Outline

1. Ontology Languages
   - Elements of an ontology language
   - Intensional and extensional level of an ontology language
   - Ontologies vs. other formalisms
2. UML class diagrams as FOL ontologies
   - Approaches to conceptual modelling
   - Formalising UML class diagram in FOL
   - Reasoning on UML class diagrams

Outline

1. Ontology Languages
   - Elements of an ontology language
   - Intensional and extensional level of an ontology language
   - Ontologies vs. other formalisms

2. UML class diagrams as FOL ontologies
   - Approaches to conceptual modelling
   - Formalising UML class diagram in FOL
   - Reasoning on UML class diagrams

Outline

1. Ontology Languages
   - Elements of an ontology language
   - Intensional and extensional level of an ontology language
   - Ontologies vs. other formalisms
2. UML class diagrams as FOL ontologies
   - Approaches to conceptual modelling
   - Formalising UML class diagram in FOL
   - Reasoning on UML class diagrams

Elements of an ontology language

- Syntax
  - Alphabet
  - Languages constructs
  - Sentences to assert knowledge
- Semantics
  - Formal meaning
- Pragmatics
  - Intended meaning
  - Usage

Static vs. dynamic aspects

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

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

Before delving into the dynamic aspects, we need a good understanding of the static ones.
In this course, we concentrate essentially on the static aspects.

Outline

1. Ontology Languages
   - Elements of an ontology language
   - Intensional and extensional level of an ontology language
   - Ontologies vs. other formalisms

2. UML class diagrams as FOL ontologies
   - Approaches to conceptual modelling
   - Formalising UML class diagram in FOL
   - Reasoning on UML class diagrams

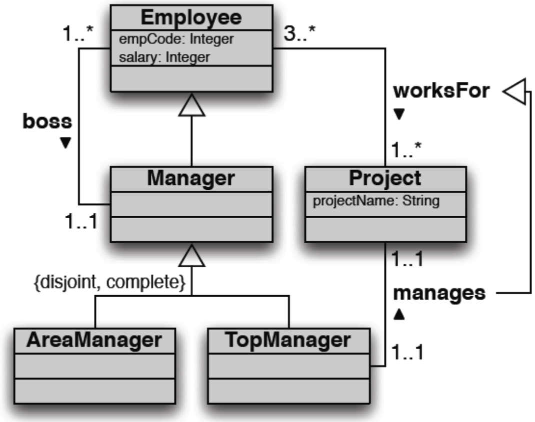
Intensional level of an ontology language

An ontology language for expressing the intensional level usually includes:

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

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

Example: ontology rendered as UML Class Diagram



Definition (Concept)

A concept is an element of an ontology that denotes a collection of instances (e.g., the set of "employees").

We distinguish between:

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

Concepts are also called classes, entity types, frames.

Properties

Definition (Property)

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

Property definition (intensional and extensional):

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

Properties are also called attributes, features, slots, data properties

Relationships

Definition (Relationship)

A relationship is an element of an ontology that expresses an association among concepts.

We distinguish between:

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

Relationships are also called associations, relationship types, roles, object properties.

Axioms

Definition (Axiom)

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

Different kinds of axioms/conditions:

- subclass relationships, e.g., $\mathsf{Manager} \sqsubseteq \mathsf{Employee}$
- equivalences, e.g., $\mathsf{Manager} \equiv \mathsf{AreaManager} \sqcup \mathsf{TopManager}$
- disjointness, e.g., $\mathsf{AreaManager} \sqcap \mathsf{TopManager} \equiv \perp$
- (cardinality) restrictions, e.g., each Employee worksFor at least 1 Project
- ...

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

Extensional level of an ontology language

At the extensional level we have individuals and facts:

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

Outline

1. Ontology Languages
   - Elements of an ontology language
   - Intensional and extensional level of an ontology language
   - Ontologies vs. other formalisms
2. UML class diagrams as FOL ontologies
   - Approaches to conceptual modelling
   - Formalising UML class diagram in FOL
   - Reasoning on UML class diagrams

Comparison with other formalisms

- Ontology languages vs. knowledge representation languages:
  - Ontologies are knowledge representation schemas.
- Ontology vs. logic:
  - Logic is the tool for assigning semantics to ontology languages.
- Ontology languages vs. conceptual data models:
  - Conceptual schemas are special ontologies, suited for conceptualizing a single logical model (database).
- Ontology languages vs. programming languages:
  - Class definitions are special ontologies, suited for conceptualizing a single structure for computation.

Classification of ontology languages

- Graph-based
  - Semantic networks
  - Conceptual graphs
  - UML class diagrams, Entity-Relationship diagrams
- Frame-based
  - Frame Systems
  - OKBC (Open Knowledge Base Connectivity), XOL (XML-based ontology language)
- Logic-based
  - Description Logics (e.g., $\mathcal{SHOIQ}$, $\mathcal{DLR}$, DL-Lite , OWL,...)
  - Rules (e.g., RuleML, LP/Prolog, F-Logic)
  - First-Order Logic (e.g., KIF)
  - Non-classical logics (e.g., non-monotonic, probabilistic)

Outline

1. Ontology Languages
   - Elements of an ontology language
   - Intensional and extensional level of an ontology language
   - Ontologies vs. other formalisms

2. UML class diagrams as FOL ontologies
   - Approaches to conceptual modelling
   - Formalising UML class diagram in FOL
   - Reasoning on UML class diagrams

