^T$</td>
</tr>
<tr>
<td>role funct.</td>
<td>$(\text{funct } Q)$</td>
<td>$(\text{funct } \text{father})$</td>
<td>$\forall o, o', o'' : (o, o', o'') \in Q^T \land (o, o'') \in Q^T \rightarrow o' = o''$</td>
</tr>
<tr>
<td>att. funct.</td>
<td>$(\text{funct } U)$</td>
<td>$(\text{funct } \text{ssn})$</td>
<td>$\forall o, v, v' : (o, v, v') \in U^T \land (o, v') \in U^T \rightarrow v = v'$</td>
</tr>
<tr>
<td>mem. asser.</td>
<td>$A(c)$</td>
<td>Father(bob)</td>
<td>$c^T \in A^T$</td>
</tr>
<tr>
<td>mem. asser.</td>
<td>$P(c_1, c_2)$</td>
<td>child(bob, ann)</td>
<td>$(c_1^T, c_2^T) \in P^T$</td>
</tr>
<tr>
<td>mem. asser.</td>
<td>$U(c, d)$</td>
<td>phone(bob, '2345')</td>
<td>$(c^T, \text{val}(d)) \in U^T$</td>
</tr>
</tbody>
</table>

Restriction on TBox assertions in $DL-Lite_{\mathcal{A}}$ ontologies

We will see that, to ensure FOL-rewritability, we have to impose a restriction on the use of functionality and role/attribute inclusions.

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

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

Formally:

- If $\exists P.C$ or $\exists P^-.C$ appears in $T$,
  then $(\text{funct } P)$ and $(\text{funct } P^-)$ are not in $T$.
- If $Q \sqsubseteq P$ or $Q \sqsubseteq P^-$ is in $T$,
  then $(\text{funct } P)$ and $(\text{funct } P^-)$ are not in $T$.
- If $U_1 \sqsubseteq U_2$ is in $T$,
  then $(\text{funct } U_2)$ is not in $T$. 
**DL-Lite\(_A\) – Example**

<table>
<thead>
<tr>
<th>Role</th>
<th>Subordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manager</td>
<td>Employee, AreaManager, TopManager</td>
</tr>
<tr>
<td>AreaManager</td>
<td>Manager, ¬TopManager</td>
</tr>
<tr>
<td>TopManager</td>
<td>Manager</td>
</tr>
<tr>
<td>Employee</td>
<td>δ(salary), ρ(salary)</td>
</tr>
<tr>
<td>Project</td>
<td>Employee, Project</td>
</tr>
</tbody>
</table>

Note: in DL-Lite\(_A\) we still cannot capture:
- completeness of the hierarchy
- number restrictions

### Complexity results for DL-Lite

- We have seen that DL-Lite\(_A\) can capture the essential features of prominent conceptual modeling formalisms.
- In the following, we will analyze reasoning in DL-Lite, and establish the following characterization of its computational properties:
  - Ontology satisfiability is polynomial in the size of TBox and ABox.
  - Query answering is:
    - PTime in the size of the TBox.
    - LogSpace in the size of the ABox, and FOL-rewritable, which means that we can leverage for it relational database technology.
- We will also see that DL-Lite is essentially the maximal DL enjoying these nice computational properties.

From (1), (2), and (3) we get the following claim:

**DL-Lite** is the representation formalism that is best suited to underly Ontology-Based Data Management systems.
Chapter III

Linking ontologies to data

Outline

1. The impedance mismatch problem
2. Ontology-Based Data Access Systems
3. Query answering in Ontology-Based Data Access Systems
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.
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.

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.

\[ D_1[SSN: String, PrName: String] \]

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

Intuitively:
- An employee should be created from her SSN: \( \text{pers}(SSN) \)
- A project should be created from its Name: \( \text{proj}(PrName) \)
Creating object identifiers

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

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

Example

- If a person is identified by its SSN, we can introduce a function symbol $\text{pers}/1$. If VRD56B25 is a SSN, then $\text{pers}(\text{VRD56B25})$ denotes a person.
- If a person is identified by its name and dateOfBirth, we can introduce a function symbol $\text{pers}/2$. Then $\text{pers}(\text{Vardi}, 25/2/56)$ 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.: Mapping assertion between a database and a TBox

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

$$\Phi \rightsquigarrow \Psi$$

where

- $\Phi$ is an arbitrary SQL query of arity $n > 0$ over $\mathcal{D}$.
- $\Psi$ is a conjunctive query over $\mathcal{T}$ of arity $n' > 0$ without non-distinguished variables, possibly involving variable terms.
Mapping assertions – Example

**D₁[SSN: String, PrName: String]**
Employees and Projects they work for

**D₂[Code: String, Salary: Int]**
Employee’s Code with salary

**D₃[Code: String, SSN: String]**
Employee’s Code with SSN

\[ m₁ : \text{SELECT } SSN, \text{PrName} \quad \sim \quad \text{Employee(s)}(SSN), \text{Project(p)}(PrName), \text{worksFor(p, proj}(PrName)) \]

\[ m₂ : \text{SELECT } SSN, \text{Salary} \quad \sim \quad \text{Employee(s)}(SSN), \text{salary(s)}(SSN, Salary) \]

\[ \text{FROM } D₁ \]

\[ \text{WHERE } D₂.\text{Code} = D₃.\text{Code} \]
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 $O = \langle T, M, D \rangle$, where

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

We need to specify the syntax and semantics of mapping assertions.

**Mapping assertions**

A mapping assertion in $M$ has the form

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

where

- $\Phi$ is an arbitrary SQL query of arity $n > 0$ over $D$;
- $\Psi$ is a conjunctive query over $T$ 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 $f(\vec{z})$, with $f \in \Lambda$ and $\vec{z} \subseteq \vec{x}$.

**Note:** we could consider also mapping assertions between the datatypes of the database and those of the ontology.
Semantics of mappings

To define the semantics of an OBDA system \( O = \langle T, M, D \rangle \), we first need to define the semantics of mappings.

\[ \text{Def.: Satisfaction of a mapping assertion with respect to a database} \]

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

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

Intuitively, \( I \) satisfies \( \Phi \sim \Psi \) w.r.t. \( D \) if all facts obtained by evaluating \( \Phi \) over \( D \) and then propagating the answers to \( \Psi \), hold in \( I \).

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

Semantics of an OBDA system

\[ \text{Def.: Model of an OBDA system} \]

An interpretation \( I \) is a \textit{model} of \( O = \langle T, M, D \rangle \) if:

- \( I \) is a model of \( T \);
- \( I \) satisfies \( M \) w.r.t. \( D \), i.e., \( I \) satisfies every assertion in \( M \) w.r.t. \( D \).

An OBDA system \( O \) is \textit{satisfiable} if it admits at least one model.
Answering queries over an OBDA system

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

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

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 $\Psi$ is constituted by the atoms $X_1, \ldots, X_k$, can be split into several mapping assertions:

$$\Phi \leadsto X_1 \quad \cdots \quad \Phi \leadsto X_k$$

This is possible, since $\Psi$ does not contain non-distinguished variables.

