Ontology and Database Systems: Knowledge Representation and Ontologies

Part 1: Modeling Information through Ontologies

Diego Calvanese

Faculty of Computer Science
European Master in Computational Logic

A.Y. 2014/2015

unibz
Fakultät für Informatik
Facoltà di Scienze e Tecnologie informatiche
Faculty of Computer Science

Part 1

Modeling Information through Ontologies

Outline of Part 1

unibz

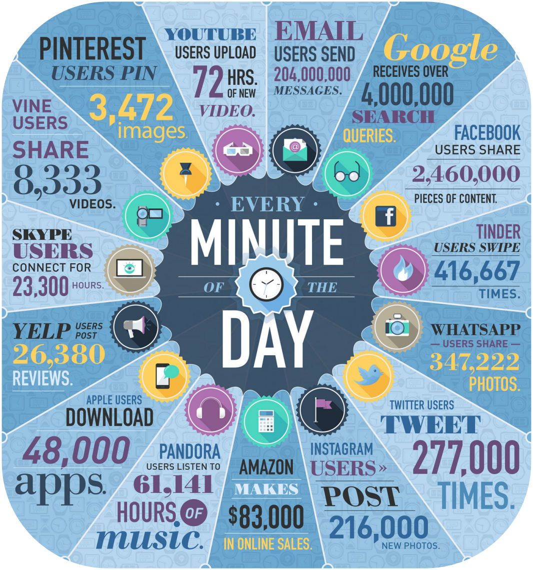
Outline of Part 1

unibz

Outline of Part 1

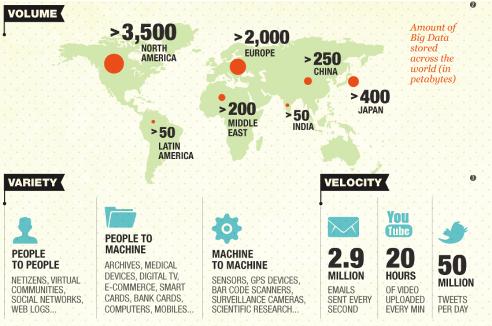
unibz

Data never sleeps



Big data

As firms move from siloed, transaction-oriented systems to more integrated, socially aware ones, they will face challenges related to customer data.
Big data is characterized by increases in data **volume**, **velocity**, **variety**, and **variability**. To improve customer engagement, companies must invest in solutions to effectively manage big data.
[Forrester Research, Inc. June 1, 2012]



unibz

New challenges in information management

Information management is a key challenge in complex systems today:

- The **volume** of information to manage is enormous.
- Data increases with incredible **velocity**.
- The **variety** of information has increased:
  - structured vs. semi-structured vs. unstructured
  - data is distributed and heterogeneous
  - human-processable vs. machine processable
- The meaning of data is **variable**, and depends on the context.
- The **veracity** of the data needs to be questioned and assessed – incompleteness, inconsistency, lack of precision.
- To understand complex data it needs to be **visualized**.
- Data increasingly represents an important **value** for an organization.

There is an increased need to access data in a uniform and integrated way, extract information, and perform various forms of analysis on it.

Traditional data management systems are not sufficient anymore to fulfill today's information management requirements.

Example 1: Statoil Exploration

Experts in geology and geophysics develop stratigraphic models of unexplored areas on the basis of data acquired from previous operations at nearby geographical locations.

Facts:

- 1,000 TB of relational data
- using diverse schemata
- spread over 2,000 tables, over multiple individual data bases

Data Access for Exploration:

- 900 experts in Statoil Exploration.
- up to 4 days for new data access queries, requiring assistance from IT-experts.
- 30–70% of time spent on data gathering.

Example 2: Siemens Energy Services

Runs service centers for power plants, each responsible for remote monitoring and diagnostics of many thousands of gas/steam turbines and associated components. When informed about potential problems, diagnosis engineers access a variety of raw and processed data.

Facts:

- several TB of time-stamped sensor data
- several GB of event data ("alarm triggered at time T")
- data grows at 30GB per day (sensor data rate 1Hz–1kHz)

Service Requests:

- over 50 service centers worldwide
- 1,000 service requests per center per year
- 80% of time per request used on data gathering

Addressing information management challenges

Several efforts come from the area of **databases**:

- New kinds of databases are studied:
  - XML databases, graph databases
  - column stores
  - probabilistic databases
  - ...
- Information integration
  - Represents one of the major challenges for the future or IT.
  - E.g., the market for information integration software has been growing at a steady rate of +9% per year since 2007.
  - The overall market value of such software was $4 billion in 2012.

unibz

The role of Knowledge Representation in AI

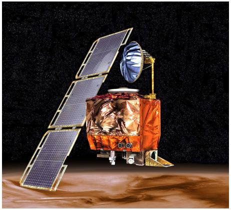
Management of complex kinds of information has traditionally been the concern of **Knowledge Representation** (KR) in AI:

- Research in AI and KR can bring important insights, solutions, techniques, and technologies: concern on **variability** and **veracity**
- However, the other v's have not received the proper attention so far: **volume**, **velocity**, **variety**.

The techniques and tools developed in KR need to be **adapted** and **extended** to address the new challenges coming from today's requirements for information management.

Emphasis is on the **semantics** of data!

- Fundamental for understanding, sharing, and reasoning.
- Example: Mars climate orbiter case in 1999: 327.6M $ lost because of a metric mixup: same data, different interpretations!



Mars Climate Orbiter 2 by NASA, JPL, Corby Waste.

unibz

Ontologies

Ontologies in Computer Science

An **ontology** is a representation scheme describing a **formal conceptualization** of a domain of interest.

The specification of an ontology usually comprises two distinct levels:

- **Intensional level**: specifies a set of conceptual elements and of constraints/axioms describing the conceptual structures of the domain.
- **Extensional level**: specifies a set of instances of the conceptual elements described at the intensional level.

Note: an ontology may contain also a **meta-level**, which specifies a set of modeling categories of which the conceptual elements are instances.

unibz

Description logic ontologies

The formal foundations for ontology languages are in logic, and specifically in description logics.

Description logics [Baader *et al.*, 2003] are fragments of first-order logic specifically tailored towards the representation of structured knowledge.

- By grounding the used formalisms in logic, the information is provided with a **formal semantics** (i.e., a meaning).
- The logic-based formalization allows one to provide **automated support** for tasks related to data management, by means of **logic-based inference**.
- **Computational aspects** are of concern, so that **tools** can provide **effective support** for automated reasoning.

In this course:

we are looking into using description logics for data management.

Outline of Part 1

unibz

Conceptual schemas in information systems

Intensional information has traditionally played an important role in information systems.

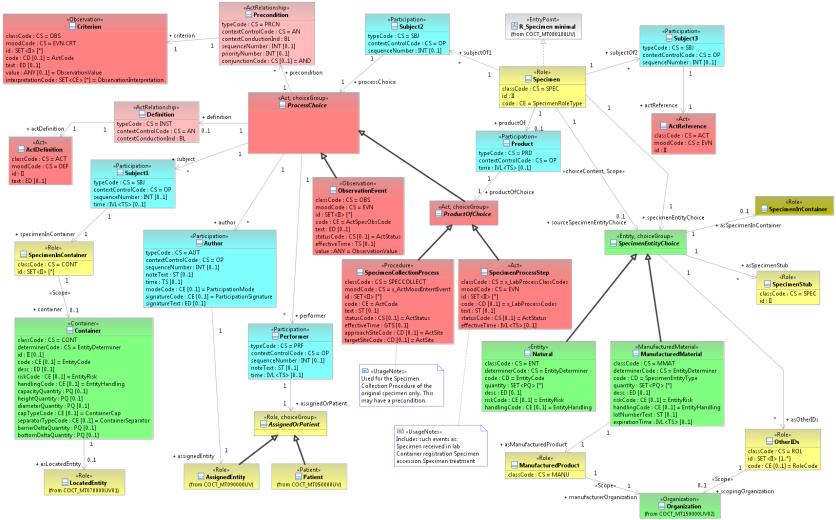
Design phase of the information system:

1. **Conceptual modeling**: from the requirements, a **conceptual schema** of the domain of interest is produced.
2. The conceptual schema is used to produce the logical data schema.
3. The data are stored according to the logical schema, and queried through it.

Conceptual Modeling

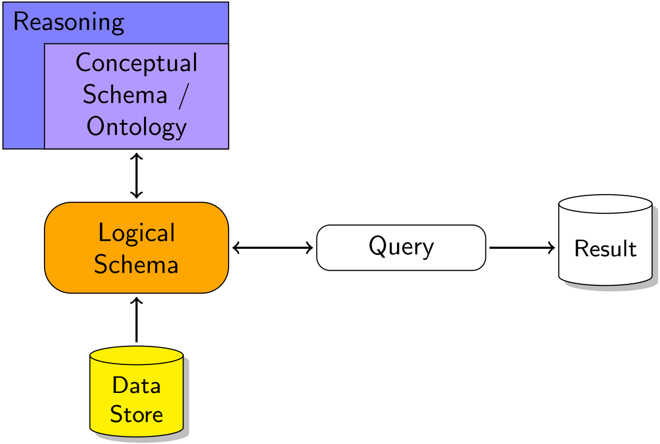
The activity of formally describing some aspects of the physical and social world around us for the purposes of understanding and communication.
[John Mylopoulos]

Managing complexity ...



unibz

Conceptual schemas used at design-time



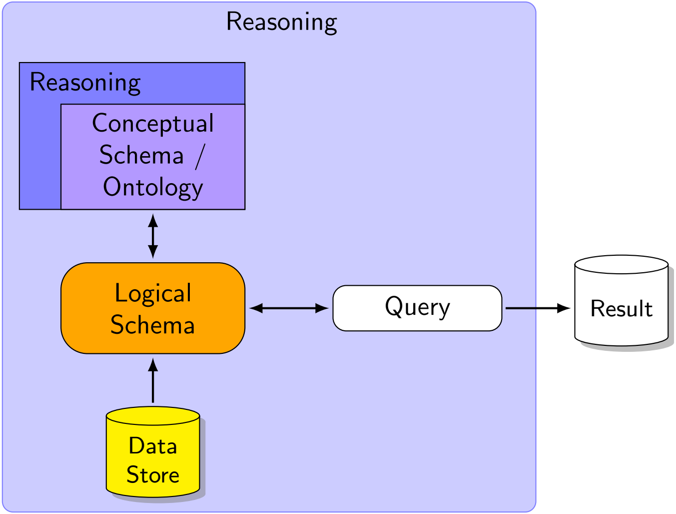
Ontologies in information systems

The role of ontologies in information systems goes beyond that of conceptual schemas.

