Knowledge Representation and Ontologies

Part 2: Description Logics

Diego Calvanese

Faculty of Computer Science
Master of Science in Computer Science

A.Y. 2011/2012

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

Part 2

Description Logics

Outline of Part 2

Outline of Part 2

1. Brief introduction to computational complexity
   - Basic definitions
   - Hardness and completeness
   - Most important complexity classes
2. Introduction to Description Logics
3. Description Logics and UML Class Diagrams
4. References

Outline of Part 2

Computational complexity (1/2)

[J.E. Hopcroft, 2007; Papadimitriou, 1994]

Computational complexity theory aims at understanding how difficult it is to solve specific problems.

- A **problem** is considered as an (in general infinite) set of instances of the problem, each encoded in some meaningful (i.e., compact) way.
- Standard complexity theory deals with **decision problems**: i.e., problems that admit a yes/no answer.
- **Algorithm** that solves a decision problem:
  - input: an instance of the problem
  - output: yes or no
- The difficulty (complexity) is measured in terms of the amount of **resources** (time, space) that the algorithm needs to solve the problem.
  $\leadsto$ complexity of the algorithm, or **upper bound**
- To measure the complexity of the problem, we consider the best possible algorithm that solves it.
  $\leadsto$ **lower bound**

Computational complexity (2/2)

- **Worst-case** complexity analysis: the complexity is measured in terms of a (complexity) function f:
  - argument: the size n of an instance of the problem (i.e., the length of its encoding)
  - result: the amount $f(n)$ of time/space needed in the worst-case to solve an instance of size n
- The **asymptotic behaviour** of the complexity function when n grows is considered.
- To abstract away from contingent issues (e.g., programming language, processor speed, etc.), we refer to an abstract computing model: **Turing Machines** (TMs).

Complexity classes

To achieve robustness wrt encoding issues, usually one does not consider specific complexity functions f, but rather families $\mathcal{C}$ of complexity functions, giving rise to complexity classes.

Def.: A time/space complexity class $\mathcal{C}$

... is the set of all problems P such that an instance of P of size n can be solved in time/space at most $C(n)$.

Note: Consider a (decision) problem P, and an encoding of the instances of P into strings over some alphabet Σ.
Once we fix such an encoding, the problem actually corresponds to a language L_P, namely the set of strings encoding those instances of the problem for which the answer is yes.

Hence, in the technical sense, a complexity class is actually a set of languages.

Outline of Part 2

Reductions

To establish lower bounds on the complexity of problems, we make use of the notion of reduction:

Def.: A reduction from a problem P_1 to a problem P_2

... is a function R (the reduction) from instance of P_1 to instances of P_2 such that:

1. R is efficiently computable (i.e., in logarithmic space), and
2. An instance I of P_1 has answer yes iff $R(I)$ has answer yes.

P_1 **reduces to** P_2 if there is a reduction R from P_1 to P_2.

Intuition: If P_1 reduces to P_2, then P_2 is at least as difficult as P_1, since we can solve an instance I of P_1 by reducing it to the instance $R(I)$ of P_2 and then solve $R(I)$.

Hardness and completeness

Def.: A problem P is hard for a complexity class $\mathcal{C}$

. . . if every problem in $\mathcal{C}$ can be reduced to P.

Def.: A problem P is complete for a complexity class $\mathcal{C}$ if

1. it is hard for $\mathcal{C}$, and
2. it belongs to $\mathcal{C}$

Intuitively, a problem that is complete for $\mathcal{C}$ is among the hardest problems in $\mathcal{C}$.

Outline of Part 2

Tractability and intractability: PTime and NP

Def.: PTime

Set of problems solvable in polynomial time by a deterministic TM.

- These problems are considered **tractable**, i.e., solvable for large inputs.
- Is a robust class (PTime computations compose).

Def.: NP

Set of problems solvable in polynomial time by a non-deterministic TM.

- These problems are believed **intractable**, i.e., unsolvable for large inputs.
- The best known algorithms actually require exponential time.
- Corresponds to a large class of practical problems, for which the following type of algorithm can be used:
  1. Non-deterministically guess a possible solution of polynomial size.
  2. Check in polynomial time that the guessed solutions is good.

Complexity classes above NP

Def.: PSpace

Set of problems solvable in polynomial space by a deterministic TM.

- Polynomial space is "not really good", since these problems may require exponential time.
- These problems are considered to be more difficult than NP problems.
- Practical algorithms and heuristics work less well than for NP problems.

Def.: ExpTime

Set of problems solvable in exponential time by a deterministic TM.

- This is the first provably intractable complexity class.
- These problems are considered to be very difficult.

Def.: NExpTime

Set of problems solvable in exponential time by a non-deterministic TM.

Complexity classes below PTIME

Def.: LOGSPACE and NLOGSPACE

Set of problems solvable in logarithmic space by a (non-)deterministic TM.

- Note: when measuring the space complexity, the size of the input does not count, and only the working memory (TM tape) is considered.
- Note 2: logarithmic space computations compose (this is not trivial).
- Correspond to reachability in undirected and directed graphs, respectively.

Def.: AC^0

Set of problems solvable in constant time using a polynomial number of processors.

- These problems are solvable efficiently even for very large inputs.
- Corresponds to the complexity of model checking a fixed FO formula when the input is the model only.

Relationship between the complexity classes

The following relationships are known:

$$\mathrm{AC}^0 \subsetneq \textsc{LogSpace} \subseteq \textsc{NLogSpace} \subseteq \textsc{PTime} \subseteq$$
$$\subseteq \textsc{NP} \subseteq \textsc{PSpace} \subseteq$$
$$\subseteq \textsc{ExpTime} \subseteq \textsc{NExpTime}$$

Moreover, we know that $\textsc{PTime} \subsetneq \textsc{ExpTime}$.