Example

$m_1$: SELECT SSN, PrName FROM D

is split into

$m_1^1$: SELECT SSN, PrName FROM D

$m_1^2$: SELECT SSN, PrName FROM D

$m_1^3$: SELECT SSN, PrName FROM D

$m_1^4$: SELECT SSN, PrName FROM D

Bottom-up approach to query answering

Consists in a straightforward application of the mappings:

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

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

- The technique is no more LOGSPACE in the data, since the ABox $A_{M,D}$ to materialize is in general polynomial in the size of the data.
- $A_{M,D}$ may be very large, and thus it may be infeasible to actually materialize it.
- Freshness of $A_{M,D}$ with respect to the underlying data source(s) may be an issue, and one would need to propagate source updates (cf. Data Warehousing).
The impedance mismatch problem
Ontology-Based Data Access Systems
Query answering in OBDA Systems
Chap. 3: Linking ontologies to relational data

Top-down approach to query answering

Consists of three steps:

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

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

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

D. Calvanese
Part 2: Ontology-Based Access to Inform.

Unfolding

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

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

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

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

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

<table>
<thead>
<tr>
<th>Employee</th>
<th>Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>empCode: Integer</td>
<td>projectName: String</td>
</tr>
<tr>
<td>salary: Integer</td>
<td>1..*</td>
</tr>
<tr>
<td>worksFor</td>
<td>1..*</td>
</tr>
</tbody>
</table>

**Query 1:**

\[
\text{m}_1: \text{SELECT SSN, PrName} \quad \sim \quad \text{Employee(pers(SSN)), projectName(proj(PrName), PrName), worksFor(pers(SSN), proj(PrName))}
\]

**Query 2:**

\[
\text{m}_2: \text{SELECT SSN, Salary} \quad \sim \quad \text{Employee(pers(SSN)), salary(pers(SSN), Salary)}
\]

WHERE \(D_2\).Code = \(D_3\).Code

We define a view \(\text{Aux}_i\) for the source query of each mapping \(m_i\).

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

- \(\text{Employee(pers(SSN))} \leftarrow \text{Aux}_1(\text{SSN, PrName})\)
- \(\text{projectName(proj(PrName), PrName)} \leftarrow \text{Aux}_1(\text{SSN, PrName})\)
- \(\text{Project(proj(PrName))} \leftarrow \text{Aux}_1(\text{SSN, PrName})\)
- \(\text{worksFor(pers(SSN), proj(PrName))} \leftarrow \text{Aux}_1(\text{SSN, PrName})\)
- \(\text{Employee(pers(SSN))} \leftarrow \text{Aux}_2(\text{SSN, Salary})\)
- \(\text{salary(pers(SSN), Salary)} \leftarrow \text{Aux}_2(\text{SSN, Salary})\)

### Unfolding – Example (cont’d)

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

\[
q(e, s) \leftarrow \text{Employee(e), salary(e, s), worksFor(e, p), projectName(p, tones)}
\]

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

\[
\begin{align*}
\vartheta(e) &= \text{pers}(SSN) \\
\vartheta(s) &= \text{Salary} \\
\vartheta(PrName) &= \text{tones} \\
\vartheta(p) &= \text{proj}(tones)
\end{align*}
\]

After applying \(\vartheta\) to \(q\), we obtain:

\[
q(\text{pers}(SSN), \text{Salary}) \leftarrow \text{Employee(pers(SSN)), salary(pers(SSN), Salary), worksFor(pers(SSN), proj(tones)), projectName(proj(tones), tones)}
\]

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

\[
q(\text{pers}(SSN), \text{Salary}) \leftarrow \text{Aux}_1(\text{SSN, tones}), \text{Aux}_2(\text{SSN, Salary}), \text{Aux}_1(\text{SSN, tones}), \text{Aux}_1(\text{SSN, tones})
\]
**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:

\[
\begin{align*}
m_1^i &: \Phi_1^i(x) \leadsto A_i(f(x)) \\
m_2^i &: \Phi_2^i(x) \leadsto A_i(f(x))
\end{align*}
\]

(for \( i \in \{1, \ldots, n\} \))

We add the view definitions: 
\( \text{Aux}_j^i(x) \leftarrow \Phi_j^i(x) \)

and introduce the clauses: 
\( A_i(f(x)) \leftarrow \text{Aux}_j^i(x) \)  
(for \( i \in \{1, \ldots, n\}, j \in \{1, 2\} \)).

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

Hence, we obtain one unfolded query 
\[
q(f(x)) \leftarrow \text{Aux}_1^i(x), \text{Aux}_2^i(x), \ldots, \text{Aux}_n^i(x)
\]

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 \textit{DL-Lite}, we obtain the following result.

**Theorem**

\begin{itemize}
  \item Query answering in a \textit{DL-Lite} OBDM system \( \mathcal{O} = (T, M, D) \) is
    \begin{itemize}
      \item \textbf{NP-complete} in the size of the query.
      \item \textbf{PTime} in the size of the \textbf{TBox} \( T \) and the \textbf{mappings} \( M \).
      \item \textbf{LogSpace} in the size of the \textbf{database} \( D \).
    \end{itemize}
\end{itemize}

**Note:** The \textbf{LogSpace} 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.
Outline

10 TBox reasoning
  ● Preliminaries
  ● Reducing to subsumption
  ● Reducing to ontology unsatisfiability

11 TBox & ABox reasoning

12 Complexity of reasoning in Description Logics

13 Reasoning in $DL$-$L_{Lite}$

14 References
Remark on used notation

In the following,

- We use “TBox” to denote either a $DL-Lite_R$ or a $DL-Lite_F$ TBox.

- $Q$, possibly with subscript, denotes a **basic role**, i.e.,
  $$Q \rightarrow P \mid P^-$$

- $C$, possibly with subscript, denotes a **general concept**, i.e.,
  $$C \rightarrow A \mid \neg A \mid \exists Q \mid \neg \exists Q$$

  where $A$ is an atomic concept and $P$ is an atomic role.

- $R$, possibly with subscript, denotes a **general role**, i.e.,
  $$R \rightarrow Q \mid \neg Q$$

TBox Reasoning services

- **Concept Satisfiability**: $C$ is satisfiable wrt $T$, if there is a model $I$ of $T$ such that $C^I$ is not empty, i.e., $T \not\models C \equiv \bot$

- **Subsumption**: $C_1$ is subsumed by $C_2$ wrt $T$, if for every model $I$ of $T$ we have $C_1^I \subseteq C_2^I$, i.e., $T \models C_1 \subseteq C_2$.

- **Equivalence**: $C_1$ and $C_2$ are equivalent wrt $T$ if for every model $I$ of $T$ we have $C_1^I = C_2^I$, i.e., $T \models C_1 \equiv C_2$.

- **Disjointness**: $C_1$ and $C_2$ are disjoint wrt $T$ if for every model $I$ of $T$ we have $C_1^I \cap C_2^I = \emptyset$, i.e., $T \models C_1 \cap C_2 \equiv \bot$

- **Functionality implication**: A functionality assertion $(funct \ Q)$ is logically implied by $T$ if for every model $I$ of $T$, we have that $(o, o_1) \in Q^I$ and $(o, o_2) \in Q^I$ implies $o_1 = o_2$, i.e., $T \models (funct \ Q)$.

  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 \( O \) is satisfiable, i.e., whether \( O \) admits at least one model.

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

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

### Theorem

- **C** is unsatisfiable wrt \( T \) iff \( T \models C \sqsubseteq \neg C \).
- \( T \models C_1 \equiv C_2 \) iff \( T \models C_1 \sqsubseteq C_2 \) and \( T \models C_2 \sqsubseteq C_1 \).
- \( C_1 \) and \( C_2 \) are disjoint iff \( T \models C_1 \sqsubseteq \neg C_2 \).

### Proof (sketch).

- “\( \Leftarrow \)” if \( T \models C \sqsubseteq \neg \) \( C \), then \( C^I \subseteq \Delta^I \setminus C^I \), for every model \( I = \langle \Delta^I, \cdot^I \rangle \) of \( T \); but this holds iff \( C^I = \emptyset \).
- “\( \Rightarrow \)” if \( C \) is unsatisfiable, then \( C^I = \emptyset \), for every model \( I \) of \( T \). Therefore \( C^I \subseteq (\neg C)^I \).
- Trivial.
- Trivial.

