Outline

Outline

Outline

Outline

Relationship with First Order Logic

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

Definition (From $\mathcal{ALC}$ TBox to FOL)

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 using translation functions t_x, one for each variable x:

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

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

Relationship with First Order Logic - Reasoning

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

Theorem

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

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

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):

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

$\rightsquigarrow \mathcal{ALC}$ 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 ALC (tradeoff expressive power vs. complexity of reasoning).

DLs as fragments of First Order Logic - Exercise

Exercise

Translate the following $\mathcal{ALC}$ concepts into FOL 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. $Father(x) \wedge \forall y.(child(x,y) \rightarrow (Doctor(y) \vee Manager(y)))$
2. $\exists y.(manages(x,y) \wedge (Company(y) \wedge \exists x.(employs(y,x) \wedge Doctor(x))))$
3. $Father(x) \wedge \forall y.(child(x,y) \rightarrow (Doctor(y) \vee \exists x.(manages(y,x) \wedge (Company(x) \wedge \exists y.(employs(x,y) \wedge Doctor(y))))))$

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.,
  $t_x(\exists P^-.C) = \exists y.P(y,x) \wedge t_y(C)$ with y a new variable
  $t_x(\forall P^-.C) = \forall y.P(y,x) \rightarrow t_y(C)$ with y a new variable
  $\rightsquigarrow \mathcal{ALCI}$ is still a fragment of $L2$
- For number restrictions, two variables do not suffice
  $\rightsquigarrow \mathcal{ALCQI}$ is a fragment of $C2$ (i.e, $L2$+counting quantifiers)

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 $\mathcal{SHIF}(\mathcal{D})$, 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,
  - $(\mathcal{D})$ stand for data types, which are necessary in any practical knowledge representation language.
- OWL1 DL is a variant of $\mathcal{SHOIN}(\mathcal{D})$, 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 $\mathcal{SROIQ}(\mathcal{D})$, 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 inthe 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`	$\{a_1\} \sqcup \ldots \{a_n\}$	$\{john\} \sqcup \{mary\}$
`ObjectAllValuesFrom`	$\forall P.C$	$\forall hasChild.Doctor$
`ObjectSomeValuesFrom`	$\exists P.C$	$\exists hasChild.Lawyer$
`ObjectMaxCardinality`	$(\leq nP)$	$(\leq 1 hasChild)$
`ObjectMinCardinality`	$(\geq nP)$	$(\geq 2 hasChild)$
...		

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

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`	$\top \sqsubseteq (\leq 1P)$	$\top \sqsubseteq (\leq 1 hasFather)$
...		

Outline

Outline

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

Encoding UML Class Diagrams in DLs

The ideas behind the encoding of a UML Class Diagram $\mathcal{D}$ in terms of a $\mathcal{T}_{\mathcal{D}}$ TBox TD 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

Definition (Encoding of UML 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:

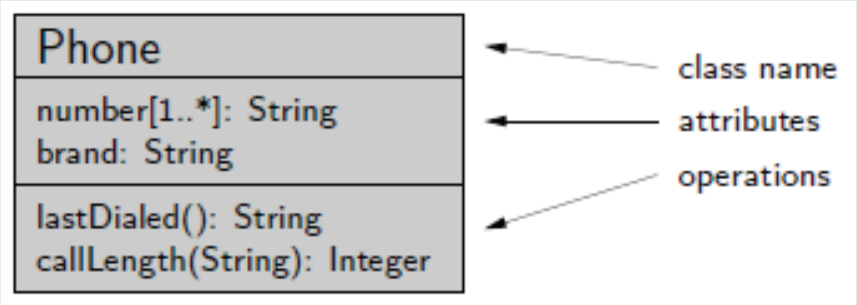
$$C \sqsubseteq \exists a \sqcap (\leq 1\ a)$$

