Ontology-Based Data Access

ISWC 2007 Tutorial

Diego Calvanese[1], Domenico Lembo[2]

[1] Free University of Bozen-Bolzano,

[2] Sapienza Università di Roma

6th Int. Semantic Web Conference
Busan, South Korea – Nov. 12th, 2007

Structure of the tutorial

1. Introduction to Ontology-Based Data Access
    1. Introduction to ontologies
    2. Ontology languages
2. Description Logics and the *DL-Lite* family
    1. A gentle introduction to DLs
    2. DLs as a formal language to specify ontologies
    3. Queries in Description Logics
    4. The *DL-Lite* family of tractable DLs
3. Reasoning in the *DL-Lite* family
    1. TBox reasoning
    2. TBox & ABox reasoning
    3. Complexity of reasoning in Description Logics
4. Linking data to ontologies
    1. The Description Logic $\textit{DL-Lite}_{\mathcal{A}}$
    2. Connecting ontologies to relational data
5. Hands-on session

Part I

Introduction to Ontology-Based Data Access

Outline

Outline

Different meanings of "Semantics"

1. Part of linguistics that studies the meaning of words and phrases.
2. 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.
3. 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.

Ontologies

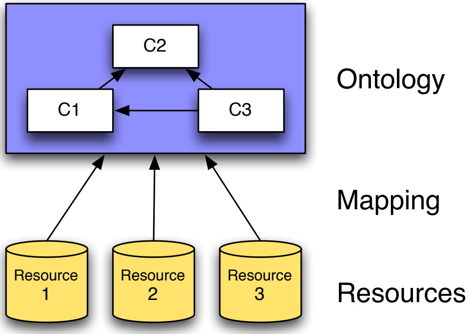
Definition

An 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.

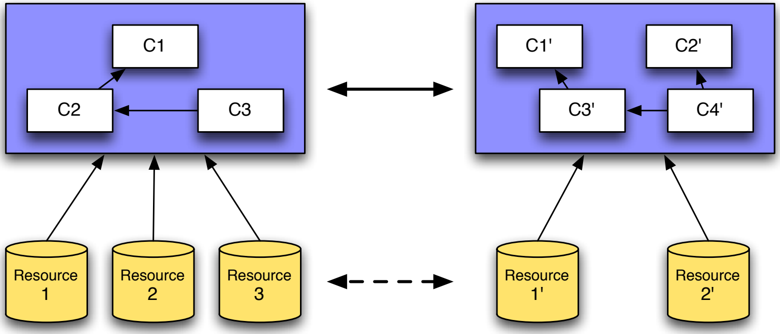
Ontology mediated data access

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.

Similar to Data Integration, but with a rich conceptual description as the global view.

Ontologies at the core of cooperation



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

Three novel challenges

1. Languages
2. Methodologies
3. Tools

. . . for specifying, building, and managing ontologies to be used in information systems.

The first challenge: 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 tutorial:

- We propose variants of ontology languages suited for managing ontologies in information systems.
- We discuss in depth the above mentioned tradeoff . . .
- . . . paying particular attention to the aspects related to data management.

The second challenge: methodologies

- Building and dealing with ontologies is a complex and challenging task.
- Building 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 tutorial:

- 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 present specific methodologies here.

The third challenge: 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 tutorial:

- We discuss the requirements for such ontology management tools.
- We present 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 tutorial 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 tutorial we concentrate essentially on the static aspects.

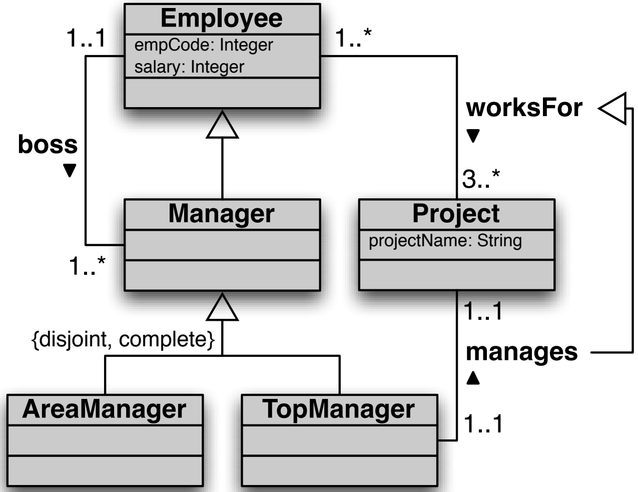
Intensional level of an ontology language

An ontology language for expressing the intensional level usually includes:

- Concepts
- Properties of concepts
- Relationships between concepts, and their properties
- Axioms
- Individuals and facts about individuals
- Queries

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

Example: ontology rendered as UML Class Diagram



Concepts

Concept

Is an element of the 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

Property

Qualifies an element (e.g., a concept) of an ontology.

Property definition (intensional and extensional):

- Name
- Type:
  - Atomic (integer, real, string, enumerated, ...)
    e.g., eye-color → { blu, brown, green, grey }
  - Structured (date, sets, lists, ...)
    e.g., date → day/month/year
- Default value

Properties are also called attributes, features, slots.

Relationships

Relationship

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

Axiom

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

Different kinds of axioms/conditions:

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

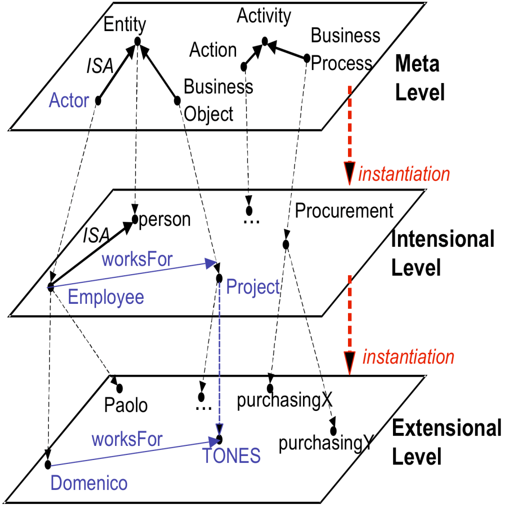
Axioms are also called assertions.
A special kind of axioms are definitions.

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., instanceOf(domenico, 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 a the tool for assigning semantics to ontology languages.
- Ontology languages vs. conceptual data models:
  Conceptual schema 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
- Frame based
  - Frame Systems
  - OKBC, XOL
- Logic based
  - Description Logics (e.g., $\mathcal{SHOIQ}$, $\mathcal{DLR}$, *DL-Lite*, OWL, ...)
  - Rules (e.g., RuleML, LP/Prolog, F-Logic)
  - First Order Logic (e.g., KIF)
  - Non-classical logics (e.g., Nonmonotonic, probabilistic)

Queries

An ontology language may also include constructs for expressing queries.

Query

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

Meta-Query

In 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, Calì&Kifer VLDB'06]

Ontology languages vs. query languages

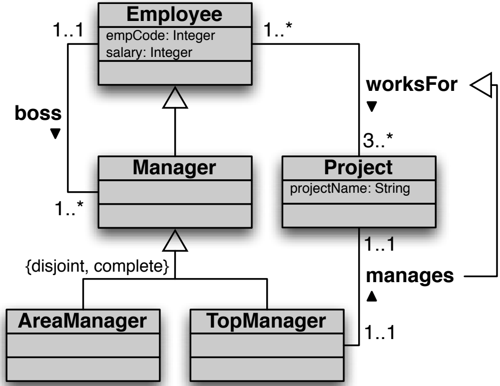
Ontology languages:

- 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



$$
\begin{aligned}
q(ce, cm, se, sm) \;\leftarrow\; & \mathsf{worksFor}(e,p) \land \mathsf{manages}(m,p) \land \mathsf{boss}(m,e) \land \\
& \mathsf{empCode}(e, ce) \land \mathsf{empCode}(m, cm) \land \\
& \mathsf{salary}(e, se) \land \mathsf{salary}(m, sm) \land se \geq sm
\end{aligned}
$$

Query answering under different assumptions

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

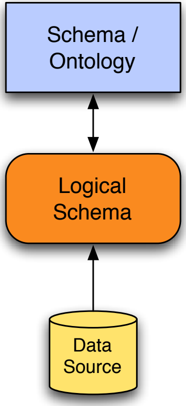
- traditional database assumption
- knowledge representation assumption

Query answering under the database assumption

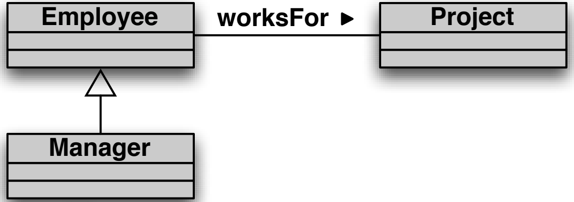
- Data are completely specified (CWA), and typically large.
- Schema/intensional information used in the design phase.
- At runtime, the data is assumed to satisfy the schema, and therefore the schema is not used.
- Queries allow for complex navigation paths in the data (cf. SQL).

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

Query answering under the database assumption (cont'd)



Query answering under the database assumption – Example



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

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

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

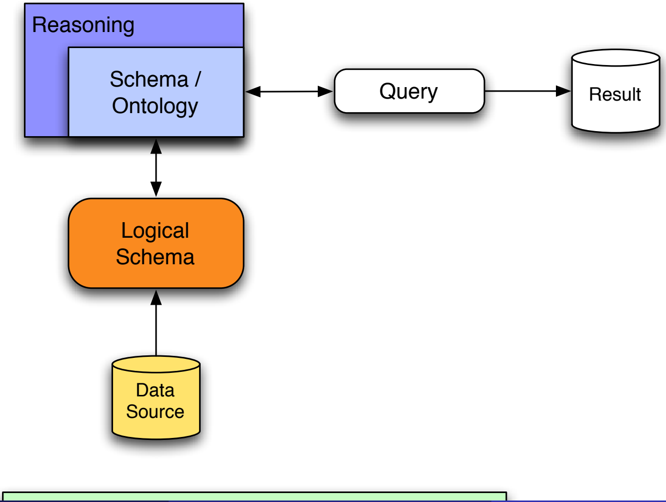
Answer: { john }

Query answering under the KR assumption

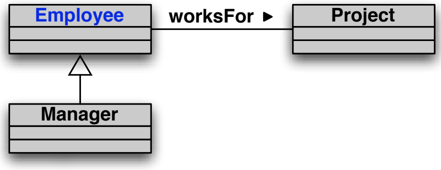
- An ontology (or conceptual schema, or knowledge base) imposes constraints on the data.
- Actual data may be incomplete or inconsistent w.r.t. such constraints.
- The system has to take into account intensional information during query answering, and overcome incompleteness or inconsistency.
- Size of the data is not considered critical (comparable to the size of the intensional information).
- Queries are typically simple, i.e., atomic (the name of a concept).