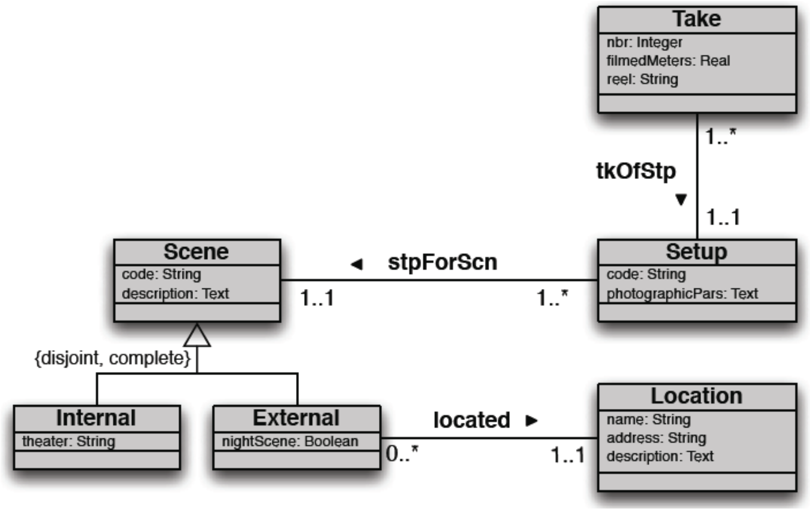
Outline

Let's start with an exercise

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.[a]

[a] Not really a bad point, in fact.

Use logic!!!

Alphabet

$Scene(x)$, $Setup(x)$, $Take(x)$, $Internal(x)$, $External(x)$, $Location(x)$, $stpForScn(x, y)$, $tkOfStp(x, y)$, $located(x, y)$,...

Axioms

$\forall x, y.code_{Scene}(x, y) \rightarrow Scene(x) \wedge String(y)$
$\forall x, y.description(x, y) \rightarrow Scene(x) \wedge Text(y)$
$\forall x, y.code_{Setup}(x, y) \rightarrow Setup(x) \wedge String(y)$
$\forall x, y.photographicPars(x, y) \rightarrow Setup(x) \wedge Text(y)$
$\forall x, y.nbr(x, y) \rightarrow Take(x) \wedge Integer(y)$
$\forall x, y.filmedMeters(x, y) \rightarrow Take(x) \wedge Real(y)$
$\forall x, y.reel(x, y) \rightarrow Take(x) \wedge String(y)$
$\forall x, y.theater(x, y) \rightarrow Internal(x) \wedge String(y)$
$\forall x, y.nightScene(x, y) \rightarrow External(x) \wedge Boolean(y)$
$\forall x, y.name(x, y) \rightarrow Location(x) \wedge String(y)$
$\forall x, y.address(x, y) \rightarrow Location(x) \wedge String(y)$
$\forall x, y.description(x, y) \rightarrow Location(x) \wedge Text(y)$
$\forall x.Scene(x) \rightarrow (1 \leq \sharp\{y | code_{Scene}(x, y)\} \leq 1)$
$\forall x.Internal(x) \rightarrow Scene(x)$
$\forall x.External(x) \rightarrow Scene(x)$
$\forall x.Internal(x) \rightarrow \neg External(x)$
$\forall x.Scene(x) \rightarrow Internal(x) \vee External(x)$

$\forall x, y.stpForScn(x, y) \rightarrow$
$Setup(x) \wedge Scene(y)$
$\forall x, y.tkOfStp(x, y) \rightarrow$
$Take(x) \wedge Setup(y)$
$\forall x, y.located(x, y) \rightarrow$
$External(x) \wedge Location(y)$
$\forall x.Setup(x) \rightarrow$
$(1 \leq \sharp\{y | stpForScn(x, y)\} \leq 1)$
$\forall y.Scene(y) \rightarrow$
$(1 \leq \sharp\{x | stpForScn(x, y))\}$
$\forall x.Take(x) \rightarrow$
$(1 \leq \sharp\{y | tkOfStp(x, y)\} \leq 1)$
$\forall x.Setup(y) \rightarrow$
$(1 \leq \sharp\{x | stpForScn(x, y)\})$
$\forall x.External(x) \rightarrow$
$(1 \leq \sharp\{y | located(x, y)\} \leq 1)$
...

Solution 2: Use logic - Discussion

Good points:

- Precise semantics.
- Formal verification.
- Allows for query answering.
- Machine comprehensible.
- Virtually unlimited expressiveness [a].

[a] Not necessarily a good point, in fact.

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.

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 value

- 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.

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.

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 reasoning.

Outline

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.
- Current version: 2.5.1 (December 2017): `http://www.omg.org/spec/UML/`
- 1999-today: de facto standard object-oriented modeling language.

References:

- Grady Booch, James Rumbaugh, Ivar Jacobson, "The unified modeling language user guide", Addison Wesley, 1999 (2nd ed., 2005)
- `http://www.omg.org/` → UML
- `http://www.uml.org/`

In this course 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).

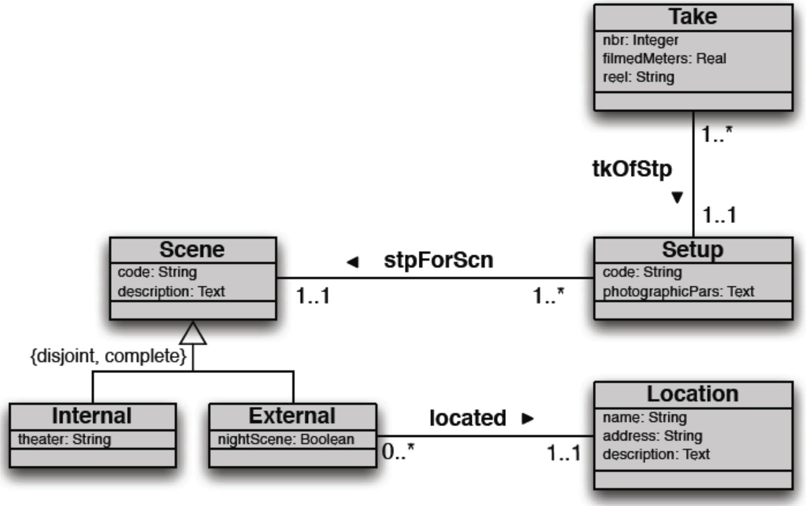
Note: This is exactly the goal of all conceptual modeling formalism, such as Entity-Relationship Diagrams (standard in Database design) or Ontologies.