Analogous reductions for role satisfiability, equivalence and disjointness.
From implication of functionalities to subsumption

**Theorem**

\[ T \models (\text{funct } Q) \iff \text{either (funct } Q) \in T \text{ (only for DL-Lite}\_F\text{ ontologies), or } T \models Q \sqsubseteq \neg Q. \]

**Proof (sketch).**

"\( \Rightarrow \)" The case in which \((\text{funct } Q) \in T\) is trivial.

Instead, if \( T \models Q \sqsubseteq \neg Q \), then \( Q^I = \emptyset \) and hence \( I \models (\text{funct } Q) \), for every model \( I \) of \( T \).

"\( \Leftarrow \)" When neither \((\text{funct } Q) \in T\) nor \( T \models Q \sqsubseteq \neg Q \), we can construct a model of \( T \) that is not a model of \((\text{funct } Q)\).

From concept subsumption to ontology unsatisfiability

**Theorem**

\[ T \models C_1 \sqsubseteq C_2 \iff \text{the ontology } O_{C_1 \sqsubseteq C_2} = (T \cup \{ \hat{A} \sqsubseteq C_1, \hat{A} \sqsubseteq \neg C_2 \}, \{ \hat{A}(c) \}) \]

is unsatisfiable, where \( \hat{A} \) is an atomic concept not in \( T \), and \( c \) is a constant.

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

**Proof (sketch).**

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

"\( \Rightarrow \)" Let \( T \models C_1 \sqsubseteq C_2 \), and suppose that \( O_{C_1 \sqsubseteq C_2} \) is satisfiable. Then there exists a model \( I \) of \( O_{C_1 \sqsubseteq C_2} \). Then \( I \models T \), and \( I \models C_1(c) \) and \( I \models \neg C_2(c) \), i.e., \( I \nvdash C_1 \sqsubseteq C_2 \). This contradicts \( T \models C_1 \sqsubseteq C_2 \).
From role subsumption to ont. unsatisfiability for $DL$-Lite$^R$

**Theorem**

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

Intuitively, $R_1$ is subsumed by $R_2$ iff the smallest ontology containing $T$ 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.

Notice that $O_{R_1 \sqsubseteq R_2}$ is inherently a $DL$-Lite$^R$ ontology.

From role subsumption to ont. unsatisfiability for $DL$-Lite$^F$

**Theorem**

Let $T$ be a $DL$-Lite$^F$ TBox, and $Q_1$, $Q_2$ two basic roles such that $Q_1 \neq Q_2$. Then,

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

10 TBox reasoning

11 TBox & ABox reasoning
- TBox & ABox Reasoning services
- Query answering
- Query answering in $DL$-$Lite^R$
- Query answering in $DL$-$Lite^F$
- Ontology satisfiability
- Ontology satisfiability in $DL$-$Lite^R$
- Ontology satisfiability in $DL$-$Lite^F$

12 Complexity of reasoning in Description Logics

13 Reasoning in $DL$-$Lite^A$

14 References
TBox reasoning | TBox & ABox reasoning | Complexity of reasoning in DLs | Reasoning in DL-Lite | References
---|---|---|---|---
| | | | | Chap. 4: Reasoning in the DL-Lite family

## 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 $\mathcal{O} \models q(\vec{c})$.

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 $O = \langle T, A \rangle$ be a **DL-Lite** ontology, $C$ a **DL-Lite** concept, and $P$ an atomic role. Then:

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

**Theorem**

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

**Theorem**

Let $O = \langle T, A \rangle$ be a **DL-Lite**$_R$ ontology and $P$ an atomic role. Then $O \models P(c_1, c_2)$ iff $O_{P(c_1, c_2)} = (T \cup \{\hat{P} \sqsubseteq \neg P\}, A \cup \{\hat{P}(c_1, c_2)\})$ is unsatisfiable, where $\hat{P}$ is an atomic role not in $O$.

---

**Certain answers**

We recall that

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

find all tuples $\vec{c}$ of constants of $A$ such that $O \models q(\vec{c})$

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

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

**Note:** We have assumed that the answer $q^T$ to a query $q$ over an interpretation $I$ is constituted by a set of tuples of **constants** of $A$, rather than objects in $\Delta^I$.
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 \( O \), a query \( q \) over \( O \), and a tuple \( \vec{c} \) of constants, **check whether** \( \vec{c} \in \text{cert}(q, O) \).

We consider a setting where the size of the data largely dominates the size of the conceptual layer, hence, we concentrate on efficiency in the size of the data.

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

Basic questions associated to query answering

1. For which ontology languages can we answer queries over an ontology efficiently?
2. How complex becomes query answering over an ontology when we consider more expressive ontology languages?
To be able to deal with data efficiently, we need to separate the contribution of $\mathcal{A}$ from the contribution of $q$ and $T$.

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

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

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

$\leadsto$ Produces $\text{cert}(q, \langle T, A \rangle)$.

Note: The “always” holds if we pose no restriction on the language in which to express the rewriting $r_{q,T}$.
**Q-rewritability (cont’d)**

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

**Def.: Q-rewritability**

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

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

Hence, Q-rewritability is tightly related to **data complexity**, i.e.:

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

**Q-rewritability: interesting cases**

Consider an ontology language $\mathcal{L}$ that enjoys Q-rewritability, for a query language $Q$:

- When $Q$ is FOL (i.e., the language enjoys **FOL-rewritability**)
  $\rightsquigarrow$ query evaluation can be done in SQL, i.e., via an RDBMS (Note: FOL is in LogSpace).

- When $Q$ is an $\text{NLogSpace}$-hard language
  $\rightsquigarrow$ query evaluation requires (at least) linear recursion.

- When $Q$ is a $\text{PTime}$-hard language
  $\rightsquigarrow$ query evaluation requires (at least) recursion (e.g., Datalog).

- When $Q$ is a $\text{coNP}$-hard language
  $\rightsquigarrow$ query evaluation requires (at least) power of Disjunctive Datalog.
We now study $Q$-rewritability of query answering over $DL$-$Lite$ ontologies.

In particular we will show that both $DL$-$Lite_R$ and $DL$-$Lite_F$ enjoy FOL-rewritability of conjunctive query answering.

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, in the following, we focus on query answering over satisfiable ontologies.

We first consider satisfiable $DL$-$Lite_R$ ontologies.
Remark

We call positive inclusions (PIs) assertions of the form

\[ Cl \sqsubseteq A \mid \exists Q \]
\[ Q_1 \sqsubseteq \exists Q \]

We call negative inclusions (NIs) assertions of the form

\[ Cl \sqsubseteq \neg A \mid \neg \exists Q \]
\[ Q_1 \sqsubseteq \neg \exists Q \]

Query answering in \( DL-Lite_R \)

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

- Using \( T \), reformulate \( q \) as a union \( r_{q,T} \) of CQs.
- Evaluate \( r_{q,T} \) directly over \( A \) managed in secondary storage via a RDBMS.

Correctness of this procedure shows FOL-rewritability of query answering in \( DL-Lite_R \).

\( \neg \) Query answering over \( DL-Lite_R \) ontologies can be done using RDMBS technology.
Query reformulation

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

**Intuition:** Use the PIs as basic rewriting rules:

\[ \text{AssistantProf} \sqsubseteq \text{Professor} \]

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

**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 \( \text{AssistantProf} \sqsubseteq \text{Professor} \) applies to the atom \( \text{Professor}(x) \). Towards the computation of the perfect reformulation, we add to the input query above, the query

\[ q(x) \leftarrow \text{AssistantProf}(x) \]

Consider now the query \( q(x) \leftarrow \text{Professor}(x) \)

and the PI \( \text{Professor} \sqsubseteq \exists \text{teaches} \)

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

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

\[ q(x) \leftarrow \text{Professor}(x) \]
Query reformulation – Constants

Conversely, for the query

\[ q(x) \leftarrow \text{teaches}(x, \text{kbdb}) \]

and the same PI as before

\[ \text{Professor} \sqsubseteq \exists \text{teaches} \]

as a logic rule:

\[ \text{teaches}(z, f(z)) \leftarrow \text{Professor}(z) \]

\text{teaches}(x, \text{kbdb}) \] does not unify with \text{teaches}(z, f(z)), since the skolem term \( f(z) \) in the head of the rule does not unify with the constant \( \text{kbdb} \).