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

Query answering under the KR assumption (cont'd)



Query answering under the KR assumption – Example



Partial DB assumption: we have a (complete) table in the database only for some concepts/relationships.

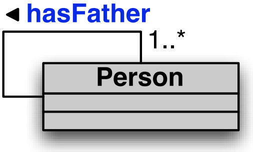
DB: Manager = { john, nick }
Project = { prA, prB }
worksFor = { (john,prA), (mary,prB) }

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

Answer: { john, nick, mary }

Rewritten query: $q(x) \leftarrow$ Employee$(x) \lor$ Manager$(x) \lor$ worksFor()

Query answering under the KR assumption – Example 2



Each person has a father, who is a person

Tables in the DB may be incompletely specified.

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

Queries:

$q_1(x,y) \leftarrow \mathsf{hasFather}(x,y)$

$q_2(x) \leftarrow \mathsf{hasFather}(x,y)$

$q_3(x) \leftarrow \mathsf{hasFather}(x,y_1), \mathsf{hasFather}(y_1,y_2), \mathsf{hasFather}(y_2,y_3)$

$q_4(x,y_3) \leftarrow \mathsf{hasFather}(x,y_1), \mathsf{hasFather}(y_1,y_2), \mathsf{hasFather}(y_2,y_3)$

Answers:

to q_1: { (john,nick), (nick,toni) }

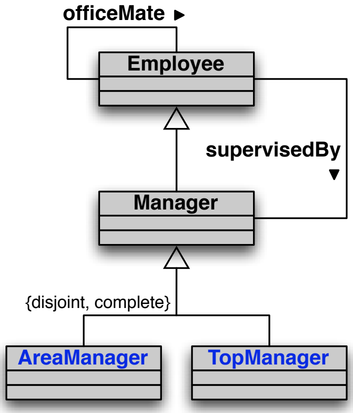
to q_2: { john, nick, toni }

to q_3: { john, nick, toni }

to q_4: { }

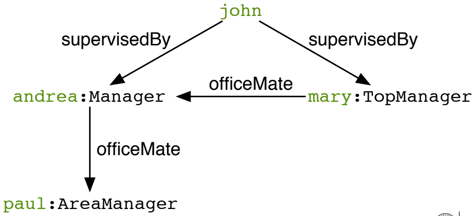
Rewritten queries: see later

QA under the KR assumption – Andrea's Example

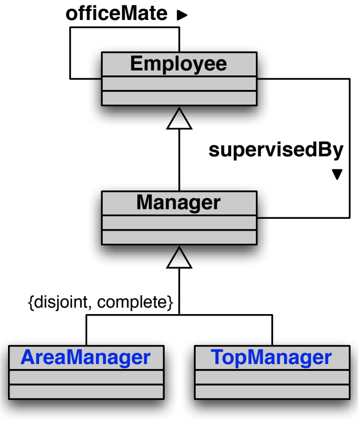


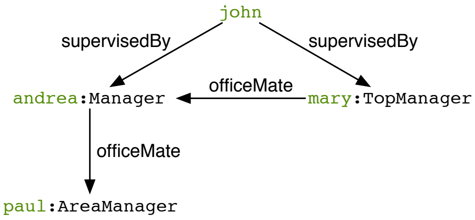
Tables may be incompletely specified.

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



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





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

Answer: { john }

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

Rewritten query? ???There is none (at least not in SQL).

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 the rest of this tutorial we provide an insight in state-of-the-art technology in this area.

Part II

Description Logics and the *DL-Lite* family

Outline

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 [BCM+03] are logics specifically designed to represent and reason on structured knowledge:

The domain 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

Description Logics 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)
- ...

Ingredients of a Description Logic

A Description Logic is characterized by:

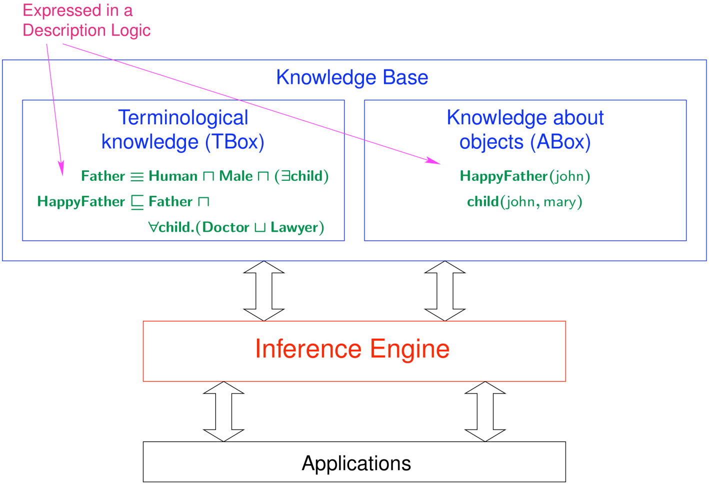
1. A description language: how to form concepts and roles
   $\mathsf{Human} \sqcap \mathsf{Male} \sqcap \exists \mathsf{hasChild} \sqcap \forall \mathsf{hasChild}.(\mathsf{Doctor} \sqcup \mathsf{Lawyer})$

2. A mechanism to specify knowledge about concepts and roles (i.e., a TBox)
   $$\begin{aligned} \mathcal{T} = \{ \ & \mathsf{Father} \equiv \mathsf{Human} \sqcap \mathsf{Male} \sqcap \exists \mathsf{hasChild}, \\ & \mathsf{HappyFather} \sqsubseteq \mathsf{Father} \sqcap \forall \mathsf{hasChild}.(\mathsf{Doctor} \sqcup \mathsf{Lawyer}) \ \} \end{aligned}$$

3. A mechanism to specify properties of objects (i.e., an ABox)
   $\mathcal{A} = \{ \ \mathsf{HappyFather}(\mathtt{john}), \quad \mathsf{hasChild}(\mathtt{john}, \mathtt{mary}) \ \}$

4. A set of inference services: how to reason on a given KB
   $\mathcal{T} \models \mathsf{HappyFather} \sqsubseteq \exists \mathsf{hasChild}.(\mathsf{Doctor} \sqcup \mathsf{Lawyer})$
   $\mathcal{T} \cup \mathcal{A} \models (\mathsf{Doctor} \sqcup \mathsf{Lawyer})(\mathtt{mary})$

Architecture of a Description Logic system



Description language

A description language is characterized by a set of constructs for building complex concepts and roles starting from atomic ones:

- concepts correspond to classes: interpreted as sets of objects
- roles corr. to relationships: interpreted as binary relations on objects

Formal semantics is given in terms of interpretations.

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

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

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

Concept constructors

Construct	Syntax	Example	Semantics
atomic concept	A	Doctor	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
atomic role	P	hasChild	$P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
atomic negation	$\neg A$	¬Doctor	$\Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$
conjunction	$C \sqcap D$	Hum ⊓ Male	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
(unqual.) exist. res.	$\exists R$	∃hasChild	$\{ a \mid \exists b. (a, b) \in R^{\mathcal{I}} \}$
value restriction	$\forall R.C$	∀hasChild.Male	$\{a \mid \forall b. (a, b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}$
bottom	$\bot$		$\emptyset$

(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

Construct	$\mathcal{AL}\cdot$	Syntax	Semantics
disjunction	$\mathcal{U}$	$C \sqcup D$	$C^\mathcal{I} \cup D^\mathcal{I}$
top		$\top$	$\Delta^\mathcal{I}$
qual. exist. res.	$\mathcal{E}$	$\exists R.C$	$\{ a \mid \exists b. (a,b) \in R^\mathcal{I} \land b \in C^\mathcal{I} \}$
(full) negation	$\mathcal{C}$	$\neg C$	$\Delta^\mathcal{I} \setminus C^\mathcal{I}$
number restrictions	$\mathcal{N}$	$(\geq k\, R)$	$\{ a \mid \#\{b \mid (a,b) \in R^\mathcal{I}\} \geq k \}$
		$(\leq k\, R)$	$\{ a \mid \#\{b \mid (a,b) \in R^\mathcal{I}\} \leq k \}$
qual. number restrictions	$\mathcal{Q}$	$(\geq k\, R.C)$	$\{ a \mid \#\{b \mid (a,b) \in R^\mathcal{I} \land b \in C^\mathcal{I}\} \geq k \}$
		$(\leq k\, R.C)$	$\{ a \mid \#\{b \mid (a,b) \in R^\mathcal{I} \land b \in C^\mathcal{I}\} \leq k \}$
inverse role	$\mathcal{I}$	R^-	$\{ (a,b) \mid (b,a) \in R^\mathcal{I} \}$
role closure	reg	$\mathcal{R}^*$	$(R^\mathcal{I})^*$

Many different DL constructs and their combinations have been investigated.

Further examples of DL constructs

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

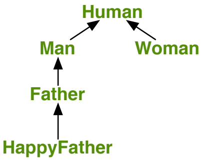
Reasoning on concept expressions

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

Basic reasoning tasks:

1. Concept satisfiability: does C admit a model?
2. Concept subsumption $C \sqsubseteq D$: does $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ hold for all interpretations $\mathcal{I}$?

Subsumption used to build the concept hierarchy:



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

Complexity of reasoning on concept expressions

Complexity of concept satisfiability: [DLNN97]

$\mathcal{AL}$, $\mathcal{ALN}$	PTIME
$\mathcal{ALU}$, $\mathcal{ALUN}$	NP-complete
$\mathcal{ALE}$	coNP-complete
$\mathcal{ALC}$, $\mathcal{ALCN}$, $\mathcal{ALCI}$, $\mathcal{ALCQI}$	PSPACE-complete

Observations:

- Two sources of complexity:
  - union ($\mathcal{U}$) of type NP,
  - existential quantification ($\mathcal{E}$) 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 $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox:

Description Logics TBox

Consists of a set of assertions on concepts and roles:

- Inclusion assertions on concepts: $C_1 \sqsubseteq C_2$
- Inclusion assertions on roles: $R_1 \sqsubseteq R_2$
- Property assertions on (atomic) roles:

  (**transitive** P) (**symmetric** P) (**domain** P C)
  (**functional** P) (**reflexive** P) (**range** P C) $\cdots$

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 $C_1 \sqsubseteq C_2,\ C_2 \sqsubseteq C_1$.