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



Use of UML Class Diagrams

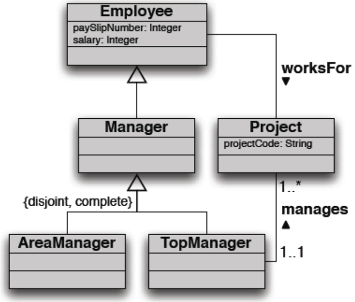
UML Class Diagrams are used in various phases of a software design:

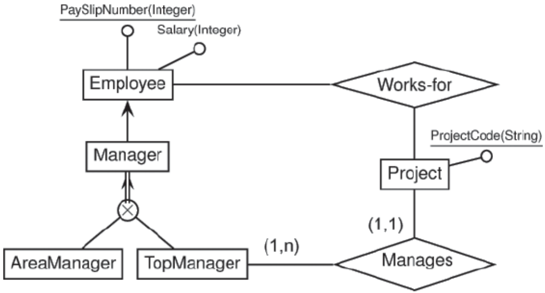
1. During the so-called analysis, where an abstract precise view of the domain of interest needs to be developed.
   $\rightsquigarrow$ the so-called "conceptual perspective".
2. During software development, to maintain an abstract view of the software to be developed.
   $\rightsquigarrow$ the so-called "implementation perspective".

In this course we focus on 1!

UML Class Diagrams and ER Schemas

UML class diagrams (when used for the conceptual perspective) closely resemble Entity-Relationship (ER) Diagrams.
Example of UML vs. ER:





Classes in UML

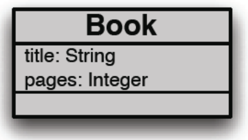
Definition (Class)

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



- the name of the class is 'Book'
- the class has two properties (attributes)

Classes in UML: instances

Definition (Instances)

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:

$$\{book_a, book_b, book_c, book_d, book_e\}$$
$$\{book_\alpha, book_\beta\}$$
$$\{book_1, book_2, book_3, \ldots, book_{500}, \ldots\}$$

Which is the actual instantiation?
We will know it only at run-time!!! - We are now at design time!

Classes in UML: formalization

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?

Definition (Class representation)

We represent a class as a FOL unary predicate!

Example

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

Definition (Association)

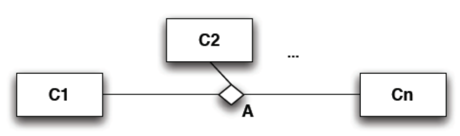
An association in UML models a relationship between two or more classes.

- 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

Associations: formalization

Definition (Association representation)



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 in the association:

$$\forall x_1, \ldots, x_n . A(x_1, \ldots, x_n) \rightarrow C_1(x_1) \wedge \cdots \wedge C_n(x_n)$$

Example

$$\forall x_1, x_2 . writtenBy(x_1, x_2) \rightarrow Book(x_1) \wedge Author(x_2)$$

Associations: multiplicity

Definition (Multiplicity Constraints)

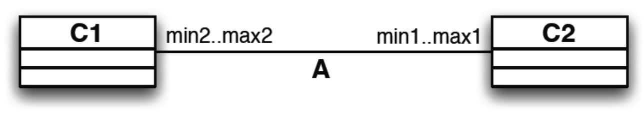
On binary associations, we can place multiplicity constraints, i.e., a minimal and maximal number of tuples in which every object participates as first (second) component.

Example

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

Definition (Multiplicity constraint representation)



Multiplicities of binary associations are easily expressible in FOL:

$$\forall x_1.C_1(x_1) \rightarrow (min_1 \leq \sharp\{x_2|A(x_1,x_2)\} \leq max_1)$$

$$\forall x_2.C_2(x_2) \rightarrow (min_2 \leq \sharp\{x_1|A(x_1,x_2)\} \leq max_2)$$

Example

$$\forall x.Book(x) \rightarrow (1 \leq \sharp\{y|written_{by}(x,y)\})$$

Note: this is an abreviation for a FOL formula expressing the cardinality of the set of possible values for y.

Expressing multiplicities in FOL

We use expressions $m \leq \sharp\{x|\varphi(x)\}$ and $\sharp\{x|\varphi(x)\} \leq n$ as abbreviations.