Outline of Part 2

1. Brief introduction to computational complexity
2. Introduction to Description Logics
   - Ingredients of Description Logics
   - Description Logics ontologies
   - Reasoning in Description Logics
   - Relationship between DLs and other representation formalisms
3. Description Logics and UML Class Diagrams
4. References

Outline of Part 2

What are Description Logics?

Description Logics (DLs) [Baader *et al.*, 2003] are **logics** specifically designed to represent and reason on structured knowledge.

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

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

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

Origins of Description Logics

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.

What are Description Logics about?

Abstractly, DLs allow one to predicate about **labeled directed graphs**:

- Vertexes represents real world objects.
- Vertexes's labels represents qualities of objects.
- Edges represents relations between (pairs of) objects.
- Vertexes' labels represents the types of relations between objects.

Every fragment of the world that can be abstractly represented in terms of a labeled directed graph is a good candidate for being represented by DLs.

What are Description Logics about? – Example 1

Exercise

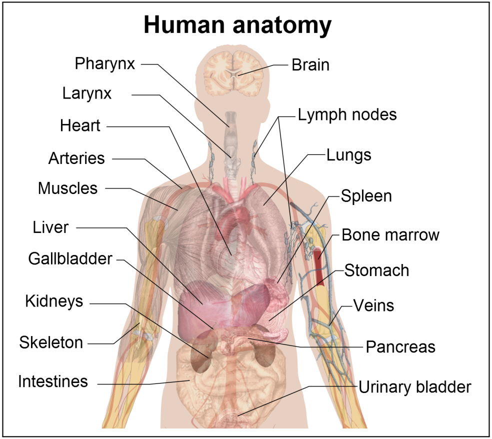
Represent Metro lines in Milan in a labelled directed graph.

What are Description Logics about? – Example 2

Exercise

Represent some aspects of Facebook as a labelled directed graph.

What are Description Logics about? – Example 3



Exercise

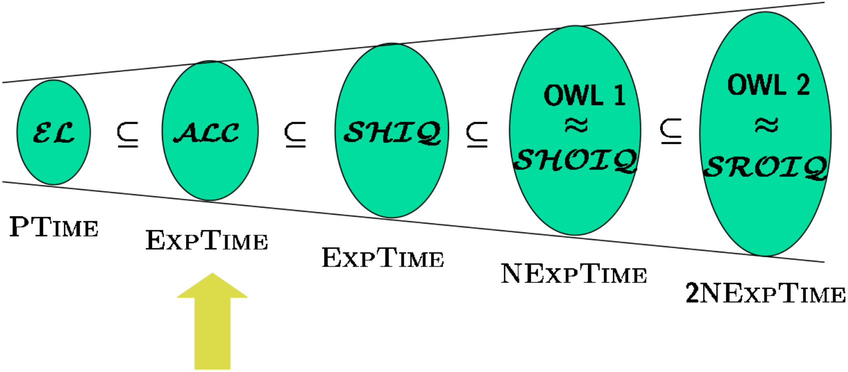
Represent some aspects of human anatomy as a labelled directed graph.

What are Description Logics about? – Example 4

Exercise

Represent some aspects of document classification as a labelled directed graph.

Many description logics



Ingredients of a Description Logic

A **DL** is characterized by:

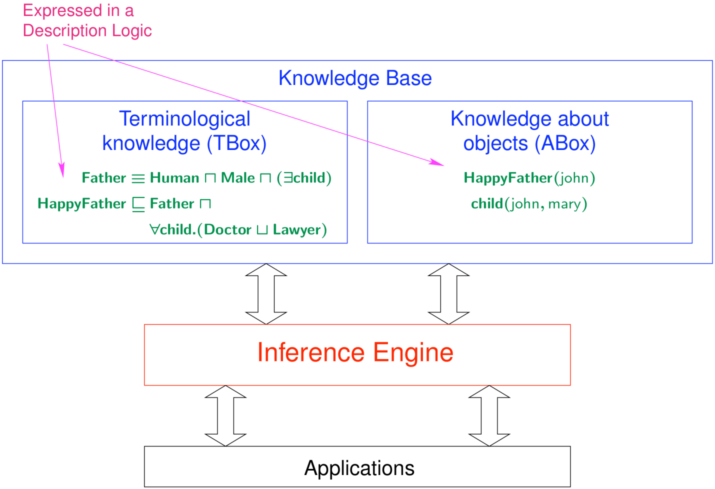
1. A **description language**: how to form concepts and roles
   $\text{Human} \sqcap \text{Male} \sqcap \exists\text{hasChild} \sqcap \forall\text{hasChild}.(\text{Doctor} \sqcup \text{Lawyer})$

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

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

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

Architecture of a Description Logic system



Description language

A description language provides the means for defining:

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

To define concepts and roles:

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

A **description language** is characterized by the set of constructs that are available for that.

Formal semantics of a description language

The **formal semantics** of DLs is given in terms of interpretations.