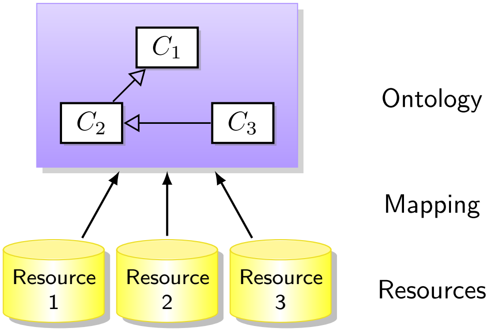
Ontologies affect the whole life-cycle of the information system:

- Ontologies, with the associated reasoning capabilities and inference tools, can provide support at design time.
- The use of ontologies can significantly simplify **maintenance** of the information system's data assets.
- The ontology is used also to support the interaction with the information system, i.e., at run-time.
  $\leadsto$ **Reasoning** to take into account the constraints coming from the ontology has to be **done at run-time**.

Ontologies used at run-time



Ontologies at the core of information systems



The usage of all system resources (data and services) is done through the domain conceptualization.

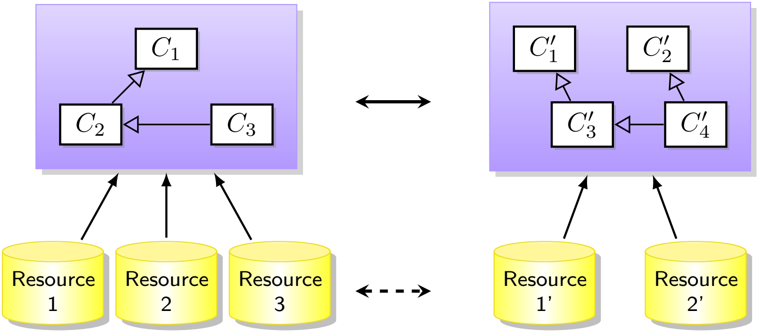
Ontology mediated access to data

Desiderata: achieve **logical transparency** in access to data:

- **Hide** to the user where and how data are stored.
- Present to the user a **conceptual view** of the data.
- Use a **semantically rich formalism** for the conceptual view.

This setting is similar to the one of Data Integration. The difference is that here the ontology provides a rich conceptual description of the data managed by the system.

Ontologies at the core of cooperation



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

Outline of Part 1

unibz

Issues in ontology-based information management

1. Choice of the formalisms to adopt
2. Efficiency and scalability
3. Tool support

unibz

Issue 1: Formalisms to adopt

1. Which is the right ontology language?
   - many proposals have been made
   - differ in expressive power and in complexity of inference
2. Which languages should we use for querying?
   - requirements for querying are different from those for modeling
3. How do we connect the ontology to available information sources?
   - mismatch between information in an ontology and data in a data source

In this course:

- We present and discuss variants of ontology languages, and study their logical and computational properties.
- We study the problem of querying data through ontologies.
- We discuss problems and solutions related to the impedance mismatch between ontologies and data sources.

Issue 2: Efficiency and scalability

- How can we handle large ontologies?
  - We have to take into account the **tradeoff** between **expressive power** and **complexity** of inference.
- How can we cope with large amounts of data?
  - What may be good for large ontologies, may not be good enough for large amounts of data.
- Can we handle multiple data sources and/or multiple ontologies?

In this course:

- We discuss in depth the above mentioned **tradeoff**.
- We will also pay attention to the aspects related to **data management**.
- We do not deal with the problem of integrating multiple information sources. This is typically addressed in *Information Integration*, to which many of the considerations we make also apply.

Issue 3: Tools

- According to the principle that "there is no meaning without a language with a formal semantics", the formal semantics becomes the solid basis for dealing with ontologies.
- Hence every kind of access to an ontology (to extract information, to modify it, etc.), requires to **fully** take into account its semantics.
- We need tools that perform reasoning over the ontology that is **sound and complete** wrt the semantics.
- The tools have to be as "efficient" as possible.

In this course:

- We discuss the requirements, the principles, and the theoretical foundations for ontology inference tools.
- We also present and use a tool for querying data sources through ontologies that has been built according to those principles.

Outline of Part 1

1. Introduction to ontologies
2. Using logic for representing knowledge
   - Language, real world, and mathematical model
   - Logical language
   - Interpretation of a logical language
   - Logical consequence
   - Inference methods
3. Ontology languages
4. UML Class Diagrams as FOL ontologies
5. References

unibz

Outline of Part 1

unibz

What is a logic?

- The main objective of a logic (there is not a unique logic but many) is to express by means of a **formal language** the knowledge about certain phenomena or a certain portion of the world.
- The language of a logic is **formal**, since it is equipped with:
  - a formal **syntax**: it tells one how to write statements in the logic;
  - a formal **semantics**: it tells one what the meaning of these statements is.
- Considering the formal semantics, one can **reason** over given knowledge, and show which knowledge is a **logical consequence** of the given one.
- A logic often allows one to encode with a precise set of deterministic rules, called **inference rules**, the basic reasoning steps that are considered to be correct by everybody (according to the semantics of the logic).
- By concatenating applications of simple inference rules, one can construct logically correct reasoning chains, which allow one to transform the initial knowledge into the conclusion one wants to derive.

Real world, language, and mathematical structure

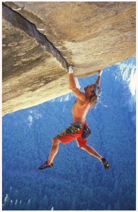
- Often we want to describe and reason about real world phenomena:
  - Providing a complete description of the real world is clearly impossible, and maybe also useless.
  - Typically one is interested in a portion of the world, e.g., a particular physical phenomenon, a social aspect, or modeling rationality of people, ...
- We use sentences of a language to describe objects of the real world, their properties, and facts that hold.
  - The language can be:
    - informal (natural lang., graphical lang., icons, ...) or
    - formal (logical lang., programming lang., mathematical lang., ...)
  - It is also possible to have mixed languages, i.e., languages with parts that are formal, and others that are informal (e.g., UML class diagrams)
- If we are also interested in a more rigorous description of the phenomena, we provide a mathematical model:
  - Is an abstraction of the portion of the real world we are interested in.
  - It represents real world entities in the form of mathematical objects, such as sets, relations, functions, ...
  - Is not commonly used in everyday communication, but is commonly adopted in science, e.g., to show that a certain argumentation is correct.

Language, real world, and mathematical model – Example

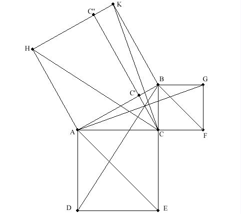
Language

In any right triangle, the area of the square whose side is the hypotenuse (the side opposite the right angle) is equal to the sum of the areas of the squares whose sides are the two legs (the two sides that meet at a right angle).

Real world



Mathematical model



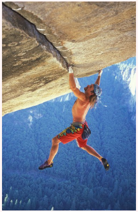
Facts about Euclidean geometry can be expressed in terms of natural language, and they can refer to one or more real world situations. (In the picture it refers to the composition of the forces in free climbing). However, the importance of the theorem lays in the fact that it describes a general property that holds in many different situations. All these different situations can be abstracted in the mathematical structure which is Euclidean geometry. So indeed the sentence can be interpreted directly in the mathematical structure. In this example the language is informal but it has an interpretation in a mathematical structure.

Language, real world, and math. model – Example 2

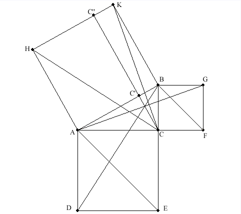
Language

In a triangle ABC, if $\widehat{BAC}$ is right, then $\overline{AB}^2 + \overline{AC}^2 = \overline{BC}^2$.

Real world



Mathematical model



This example is obtained from the previous one by taking a language that is "more formal". Indeed the language mixes informal statements (e.g., "if ... then ..." or "is right") with some formal notation.

E.g., $\widehat{BAC}$ is an unambiguous and compact way to denote an angle.

Similarly $\overline{AB}^2 + \overline{AC}^2 = \overline{BC}^2$ is a rigorous description of an equation that holds between the lengths of the triangle sides.

Language, real world, and math. model – Example 3

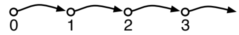
Language

$x - 2y + 3$

$x + y = 0$

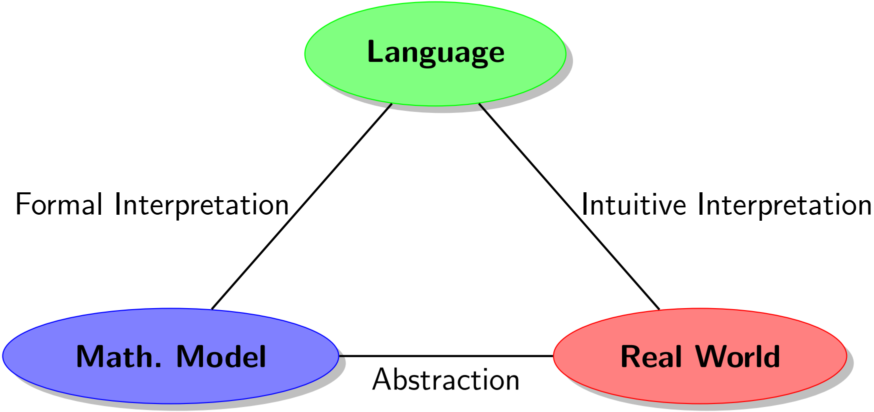
Real world

Mathematical model



In this example the language is purely formal, i.e., the language of arithmetic.
This abstract language is used to represent many situations in the real world (in the primary school we have many examples about apples, pears, and how they cost, which are used by teachers to explain to kids the intuitive meaning of the basic operations on numbers).
The mathematical model in this case is the structure of natural numbers.

Connections between language, world, and math. model



Connections between language, world, and math. model

Intuitive interpretation (or informal semantics)

When you propose a new language (or when you have to learn a new language) it is important to associate to every element of the language an interpretation in the real world. This is called the intuitive interpretation (or informal semantics). E.g., in learning a new programming language, you need to understand what is the effect in terms of execution of all the languages construct. For this reason the manual, typically, reports in natural language and with examples, the behavior of the language primitives. This is far to be a formal interpretation into a mathematical model. Therefore it is an informal interpretation.

Formal interpretation (or formal semantics)

Is a function that allows one to transform the elements of the language (i.e., symbols, words, complex sentences, ...) into one or more elements of the mathematical structure. It is indeed the formalization of the intuitive interpretation (or the intuitive semantics).

Abstraction

Is the link that connects the real world with it's mathematical and abstract representation into a mathematical structure. If a certain situation is supposed to be abstractly described by a given structure, then the abstraction connects the elements that participate to the situation, with the components of the mathematical structure, and the properties that hold in the situation with the mathematical properties that hold in the structure.

Outline of Part 1

Logic

Logic is a special case of the framework we have just seen, where the following important components are defined:

- The language is a **logical language**.
- The formal interpretation allows one to define a notion of **truth**.
- It is possible to define a notion of **logical consequence** between formulas. I.e., if a set Γ of formulas are true then also φ is true.

Formal language

- We are given a non-empty set Σ of symbols called **alphabet**.
- A formal language (over Σ) is a subset L of Σ^*, i.e., a set of finite strings of symbols in Σ.
- The elements of L are called **well formed phrases**.
- Formal languages can be specified by means of a **grammar**, i.e., a set of formation rules that allow one to build complex well formed phrases starting from simpler ones.