TBox assertions:

- Inclusion assertions on concepts:

$$\begin{array}{rcl} \mathsf{Father} & \equiv & \mathsf{Human} \sqcap \mathsf{Male} \sqcap \exists \mathsf{hasChild} \\ \mathsf{HappyFather} & \sqsubseteq & \mathsf{Father} \sqcap \forall \mathsf{hasChild}.(\mathsf{Doctor} \sqcup \mathsf{Lawyer} \sqcup \mathsf{HappyPerson}) \\ \mathsf{HappyAnc} & \sqsubseteq & \forall \mathsf{descendant}.\mathsf{HappyFather} \\ \mathsf{Teacher} & \sqsubseteq & \neg\mathsf{Doctor} \sqcap \neg\mathsf{Lawyer} \end{array}$$

- Inclusion assertions on roles:

$$\mathsf{hasChild} \sqsubseteq \mathsf{descendant} \qquad \mathsf{hasFather} \sqsubseteq \mathsf{hasChild}^-$$

- Property assertions on roles:
  (**transitive** descendant), (**reflexive** descendant),
  (**functional** hasFather)

ABox membership assertions:

- $\mathsf{Teacher}(\mathtt{mary})$, $\mathsf{hasFather}(\mathtt{mary}, \mathtt{john})$, $\mathsf{HappyAnc}(\mathtt{john})$

Semantics of a Description Logics knowledge base

The semantics is given by specifying when an interpretation $\mathcal{I}$ satisfies an assertion:

- $C_1 \sqsubseteq C_2$ is satisfied by $\mathcal{I}$ if $C_1^\mathcal{I} \subseteq C_2^\mathcal{I}$.
- $R_1 \sqsubseteq R_2$ is satisfied by $\mathcal{I}$ if $R_1^\mathcal{I} \subseteq R_2^\mathcal{I}$.
- A property assertion (**prop** P) is satisfied by $\mathcal{I}$ if $P^\mathcal{I}$ is a relation that has the property **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

Model of a DL knowledge base

An interpretation $\mathcal{I}$ is a model of $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ 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 ...

Logical implication

$\mathcal{O}$ logically implies and assertion α, written $\mathcal{O} \models \alpha$, if α 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., $\mathcal{T} \models C_1 \sqsubseteq C_2$.
- Equivalence: C_1 and C_2 are equivalent wrt $\mathcal{T}$ if for every model $\mathcal{I}$ of $\mathcal{T}$ we have $C_1^{\mathcal{I}} = C_2^{\mathcal{I}}$, i.e., $\mathcal{T} \models C_1 \equiv C_2$.
- Disjointness: C_1 and C_2 are disjoint wrt $\mathcal{T}$ if for every model $\mathcal{I}$ of $\mathcal{T}$ we have $C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} = \emptyset$, i.e., $\mathcal{T} \models C_1 \sqcap C_2 \equiv \bot$.
- Functionality implication: A functionality assertion $(\textbf{funct}\ R)$ 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., $\mathcal{T} \models (\textbf{funct}\ R)$.

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

- Inclusion assertions on concepts:

$$\begin{array}{rcl} \mathsf{Father} & \equiv & \mathsf{Human} \sqcap \mathsf{Male} \sqcap \exists \mathsf{hasChild} \\ \mathsf{HappyFather} & \sqsubseteq & \mathsf{Father} \sqcap \forall \mathsf{hasChild}.(\mathsf{Doctor} \sqcup \mathsf{Lawyer} \sqcup \mathsf{HappyPerson}) \\ \mathsf{HappyAnc} & \sqsubseteq & \forall \mathsf{descendant}.\mathsf{HappyFather} \\ \mathsf{Teacher} & \sqsubseteq & \neg \mathsf{Doctor} \sqcap \neg \mathsf{Lawyer} \end{array}$$

- Inclusion assertions on roles:

$$\mathsf{hasChild} \sqsubseteq \mathsf{descendant} \qquad \mathsf{hasFather} \sqsubseteq \mathsf{hasChild}^-$$

- Property assertions on roles:
  (**transitive** descendant), (**reflexive** descendant), (**functional** hasFather)

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

- Membership assertions:
  $\mathsf{Teacher}(\mathtt{mary})$, $\mathsf{hasFather}(\mathtt{mary}, \mathtt{john})$, $\mathsf{HappyAnc}(\mathtt{john})$

The above TBox and ABox logically imply: $\mathsf{HappyPerson}(\mathtt{mary})$

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

3 A gentle introduction to Description Logics

4 DLs as a formal language to specify ontologies
- DLs to specify ontologies
- DLs vs. OWL
- DLs vs. UML Class Diagrams

5 Queries in Description Logics

6 The *DL-Lite* family of tractable Description Logics

Relationship between DLs and ontology formalisms

- Description Logics are nowadays advocated to provide the foundations for ontology languages.
- Different versions of the Ontology Web Language (OWL) have been defined as syntactic variants of certain Description Logics.
- DLs are also ideally suited to capture the fundamental features of conceptual modeling formalims 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 Ontology Web Language (OWL) comes in different variants:

- OWL-Lite is a variant of the DL $\mathcal{SHIN}(D)$, where:
  - $\mathcal{S}$ 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
  - $\mathcal{N}$ stands for (unqualified) number restrictions
  - (D) stand for data types, which are necessary in any practical knowledge representation language
- OWL-DL is a variant of $\mathcal{SHOIQ}(D)$, where:
  - $\mathcal{O}$ stands for nominals, which means the possibility of using individuals in the TBox (i.e., the intensional part of the ontology)
  - $\mathcal{Q}$ stands for qualified number restrictions

DL constructs vs. OWL constructs

OWL contructor	DL constructor	Example
intersectionOf	$C_1 \sqcap \cdots \sqcap C_n$	Human $\sqcap$ Male
unionOf	$C_1 \sqcup \cdots \sqcup C_n$	Doctor $\sqcup$ Lawyer
complementOf	$\neg C$	$\neg$Male
oneOf	$\{a_1\} \sqcup \cdots \sqcup \{a_n\}$	{john} $\sqcup$ {mary}
allValuesFrom	$\forall P.C$	$\forall$hasChild.Doctor
someValuesFrom	$\exists P.C$	$\exists$hasChild.Lawyer
maxCardinality	$(\leq n\, P)$	($\leq$ 1 hasChild)
minCardinality	$(\geq n\, P)$	($\geq$ 2 hasChild)

DL axioms vs. OWL axioms

OWL axiom	DL syntax	Example
subClassOf	$C_1 \sqsubseteq C_2$	Human $\sqsubseteq$ Animal $\sqcap$ Biped
equivalentClass	$C_1 \equiv C_2$	Man $\equiv$ Human $\sqcap$ Male
disjointWith	$C_1 \sqsubseteq \neg C_2$	Man $\sqsubseteq \neg$Female
sameIndividualAs	$\{a_1\} \equiv \{a_2\}$	{presBush} $\equiv$ {G.W.Bush}
differentFrom	$\{a_1\} \sqsubseteq \neg\{a_2\}$	{john} $\sqsubseteq \neg${peter}
subPropertyOf	$P_1 \sqsubseteq P_2$	hasDaughter $\sqsubseteq$ hasChild
equivalentProperty	$P_1 \equiv P_2$	hasCost $\equiv$ hasPrice
inverseOf	$P_1 \equiv P_2^-$	hasChild $\equiv$ hasParent$^-$
transitiveProperty	$P^+ \sqsubseteq P$	ancestor$^+$ $\sqsubseteq$ ancestor
functionalProperty	$\top \sqsubseteq (\leq 1\, P)$	$\top \sqsubseteq (\leq 1$ hasFather$)$
inverseFunctionalProperty	$\top \sqsubseteq (\leq 1\, P^-)$	$\top \sqsubseteq (\leq 1$ hasSSN$^-)$

DLs vs. UML Class Diagrams

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

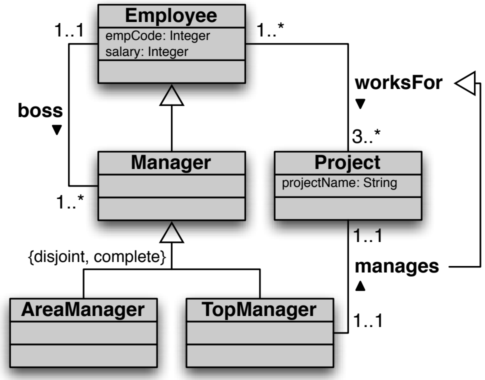
Encoding UML Class Diagrams in DLs

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

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

We illustrate the encoding by means of an example.

Encoding UML Class Diagrams in DLs – Example



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

$$(\textbf{funct}\ \mathsf{manages})$$
$$(\textbf{funct}\ \mathsf{manages}^-)$$

$$\mathsf{manages} \sqsubseteq \mathsf{worksFor}$$

$$\cdots$$

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 $\mathcal{T}$ in terms of a UML Class Diagram $\mathcal{T}_{\mathcal{D}}$ 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:
  1. the possibility to use disjunction (covering constraints)
  2. the interaction between role inclusions and functionality constraints (maximum 1 cardinality)

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

- 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 allows 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

3 A gentle introduction to Description Logics

4 DLs as a formal language to specify ontologies

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

6 The *DL-Lite* family of tractable Description Logics

Queries over Description Logics ontologies

We need more complex queries than simple concept (or role) expressions.

A conjunctive query $q(\vec{x})$ over an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ has the form:

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

where:

- $\vec{x}$ is a tuple of so-called distinguished variables.
  The number of variables in $\vec{x}$ is called the arity of q.
- $\vec{y}$ is a tuple of so-called non-distinguished variables,
- $q(\vec{x})$ is called the head of q.
- $conj(\vec{x}, \vec{y})$, called the body of q, is a conjunction of atoms, where each atom:
  - has as predicate symbol an atomic concept or role of $\mathcal{T}$,
  - may use variables in $\vec{x}$ and $\vec{y}$,
  - may use constants that are individuals of $\mathcal{A}$.

Queries over Description Logics ontologies (cont'd)

Note: we may also use for CQs a simplified notation

$$q(\vec{x}) \leftarrow body(\vec{x}, \vec{y})$$

where $body(\vec{x}, \vec{y})$ is a sequence constituted by the atoms in $conj(\vec{x}, \vec{y})$.

Example of conjunctive query

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

In simplified notation:

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

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

Certain answers to a query

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

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

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

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

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

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

Query answering over ontologies

Query answering over an ontology $\mathcal{O}$

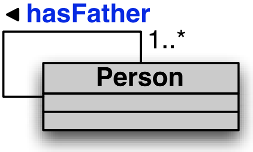
Is the problem of computing the certain answers to a query over $\mathcal{O}$.