In this case, the PI does not apply to the atom \text{teaches}(x, \text{kbdb})..

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:

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

Query reformulation – 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

\[ q(x) \leftarrow \text{teaches}(x, y), \text{Course}(y) \]

and the PI

\[ \text{Professor} \sqsubseteq \exists \text{teaches} \]

as a logic rule:

\[ \text{teaches}(z, f(z)) \leftarrow \text{Professor}(z) \]

The PI above does not apply to the atom \text{teaches}(x, y).
Consider now the query \( q(x) \leftarrow \text{teaches}(x, y), \text{teaches}(z, y) \)

and the PI

\[
\text{Professor} \sqsubseteq \exists \text{teaches} \quad \text{as a logic rule:} \quad \text{teaches}(z, f(z)) \leftarrow \text{Professor}(z)
\]

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

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

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

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

\[
q(x) \leftarrow \text{Professor}(x)
\]

Reformulate the CQ \( q \) into a set of queries: apply to \( q \) and the computed queries in all possible ways the PIs in the TBox \( T \):

\[
A_1 \sqsubseteq A_2 \quad \ldots, A_2(x), \ldots \leadsto \ldots, A_1(x), \ldots
\]

\[
\exists P \sqsubseteq A \quad \ldots, A(x), \ldots \leadsto \ldots, P(x, z), \ldots
\]

\[
\exists P^- \sqsubseteq A \quad \ldots, A(x), \ldots \leadsto \ldots, P^-(x, z), \ldots
\]

\[
A \sqsubseteq \exists P \quad \ldots, P(x, z), \ldots \leadsto \ldots, A(x), \ldots
\]

\[
A \sqsubseteq \exists P^- \quad \ldots, P^-(x, z), \ldots \leadsto \ldots, A(x), \ldots
\]

\[
\exists P_1 \sqsubseteq \exists P_2 \quad \ldots, P_2(x, z), \ldots \leadsto \ldots, P_1(x, z), \ldots
\]

\[
P_1 \sqsubseteq P_2 \quad \ldots, P_2(x, y), \ldots \leadsto \ldots, P_1(x, y), \ldots
\]

\( \_ \) denotes an \textit{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.

Unifying atoms can make rules applicable that were not so before.

The UCQ resulting from this process is the \textit{perfect reformulation} \( r_{q,T} \).
Query reformulation algorithm

**Algorithm** PerfectRef\( (q, T_P) \)

**Input:** conjunctive query \( q \), set of DL-Lite\(_R\) PIs \( T_P \)

**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 \( T_P \) do

  if \( I \) is applicable to \( g \)

  then \( PR := PR \cup \{ q|g/(g,I) \} \)

for each \( g_1, g_2 \) in \( q \) do

  if \( g_1 \) and \( g_2 \) unify

  then \( PR := PR \cup \{ \tau(\text{reduce}(q, g_1, g_2)) \} \)

until \( PR' = PR \)

return \( PR \)

Notice that NIs do not play any role in the reformulation of the query.

ABox storage

**ABox** \( A \) stored as a **relational database** in a standard RDBMS as follows:

- For each **atomic concept** \( A \) used in the ABox:
  - define a **unary relational table** \( \text{tab}_A \)
  - populate \( \text{tab}_A \) with each \( \langle c \rangle \) such that \( A(c) \in A \)

- For each **atomic role** \( P \) used in the ABox,
  - define a **binary relational table** \( \text{tab}_P \)
  - populate \( \text{tab}_P \) with each \( \langle c_1, c_2 \rangle \) such that \( P(c_1, c_2) \in A \)

We denote with \( \text{DB}(A) \) the database obtained as above.
Query evaluation

Let $r_{q,T}$ be the UCQ returned by the algorithm $\text{PerfectRef}(q, T)$.

- We denote by $\text{SQL}(r_{q,T})$ the encoding of $r_{q,T}$ into an SQL query over $DB(A)$.

- We indicate with $\text{Eval}(\text{SQL}(r_{q,T}), DB(A))$ the evaluation of $\text{SQL}(r_{q,T})$ over $DB(A)$.

Query answering in $\text{DL-Lite}_R$

**Theorem**

Let $T$ be a $\text{DL-Lite}_R$ TBox, $T_P$ the set of PIs in $T$, $q$ a CQ over $T$, and let $r_{q,T} = \text{PerfectRef}(q, T_P)$. Then, for each ABox $\mathcal{A}$ such that $\langle T, \mathcal{A} \rangle$ is satisfiable, we have that $\text{cert}(q, \langle T, \mathcal{A} \rangle) = \text{Eval}(\text{SQL}(r_{q,T}), DB(A))$.

In other words, query answering over a satisfiable $\text{DL-Lite}_R$ ontology is FOL-rewritable.

Notice that we did not mention NIs of $T$ in the theorem above. Indeed, when the ontology is satisfiable, we can ignore NIs and answer queries as if NIs were not specified in $T$. 
Query answering in $DL$-$Lite_R$ – Example

**TBox:**

- Professor $\sqsubseteq \exists$teaches
- $\exists$teaches $\sqsubseteq$ Course

**Query:**

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

**Perfect Reformulation:**

1. $$q(x) \leftarrow \text{teaches}(x, y), \text{Course}(y)$$
2. $$q(x) \leftarrow \text{teaches}(x, y), \text{teaches}(-, y)$$
3. $$q(x) \leftarrow \text{teaches}(x, -)$$
4. $$q(x) \leftarrow \text{Professor}(x)$$

**ABox:**

- teaches(john, kbdb)
- Professor(mary)

It is easy to see that $Eval(SQL(r_q, T), DB(A))$ in this case produces as answer \{john, mary\}.

---

Query answering in $DL$-$Lite_R$ – An interesting example

**TBox:**

- Person $\sqsubseteq \exists$hasFather
- $\exists$hasFather $\sqsubseteq$ Person

**ABox:**

- Person(mary)

**Query:**

$$q(x) \leftarrow \text{Person}(x), \text{hasFather}(x, y_1), \text{hasFather}(y_1, y_2), \text{hasFather}(y_2, y_3)$$

1. $$q(x) \leftarrow \text{Person}(x), \text{hasFather}(x, y_1), \text{hasFather}(y_1, y_2), \text{hasFather}(y_2, -)$$
   \[ \Downarrow \text{Apply Person $\sqsubseteq \exists$hasFather to the atom hasFather}(y_2, -) \]
2. $$q(x) \leftarrow \text{Person}(x), \text{hasFather}(x, y_1), \text{hasFather}(y_1, y_2), \text{Person}(y_2)$$
   \[ \Downarrow \text{Apply $\exists$hasFather $\sqsubseteq$ Person to the atom Person}(y_2) \]
3. $$q(x) \leftarrow \text{Person}(x), \text{hasFather}(x, y_1), \text{hasFather}(y_1, y_2), \text{hasFather}(-, y_2)$$
   \[ \Downarrow \text{Unify atoms hasFather}(y_1, y_2) \text{ and hasFather}(-, y_2) \]
4. $$q(x) \leftarrow \text{Person}(x), \text{hasFather}(x, y_1), \text{hasFather}(y_1, y_2)$$
   \[ \Downarrow \ldots \]
5. $$q(x) \leftarrow \text{Person}(x), \text{hasFather}(x, -)$$
   \[ \Downarrow \text{Apply Person $\sqsubseteq \exists$hasFather to the atom hasFather}(x, -) \]
6. $$q(x) \leftarrow \text{Person}(x)$$
Query answering in $\text{DL-Lite}_F$

If we limit our attention to PIs, we can say that $\text{DL-Lite}_F$ ontologies are $\text{DL-Lite}_R$ ontologies of a special kind (i.e., with no PIs between roles).