Logical language

A language of a logic, i.e., a logical language is a formal language that has the following characteristics:

- **Alphabet**: its symbols typically indicate the basic (atomic) components of the (part of the) world the logic is supposed to describe.
  Examples: individuals, functions, operators, truth-values, propositions, ...
- **Grammar**: contains rules for two types of phrases:
  - **Formulas**: denote propositions, i.e., objects that can assume some truth value (e.g., true, false, true in certain situations, true with probability of 3%, true/false in a period of time, ...).
  - **Terms**: denote objects of the world (e.g., cats, time points, quantities, ...).

Alphabet

The alphabet of a logical language is composed of two classes of symbols:

- **Logical constants**, whose formal interpretation is constant and fixed by the logic (e.g., $\wedge$, $\forall$, $=$, ...).
- **Non logical symbols**, whose formal interpretation is not fixed by the logic, and must be defined by the "user".

We can make an analogy with programming languages (say C, C++, python):

- Logical constants correspond to reserved words (whose meaning is fixed by the interpreter/compiler).
- Non logical symbols correspond to the identifiers that are introduced by the programmer for defining functions, variables, procedures, classes, attributes, methods, ...
  The meaning of these symbols is fixed by the programmer.

Alphabet: Logical constants – Example

The logical constants depend on the logic we are considering:

- Propositional logic: $\land$ (conjunction), $\lor$ (disjunction), $\neg$ (negation), $\supset$ (implication), $\equiv$ (equivalence), $\bot$ (falsity).
  These are usually called **propositional connectives**.

- Predicate logic: in addition to the propositional connectives, we have **quantifiers**:
  - universal quantifier $\forall$, standing for "every object is such that ..."
  - existential quantifier $\exists$, standing for "there is some object that ..."

- Modal logic: in addition to the propositional connectives, we have **modal operators**:
  - $\Box$, standing for "it is necessarily true that ..."
  - $\Diamond$, standing for "it is possibly true that ...".

Alphabet: Non-logical symbols – Example

- Propositional logic: non logical symbols are called **propositional variables**, and represent (i.e., have intuitive interpretation) propositions. The proposition associated to each propositional variable is not fixed by the logic.
- Predicate logic: there are four families of non logical symbols:
  - **Variable symbols**, which represent any object.
  - **Constant symbols**, which represent specific objects.
  - **Function symbols**, which represent transformations on objects.
  - **Predicate symbols**, which represent relations between objects.
- Modal logic: non logical symbols are the same as in propositional logic, i.e., propositional variables.

Example of grammar: Language of propositional logic

Grammar of propositional logic

Allows one to define the unique class of phrases, called **formulas** (or well formed formulas), which denote propositions.

$$
\begin{array}{lcll}
\mathit{Formula} & \longrightarrow & P & (P \text{ is a propositional variable}) \\
 & \mid & (\mathit{Formula} \land \mathit{Formula}) & \\
 & \mid & (\mathit{Formula} \lor \mathit{Formula}) & \\
 & \mid & (\mathit{Formula} \rightarrow \mathit{Formula}) & \\
 & \mid & (\neg \mathit{Formula}) &
\end{array}
$$

Example (Well formed formulas)

$$(P \land (Q \rightarrow R)) \qquad ((P \rightarrow (Q \rightarrow R)) \lor P)$$

These formulas are well formed, because there is a sequence of applications of grammar rules that generates them.
Exercise: list the rules in each case.

Example (Non well formed formulas)

$$P(Q \rightarrow R)$$
$$(P \rightarrow \lor P)$$

Example of grammar: Language of first order logic

Grammar of first order logic

$Term$	$\longrightarrow$	x	(x is a variable symbol)
	$\mid$	c	(c is a constant symbol)
	$\mid$	$f(Term, \ldots, Term)$	(f is a function symbol)
$Formula$	$\longrightarrow$	A	(A is a propositional symbol)
	$\mid$	$P(Term, \ldots, Term)$	(P is a predicate symbol)
	$\mid$	$(Formula \land Formula)$	
	$\mid$	$(Formula \lor Formula)$	
	$\mid$	$(Formula \rightarrow Formula)$	
	$\mid$	$\neg Formula$	
	$\mid$	$\forall x(Formula)$	(x is a variable symbol)
	$\mid$	$\exists x(Formula)$	(x is a variable symbol)

The rules define two types of phrases:

- **terms** denote objects (they are like noun phrases in natural language)
- **formulas** denote propositions (they are like sentences in natural language)

Exercise

Give examples of terms and formulas, and of phrases that are neither of the two.

Example of grammar: Language of a description logic

Grammar of the description logic $\mathcal{ALC}$

$$
\begin{array}{lcll}
\mathit{Concept} & \longrightarrow & A & (A \text{ is a concept symbol}) \\
& \mid & \mathit{Concept} \sqcup \mathit{Concept} & \\
& \mid & \mathit{Concept} \sqcap \mathit{Concept} & \\
& \mid & \neg \mathit{Concept} & \\
& \mid & \exists \mathit{Role}.\mathit{Concept} & \\
& \mid & \forall \mathit{Role}.\mathit{Concept} & \\
\mathit{Role} & \longrightarrow & R & (R \text{ is a role symbol}) \\
\mathit{Individual} & \longrightarrow & a & (a \text{ is an individual symbol}) \\
\mathit{Formula} & \longrightarrow & \mathit{Concept} \sqsubseteq \mathit{Concept} & \\
& \mid & \mathit{Concept}(\mathit{Individual}) & \\
& \mid & \mathit{Role}(\mathit{Individual}, \mathit{Individual}) &
\end{array}
$$

Example (Concepts and formulas of the DL $\mathcal{ALC}$)

- Concepts: $A \sqcap B$, $A \sqcup \exists R.C$, $\forall S.(C \sqcup \forall R.D) \sqcap \neg A$
- Formulas: $A \sqsubseteq B$, $A \sqsubseteq \exists R.B$, $A(a)$, $R(a, b)$, $\exists R.C(a)$

Outline of Part 1

Intuitive interpretation of a logical language

Non logical symbols do not have a fixed formal interpretation, but they usually have a fixed intuitive interpretation. Consider for instance:

Type	Symbol	Intuitive interpretation
propositional variable	$rain$	it is raining
constant symbol	$MobyDick$	the whale of a novel by Melville
function symbol	$color(x)$	the color of the object x
predicate symbol	$Friends(x, y)$	x and y are friends

The intuitive interpretation of the non logical symbols does not affect the logic itself.

- In other words, changing the intuitive interpretation does not affect the properties that will be proved in the logic.
- Similarly, replacing these logical symbols with less evocative ones, like r, M, $c(x)$, $F(x, y)$ will not affect the logic.

unibz

Interpretation of complex formulas

The intuitive interpretation of complex formulas is done by combining the intuitive interpretations of the components of the formulas.

Example

Consider the propositional formula:

$$(raining \vee snowing) \rightarrow \neg go\_to\_the\_beach$$

If the intuitive interpretations of the symbols are:

symbol	**intuitive meaning**
$raining$	it is raining
$snowing$	it is snowing
$go\_to\_the\_beach$	we go to the beach
$\vee$	either ... or ...
$\rightarrow$	if ... then ...
$\neg$	it is not the case that ...

then the above formula intuitively represent the proposition:

if (it is raining or it is snowing) then it is not the case that (we go to the beach)

Formal model

- **Class of models:** The models in which a logic is *formally interpreted* are the members of a class of algebraic structures, each of which is an abstract representation of the relevant aspects of the (portion of the) world we want to formalize with this logic.
- **Models** represent only the components and aspects of the world that are relevant to a certain analysis, and **abstract away from irrelevant facts**.
  Example: if we are interested in the everage temperature of each day, we can represent time with the natural numbers and use a function that associates to each natural number a floating point number (the average temperature of the day corresponding to the point).
- **Applicability of a model:** Since the real world is complex, in the construction of the formal model, we usually do **simplifying assumptions** that bound the usability of the logic to the cases in which these assumptions are verified.
  Example: if we take integers as formal model of time, then this model is not applicable to represent continuous change.
- Each **model represents** a single possible (or impossible) state of the world. The class of models of a logic will represent all the (im)possible states of the world.

Formal interpretation

- Given a structure S and a logical language L, the **formal interpretation** in S of L is a function that associates an element of S to any non logical symbol of the alphabet.
- The formal interpretation in the algebraic structure is the parallel counterpart (or better, the formalization) of the intuitive interpretation in the real world.
- The formal interpretation is specified only for the non logical symbols.
- Instead, the formal interpretation of the logical symbols is fixed by the logic.
- The formal interpretation of a complex expression e, obtained as a combination of the sub-expressions $e_1, \ldots, e_n$, is uniquely determined as a function of the formal interpretation of the sub-components $e_1, \ldots, e_n$.

Truth in a structure: Models

- As said, the goal of logic is the formalization of what is true/false in a particular world. The particular world is formalized by a structure, also called an **interpretation**.
- The main objective of the formal interpretation is that it allows to define when a **formula is true in an interpretation**.
- Every logic therefore defines the **satisfiability relation** (denoted by $\models$) between interpretations and formulas.
- If $\mathcal{I}$ is an interpretation and φ a formula, then

$$\mathcal{I} \models \varphi$$

  stands for the fact that $\mathcal{I}$ **satisfies** φ, or equivalently that φ **is true in** $\mathcal{I}$.
- An interpretation $\mathcal{M}$ such that $\mathcal{M} \models \varphi$ is called a **model** of φ.

(Un)satisfiability and validity

On the basis of truth in an interpretation ($\models$) the following notions are defined in any logic:

- φ is **satisfiable** if it has model, i.e., if there is a structure $\mathcal{M}$ such that $\mathcal{M} \models \varphi$.
- φ is **un-satisfiable** if it is not satisfiable, i.e., it has no models.
- φ is **valid**, denoted $\models \varphi$, if is true in all interpretations.

unibz

Logical consequence (or implication)

- The notion of **logical consequence** (or implication) is defined on the basis of the notion of truth in an interpretation.
- Intuitively, a formula φ is a logical consequence of a set of formulas (sometimes called assumptions) Γ (denoted $\Gamma \models \varphi$) if such a formula is true under this set of assumptions.
- Formally, $\Gamma \models \varphi$ holds when:

  For all interpretations $\mathcal{I}$, if $\mathcal{I} \models \Gamma$ then $\mathcal{I} \models \varphi$.

  In words: φ is true in all the possible situations in which all the formulas in Γ are true.
- Notice that the two relations, "truth in a model" and "logical consequence" are denoted by the same symbol $\models$ (this should remind you that they are tightly connected).

Outline of Part 1

unibz

Difference between $\models$ and implication ($\rightarrow$)

At a first glance $\models$ looks like implication (usually denoted by $\rightarrow$ or $\supset$).
Indeed in most of the cases they represent the same relation between formulas.