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

- a nonempty set $\Delta^{\mathcal{I}}$, called the interpretation domain (of $\mathcal{I}$)
- an interpretation function $\cdot^{\mathcal{I}}$, which maps
  - 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	$\{ o \mid \exists o'. (o, o') \in R^{\mathcal{I}} \}$
value restriction	$\forall R.C$	∀hasChild.Male	$\{o \mid \forall o'. (o, o') \in R^{\mathcal{I}} \rightarrow o' \in C^{\mathcal{I}}\}$
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.	$\mathcal{E}$	$\exists R.C$	$\{\, o \mid \exists o'.\, (o,o') \in R^{\mathcal{I}} \land o' \in C^{\mathcal{I}} \,\}$
(full) negation	$\mathcal{C}$	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
number restrictions	$\mathcal{N}$	$(\geq k\, R)$	$\{\, o \mid \#\{o' \mid (o,o') \in R^{\mathcal{I}}\} \geq k \,\}$
		$(\leq k\, R)$	$\{\, o \mid \#\{o' \mid (o,o') \in R^{\mathcal{I}}\} \leq k \,\}$
qual. number restrictions	$\mathcal{Q}$	$(\geq k\, R.C)$	$\{\, o \mid \#\{o' \mid (o,o') \in R^{\mathcal{I}} \land o' \in C^{\mathcal{I}}\} \geq k \,\}$
		$(\leq k\, R.C)$	$\{\, o \mid \#\{o' \mid (o,o') \in R^{\mathcal{I}} \land o' \in C^{\mathcal{I}}\} \leq k \,\}$
inverse role	$\mathcal{I}$	R^{-}	$\{\, (o,o') \mid (o',o) \in R^{\mathcal{I}} \,\}$
role closure	reg	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: $\exists \mathsf{hasChild}^{*}.\mathsf{Doctor}$

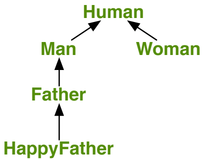
Reasoning on concept expressions

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 is used to build the concept hierarchy:



Exercise

Show that if DL is propositionally closed then (1) and (2) are mutually reducible.

Complexity of reasoning on concept expressions

Complexity of concept satisfiability: [Donini *et al.*, 1997]

$\mathcal{AL}$, $\mathcal{ALN}$	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.

Outline of Part 2

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**:

Def.: 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) $\cdots$

Def.: Description Logics ABox

Consists of a set of **assertions** on individuals: (we use c_i to denote individuals)

- Membership assertions for concepts: $A(c)$
- Membership assertions for roles: $P(c_1, c_2)$
- Equality and distinctness assertions: $c_1 \approx c_2$, $c_1 \not\approx c_2$

Description Logics ontology – 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:

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

- Inclusion assertions on roles:

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

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

ABox membership assertions:

- $\text{Teacher}(\texttt{mary})$, $\text{hasFather}(\texttt{mary}, \texttt{john})$, $\text{HappyAnc}(\texttt{john})$

unibz.it

Semantics of a Description Logics ontology

The semantics is given by specifying when an interpretation $\mathcal{I}$ **satisfies** an assertion α, denoted $\mathcal{I} \models \alpha$.

TBox Assertions:

- $\mathcal{I} \models C_1 \sqsubseteq C_2$ if $C_1^\mathcal{I} \subseteq C_2^\mathcal{I}$.
- $\mathcal{I} \models R_1 \sqsubseteq R_2$ if $R_1^\mathcal{I} \subseteq R_2^\mathcal{I}$.
- $\mathcal{I} \models (\textbf{prop } P)$ 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.)

ABox Assertions:

We need first to extend the interpretation function $\cdot^\mathcal{I}$, so that it maps each individual c to an element $c^\mathcal{I}$ of $\Delta^\mathcal{I}$.

- $\mathcal{I} \models A(c)$ if $c^\mathcal{I} \in A^\mathcal{I}$.
- $\mathcal{I} \models P(c_1, c_2)$ if $(c_1^\mathcal{I}, c_2^\mathcal{I}) \in P^\mathcal{I}$.
- $\mathcal{I} \models c_1 \approx c_2$ if $c_1^\mathcal{I} = c_2^\mathcal{I}$.
- $\mathcal{I} \models c_1 \not\approx c_2$ if $c_1^\mathcal{I} \neq c_2^\mathcal{I}$.

Model of a Description Logics ontology

Def.: Model

An interpretation $\mathcal{I}$ is a **model** of:

- an assertion α, if it satisfies α.
- a TBox $\mathcal{T}$, if it satisfies all assertions in $\mathcal{T}$.
- an ABox $\mathcal{A}$, if it satisfies all assertions in $\mathcal{A}$.
- an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, if it is a model of both $\mathcal{T}$ and $\mathcal{A}$.

Note: We use $\mathcal{I} \models \beta$ to denote that interpretation $\mathcal{I}$ is a **model** of β (where β stands for an assertion, TBox, ABox, or ontology).

Interpretation of individuals

We may make some assumptions on how individuals are interpreted.

Unique name assumption (UNA)

When c_1 and c_2 are two individuals such that $c_1 \neq c_2$, then $c_1^{\mathcal{I}} \neq c_2^{\mathcal{I}}$.

Note: When the UNA holds, equality and distinctness assertions are meaningless.

Standard name assumption (SNA)

The UNA holds, and moreover individuals are interpreted in the same way in all interpretations.
Hence, we may assume that $\Delta^{\mathcal{I}}$ contains the set of individuals, and that for each interpretation $\mathcal{I}$, we have that $c^{\mathcal{I}} = c$ (then, c is called a **standard name**).

Outline of Part 2

Logical implication

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

Def.: Logical implication

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

We can provide an analogous definition for a TBox $\mathcal{T}$ instead of an ontology $\mathcal{O}$.