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

Example



- To encode the class Phone, we introduce a concept Phone.
- Encoding of the attributes number and brand:
  $\exists number \sqsubseteq Phone \quad \exists number^- \sqsubseteq String$
  $\exists brand \sqsubseteq Phone \quad \exists brand^- \sqsubseteq String$
- Encoding of the multiplicities of the attributes number and brand:

$$Phone \sqsubseteq \exists number$$
$$Phone \sqsubseteq \exists brand \sqcap (\leq 1\ brand)$$

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

Encoding of associations

The encoding of associations depends on:

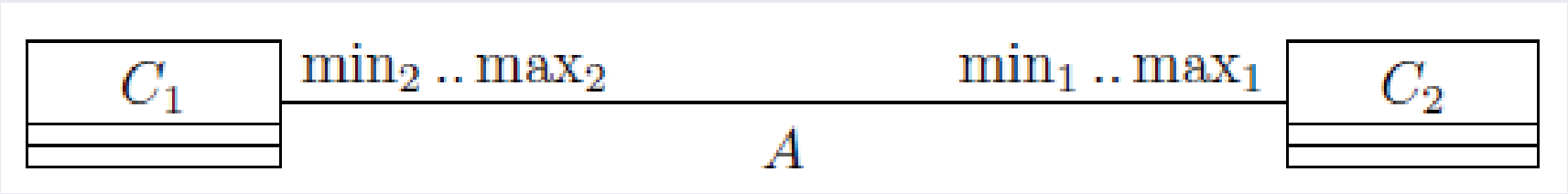
- the presence/absence of an association class;
- the arity of the association.

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

Definition (Encoding of UML binary associations without association class)



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

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

- 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:

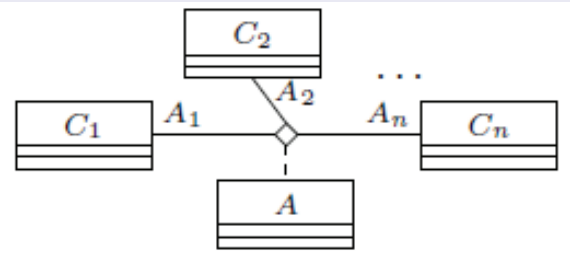
$$C_1 \sqsubseteq (\geq min_1\ A) \sqcap (\leq max_1\ A)$$

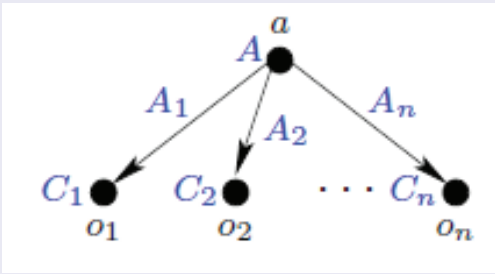
  - each instance of class C_2 is connected through association A^- to at least min_2 and at most max_2 instances of C_1:

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

Encoding of associations via reification

Definition (Encoding of UML 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:

$$\exists A_i \sqsubseteq A, \qquad for \quad i \in \{1, \ldots, n\}$$

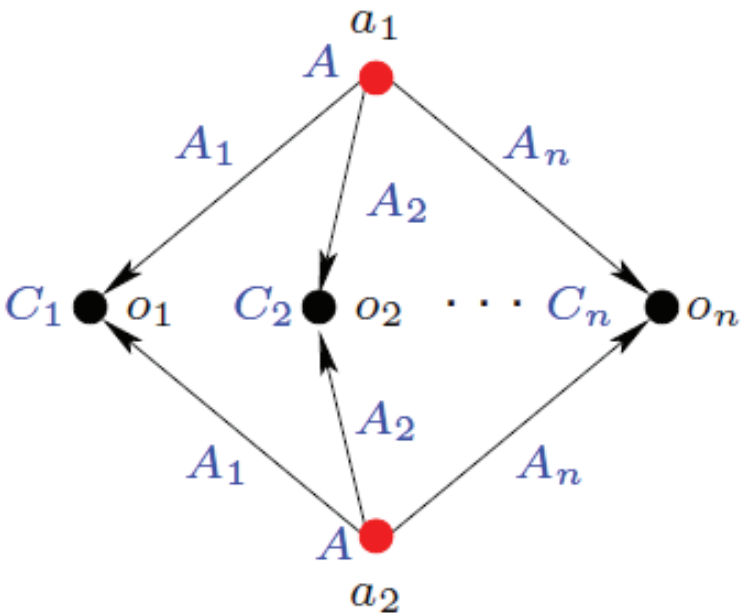
$$\exists A_i^- \sqsubseteq C_i, \qquad for \quad i \in \{1, \ldots, n\}$$