Computing certain answers is a form of logical implication:

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

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

Query answering over ontologies – Example



TBox $\mathcal{T}$:

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

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

Queries:

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

Certain answers:

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

Unions of conjunctive queries

We consider also unions of CQs.

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

$$q(\vec{x}) \;\leftarrow\; \exists\vec{y_1}.\, conj(\vec{x},\vec{y_1}) \vee \cdots \vee \exists\vec{y_k}.\, conj(\vec{x},\vec{y_k})$$

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

Example

$$q(x) \leftarrow (\mathsf{Manager}(x) \wedge \mathsf{worksFor}(x,\mathtt{tones})) \;\vee$$
$$(\exists y.\, \mathsf{boss}(x,y) \wedge \mathsf{worksFor}(y,\mathtt{tones}))$$

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

Data and combined complexity

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

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

- Data complexity: 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).

$\leadsto$ Data complexity is the relevant complexity measure.

Complexity of query answering in DLs

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?

Outline

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 now two incomparable languages of this family, *DL-Lite*$_{\mathcal{F}}$, *DL-Lite*$_{\mathcal{R}}$ (we use *DL-Lite* to refer to both).
- We will see that *DL-Lite* has nice computational properties:
  - PTime in the size of the TBox (schema complexity)
  - LogSpace in the size of the ABox (data complexity)
  - enjoys FOL-rewritability
- We will see that *DL-Lite*$_{\mathcal{F}}$ and *DL-Lite*$_{\mathcal{R}}$ are in some sense the maximal DLs with these nice computational properties, which are lost with minimal additions of constructs.

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

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

TBox assertions:

- Concept inclusion assertions: $Cl \sqsubseteq Cr$, with:

$$\begin{array}{rcl} Cl & \longrightarrow & A \mid \exists Q \\ Cr & \longrightarrow & A \mid \exists Q \mid \neg A \mid \neg \exists Q \\ Q & \longrightarrow & P \mid P^- \end{array}$$

- Functionality assertions: $(\textbf{funct}\ Q)$

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$_{\mathcal{R}}$ ontologies

TBox assertions:

- Concept inclusion assertions: $Cl \sqsubseteq Cr$, with:

$$\begin{array}{rcl} Cl & \longrightarrow & A \mid \exists Q \\ Cr & \longrightarrow & A \mid \exists Q \mid \neg A \mid \neg \exists Q \\ Q & \longrightarrow & P \mid P^- \end{array}$$

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

$$R \longrightarrow Q \mid \neg Q$$

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.

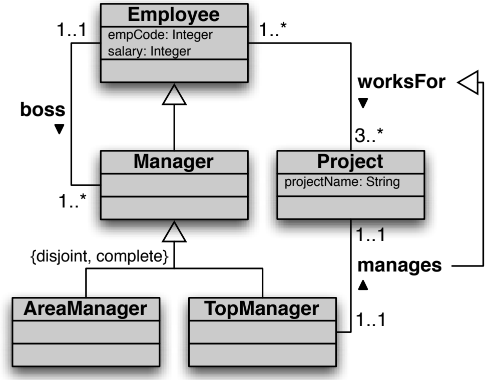
Semantics of *DL-Lite*

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

Capturing basic ontology constructs in *DL-Lite*

ISA between classes	$A_1 \sqsubseteq A_2$
Disjointness between classes	$A_1 \sqsubseteq \neg A_2$
Domain and range of relations	$\exists P \sqsubseteq A_1$ $\quad \exists P^- \sqsubseteq A_2$
Mandatory participation	$A_1 \sqsubseteq \exists P$ $\quad A_2 \sqsubseteq \exists P^-$
Functionality of relations (in *DL-Lite*$_\mathcal{F}$)	$(\mathbf{funct}\ P)$ $\quad (\mathbf{funct}\ P^-)$
ISA between relations (in *DL-Lite*$_\mathcal{R}$)	$Q_1 \sqsubseteq Q_2$
Disjointness between relations (in *DL-Lite*$_\mathcal{R}$)	$Q \sqsubseteq \neg Q$

DL-Lite – Example



$$
\begin{array}{rcl}
\mathsf{Manager} & \sqsubseteq & \mathsf{Employee}\\
\mathsf{AreaManager} & \sqsubseteq & \mathsf{Manager}\\
\mathsf{TopManager} & \sqsubseteq & \mathsf{Manager}\\
\mathsf{AreaManager} & \sqsubseteq & \neg\mathsf{TopManager}\\
\mathsf{Employee} & \sqsubseteq & \exists\mathsf{salary}\\
\exists\mathsf{salary}^- & \sqsubseteq & \mathsf{Integer}\\
\exists\mathsf{worksFor} & \sqsubseteq & \mathsf{Employee}\\
\exists\mathsf{worksFor}^- & \sqsubseteq & \mathsf{Project}\\
\mathsf{Employee} & \sqsubseteq & \exists\mathsf{worksFor}\\
\mathsf{Project} & \sqsubseteq & \exists\mathsf{worksFor}^-\\
 & \vdots &
\end{array}
$$

Additionally, in *DL-Lite*$_\mathcal{F}$: (**funct** manages), (**funct** manages$^-$), . . .
in *DL-Lite*$_\mathcal{R}$: manages $\sqsubseteq$ worksFor

Note: in *DL-Lite* we cannot capture: – completeness of the hierarchy,
– number restrictions

Properties of *DL-Lite*

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

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

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

$$Cl \sqsubseteq Cr_1 \sqcap Cr_2 \quad \text{is equivalent to} \quad \begin{array}{lcl} Cl & \sqsubseteq & Cr_1 \\ Cl & \sqsubseteq & Cr_2 \end{array}$$

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

Properties of $DL\text{-}Lite_{\mathcal{F}}$

$DL\text{-}Lite_{\mathcal{F}}$ does not enjoy the finite model property.

Example

TBox $\mathcal{T}$:

$$\begin{array}{llll} \mathsf{Nat} \sqsubseteq \exists \mathsf{succ} & \qquad & \exists \mathsf{succ}^- \sqsubseteq \mathsf{Nat} \\ \mathsf{Zero} \sqsubseteq \mathsf{Nat} \sqcap \neg \exists \mathsf{succ}^- & \qquad & (\textbf{funct}\ \mathsf{succ}^-) \end{array}$$

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

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

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

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

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

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

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

Complexity results for *DL-Lite*

1. We have seen that $\textit{DL-Lite}_{\mathcal{F}}$ and $\textit{DL-Lite}_{\mathcal{R}}$ can capture the essential features of prominent conceptual modeling formalisms.
2. In the next part, 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.
3. We will also see that *DL-Lite* is essentially the maximal DL enjoying these nice computational properties.

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

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

Part III

Reasoning in the *DL-Lite* family

Outline

Outline

7. TBox reasoning
   - Reducing to subsumption
   - Reducing to ontology satisfiability
8. TBox & ABox Reasoning
9. Complexity of reasoning in Description Logics

Remark

In the following,

- a TBox $\mathcal{T}$ that is either a *DL-Lite*$_\mathcal{R}$ or a *DL-Lite*$_\mathcal{F}$ TBox is simply called TBox.
- C, possibly with subscript, denotes a general concept, i.e.,

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

  where A is an atomic concept, P is an atomic role, and Q is a basic role.
- R, possibly with subscript, denotes a general role, i.e.,

$$R \longrightarrow Q \mid \neg Q$$

Reasoning services

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

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

From TBox reasoning to ontology satisfiability

In the following we will show how to reduce TBox reasoning to ontology satisfiability.

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

- We first will show how to reduce TBox reasoning services to concept/role subsumption.
- Then we will provide reductions from concept/role subsumption to ontology satisfiability.

Concept/role satisfiability, equivalence, and disjointness

Theorem

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

Proof (sketch)

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

Analogous reductions for role satisfiability, equivalence and disjointness.

From implication of functionalities to subsumption

Theorem

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

Proof (sketch)

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

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

From concept subsumption to ontology satisfiability

Theorem

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

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

Proof (sketch)

"$\Leftarrow$" Suppose that $\mathcal{O}_{C_1 \sqsubseteq C_2}$ is unsatisfiable, but $\mathcal{T} \not\models C_1 \sqsubseteq C_2$, i.e., there exists a model $\mathcal{I}$ of $\mathcal{T}$ such that $C_1^{\mathcal{I}} \not\subseteq C_2^{\mathcal{I}}$. From $\mathcal{I}$ we construct a model for $\mathcal{O}_{C_1 \sqsubseteq C_2}$, thus getting a contradiction.

"$\Rightarrow$" Suppose that $\mathcal{O}_{C_1 \sqsubseteq C_2}$ is satisfiable, and let $\mathcal{I}$ be a model of $\mathcal{O}_{C_1 \sqsubseteq C_2}$. Then $\mathcal{I} \models \mathcal{T}$, and $\mathcal{I} \models C_1(c)$ and $\mathcal{I} \models \neg C_2(c)$, i.e., $\mathcal{I} \not\models C_1 \sqsubseteq C_2$, i.e., $\mathcal{T} \not\models C_1 \sqsubseteq C_2$.

From role subsumption to ontology satisfiability

Theorem

Let $\mathcal{T}$ be a *DL-Lite*$_{\mathcal{R}}$ TBox, Q_1 and Q_2 two general roles, $\hat{P}$ an atomic role not in $\mathcal{T}$, and c_1, c_2 two constants. $\mathcal{T} \models R_1 \sqsubseteq R_2$ iff the ontology $\mathcal{O}_{R_1 \sqsubseteq R_2} = \langle \mathcal{T} \cup \{\hat{P} \sqsubseteq R_1, \hat{P} \sqsubseteq \neg R_2\},\ \{\hat{P}(c_1, c_2)\}\rangle$ is unsatisfiable.

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

Proof (sketch)

Analogous to above.

Notice that $\mathcal{O}_{Q_1 \sqsubseteq Q_2}$ *is inherently a DL-Lite*$_{\mathcal{R}}$ *ontology.*

From role subsumption to ontology satisfiability (cont'd)

Theorem

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

1. $\mathcal{T} \models Q_1 \sqsubseteq Q_2$ iff Q_1 is unsatisfiable.
2. $\mathcal{T} \models \neg Q_1 \sqsubseteq Q_2$ iff $\mathcal{T}$ is unsatisfiable.
3. $\mathcal{T} \models Q_1 \sqsubseteq \neg Q_2$ iff either
   - (a) $\exists Q_1$ and $\exists Q_2$ are disjoint, or
   - (b) $\exists Q_1^-$ and $\exists Q_2^-$ are disjoint.

