Description Logics for Conceptual Modeling

Diego Calvanese

KRDB Research Centre for Knowledge and Data
Free University of Bozen-Bolzano, Italy

Currently on sabbatical leave at Technical University Vienna, Austria

EPCL Basic Training Camp 2012/2013
10–21/12/2012 Dresden, Germany
Outline

1. Introduction to conceptual modeling
2. Formalizing UML Class Diagrams in FOL
3. Forms of reasoning on UML Class Diagrams
4. Brief overview of Description Logics
5. Reasoning on UML Class Diagrams using Description Logics
6. References
Outline

1. Introduction to conceptual modeling
2. Formalizing UML Class Diagrams in FOL
3. Forms of reasoning on UML Class Diagrams
4. Brief overview of Description Logics
5. Reasoning on UML Class Diagrams using Description Logics
6. References
Modeling information systems

**Information system**

Is a system that collects, stores, and processes information about the **state of a domain** of interest.

To function properly, an information system needs to:

- maintain an internal representation of the state of the domain,
- provide information about the state,
- change the state by performing actions.

*Note:* In this lecture, we ignore the dynamic aspects related to modeling the evolution of the state and the state changes, and concentrate on modeling the **static aspects only**.
To design an information system, we need proper formalisms for representing the state of the domain of interest.

- We are interested in capturing the **semantics** of the domain of interest, through suitable constraints that rule out undesired states.

- We are also interested in **automated support** to the design tasks, by suitably manipulating the representation at the semantic level.

- Increasingly, information systems may use the semantic representation also at operation time, to better support the user interaction with the system, and the functioning of the system itself.
Semantic model of the domain of interest

We aim at obtaining a description of the state of the domain of interest in **semantic terms**.

One can proceed as follows:

1. Represent the domain of interest as a **conceptual schema**, similar to those used at design time to design a database.

2. Formalize the conceptual schema as a **logical theory** (also called an ontology).

3. Use the resulting logical theory to:
   - **reason** about the domain of interest at design time, and
   - extract information from the representation at run-time, through **query answering**.
Let’s start with an exercise

**Requirements:** We are interested in building a software application to manage filmed scenes for realizing a movie, by following the so-called “Hollywood Approach”.

Every scene is identified by a code (a string) and is described by a text in natural language.

Every scene is filmed from different positions (at least one), each of this is called a setup. Every setup is characterized by a code (a string) and a text in natural language where the photographic parameters are noted (e.g., aperture, exposure, focal length, filters, etc.). Note that a setup is related to a single scene.

For every setup, several takes may be filmed (at least one). Every take is characterized by a (positive) natural number, a real number representing the number of meters of film that have been used for shooting the take, and the code (a string) of the reel where the film is stored. Note that a take is associated to a single setup.

Scenes are divided into internals that are filmed in a theater, and externals that are filmed in a location and can either be “day scene” or “night scene”. Locations are characterized by a code (a string) and the address of the location, and a text describing them in natural language.

Write a precise specification of this domain using any formalism you like!
Solution 1: Use conceptual modeling diagrams (UML)!
Solution 1: Use conceptual modeling diagrams – Discussion

Good points:
- Easy to generate (it’s the standard in software design).
- Easy to understand for humans.
- Well disciplined, well-established methodologies available.

Bad points:
- No precise semantics (people that use it wave hands about it).
- Verification (or better validation) done informally by humans.
- Machine incomprehensible (because of lack of formal semantics).
- Automated reasoning and query answering out of question.
- Limited expressiveness (\*).

\(*\) Not really a bad point, in fact.
Solution 2: Use logic!!

**Alphabet:** \(\text{Scene}(x), \text{Setup}(x), \text{Take}(x), \text{Internal}(x), \text{External}(x), \text{Location}(x), \)
\(\text{stpForScn}(x, y), \text{tkOfStp}(x, y), \text{located}(x, y), \ldots\)

**Axioms:**

\[
\begin{align*}
\forall x, y. & \text{codeScene}(x, y) \rightarrow \text{Scene}(x) \land \text{String}(y) \\
\forall x, y. & \text{description}(x, y) \rightarrow \text{Scene}(x) \land \text{Text}(y) \\
\forall x, y. & \text{codeSetup}(x, y) \rightarrow \text{Setup}(x) \land \text{String}(y) \\
\forall x, y. & \text{photographicPars}(x, y) \rightarrow \text{Setup}(x) \land \text{Text}(y) \\
\forall x, y. & \text{nbr}(x, y) \rightarrow \text{Take}(x) \land \text{Integer}(y) \\
\forall x, y. & \text{filmemMeters}(x, y) \rightarrow \text{Take}(x) \land \text{Real}(y) \\
\forall x, y. & \text{reel}(x, y) \rightarrow \text{Take}(x) \land \text{String}(y) \\
\forall x, y. & \text{theater}(x, y) \rightarrow \text{Internal}(x) \land \text{String}(y) \\
\forall x, y. & \text{name}(x, y) \rightarrow \text{External}(x) \land \text{Boolean}(y) \\
\forall x, y. & \text{address}(x, y) \rightarrow \text{Location}(x) \land \text{String}(y) \\
\forall x, y. & \text{description}(x, y) \rightarrow \text{Location}(x) \land \text{Text}(y) \\
\forall x. & \text{Scene}(x) \rightarrow (1 \leq \#\{y \mid \text{codeScene}(x, y)\} \leq 1) \\
\forall x. & \text{Internal}(x) \rightarrow \text{Scene}(x) \\
\forall x. & \text{External}(x) \rightarrow \text{Scene}(x) \\
\forall x. & \text{Internal}(x) \rightarrow \neg \text{External}(x) \\
\forall x. & \text{Scene}(x) \rightarrow \text{Internal}(x) \lor \text{External}(x)
\end{align*}
\]
Solution 2: Use logic – Discussion

Good points:

- Precise semantics.
- Formal verification.
- Allows for query answering.
- Machine comprehensible.
- Virtually unlimited expressiveness (*).

Bad points:

- Difficult to generate.
- Difficult to understand for humans.
- Too unstructured (making reasoning difficult), no well-established methodologies available.
- Automated reasoning may be impossible.

(*) Not really a bad point, in fact.
Solution 3: Use both!!!