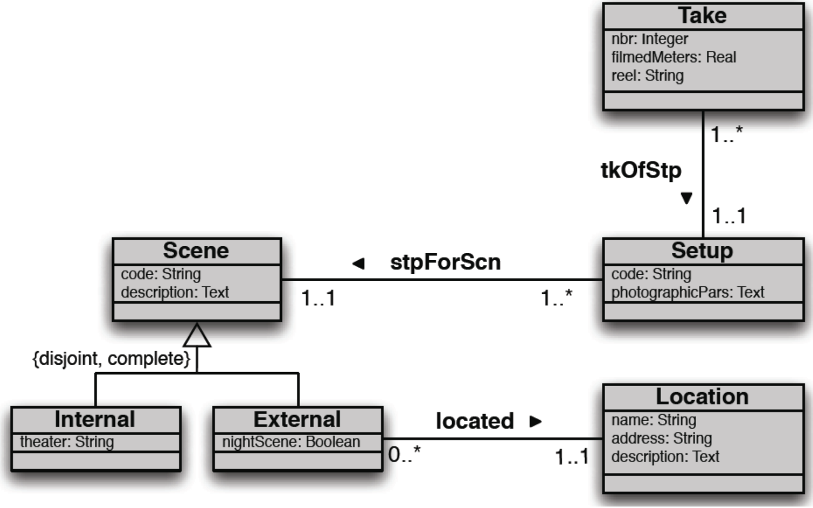
Minimum cardinality $m \leq \sharp\{x|\varphi(x)\}$

$$m \leq \sharp\{x|\varphi(x)\} = \exists x_1, \dots, x_m.(\varphi(x_1) \wedge \dots \wedge \varphi(x_m) \wedge \bigwedge_{\substack{1 \leq i < m \\ i < j \leq m}} x_i \neq x_j)$$

Maximum cardinality $\sharp\{x|\varphi(x)\} \leq n$

$$\sharp\{x|\varphi(x)\} \leq n = \forall x_1, \dots, x_n, x_{n+1}.((\varphi(x_1) \wedge \dots \wedge \varphi(x_{n+1}) \rightarrow \bigvee_{\substack{1 \leq i < n \\ i < j \leq n+1}} x_i = x_j)$$

Note: We need FOL with equality


Take
nbr: Integer
filmedMeters: Real
reel: String
1..*
tkOfStp
1..1
Setup
code: String
photographicPars: Text
stpForScn
1..1
1..*
Scene
code: String
description: Text
{disjoint, complete}
Internal
theater: String
External
nightScene: Boolean
located
0..*
1..1
Location
name: String
address: String
description: Text

In our example...

Alphabet

$Scene(x)$, $Setup(x)$, $Take(x)$, $Internal(x)$, $External(x)$, $Location(x)$, $stpForScn(x, y)$, $tkOfStp(x, y)$, $located(x, y)$,...

Axioms

$$
\begin{array}{ll}
\forall x, y.code_{Scene}(x, y) \rightarrow Scene(x) \wedge String(y) & \forall x, y.stpForScn(x, y) \rightarrow \\
\forall x, y.description(x, y) \rightarrow Scene(x) \wedge Text(y) & Setup(x) \wedge Scene(y) \\
\forall x, y.code_{Setup}(x, y) \rightarrow Setup(x) \wedge String(y) & \forall x, y.tkOfStp(x, y) \rightarrow \\
\forall x, y.photographicPars(x, y) \rightarrow Setup(x) \wedge Text(y) & Take(x) \wedge Setup(y) \\
\forall x, y.nbr(x, y) \rightarrow Take(x) \wedge Integer(y) & \forall x, y.located(x, y) \rightarrow \\
\forall x, y.filmedMeters(x, y) \rightarrow Take(x) \wedge Real(y) & External(x) \wedge Location(y) \\
\forall x, y.reel(x, y) \rightarrow Take(x) \wedge String(y) & \forall x.Setup(x) \rightarrow \\
\forall x, y.theater(x, y) \rightarrow Internal(x) \wedge String(y) & (1 \leq \sharp\{y|stpForScn(x, y)\} \leq 1) \\
\forall x, y.nightScene(x, y) \rightarrow External(x) \wedge Boolean(y) & \forall y.Scene(y) \rightarrow \\
\forall x, y.name(x, y) \rightarrow Location(x) \wedge String(y) & (1 \leq \sharp\{x|stpForScn(x, y))\} \\
\forall x, y.address(x, y) \rightarrow Location(x) \wedge String(y) & \forall x.Take(x) \rightarrow \\
\forall x, y.description(x, y) \rightarrow Location(x) \wedge Text(y) & (1 \leq \sharp\{y|tkOfStp(x, y)\} \leq 1) \\
\forall x.Scene(x) \rightarrow (1 \leq \sharp\{y|code_{Scene}(x, y)\} \leq 1) & \forall x.Setup(y) \rightarrow \\
\forall x.Internal(x) \rightarrow Scene(x) & (1 \leq \sharp\{x|stpForScn(x, y)\}) \\
\forall x.External(x) \rightarrow Scene(x) & \forall x.External(x) \rightarrow \\
\forall x.Internal(x) \rightarrow \neg External(x) & (1 \leq \sharp\{y|located(x, y)\} \leq 1) \\
\forall x.Scene(x) \rightarrow Internal(x) \vee External(x) & \ldots
\end{array}
$$

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 faction)

Definition (In FOL)

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

Attributes

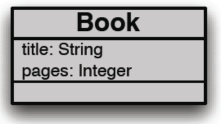
Definition (Attribute)

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



-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.