As for NIs and functionality assertions, it is possible to prove that they can be disregarded in query answering over satisfiable $\text{DL-Lite}_F$ ontologies.

From this the following result follows immediately.

**Theorem**

Let $T$ be a $\text{DL-Lite}_F$ TBox, $T_P$ the set of PIs in $T$, $q$ a CQ over $T$, and let $r_q,T = \text{PerfectRef}(q, T_P)$. Then, for each ABox $\mathcal{A}$ such that $\langle T, \mathcal{A} \rangle$ is satisfiable, we have that $\text{cert}(q, \langle T, \mathcal{A} \rangle) = \text{Eval}(\text{SQL}(r_q,T), \text{DB}(\mathcal{A}))$.

In other words, query answering over a satisfiable $\text{DL-Lite}_F$ ontology is FOL-rewritable.

Satisfiability of ontologies with only PIs

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

Remember that solving this problem allow us to solve TBox reasoning and identify cases in which query answering is meaningless.

A first notable result tells us that PIs alone cannot cause an ontology to become unsatisfiable.

**Theorem**

Let $\mathcal{O} = \langle T, \mathcal{A} \rangle$ be either a $\text{DL-Lite}_R$ or a $\text{DL-Lite}_F$ ontology, where $T$ contains only PIs. Then, $\mathcal{O}$ is satisfiable.
NIs, however, can make a $DL-Lite_R$ ontology unsatisfiable.

**Example**

**TBox $\mathcal{T}$:** Professor $\sqsubseteq$ ¬Student  
$\exists$teaches $\sqsubseteq$ Professor

**ABox $\mathcal{A}$:** teaches(john,kbdb) 
Student(john)

In what follows we provide a mechanism to establish, in an efficient way, whether a $DL-Lite_R$ ontology is satisfiable.

### Checking satisfiability of $DL-Lite_R$ ontologies

Satisfiability of a $DL-Lite_R$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is reduced to evaluating a FOL-query (in fact a UCQ) over $DB(\mathcal{A})$.

**We proceed as follows:** Let $\mathcal{T}_P$ the set of PIs in $\mathcal{T}$.

1. For each NI $N$ between concepts (resp. roles) in $\mathcal{T}$, we ask $\langle \mathcal{T}_P, \mathcal{A} \rangle$ whether there exists some individual (resp. pair of individuals) that contradicts $N$, i.e., we construct over $\langle \mathcal{T}_P, \mathcal{A} \rangle$ a boolean CQ $q_N()$ such that
   $$\langle \mathcal{T}_P, \mathcal{A} \rangle \models q_N() \iff \langle \mathcal{T}_P \cup \{N\}, \mathcal{A} \rangle \text{ is unsatisfiable}$$

2. We exploit $PerfectRef$ to verify whether $\langle \mathcal{T}_P, \mathcal{A} \rangle \models q_N()$, i.e., we compute $PerfectRef(q_N, \mathcal{T}_P)$, and evaluate it (in fact, its SQL encoding) over $DB(\mathcal{A})$. 

D. Calvanese
Part 2: Ontology-Based Access to Inform.
KBDB – 2008/2009 (177/220)
Satisfiability of $\mathit{DL-Lite}_R$ ontologies – Example

Pls $T_P$: $\exists$teaches $\sqsubseteq$ Professor

NI $N$: Professor $\sqsubseteq \neg$Student

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

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

ABox $A$: teaches(john, kbdb)
Student(john)

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

Queries for NI$s$

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

$$
\begin{align*}
A_1 & \sqsubseteq \neg A_2 & \sim & q_N() \leftarrow A_1(x), A_2(x) \\
\exists P & \sqsubseteq \neg A & \text{or} & A \sqsubseteq \neg \exists P & \sim & q_N() \leftarrow P(x,y), A(x) \\
\exists P^- & \sqsubseteq \neg A & \text{or} & A \sqsubseteq \neg \exists P^- & \sim & q_N() \leftarrow P(y,x), A(x) \\
\exists P_1 & \sqsubseteq \neg \exists P_2 & \sim & q_N() \leftarrow P_1(x,y), P_2(x,z) \\
\exists P_1^- & \sqsubseteq \neg \exists P^-_2 & \sim & q_N() \leftarrow P_1(x,y), P^-_2(z,x) \\
\exists P_2 & \sqsubseteq \neg \exists P_2^- & \sim & q_N() \leftarrow P_2(y,x), P_2(y,z) \\
\exists P^-_1 & \sqsubseteq \neg \exists P^-_2 & \sim & q_N() \leftarrow P^-_1(x,y), P_2(z,y) \\
P_1 & \sqsubseteq \neg P_2 & \text{or} & P^-_1 \sqsubseteq \neg P^-_2 & \sim & q_N() \leftarrow P_1(x,y), P_2(x,y) \\
P^-_1 & \sqsubseteq \neg P_2 & \text{or} & P_1 \sqsubseteq \neg P^-_2 & \sim & q_N() \leftarrow P^-_1(x,y), P_2(y,x)
\end{align*}
$$
**Lemma (Separation for DL-Lite\(_R\))**

Let \( O = \langle T, A \rangle \) be a DL-Lite\(_R\) ontology, and \( T_P \) the set of PIs in \( T \). Then, \( O \) is unsatisfiable iff there exists a NI \( N \in T \) such that \( \langle T_P, A \rangle \models q_N() \).

The lemma relies on the following properties:
- NIs do not interact with each other.
- Interaction between NIs and PIs can be managed through \( \text{PerfectRef} \).

Notably, each NI can be processed individually.

**DL-Lite\(_R\): FOL-rewritability of satisfiability**

From the previous lemma and the theorem on query answering for satisfiable DL-Lite\(_R\) ontologies, we get the following result.

**Theorem**

Let \( O = \langle T, A \rangle \) be a DL-Lite\(_R\) ontology, and \( T_P \) the set of PIs in \( T \). Then, \( O \) is unsatisfiable iff there exists a NI \( N \in T \) such that \( \text{Eval}(\text{SQL}(\text{PerfectRef}(q_N, T_P)), \text{DB}(A)) \) returns true.

In other words, satisfiability of a DL-Lite\(_R\) ontology can be reduced to FOL-query evaluation.
Unsatisfiability in $DL$-$\mathcal{F}$ ontologies can be caused by NIs or by functionality assertions.

Example

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

ABox $\mathcal{A}$:
- Student(john)
- teaches(john, kdb)
- teaches(michael, kdb)

In what follows we extend to $DL$-$\mathcal{F}$ ontologies the technique for $DL$-$\mathcal{R}$ ontology satisfiability given before.

Satisfiability of a $DL$-$\mathcal{F}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is reduced to evaluating a FOL-query over $DB(\mathcal{A})$.

We deal with NIs exactly as done in $DL$-$\mathcal{R}$ ontologies (indeed, limited to NIs, $DL$-$\mathcal{F}$ ontologies are $DL$-$\mathcal{R}$ ontologies of a special kind).