*Note:* these two approaches seem to be orthogonal, but in fact they can be used together cooperatively.

### Basic idea:
- Assign formal semantics to constructs of the conceptual design diagrams.
- Use conceptual design diagrams as usual, taking advantage of methodologies developed for them in Software Engineering.
- Read diagrams as logical theories when needed, i.e., for formal understanding, verification, automated reasoning, etc.

### Added values:
- Inherited from conceptual modeling diagrams: ease-to-use for humans
- Inherit from logic: formal semantics and reasoning tasks, which are needed for formal verification and machine manipulation.
Solution 3: Use both!!! (cont’d)

**Important:**

The logical theories that are obtained from conceptual modeling diagrams are of a specific form.

- Their expressiveness is limited (or better, well-disciplined).
- One can exploit the particular form of the logical theory to simplify reasoning.
- The aim is getting:
  - decidability, and
  - reasoning procedures that match the intrinsic computational complexity of reasoning over the conceptual modeling diagrams.
Conceptual models vs. logic

We illustrate now what we get from interpreting conceptual modeling diagrams in logic.

We will use:

- As conceptual modeling diagrams: **UML Class Diagrams**.
  - Note: we could equivalently use Entity-Relationship Diagrams instead of UML.
- As logic:
  - **First-Order Logic**, to formally capture semantics and the different forms of reasoning over conceptual modeling diagrams.
  - **Description Logics**, to show that automated reasoning over conceptual modeling diagrams is feasible.
Outline

1. Introduction to conceptual modeling

2. Formalizing UML Class Diagrams in FOL
   - Classes
   - Associations
   - Attributes
   - Generalizations
   - Associations classes

3. Forms of reasoning on UML Class Diagrams

4. Brief overview of Description Logics

5. Reasoning on UML Class Diagrams using Description Logics

6. References
The Unified Modeling Language (UML)

The **Unified Modeling Language (UML)** was developed in 1994 by unifying and integrating the most prominent object-oriented modeling approaches:

- Booch
- Rumbaugh: Object Modeling Technique (OMT)
- Jacobson: Object-Oriented Software Engineering (OOSE)

**History:**

- 1995, version 0.8, Booch, Rumbaugh; 1996, version 0.9, Booch, Rumbaugh, Jacobson; version 1.0 BRJ + Digital, IBM, HP, …
- UML 1.4.2 is industrial standard ISO/IEC 19501.
- 1999–today: **de facto** standard object-oriented modeling language.

**References:**

In this lecture we deal only with one of the most prominent components of UML: UML Class Diagrams.

A UML Class Diagram is used to represent explicitly the information on a domain of interest (typically the application domain of software).

Note: This is exactly the goal of all conceptual modeling formalism, such as Entity-Relationship Diagrams (standard in Database design) or Ontologies.
UML Class Diagrams (cont’d)

The UML class diagram models the domain of interest in terms of:

- objects grouped into **classes**;
- **associations**, representing relationships between classes;
- **attributes**, representing simple properties of the instances of classes;
  
  *Note:* here we do not deal with “operations”.

- **sub-classing**, i.e., ISA and generalization relationships.
Example of a UML Class Diagram

**Scene**
- code: String
- description: Text

**Internal**
- theater: String

**External**
- nightScene: Boolean

**Take**
- nbr: Integer
- filmedMeters: Real
- reel: String

**Setup**
- code: String
- photographicPars: Text

**Location**
- name: String
- address: String
- description: Text

**TkOfStp**
- 1..*

**stpForScn**
- 1..1

**Located**
- 0..*

**Take**
- 1..1

**Setup**
- 1..*

**Scene**
- 1..1

**Location**
- 1..1

**Internal**
- 1..1

**External**
- 0..*

**{disjoint, complete}**

Diego Calvanese (FUB)
Use of UML Class Diagrams

UML Class Diagrams are used in various phases of a software design:

1. During the so-called **analysis**, where an abstract precise view of the domain of interest needs to be developed.
   \( \leadsto \) the so-called “**conceptual perspective**”.

2. During **software development**, to maintain an abstract view of the software to be developed.
   \( \leadsto \) the so-called “**implementation perspective**”.

*In this lecture we focus on 1!*
UML class diagrams (when used for the conceptual perspective) closely resemble Entity-Relationship (ER) Diagrams.

Example of UML vs. ER:
Outline

1. Introduction to conceptual modeling

2. Formalizing UML Class Diagrams in FOL
   - Classes
   - Associations
   - Attributes
   - Generalizations
   - Associations classes

3. Forms of reasoning on UML Class Diagrams

4. Brief overview of Description Logics

5. Reasoning on UML Class Diagrams using Description Logics

6. References
**Classes in UML**

A **class** in UML models a **set of objects** (its “instances”) that share certain common properties, such as **attributes**, **operations**, etc.

Each class is characterized by:

- a **name** (which must be unique in the whole class diagram),
- a **set of (local) properties**, namely **attributes** and **operations** (see later).

**Example**

<table>
<thead>
<tr>
<th>Book</th>
</tr>
</thead>
<tbody>
<tr>
<td>title: String</td>
</tr>
<tr>
<td>pages: Integer</td>
</tr>
</tbody>
</table>

- the name of the class is ‘Book’
- the class has two properties (attributes)
The objects that belong to a class are called **instances** of the class. They form a so-called **instantiation** (or **extension**) of the class.

**Example**

Here are some possible instantiations of our class Book:

\[
\{\text{book}_a, \text{book}_b, \text{book}_c, \text{book}_d, \text{book}_e\} \\
\{\text{book}_\alpha, \text{book}_\beta\} \\
\{\text{book}_1, \text{book}_2, \text{book}_3, \ldots, \text{book}_{500}, \ldots\}
\]

Which is the actual instantiation?

**We will know it only at run-time!!!** – We are now at design time!
A class represents a set of objects. ... But which set? We don’t actually know. So, how can we assign a semantics to such a class?

We represent a class as a **FOL unary predicate**!

**Example**

For our class Book, we introduce a predicate $Book(x)$. 