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

From role subsumption to ontology satisfiability (cont'd)

Theorem

Let $\mathcal{T}$ be a *DL-Lite*$_{\mathcal{F}}$ TBox, Q_1 and Q_2 two basic roles such that $Q_1 \neq Q_2$, $\hat{A}$ an atomic concept not in $\mathcal{T}$, and c a constant. Then,

1. $\mathcal{T} \models Q_1 \sqsubseteq Q_2$ iff either
   - (a) the ontology $\mathcal{O}_{\exists Q_1 \sqsubseteq \neg \exists Q_1} = \langle \mathcal{T} \cup \{\hat{A} \sqsubseteq \exists Q_1\},\ \{\hat{A}(c)\}\rangle$ is unsatisfiable, or
   - (b) the ontology $\mathcal{O}_{\exists Q_1^- \sqsubseteq \neg \exists Q_1^-} = \langle \mathcal{T} \cup \{\hat{A} \sqsubseteq \exists Q_1^-\},\ \{\hat{A}(c)\}\rangle$ is unsatisfiable.
2. $\mathcal{T} \models \neg Q_1 \sqsubseteq Q_2$ iff $\mathcal{T}$ is unsatisfiable.
3. $\mathcal{T} \models Q_1 \sqsubseteq \neg Q_2$ iff either
   - (a) the ontology $\mathcal{O}_{\exists Q_1 \sqsubseteq \neg \exists Q_2} = \langle \mathcal{T} \cup \{\hat{A} \sqsubseteq \exists Q_1, \hat{A} \sqsubseteq \exists Q_2\},\ \{\hat{A}(c)\}\rangle$ is unsatisfiable, or
   - (b) the ontology $\mathcal{O}_{\exists Q_1^- \sqsubseteq \neg \exists Q_2^-} = \langle \mathcal{T} \cup \{\hat{A} \sqsubseteq \exists Q_1^-, \hat{A} \sqsubseteq \exists Q_2^-\},\ \{\hat{A}(c)\}\rangle$ is unsatisfiable.

- The results above say us that we can support TBox reasoning services relying on ontology satisfiability services.
- 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

7 TBox reasoning

8 TBox & ABox Reasoning
- Query answering
- Ontology satisfiability

9 Complexity of reasoning in Description Logics

Query answering and instance checking

- Concept Instance Checking: Verify wether 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 wether 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 $\mathcal{O} \models q(\vec{c})$.

Notice that instance checking is a special case of query answering: it amounts to answering the boolean query $C(c)$ (resp., $Q(c_1, c_2)$) over $\mathcal{O}$ (in this case $\vec{c}$ is the empty tuple).

Certain answers

We recall that

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

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

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

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

Data complexity of query answering

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

We consider a setting where the size of the data largely dominates the size of the conceptual layer $\rightsquigarrow$ 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:

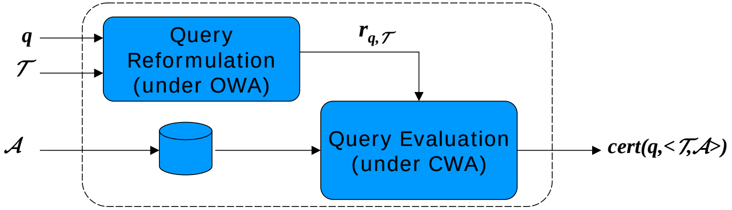
1. For which ontology languages can we answer queries over an ontology efficiently?
2. How complex becomes query answering over an ontology when we consider more expressive ontology languages?

Data complexity and $\mathcal{Q}$-rewritability

To study data complexity, we need to separate the contribution of $\mathcal{A}$ from the contribution of q and $\mathcal{T}$

$\leadsto$ Study $\mathcal{Q}$-rewritability for query language $\mathcal{Q}$.

$\mathcal{Q}$-rewritability



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

1. Perfect reformulation: producing the query $r_{q,\mathcal{T}}$, namely the function $cert[q,\mathcal{T}](\cdot)$
2. Query evaluation: evaluating $r_{q,\mathcal{T}}$ over the ABox $\mathcal{A}$ seen as a complete database, and forgetting about the TBox $\mathcal{T}$ $\leadsto$ Produces $cert(q,\langle\mathcal{T},\mathcal{A}\rangle)$

Let $\mathcal{Q}$ be a query language

Query answering for an ontology language is $\mathcal{Q}$-rewritable if $r_{q,\mathcal{T}}$ is in $\mathcal{Q}$.

$\mathcal{Q}$-rewritability: interesting cases

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

- When $\mathcal{Q}$ is FOL (i.e., the language enjoys FOL-rewritability)
  $\leadsto$ Query evaluation can be done in SQL, i.e., via an RDBMS (*Note:* FOL is in LogSpace).
- When $\mathcal{Q}$ is an NLogSpace-hard language
  $\leadsto$ Query evaluation requires (at least) linear recursion.
- When $\mathcal{Q}$ is a PTime-hard language
  $\leadsto$ Query evaluation requires (at least) recursion (e.g., Datalog).
- When $\mathcal{Q}$ is a coNP-hard language
  $\leadsto$ Query evaluation requires (at least) power of Disjunctive Datalog.

- We now study $\mathcal{Q}$-rewritability of query answering over *DL-Lite* ontologies.
- In particular we will show that both *DL-Lite*$_{\mathcal{R}}$ and *DL-Lite*$_{\mathcal{F}}$ enjoy FOL-rewritability of conjunctive query answering.

Query answering over unsatisfiable ontologies

- 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*$_{\mathcal{R}}$ ontologies.

Remark

we call positive inclusions (PIs) assertions of the form

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

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

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

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

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

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

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

Query answering in *DL-Lite*$_\mathcal{R}$: Query reformulation

Intuition: Use the PIs as basic rewriting rules

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

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

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

Basic rewriting step:

when the atom unifies with the **head** of the rule.

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

Towards the computation of the perfect reformulation, we add to the input query above the following query

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

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

Query answering in $DL\text{-}Lite_{\mathcal{R}}$: Query reformulation (cont'd)

Consider now the query

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

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

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

We add to the reformulation the query

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

Query answering in $\mathit{DL\text{-}Lite}_{\mathcal{R}}$: Query reformulation (cont'd)

Conversely, for the query

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

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

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

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

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

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

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

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

An analogous behavior with join variables

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

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

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

The PI above does not apply to the atom teaches(x, y).

Conversely, the PI

$$\exists \mathsf{teaches}^- \sqsubseteq \mathsf{Course}$$

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

applies to the atom Course(y).

We add to the perfect reformulation the query

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

Query answering in $\mathit{DL\text{-}Lite}_{\mathcal{R}}$: Query reformulation (cont'd)

We now have the query

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

The PI $\qquad \mathsf{Professor} \sqsubseteq \exists\mathsf{teaches}$

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

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

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

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

We can now apply the PI above, and add to the reformulation the query

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

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

Reformulate the CQ q into a set of queries: apply to q in all possible ways the PIs in the TBox $\mathcal{T}$:

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

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

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

Unifying atoms can make applicable rules that could not be applied otherwise.

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

Algorithm PerfectRef $(q, \mathcal{T}_P)$
Input: conjunctive query q, set of *DL-Lite*$_\mathcal{R}$ PIs $\mathcal{T}_P$
Output: union of conjunctive queries PR
$PR := \{q\}$;
repeat
  $PR' := PR$;
  for each $q \in PR'$ **do**
  (a) **for each** g in q **do**
      for each PI I in $\mathcal{T}_P$ **do**
        if I is applicable to g
        then $PR := PR \cup \{\, q[g/(g, I)]\,\}$
  (b) **for each** g_1, g_2 in q **do**
      if g_1 and g_2 unify
      then $PR := PR \cup \{\tau(\textit{reduce}(q, g_1, g_2))\}$;
until $PR' = PR$;
return PR

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

Query answering in $DL\text{-}Lite_{\mathcal{R}}$: ABox storage

ABox $\mathcal{A}$ 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 $\mathtt{tab}_A$
  - populate $\mathtt{tab}_A$ with each $\langle d \rangle$ such that $A(c) \in \mathcal{A}$
- For each atomic role P used in the ABox,
  - define a binary relational table $\mathtt{tab}_P$
  - populate $\mathtt{tab}_P$ with each $\langle a, b \rangle$ such that $P(c_1, c_2) \in \mathcal{A}$

We denote with DB($\mathcal{A}$) the database obtained as above.

Query answering in *DL-Lite*$_{\mathcal{R}}$: Query evaluation

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

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

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

Theorem

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

In other words, query answering over a satisfiable *DL-Lite*$_\mathcal{R}$ ontology is FOL-rewritable.

Notice that we did not mention NIs of $\mathcal{T}$ in the theorem above. Indeed, when the ontology is satisfiable, we can ignore NIs and answer queries as NIs were not specified in $\mathcal{T}$.

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

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

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

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

ABox: $\mathsf{teaches}(\mathtt{John}, \mathtt{databases})$
$\mathsf{Professor}(\mathtt{Mary})$

It is easy to see that $\mathsf{Eval}(\mathsf{SQL}(r_{q,\mathcal{T}}), \mathsf{DB}(\mathcal{A}))$ in this case produces the set $\{\mathtt{John}, \mathtt{Mary}\}$.

Query answering in *DL-Lite*$_{\mathcal{R}}$: An interesting case

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

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

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

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

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

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

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

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

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

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

$\Downarrow$

...

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

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

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

Query answering in $\textit{DL-Lite}_{\mathcal{F}}$

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

As for NIs and functionality assertions, it is possible to show that they can be disregarded in query answering over satisfiable $\textit{DL-Lite}_{\mathcal{F}}$ ontologies.

The following result is therefore straightforward.

Theorem

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

In other words, query answering over a satisfiable $\textit{DL-Lite}_{\mathcal{F}}$ ontology is FOL-rewritable.

Satisfiability of ontologies with only PIs

Let us now attack the problem of establishing whether a 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 says us that PIs alone cannot generate ontology unsatisfiability.

Theorem

Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be either a *DL-Lite*$_\mathcal{R}$ or a *DL-Lite*$_\mathcal{F}$ ontology, where $\mathcal{T}$ contains only PIs. Then, $\mathcal{O}$ is satisfiable.

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

Unsatisfiability in *DL-Lite*$_\mathcal{R}$ ontologies can be however caused by NIs

Example: **TBox $\mathcal{T}$:** $\mathsf{Professor} \sqsubseteq \neg\mathsf{Student}$
$\exists\mathsf{teaches} \sqsubseteq \mathsf{Professor}$

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

In what follows we provide a mechanism to establish, in an efficient way, whether a *DL-Lite*$_\mathcal{R}$ ontology is satisfiable.