Similarity

- For instance, in propositional logic (but not only) the fact that φ is a logical consequence of the singleton set $\{\psi\}$, i.e., $\{\psi\} \models \varphi$, can be encoded in the formula $\psi \rightarrow \varphi$.
- Similarly, the fact that φ is a logical consequence of the set of formulas $\{\varphi_1, \ldots, \varphi_n\}$, i.e., $\{\varphi_1, \ldots, \varphi_n\} \models \varphi$ can be encoded by the formula $\varphi_1 \wedge \cdots \wedge \varphi_n \rightarrow \varphi$.

Difference

- When $\Gamma = \{\gamma_1, \gamma_2, \ldots\}$ is an infinite set of formulas, the fact that φ is a logical consequence of Γ cannot be represented with a formula $\gamma_1 \wedge \gamma_2 \wedge \ldots \rightarrow \varphi$ because this would be infinite, and in logic all the formulas are finite. (Actually there are logics, called infinitary logics, where formulas can have infinite size.)

Logical consequence, validity and (un)satisfiability

Exercise

Show that if $\Gamma = \emptyset$, then $\Gamma \models \varphi \iff \varphi$ is valid.

Solution

($\Longrightarrow$) Since Γ is empty, every interpretation $\mathcal{I}$ satisfies all the formulas in Γ. Therefore, if $\Gamma \models \varphi$, then every interpretation $\mathcal{I}$ must satisfy φ, hence φ is valid.
($\Longleftarrow$) If φ is valid, then every $\mathcal{I}$ is such that $\mathcal{I} \models \varphi$. Hence, whatever Γ is (in particular, when $\Gamma = \emptyset$), every model of Γ is also a model of φ, and so $\Gamma \models \varphi$.

Exercise

Show that if φ is unsatisfiable then $\{\varphi\} \models \psi$ for every formula ψ.

Solution

If φ is unsatisfiable then it has no model, which implies that each interpretation that satisfies φ (namely, none) satisfies also ψ, independently from ψ.

Properties of logical consequence

Property

Show that the following properties hold for the logical consequence relation defined above:

Reflexivity: $\Gamma \cup \{\varphi\} \models \varphi$

Monotonicity: $\Gamma \models \varphi$ implies that $\Gamma \cup \Sigma \models \varphi$

Cut: $\Gamma \models \varphi$ and $\Sigma \cup \{\varphi\} \models \psi$ implies that $\Gamma \cup \Sigma \models \psi$

Solution

Reflexivity: If $\mathcal{I}$ satisfies all the formulas in $\Gamma \cup \{\varphi\}$ then it satisfies also φ, and therefore $\Gamma \cup \{\varphi\} \models \varphi$.

Monotonicity: Let $\mathcal{I}$ be an interpretation that satisfies all the formulas in $\Gamma \cup \Sigma$. Then it satisfies all the formulas in Γ, and if $\Gamma \models \varphi$, then $\mathcal{I} \models \varphi$. Therefore, we can conclude that $\Gamma \cup \Sigma \models \varphi$.

Cut: Let $\mathcal{I}$ be an interpretation that satisfies all the formulas in $\Gamma \cup \Sigma$. Then it satisfies all the formulas in Γ, and if $\Gamma \models \varphi$, then $\mathcal{I} \models \varphi$. This implies that $\mathcal{I}$ satisfies all the formulas in $\Sigma \cup \{\varphi\}$. Then, since $\Sigma \cup \{\varphi\} \models \psi$, we have that $\mathcal{I} \models \psi$. Therefore we can conclude that $\Gamma \cup \Sigma \models \psi$.

Checking logical consequence

Problem

Does there exist an algorithm that checks if a formula φ is a logical consequence of a set of formulas Γ?

Solution 1: If Γ is finite and the set of models of the logic is finite, then it is possible to directly apply the definition by checking for every interpretation $\mathcal{I}$, that if $\mathcal{I} \models \Gamma$ then, $\mathcal{I} \models \varphi$.

Solution 2: If Γ is infinite or the set of models is infinite, then Solution 1 is not applicable as it would run forever.
An alternative solution could be to generate, starting from Γ, all its logical consequences by applying a set of rules.

Checking logical consequence

Propositional logic: The method based on **truth tables** can be used to check logical consequence by enumerating all the interpretations of Γ and φ and checking if every time all the formulas in Γ are true then φ is also true.
This is possible because, when Γ is finite then there are a finite number of interpretations.

First order logic: A first order language in general has an infinite number of interpretations. Therefore, to check logical consequence, it is not possible to apply a method that enumerates all the possible interpretations, as in truth tables.

Modal logic: presents the same problem as first order logic.
In general for a set of formulas Γ, there is an infinite number of interpretations, which implies that a method that enumerates all the interpretations is not effective.

Outline of Part 1

Checking logical consequence – Deductive methods

- An alternative method for determining if a formula is a logical consequence of a set of formulas is based on **inference rules**.
- An inference rule is a rewriting rule that takes a set of formulas and transforms it in another formulas.
- The following are examples of **inference rules**.

$$\frac{\varphi \quad \psi}{\varphi \land \psi} \qquad \frac{\varphi \quad \psi}{\varphi \rightarrow \psi} \qquad \frac{\forall x.\varphi(x)}{\varphi(c)} \qquad \frac{\exists x.\varphi(x)}{\varphi(d)}$$

- Differently from truth tables, which apply a brute force exhaustive analysis not interpretable by humans, the deductive method simulates human argumentation and provides also an understandable explanation (i.e., a **deduction**) of the reason why a formula is a logical consequence of a set of formulas.

Inference rules to check logical consequence – Example

Let $\Gamma = \{p \to q, \neg p \to r, q \lor r \to s\}$.

The following is a deduction (an explanation of) the fact that s is a logical consequence of Γ, i.e., that $\Gamma \models s$, which uses the following inference rules:

$$\frac{\varphi \to \psi \quad \neg\varphi \to \vartheta}{\psi \lor \vartheta} \ (*) \qquad\qquad \frac{\varphi \quad \varphi \to \psi}{\psi} \ (**)$$

Example of deduction

(1)	$p \to q$	Belongs to Γ.
(2)	$\neg p \to r$	Belongs to Γ.
(3)	$q \lor r$	By applying (*) to (1) and (2).
(4)	$q \lor r \to s$	Belongs to Γ.
(5)	s	By applying (**) to (3) and (4).

Hilbert-style inference methods

In a Hilbert-style deduction system, a formal deduction is a **finite sequence of formulas**

$$\begin{array}{c}\varphi_1\\ \varphi_2\\ \varphi_3\\ \vdots\\ \varphi_n\end{array}$$

where each φ_i

- is either an **axiom**, or
- it is derived from previous formulas $\varphi_{j_1}, \ldots, \varphi_{j_k}$ with $j_1, \ldots, j_k < i$, by applying the **inference rule**

$$\frac{\varphi_{j_1},\ \ldots,\ \varphi_{j_k}}{\varphi_i}$$

unibz

Hilbert axioms for classical propositional logic

Axioms

A1 $\varphi \to (\psi \to \varphi)$

A2 $(\varphi \to (\psi \to \theta)) \to ((\varphi \to \psi) \to (\varphi \to \theta))$

A3 $(\neg\psi \to \neg\varphi) \to ((\neg\psi \to \varphi) \to \psi)$

Inference rule(s)

MP $$\frac{\varphi \quad \varphi \to \psi}{\psi}$$

Example (Proof of $A \to A$)

1. A1 $A \to ((A \to A) \to A)$
2. A2 $(A \to ((A \to A) \to A)) \to ((A \to (A \to A)) \to (A \to A))$
3. MP(1,2) $(A \to (A \to A)) \to (A \to A)$
4. A1 $(A \to (A \to A))$
5. MP(4,3) $A \to A$

Refutation

Reasoning by refutation is based on the principle of "**Reductio ad absurdum**".

Reductio ad absurdum

In order to show that a proposition φ is true, we assume that it is false (i.e., that $\neg\varphi$ holds) and try to infer a contradictory statement, such as $A \wedge \neg A$ (usually denoted by $\bot$, i.e., the false statement).

Reasoning by refutation is one of the most important principles for building **automated decision procedures**. This is mainly due to the fact that, proving a formula φ corresponds to the reduction of $\neg\varphi$ to $\bot$.

Propositional resolution

Propositional resolution is the most simple example of reasoning via refutation. The procedure can be described as follows:

Propositional resolution

INPUT: a propositional formula φ
OUTPUT: $\models \varphi$ or $\not\models \varphi$

1. Convert $\neg\varphi$ to conjunctive normal form, i.e., to a set C of formulas (called clauses) of the form
$$p_1 \vee \cdots \vee p_k \vee \neg p_{k+1} \vee \cdots \vee \neg p_n$$
that is logically equivalent to φ.
2. Apply exhaustively the following inference rule
$$\frac{c \vee p \quad \neg p \vee c'}{c \vee c'} \textit{ Resolution}$$
and add $c \vee c'$ to C
3. if C contains two clauses p and $\neg p$ then return $\models \varphi$ otherwise return $\not\models \varphi$

Inference based on satisfiability checking

In order to show that $\models \varphi$ (i.e., that φ is valid) we search for a model of $\neg\varphi$, i.e., we show that $\neg\varphi$ **is satisfiable**.

If we are not able to find such a model, then we can conclude that there is no model of $\neg\varphi$, i.e., that all the models satisfy φ, which is: that φ is valid.

unibz

Inference based on satisfiability checking

There are two basic methods of searching for a model for φ:

SAT based decision procedures

- This method incrementally builds a model.
- At every stage it defines a "partial model" μ_i and does an early/lazy check if φ can be true in some extension of μ_i.
- At each point the algorithm has to decide how to extend μ_i to μ_{i+1} until constructs a full model for φ.

Tableaux based decision procedures

- This method builds the model of φ via a "top down" approach.
- I.e., φ is decomposed in its sub-formulas $\varphi_1, \ldots, \varphi_n$ and the algorithm recursively builds n models $M_1, \ldots, M_n$ for them.
- The model M of φ is obtained by a suitable combination of $M_1, \ldots, M_n$.

unibz

SAT based decision procedure – Example

We illustrate a SAT based decision procedure on a propositional logic example.

To find a model for $(p \vee q) \wedge \neg p$, we proceed as follows:

Partial model	lazy evaluation	result of lazy evaluation	
$\mu_0 = \{p = \textit{true}\}$	$(\textit{true} \vee q) \wedge \neg\textit{true}$	*false*	(backtrack)
$\mu_1 = \{p = \textit{false}\}$	$(\textit{false} \vee q) \wedge \neg\textit{false}$	q	(continue)
$\mu_2 = \{p = \textit{false}$, $q = \textit{true}\}$	$(\textit{false} \vee \textit{true}) \wedge \neg\textit{false}$	*true*	(success!)

Soundness and Completeness

Let R be an inference method, and let $\vdash_R$ denote the corresponding inference relation.