Outline

1. Introduction to conceptual modeling

2. Formalizing UML Class Diagrams in FOL
   - Classes
   - Associations
   - Attributes
   - Generalizations
   - Associations classes

3. Forms of reasoning on UML Class Diagrams

4. Brief overview of Description Logics

5. Reasoning on UML Class Diagrams using Description Logics

6. References
An **association** in UML models a **relationship** between two or more classes.

- At the instance level, an association is a relation between the instances of two or more classes.
- Associations model properties of classes that are **non-local**, in the sense that they involve other classes.
- An association between $n$ classes is a property of each of these classes.

---

**Example**

![UML diagram showing an association between Author and Book classes](image-url)
We can represent an \( n \)-ary association \( A \) among classes \( C_1, \ldots, C_n \) as an \( n \)-ary predicate \( A \) in FOL.

We assert that the components of the predicate must belong to the classes participating to the association:

\[
\forall x_1, \ldots, x_n . \ A(x_1, \ldots, x_n) \rightarrow C_1(x_1) \land \cdots \land C_n(x_n)
\]

**Example**

\[
\forall x_1, x_2 . \ writtenBy(x_1, x_2) \rightarrow Book(x_1) \land Author(x_2)
\]
Associations: multiplicity

On binary associations, we can place multiplicity constraints, i.e., a minimal and maximal number of tuples in which every object participates as first (second) component.

Example

<table>
<thead>
<tr>
<th>Book</th>
<th>writtenBy</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>title: String</td>
<td>0..*</td>
<td>1..*</td>
</tr>
<tr>
<td>pages: Integer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: UML multiplicities for associations are look-across and are not easy to use in an intuitive way for n-ary associations. So typically they are not used at all.

In contrast, in ER Schemas, multiplicities are not look-across and are easy to use, and widely used.
**Associations: formalization of multiplicities**

Multiplicities of binary associations are easily expressible in FOL:

\[
\forall x_1. \ C_1(x_1) \rightarrow (\text{min}_1 \leq \#\{x_2 \mid A(x_1, x_2)\} \leq \text{max}_1)
\]

\[
\forall x_2. \ C_2(x_2) \rightarrow (\text{min}_2 \leq \#\{x_1 \mid A(x_1, x_2)\} \leq \text{max}_2)
\]

**Example**

\[
\forall x. \ \text{Book}(x) \rightarrow (1 \leq \#\{y \mid \text{written}_by(x, y)\})
\]

*Note:* this is a shorthand for a FOL formula expressing the cardinality of the set of possible values for \(y\).
In our example ...

```
Scene
  code: String
  description: Text

Internal
  theater: String

External
  nightScene: Boolean

Take
  nbr: Integer
  filmedMeters: Real
  reel: String

Setup
  code: String
  photographicPars: Text

Location
  name: String
  address: String
  description: Text

stpForScn
  1..1

tkOfStp
  1..* => 1..1

located
  0..* => 1..1
```

Diego Calvanese (FUB)
In our example ...

**Alphabet:** $\text{Scene}(x), \text{Setup}(x), \text{Take}(x), \text{Internal}(x), \text{External}(x), \text{Location}(x),\ stpForScn(x, y), \ tkOfStp(x, y), \ located(x, y), \ldots$

**Axioms:**

\[
\forall x, y. \ code_{\text{Scene}}(x, y) \rightarrow \text{Scene}(x) \land \text{String}(y)
\]

\[
\forall x, y. \ description(x, y) \rightarrow \text{Scene}(x) \land \text{Text}(y)
\]

\[
\forall x, y. \ code_{\text{Setup}}(x, y) \rightarrow \text{Setup}(x) \land \text{String}(y)
\]

\[
\forall x, y. \ photographicPars(x, y) \rightarrow \text{Setup}(x) \land \text{Text}(y)
\]

\[
\forall x, y. \ nbr(x, y) \rightarrow \text{Take}(x) \land \text{Integer}(y)
\]

\[
\forall x, y. \ filmedMeters(x, y) \rightarrow \text{Take}(x) \land \text{Real}(y)
\]

\[
\forall x, y. \ reel(x, y) \rightarrow \text{Take}(x) \land \text{String}(y)
\]

\[
\forall x, y. \ theater(x, y) \rightarrow \text{Internal}(x) \land \text{String}(y)
\]

\[
\forall x, y. \ nightScene(x, y) \rightarrow \text{External}(x) \land \text{String}(y)
\]

\[
\forall x, y. \ name(x, y) \rightarrow \text{Location}(x) \land \text{String}(y)
\]

\[
\forall x, y. \ address(x, y) \rightarrow \text{Location}(x) \land \text{String}(y)
\]

\[
\forall x, y. \ description(x, y) \rightarrow \text{Location}(x) \land \text{Text}(y)
\]

\[
\forall x. \ \text{Scene}(x) \rightarrow (1 \leq \#\{y \mid code_{\text{Scene}}(x, y)\} \leq 1)
\]

\[
\forall x. \ \text{Internal}(x) \rightarrow \text{Scene}(x)
\]

\[
\forall x. \ \text{External}(x) \rightarrow \text{Scene}(x)
\]

\[
\forall x. \ \text{Internal}(x) \rightarrow \neg \text{External}(x)
\]

\[
\forall x. \ \text{Scene}(x) \rightarrow \text{Internal}(x) \lor \text{External}(x)
\]
**Associations: most interesting multiplicities**

The most interesting multiplicities are:

- **0..*:** unconstrained
- **1..*:** mandatory participation
- **0..1:** functional participation (the association is a partial function)
- **1..1:** mandatory and functional participation (the association is a total function)

**In FOL:**

- **0..*:** no constraint
- **1..*:** \( \forall x \cdot C_1(x) \rightarrow \exists y \cdot A(x, y) \)
- **0..1:** \( \forall x \cdot C_1(x) \rightarrow \forall y, y' . A(x, y) \land A(x, y') \rightarrow y = y' \)
  (or simply \( \forall x, y, y' . A(x, y) \land A(x, y') \rightarrow y = y' \))
- **1..1:** \( (\forall x \cdot C_1(x) \rightarrow \exists y . A(x, y)) \land (\forall x, y, y' . A(x, y) \land A(x, y') \rightarrow y = y') \)
Outline

1. Introduction to conceptual modeling

2. Formalizing UML Class Diagrams in FOL
   - Classes
   - Associations
   - Attributes
   - Generalizations
   - Associations classes

3. Forms of reasoning on UML Class Diagrams

4. Brief overview of Description Logics

5. Reasoning on UML Class Diagrams using Description Logics

6. References
Attributes

An **attribute** models a local property of a class.

