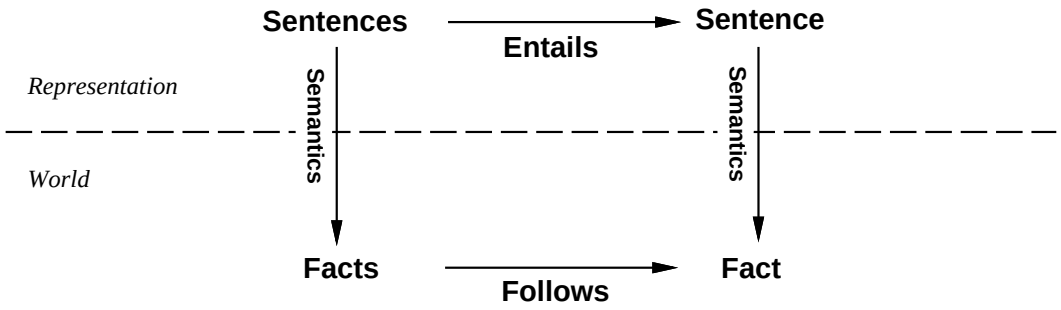
Logic



A logic allows the axiomatization of the domain information, and the drawing of conclusions from that information.

- Syntax
- Semantics
- Logical inference = *reasoning*

Summary: what is a logic

Clearly distinguish the definitions of:

- the *formal language*
  - Semantics
  - Expressive Power
- the *reasoning problem*
  - Decidability
  - Computational Complexity
- the *problem solving procedure*
  - Soundness and Completeness
  - (Asymptotic) Complexity

The ideal computational Logic

- Expressive
- With decidable reasoning problems
- With sound and complete reasoning procedures
- With efficient reasoning procedures – possibly sub-optimal

Description Logics

Description Logics explore the "most" interesting expressive decidable logics with "classical" semantics, equipped with "good" reasoning procedures.

Model Checking

Verify that a given interpretation $\mathcal{I}$ is a model for a closed formula φ:

$\models_{\mathcal{I}} \varphi$

An interpretation is also called a *relational structure*.

Example:

$\Delta = \{a, b\}$,
$P(a)$,
$Q(b)$.

is a model of the formula:

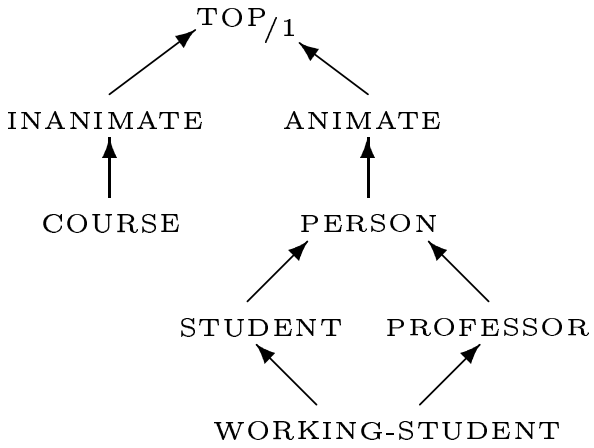
$\exists y.\ [\ P(y) \wedge \neg Q(y)\ ] \wedge \forall z.\ [\ P(z) \vee Q(z)\ ]$

Other Reasoning Problems

- **Subsumption**
  - $\varphi \sqsupseteq \psi$,
    φ and ψ predicate symbols of the same arity
  - φ subsumes ψ
  - $\models \forall \hat{x}.\ [\ \psi(\hat{x}) \rightarrow \varphi(\hat{x})\ ]$
- **Instance Checking**
  - The constant a is an instance of the unary predicate P
  - $\Gamma \models P(a)$

Subsumption

- *Subsumption* can be seen as a binary relation in the space of predicates with same arity: $\varphi_{/n} \sqsupseteq \psi_{/n}$.
- The subsumption relation is a *partial ordering* relation in the space of predicates of same arity.
  - **Exercise**: prove it.
  - *Hint*: a partial ordering relation is a transitive, reflexive, and antisymmetric relation.



Decidability

Given a logic $\mathcal{L}$, a reasoning problem is said to be **decidable** if there exists a computational process (e.g., an algorithm, a Turing Machine, a computer program, etc.) that solves the problem in a finite number of steps, i.e., the process always terminates.

- The problem of deciding whether a formula φ is logically implied by a theory Γ is undecidable in full FOL.
- Logical implication is decidable if we restrict to propositional calculus.
- Logical implication is decidable if we restrict to FOL using only at most *two* variable names; such language is called $\mathcal{L}_2$.

The property of (un)decidability is a general property of the problem and not of a particular algorithm solving it.

Expressive Power

- Some logics can be made decidable by sacrificing some *expressive power*.
- A logical language $\mathcal{L}_a$ has more expressive power than a logical language $\mathcal{L}_b$, if each formula of $\mathcal{L}_b$ denotes the "same" set of models of its correspondent formula of $\mathcal{L}_a$, and if there is a formula of $\mathcal{L}_a$ denoting a set of models which is denoted by no formula in $\mathcal{L}_b$.

Example:

Consider $\mathcal{L}_a$ as FOL, and $\mathcal{L}_b$ as FOL without negation and disjunction. Given a common domain, the $\mathcal{L}_a$ formula $\exists x.\ [\ P(x) \vee Q(x)\ ]$ has a set of models which can not be captured by any formula of $\mathcal{L}_b$.
(Exercise: check it out with $\Delta = \{a\}$

Problems and Algorithms

- A **problem** is a general question to be answered.
  A problem is described by giving:
  - a general description of all its parameters, and
  - a statement of what properties the answer, or *solution*, is required to satisfy.

  An *instance* of a problem is obtained by specifying values for all parameters.
- An **algorithm** is a step-by-step procedure fro solving problems.
  An algorithm is said to *solve* a problem Π if
  - it can be applied to any instance I of Π, and
  - it is guaranteed always to produce a solution for that instance I.