To deal with functionality assertions, we proceed as follows:

- For each functionality assertion $F \in \mathcal{T}$, we ask if there exist two pairs of individuals in $\mathcal{A}$ that contradict $F$, i.e., we pose over $\mathcal{A}$ a boolean FOL query $q_F()$ such that
  \[ \mathcal{A} \models q_F() \iff \langle \{F\}, \mathcal{A} \rangle \text{ is unsatisfiable.} \]
- To verify if $\mathcal{A} \models q_F()$, we evaluate $SQL(q_F)$ over $DB(\mathcal{A})$. 
Queries for functionality assertions

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

\[
\begin{align*}
\text{funct } P & \quad \mapsto \quad q_F() \leftarrow P(x, y), P(x, z), y \neq z \\
\text{funct } P^- & \quad \mapsto \quad q_F() \leftarrow P(x, y), P(z, y), x \neq z
\end{align*}
\]

**Example**

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

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

ABox $A$: teaches(john, kbdb)

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

**Lemma (Separation for DL-Lite$_F$)**

Let $O = \langle T, A \rangle$ be a DL-Lite$_F$ ontology, and $T_P$ the set of PIs in $T$. Then, $O$ is unsatisfiable iff one of the following condition holds:

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

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

(a) relies on the properties that NIs do not interact with each other, and interaction between NIs and PIs can be managed through PerfectRef.

(b) exploits the property that NIs and PIs do not interact with functionalities: indeed, no functionality assertions are contradicted in a DL-Lite$_F$ ontology $O$, beyond those explicitly contradicted by the ABox.

Notably, the lemma asserts that to check ontology satisfiability, each NI and each functionality assertion can be processed individually.
From the previous lemma and the theorem on query answering for satisfiable $DL$-$Lite_\mathcal{F}$ ontologies, we get the following result.

**Theorem**

Let $\mathcal{O} = \langle T, A \rangle$ be a $DL$-$Lite_\mathcal{F}$ ontology, and $T_P$ the set of PIs in $T$. Then, $\mathcal{O}$ is unsatisfiable iff one of the following conditions holds:

(a) There exists a NI $N \in T$ such that $\text{Eval}(\text{SQL}(\text{PerfectRef}(q_N, T_P)), \text{DB}(A))$ returns true.

(b) There exists a functionality assertion $F \in T$ such that $\text{Eval}(\text{SQL}(q_F), \text{DB}(A))$ returns true.

In other words, satisfiability of a $DL$-$Lite_\mathcal{F}$ ontology can be reduced to FOL-query evaluation.

**Outline**

1. TBox reasoning
2. TBox & ABox reasoning
3. Complexity of reasoning in Description Logics
   - Complexity of reasoning in $DL$-$Lite$
   - Data complexity of query answering in DLs beyond $DL$-$Lite$
   - $\text{NLogSpace}$-hard DLs
   - $\text{PTime}$-hard DLs
   - $\text{coNP}$-hard DLs
4. Reasoning in $DL$-$Lite_\mathcal{A}$
5. References
Complexity of query answering over satisfiable ontologies

Theorem

Query answering over both $DL-Lite_R$ and $DL-Lite_F$ ontologies is

1. **NP-complete** in the size of query and ontology (combined comp.).
2. **PTime** in the size of the ontology.
3. **LogSpace** in the size of the ABox (data complexity).

Proof (sketch).

1. **Guess** the derivation of one of the CQs of the perfect reformulation, 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 reformulation is polynomial in the size of the TBox, and we can compute them in PTIME.
3. Is the data complexity of evaluating FOL queries over a DB.

Complexity of ontology satisfiability

Theorem

Checking satisfiability of both $DL-Lite_R$ and $DL-Lite_F$ ontologies is

1. **PTime** in the size of the ontology (combined complexity).
2. **LogSpace** in the size of the ABox (data complexity).

Proof (sketch).

Follows directly from the algorithm for ontology satisfiability and the complexity of query answering over satisfiable ontologies.
Complexity of TBox reasoning

**Theorem**

TBox reasoning over both DL-Lite\(_R\) and DL-Lite\(_F\) ontologies is PTime in the size of the TBox (schema complexity).

**Proof (sketch).**

Follows from the previous theorem, and from the reduction of TBox reasoning to ontology satisfiability. Indeed, the size of the ontology constructed in the reduction is polynomial in the size of the input TBox.

Beyond DL-Lite

Can we further extend these results to more expressive ontology languages?

Essentially NO! (unless we take particular care)
Beyond DL-Lite

We now consider DL languages that allow for constructs not present in DL-Lite or for combinations of constructs that are not legal in DL-Lite.

We recall here syntax and semantics of constructs used in what follows.

<table>
<thead>
<tr>
<th>Construct</th>
<th>Syntax</th>
<th>Example</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>conjunction</td>
<td>$C_1 \sqcap C_2$</td>
<td>Doctor $\cap$ Male</td>
<td>$C_1^T \sqcap C_2^T$</td>
</tr>
<tr>
<td>disjunction</td>
<td>$C_1 \sqcup C_2$</td>
<td>Doctor $\sqcup$ Lawyer</td>
<td>$C_1^T \sqcup C_2^T$</td>
</tr>
<tr>
<td>qual. exist. restr.</td>
<td>$\exists Q.C$</td>
<td>$\exists$child.Male</td>
<td>${a \mid \exists b. (a,b) \in Q^T \land b \in C^T }$</td>
</tr>
<tr>
<td>qual. univ. restr.</td>
<td>$\forall Q.C$</td>
<td>$\forall$child.Male</td>
<td>${a \mid \forall b. (a,b) \in Q^T \rightarrow b \in C^T }$</td>
</tr>
</tbody>
</table>

Summary of results on data complexity

<table>
<thead>
<tr>
<th>Cl</th>
<th>$DL$-$\mathcal{L}_F$</th>
<th>$DL$-$\mathcal{L}_R$</th>
<th>$\mathcal{F}$</th>
<th>$\mathcal{R}$</th>
<th>Data complexity of query answering</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$DL$-$\mathcal{L}_F$</td>
<td>$\sqrt{}$</td>
<td>$-$</td>
<td>$-$</td>
<td>in LogSpace</td>
</tr>
<tr>
<td>2</td>
<td>$DL$-$\mathcal{L}_R$</td>
<td>$-$</td>
<td>$\sqrt{}$</td>
<td>$-$</td>
<td>in LogSpace</td>
</tr>
<tr>
<td>3</td>
<td>$A \mid \exists P.A$</td>
<td>$A$</td>
<td>$-$</td>
<td>$-$</td>
<td>NLogSpace-hard</td>
</tr>
<tr>
<td>4</td>
<td>$A \mid \forall P.A$</td>
<td>$A \mid \forall P.A$</td>
<td>$-$</td>
<td>$-$</td>
<td>NLogSpace-hard</td>
</tr>
<tr>
<td>5</td>
<td>$A \mid \exists P.A$</td>
<td>$A \mid \exists P.A$</td>
<td>$\sqrt{}$</td>
<td>$-$</td>
<td>NLogSpace-hard</td>
</tr>
<tr>
<td>6</td>
<td>$A \mid \exists P.A \mid A_1 \sqcap A_2$</td>
<td>$A$</td>
<td>$-$</td>
<td>$-$</td>
<td>PTIME-hard</td>
</tr>
<tr>
<td>7</td>
<td>$A \mid A_1 \sqcap A_2$</td>
<td>$A \mid A_1 \sqcap A_2$</td>
<td>$\sqrt{}$</td>
<td>$-$</td>
<td>PTIME-hard</td>
</tr>
<tr>
<td>8</td>
<td>$A \mid A_1 \sqcap A_2$</td>
<td>$A \mid A_1 \sqcap A_2$</td>
<td>$\sqrt{}$</td>
<td>$-$</td>
<td>PTIME-hard</td>
</tr>
<tr>
<td>9</td>
<td>$A \mid \exists P.A \mid \exists P^- A$</td>
<td>$A \mid \exists P^-$</td>
<td>$-$</td>
<td>$-$</td>
<td>PTIME-hard</td>
</tr>
<tr>
<td>10</td>
<td>$A \mid \exists P^- A$</td>
<td>$A \mid \exists P^- A$</td>
<td>$\sqrt{}$</td>
<td>$\sqrt{}$</td>
<td>PTIME-hard</td>
</tr>
<tr>
<td>11</td>
<td>$A \mid \exists P.A$</td>
<td>$A \mid \exists P.A$</td>
<td>$\sqrt{}$</td>
<td>$-$</td>
<td>PTIME-hard</td>
</tr>
<tr>
<td>12</td>
<td>$A \mid \neg A$</td>
<td>$A$</td>
<td>$-$</td>
<td>$-$</td>
<td>coNP-hard</td>
</tr>
<tr>
<td>13</td>
<td>$A$</td>
<td>$A \mid A_1 \sqcup A_2$</td>
<td>$-$</td>
<td>$-$</td>
<td>coNP-hard</td>
</tr>
<tr>
<td>14</td>
<td>$A \mid \forall P.A$</td>
<td>$A$</td>
<td>$-$</td>
<td>$-$</td>
<td>coNP-hard</td>
</tr>
</tbody>
</table>