It is characterized by:

- a **name** (which is unique only in the class it belongs to),
- a **type** (a collection of possible values),
- and possibly a **multiplicity**.

**Example**

<table>
<thead>
<tr>
<th>Book</th>
</tr>
</thead>
<tbody>
<tr>
<td>title: String</td>
</tr>
<tr>
<td>pages: Integer</td>
</tr>
</tbody>
</table>

- The name of one of the attributes is ‘title’.
- Its type is ‘String’.
Attributes as functions

Attributes (without explicit multiplicity) are:

- **mandatory** (must have at least a value), and
- **single-valued** (can have at most one value).

That is, they are **total functions** from the instances of the class to the values of the type they have.

**Example**

\[ book_3 \] has as value for the attribute ‘title’ the String: "The little digital video book".
Attributes with multiplicity

More generally attributes may have an explicit multiplicity (similar to that of associations).

Example

<table>
<thead>
<tr>
<th>Book</th>
</tr>
</thead>
<tbody>
<tr>
<td>title: String</td>
</tr>
<tr>
<td>pages: Integer</td>
</tr>
<tr>
<td>keywords: String {1..5}</td>
</tr>
</tbody>
</table>

- The attribute ‘title’ has an implicit multiplicity of 1..1.
- The attribute ‘keywords’ has an explicit multiplicity of 1..5.

Note: When the multiplicity is not specified, then it is assumed to be 1..1.
Attributes: formalization

Since attributes may have a multiplicity different from 1..1, they are better formalized as binary predicates, with suitable assertions representing types and multiplicity.

Given an attribute \( att \) of a class \( C \) with type \( T \) and multiplicity \( i..j \), we capture it in FOL as a binary predicate \( att_C(x, y) \) with the following assertions:

- An assertion for the attribute type:

\[
\forall x, y. \ att_C(x, y) \rightarrow C(x) \land T(y)
\]

- An assertion for the multiplicity:

\[
\forall x. \ C(x) \rightarrow (i \leq \#\{y \mid att_C(x, y)\} \leq j)
\]
Attributes: example of formalization

\[
\forall x, y. \text{title}_B(x, y) \rightarrow \text{Book}(x) \land \text{String}(y)
\]

\[
\forall x. \text{Book}(x) \rightarrow (1 \leq \# \{ y \mid \text{title}_B(x, y) \} \leq 1)
\]

\[
\forall x, y. \text{pages}_B(x, y) \rightarrow \text{Book}(x) \land \text{Integer}(y)
\]

\[
\forall x. \text{Book}(x) \rightarrow (1 \leq \# \{ y \mid \text{pages}_B(x, y) \} \leq 1)
\]

\[
\forall x, y. \text{keywords}_B(x, y) \rightarrow \text{Book}(x) \land \text{String}(y)
\]

\[
\forall x. \text{Book}(x) \rightarrow (1 \leq \# \{ y \mid \text{keywords}_B(x, y) \} \leq 5)
\]
In our example ...
Attributes

Alphabet: \(\text{Scene}(x), \text{Setup}(x), \text{Take}(x), \text{Internal}(x), \text{External}(x), \text{Location}(x), \)

\(\text{stpForScn}(x, y), \text{tkOfStp}(x, y), \text{located}(x, y), \ldots\)

Axioms:

\(\forall x, y. \text{code}_{\text{Scene}}(x, y) \rightarrow \text{Scene}(x) \land \text{String}(y)\)
\(\forall x, y. \text{description}(x, y) \rightarrow \text{Scene}(x) \land \text{Text}(y)\)
\(\forall x, y. \text{code}_{\text{Setup}}(x, y) \rightarrow \text{Setup}(x) \land \text{String}(y)\)
\(\forall x, y. \text{photographicPars}(x, y) \rightarrow \text{Setup}(x) \land \text{Text}(y)\)
\(\forall x, y. \text{nbr}(x, y) \rightarrow \text{Take}(x) \land \text{Integer}(y)\)
\(\forall x, y. \text{filmedMeters}(x, y) \rightarrow \text{Take}(x) \land \text{Real}(y)\)
\(\forall x, y. \text{reel}(x, y) \rightarrow \text{Take}(x) \land \text{String}(y)\)
\(\forall x, y. \text{theater}(x, y) \rightarrow \text{Internal}(x) \land \text{String}(y)\)
\(\forall x, y. \text{name}(x, y) \rightarrow \text{Location}(x) \land \text{String}(y)\)
\(\forall x, y. \text{address}(x, y) \rightarrow \text{Location}(x) \land \text{String}(y)\)
\(\forall x, y. \text{description}(x, y) \rightarrow \text{Location}(x) \land \text{Text}(y)\)
\(\forall x. \text{Scene}(x) \rightarrow (1 \leq \#\{y \mid \text{code}_{\text{Scene}}(x, y)\} \leq 1)\)
\(\forall x. \text{Internal}(x) \rightarrow \text{Scene}(x)\)
\(\forall x. \text{External}(x) \rightarrow \text{Scene}(x)\)
\(\forall x. \text{Internal}(x) \rightarrow \lnot \text{External}(x)\)
\(\forall x. \text{Scene}(x) \rightarrow \text{Internal}(x) \lor \text{External}(x)\)
Outline

1. Introduction to conceptual modeling
2. Formalizing UML Class Diagrams in FOL
   - Classes
   - Associations
   - Attributes
   - Generalizations
   - Associations classes
3. Forms of reasoning on UML Class Diagrams
4. Brief overview of Description Logics
5. Reasoning on UML Class Diagrams using Description Logics
6. References
ISA and generalizations

The ISA relationship is of particular importance in conceptual modeling: a class \( C \) ISA a class \( C' \) if every instance of \( C \) is also an instance of \( C' \).

In UML, the ISA relationship is modeled through the notion of generalization.