TBox reasoning

- **TBox Satisfiability:** $\mathcal{T}$ is satisfiable, if it admits at least one model.
- **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 every model of $\mathcal{O}$, 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 every model of $\mathcal{O}$, i.e., whether $\mathcal{O} \models R(c_1, c_2)$.
- **Query Answering:** see later ...

Reasoning in Description Logics – Example

TBox:

- Inclusion assertions on concepts:

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

- Inclusion assertions on roles:

$\text{hasChild} \sqsubseteq \text{descendant}$ $\quad\quad$ $\text{hasFather} \sqsubseteq \text{hasChild}^-$

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

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

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

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

Relationship among TBox reasoning tasks

The TBox reasoning tasks are mutually **reducible** to each other, provided the description language is propositionally closed:

TBox satisfiability to concept satisfiability to concept non-subsumption

$\mathcal{T}$ satisfiable iff $\mathcal{T} \not\models \top \equiv \bot$ (i.e., $\top$ satisfiable w.r.t. $\mathcal{T}$) iff not $\mathcal{T} \models \top \sqsubseteq \bot$

Concept subsumption to concept unsatisfiability

$\mathcal{T} \models C_1 \sqsubseteq C_2$ iff $\mathcal{T} \models C_1 \sqcap \neg C_2 \equiv \bot$
(i.e., $C_1 \sqcap \neg C_2$ unsatisfiable w.r.t. $\mathcal{T}$)

Concept satisfiability to TBox satisfiability

$\mathcal{T} \not\models C \equiv \bot$ iff $\mathcal{T} \cup \{ \top \sqsubseteq \exists P_{new} \sqcap \forall P_{new}.C \}$ satisfiable
(where P_{new} is a new atomic role)

Relationship among reasoning tasks

TBox reasoning can be reduced to reasoning over an ontology:

Concept satisfiability to ontology satisfiability

C satisfiable wrt $\mathcal{T}$ iff $\langle \mathcal{T} \cup \{A_{new} \sqsubseteq C\},\ \{A(c_{new})\}\rangle$ is satisfiable
(where A_{new} is a new atomic concept and c_{new} is a new individual)

Exercise

Show mutual reductions between the remaining (TBox and ontology) reasoning tasks.

Internalization of the TBox:

- In some (very expressive) DLs, it is possible to reduce reasoning wrt a TBox to reasoning over concept expressions only, i.e., the whole TBox can be internalized into a single concept.
- Whether this is possible depends on the available role and concept constructors, and the details differ for each DL.

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., [Donini, 2003]).

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++, . . .) [Möller and Haarslev, 2003].

Outline of Part 2

1. Brief introduction to computational complexity
2. Introduction to Description Logics
   - Ingredients of Description Logics
   - Description Logics ontologies
   - Reasoning in Description Logics
   - Relationship between DLs and other representation formalisms
3. Description Logics and UML Class Diagrams
4. References

Relationship with First Order Logic

Most DLs are well-behaved fragments of First Order Logic.

To translate an $\mathcal{ALC}$ TBox to FOL:

1. Introduce: a unary predicate $A(x)$ for each atomic concept A
   a binary predicate $P(x, y)$ for each atomic role P
2. Translate complex concepts as follows, using translation functions t_x, one for each variable x:

$$t_x(A) = A(x) \qquad t_x(C \sqcap D) = t_x(C) \wedge t_x(D)$$
$$t_x(\neg C) = \neg t_x(C) \qquad t_x(C \sqcup D) = t_x(C) \vee t_x(D)$$
$$t_x(\exists P.C) = \exists y.\, P(x, y) \wedge t_y(C)$$
$$t_x(\forall P.C) = \forall y.\, P(x, y) \rightarrow t_y(C)$$

(with y a new variable)

3. Translate a TBox $\mathcal{T} = \bigcup_i \{ C_i \sqsubseteq D_i \}$ as the FOL theory:

$$\Gamma_{\mathcal{T}} = \bigcup_i \{ \forall x.\, t_x(C_i) \rightarrow t_x(D_i) \}$$

4. Translate an ABox $\mathcal{A} = \bigcup_i \{ A_i(c_i) \} \cup \bigcup_j \{ P_j(c'_j, c''_j) \}$ as the FOL th.:

$$\Gamma_{\mathcal{A}} = \bigcup_i \{ A_i(c_i) \} \cup \bigcup_j \{ P_j(c'_j, c''_j) \}$$

unibz.it

Relationship with First Order Logic – Reasoning

There is a direct correspondence between DL reasoning services and FOL reasoning services:

C is satisfiable	iff	its translation $t_x(C)$ is satisfiable
$C \sqsubseteq D$	iff	$t_x(C) \rightarrow t_x(D)$ is valid
C is satisfiable w.r.t. $\mathcal{T}$	iff	$\Gamma_{\mathcal{T}} \cup \{ \exists x. t_x(C) \}$ is satisfiable
$\mathcal{T} \models C \sqsubseteq D$	iff	$\Gamma_{\mathcal{T}} \models \forall x. (t_x(C) \rightarrow t_x(D))$

Relationship with First Order Logic – Exercise

Translate the following $\mathcal{ALC}$ concepts into FOL formulas:

1. Father $\sqcap$ $\forall$child.(Doctor $\sqcup$ Manager)
2. $\exists$manages.(Company $\sqcap$ $\exists$employs.Doctor)
3. Father $\sqcap$ $\forall$child.(Doctor $\sqcup$ $\exists$manages.(Company $\sqcap$ $\exists$employs.Doctor))

Solution:

1. $\mathsf{Father}(x) \wedge \forall y.\,(\mathsf{child}(x,y) \rightarrow (\mathsf{Doctor}(y) \vee \mathsf{Manager}(y)))$
2. $\exists y.\,(\mathsf{manages}(x,y) \wedge (\mathsf{Company}(y) \wedge \exists w.\,(\mathsf{employs}(y,w) \wedge \mathsf{Doctor}(w))))$
3. $\mathsf{Father}(x) \wedge \forall y.\,(\mathsf{child}(x,y) \rightarrow (\mathsf{Doctor}(y) \vee$
   $\exists w.\,(\mathsf{manages}(y,w) \wedge (\mathsf{Company}(w) \wedge \exists z.\,(\mathsf{employs}(w,z) \wedge \mathsf{Doctor}(z))))))$

DLs as fragments of First Order Logic

The previous translation shows us that DLs are a fragment of First Order Logic.