Computational Complexity

- The goal of complexity theory is to classify problems as to their intrinsic computational difficulty into general *complexity classes*.
- Given a problem, how much computing power and/or resources (e.g., *time*, *space*) do we need in order to solve it **in the worst case**?
- The complexity class to which a problem belongs is a general property of the problem and not of a particular algorithm solving it.
- Distinguish among:
  - worst case $\Longleftarrow$
  - average case
  - hard and easy cases

Complexity classes

- P
- NP — coNP
- PH
- PSPACE
- EXPTIME
- NEXPTIME
- DECIDABLE

Complexity of problems

Complexity and expressive power:

- Satisfiability of formulas in propositional calculus is NP-complete.
- Unsatisfiability of formulas in propositional calculus is coNP-complete.
- Satisfiability of QBF formulas in FOL where there is a finite nesting of quantifiers is PSPACE-complete:
  $Q_1 x_1.\ Q_2 x_2.\ \ldots\ [\ \varphi(x_1, x_2, \ldots)\ ]$,
  where Q_i is either $\forall$ or $\exists$.
- Satisfiability of formulas in the propositional *dynamic* normal modal logic (PDL) is EXPTIME-complete. *(Note: PDL has non first order expressible features, but it doesn't include full FOL)*
- Satisfiability of $\mathcal{L}_2$ formulas is NEXPTIME-complete.

Complexity of problems

Computational complexity depends on the reasoning problem:

- Model checking of FOL formulas is polynomial.

Reasoning Procedures

A reasoning procedure is an algorithm trying to solve *specific instances* of a *specific reasoning problem* in a *given logic*.

- Whenever a **sound** reasoning procedure claims to have found a solution for a given instance of the problem, then this is actually a solution.
  - "no wrong inferences are drawn"
  - A sound procedure may fail to find the solution for some instances of the problem, when they do actually have one.
- Whenever an instance of the problem has a solution, a **complete** reasoning procedure computes the solution for that instance.
  - "all the correct inferences are drawn"
  - A complete procedure may claim to have found a solution for some instances of the problem, when they do not have one.

Decision Procedures

A **decision procedure** is a procedure trying to solve a decision problem, i.e., a problem whose only possible answers are YES or NO.

Let's consider two decision procedures:

- F, which always returns the result NO independently from its input
- T, which always returns the result YES independently from its input

Let's consider the problem of checking satisfiability of FOL formulas.

- F is a sound and terminating algorithm for computing satisfiability of FOL formulas.
- T is a complete and terminating algorithm for computing satisfiability of FOL formulas.

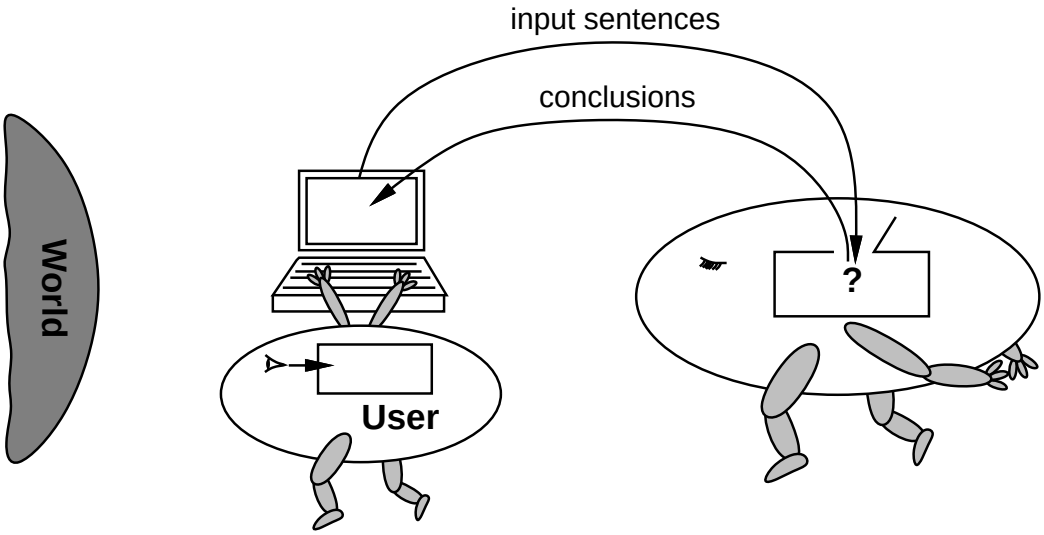
Sound and Incomplete Algorithms

- Sound and incomplete algorithms are very popular: they are considered *good* approximations of problem solving procedures.
- Sound and incomplete algorithms may reduce the algorithm complexity (see later) with respect to the computational worst case complexity.
- Sound and incomplete algorithms are often used due to the inability of programmers to find sound and complete algorithms.

Dual problems

Can we use a sound but incomplete decision procedure for a problem to solve the **dual problem** by inverting the answers?

T is an unsound procedure for computing unsatisfiability of FOL formulas. *(Why?)*



Incompleteness of the reasoning procedures of the reasoning agent leads to *unsound* reasoning of the whole agent, if the main system relies on negative conclusions of the reasoning agent module.

The classical inference systems

Usually proof theory studies the properties of inference systems for logical implication based on some sort of *sequent calculus*, or of *natural deduction*. While this seems appropriate from a mere theoretical point of view, it turns out that such systems are hardly implementable, due to their non-deterministic and non-constructive foundations.

Basically, these proof systems formalize the notion of *derivability* "$\Gamma \vdash \varphi$" with a set of inference rules. In this context:

- Soundness: if $\Gamma \vdash \varphi$ then $\Gamma \models \varphi$
- Completeness: if $\Gamma \models \varphi$ then $\Gamma \vdash \varphi$

Incompleteness

- Sequent calculus and natural deduction form sound and complete procedures for computing logical implication.
- However, for FOL it is not guaranteed their termination, since the problem is undecidable.
- It is easy to have incomplete but terminating procedures, by simply dropping some of the inference rules from the complete set.
- This is a general methodology for characterizing the incompleteness of algorithms: *find a complete set of inference rules, and characterize the incomplete procedure with a subset of them.*

Structural Description Logics – OUTLINE

- Description Logics
  - The need for a formalism
    - O-O ambiguities
  - A structure to FOL
    - A predicate level language
- Examples from O-O
- $\mathcal{FL}^-$: the simplest structural description logics
  - Syntax
  - Semantics
  - Reasoning problems
  - Reasoning procedures

Description Logics

- A logical reconstruction and *unifying* formalism for the representation tools
  - Frame-based systems
  - Semantic Networks
  - Object-Oriented representations
  - Semantic data models
  - Type systems
  - Feature Logics
  - ...
- A *structured* fragment of predicate logic
- Provide theories and systems for *expressing* structured information and for *accessing* and *reasoning* with it in a principled way.