**Example**

```
Person
  name: String

Author
  kindOfWriter: String

The attribute ‘name’ is inherited by ‘Author’.
```
A **generalization** involves a **superclass** (base class) and one or more **subclasses**: every instance of each subclass is also an instance of the superclass.
Generalizations with constraints

The ability of having more subclasses in the same generalization, allows for placing suitable constraints on the classes involved in the generalization.
Generalizations with constraints (cont’d)

Most notable and used constraints:

- **Disjointness**, which asserts that different subclasses cannot have common instances (i.e., an object cannot be at the same time instance of two disjoint subclasses).

- **Completeness** (aka “covering”), which asserts that every instance of the superclass is also an instance of at least one of the subclasses.

Example
Generalizations: formalization

\[ \forall x. C_i(x) \rightarrow C'(x), \quad \text{for } 1 \leq i \leq k \]

Disjointness:
\[ \forall x. C_i(x) \rightarrow \neg C_j(x), \quad \text{for } 1 \leq i < j \leq k \]

Completeness:
\[ \forall x. C'(x) \rightarrow \bigvee_{i=1}^{k} C_i(x) \]
Generalizations: example of formalization

\[ \forall x. \text{Child}(x) \rightarrow \text{Person}(x) \]
\[ \forall x. \text{Teenager}(x) \rightarrow \text{Person}(x) \]
\[ \forall x. \text{Adult}(x) \rightarrow \text{Person}(x) \]

\[ \forall x. \text{Child}(x) \rightarrow \neg \text{Teenager}(x) \]
\[ \forall x. \text{Child}(x) \rightarrow \neg \text{Adult}(x) \]
\[ \forall x. \text{Teenager}(x) \rightarrow \neg \text{Adult}(x) \]

\[ \forall x. \text{Person}(x) \rightarrow (\text{Child}(x) \vee \text{Teenager}(x) \vee \text{Adult}(x)) \]
In our example ...

Diego Calvanese (FUB)
DLs for Conceptual Modeling
EPCL BTC – 10–21/12/2012 (48/113)
In our example ...

**Alphabet:**  \( \text{Scene}(x), \text{Setup}(x), \text{Take}(x), \text{Internal}(x), \text{External}(x), \text{Location}(x), \)  
\( \text{stpForScn}(x, y), \text{tkOfStp}(x, y), \text{located}(x, y), \ldots \)  

**Axioms:**  

\[
\forall x, y. \text{code}_{\text{Scene}}(x, y) \rightarrow \text{Scene}(x) \land \text{String}(y)
\]
\[
\forall x, y. \text{description}(x, y) \rightarrow \text{Scene}(x) \land \text{Text}(y)
\]
\[
\forall x, y. \text{code}_{\text{Setup}}(x, y) \rightarrow \text{Setup}(x) \land \text{String}(y)
\]
\[
\forall x, y. \text{photographicPars}(x, y) \rightarrow \text{Setup}(x) \land \text{Text}(y)
\]
\[
\forall x, y. \text{nbr}(x, y) \rightarrow \text{Take}(x) \land \text{Integer}(y)
\]
\[
\forall x, y. \text{filmedMeters}(x, y) \rightarrow \text{Take}(x) \land \text{Real}(y)
\]
\[
\forall x, y. \text{reel}(x, y) \rightarrow \text{Take}(x) \land \text{String}(y)
\]
\[
\forall x, y. \text{theater}(x, y) \rightarrow \text{Internal}(x) \land \text{String}(y)
\]
\[
\forall x, y. \text{nightScene}(x, y) \rightarrow \text{External}(x) \land \text{Boolean}(y)
\]
\[
\forall x, y. \text{name}(x, y) \rightarrow \text{Location}(x) \land \text{String}(y)
\]
\[
\forall x, y. \text{address}(x, y) \rightarrow \text{Location}(x) \land \text{String}(y)
\]
\[
\forall x. \text{description}(x, y) \rightarrow (1 \leq \#\{y \mid \text{code}_{\text{Scene}}(x, y)\} \leq 1)
\]
\[
\forall x. \text{Scene}(x) \rightarrow (1 \leq \#\{y \mid \text{code}_{\text{Scene}}(x, y)\} \leq 1)
\]
\[
\forall x. \text{Internal}(x) \rightarrow \text{Scene}(x)
\]
\[
\forall x. \text{External}(x) \rightarrow \text{Scene}(x)
\]
\[
\forall x. \text{Internal}(x) \rightarrow \neg\text{External}(x)
\]
\[
\forall x. \text{Scene}(x) \rightarrow \text{Internal}(x) \lor \text{External}(x)
\]
Outline

1. Introduction to conceptual modeling

2. Formalizing UML Class Diagrams in FOL
   - Classes
   - Associations
   - Attributes
   - Generalizations
   - Associations classes

3. Forms of reasoning on UML Class Diagrams

4. Brief overview of Description Logics

5. Reasoning on UML Class Diagrams using Description Logics

6. References
Sometimes we may want to assert properties of associations. In UML to do so we resort to **association classes**:

- That is, we associate to an association a class whose instances are in **bijection** with the tuples of the association.
- Then we use the association class exactly as a UML class (modeling local and non-local properties).
Association class – Example

```
Book
| title: String |
| pages: Integer |

Author
| contribution: String |

writtenBy
```

```
1..* 0..* 1..*
```

Diego Calvanese (FUB)
Association class – Example (cont’d)

```
Book
  title: String
  pages: Integer

writtenBy
  contribution: String

Author
  with

Contract
  with

1..1
0..1
```

```
1..*
0..1
```
The process of putting in correspondence objects of a class (the association class) with tuples in an association is formally described as **reification**.