In particular, we can translate complex concepts using just two translation functions t_x and t_y (thus **reusing the same variables**):

$$\begin{aligned} t_x(A) &= A(x) \\ t_x(\neg C) &= \neg C(x) \\ t_x(C \sqcap D) &= t_x(C) \wedge t_x(D) \\ t_x(C \sqcup D) &= t_x(C) \vee t_x(D) \\ t_x(\exists P.C) &= \exists y.\, P(x,y) \wedge t_y(C) \\ t_x(\forall P.C) &= \forall y.\, P(x,y) \rightarrow t_y(C) \end{aligned}$$

$$\begin{aligned} t_y(A) &= A(y) \\ t_y(\neg C) &= \neg C(y) \\ t_y(C \sqcap D) &= t_y(C) \wedge t_y(D) \\ t_y(C \sqcup D) &= t_y(C) \vee t_y(D) \\ t_y(\exists P.C) &= \exists x.\, P(y,x) \wedge t_x(C) \\ t_y(\forall P.C) &= \forall x.\, P(y,x) \rightarrow t_x(C) \end{aligned}$$

$\leadsto$ $\mathcal{ALC}$ is a fragment of L2, i.e., FOL with 2 variables, known to be decidable (NExpTime-complete).

Note: FOL with 2 variables is more expressive than $\mathcal{ALC}$ (tradeoff expressive power vs. complexity of reasoning).

DLs as fragments of First Order Logic – Exercise

Translate the following $\mathcal{ALC}$ concepts into L2 formulas (i.e., into FOL formulas that use only variables x and y):

1. Father $\sqcap$ $\forall$child.(Doctor $\sqcup$ Manager)
2. $\exists$manages.(Company $\sqcap$ $\exists$employs.Doctor)
3. Father $\sqcap$ $\forall$child.(Doctor $\sqcup$ $\exists$manages.(Company $\sqcap$ $\exists$employs.Doctor))

Solution:

1. $\text{Father}(x) \land \forall y. (\text{child}(x, y) \rightarrow (\text{Doctor}(y) \lor \text{Manager}(y)))$
2. $\exists y. (\text{manages}(x, y) \land (\text{Company}(y) \land \exists x. (\text{employs}(y, x) \land \text{Doctor}(x))))$
3. $\text{Father}(x) \land \forall y. (\text{child}(x, y) \rightarrow (\text{Doctor}(y) \lor$
   $\exists x. (\text{manages}(y, x) \land (\text{Company}(x) \land \exists y. (\text{employs}(x, y) \land \text{Doctor}(y))))))$

DLs as fragments of First Order Logic (Cont'd)

The previous translations can be extended to other constructs:

- For **inverse roles**, swap the variables in the role predicate, i.e.,

$$t_x(\exists P^-.C) = \exists y.\, P(y,x) \wedge t_y(C) \qquad \text{with } y \text{ a new variable}$$
$$t_x(\forall P^-.C) = \forall y.\, P(y,x) \rightarrow t_y(C) \qquad \text{with } y \text{ a new variable}$$

  $\leadsto$ $\mathcal{ALCI}$ is still a fragment of L2

- For **number restrictions**, two variables do not suffice;
  but, $\mathcal{ALCQI}$ is a fragment of C2 (i.e, L2+counting quantifiers)

Relationship with Modal Logics

$\mathcal{ALC}$ is a syntactic variant of $\mathbf{K}_m$ (i.e., multi-modal $\mathbf{K}$):

$$\begin{array}{rcl} C \sqcap D & \Leftrightarrow & C \wedge D \\ C \sqcup D & \Leftrightarrow & C \vee D \\ \neg C & \Leftrightarrow & \neg C \end{array} \qquad \begin{array}{rcl} \exists P.C & \Leftrightarrow & \Diamond_P\, C \\ \forall P.C & \Leftrightarrow & \Box_P\, C \end{array}$$

- no correspondence for inverse roles
- no correspondence for number restrictions

$\leadsto$ **Concept consistency, subsumption in $\mathcal{ALC}$ $\Leftrightarrow$ Satisfiability, validity in $\mathbf{K}_m$**

To encode inclusion assertions, *axioms* are used
$\leadsto$ Logical implication in DLs corresponds to "global logical implication" in Modal Logics

Relationship between DLs and ontology formalisms

- DLs are nowadays advocated to provide the foundations for ontology languages.
- Different versions of the W3C standard **Web Ontology Language (OWL)** have been defined as syntactic variants of certain DLs.
- DLs are also ideally suited to capture the fundamental features of conceptual modeling formalism 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 the correspondence with OWL, highlighting essential DL constructs.
- We will come back a bit later to the correspondence between UML Class Diagrams and DLs.