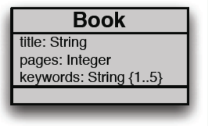
Example

$book_3$ 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



-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

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

Definition (Attribute representation)

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:

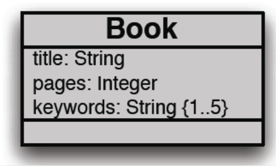
$$\forall x, y.att_C(x, y) \rightarrow C(x) \wedge T(y)$$

- An assertion for the multiplicity:

$$\forall x.C(x) \rightarrow (i \leq \sharp\{y|att_C(x, y)\} \leq j)$$

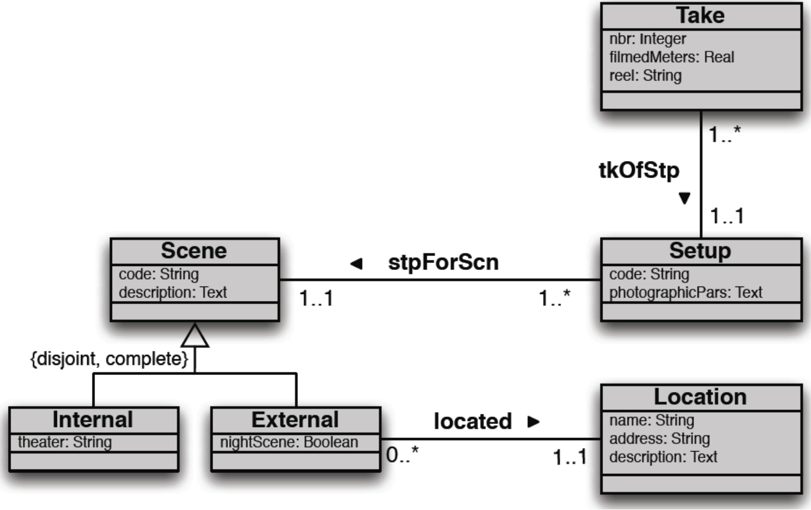
Attributes: example of formalization

Example



$$\forall x, y. title_B(x,y) \rightarrow Book(x) \wedge String(y)$$
$$\forall x. Book(x) \rightarrow (1 \leq \sharp\{y | title_B(x,y)\} \leq 1)$$
$$\forall x, y. pages_B(x,y) \rightarrow Book(x) \wedge Integer(y)$$
$$\forall x. Book(x) \rightarrow (1 \leq \sharp\{y | pages_B(x,y)\} \leq 1)$$
$$\forall x, y. keywords_B(x,y) \rightarrow Book(x) \wedge String(y)$$
$$\forall x. Book(x) \rightarrow (1 \leq \sharp\{y | keywords_B(x,y)\} \leq 5)$$

In our example...



In our Example

Alphabet

$Scene(x)$, $Setup(x)$, $Take(x)$, $Internal(x)$, $External(x)$, $Location(x)$, $stpForScn(x,y)$, $tkOfStp(x,y)$, $located(x,y)$,...

Axioms

$\forall x, y.code_{Scene}(x,y) \rightarrow Scene(x) \wedge String(y)$
$\forall x, y.description(x,y) \rightarrow Scene(x) \wedge Text(y)$
$\forall x, y.code_{Setup}(x,y) \rightarrow Setup(x) \wedge String(y)$
$\forall x, y.photographicPars(x,y) \rightarrow Setup(x) \wedge Text(y)$
$\forall x, y.nbr(x,y) \rightarrow Take(x) \wedge Integer(y)$
$\forall x, y.filmedMeters(x,y) \rightarrow Take(x) \wedge Real(y)$
$\forall x, y.reel(x,y) \rightarrow Take(x) \wedge String(y)$
$\forall x, y.theater(x,y) \rightarrow Internal(x) \wedge String(y)$
$\forall x, y.nightScene(x,y) \rightarrow External(x) \wedge Boolean(y)$
$\forall x, y.name(x,y) \rightarrow Location(x) \wedge String(y)$
$\forall x, y.address(x,y) \rightarrow Location(x) \wedge String(y)$
$\forall x, y.description(x,y) \rightarrow Location(x) \wedge Text(y)$
$\forall x.Scene(x) \rightarrow (1 \leq \sharp\{y|code_{Scene}(x,y)\} \leq 1)$
$\forall x.Internal(x) \rightarrow Scene(x)$
$\forall x.External(x) \rightarrow Scene(x)$
$\forall x.Internal(x) \rightarrow \neg External(x)$
$\forall x.Scene(x) \rightarrow Internal(x) \vee External(x)$

$\forall x, y.stpForScn(x,y) \rightarrow$
$Setup(x) \wedge Scene(y)$
$\forall x, y.tkOfStp(x,y) \rightarrow$
$Take(x) \wedge Setup(y)$
$\forall x, y.located(x,y) \rightarrow$
$External(x) \wedge Location(y)$
$\forall x.Setup(x) \rightarrow$
$(1 \leq \sharp\{y|stpForScn(x,y)\} \leq 1)$
$\forall y.Scene(y) \rightarrow$
$(1 \leq \sharp\{x|stpForScn(x,y))\}$
$\forall x.Take(x) \rightarrow$
$(1 \leq \sharp\{y|tkOfStp(x,y)\} \leq 1)$
$\forall x.Setup(y) \rightarrow$
$(1 \leq \sharp\{x|stpForScn(x,y)\})$
$\forall x.External(x) \rightarrow$
$(1 \leq \sharp\{y|located(x,y)\} \leq 1)$
$\ldots$

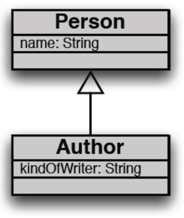
ISA and generalizations

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'.

Generalization

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

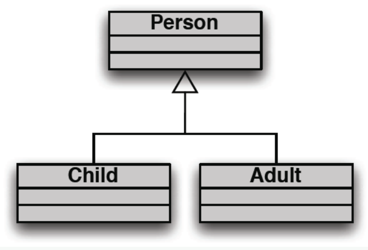
Example



The attibute 'name' is inherited by 'Author'.

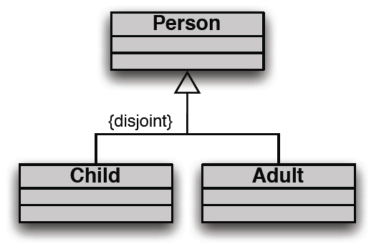
A generalization involves a superclass (base class) and one or more subclasses: every instance of each subclass is also an instance of the superclass.