That is:

- We introduce a unary predicate $A$ for the association class $A$.
- We introduce $n$ new binary predicates $A_1, \ldots, A_n$, one for each of the components of the association.
- We introduce suitable assertions so that objects in the extension of the unary-predicate $A$ are in bijection with tuples in the $n$-ary association $A$. 
Association classes: formalization (cont’d)

FOL Assertions are needed for stating a bijection between instances of the association class and instances of the association:

\[
\forall x, y. \ A_i(x, y) \rightarrow A(x) \land C_i(y), \quad \text{for } i \in \{1, \ldots, n\}
\]

\[
\forall x. \ A(x) \rightarrow \exists y. \ A_i(x, y), \quad \text{for } i \in \{1, \ldots, n\}
\]

\[
\forall x, y, y'. \ A_i(x, y) \land A_i(x, y') \rightarrow y = y', \quad \text{for } i \in \{1, \ldots, n\}
\]

\[
\forall x, x', y_1, \ldots, y_n. \ \land_{i=1}^{n} (A_i(x, y_i) \land A_i(x', y_i)) \rightarrow x = x'
\]
Association classes: example of formalization

∀x, y. \( wb_1(x, y) \rightarrow writtenBy(x) \land Book(y) \)
∀x, y. \( wb_2(x, y) \rightarrow writtenBy(x) \land Author(y) \)
∀x. writtenBy(x) \rightarrow \exists y. \( wb_1(x, y) \)
∀x. writtenBy(x) \rightarrow \exists y. \( wb_2(x, y) \)
∀x, y, y'. \( wb_1(x, y) \land wb_1(x, y') \rightarrow y = y' \)
∀x, y, y'. \( wb_2(x, y) \land wb_2(x, y') \rightarrow y = y' \)
∀x, x', y_1, y_2. \( wb_1(x, y_1) \land wb_1(x', y_1) \land wb_2(x, y_2) \land wb_2(x', y_2) \rightarrow x = x' \)
## Outline

1. Introduction to conceptual modeling
2. Formalizing UML Class Diagrams in FOL
3. Forms of reasoning on UML Class Diagrams
4. Brief overview of Description Logics
5. Reasoning on UML Class Diagrams using Description Logics
6. References
Class consistency

A class is **consistent**, if the class diagram admits an instantiation in which the class has a non-empty set of instances.

Let $\Gamma$ be the set of FOL assertions corresponding to the UML Class Diagram, and $C(x)$ the predicate corresponding to a class $C$ of the diagram.

Then $C$ is consistent iff

$$\Gamma \not\models \forall x. C(x) \rightarrow false$$

i.e., there exists a model of $\Gamma$ in which the extension of $C(x)$ is not the empty set.

**Note:** Corresponding FOL reasoning task: satisfiability.
Class consistency: example (by E. Franconi)

\[ \Gamma \models \forall x. \text{LatinLover}(x) \rightarrow \text{false} \]
Whole diagram consistency

A class diagram is **consistent**, if it admits an instantiation, i.e., if its classes can be populated without violating any of the conditions imposed by the diagram.

Let $\Gamma$ be the set of FOL assertions corresponding to the UML Class Diagram.

Then, **the diagram is consistent** iff

$$\Gamma \text{ is satisfiable}$$

i.e., $\Gamma$ admits at least one model.

(Remember that FOL models cannot be empty.)

*Note:* Corresponding FOL reasoning task: **satisfiability**.
Class subsumption

A class $C_1$ **is subsumed by** a class $C_2$ (or $C_2$ subsumes $C_1$), if the class diagram implies that $C_2$ is a generalization of $C_1$.

Let $\Gamma$ be the set of FOL assertions corresponding to the UML Class Diagram, and $C_1(x), C_2(x)$ the predicates corresponding to the classes $C_1$, and $C_2$ of the diagram.

Then $C_1$ **is subsumed by** $C_2$ iff

$$\Gamma \models \forall x. C_1(x) \rightarrow C_2(x)$$

*Note:* Corresponding FOL reasoning task: **logical implication.**
Class subsumption: example

\[
\Gamma \models \forall x. \text{LatinLover}(x) \rightarrow \text{false}
\]

\[
\Gamma \models \forall x. \text{Italian}(x) \rightarrow \text{Lazy}(x)
\]
Class subsumption: another example (by E. Franconi)

$$\Gamma \models \forall x. \text{ItalianProf}(x) \rightarrow \text{LatinLover}(x)$$

*Note:* this is an example of reasoning by cases.
Two classes $C_1$ and $C_2$ are equivalent, if $C_1$ and $C_2$ denote the same set of instances in all instantiations of the class diagram.

Let $\Gamma$ be the set of FOL assertions corresponding to the UML Class Diagram, and $C_1(x)$, $C_2(x)$ the predicates corresponding to the classes $C_1$, and $C_2$ of the diagram.

Then $C_1$ and $C_2$ are equivalent iff

$$\Gamma \models \forall x. C_1(x) \leftrightarrow C_2(x)$$

**Note:**

- If two classes are equivalent then one of them is redundant.
- Determining equivalence of two classes allows for their merging, thus reducing the complexity of the diagram.
Class equivalence: example

\[ \Gamma \models \forall x. \text{LatinLover}(x) \rightarrow \text{false} \]
\[ \Gamma \models \forall x. \text{Italian}(x) \rightarrow \text{Lazy}(x) \]
\[ \Gamma \models \forall x. \text{Lazy}(x) \equiv \text{Italian}(x) \]
Implicit consequence

The properties of various classes and associations may interact to yield stricter multiplicities or typing than those explicitly specified in the diagram.

More generally . . .

A property $\mathcal{P}$ is an *(implicit) consequence* of a class diagram if $\mathcal{P}$ holds whenever all conditions imposed by the diagram are satisfied.

Let $\Gamma$ be the set of FOL assertion corresponding to the UML Class Diagram, and $\mathcal{P}$ (the formalization in FOL of) the property of interest.

Then $\mathcal{P}$ is an implicit consequence iff

$$\Gamma \models \mathcal{P}$$

i.e., the property $\mathcal{P}$ holds in every model of $\Gamma$.

*Note:* Corresponding FOL reasoning task: logical implication.
Implicit consequences: example

\[ \Gamma \models \forall x. \text{AdvCourse}(x_2) \rightarrow \sharp \{ x_1 \mid \text{gradAttends}(x_1, x_2) \} \leq 15 \]

\[ \Gamma \models \forall x. \text{GradStudent}(x) \rightarrow \text{Student}(x) \]

\[ \Gamma \not\models \forall x. \text{AdvCourse}(x) \rightarrow \text{Course}(x) \]
Due to the multiplicities, the classes `NaturalNumber` and `EvenNumber` are in bijection. As a consequence, in every instantiation of the diagram, “the classes `NaturalNumber` and `EvenNumber` contain the same number of objects”.