All NLogSpace and PTIME hardness results hold already for atomic queries.
Observations

- **DL-Lite-family** is FOL-rewritable, hence **LogSpace** – holds also with *n*-ary relations \( \sim \) **DLR-Lite**\(_F\) and **DLR-Lite**\(_R\).
- **RDFS** is a subset of **DL-Lite**\(_R\) \( \sim \) is FOL-rewritable, hence **LogSpace**.
- **Horn-SHIQ** [HMS05] is **PTime-hard** even for instance checking (line 11).
- **DLP** [GHVD03] is **PTime-hard** (line 6)
- **EL** [BBL05] is **PTime-hard** (line 6).

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**:

<table>
<thead>
<tr>
<th>( Cl )</th>
<th>( Cr )</th>
<th>( F )</th>
<th>( R )</th>
<th>Data complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>( A \mid \exists P.A )</td>
<td>( A )</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

Hardness proof is by a reduction from reachability in directed graphs:
- TBox \( T \): a single inclusion assertion \( \exists P.A \sqsubseteq A \)
- ABox \( A \): encodes graph using \( P \) and asserts \( A(d) \)

Result:
\( \langle T , A \rangle \models A(s) \) iff \( d \) is reachable from \( s \) in the graph.
**NLogSpace-hard cases**

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

<table>
<thead>
<tr>
<th>Cl</th>
<th>Cr</th>
<th>F</th>
<th>R</th>
<th>Data complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \mid \exists P.A$</td>
<td>$A$</td>
<td>$-</td>
<td>$</td>
<td>$-</td>
</tr>
</tbody>
</table>

By reduction from reachability in directed graphs

| A  | $A \mid \forall P.A$ | $-|$ | $-|$ | NLogSpace-hard |
|----|----------------------|----|----|-----------------|

Follows from 3 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 $P'$.

| A  | $A \mid \exists P.A$ | $\sqrt{\ }$ | $-|$ | NLogSpace-hard |
|----|---------------------|----|----|-----------------|

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

**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 $PS$, checking whether $t$ is accessible, is PTIME-complete.
Reduction from Path System Accessibility

Given an instance $PS = (N, E, S, t)$, we construct

- TBox $T$ consisting of the inclusion assertions
  \[
  \exists P_1.A \sqsubseteq B_1 \\
  \exists P_2.A \sqsubseteq B_2 \\
  B_1 \sqcap B_2 \sqsubseteq A \\
  \exists P_3.A \sqsubseteq A
  \]

- ABox $A$ encoding the accessibility relation using $P_1$, $P_2$, and $P_3$, and asserting $A(s)$ for each source node $s \in S$

\[
\begin{align*}
  e_1 &= (n, \ldots) \\
  e_2 &= (n, s_1, s_2) \\
  e_3 &= (n, \ldots)
\end{align*}
\]

Result:
\[
\langle T, A \rangle \models A(t) \text{ iff } t \text{ is accessible in } PS.
\]

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:

<table>
<thead>
<tr>
<th></th>
<th>$CI$</th>
<th>$CR$</th>
<th>$\mathcal{F}$</th>
<th>$\mathcal{R}$</th>
<th>Data complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>$A$</td>
<td>$\neg A$</td>
<td>$A$</td>
<td>$\neg$</td>
<td>coNP-hard</td>
</tr>
<tr>
<td>12</td>
<td>$A$</td>
<td>$A$</td>
<td>$A_1 \sqcap A_2$</td>
<td>$\neg$</td>
<td>coNP-hard</td>
</tr>
<tr>
<td>13</td>
<td>$A$</td>
<td>$\forall P.A$</td>
<td>$A$</td>
<td>$\neg$</td>
<td>coNP-hard</td>
</tr>
</tbody>
</table>

All three cases are proved by adapting the proof of coNP-hardness of instance checking for $\mathcal{ALC}$ by [DLNS94].
2+2-SAT: satisfiability of a 2+2-CNF formula, i.e., a CNF formula where each clause has exactly 2 positive and 2 negative literals.

Example: $\varphi = c_1 \land c_2 \land c_3$, with
\[ c_1 = v_1 \lor v_2 \lor \neg v_3 \lor \neg v_4 \]
\[ c_2 = \text{false} \lor \text{false} \lor \neg v_1 \lor \neg v_4 \]
\[ c_3 = \text{false} \lor v_4 \lor \text{true} \lor \neg v_2 \]

2+2-SAT is NP-complete \[DLNS94\].

Reduction from 2+2-SAT

2+2-CNF formula $\varphi = c_1 \land \cdots \land c_k$ over variables $v_1, \ldots, v_n$, $\text{true}$, $\text{false}$

- Ontology is over concepts $L$, $T$, $F$, roles $P_1$, $P_2$, $N_1$, $N_2$ and individuals $v_1, \ldots, v_n$, true, false, $c_1, \ldots, c_k$
- ABox $A_\varphi$ constructed from $\varphi$:
  - for each propositional variable $v_i$: $L(v_i)$
  - for each clause $c_j = v_{j1} \lor v_{j2} \lor \neg v_{j3} \lor \neg v_{j4}$:
    - $P_1(c_j, v_{j1})$, $P_2(c_j, v_{j2})$, $N_1(c_j, v_{j3})$, $N_2(c_j, v_{j4})$
    - $T(\text{true})$, $F(\text{false})$
- TBox $T = \{L \sqsubseteq T \sqcup F\}$
- $q() \leftarrow P_1(c, v_1), P_2(c, v_2), N_1(c, v_3), N_2(c, v_4), F(v_1), F(v_2), T(v_3), T(v_4)$

Note: the TBox $T$ and the query $q$ do not depend on $\varphi$, hence this reduction works for data complexity.
Reduction from 2+2-SAT (cont’d)

Lemma

\( \langle T, A_\varphi \rangle \not \models q() \) iff \( \varphi \) is satisfiable.

Proof (sketch).

"\( \Rightarrow \)" If \( \langle T, A_\varphi \rangle \not \models q() \), then there is a model \( I \) of \( \langle T, A_\varphi \rangle \) s.t. \( I \not \models q() \). We define a truth assignment \( \alpha_I \) by setting \( \alpha_I(v_i) = true \) iff \( v_i^T \in T^I \). Notice that, since \( L \subseteq T \sqcup F \), if \( v_i^T \notin T^I \), then \( v_i^T \in F^I \).

It is easy to see that, since \( q() \) asks for a false clause and \( I \not \models q() \), for each clause \( c_j \), one of the literals in \( c_j \) evaluates to \( true \) in \( \alpha_I \).