Example



The ability of having more subclasses in the same generalization, allows for placing suitable constraints on the classes involved in the generalization.

Example

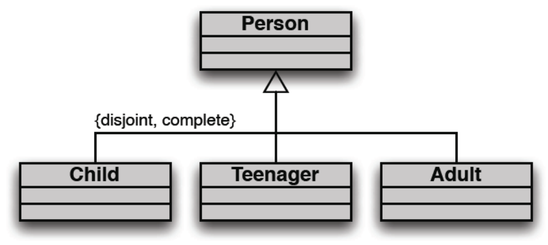


Generalizations with constraints (cont'd)

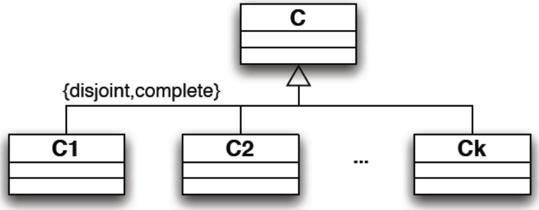
Most notable and used constraints:

- 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



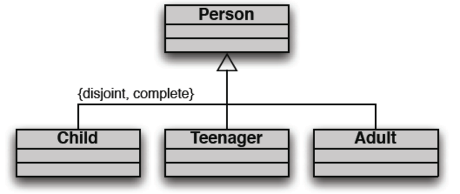
Generalizations: formalization



Definition (Generalization representation)

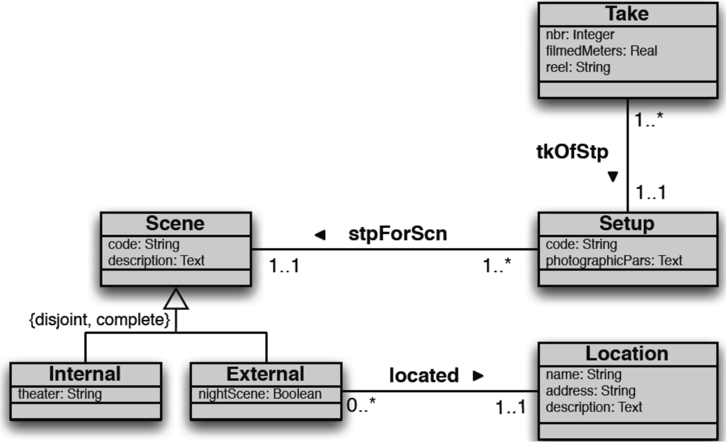
ISA:	$\forall x.C_i(x) \rightarrow C(x)$,	for $1 \leq i \leq k$
Disjointness:	$\forall x.C_i(x) \rightarrow \neg C_j(x)$,	for $1 \leq i < j \leq k$
Completeness:	$\forall x.C(x) \rightarrow \bigvee_{i=1}^{k} C_i(x)$	

Example



$$\forall x.Child(x) \to Person(x)$$
$$\forall x.Teenager(x) \to Person(x)$$
$$\forall x.Adult(x) \to Person(x)$$
$$\forall x.Child(x) \to \neg Teenager(x)$$
$$\forall x.Child(x) \to \neg Adult(x)$$
$$\forall x.Teenager(x) \to \neg Adult(x)$$
$$\forall x.Person(x) \to (Child(x) \lor Teenager(x) \lor Adult(x))$$

In our example ...



In our Example

Alphabet

$Scene(x)$, $Setup(x)$, $Take(x)$, $Internal(x)$, $External(x)$, $Location(x)$, $stpForScn(x, y)$, $tkOfStp(x, y)$, $located(x, y)$,...

Axioms

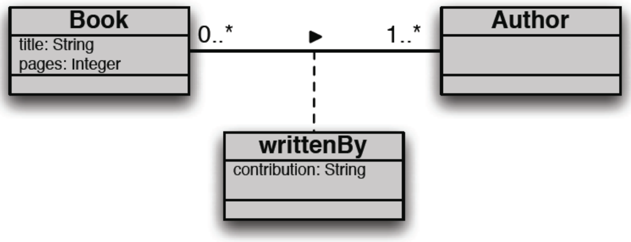
$\forall x, y.code_{Scene}(x, y) \to Scene(x) \land String(y)$
$\forall x, y.description(x, y) \to Scene(x) \land Text(y)$
$\forall x, y.code_{Setup}(x, y) \to Setup(x) \land String(y)$
$\forall x, y.photographicPars(x, y) \to Setup(x) \land Text(y)$
$\forall x, y.nbr(x, y) \to Take(x) \land Integer(y)$
$\forall x, y.filmedMeters(x, y) \to Take(x) \land Real(y)$
$\forall x, y.reel(x, y) \to Take(x) \land String(y)$
$\forall x, y.theater(x, y) \to Internal(x) \land String(y)$
$\forall x, y.nightScene(x, y) \to External(x) \land Boolean(y)$
$\forall x, y.name(x, y) \to Location(x) \land String(y)$
$\forall x, y.address(x, y) \to Location(x) \land String(y)$
$\forall x, y.description(x, y) \to Location(x) \land Text(y)$
$\forall x.Scene(x) \to (1 \leq \sharp\{y | code_{Scene}(x, y)\} \leq 1)$
$\forall x.Internal(x) \to Scene(x)$
$\forall x.External(x) \to Scene(x)$
$\forall x.Internal(x) \to \neg External(x)$
$\forall x.Scene(x) \to Internal(x) \lor External(x)$