Due to the ISA relationship, every instance of `EvenNumber` is also an instance of `NaturalNumber`, i.e., we have that

$$ \Gamma \models \forall x. \text{EvenNumber}(x) \rightarrow \text{NaturalNumber}(x) $$
Unrestricted vs. finite model reasoning (cont’d)

Question: Does also the reverse implication hold, i.e.,

\[ \Gamma \models \forall x. \text{NaturalNumber}(x) \rightarrow \text{EvenNumber}(x) \]?

- if the domain is \textit{infinite}, the implication \textit{does not hold}.
- If the domain is \textit{finite}, the implication \textit{does hold}.

**Finite model reasoning:** means reasoning only with respect to models with a finite domain.

- Finite model reasoning is interesting for standard databases.
- The previous example shows that in UML Class Diagrams, finite model reasoning is \textit{different} form unrestricted model reasoning.
Questions

In the above examples reasoning could be easily carried out on intuitive grounds. However, two questions come up.

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

   - We cannot do so by directly relying on FOL!
   - But we can use specialized logics with better computational properties. A form of such specialized logics are **Description Logics**.

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

   - What is the worst-case situation?
   - Can we single out **interesting fragments** on which to reason efficiently?
Outline

1. Introduction to conceptual modeling
2. Formalizing UML Class Diagrams in FOL
3. Forms of reasoning on UML Class Diagrams
4. Brief overview of Description Logics
   - Description language
   - Description Logics ontologies
5. Reasoning on UML Class Diagrams using Description Logics
6. References
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.

Hence, DLs have a purpose similar to that of conceptual modeling formalisms.
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)
   \[
   \mathcal{T} = \{ \text{Father} \equiv \text{Human} \sqcap \text{Male} \sqcap \exists \text{hasChild}, \text{HappyFather} \subseteq \text{Father} \sqcap \forall \text{hasChild}.(\text{Doctor} \sqcup \text{Lawyer}) \}
   \]

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

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

1. Introduction to conceptual modeling
2. Formalizing UML Class Diagrams in FOL
3. Forms of reasoning on UML Class Diagrams
4. Brief overview of Description Logics
   - Description language
   - Description Logics ontologies
5. Reasoning on UML Class Diagrams using Description Logics
6. References
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

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

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

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

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

Many different DL constructs and their combinations have been investigated.
Outline

1. Introduction to conceptual modeling
2. Formalizing UML Class Diagrams in FOL
3. Forms of reasoning on UML Class Diagrams
4. Brief overview of Description Logics
   - Description language
   - Description Logics ontologies
5. Reasoning on UML Class Diagrams using Description Logics
6. References
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 $P \ C$)  \ldots

**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:**
  
  Father $\equiv$ Human $\cap$ Male $\cap$ $\exists$hasChild
  
  HappyFather $\sqsubseteq$ Father $\cap$ $\forall$hasChild.(Doctor $\sqcup$ Lawyer $\sqcup$ HappyPerson)
  
  HappyAnc $\sqsubseteq$ $\forall$descendant.HappyFather
  
  Teacher $\sqsubseteq$ $\neg$Doctor $\sqcap$ $\neg$Lawyer

- **Inclusion assertions on roles:**
  
  hasChild $\sqsubseteq$ descendant
  
  hasFather $\sqsubseteq$ hasChild$\neg$

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

**ABox membership assertions:**

- Teacher(mary), hasFather(mary, john), HappyAnc(john)
Semantics of a Description Logics ontology

The semantics is given by specifying when an interpretation $\mathcal{I}$ satisfies an assertion $\alpha$, 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 $R_1^\mathcal{I} \subseteq R_2^\mathcal{I}$.
- $\mathcal{I} \models (\text{prop } P)$ if $P^\mathcal{I}$ is a relation that has the property $\text{prop}$.
  (Note: domain and range assertions can be expressed by means of concept inclusion assertions.)

**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 $\alpha$, if it satisfies $\alpha$.
- 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 $\beta$ (where $\beta$ stands for an assertion, TBox, ABox, or ontology).
Outline

1. Introduction to conceptual modeling
2. Formalizing UML Class Diagrams in FOL
3. Forms of reasoning on UML Class Diagrams
4. Brief overview of Description Logics
5. Reasoning on UML Class Diagrams using Description Logics
   - Reducing reasoning in UML to reasoning in DLs
   - Reducing reasoning in DLs to reasoning in UML
6. References
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 $\mathcal{T}_D$.
  - one that associates to each DL TBox $\mathcal{T}$ a UML Class Diagram $\mathcal{D}_T$.
- The transformations are not model-preserving, but are based on a correspondence between instantiations of the Class Diagram and models of the associated 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

1. Introduction to conceptual modeling
2. Formalizing UML Class Diagrams in FOL
3. Forms of reasoning on UML Class Diagrams
4. Brief overview of Description Logics
5. Reasoning on UML Class Diagrams using Description Logics
   - Reducing reasoning in UML to reasoning in DLs
   - Reducing reasoning in DLs to reasoning in UML
6. References
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 \( a \):
    \[
    \exists a \sqsubseteq C \quad \exists a^\neg \sqsubseteq T
    \]
  - To encode the **multiplicity** \([m..n]\) of \( a \):
    \[
    C \sqsubseteq (\geq m a) \cap (\leq n a)
    \]
    - 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:
      \[
      C \sqsubseteq \exists a \cap (\leq 1 a)
      \]

**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**:

- \( \exists \text{number} \sqsubseteq \text{Phone} \)
- \( \exists \text{brand} \sqsubseteq \text{Phone} \)

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

- \( \text{Phone} \sqsubseteq \exists \text{number} \)
- \( \text{Phone} \sqsubseteq \exists \text{brand} \sqcap (\leq 1 \text{brand}) \)

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.