Definition (Soundness of an inference method)

An inference method R is **sound** if

$$\begin{aligned} \vdash_R \varphi \quad &\Longrightarrow \quad \models \varphi \\ \Gamma \vdash_R \varphi \quad &\Longrightarrow \quad \Gamma \models \varphi \qquad \text{(strongly sound)} \end{aligned}$$

Definition (Completeness of an inference method)

An inference method R is **complete** if

$$\begin{aligned} \models \varphi \quad &\Longrightarrow \quad \vdash_R \varphi \\ \Gamma \models \varphi \quad &\Longrightarrow \quad \Gamma \vdash_R \varphi \qquad \text{(strongly complete)} \end{aligned}$$

Outline of Part 1

Outline of Part 1

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 of Part 1

unibz

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)

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).

Concepts

Def.: 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

Def.: 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 → { blu, brown, green, grey }
  - structured (date, set, list, ...)
    e.g., date → day/month/year
- The definition may also specify a default value.

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

Relationships

Def.: 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

Def.: Axiom

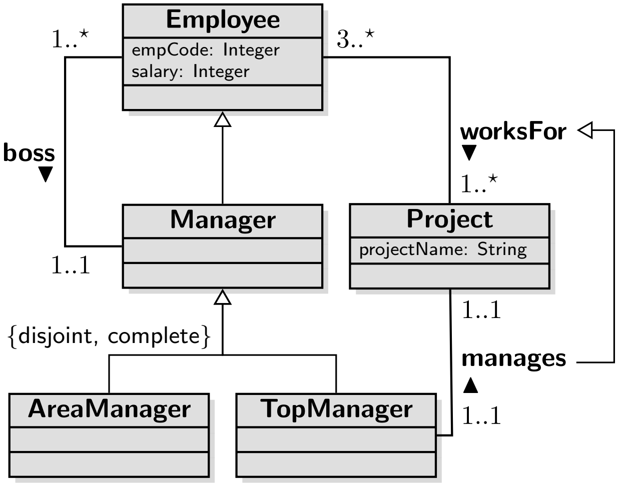
Is a logical formula that expresses at the intensional level a condition that must be satisified by the elements at the extensional level.

Different kinds of axioms/conditions:

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

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

Example: ontology rendered as UML Class Diagram



Outline of Part 1

unibz

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: 2.4.1 (Aug. 2011): 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/

UML Class Diagrams

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

A UML Class Diagram (or shortly, UML-CD) 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.

unibz

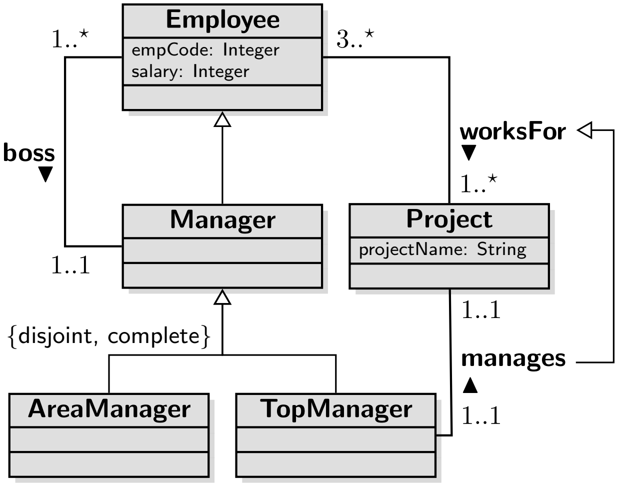
UML Class Diagrams (cont'd)

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.

unibz

Example of a UML Class Diagram



Use of UML Class Diagrams

UML CDs 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**".

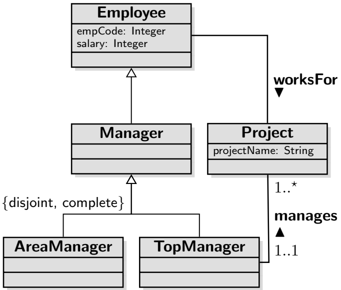
2. During software development, to maintain an abstract view of the software to be developed.
   $\rightsquigarrow$ the so-called "implementation perspective".

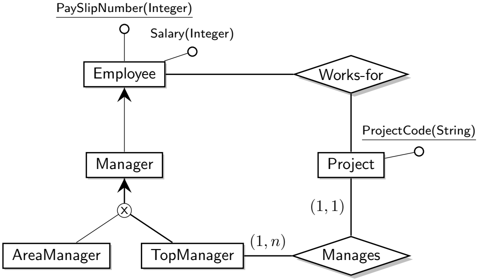
In this course we focus on 1!

UML Class Diagrams and ER Schemas

UML CDs (when used for the conceptual perspective) closely resemble Entity-Relationship (ER) Diagrams.

Example of UML vs. ER:





Outline of Part 1

unibz

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, XOL
- Logic based
  - **Description Logics** (e.g., $\mathcal{SHOIQ}$, $\mathcal{DLR}$, **DL-Lite** , OWL, . . .)
  - Rules (e.g., RuleML, LP/Prolog, F-Logic)
  - First Order Logic (e.g., KIF)
  - Non-classical logics (e.g., non-monotonic, probabilistic)

Outline of Part 1

unibz

Outline of Part 1

Modeling the domain of interest

We aim at obtaining a description of the data 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**, namely the **ontology**.
3. Use the resulting logical theory for **reasoning** and **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.

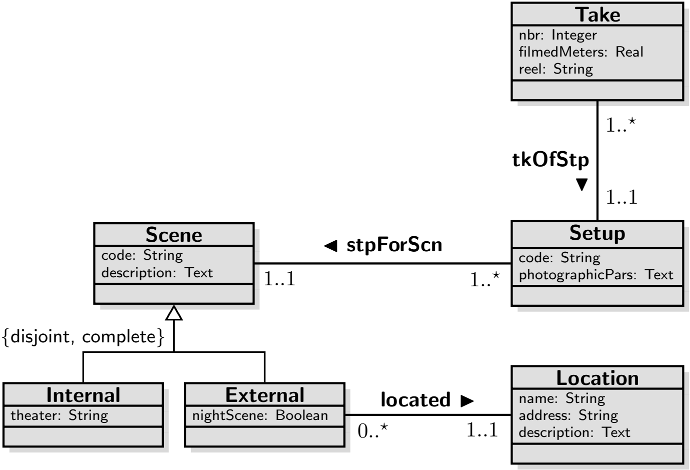
Every scene is filmed from different positions (at least one), each of these is called a setup. Every setup is characterized by a code (a string) and a text in natural language where the photographic parameters are noted (e.g., aperture, exposure, focal length, filters, etc.). Note that a setup is related 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: $Scene(x)$, $Setup(x)$, $Take(x)$, $Internal(x)$, $External(x)$, $Location(x)$, $stpForScn(x,y)$, $tkOfStp(x,y)$, $located(x,y)$, $\ldots$.

Axioms:

$\forall x,y.\ code_{Scene}(x,y) \rightarrow Scene(x) \land String(y)$
$\forall x,y.\ description(x,y) \rightarrow Scene(x) \land Text(y)$

$\forall x,y.\ code_{Setup}(x,y) \rightarrow Setup(x) \land String(y)$
$\forall x,y.\ photographicPars(x,y) \rightarrow Setup(x) \land Text(y)$

$\forall x,y.\ nbr(x,y) \rightarrow Take(x) \land Integer(y)$
$\forall x,y.\ filmedMeters(x,y) \rightarrow Take(x) \land Real(y)$
$\forall x,y.\ reel(x,y) \rightarrow Take(x) \land String(y)$

$\forall x,y.\ theater(x,y) \rightarrow Internal(x) \land String(y)$

$\forall x,y.\ nightScene(x,y) \rightarrow External(x) \land Boolean(y)$

$\forall x,y.\ name(x,y) \rightarrow Location(x) \land String(y)$
$\forall x,y.\ address(x,y) \rightarrow Location(x) \land String(y)$
$\forall x,y.\ description(x,y) \rightarrow Location(x) \land Text(y)$

$\forall x.\ Scene(x) \rightarrow (1 \leq \sharp\{y \mid code_{Scene}(x,y)\} \leq 1)$

$\forall x.\ Internal(x) \rightarrow Scene(x)$
$\forall x.\ External(x) \rightarrow Scene(x)$
$\forall x.\ Internal(x) \rightarrow \neg External(x)$
$\forall x.\ Scene(x) \rightarrow Internal(x) \lor External(x)$

$\forall x,y.\ stpForScn(x,y) \rightarrow Setup(x) \land Scene(y)$
$\forall x,y.\ tkOfStp(x,y) \rightarrow Take(x) \land Setup(y)$
$\forall x,y.\ located(x,y) \rightarrow External(x) \land Location(y)$

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

unibz

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 good point, in fact.*

unibz

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 also use Entity-Relationship Diagrams instead of UML CDs.
- as logic: **First-Order Logic** to formally capture semantics and reasoning.

unibz

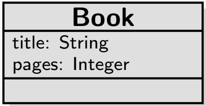
Classes in UML CDs

A **class** in a UML CD 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 CDs: 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:

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

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

Classes in UML CDs: 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?

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

Example

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

unibz

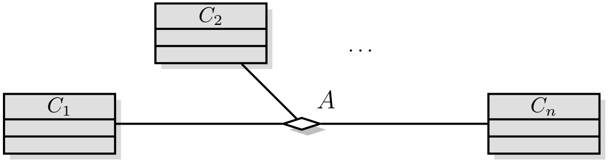
Associations

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



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

$$\forall x_1, x_2.\, \mathit{writtenBy}(x_1, x_2) \rightarrow \mathit{Book}(x_1) \land \mathit{Author}(x_2)$$

Associations: multiplicity

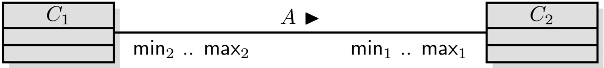
On binary associations, we can place **multiplicity constraints**, i.e., a minimal and maximal number of tuples in which every object participates as first (second) component.

Example

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:

$$\forall x_1. C_1(x_1) \rightarrow (min_1 \leq \sharp\{x_2 \mid A(x_1, x_2)\} \leq max_1)$$
$$\forall x_2. C_2(x_2) \rightarrow (min_2 \leq \sharp\{x_1 \mid A(x_1, x_2)\} \leq max_2)$$

Example

$$\forall x. Book(x) \rightarrow (1 \leq \sharp\{y \mid written\_by(x, y)\})$$

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

Expressing multiplicities in FOL

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

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

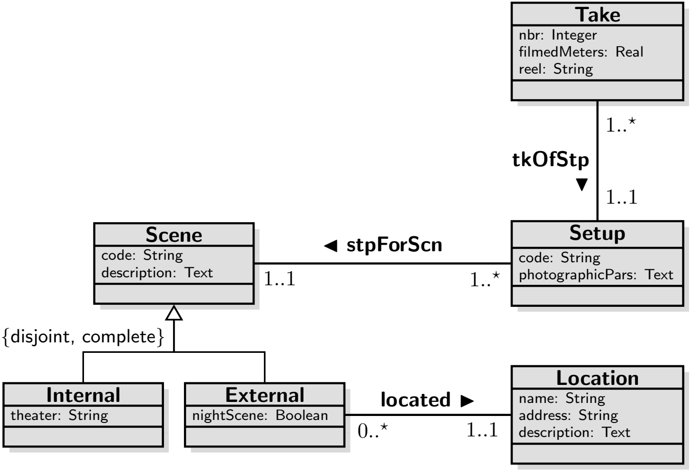
$$m \leq \sharp\{x \mid \varphi(x)\} \quad = \quad \exists x_1, \ldots, x_m . \varphi(x_1) \land \cdots \land \varphi(x_m) \land \bigwedge_{\substack{1 \leq i < m \\ i < j \leq m}} x_i \neq x_j$$

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

$$\sharp\{x \mid \varphi(x)\} \leq n \quad = \quad \forall x_1, \ldots, x_n, x_{n+1} . (\varphi(x_1) \land \cdots \land \varphi(x_{n+1})) \rightarrow \bigvee_{\substack{1 \leq i \leq n \\ i < j \leq n+1}} x_i = x_j$$

Note: we need **FOL with equality**.