$\forall x, y.stpForScn(x, y) \to$
$Setup(x) \land Scene(y)$
$\forall x, y.tkOfStp(x, y) \to$
$Take(x) \land Setup(y)$
$\forall x, y.located(x, y) \to$
$External(x) \land Location(y)$
$\forall x.Setup(x) \to$
$(1 \leq \sharp\{y | stpForScn(x, y)\} \leq 1)$
$\forall y.Scene(y) \to$
$(1 \leq \sharp\{x | stpForScn(x, y))\}$
$\forall x.Take(x) \to$
$(1 \leq \sharp\{y | tkOfStp(x, y)\} \leq 1)$
$\forall x.Setup(y) \to$
$(1 \leq \sharp\{x | stpForScn(x, y)\})$
$\forall x.External(x) \to$
$(1 \leq \sharp\{y | located(x, y)\} \leq 1)$
$\ldots$

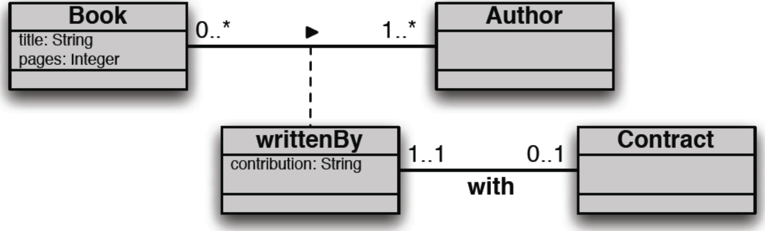
Sometimes we may want to assert properties of associations. In UML to do so we resort to association classes:

- 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).

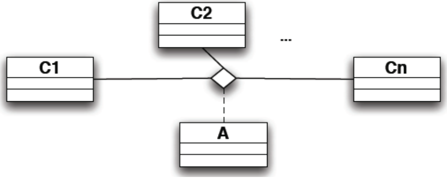
Association class - Example



Association class - Example (cont'd)



Association classes: formalization

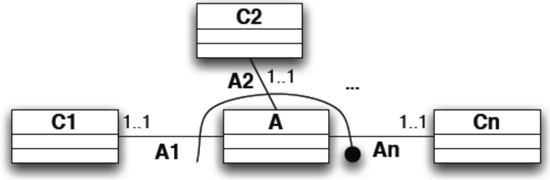


Definition (Reification)

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)



Definition

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

$$\forall x, y. A_i(x, y) \rightarrow A(x) \wedge C_i(y), \qquad for\ i \in \{1, \dots, n\}$$

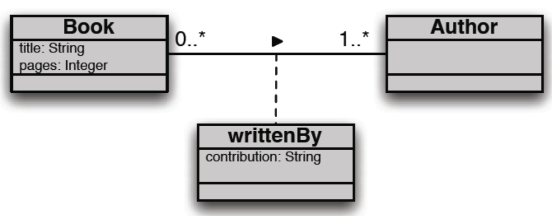
$$\forall x. A(x) \rightarrow \exists y. A_i(x, y), \qquad for\ i \in \{1, \dots, n\}$$

$$\forall x, y, y'. A_i(x, y) \wedge A_i(x, y') \rightarrow y = y', \qquad for\ i \in \{1, \dots, n\}$$

$$\forall x, x', y_1, \dots, y_n. \bigwedge_{i=1}^{n} (A_i(x, y_i) \wedge A_i(x', y_i)) \rightarrow x = x'$$

Association classes: example of formalization

Example



$$\forall x, y.wb_1(x, y) \to writtenBy(x) \land Book(y)$$
$$\forall x, y.wb_2(x, y) \to writtenBy(x) \land Author(y)$$
$$\forall x.writtenBy(x) \to \exists y.wb_1(x, y)$$
$$\forall x.writtenBy(x) \to \exists y.wb_2(x, y)$$
$$\forall x, y, y'.wb_1(x, y) \land wb_1(x, y') \to y = y'$$
$$\forall x, y, y'.wb_2(x, y) \land wb_2(x, y') \to y = y'$$
$$\forall x, x', y_1, y_2.wb_1(x, y_1) \land wb_1(x', y_1) \land wb_2(x, y_2) \land wb_2(x', y_2) \to x = x'$$

Outline

Forms of reasoning: class consistency

Definition (Class Consistency)

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

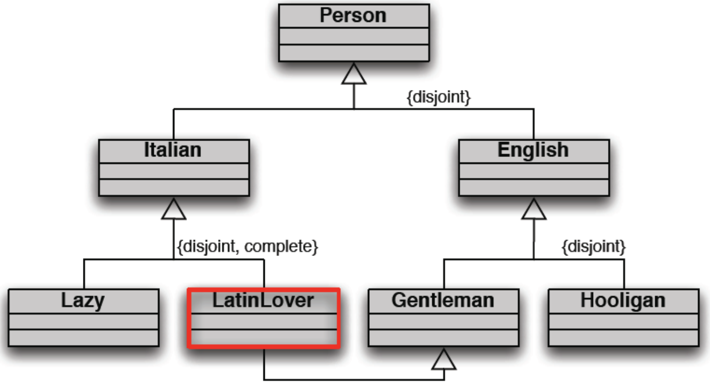
Theorem

Let Γ 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 Γ in which the extension of $C(x)$ is not the empty set.

Note: Corresponding FOL reasoning task: satisfiability.



$$\Gamma \models \forall x.LatinLover(x) \rightarrow false$$

Forms of reasoning: whole diagram consistency

Definition (Class 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.

Theorem

Let Γ be the set of FOL assertions corresponding to the UML Class Diagram. Then, the diagram is consistent iff

$$\Gamma \textit{ is satisfiable}$$

i.e., Γ admits at least one model. (Remember that FOL models cannot be empty.)