<table>
<thead>
<tr>
<th>Arity</th>
<th>Without association class</th>
<th>With association class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary</td>
<td>via a DL role</td>
<td>via reification</td>
</tr>
<tr>
<td>Non-binary</td>
<td>via reification</td>
<td>via reification</td>
</tr>
</tbody>
</table>

*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:

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

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 $C_2$:

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

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

$$C_2 \sqsubseteq (\geq \min_2 A^-) \cap (\leq \max_2 A^-)$$
Binary associations without association class – Example

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:

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

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):

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

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

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 $C_2$:

  $$C_1 \sqsubseteq (\geq \min_1 A^-_1) \cap (\leq \max_1 A^-_1)$$

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

  $$C_2 \sqsubseteq (\geq \min_2 A^-_2) \cap (\leq \max_2 A^-_2)$$
Associations with association class – Example

\[ \exists \text{place} \sqsubseteq \text{Origin} \]
\[ \text{Origin} \sqsubseteq \exists \text{place} \sqcap (\leq 1 \text{ place}) \]
\[ \exists \text{call} \sqsubseteq \text{Origin} \]
\[ \exists \text{from} \sqsubseteq \text{Origin} \]
\[ \text{Origin} \sqsubseteq \exists \text{call} \sqcap (\leq 1 \text{ call}) \sqcap \exists \text{from} \sqcap (\leq 1 \text{ from}) \]
\[ \text{PhoneCall} \sqsubseteq (\geq 1 \text{ call} \neg) \sqcap (\leq 1 \text{ call} \neg) \]

\[ \exists \text{place} \neg \sqsubseteq \text{String} \]
\[ \exists \text{call} \neg \sqsubseteq \text{PhoneCall} \]
\[ \exists \text{from} \neg \sqsubseteq \text{Phone} \]

Diego Calvanese (FUB)
Encoding of ISA and generalization

- When the generalization is **disjoint**:

  \[ C_i \subseteq \neg C_j \quad \text{for } 1 \leq i < j \leq k \]

- When the generalization is **complete**:

  \[ C \subseteq C_1 \sqcup \cdots \sqcup C_k \]
Encoding of ISA between associations

Without reification:

\[
\begin{array}{c}
C_1 \quad A \quad C_2 \\
\end{array}
\]

Role inclusion assertion: \( A' \sqsubseteq A \)

With reification:

\[
\begin{array}{c}
C_1 \quad A_1 \quad A_2 \quad C_2 \\
\end{array}
\]

Concept inclusion assert.: \( A' \sqsubseteq A \)

Role inclusion assertions: \( A'_1 \sqsubseteq A_1 \)
\( A'_2 \sqsubseteq A_2 \)
ISA and generalization – Example

ETACSpone ⊑ CellPhone
GSMphone ⊑ CellPhone
UMTSphone ⊑ CellPhone

ETACSpone ⊑ ¬GSMphone
ETACSpone ⊑ ¬UMTSphone
GSMphone ⊑ ¬UMTSphone

CellPhone ⊑ ETACSpone ⊔ GSMphone ⊔ UMTSphone
Encoding UML Class Diagrams in DLs – Example

Diego Calvanese (FUB)
Encoding UML Class Diagrams in DLs – Example 2

Manager ⊑ Employee
AreaManager ⊑ Manager
TopManager ⊑ Manager
AreaManager ⊑ ¬TopManager
Manager ⊑ AreaManager ⊔ TopManager

∃salary ∇ ⊑ Integer
∃salary ⊑ Employee
Employee ⊑ ∃salary ∩ (∈ 1 salary)

∃worksFor ∇ ⊑ Employee
∃worksFor ⊑ Project
Employee ⊑ ∃worksFor
Project ⊑ (∈ 3 worksFor ∇)
manages ⊑ worksFor

...
Reasoning on UML Class Diagrams using DLs

The encoding shows that we can reason over UML Class Diagrams as follows:

1. Transform the UML Class Diagram $D$ into a DL TBox $T_D$.
2. Use standard techniques to reason on $T_D$, and transfer the results to $D$.

Hence, reasoning over UML Class Diagrams:

- is computationally not harder than reasoning over ontologies in expressive DLs, i.e., in ExpTime.
- can be done effectively by using state-of-the-art reasoners for expressive DLs.

Questions

Is it actually necessary to make use of a formalism with such a high computational complexity?
Can we use instead an encoding in a computationally simpler formalism?

We will show in the following that this complexity is intrinsic in UML Class Diagrams, unless we pose some restrictions on the form of the diagram.
Outline

1. Introduction to conceptual modeling
2. Formalizing UML Class Diagrams in FOL
3. Forms of reasoning on UML Class Diagrams
4. Brief overview of Description Logics
5. Reasoning on UML Class Diagrams using Description Logics
   - Reducing reasoning in UML to reasoning in DLs
   - Reducing reasoning in DLs to reasoning in UML
6. References
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:

- $A \sqsubseteq B$
- $A \sqsubseteq \neg B$
- $A \sqsubseteq B_1 \sqcup B_2$
- $A \sqsubseteq \exists P . B$
- $A \sqsubseteq \forall P . B$
Encoding of inclusion and of disjointness

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

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

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

For each assertion $A \subseteq \exists P.B$ of $\mathcal{T}$, introduce the auxiliary class $C_{P_{AB}}$ and the associations $P_{AB1}$ and $P_{AB2}$, and add the following to $\mathcal{D_T}$:
For each assertion $A \subseteq \forall P.B$ of $\mathcal{T}$, introduce the auxiliary classes $\tilde{A}$, $C_{PAB}$, and $\overline{C}_{PAB}$, and the associations $P_{AB1}$, $P_{\overline{A}B1}$, and $P_{AB2}$, and add the following to $\mathcal{D}_T$: 
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 $\text{ExpTime}$-hard.
From this, we obtain:

**Theorem**

Reasoning over UML Class Diagrams is $\text{ExpTime}$-hard.
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., \( \text{ExpTime} \)-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; Artale et al., 2009]:
- \( \text{NLogSpace} \)-complete in combined complexity
- in \( \text{LogSpace} \) in data complexity
UML Class Diagrams for efficient access to data

We are interested in using UML Class Diagrams to specify ontologies that are suitable for accessing data efficiently, i.e., we are interested in using UML Class Diagrams for **ontology-based data access**:  
- Besides intensional reasoning, one is interested in **query answering**.  
- The relevant complexity measure is **data complexity**, i.e., the complexity measured in the size of the data only.

**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

Reasoning over extended ER models.

The DL-Lite family and relations.

[Baader et al., 2003] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors.

[Berardi et al., 2005] Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo.
Reasoning on UML class diagrams.