DLs vs. OWL

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

- **OWL1 Lite** is a variant of the DL $\mathcal{SHIF}(D)$, 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,
  - $\mathcal{F}$ stands for functionality of roles,
  - (D) stand for data types, which are necessary in any practical knowledge representation language.
- **OWL1 DL** is a variant of $\mathcal{SHOIN}(D)$, where:
  - $\mathcal{O}$ stands for nominals, which means the possibility of using individuals in the TBox (i.e., the intensional part of the ontology),
  - $\mathcal{N}$ stands for (unqualified) number restrictions.

DLs vs. OWL2

A new version of OWL, **OWL2**, is currently being standardized by the W3C:

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

- OWL2 defines also three **profiles**: OWL2 QL, OWL2 EL, OWL2 RL.
  - Each profile corresponds to a syntactic fragment (i.e., a sub-language) of OWL2 DL that is targeted towards a specific use.
  - The restrictions in each profile guarantee better computational properties than those of OWL2 DL.
  - The **OWL2 QL** profile is derived from the DLs of the *DL-Lite* family (see later).

DL constructs vs. OWL constructs

OWL contructor	DL constructor	Example
`ObjectIntersectionOf`	$C_1 \sqcap \cdots \sqcap C_n$	Human $\sqcap$ Male
`ObjectUnionOf`	$C_1 \sqcup \cdots \sqcup C_n$	Doctor $\sqcup$ Lawyer
`ObjectComplementOf`	$\neg C$	$\neg$Male
`ObjectOneOf`	$\{a_1\} \sqcup \cdots \sqcup \{a_n\}$	{john} $\sqcup$ {mary}
`ObjectAllValuesFrom`	$\forall P.C$	$\forall$hasChild.Doctor
`ObjectSomeValuesFrom`	$\exists P.C$	$\exists$hasChild.Lawyer
`ObjectMaxCardinality`	$(\leq n\, P)$	($\leq$ 1 hasChild)
`ObjectMinCardinality`	$(\geq n\, P)$	($\geq$ 2 hasChild)

...

Note: all constructs come also in the `Data...` instead of `Object...` variant.

unibz.it

DL axioms vs. OWL axioms

OWL axiom	DL syntax	Example
SubClassOf	$C_1 \sqsubseteq C_2$	Human $\sqsubseteq$ Animal $\sqcap$ Biped
EquivalentClasses	$C_1 \equiv C_2$	Man $\equiv$ Human $\sqcap$ Male
DisjointClasseses	$C_1 \sqsubseteq \neg C_2$	Man $\sqsubseteq$ $\neg$Female
SameIndividual	$\{a_1\} \equiv \{a_2\}$	{presBush} $\equiv$ {G.W.Bush}
DifferentIndividuals	$\{a_1\} \sqsubseteq \neg\{a_2\}$	{john} $\sqsubseteq$ $\neg${peter}
SubObjectPropertyOf	$P_1 \sqsubseteq P_2$	hasDaughter $\sqsubseteq$ hasChild
EquivalentObjectProperties	$P_1 \equiv P_2$	hasCost $\equiv$ hasPrice
InverseObjectProperties	$P_1 \equiv P_2^-$	hasChild $\equiv$ hasParent$^-$
TransitiveObjectProperty	$P^+ \sqsubseteq P$	ancestor$^+$ $\sqsubseteq$ ancestor
FunctionalObjectProperty	$\top \sqsubseteq (\leq 1\, P)$	$\top \sqsubseteq (\leq 1$ hasFather$)$

...

Outline of Part 2

1. Brief introduction to computational complexity
2. Introduction to Description Logics
3. Description Logics and UML Class Diagrams
   - UML Class Diagrams as ontology formalisms
   - Reducing reasoning in UML to reasoning in DLs
   - Reducing reasoning in DLs to reasoning in UML
   - Reasoning on UML Class Diagrams
4. References

Outline of Part 2

Reasoning on UML Class Diagrams

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

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

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

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

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

DLs vs. UML Class Diagrams

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

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

Outline of Part 2

Encoding UML Class Diagrams in DLs

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

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

Encoding of classes and attributes

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

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

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

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

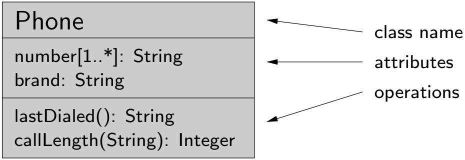
    - When m is 0, we omit the first conjunct.
    - When n is $*$, we omit the second conjunct.
    - When the multiplicity is $[0..*]$ we omit the whole assertion.
    - When the multiplicity is missing (i.e., $[1..1]$), the assertion becomes:

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

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

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

Encoding of classes and attributes – Example



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

$$\exists \mathsf{number} \sqsubseteq \mathsf{Phone} \qquad \exists \mathsf{number}^- \sqsubseteq \mathsf{String}$$
$$\exists \mathsf{brand} \sqsubseteq \mathsf{Phone} \qquad \exists \mathsf{brand}^- \sqsubseteq \mathsf{String}$$

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

$$\mathsf{Phone} \sqsubseteq \exists \mathsf{number}$$
$$\mathsf{Phone} \sqsubseteq \exists \mathsf{brand} \sqcap (\leq 1\, \mathsf{brand})$$

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

Encoding of associations

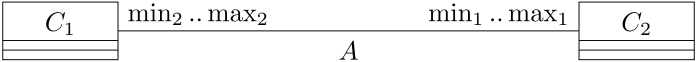
The encoding of associations depends on:

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

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

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

Encoding of binary associations without association class



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

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

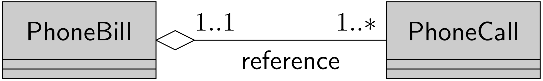
- To encode the multiplicities of A:
  - each instance of class C_1 is connected through association A to at least $\min_1$ and at most $\max_1$ instances of C_2:

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

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

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

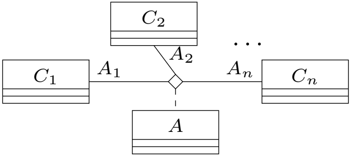
Binary associations without association class – Example

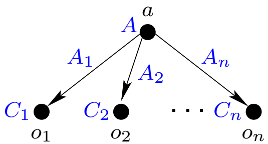


$$
\begin{array}{rcl}
\exists \mathsf{reference} & \sqsubseteq & \mathsf{PhoneBill} \\
\exists \mathsf{reference}^- & \sqsubseteq & \mathsf{PhoneCall} \\
\mathsf{PhoneBill} & \sqsubseteq & (\geq 1\, \mathsf{reference}) \\
\mathsf{PhoneCall} & \sqsubseteq & (\geq 1\, \mathsf{reference}^-) \sqcap (\leq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

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

Encoding of associations via reification





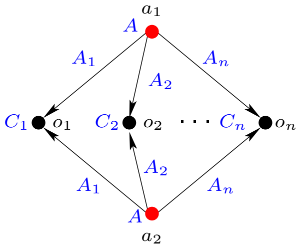
- An association A is represented by a concept A.
- Each instance a of A represents an instance $(o_1, \ldots, o_n)$ of the association.
- n (binary) roles $A_1, \ldots, A_n$ are used to connect an object a representing a tuple to objects $o_1, \ldots, o_n$ representing the components of the tuple.
-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

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

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

Encoding of associations via reification

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



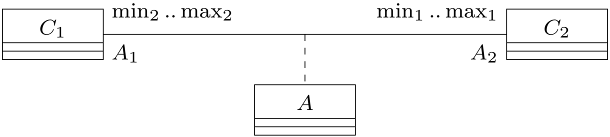
To rule out such a situation we could add an identification assertion (see later):

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

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

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

Multiplicities of binary associations with association class



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

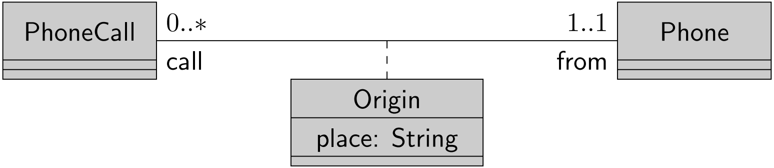
- each instance of class C_1 is connected through association A to at least $\min_1$ and at most $\max_1$ instances of C_2:

$$C_1 \sqsubseteq (\geq \min_1 \ A_1^-) \sqcap (\leq \max_1 \ A_1^-)$$

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

$$C_2 \sqsubseteq (\geq \min_2 \ A_2^-) \sqcap (\leq \max_2 \ A_2^-)$$

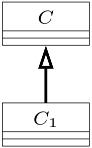
Associations with association class – Example



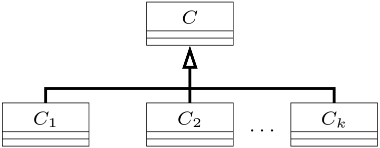
$$
\begin{array}{rcl}
\exists \mathsf{place} & \sqsubseteq & \mathsf{Origin} \\
\mathsf{Origin} & \sqsubseteq & \exists \mathsf{place} \sqcap (\leq 1\, \mathsf{place}) \\
\exists \mathsf{call} & \sqsubseteq & \mathsf{Origin} \\
\exists \mathsf{from} & \sqsubseteq & \mathsf{Origin} \\
\mathsf{Origin}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

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

Encoding of ISA and generalization



$$C_1 \sqsubseteq C$$



$$C_1 \sqsubseteq C$$
$$\vdots$$
$$C_k \sqsubseteq C$$

- When the generalization is **disjoint**:

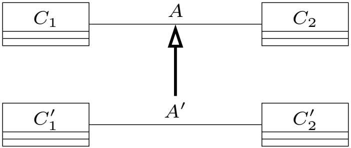
$$C_i \sqsubseteq \neg C_j \qquad \text{for } 1 \leq i < j \leq k$$

- When the generalization is **complete**:

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

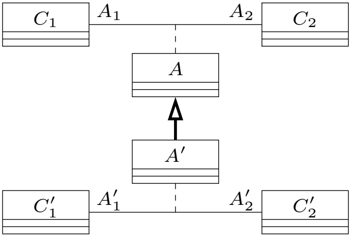
Encoding of ISA between associations

- Without reification:



Role inclusion assertion: $A' \sqsubseteq A$

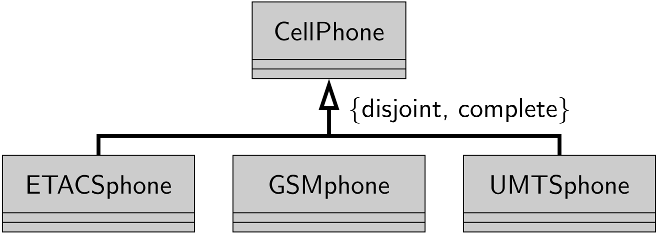
- With reification:



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

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

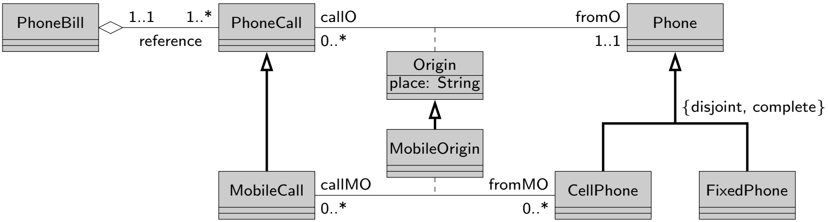
ISA and generalization – Example



$$\begin{array}{rcl} \text{ETACSphone} & \sqsubseteq & \text{CellPhone} \\ \text{GSMSphone} & \sqsubseteq & \text{CellPhone} \\ \text{UMTSSphone} & \sqsubseteq & \text{CellPhone} \end{array} \qquad \begin{array}{rcl} \text{ETACSphone} & \sqsubseteq & \neg\text{GSMPhone} \\ \text{ETACSphone} & \sqsubseteq & \neg\text{UMTSPhone} \\ \text{GSMphone} & \sqsubseteq & \neg\text{UMTSPhone} \end{array}$$

$$\text{CellPhone} \sqsubseteq \text{ETACSphone} \sqcup \text{GSMphone} \sqcup \text{UMTSPhone}$$

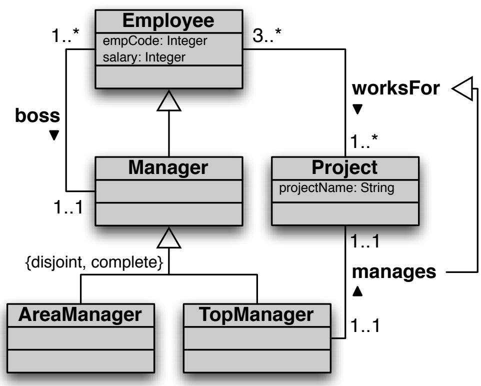
Encoding UML Class Diagrams in DLs – Example



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

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

Encoding UML Class Diagrams in DLs – Example 2



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

Outline of Part 2

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

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

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

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

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

Encoding DL TBoxes in UML Class Diagrams

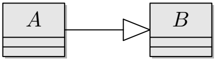
Given a primitive $\mathcal{ALC}^-$ TBox $\mathcal{T}$, we construct $\mathcal{D}_\mathcal{T}$ as follows:

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

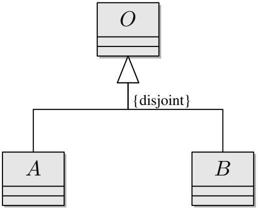
$$\begin{array}{lll} A & \sqsubseteq & B \\ A & \sqsubseteq & \neg B \\ A & \sqsubseteq & B_1 \sqcup B_2 \end{array} \qquad \begin{array}{lll} A & \sqsubseteq & \exists P.B \\ A & \sqsubseteq & \forall P.B \end{array}$$

Encoding of inclusion and of disjointness

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

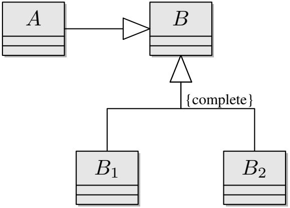


For each assertion $A \sqsubseteq \neg B$ of $\mathcal{T}$, add the following to $\mathcal{D}_\mathcal{T}$:



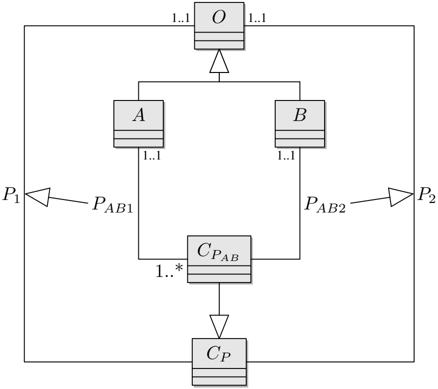
Encoding of union

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



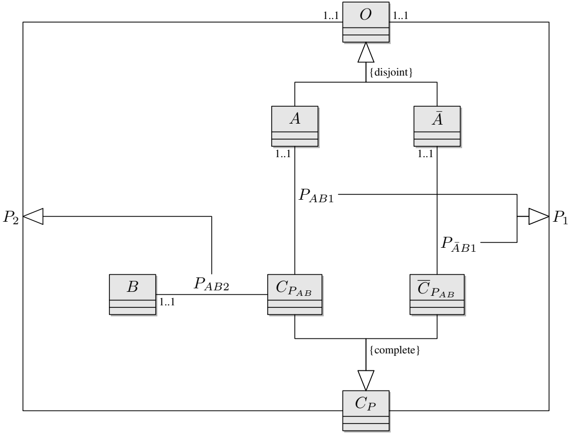
Encoding of existential quantification

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



Encoding of universal quantification

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



Complexity of reasoning on UML Class Diagrams

Lemma

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

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

Theorem

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

Outline of Part 2

Reasoning on UML Class Diagrams using DLs

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

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

- NLogSpace-complete in combined complexity
- in LogSpace in data complexity (see later)

Efficient reasoning on UML Class Diagrams

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

Questions

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

References I

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

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

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

[Donini *et al.*, 1997] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Werner Nutt.
The complexity of concept languages.
Information and Computation, 134:1–58, 1997.

References II

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

[J.E. Hopcroft, 2007] J.D. Ullman J.E. Hopcroft, R. Motwani.
Introduction to Automata Theory, Languages, and Computation.
Addison Wesley Publ. Co., 3 edition, 2007.

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

[Papadimitriou, 1994] Christos H. Papadimitriou.
Computational Complexity.
Addison Wesley Publ. Co., 1994.