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

Satisfiability of a *DL-Lite*$_{\mathcal{R}}$ 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:

1. Let $\mathcal{T}_P$ the set of PIs in $\mathcal{T}$
2. For each NI between concepts (resp. roles) in $\mathcal{T}$, we ask $\langle \mathcal{T}_P, \mathcal{A} \rangle$ if there exists some individual (resp. pair of individuals) that contradicts N, i.e., we pose 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 $\langle \mathcal{T}_P \cup \{N\}, \mathcal{A} \rangle$ is unsatisfiable
3. We exploit PerfectRef to verify if $\langle \mathcal{T}_P, \mathcal{A} \rangle \models q_N$, i.e., we compute PerfectRef($q_N, \mathcal{T}_P$), and evaluate it (in fact its SQL encoding) over DB($\mathcal{A}$).

Example

PIs $\mathcal{T}_P$: $\exists$teaches $\sqsubseteq$ Professor

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

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

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

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

It is easy to see that $\langle \mathcal{T}_P, \mathcal{A} \rangle \models q_N$, and that
$\langle \mathcal{T}_P \cup \{\mathsf{Professor} \sqsubseteq \neg\mathsf{Student}\}, \mathcal{A} \rangle$ is unsatisfiable.

Queries for NIs

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

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

Given a NI $N \in \mathcal{T}$, we denote with q_N the corresponding CQ.

DL-Lite$_\mathcal{R}$: From satisfiability to query answering

Lemma [separation]

Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a *DL-Lite*$_\mathcal{R}$ ontology, and $\mathcal{T}_P$ the set of PIs in $\mathcal{T}$. Then, $\mathcal{O}$ is unsatisfiable iff there exists a NI $N \in \mathcal{T}$ such that $\langle \mathcal{T}_P, \mathcal{A} \rangle \models q_N$.

The lemma relies on the properties that NIs do not interact with each other, and interaction between NIs and PIs can be managed through PerfectRef. Notably, each NI can be processed individually.

DL-Lite$_{\mathcal{R}}$: FOL-rewritability of satisfiability

From the lemma above and the theorem on query answering for satisfiable *DL-Lite*$_{\mathcal{R}}$, we get the following result

Theorem

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

In other words, satisfiability of *DL-Lite*$_{\mathcal{R}}$ ontology can be reduced to FOL-query evaluation.

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

Unsatisfiability in *DL-Lite*$_\mathcal{F}$ ontologies can be caused by NIs and functionality assertions.

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

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

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

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

Satisfiability of a *DL-Lite*$_\mathcal{F}$ 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-Lite*$_\mathcal{R}$ ontologies (indeed, limited to NIs, *DL-Lite*$_\mathcal{F}$ ontologies are *DL-Lite*$_\mathcal{R}$ ontologies of a special kind).

As for functionality assertions we proceed as follows:

1. For each functionality assertion $F \in \mathcal{T}$ we ask if there exists 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 $\langle \{F\}, \mathcal{A} \rangle$ is unsatisfiable
2. To verify if $\mathcal{A} \models q_F$, we evaluate SQL(q_F) over DB($\mathcal{A}$).

Example

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

Query q_F: $q() \leftarrow TeachesTo(x,y), TeachesTo(z,y), x \neq z$

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

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

Queries for functionality assertions

For each functionality assertion in $\mathcal{T}$ we compute a FOL query according to the following rules:

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

Given a functionality assertion $F \in \mathcal{T}$, we denote with q_F the corresponding FOL query.

DL-Lite$_{\mathcal{R}}$: From satisfiability to query answering

Lemma

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

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

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

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

It also exploits the properties that NIs and PIs do not interact with functionalities: indeed, no functionality assertions are contradicted in a *DL-Lite*$_{\mathcal{F}}$ ontology $\mathcal{O}$, beyond those explicitly contradicted by the ABox.

Notably, the lemma asserts that to check ontology satisfiability, each NI and each functionality can be processed individually.

DL-Lite$_{\mathcal{R}}$: FOL-rewritability of satisfiability

By the lemma above and the theorem on query answering for satisfiable *DL-Lite*$_{\mathcal{F}}$, the following result follows

Theorem

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

(a) there exists a NI $N \in \mathcal{T}$ such that $\text{Eval}(\text{SQL}(\text{PerfectRef}(q_N, \mathcal{T}_P)), \text{DB}(\mathcal{A}))$ returns *true*

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

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

Outline

Complexity of query answering over satisfiable ontologies

Theorem

Query answering over both $\textit{DL-Lite}_{\mathcal{R}}$ and $\textit{DL-Lite}_{\mathcal{F}}$ 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. We guess the derivation of one of the CQs of the perfect rewriting, and an assignment to its existential variables. Checking the derivation and evaluating the guessed CQ over the ABox is then polynomial in combined complexity. NP-hardness follows from combined complexity of evaluating CQs over a database.
2. The number of CQs in the perfect reformulation is polynomial in the size of the TBox, and we can get them in PTIME.
3. Is the data complexity of evaluating FOL queries over a database.

Complexity of ontology satisfiability

Theorem

Checking satisfiability of both $DL\text{-}Lite_{\mathcal{R}}$ and $DL\text{-}Lite_{\mathcal{F}}$ 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*$_\mathcal{R}$ and *DL-Lite*$_\mathcal{F}$ 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.

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.

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

Summary of results on data complexity

	Cl	Cr	$\mathcal{F}$	$\mathcal{R}$	Data complexity of query answering
1	*DL-Lite*$_{\mathcal{F}}$		$\surd$	–	in LogSpace
2	*DL-Lite*$_{\mathcal{R}}$		–	$\surd$	in LogSpace
3	$A \mid \exists P.A$	A	–	–	NLogSpace-hard
4	A	$A \mid \forall P.A$	–	–	NLogSpace-hard
5	A	$A \mid \exists P.A$	$\surd$	–	NLogSpace-hard
6	$A \mid \exists P.A \mid A_1 \sqcap A_2$	A	–	–	PTime-hard
7	$A \mid A_1 \sqcap A_2$	$A \mid \forall P.A$	–	–	PTime-hard
8	$A \mid A_1 \sqcap A_2$	$A \mid \exists P.A$	$\surd$	–	PTime-hard
9	$A \mid \exists P.A \mid \exists P^-.A$	$A \mid \exists P$	–	–	PTime-hard
10	A	$A \mid \exists P.A \mid \exists P^-.A$	$\surd$	–	PTime-hard
11	$A \mid \exists P.A$	$A \mid \exists P.A$	$\surd$	–	PTime-hard
12	$A \mid \neg A$	A	–	–	coNP-hard
13	A	$A \mid A_1 \sqcup A_2$	–	–	coNP-hard
14	$A \mid \forall P.A$	A	–	–	coNP-hard

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 $\leadsto$ $\mathit{DLR\text{-}Lite}_{\mathcal{F}}$ and $\mathit{DLR\text{-}Lite}_{\mathcal{R}}$.
- RDFS is a subset of $\mathit{DL\text{-}Lite}_{\mathcal{R}}$ $\leadsto$ is FOL-rewritable, hence LogSpace.
- Horn-$\mathcal{SHIQ}$ [HMS05] is PTime-hard even for instance checking (line 11).
- DLP [GHVD03] is PTime-hard (line 6)
- $\mathcal{EL}$ [BBL05] is PTime-hard (line 6).

Qualified existential quantification in the lhs of inclusions

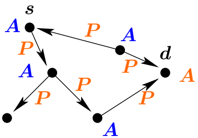
Adding qualified existential on the lhs of inclusions makes instance checking (and hence query answering) NLogSpace-hard:

	Cl	Cr	$\mathcal{F}$	$\mathcal{R}$	Data complexity
3	$A \mid \exists P.A$	A	$-$	$-$	NLogSpace-hard

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

Ontology $\mathcal{O}$: a single inclusion assertion $\exists P.A \sqsubseteq A$

Database $\mathcal{D}$: encodes graph using P and asserts $A(d)$



Result:

$(\mathcal{O}, \mathcal{D}) \models A(s)$ 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:

	Cl	Cr	$\mathcal{F}$	$\mathcal{R}$	Data complexity
3	$A \mid \exists P.A$	A	$-$	$-$	NLogSpace-hard

By reduction from reachability in directed graphs

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

Follows from 3 by replacing $\exists P.A_1 \sqsubseteq A_2$ with $A_1 \sqsubseteq \forall P^-.A_2$

5	A	$A \mid \exists P.A$	$\surd$	$-$	NLogSpace-hard

Proved by simulating in the reduction $\exists P.A_1 \sqsubseteq A_2$ via $A_1 \sqsubseteq \exists P^-.A_2$ and $(\textbf{funct}\ P^-)$

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 $\textsc{PTime}$-complete.

Reduction from Path System Accessibility

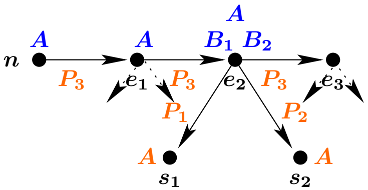
Given an instance $PS = (N, E, S, t)$, we construct

- TBox $\mathcal{T}$ consisting of the inclusion assertions

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

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

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



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

coNP-hard cases

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

Query answering is coNP-hard in data complexity for:

	Cl	Cr	$\mathcal{F}$	$\mathcal{R}$	Data complexity
14	$A \mid \neg A$	A	–	–	coNP-hard
15	A	$A \mid A_1 \sqcup A_2$	–	–	coNP-hard
16	$A \mid \forall P.A$	A	–	–	coNP-hard

All three cases are proved by adapting the proof of coNP-hardness of instance checking for $\mathcal{ALE}$ by [DLNS94].

2+2-SAT

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

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

$$\begin{aligned} c_1 &= \ell_1 \vee \ell_2 \vee \neg\ell_3 \vee \neg\ell_4 \\ c_2 &= \mathit{false} \vee \mathit{false} \vee \neg\ell_1 \vee \neg\ell_4 \\ c_3 &= \mathit{false} \vee \ell_4 \vee \neg\mathit{true} \vee \neg\ell_2 \end{aligned}$$

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 letters $\ell_1, \ldots, \ell_n$, *true*, *false*

- ABox $\mathcal{A}_\varphi$ constructed from φ (concepts L, T, F, roles P_1, P_2, N_1, N_2):
  - for each letter ℓ_i: $L(\ell_i)$
  - for each clause $c = \ell_1 \lor \ell_2 \lor \neg\ell_3 \lor \neg\ell_4$:
    $P_1(c, \ell_1)$, $P_2(c, \ell_2)$, $N_1(c, \ell_3)$, $N_2(c, \ell_4)$
  - $T($*true*$)$, $F($*false*$)$
- TBox $\mathcal{T} = \{ L \sqsubseteq T \sqcup F \}$
- $q() \leftarrow P_1(c, \ell_1), P_2(c, \ell_2), N_1(c, \ell_3), N_2(c, \ell_4),$
  $F(\ell_1), F(\ell_2), T(\ell_3), T(\ell_4)$

We have: $\langle \mathcal{T}, \mathcal{A}_\varphi \rangle \models q$ iff φ is not satisfiable.
Intuition: each model of $\mathcal{T}$ partitions L into T and F, and corresponds to a truth assignment to $\ell_1, \ldots, \ell_n$. q asks for a false clause.