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:

$\forall x,y.\ code_{Scene}(x,y) \rightarrow Scene(x) \wedge String(y)$
$\forall x,y.\ description(x,y) \rightarrow Scene(x) \wedge Text(y)$

$\forall x,y.\ code_{Setup}(x,y) \rightarrow Setup(x) \wedge String(y)$
$\forall x,y.\ photographicPars(x,y) \rightarrow Setup(x) \wedge Text(y)$

$\forall x,y.\ nbr(x,y) \rightarrow Take(x) \wedge Integer(y)$
$\forall x,y.\ filmedMeters(x,y) \rightarrow Take(x) \wedge Real(y)$
$\forall x,y.\ reel(x,y) \rightarrow Take(x) \wedge String(y)$

$\forall x,y.\ theater(x,y) \rightarrow Internal(x) \wedge String(y)$

$\forall x,y.\ nightScene(x,y) \rightarrow External(x) \wedge Boolean(y)$

$\forall x,y.\ name(x,y) \rightarrow Location(x) \wedge String(y)$
$\forall x,y.\ address(x,y) \rightarrow Location(x) \wedge String(y)$
$\forall x,y.\ description(x,y) \rightarrow Location(x) \wedge Text(y)$

$\forall x.\ Scene(x) \rightarrow (1 \leq \sharp\{y \mid code_{Scene}(x,y)\} \leq 1)$

$\forall x.\ Internal(x) \rightarrow Scene(x)$
$\forall x.\ External(x) \rightarrow Scene(x)$
$\forall x.\ Internal(x) \rightarrow \neg External(x)$
$\forall x.\ Scene(x) \rightarrow Internal(x) \vee External(x)$

$\forall x,y.\ stpForScn(x,y) \rightarrow Setup(x) \wedge Scene(y)$
$\forall x,y.\ tkOfStp(x,y) \rightarrow Take(x) \wedge Setup(y)$
$\forall x,y.\ located(x,y) \rightarrow External(x) \wedge Location(y)$

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

unibz

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)

In FOL:

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

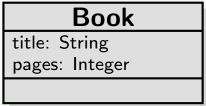
Attributes

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



– 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.

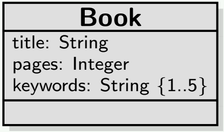
Example

$book_3$ 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 $1..1$.

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:

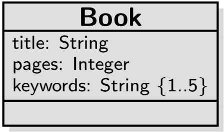
- An assertion for the attribute **type**:

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

- An assertion for the **multiplicity**:

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

Attributes – Exercise on formalization



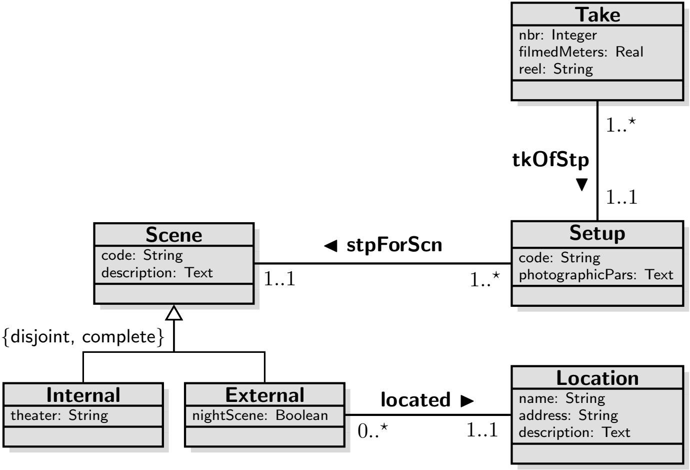
Provide the FOL formalization of the attributes of the above class.

$$\forall x, y.\, \mathit{title}_B(x,y) \rightarrow \mathit{Book}(x) \land \mathit{String}(y)$$
$$\forall x.\, \mathit{Book}(x) \rightarrow (1 \leq \sharp\{y \mid \mathit{title}_B(x,y)\} \leq 1)$$

$$\forall x, y.\, \mathit{pages}_B(x,y) \rightarrow \mathit{Book}(x) \land \mathit{Integer}(y)$$
$$\forall x.\, \mathit{Book}(x) \rightarrow (1 \leq \sharp\{y \mid \mathit{pages}_B(x,y)\} \leq 1)$$

$$\forall x, y.\, \mathit{keywords}_B(x,y) \rightarrow \mathit{Book}(x) \land \mathit{String}(y)$$
$$\forall x.\, \mathit{Book}(x) \rightarrow (1 \leq \sharp\{y \mid \mathit{keywords}_B(x,y)\} \leq 5)$$

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:

$\forall x,y.\, code_{Scene}(x,y) \rightarrow Scene(x) \wedge String(y)$
$\forall x,y.\, description(x,y) \rightarrow Scene(x) \wedge Text(y)$

$\forall x,y.\, code_{Setup}(x,y) \rightarrow Setup(x) \wedge String(y)$
$\forall x,y.\, photographicPars(x,y) \rightarrow Setup(x) \wedge Text(y)$

$\forall x,y.\, nbr(x,y) \rightarrow Take(x) \wedge Integer(y)$
$\forall x,y.\, filmedMeters(x,y) \rightarrow Take(x) \wedge Real(y)$
$\forall x,y.\, reel(x,y) \rightarrow Take(x) \wedge String(y)$

$\forall x,y.\, theater(x,y) \rightarrow Internal(x) \wedge String(y)$

$\forall x,y.\, nightScene(x,y) \rightarrow External(x) \wedge Boolean(y)$

$\forall x,y.\, name(x,y) \rightarrow Location(x) \wedge String(y)$
$\forall x,y.\, address(x,y) \rightarrow Location(x) \wedge String(y)$
$\forall x,y.\, description(x,y) \rightarrow Location(x) \wedge Text(y)$

$\forall x.\, Scene(x) \rightarrow (1 \leq \sharp\{y \mid code_{Scene}(x,y)\} \leq 1)$

$\forall x.\, Internal(x) \rightarrow Scene(x)$
$\forall x.\, External(x) \rightarrow Scene(x)$
$\forall x.\, Internal(x) \rightarrow \neg External(x)$
$\forall x.\, Scene(x) \rightarrow Internal(x) \vee External(x)$

$\forall x,y.\, stpForScn(x,y) \rightarrow Setup(x) \wedge Scene(y)$
$\forall x,y.\, tkOfStp(x,y) \rightarrow Take(x) \wedge Setup(y)$
$\forall x,y.\, located(x,y) \rightarrow External(x) \wedge Location(y)$

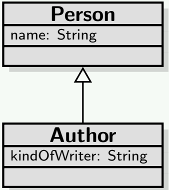
$\forall x.\, Setup(x) \rightarrow (1 \leq \sharp\{y \mid stpForScn(x,y)\} \leq 1)$
$\forall y.\, Scene(y) \rightarrow (1 \leq \sharp\{x \mid stpForScn(x,y))\}$
$\forall x.\, Take(x) \rightarrow (1 \leq \sharp\{y \mid tkOfStp(x,y)\} \leq 1)$
$\forall x.\, Setup(y) \rightarrow (1 \leq \sharp\{x \mid tkOfStp(x,y)\})$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

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

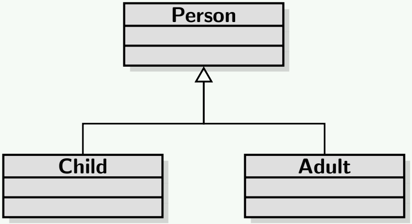


The attibute 'name' is inherited by 'Author'.

Generalizations

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

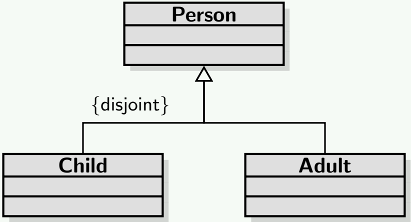


unibz

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.

Example

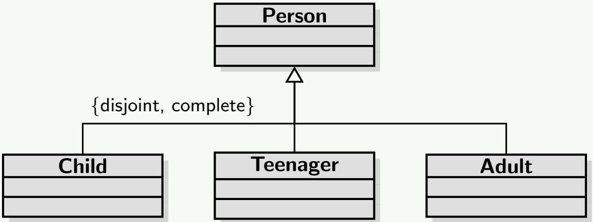


Generalizations with constraints (cont'd)

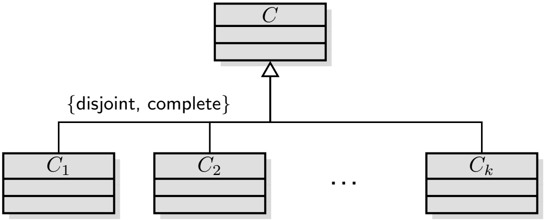
Most notable and used constraints:

- **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

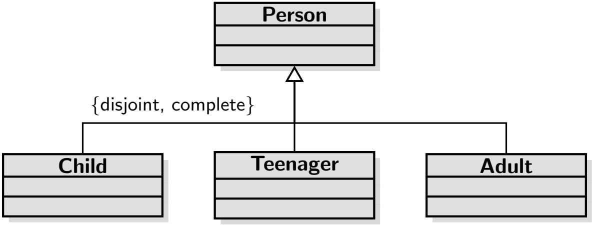


ISA:	$\forall x. C_i(x) \rightarrow C(x)$,	for $1 \leq i \leq k$
Disjointness:	$\forall x. C_i(x) \rightarrow \neg C_j(x)$,	for $1 \leq i < j \leq k$
Completeness:	$\forall x. C(x) \rightarrow \bigvee_{i=1}^{k} C_i(x)$	