Applications

Description logics based systems are currently in use in many applications.

- Configuration
- Conceptual Modeling
- Query Optimization and View Maintenance
- Natural Language Semantics
- I3 (Intelligent Integration of Information)
- Information Access and Intelligent Interfaces
- Formal Specifications in Engineering
- Terminologies and Ontologies
- Software Management
- Planning
- ...

A formalism

- Description Logics formalize many *Object-Oriented* representation approaches.
- As such, their purpose is to disambiguate many imprecise representations.

Frames or Objects

- Identifier
- Class
- Instance
- Slot (attribute)
  - Value
    - Identifier
    - Default
  - Value restriction
    - Type
    - Concrete Domain
    - Cardinality
    - Attached method

Ambiguities: classes and instances

Person : AGE : Number, SEX : M, F, HEIGHT : Number, WIFE : Person.

john : AGE : 29, SEX : M, HEIGHT : 76, WIFE : *mary*.

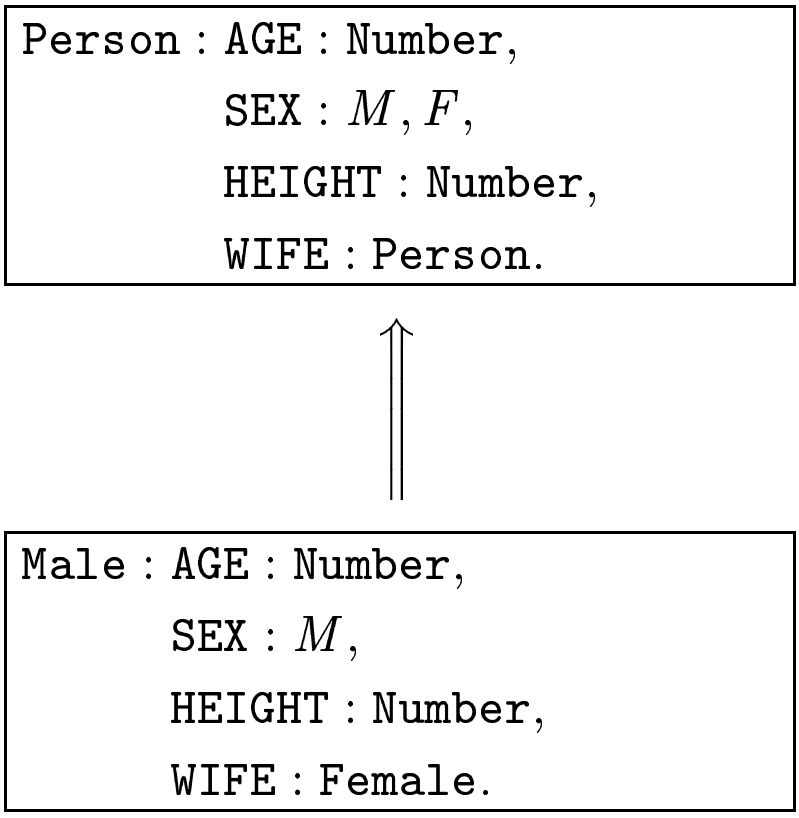
Ambiguities: classes and instances

29'er : AGE : 29, SEX : M, HEIGHT : Number, WIFE : Person.

john : AGE : 29, SEX : M, HEIGHT : Number, WIFE : Person.

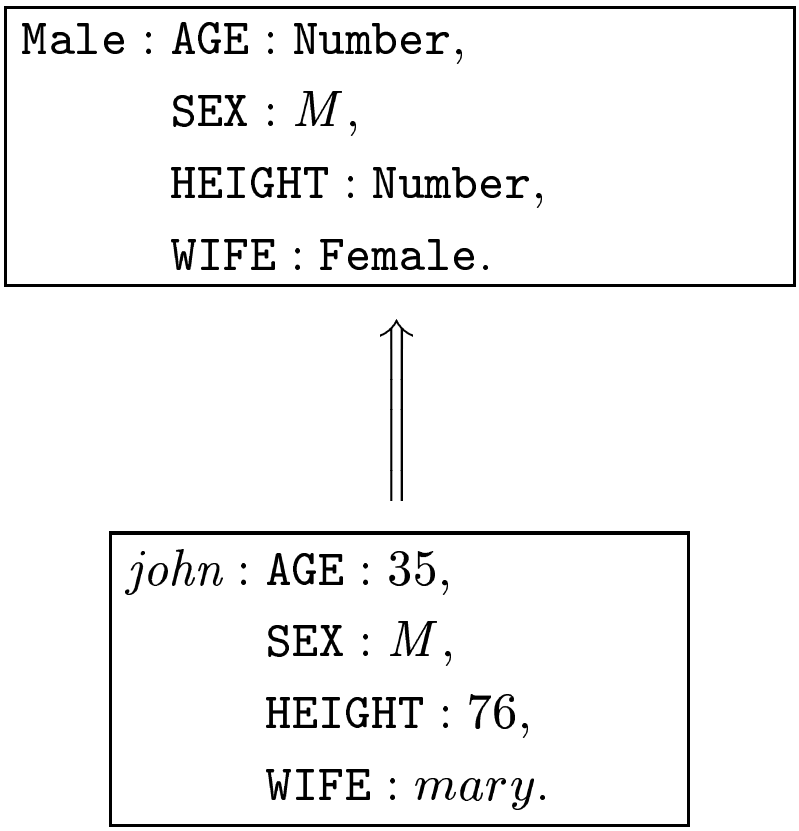
Ambiguities: is-a

Sub-class:



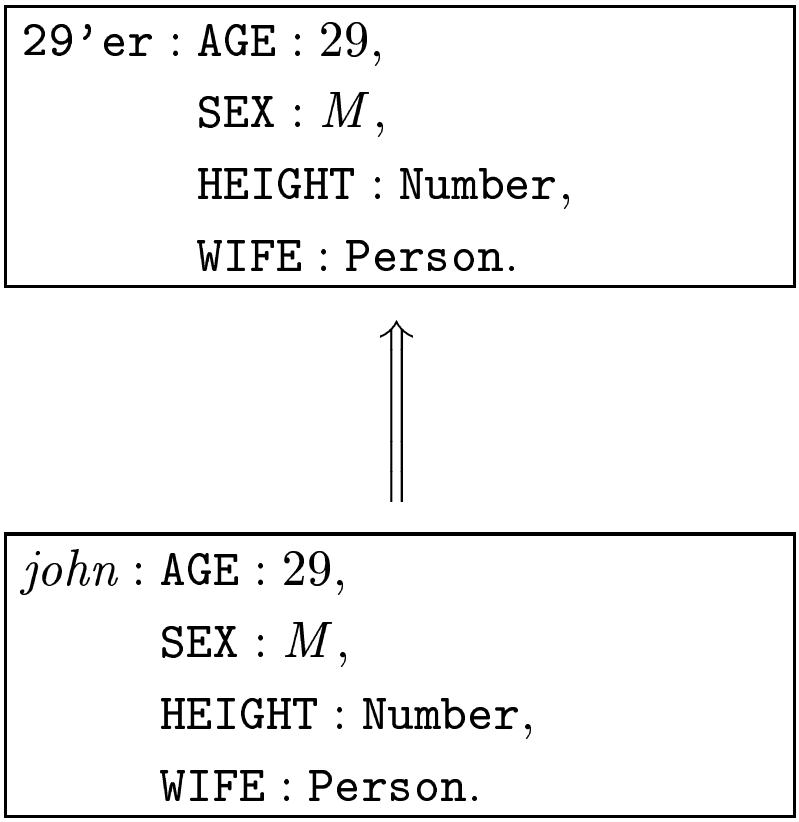
Ambiguities: is-a

Instance-of:



Ambiguities: is-a

Instance-of:



Ambiguities: relations

Implicit relation:

$$\begin{array}{l} john : \texttt{AGE} : 35, \\ \qquad\quad \texttt{SEX} : M, \\ \qquad\quad \texttt{HEIGHT} : 76, \\ \qquad\quad \underline{\texttt{WIFE}} : mary. \end{array}$$

$$\begin{array}{l} mary : \texttt{AGE} : 32, \\ \qquad\quad \texttt{SEX} : F, \\ \qquad\quad \texttt{HEIGHT} : 59, \\ \qquad\quad \underline{\texttt{HUSBAND}} : john. \end{array}$$

Ambiguities: relations

Explicit relation:

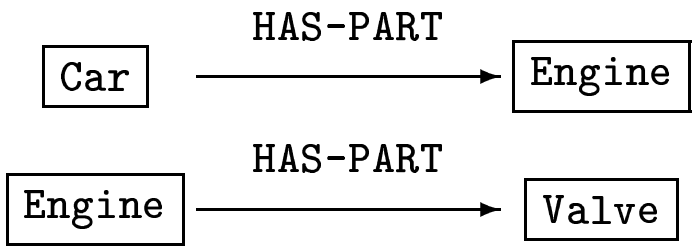
john : AGE : 35,
SEX : *M*,
HEIGHT : 76.

mary : AGE : 32,
SEX : *F*,
HEIGHT : 59.

m-j-family : WIFE : *mary*,
HUSBAND : *john*.

Ambiguities: relations

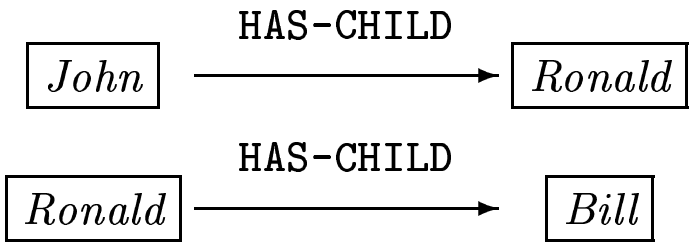
Special relation:



$\Longrightarrow$

Ambiguities: relations

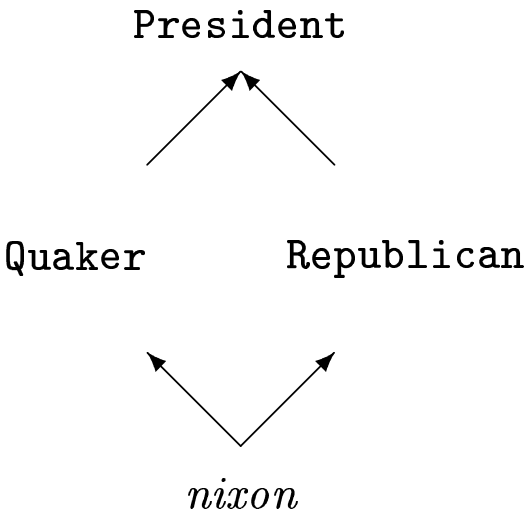
Normal relation:



$\not\Longrightarrow$

Ambiguities: default

The *Nixon* diamond:

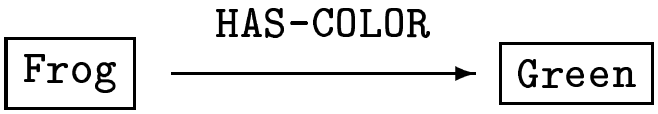


Quakers are pacifist, Republicans are not pacifist.

$\Longrightarrow$ Is Nixon pacifist or not pacifist?