"\( \Leftarrow \)" From a truth assignment \( \alpha \) that satisfies \( \varphi \), we construct an interpretation \( I_\alpha \) with \( \Delta_\alpha = \{ c_1, \ldots, c_k, v_1, \ldots, v_n, t, f \} \), and:

- \( c_j^\alpha = c_j \), \( v_i^\alpha = v_i \), \( true^\alpha = t \), \( false^\alpha = f \)
- \( T_\alpha = \{ v_i \mid \alpha(v_i) = true \} \cup \{ t \} \), \( F_\alpha = \{ v_i \mid \alpha(v_i) = false \} \cup \{ f \} \)

It is easy to see that \( I_\alpha \) is a model of \( \langle T, A_\varphi \rangle \) and that \( I_\alpha \not \models q() \).
Combining functionalities and role inclusions

We have seen till now that:

- By including in DL-Lite both functionality of roles and qualified existential quantification (i.e., \(\exists P.A\)), query answering becomes NLogSpace-hard (and PTime-hard with also inverse roles) in data complexity (see Part 3).
- Qualified existential quantification can be simulated by using role inclusion assertions (see Part 2).
- 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, we need to restrict the interaction of functionality and role inclusions.

Let us analyze on an example the effect of an unrestricted interaction.

Combining functionalities and role inclusions – Example

\[
\text{TBox } T: \quad A \sqsubseteq \exists P \quad P \sqsubseteq S \\
\exists P^- \sqsubseteq A \quad \text{(funct } S) \\
\text{ABox } A: \quad A(c_1), \quad S(c_1,c_2), \quad S(c_2,c_3), \ldots, \quad S(c_{n-1},c_n) \\
\]

\[
\begin{align*}
A(c_1), \quad A \sqsubseteq \exists P & \quad \models \quad P(c_1,x), \quad \text{ for some } x \\
P(c_1,x), \quad P \sqsubseteq S & \quad \models \quad S(c_1,x) \\
S(c_1,x), \quad S(c_1,c_2), \quad \text{(funct } S) & \quad \models \quad x = c_2 \\
P(c_1,c_2), \quad \exists P^- \sqsubseteq A & \quad \models \quad A(c_2) \\
A(c_2), \quad A \sqsubseteq \exists P & \quad \models \quad A(c_n) \\
\end{align*}
\]

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 \(\langle T, A \rangle\) is true:

\[
q() \leftarrow A(z_1), S(z_1, z_2), S(z_2, z_3), \ldots, S(z_{n-1}, z_n), A(z_n)
\]
Interaction between functionalities and role inclusions

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

Since we do not want to limit cycles of ISA, we pose suitable restrictions on the combination of functionality and role inclusions.

Complexity of DL-Lite with funct. and role inclusions

Let $DL-Lite_{FR}$ be the DL that is the union of $DL-Lite_F$ and $DL-Lite_R$, i.e., the DL-Lite logic that allows for using both role functionality and role inclusions without any restrictions.

**Theorem** [ACKZ09]

For $DL-Lite_{FR}$ 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).
Restriction on TBox assertions in $DL\text{-}Lite_A$ ontologies

To ensure FOL-rewritability, in $DL\text{-}Lite_A$ we have imposed a restriction on the use of functionality and role/attribute inclusions.

**Restriction on $DL\text{-}Lite_A$ TBoxes**

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

Since qualified existentials in the right-hand side of concept inclusions are encoded using role inclusions, this restriction affects also qualified existentials.

Formally:

- If $\exists P.C$ or $\exists P^-.C$ appears in $T$, then $(\text{funct } P)$ and $(\text{funct } P^-)$ are not in $T$.
- If $Q \sqsubseteq P$ or $Q \sqsubseteq P^-$ is in $T$, then $(\text{funct } P)$ and $(\text{funct } P^-)$ are not in $T$.
- If $U_1 \sqsubseteq U_2$ is in $T$, then $(\text{funct } U_2)$ is not in $T$.

Reasoning in $DL\text{-}Lite_A$ – Separation

It is possible to show that, by virtue of the restriction on the use of role inclusion and functionality assertions, all nice properties of $DL\text{-}Lite_F$ and $DL\text{-}Lite_R$ continue to hold also for $DL\text{-}Lite_A$.

In particular, w.r.t. satisfiability of a $DL\text{-}Lite_A$ ontology $O$, we have:

- NIs do not interact with each other.
- NIs and PIs do not interact with functionality assertions.

We obtain that for $DL\text{-}Lite_A$ a separation result holds:

- Each NI and each functionality can be checked independently from the others.
- A functionality assertion is contradicted in an ontology $O = \langle T, A \rangle$ only if it is explicitly contradicted by its ABox $A$. 

Ontology satisfiability in $DL{-}Lite_\mathcal{A}$

Due to the separation property, we can associate
- to each NI $N$ a boolean CQ $q_N()$, and
- to each functionality assertion $F$ a boolean FOL query $q_F()$
and check satisfiability of $O$ by suitably evaluating $q_N()$ and $q_F()$.

**Theorem**

Let $O = \langle T, A \rangle$ be a $DL{-}Lite_\mathcal{A}$ ontology, and $T_P$ the set of PIs in $O$. Then, $O$ is unsatisfiable iff one of the following condition holds:

- There exists a NI $N \in T$ such that $\text{Eval}(\text{SQL}(\text{PerfectRef}(q_N, T_P)), DB(A))$ returns true.
- There exists a functionality assertion $F \in T$ such that $\text{Eval}(\text{SQL}(q_F), DB(A))$ returns true.

Query answering in $DL{-}Lite_\mathcal{A}$

- Queries over $DL{-}Lite_\mathcal{A}$ ontologies are analogous to those over $DL{-}Lite_R$ and $DL{-}Lite_F$ ontologies, except that they can also make use of attribute and domain atoms.
- Exploiting the previous result, the query answering algorithm of $DL{-}Lite_R$ can be easily extended to deal with $DL{-}Lite_\mathcal{A}$ ontologies:
  - Assertions involving attribute domain and range can be dealt with as for role domain and range assertions.
  - $\exists Q.C$ in the right hand-side of concept inclusion assertions can be eliminated by making use of role inclusion assertions.
  - Disjointness of roles and attributes can be checked similarly as for disjointness of concepts, and does not interact further with the other assertions.
Complexity of reasoning in $\text{DL-Lite}_A$

As for ontology satisfiability, $\text{DL-Lite}_A$ maintains the nice computational properties of $\text{DL-Lite}_R$ and $\text{DL-Lite}_F$ also w.r.t. query answering. Hence, we get the same characterization of computational complexity.

**Theorem** [PLC$^+$08, ACKZ09]

For $\text{DL-Lite}_A$ ontologies:

- Checking **satisfiability of the ontology** is
  - $\text{NLogSpace-complete}$ in the size of the **ontology** (combined complexity).
  - $\text{LogSpace}$ in the size of the **ABox** (data complexity).
- **TBox reasoning** is $\text{NLogSpace-complete}$ in the size of the **TBox**.
- **Query answering** is
  - $\text{NP-complete}$ in the size of the query and the ontology (comb. com.).
  - $\text{NLogSpace-complete}$ in the size of the **ontology**.
  - $\text{LogSpace}$ in the size of the **ABox** (data complexity).
References I


References II


References III

Linking data to ontologies: The description logic DL-Lite_A.

Data complexity of query answering in description logics.

Tractable reasoning and efficient query answering in description logics: The DL-Lite family.

References IV

[CK06] A. Calì and M. 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.

The complexity of concept languages.

Deduction in concept languages: From subsumption to instance checking.

[Don03] F. M. Donini.
Complexity of reasoning.
In Baader et al. [BCM+03], chapter 3, pages 96–136.
References V

Conjunctive query answering for the description logic $SHIQ$.

Description logic programs: Combining logic programs with description logic.

Data complexity of reasoning in very expressive description logics.

Combining Horn rules and description logics in CARIN.

References VI

Inverse roles make conjunctive queries hard.

Description logic systems.
In Baader et al. [BCM+03], chapter 8, pages 282–305.

Characterizing data complexity for conjunctive query answering in expressive description logics.

Linking data to ontologies.
[Sch93] A. Schaerf.
On the complexity of the instance checking problem in concept languages with existential quantification.