$$A \sqsubseteq \exists A_1 \sqcap \cdots \sqcap \exists A_n \sqcap (\leq 1\ A_1) \sqcap \cdots \sqcap (\leq 1\ A_n)$$

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:

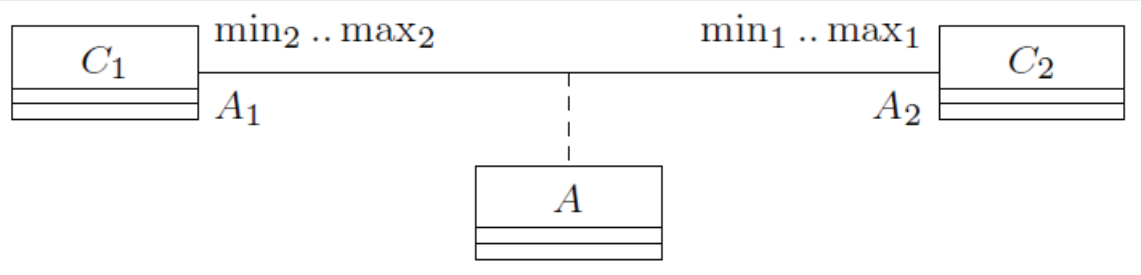


To rule out such a situation we could add an identification assertion (see later):
(**id** A $A_1, \ldots, A_n$)

Note: in a tree-model the above situation cannot occur.
$\rightsquigarrow$ 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

Definition (Encoding of multiplicities of UNL 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:

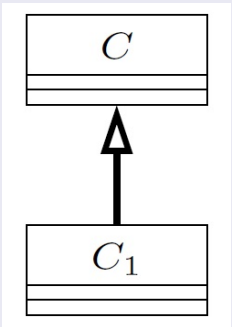
$$C_1 \sqsubseteq (\geq min_1\ A_1^-) \sqcap (\leq max_1\ A_1^-)$$

- 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:

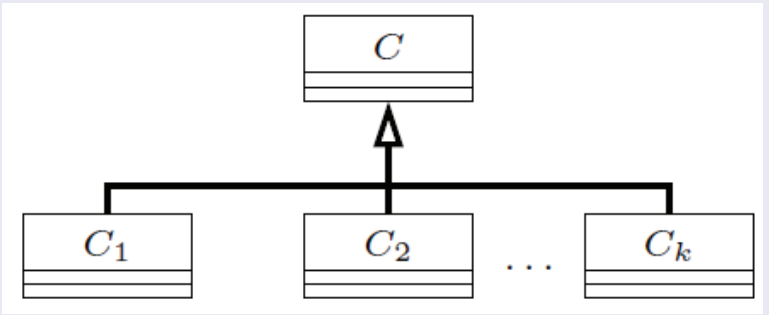
$$C_2 \sqsubseteq (\geq min_2\ A_2^-) \sqcap (\leq max_2\ A_2^-)$$

Encoding of ISA and generalization

Definition (Encoding of UML ISA and generalization)



$$C_1 \sqsubseteq C$$



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

- When the generalization is disjoint:

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

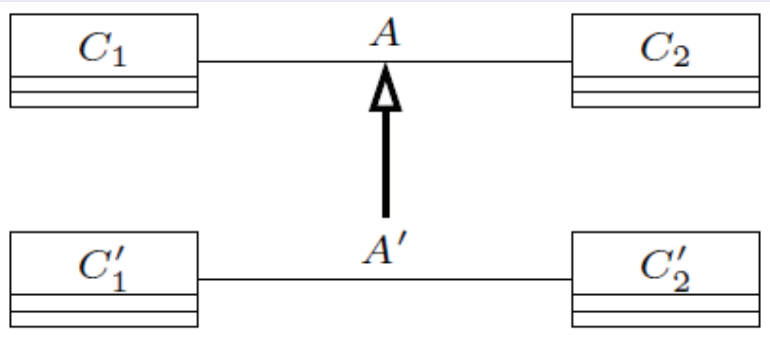
- When the generalization is complete:

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

Encoding of ISA between associations

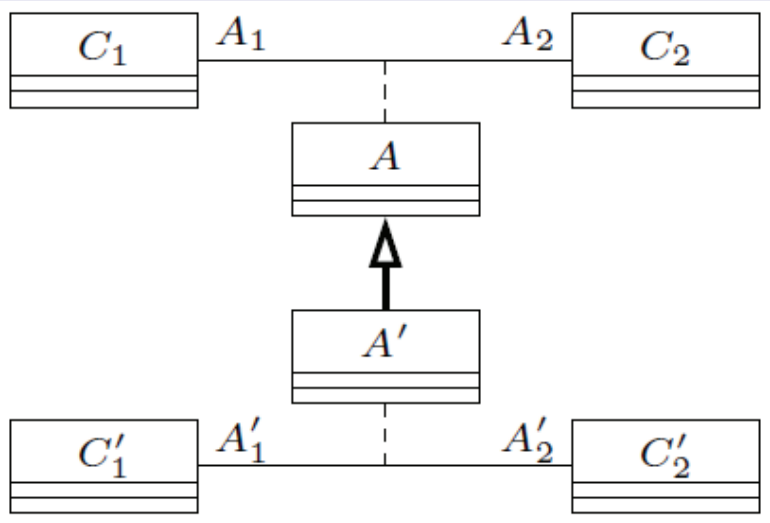
Definition (Encoding of UML ISA between associations)

- Without reification:



Role inclusion assertion: $A' \sqsubseteq A$

- With reification:



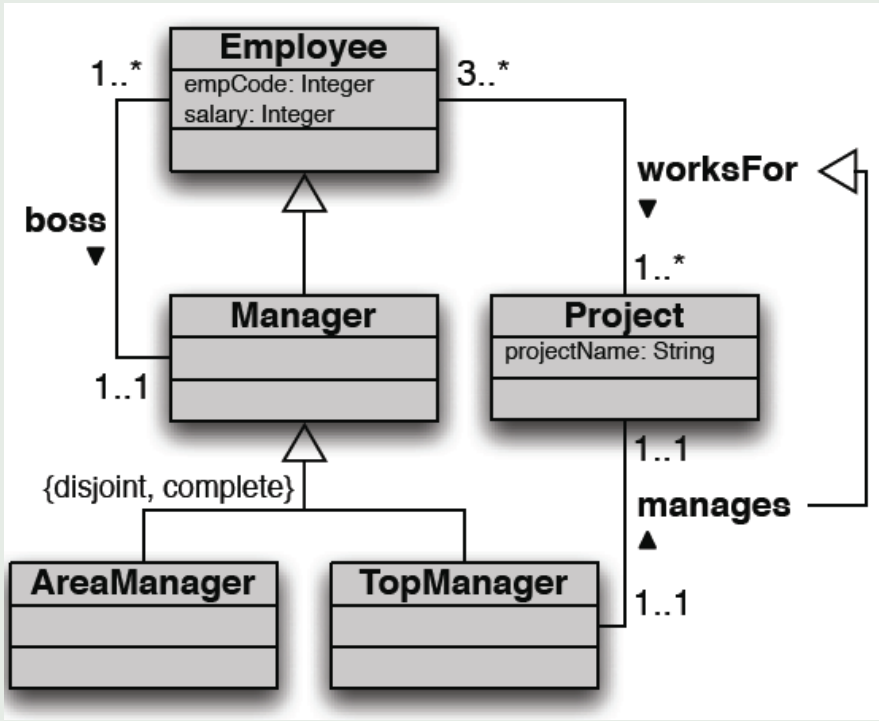
Concept inclusion assertions: $A' \sqsubseteq A$

Role inclusion assertions: $A_1' \sqsubseteq A_1$

$A_2' \sqsubseteq A_2$

Encoding UML Class Diagrams in DLs - Example 2

Example



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

Outline

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.