Ambiguities: quantification

What is the exact meaning of:



- Every frog is just green
- Every frog is also green
- Every frog is of some green
- There is a frog, which is just green
- ...
- Frogs are typically green, but there may be exceptions

False friends

- The meaning of object-oriented representations is logically very ambiguous.
- The appeal of the graphical nature of object-oriented representation tools has led to forms of reasoning that do not fall into standard logical categories, and are not yet very well understood.
- It is unfortunately much easier to develop some algorithm that appears to reason over structures of a certain kind, than to *justify* its reasoning by explaining what the structures are saying about the domain.

A structured logic

- Any (basic) Description Logic is a fragment of FOL.
- The representation is at the *predicate level*: no variables are present in the formalism.
- A Description Logic theory is divided in two parts:
  - the definition of predicates ($TBox$)
  - the assertion over constants ($ABox$)
- Any (basic) Description Logic is a subset of $\mathcal{L}_3$, i.e. the function-free FOL using only at most *three* variable names.

Why not FOL

If FOL is directly used without additional restrictions then

- the structure of the knowledge is destroyed, and it can not be exploited for driving the inference;
- the expressive power is too high for obtaining decidable and efficient inference problems;
- the inference power may be too low for expressing interesting, but still decidable theories.

Structured Inheritance Networks:
KLONE

- Structured Descriptions
  - corresponding to the complex relational structure of objects,
  - built using a restricted set of epistemologically adequate constructs
- distinction between conceptual (*terminological*) and instance (*assertional*) knowledge;
- central role of automatic classification for determining the subsumption – i.e., universal implication – lattice;
- strict reasoning, no defaults.

Types of the TBox Language

- **Concepts** – denote *entities*
  (unary predicates, classes)

  Example: `Student, Married`

  $\{x \mid \texttt{Student}(x)\}$,
  $\{x \mid \texttt{Married}(x)\}$

- **Roles**– denote *properties*
  (binary predicates, relations)

  Example: `FRIEND, LOVES`

  $\{\langle x, y\rangle \mid \texttt{FRIEND}(x, y)\}$,
  $\{\langle x, y\rangle \mid \texttt{LOVES}(x, y)\}$

Concept Expressions

Description Logics organize the information in classes – *concepts* – gathering homogeneous data, according to the relevant common properties among a collection of instances.

Example:

$\texttt{Student} \sqcap \exists \texttt{FRIEND.Married}$

$$\{x \mid \texttt{Student}(x) \wedge \exists y.\, \texttt{FRIEND}(x, y) \wedge \texttt{Married}(y)\}$$

A note on λ's

In general, λ is an explicit way of forming *names* of functions:

$\boxed{\lambda x.\ f(x)}$ is the function that, given input x, returns the value $f(x)$

The λ-conversion rule says that:

$$(\lambda x.\ f(x))(a) = f(a)$$

Thus, $\boxed{\lambda x.\ (x^2 + 3x - 1)}$ is the function that applied to 2 gives 9:

$$(\lambda x.\ (x^2 + 3x - 1))(2) = 9$$

We can give a name to this function, so that:

$$f_{231} \doteq \lambda x.\ (x^2 + 3x - 1)$$
$$f_{231}(2) = 9$$

λ to define predicates

Predicates are special case of functions: they are *truth* functions. So, if we think of a formula $P(x)$ as denoting a truth value which may vary as the value of x varies, we have:

$\lambda x.$ $P(x)$ denotes a function from domain individuals to truth values.

In this way, as we have learned from FOL, P denotes exactly the set of individuals for which it is true. So, $P(a)$ means that the individual a makes the predicate P true, or, in other words, that a is in the extension of P.

For example, we can write for the *unary* predicate `Person`:

$$\texttt{Person} \doteq \lambda x.\ \texttt{Person}(x)$$

which is equivalent to say that `Person` denotes the *set* of persons:

$$\begin{gathered}\texttt{Person} \rightsquigarrow \{x \mid \texttt{Person}(x)\} \\ \texttt{Person}^{\mathcal{I}} = \{x \mid \texttt{Person}(x)\} \\ \texttt{Person}(\mathit{john}) \quad \text{IFF} \quad \mathit{john}^{\mathcal{I}} \in \texttt{Person}^{\mathcal{I}}\end{gathered}$$

In the same way for the *binary* predicate `FRIEND`:

$$\begin{gathered}\texttt{FRIEND} \doteq \lambda x, y.\ \texttt{FRIEND}(x, y) \\ \texttt{FRIEND}^{\mathcal{I}} = \{\langle x, y\rangle \mid \texttt{FRIEND}(x, y)\}\end{gathered}$$

The functions we are defining with the λ operator may be parametric:

$$\texttt{Student} \sqcap \texttt{Worker} =$$
$$\lambda x.\ (\texttt{Student}(x) \wedge \texttt{Worker}(x))$$

$$(\texttt{Student} \sqcap \texttt{Worker})^{\mathcal{I}} =$$
$$\{x \mid\ (\texttt{Student}(x) \wedge \texttt{Worker}(x)\}$$

$$(\texttt{Student} \sqcap \texttt{Worker})^{\mathcal{I}} = \texttt{Student}^{\mathcal{I}} \cap \texttt{Worker}^{\mathcal{I}}$$

(Verify as exercise)

Concept Expressions

$$(\texttt{Student} \sqcap \exists \texttt{FRIEND}.\,\texttt{Married})^{\mathcal{I}}$$

$$=$$

$$(\texttt{Student})^{\mathcal{I}} \cap (\exists \texttt{FRIEND}.\,\texttt{Married})^{\mathcal{I}}$$

$$=$$

$$\{x \mid \texttt{Student}(x)\} \cap$$
$$\{x \mid \exists y.\,\texttt{FRIEND}(x, y) \wedge \texttt{Married}(y)\}$$

$$=$$

$$\{x \mid \texttt{Student}(x) \wedge$$
$$\exists y.\,\texttt{FRIEND}(x, y) \wedge \texttt{Married}(y)\}$$