Note: ISA and completeness can be formalized more compactly with:

$$\forall x. C(x) \leftrightarrow \bigvee_{i=1}^{k} C_i(x)$$

Generalizations – Exercise on formalization



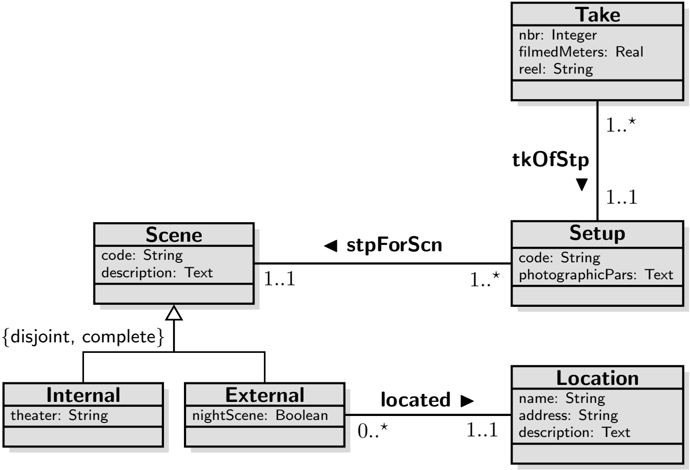
Provide the FOL formalization of the above generalization hierarchy.

$$\forall x.\, Child(x) \rightarrow Person(x)$$
$$\forall x.\, Teenager(x) \rightarrow Person(x)$$
$$\forall x.\, Adult(x) \rightarrow Person(x)$$

$$\forall x.\, Child(x) \rightarrow \neg Teenager(x)$$
$$\forall x.\, Child(x) \rightarrow \neg Adult(x)$$
$$\forall x.\, Teenager(x) \rightarrow \neg Adult(x)$$

$$\forall x.\, Person(x) \rightarrow (Child(x) \vee Teenager(x) \vee Adult(x))$$

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:

$\forall x,y.\, code_{Scene}(x,y) \rightarrow Scene(x) \wedge String(y)$
$\forall x,y.\, description(x,y) \rightarrow Scene(x) \wedge Text(y)$

$\forall x,y.\, code_{Setup}(x,y) \rightarrow Setup(x) \wedge String(y)$
$\forall x,y.\, photographicPars(x,y) \rightarrow Setup(x) \wedge Text(y)$

$\forall x,y.\, nbr(x,y) \rightarrow Take(x) \wedge Integer(y)$
$\forall x,y.\, filmedMeters(x,y) \rightarrow Take(x) \wedge Real(y)$
$\forall x,y.\, reel(x,y) \rightarrow Take(x) \wedge String(y)$

$\forall x,y.\, theater(x,y) \rightarrow Internal(x) \wedge String(y)$

$\forall x,y.\, nightScene(x,y) \rightarrow External(x) \wedge Boolean(y)$

$\forall x,y.\, name(x,y) \rightarrow Location(x) \wedge String(y)$
$\forall x,y.\, address(x,y) \rightarrow Location(x) \wedge String(y)$
$\forall x,y.\, description(x,y) \rightarrow Location(x) \wedge Text(y)$

$\forall x.\, Scene(x) \rightarrow (1 \leq \sharp\{y \mid code_{Scene}(x,y)\} \leq 1)$

$\forall x.\, Internal(x) \rightarrow Scene(x)$
$\forall x.\, External(x) \rightarrow Scene(x)$
$\forall x.\, Internal(x) \rightarrow \neg External(x)$
$\forall x.\, Scene(x) \rightarrow Internal(x) \vee External(x)$

$\forall x,y.\, stpForScn(x,y) \rightarrow Setup(x) \wedge Scene(y)$
$\forall x,y.\, tkOfStp(x,y) \rightarrow Take(x) \wedge Setup(y)$
$\forall x,y.\, located(x,y) \rightarrow External(x) \wedge Location(y)$

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

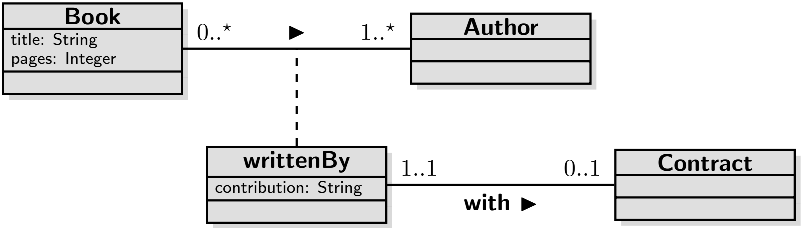
unibz

Association classes

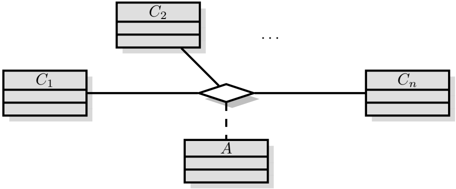
Sometimes we may want to assert properties of associations. In UML to do so we resort to **association classes**:

- 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 classes: reification

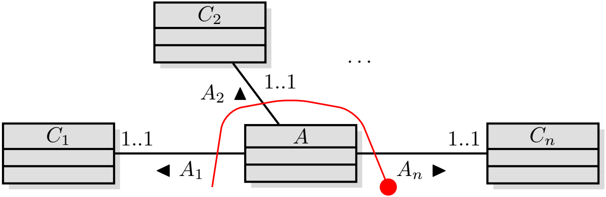


Reification

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

- 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 of reification



FOL assertions are needed for ensuring that one can establish a bijection between instances of the association class and instances of the association:

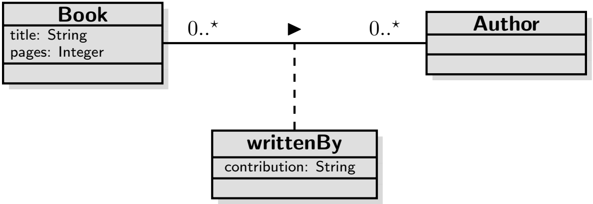
$$\forall x, y.\, A_i(x, y) \rightarrow A(x) \wedge C_i(y), \qquad \text{for } i \in \{1, \ldots, n\}$$

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

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

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

Association classes – Exercise on formalization



Provide the FOL formalization of the above association class.

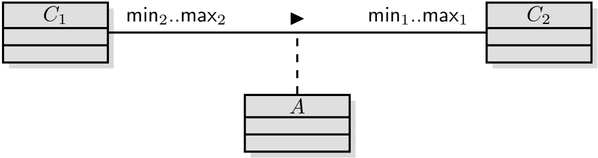
$$\forall x, y.\, wb_1(x,y) \rightarrow writtenBy(x) \wedge Book(y)$$
$$\forall x, y.\, wb_2(x,y) \rightarrow writtenBy(x) \wedge Author(y)$$

$$\forall x.\, writtenBy(x) \rightarrow \exists y.\, wb_1(x,y)$$
$$\forall x.\, writtenBy(x) \rightarrow \exists y.\, wb_2(x,y)$$
$$\forall x, y, y'.\, wb_1(x,y) \wedge wb_1(x,y') \rightarrow y = y'$$
$$\forall x, y, y'.\, wb_2(x,y) \wedge wb_2(x,y') \rightarrow y = y'$$

$$\forall x, x', y_1, y_2.\, wb_1(x,y_1) \wedge wb_1(x',y_1) \wedge wb_2(x,y_2) \wedge wb_2(x',y_2) \rightarrow x = x'$$

unibz

Association classes: formalization of multiplicities

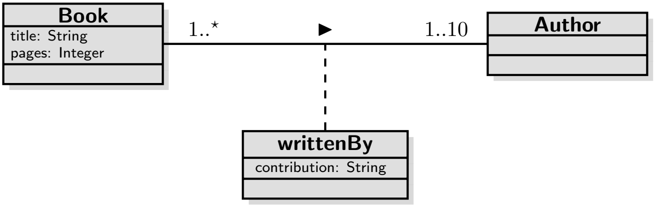


The multiplicities are expressed over (the second component of) the binary predicates A_1 and A_2 representing the components of the association:

$$\forall y.\, C_1(y) \rightarrow min_1 \leq \sharp\{x \mid A_1(x,y)\} \leq max_1$$

$$\forall y.\, C_2(y) \rightarrow min_2 \leq \sharp\{x \mid A_2(x,y)\} \leq max_2$$

Association classes with multiplicities – Exercise



Considering the previous formalization of the association class *writtenBy*, provide the FOL formalization of the multiplicities.

$$\forall y. Book(y) \rightarrow 1 \leq \sharp\{x \mid wb_1(x,y)\} \leq 10$$

$$\forall y. Author(y) \rightarrow 1 \leq \sharp\{x \mid wb_2(x,y)\}$$

Outline of Part 1

unibz

Reasoning on UML CDs via FOL reasoning

There are several properties that are of interest for a UML CD, and that can be phrased as forms of inference on the diagram:

- Consistency of the whole diagram.
- Consistency of classes and associations, to avoid maintaining in the diagram useless information.
- Subsumption of classes (or associations), to detect when properties of a more general class are inherited by a more specific class.
- Equivalence of classes (or associations), to detect and eliminate redundancy in the diagram.
- Implication of properties, to make implicit information explicit.

In the following, we formally define the above reasoning tasks on UML CDs, and show how they can be recast in terms of FOL reasoning.

To this aim, given a UML CD $\mathcal{D}$, we denote with $\Gamma_{\mathcal{D}}$ the set of FOL sentences that encode in FOL all constructs of $\mathcal{D}$.

Forms of reasoning: class and association consistency

A class C is **consistent** in a UML CD $\mathcal{D}$, if $\mathcal{D}$ admits an instantiation in which C has a non-empty set of instances.

Let $\Gamma_{\mathcal{D}}$ be the set of FOL sentences encoding $\mathcal{D}$, and $C(x)$ the predicate corresponding to the class C.

Then C **is consistent in** $\mathcal{D}$ iff

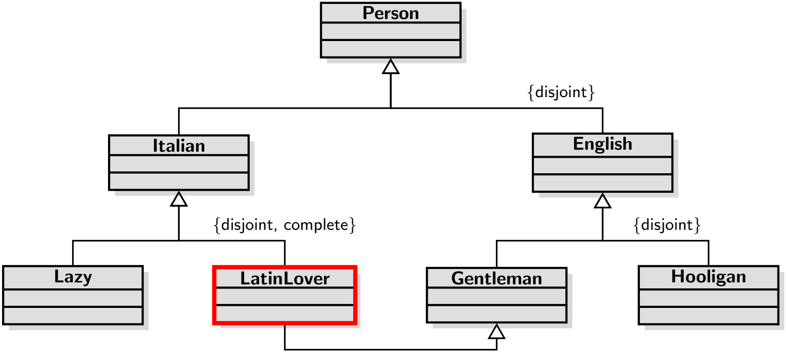
$\Gamma_{\mathcal{D}} \cup \{\exists x. C(x)\}$ is satisfiable (or, $\Gamma_{\mathcal{D}} \not\models \forall x. C(x) \rightarrow \mathit{false}$)

i.e., there exists a model of $\Gamma_{\mathcal{D}}$ in which the extension of $C(x)$ is not empty.

Corresponding FOL reasoning task: satisfiability.

Similarly, we can define consistency of an association in a UML CD.

Class consistency – Example (by E. Franconi)



$$\Gamma_{\mathcal{D}} \models \forall x. \mathit{LatinLover}(x) \rightarrow \mathit{false}$$

Forms of reasoning: diagram consistency

A UML CD $\mathcal{D}$ is **consistent**, if it admits an instantiation, i.e., if its classes can be populated without violating any of the conditions imposed by $\mathcal{D}$.