Part IV

Linking data to ontologies

Outline

Outline

10 The Description Logic *DL-Lite*$_{\mathcal{A}}$
- Missing features in *DL-Lite*
- Combining functionality and role inclusions
- Syntax and semantics of *DL-Lite*$_{\mathcal{A}}$
- Reasoning in *DL-Lite*$_{\mathcal{A}}$

11 Connecting ontologies to relational data

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*$_{\mathcal{F}}$ and *DL-Lite*$_{\mathcal{R}}$), but *were not allowed to be used together*.

Let us first analyze this last point.

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

TBox $\mathcal{T}$: $A \sqsubseteq \exists P$ $\quad P \sqsubseteq S$

$\exists P^- \sqsubseteq A$ $\quad$ $(\textbf{funct } S)$

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

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

Hence, we get:

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

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

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

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

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

DL-Lite$_\mathcal{A}$ is a Description Logic designed to capture as much features as possible of conceptual data models, while preserving nice computational properties for query answering.

- Enjoys FOL-rewritability, and hence is LogSpace in data complexity.
- 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, ecc.
- Values are connected to objects through attributes, rather than roles (we consider here only concept attributes and not role attributes [CDGL+06a]).

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

- Concept expressions:

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

- Role expressions:

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

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

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

- Attribute expressions:

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

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

We make use of an alphabet Γ of constants, partitioned into:

- an alphabet Γ_O of object constants.
- an alphabet Γ_V of value constants, in turn partitioned into alphabets Γ_{V_i}, one for each RDF datatype T_i.

The interpretation domain $\Delta^{\mathcal{I}}$ is partitioned into:

- a domain of objects $\Delta_O^{\mathcal{I}}$
- a domain of values $\Delta_V^{\mathcal{I}}$

The semantics of *DL-Lite*$_{\mathcal{A}}$ descriptions is determined as usual, considering the following:

- The interpretation $C^{\mathcal{I}}$ of a concept C is a subset of $\Delta_O^{\mathcal{I}}$.
- The interpretation $R^{\mathcal{I}}$ of a role R is a subset of $\Delta_O^{\mathcal{I}} \times \Delta_O^{\mathcal{I}}$.
- The interpretation *val*(v) of each value constant v in Γ_V and RDF datatype T_i is given a priori (e.g., all strings for `xsd:string`).
- The interpretation $V^{\mathcal{I}}$ of an attribute V is a subset of $\Delta_O^{\mathcal{I}} \times \Delta_V^{\mathcal{I}}$.

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

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

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

TBox assertions can have the following forms:

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

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

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

As shown, to ensure FOL-rewritability, we have to impose a restriction on the use of functionality and role/attribute inclusions.

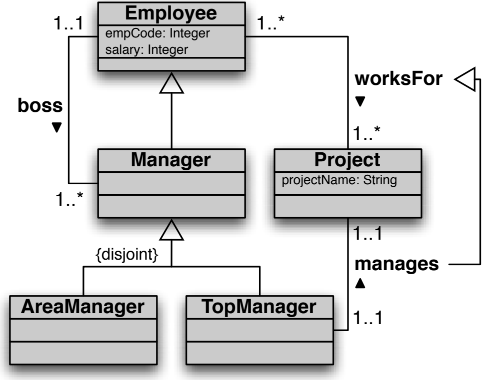
Restriction on *DL-Lite* TBoxes

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

Formally:

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

DL-Lite$_{\mathcal{A}}$ – Example



$$
\begin{array}{rcl}
\mathsf{Manager} & \sqsubseteq & \mathsf{Employee} \\
\mathsf{AreaManager} & \sqsubseteq & \mathsf{Manager} \\
\mathsf{TopManager} & \sqsubseteq & \mathsf{Manager} \\
\mathsf{AreaManager} & \sqsubseteq & \neg\mathsf{TopManager} \\
\mathsf{Employee} & \sqsubseteq & \delta(\mathsf{salary}) \\
\delta(\mathsf{salary}) & \sqsubseteq & \mathsf{Employee} \\
\rho(\mathsf{salary}) & \sqsubseteq & \mathtt{xsd:int} \\
 & (\textbf{funct}\ \mathsf{salary}) & \\
\exists\mathsf{worksFor} & \sqsubseteq & \mathsf{Employee} \\
\exists\mathsf{worksFor}^- & \sqsubseteq & \mathsf{Project} \\
\mathsf{Employee} & \sqsubseteq & \exists\mathsf{worksFor} \\
\mathsf{Project} & \sqsubseteq & \exists\mathsf{worksFor}^- \\
 & (\textbf{funct}\ \mathsf{manages}) & \\
 & (\textbf{funct}\ \mathsf{manages}^-) & \\
\mathsf{manages} & \sqsubseteq & \mathsf{worksFor} \\
 & \vdots &
\end{array}
$$

Note: in *DL-Lite*$_{\mathcal{A}}$ we still cannot capture:
– completeness of the hierarchy
– number restrictions

Reasoning in *DL-Lite*$_{\mathcal{A}}$ – 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-Lite*$_{\mathcal{F}}$ and *DL-Lite*$_{\mathcal{R}}$ continue to hold also for *DL-Lite*$_{\mathcal{A}}$.

In particular, w.r.t. satisfiability of a *DL-Lite*$_{\mathcal{A}}$ ontology $\mathcal{O}$, we have:

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

We obtain that for *DL-Lite*$_{\mathcal{A}}$ a separation result holds:

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

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 CQ q_F.

and check satisfiability of $\mathcal{O}$ by suitably evaluating q_N and q_F.

Theorem

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

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

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

- Queries over $DL\text{-}Lite_{\mathcal{A}}$ ontologies are analogous to those over $DL\text{-}Lite_{\mathcal{R}}$ and $DL\text{-}Lite_{\mathcal{F}}$ ontologies, except that they can also make use of attribute and domain atoms.
- Exploiting the previous result, the query answering algorithm of $DL\text{-}Lite_{\mathcal{R}}$ can be easily extended to deal with $DL\text{-}Lite_{\mathcal{A}}$ 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 *DL-Lite*$_{\mathcal{A}}$

As for ontology satisfiability, *DL-Lite*$_{\mathcal{A}}$ maintains the nice computational properties of *DL-Lite*$_{\mathcal{R}}$ and *DL-Lite*$_{\mathcal{F}}$ also w.r.t. query answering. Hence, we get the same characterization of computational complexity.

Theorem

For *DL-Lite*$_{\mathcal{A}}$ ontologies:

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

Outline

Managing ABoxes

In all the previous discussion, we have 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.
  - An 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 ABoxes become large, managing them in secondary storage may be required, but this is again handled directly by the reasoner.

Data in external sources

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

- When the ABox is very large, so that it requires relational database technology.
- When 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 situation by keeping the data in the external (relational) storage, and performing query answering by leveraging the capabilities of the relational engine.

The impedance mismatch problem

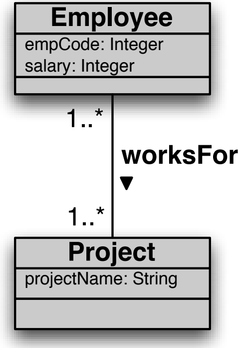
We have to deal with the impedance mismatch problem:

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

The solution is to define a mapping language that allows for specifying how to transform data into objects:

- Basic idea: use Skolem functions in the head of the mapping to "generate" the objects.
- Semantics: objects are denoted by terms (of exactly one level of nesting), and different terms denote different objects (unique name assumption on terms).

Impedance mismatch – Example



Actual data is stored in a DB:
– An Employee is identified by her *SSN*.
– A Project is identified by its *name*.

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

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

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

...

Intuitively:

- An employee should be created from her *SSN*: **pers**(*SSN*)
- A project should be created from its *Name*: **proj**(*PrName*)

Creating object identifiers

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

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

Example

- If a person is identified by its *SSN*, we can introduce a function symbol $\mathbf{pers}/1$. If $\mathtt{VRD56B25}$ is a *SSN*, then $\mathbf{pers}(\mathtt{VRD56B25})$ denotes a person.
- If a person is identified by its *name* and *dateOfBirth*, we can introduce a function symbol $\mathbf{pers}/2$. Then $\mathbf{pers}(\mathtt{Vardi}, \mathtt{25/2/56})$ 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 as object terms, but with variables instead of values as arguments of the functions.

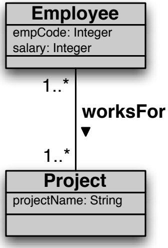
A mapping assertion between a database $\mathcal{D}$ and a TBox $\mathcal{T}$ has the form

$$\Phi \leadsto \Psi$$

where

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

Mapping assertions – Example



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

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

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

...

M_1: `SELECT SSN, PrName FROM D1` $\leadsto$ Employee(**pers**(*SSN*)), Project(**proj**(*PrName*)), projectName(**proj**(*PrName*), *PrName*), workFor(**pers**(*SSN*), **proj**(*PrName*))

M_2: `SELECT SSN, Salary FROM D2, D3 WHERE D2.CODE = D3.CODE` $\leadsto$ Employee(**pers**(*SSN*)), salary(**pers**(*SSN*), *Salary*)

Ontology-Based Data Access System

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

Ontology-Based Data Access System

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

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

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

Semantics of mappings

We first need to define the semantics of mappings.

Definition

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

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

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

Note: $\Psi[\vec{x}/\vec{v}]$ denotes Ψ where each x_i has been substituted with v_i.

Semantics of an OBDA system

Model of an OBDA system

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

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

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

Answering queries over an OBDA system

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

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

Two approaches to exploit the mapping:

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

Note: Both approaches require to first split the TBox queries in the mapping assertions into their constituent atoms. This is possible, since all variables in such queries are distinguished.