Objects: classes

Student

Person	
name:	[String]
address:	[String]
enrolled:	[Course]

$$\{x \mid \texttt{Student}(x)\} =$$
$$\{x \mid \texttt{Person}(x) \wedge$$
$$(\exists y.\ \texttt{NAME}(x, y) \wedge \texttt{String}(y)) \wedge$$
$$(\exists z.\ \texttt{ADDRESS}(x, z) \wedge \texttt{String}(z)) \wedge$$
$$(\exists w.\ \texttt{ENROLLED}(x, w) \wedge \texttt{Course}(w))\ \}$$

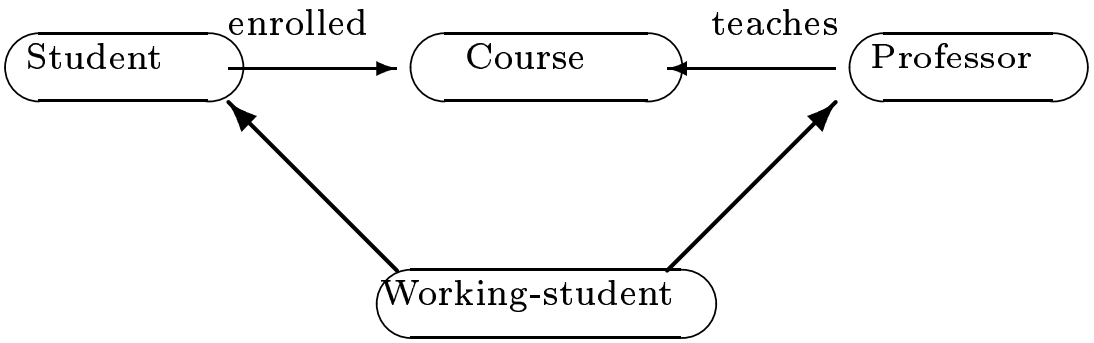
$$\texttt{Student} \doteq \texttt{Person} \sqcap$$
$$\exists \texttt{NAME}.\,\texttt{String} \sqcap$$
$$\exists \texttt{ADDRESS}.\,\texttt{String} \sqcap$$
$$\exists \texttt{ENROLLED}.\,\texttt{Course}$$

Objects: instances

s1: Student	
name:	“John”
address:	“Abbey Road. . .”
enrolled:	cs415

$$\texttt{Student}(\texttt{s1}) \wedge$$
$$\texttt{NAME}(\texttt{s1}, \text{“}\texttt{john}\text{”}) \wedge \texttt{String}(\text{“}\texttt{john}\text{”}) \wedge$$
$$\texttt{ADDRESS}(\texttt{s1}, \text{“}\texttt{abbey-road}\text{”}) \wedge$$
$$\texttt{String}(\text{“}\texttt{abbey-road}\text{”}) \wedge$$
$$\texttt{ENROLLED}(\texttt{s1}, \texttt{cs415}) \wedge \texttt{Course}(\texttt{cs415})$$

Semantic Networks



$\forall x.\ \texttt{Student}(x) \rightarrow$
$\quad \exists y.\ \texttt{ENROLLED}(x, y) \wedge \texttt{Course}(y)$

$\forall x.\ \texttt{Professor}(x) \rightarrow$
$\quad \exists y.\ \texttt{TEACHES}(x, y) \wedge \texttt{Course}(y)$

$\forall x.\ \texttt{Working-student}(x) \rightarrow$
$\quad \texttt{Student}(x) \wedge \texttt{Professor}(x)$

$\texttt{Student} \sqsubseteq \exists \texttt{ENROLLED}.\texttt{Course}$

$\texttt{Professor} \sqsubseteq \exists \texttt{TEACHES}.\texttt{Course}$

$\texttt{Working-student} \sqsubseteq \texttt{Student}$

$\texttt{Working-student} \sqsubseteq \texttt{Professor}$

Quantification

- $\texttt{Frog} \sqsubseteq \exists \texttt{HAS-COLOR}.\texttt{Green}$:
  Every frog is also green
- $\texttt{Frog} \sqsubseteq \forall \texttt{HAS-COLOR}.\texttt{Green}$:
  Every frog is just green
- $\texttt{Frog} \sqsubseteq \forall \texttt{HAS-COLOR}.\texttt{Green}$
  $\texttt{Frog}(x)$, $\texttt{HAS-COLOR}(x, y)$:
  There is a frog, which is just green

Every frog is also green

$\texttt{Frog} \sqsubseteq \exists \texttt{HAS-COLOR}.\texttt{Green}$

$\forall x.\ \texttt{Frog}(x) \rightarrow$
$\quad \exists y.\ (\texttt{HAS-COLOR}(x, y) \wedge \texttt{Green}(y))$

Exercise: is this a model?

$\texttt{Frog}(oscar)$, $\texttt{Green}(green)$,
$\texttt{HAS-COLOR}(oscar,green)$,
$\texttt{Red}(red)$,
$\texttt{HAS-COLOR}(oscar,red)$.

Every frog is only green

$\texttt{Frog} \sqsubseteq \forall \texttt{HAS-COLOR}.\texttt{Green}$

$\forall x.\ \texttt{Frog}(x) \rightarrow$
$\quad \forall y.\ (\texttt{HAS-COLOR}(x, y) \rightarrow \texttt{Green}(y))$

Exercise: is this a model?

$\texttt{Frog}(oscar)$, $\texttt{Green}(green)$,
$\texttt{HAS-COLOR}(oscar,green)$,
$\texttt{Red}(red)$,
$\texttt{HAS-COLOR}(oscar,red)$.

and this?

$\texttt{Frog}(sing)$,
$\texttt{AGENT}(sing,oscar)$.

Objects: adequacy

(Exercise) Check whether the found representation for `Student` is an adequate one.