Let $\Gamma_{\mathcal{D}}$ be the set of FOL sentences encoding $\mathcal{D}$.

Then, **the UML CD $\mathcal{D}$ is consistent** iff

$$\Gamma_{\mathcal{D}} \text{ is satisfiable}$$

i.e., $\Gamma_{\mathcal{D}}$ admits at least one model.
(Remember that FOL models have a non-empty domain.)

Corresponding FOL reasoning task: satisfiability.

Note: With the UML constructs we have considered, every UML CD $\mathcal{D}$ is consistent. Indeed, every interpretation in which the extension of all predicates is empty is a model of $\Gamma_{\mathcal{D}}$.

Forms of reasoning: full diagram consistency

A UML CD $\mathcal{D}$ is **fully consistent**, if it admits an instantiation in which all classes and associations have a non-empty extension.

- Let $\mathcal{C}$ be the set of classes in $\mathcal{D}$.
- Let k be the maximum arity of associations in $\mathcal{D}$, and let $\mathcal{A}_n$, for $n \in \{2, \ldots, k\}$, be the set of associations of arity n in $\mathcal{D}$.
- Let $\Gamma_\mathcal{D}$ be the set of FOL sentences encoding $\mathcal{D}$.

Then, **the UML CD $\mathcal{D}$ is fully consistent** iff

$$\Gamma_\mathcal{D} \cup \bigcup_{C \in \mathcal{C}} \{\exists x. C(x)\} \cup \bigcup_{n=1}^{k} \bigcup_{A \in \mathcal{A}_n} \{\exists x_1, \ldots, x_n. A(x_1, \ldots, x_n)\}$$

is satisfiable.

Corresponding FOL reasoning task: satisfiability.

Exercise

Show that a UML CD $\mathcal{D}$ is fully consistent iff every class and every association of $\mathcal{D}$ is consistent in $\mathcal{D}$.

Forms of reasoning: class subsumption

A class C_1 **is subsumed by** a class C_2 (or C_2 subsumes C_1), in a UML CD $\mathcal{D}$ if $\mathcal{D}$ that C_2 is a generalization of C_1.

Let $\Gamma_{\mathcal{D}}$ be the set of FOL sentences encoding $\mathcal{D}$, and $C_1(x)$, $C_2(x)$ the predicates corresponding to the classes C_1, and C_2 of $\mathcal{D}$.

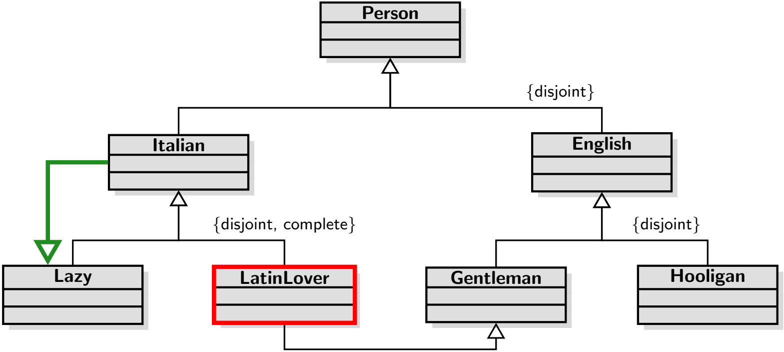
Then C_1 **is subsumed by** C_2 **in** $\mathcal{D}$ iff

$$\Gamma_{\mathcal{D}} \models \forall x. C_1(x) \rightarrow C_2(x)$$

Corresponding FOL reasoning task: logical implication.

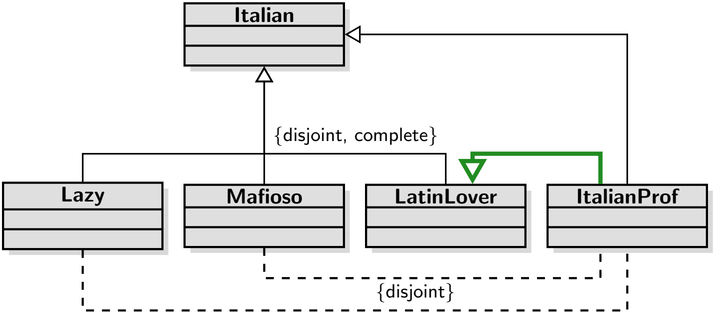
unibz

Class subsumption – Example



$$\Gamma_{\mathcal{D}} \models \forall x.\, \mathit{LatinLover}(x) \rightarrow \mathit{false}$$
$$\Gamma_{\mathcal{D}} \models \forall x.\, \mathit{Italian}(x) \rightarrow \mathit{Lazy}(x)$$

Class subsumption: another example (by E. Franconi)



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

Note: this is an example of reasoning by cases.

Forms of reasoning: class equivalence

Two classes C_1 and C_2 are **equivalent** in a UML CD $\mathcal{D}$, if C_1 and C_2 denote the same set of instances in all instantiations of $\mathcal{D}$.

Let $\Gamma_{\mathcal{D}}$ be the set of FOL sentences encoding $\mathcal{D}$, and $C_1(x)$, $C_2(x)$ the predicates corresponding to the classes C_1, and C_2 of $\mathcal{D}$.

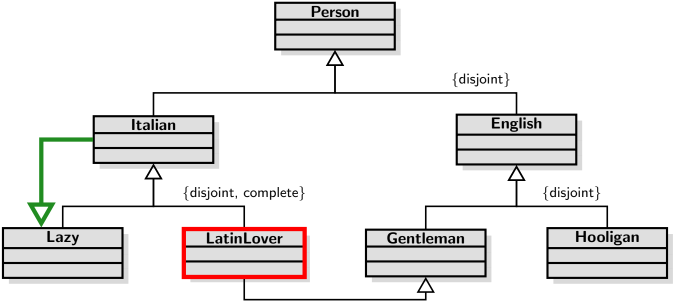
Then **C_1 and C_2 are equivalent in $\mathcal{D}$** iff

$$\Gamma_{\mathcal{D}} \models \forall x. C_1(x) \leftrightarrow C_2(x)$$

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



$$\Gamma_{\mathcal{D}} \models \forall x.\, LatinLover(x) \rightarrow false$$
$$\Gamma_{\mathcal{D}} \models \forall x.\, Italian(x) \rightarrow Lazy(x)$$
$$\Gamma_{\mathcal{D}} \models \forall x.\, Lazy(x) \equiv Italian(x)$$

Forms of reasoning: 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 UML CD $\mathcal{D}$ if $\mathcal{P}$ holds whenever all conditions imposed by $\mathcal{D}$ are satisfied.

Let $\Gamma_{\mathcal{D}}$ be the set of FOL sentences encoding $\mathcal{D}$, and
let $\mathcal{P}$ be (the formalization in FOL of) the property of interest.

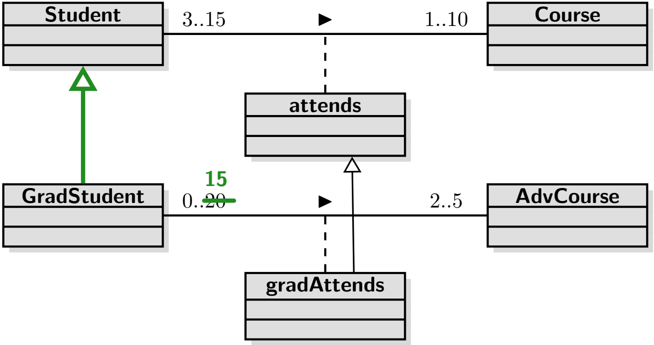
Then $\mathcal{P}$ is an implicit consequence of $\mathcal{D}$ iff

$$\Gamma_{\mathcal{D}} \models \mathcal{P}$$

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

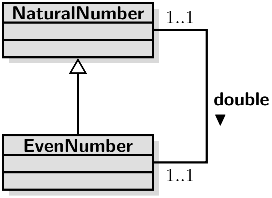
Corresponding FOL reasoning task: logical implication.

Implicit consequences – Example



$$\Gamma_{\mathcal{D}} \models \forall x.\, AdvCourse(x_2) \rightarrow \sharp\{x_1 \mid gradAttends(x_1, x_2)\} \leq 15$$
$$\Gamma_{\mathcal{D}} \models \forall x.\, GradStudent(x) \rightarrow Student(x)$$
$$\Gamma_{\mathcal{D}} \not\models \forall x.\, AdvCourse(x) \rightarrow Course(x)$$

Unrestricted vs. finite model reasoning



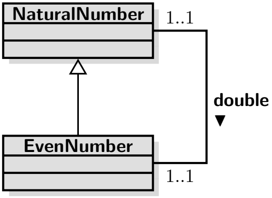
Due to the ISA relationship, every instance of $EvenNumber$ is also an instance of $NaturalNumber$, i.e., we have that

$$\Gamma_{\mathcal{D}} \models \forall x. EvenNumber(x) \rightarrow NaturalNumber(x)$$

Question: Does also the reverse implication hold? I.e.,

$$\Gamma_{\mathcal{D}} \models \forall x. NaturalNumber(x) \rightarrow EvenNumber(x) \quad ?$$

Unrestricted vs. finite model reasoning (cont'd)



- Due to the $1..1$ multiplicities, the classes $NaturalNumber$ and $EvenNumber$ are in bijection.
- Hence, in every instantiation of the diagram, $NaturalNumber$ and $EvenNumber$ contain the same number of instances.

Does it hold that $\Gamma_{\mathcal{D}} \models \forall x. NaturalNumber(x) \rightarrow EvenNumber(x)$?

- If the domain is infinite, the implication does not hold.
- If the domain is finite, the implication does hold.

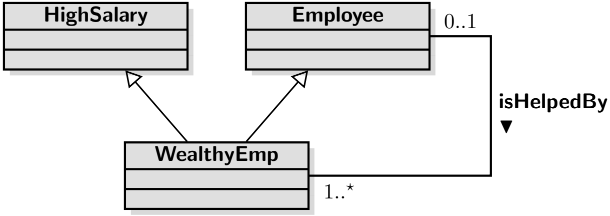
Finite model reasoning

Finite model reasoning

Reasoning is done only with respect to models with a finite domain.

- Finite model reasoning is interesting for standard databases, which are (typically) assumed to be finite.
- The previous example shows that in UML CDs, finite model reasoning is different from unrestricted model reasoning.

Finite model reasoning – Example



Let Paul be an employee: $Employee(Paul)$

Question: Does it follow that Paul has a high salary, i.e.,

$$\Gamma_{\mathcal{D}} \models HighSalary(Paul) ?$$

Answer: If we consider **only finite models**, then the answer is **"yes"**.

Questions

In the above simple 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 CDs?
    - 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 CDs in general?
    - What is the worst-case situation?
    - Can we single out **interesting fragments** on which to reason efficiently?

We will address also **answering queries** over such diagrams, which is in general a more complicated task than satisfiability or subsumption.

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

References I

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