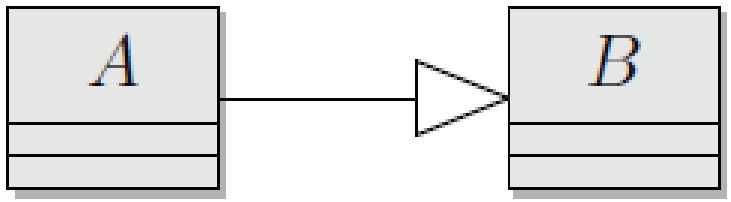
We need to encode the following kinds of inclusion assertions:

$$\begin{array}{lcl} A & \sqsubseteq & B \\ A & \sqsubseteq & \neg \mathsf{B} \\ A & \sqsubseteq & B_1 \sqcup B_2 \end{array}$$

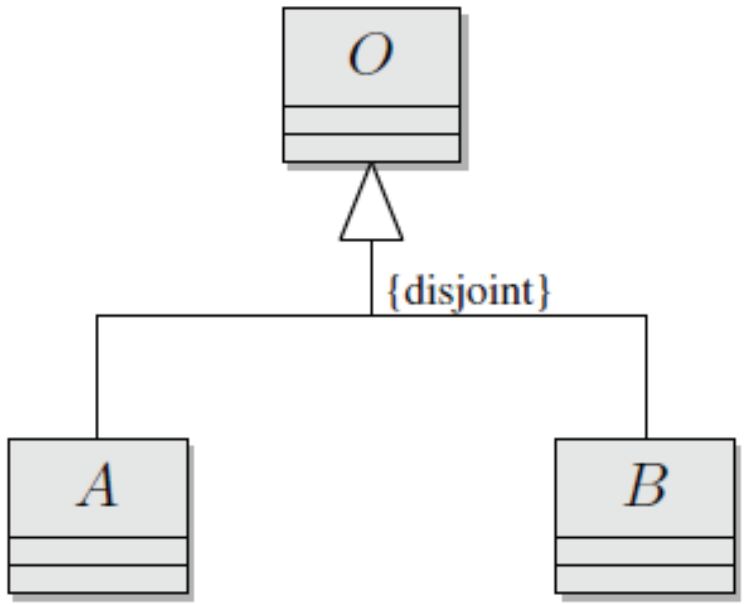
$$\begin{array}{lcl} A & \sqsubseteq & \exists P.B \\ A & \sqsubseteq & \forall P.\mathsf{B} \end{array}$$

Encoding of inclusion and of disjointness

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

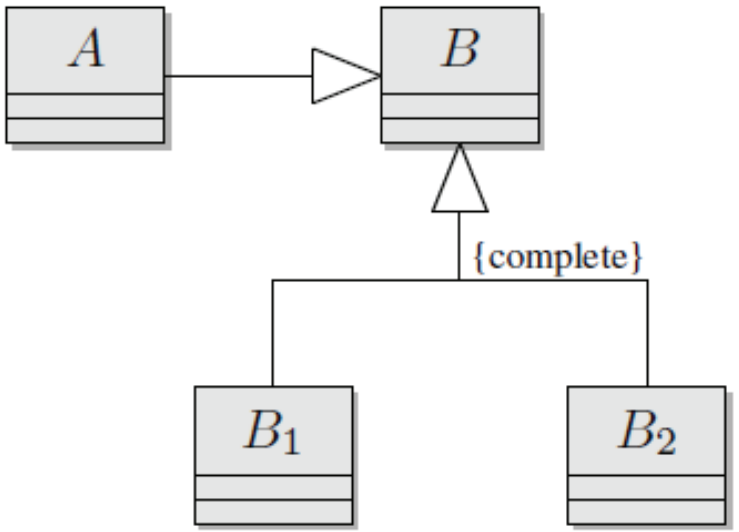


For each assertion $A \sqsubseteq \neg B$ of $\mathcal{T}$, add the following to $\mathcal{D}_{\mathcal{T}}$:



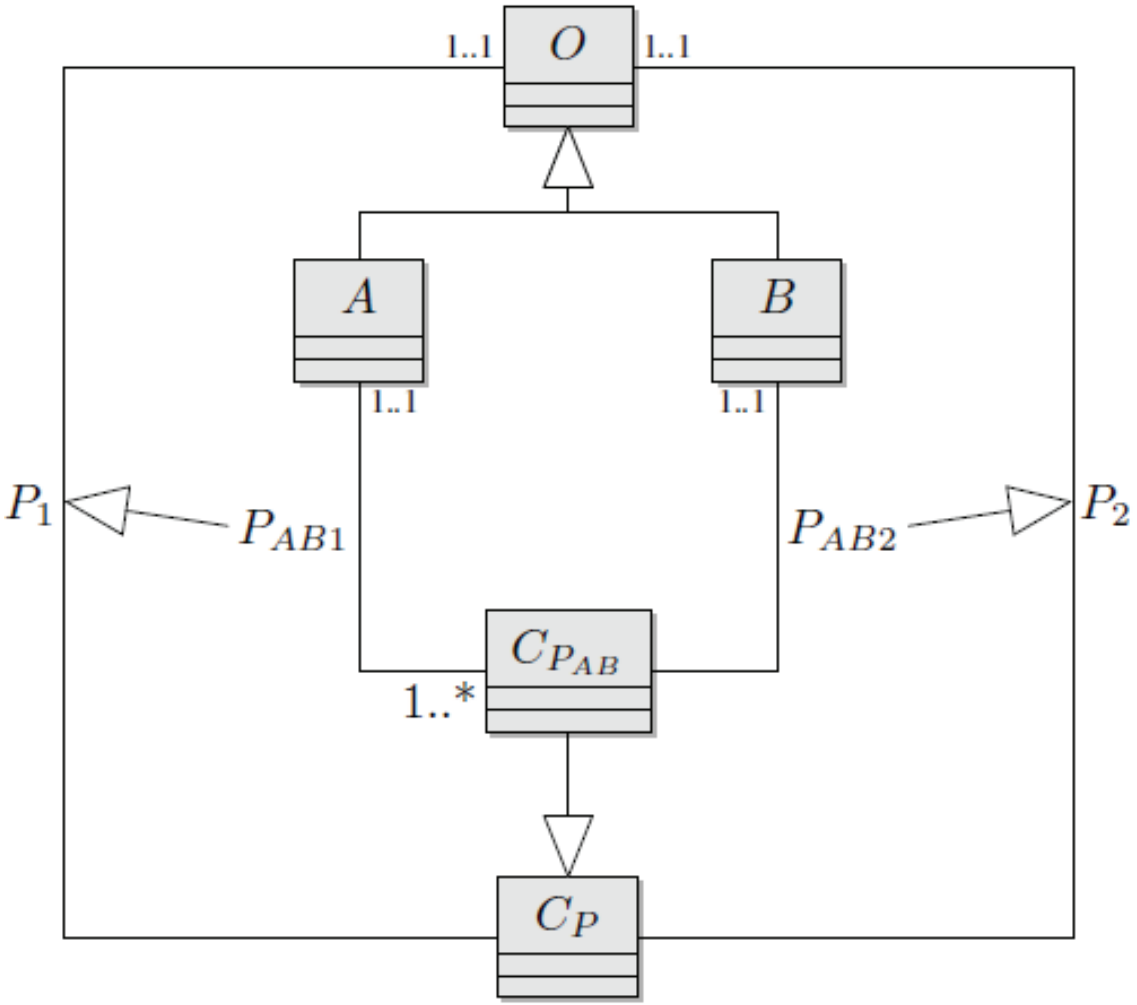
Encoding of union

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



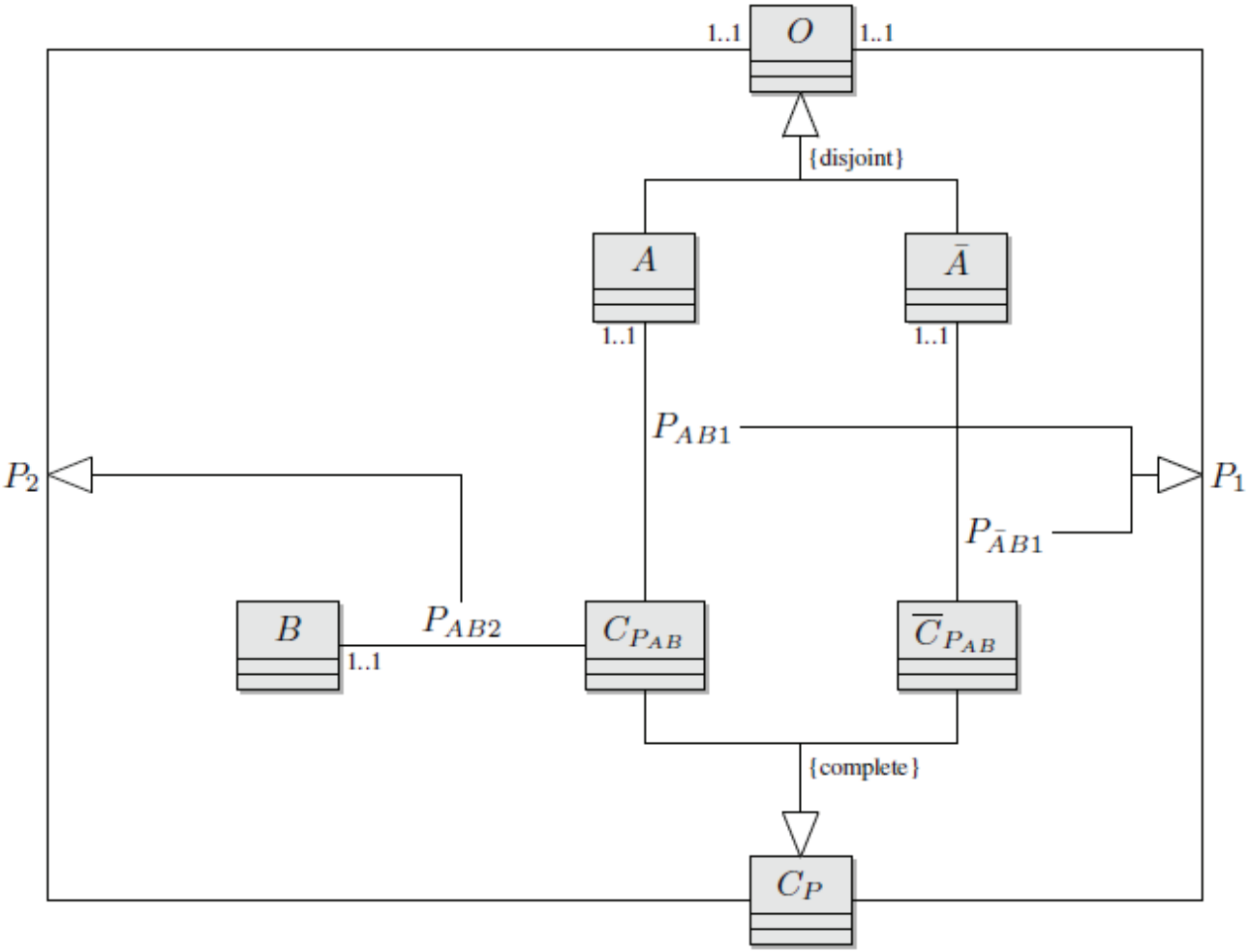
Encoding of existential quantification

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



Encoding of universal quantification

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



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

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