Student

Person	
name:	[String]
address:	[String]
enrolled:	[Course]

$$\{x \mid \texttt{Student}(x)\} =$$
$$\{x \mid \texttt{Person}(x) \wedge$$
$$(\exists y.\ \texttt{NAME}(x,y) \wedge \texttt{String}(y)) \wedge$$
$$(\exists z.\ \texttt{ADDRESS}(x,z) \wedge \texttt{String}(z)) \wedge$$
$$(\exists w.\ \texttt{ENROLLED}(x,w) \wedge \texttt{Course}(w))\ \}$$

$$\texttt{Student} \doteq \texttt{Person} \sqcap$$
$$\exists \texttt{NAME}.\,\texttt{String} \sqcap$$
$$\exists \texttt{ADDRESS}.\,\texttt{String} \sqcap$$
$$\exists \texttt{ENROLLED}.\,\texttt{Course}$$

Another example

$$
\begin{array}{l}
(\texttt{Student} \sqcap \\
\quad \exists \texttt{dept.CS} \ \sqcap \\
\quad \geq 3 \ \texttt{enrolled-course.} \\
\qquad (\texttt{Graduate-Course} \sqcap \exists \texttt{dept.CS-dept}))
\end{array}
$$

$$
\begin{array}{l}
\lambda x.\ [\texttt{Student}(x) \wedge \texttt{dept}(x, \texttt{CS}) \wedge \\
\quad \exists y_1 y_2 y_3 (y_1 \neq y_2 \wedge y_1 \neq y_3 \wedge y_2 \neq y_3 \wedge \\
\qquad \texttt{enrolled-course}(x, y_1) \wedge \texttt{Grad-Course}(y_1) \wedge \\
\qquad\quad \exists z (\texttt{dept}(y_1, z) \wedge \texttt{CS-dept}(z)) \wedge \\
\qquad \texttt{enrolled-course}(x, y_2) \wedge \texttt{Grad-Course}(y_2) \wedge \\
\qquad\quad \exists z (\texttt{dept}(y_2, z) \wedge \texttt{CS-dept}(z)) \wedge \\
\qquad \texttt{enrolled-course}(x, y_3) \wedge \texttt{Grad-Course}(y_3) \wedge \\
\qquad\quad \exists z (\texttt{dept}(y_3, z) \wedge \texttt{CS-dept}(z)))]
\end{array}
$$

CLUMSY!

Analytic reasoning

(Let's see this intuitively, by now.)

Person

subsumes

(Person **with every** male friend **is a** doctor)

subsumes

(Person **with every** friend **is a**
(Doctor **with a** specialty **is** surgery))

(Person **with** $\geq$ 2 children)

subsumes

(Person **with** $\geq$ 3 male children)

(Person **with** $\geq$ 3 young children)

disjoint

(Person **with** $\leq$ 2 children)

$\mathcal{FL}^-$

- In the following part of this lecture, we will concentrate ourself to the simplest conceivable *structural* description logic: $\mathcal{FL}^-$.
- We will consider $\mathcal{FL}^-$ as a logical language. Thus, we are going to speak about:
  - Syntax
  - Semantics
  - Reasoning problems
    - Decidability
    - Complexity
  - Reasoning procedures
    - Soundness
    - Completeness
    - Asymptotic complexity

The grammar

$$C, D \quad \rightarrow \quad A \mid C \sqcap D \mid \forall R.\, C \mid \exists R$$

$A \in \mathit{atomic\text{-}concept}$

$R \in \mathit{atomic\text{-}role}$

$C, D \in \mathit{concept}$

$$\begin{aligned} \mathit{concept} \quad ::= \quad & \langle \mathit{atomic\text{-}concept} \rangle \mid \\ & \langle \mathit{concept} \rangle \sqcap \langle \mathit{concept} \rangle \mid \\ & \exists \langle \mathit{atomic\text{-}role} \rangle \mid \\ & \forall \langle \mathit{atomic\text{-}role} \rangle . \langle \mathit{concept} \rangle \end{aligned}$$

Alternative grammar

$$
\begin{array}{lll}
\textit{concept} & ::= & \langle \textit{atomic-concept} \rangle \mid \\
 & & (: \texttt{and}\ \langle \textit{concept} \rangle \ldots \langle \textit{concept} \rangle) \mid \\
 & & (: \texttt{some}\ \langle \textit{atomic-role} \rangle) \mid \\
 & & (: \texttt{all}\ \langle \textit{atomic-role} \rangle\ \langle \textit{concept} \rangle)
\end{array}
$$

Intuitive semantics

Intuitively (as we already know):

- *Concepts* represent classes, i.e., sets of individuals.
- *Roles* represent relations between pairs of individuals.

- Atomic concepts are the names of primitive (undefined) concepts.
- `:and` constructions represent conjoined concepts, so, for example, `(:and Adult Male Person)` would represent the concept of something that is at the same time an adult, a male, and a person.
- This allows us to put several properties (i.e. *super-concepts* or attribute restrictions) together in the definition of a concept.

Quantifiers

- The `:all` construct provides a concept restriction on the values of an attribute (x is an `(:all R C)` if and only if each `R` of x is a `C`. Thus, `(:all CHILD Doctor)` corresponds to the concept of something all of whose children are doctors. It is a way to *restrict* the value of a slot at a frame.

- The `:some` operator guarantees that there will be at least one value for the attribute named (x is a `(:some R)` if and only if x has at least one `R`. For instance, `(:and Person (:some CHILD))` would represent the concept of a parent. This is a way to introduce a slot at a frame.

Formal Semantics

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

- a nonempty set Δ (the *domain*)
- a function $\cdot^{\mathcal{I}}$ (the *interpretation function*) that maps
  - every *concept* to a subset of Δ
  - every *role* to a subset of $\Delta \times \Delta$

Extension of concepts

An interpretation function $\cdot^{\mathcal{I}}$ is an extension function iff:

$$\begin{aligned}
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(\forall R.\, C)^{\mathcal{I}} &= \{x \in \Delta \mid \forall y.\ (x,y) \in R^{\mathcal{I}} \Rightarrow \\
&\qquad\qquad y \in C^{\mathcal{I}}\} \\
(\exists R)^{\mathcal{I}} &= \{x \in \Delta \mid \exists y.\ (x,y) \in R^{\mathcal{I}}\}
\end{aligned}$$

Recall that $C^{\mathcal{I}}$ is a set of all the individual in the extension of C: so, writing $x \in C^{\mathcal{I}}$ has the same truth value as $C(x)$. Analogously, $(x,y) \in R^{\mathcal{I}}$ is the same as $R(x,y)$.