Bottom-up approach to query answering

Consists in a straightforward application of the mappings:

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

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

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

Top-down approach to query answering

Consists of three steps:

1. Reformulation: Compute the perfect reformulation $q' = \text{PerfectRef}(q, \mathcal{T}_P)$ of the original query q, using the PIs $\mathcal{T}_P$ of the TBox $\mathcal{T}$.
2. Unfolding: Compute from q' a new query q'' by unfolding q' using (the split version of) the mappings $\mathcal{M}$.
   - Essentially, each atom in q' that unifies with an atom in Ψ is substituted with the corresponding query Φ over the database.
   - The unfolded query q'' is such that $\text{Eval}(q'', \mathcal{D}) = \text{Eval}(q', \mathcal{A}_{\mathcal{M},\mathcal{D}})$.
3. Evaluation: Delegate the evaluation of q'' to the relational DBMS managing $\mathcal{D}$.

For details, see [PLC+07].

Computational complexity of query answering

Theorem

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

1. NP-complete in the size of the query.
2. PTIME in the size of the TBox $\mathcal{T}$ and the mappings $\mathcal{M}$.
3. LOGSPACE in the size of the database $\mathcal{D}$.

Moreover, the LOGSPACE result is actually a consequence of the fact that query answering in such a setting can be reduced to evaluating an SQL query over the relational database.

Part V

Hands-on session

Outline

Outline

The system MASTRO

- MASTRO is a Java-based tool for Ontology-based Data Access.
- It allows for the specification of Ontologies in the DL *DL-Lite*$_{\mathcal{A}}$.
- In MASTRO, *DL-Lite*$_{\mathcal{A}}$ TBoxes are connected to an external RDBMS trough suitable mappings.
- In each mapping a generic SQL query over the external RDBMS is put in correspondence with a CQ without existential variables expressed over the *DL-Lite*$_{\mathcal{A}}$ TBox.

Mastro vs QuOnto

- At its core, Mastro uses QuOnto (http://www.dis.uniroma1.it/~quonto/) a reasoner for *DL-Lite*$_{\mathcal{A}}$, which provides query reformulation services (QuOnto implements the algorithm PerfectRef).
- Notice that QuOnto is not designed to support ontology-based data access, and therefore it is not able to deal with mappings to external RDBMs $\leadsto$ Mastro provides this support.

Reasoning in MASTRO

- The basic services provided by MASTRO are
  - Specification of a *DL-Lite*$_{\mathcal{A}}$ OBDA System
  - Query answering, for computing certain answer for (unions of) conjunctive queries over *DL-Lite*$_{\mathcal{A}}$ OBDA Systems
  - Consistency check, for verifying satisfiability of *DL-Lite*$_{\mathcal{A}}$ OBDA Systems

  These are the only services supported by the version of MASTRO demonstrated at the tutorial.
- However, the full version of MASTRO also allows for TBox reasoning, meta-level reasoning, and ontology updates.
- We are currently working also on query answering of complex (i.e., FOL) queries, introduction on new *DL-Lite* constructs (e.g., identification assertions).

Input formats

- Mastro has its own Java-based interface, and accepts inputs in a proprietary XML format.
- That is, to give as input a TBox, an ABox, i.e., a set of mapping assertions to an external RDBMS, and a query, we must specify them into XML, according to a specific DTD.
- Nonetheless, the XML syntax to be used is very simple.

Outline

XML TBox: alphabet

```
<alphabet>
    <atomicC>professor</atomicC>
    <atomicC>assistantProf</atomicC>
             .....
    <atomicCA>name</atomicCA>
             ....
    <atomicR>WORKS_FOR</atomicR>
             ....
    <atomicRA>date</atomicRA>
</alphabet>
```

XML TBox: inclusion assertions

assistantProfessor $\sqsubseteq \neg$*fullProf*

```
<inclusionAssertion>
  <basicC>
    <atomicC>assistantProf</atomicC>
  </basicC>
     <generalC>
       <signedC sign="negative">
          <basicC>
            <atomicC>fullProf</atomicC>
          </basicC>
       </signedC>
     </generalC>
</inclusionAssertion>
```

$\exists TAKES\_COURSE^{-} \sqsubseteq course$

```
<inclusionAssertion>
  <basicC>
     <exists>
       <basicR dir="inverse">
         <atomicR>TAKES_COURSE</atomicR>
       </basicR>
     </exists>
  </basicC>
  <generalC>
     <signedC sign="positive">
        <basicC>
            <atomicC>course</atomicC>
        </basicC>
     </signedC>
  </generalC>
</inclusionAssertion>
```

XML TBox: functionality assertions

(**funct** *ENROLLED*)

```
<funct>
    <basicR dir="direct">
        <atomicR>ENROLLED</atomicR>
    </basicR>
</funct>
```

(**funct** term)

```
<funct>
    <atomicCA>term</atomicCA>
</funct>
```

XML ABox: mapping specification

```
SELECT C_Name, Term,
       Prof_Id
FROM Course_Tab
```

$\leadsto$ *course*(**course**(C_Name)),
name(**course**(C_Name), C_Name),
term(**course**(C_Name), Term),
professor(**prof**(Prof_Id)),
TEACHES(**prof**(Prof_Id), **course**(C_Name))

It is probably better to see the mapping as follows

```
SELECT C_Name, Term,
       Prof_Id
FROM Course_Tab
```

$\leadsto$ *course*(X), $X =$ **course**(C_Name),
name(X, Y), $Y =$ C_Name,
term(X, Z), $Z =$ Term,
professor(W), $W =$ **prof**(Prof_Id),
TEACHES(W, X)

XML ABox: mapping specification

```
SELECT C_Name, Term,
       Prof_Id
FROM Course_Tab
```

$\leadsto$ *course*(X), $X =$ **course**(C_Name), name(X, Y), $Y =$ C_Name, term(X, Z), $Z =$ Term, *professor*(W), $W =$ **prof**(Prof_Id), *TEACHES*(W, X)

```
<mapping>
   <head>
       <CQBody>
         ........
          <atom>
              <AtomicConceptAttribute name="term">
                   <term>
                      <var name="X"/>
                    </term>
                    <term>
                      <var name="Z"/>
                   </term>
              </AtomicConceptAttribute>
          </atom>
         ........
        </CQBody>
    </head>
         ........
```

XML ABox: mapping specification

```
SELECT C_Name, Term,
          Prof_Id
FROM Course_Tab
```

$\leadsto$ $\mathit{course}(X), X = \mathbf{course}(\texttt{C\_Name})$, $\mathrm{name}(X, Y), Y = \texttt{C\_Name}$, $\mathrm{term}(X, Z), Z = \texttt{Term}$, $\mathit{professor}(W), W = \mathbf{prof}(\texttt{Prof\_Id}), \mathit{TEACHES}(W, X)$

```
<mapping>
      ........
      <map>
         <objMap>
             <dtVar>X</dtVar>
             <sqlObjVar funct="course">C_Name</sqlObjVar>
          </objMap>
      </map>
      ........
      <map>
         <valueMap>
             <dtVar>Z</dtVar>
             <sqlValueVar type="xs:integer">Term</sqlValueVar>
             </valueMap>
      </map>
      <body>SELECT C_Name, Term, Prof_Id FROM Course_Tab</body>
      ........
</mapping>
```

Acknowledgements

- Giuseppe De Giacomo
- Enrico Franconi
- Maurizio Lenzerini
- Marco Ruzzi
- Antonella Poggi
- Riccardo Rosati
- Sergio Tessaris

References I

[ACK+07] A. Artale, D. Calvanese, R. Kontchakov, V. Ryzhikov, and M. Zakharyaschev.
Reasoning over extended ER models.
In *Proc. of the 26th Int. Conf. on Conceptual Modeling (ER 2007)*, volume 4801 of *Lecture Notes in Computer Science*, pages 277–292. Springer, 2007.

[BBL05] F. Baader, S. Brandt, and C. Lutz.
Pushing the $\mathcal{EL}$ envelope.
In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 364–369, 2005.

[BCDG05] D. Berardi, D. Calvanese, and G. De Giacomo.
Reasoning on UML class diagrams.
Artificial Intelligence, 168(1–2):70–118, 2005.

References II

[BCM+03] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors.
The Description Logic Handbook: Theory, Implementation and Applications.
Cambridge University Press, 2003.

[CDGL98] D. Calvanese, G. De Giacomo, and M. Lenzerini.
On the decidability of query containment under constraints.
In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 149–158, 1998.

[CDGL+05a] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati.
DL-Lite: Tractable description logics for ontologies.
In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, pages 602–607, 2005.

References III

[CDGL+05b] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati.
Tailoring OWL for data intensive ontologies.
In *Proc. of the Workshop on OWL: Experiences and Directions (OWLED 2005)*, 2005.

[CDGL+06a] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, and R. Rosati.
Linking data to ontologies: The description logic DL-Lite$_A$.
In *Proc. of the 2nd Workshop on OWL: Experiences and Directions (OWLED 2006)*, 2006.

[CDGL+06b] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati.
Data complexity of query answering in description logics.
In *Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 260–270, 2006.

References IV

[CDGL+07] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati.
Tractable reasoning and efficient query answering in description logics: The DL-Lite family.
J. of Automated Reasoning, 39(3):385–429, 2007.

[DLNN97] F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt.
The complexity of concept languages.
Information and Computation, 134:1–58, 1997.

[DLNS94] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf.
Deduction in concept languages: From subsumption to instance checking.
J. of Logic and Computation, 4(4):423–452, 1994.

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

References V

[GHLS07] B. Glimm, I. Horrocks, C. Lutz, and U. Sattler.
Conjunctive query answering for the description logic $\mathcal{SHIQ}$.
In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, pages 399–404, 2007.

[GHVD03] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker.
Description logic programs: Combining logic programs with description logic.
In *Proc. of the 12th Int. World Wide Web Conf. (WWW 2003)*, pages 48–57, 2003.

[HMS05] U. Hustadt, B. Motik, and U. Sattler.
Data complexity of reasoning in very expressive description logics.
In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 466–471, 2005.

References VI

[LR98] A. Y. Levy and M.-C. Rousset.
Combining Horn rules and description logics in CARIN.
Artificial Intelligence, 104(1–2):165–209, 1998.

[Lut07] C. Lutz.
Inverse roles make conjunctive queries hard.
In *Proc. of the 2007 Description Logic Workshop (DL 2007)*, 2007.

[MH03] R. Möller and V. Haarslev.
Description logic systems.
In Baader et al. [BCM+03], chapter 8, pages 282–305.

[OCE06] M. M. Ortiz, D. Calvanese, and T. Eiter.
Characterizing data complexity for conjunctive query answering in expressive description logics.
In *Proc. of the 21st Nat. Conf. on Artificial Intelligence (AAAI 2006)*, 2006.

References VII

[PLC$^+$07] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati.
Linking data to ontologies.
J. on Data Semantics, 2007.
To appear.