Note: Corresponding FOL reasoning task: satisfiability.

Forms of reasoning: class subsumption

Definition (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.

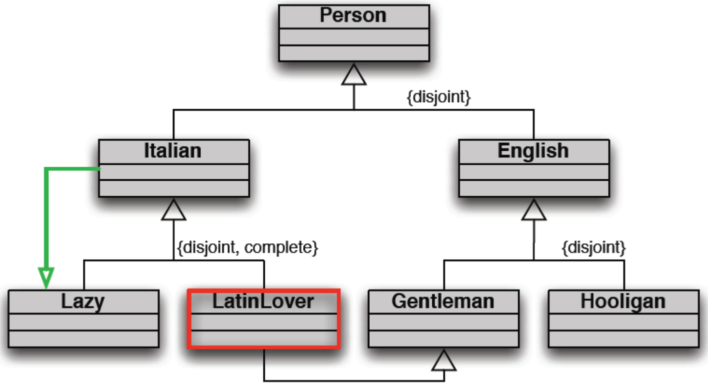
Theorem

Let Γ 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

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

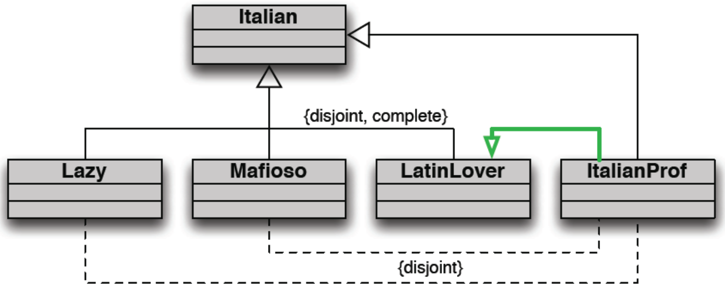
Note: Corresponding FOL reasoning task: logical implication.

Class subsumption: example



$$\Gamma \models \forall x. LatinLover(x) \rightarrow false$$

$$\Gamma \models \forall x. Italian(x) \rightarrow Lazy(x)$$



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

Note: this is an example of reasoning by cases.

Forms of reasoning: class equivalence

Definition (Class Equivalence)

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.

Theorem

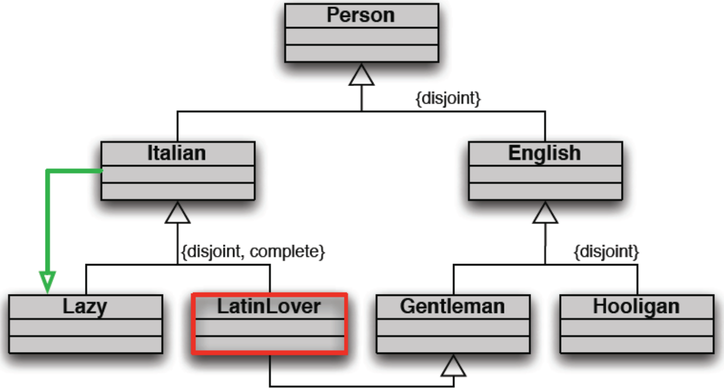
Let Γ be the set of FOL assertions corresponding to the UML Class Diagram, and $C_1(x), C_2(x)$ the predicates corresponding to the classes C_1, and C_2 of the diagram. Then C_1 and C_2 are equivalent iff

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

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



$$\Gamma \models \forall x.ItalianLover(x) \rightarrow false$$

$$\Gamma \models \forall x.Italian(x) \rightarrow Lazy(x)$$

$$\Gamma \models \forall x.Lazy(x) \equiv Italian(x)$$

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

Definition (Implicit Consequence)

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.

Theorem

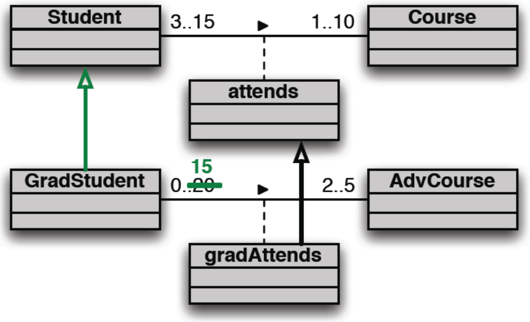
Let Γ be the set of FOL assertions corresponding to the UML Class Diagram, and $\mathcal{P}$ (the formalization in FOL of) the property of interest
Then $\mathcal{P}$ is an implicit consequence iff

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

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

Note: Corresponding FOL reasoning task: logical implication.

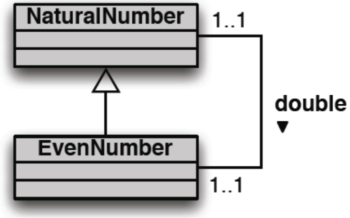
Implicit consequences: example



$\Gamma \models \forall x. AdvCourse(x_2) \rightarrow \sharp\{x_1 | gradAttends(x_1, x_2)\} \leq 15$

$\Gamma \models \forall x. GradStudent(x) \rightarrow Student(x)$

$\Gamma \not\models \forall x. AdvCourse(x) \rightarrow Course(x)$



- 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

$$\Gamma \models \forall x.EvenNumber(x) \rightarrow NaturalNumber(x)$$

Unrestricted vs. finite model reasoning (cont'd)

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

$$\Gamma \models \forall x.NaturalNumber(x) \rightarrow EvenNumber(x) \quad ?$$

- if the domain is infinite, the implication does not hold.
- If the domain is finite, the implication 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 different from 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?

Note: all what we have said holds for Entity-Relationship Diagrams as well