Exercises

Choose a domain Δ and an extension functions over Δ; compute the extensions of the following concepts:

- `(:and Adult Male)`
- `(:and Adult Male Rich)`
- `(:all CHILD (:and Adult Male))`
- `(:some CHILD)`
- $\exists$`CHILD` $\sqcap \forall$`CHILD.` ($\exists$`CHILD` $\sqcap$ `Adult`)

The subsumption problem

$\mathtt{C} \sqsubseteq \mathtt{D}$

$\mathtt{C}$ *is subsumed by* $\mathtt{D}$

iff

for any domain Δ
and any extension function $\cdot^{\mathcal{I}}$ over Δ:

$\mathtt{C}^{\mathcal{I}} \subseteq \mathtt{D}^{\mathcal{I}}$

i.e.,

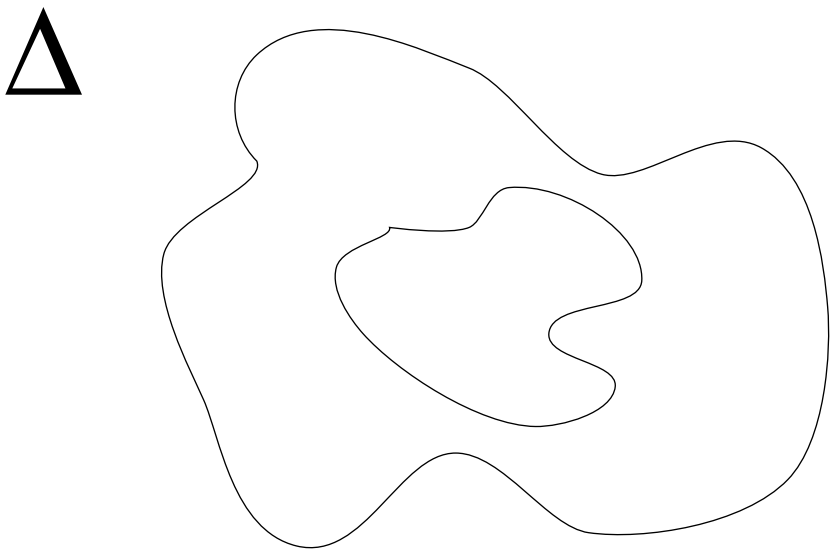
$\forall x.\ C(x) \rightarrow D(x)$

Simple examples

- (:and Adult Male) $\sqsubseteq$ Adult
- (:and Adult Male Rich) $\sqsubseteq$
  (:and Adult Male)
- (:all CHILD (:and Adult Male)) $\sqsubseteq$
  (:all CHILD Adult)
- (:and (:all CHILD Adult)(:some CHILD)) $\sqsubseteq$
  (:all CHILD Adult)
- (:all CHILD Adult) $\not\sqsubseteq$ (:some CHILD)
- (:some CHILD) $\not\sqsubseteq$ (:all CHILD Adult)

The universal quantifier

$(\forall R.\, C)^{\mathcal{I}} =$



Computational properties

Subsumption for $\mathcal{FL}^-$ has the following computational properties, which will be proved constructively:

- Decidable
- in P

The subsumption *structural* algorithm

- The algorithm for computing Subsumption is based on structural comparisons between concept expressions.
- At the heart of structural comparison is the idea that if the two concept expressions to be compared are made of subexpressions, one can compare separately one subexpression of a concept with all those of the others.

The normal form

The algorithm works in two phases: first, concepts are rewritten in a normal form, then their structures are compared.

Normal Form:

1. All nested conjunctions are flattened, i.e. $A \sqcap (B \sqcap C) \rightsquigarrow A \sqcap B \sqcap C$.
2. All conjunctions of universal quantifications are factorized, i.e. $\forall R.\, C \sqcap \forall R.\, D \rightsquigarrow \forall R.\, (C \sqcap D)$.

The rewritten concepts are logically equivalent to the previous ones, hence subsumption is preserved by this transformation.

(Exercise: prove it)

The core algorithm: SUBS?[C,D]

Let $C = C_1 \sqcap \cdots \sqcap C_n$ and $D = D_1 \sqcap \cdots \sqcap D_m$ (in normal form).

Then SUBS?[C,D] returns TRUE if and only if for all C_i:

1. if C_i is either an atomic concept, or is a concept of the form $\exists R$, then there exists a D_j such that $C_i = D_j$;
2. if C_i is a concept of the form $\forall R.\, C'$, then there exists a D_j of the form $\forall R.\, D'$ (same atomic role R) such that SUBS?[C',D'].

Simple exercises

Check the following subsumption using the structural algorithm:

- `(:and Adult Male)` $\sqsubseteq$ `Adult`
- `(:and Adult Male Rich)` $\sqsubseteq$ `(:and Adult Male)`
- `(:all CHILD (:and Adult Male))` $\sqsubseteq$ `(:all CHILD Adult)`
- `(:and (:all CHILD Adult)(:some CHILD))` $\sqsubseteq$ `(:all CHILD Adult)`
- `(:all CHILD Adult)` $\not\sqsubseteq$ `(:some CHILD)`
- `(:some CHILD)` $\not\sqsubseteq$ `(:all CHILD Adult)`

Asymptotic complexity

- By induction on the nesting of $\forall$-quantifiers, one can prove that the complexity of the above algorithm is $O(|C| \times |D|)$ (i.e., quadratic in the length of the longest argument).
- If subexpressions of each concept are ordered (e.g. lexicographically), then it can be shown that the complexity is only linear, so the dominant factor becomes the complexity of ordering concepts.

Soundness

Remember: whenever a **sound** reasoning procedure claims to have found a solution for a given instance of the problem, then this is actually a solution.

The structural subsumption algorithm is sound since, when it says that a concept $\mathtt{C}$ subsumes a concept $\mathtt{D}$ – i.e., SUBS?$[C, D]$ returns true – then it holds that $\mathtt{D}^{\mathcal{I}} \subseteq \mathtt{C}^{\mathcal{I}}$ for all interpretations.

Observation: the part of the algorithm computing the normal form does not change the extension of the concepts for any interpretation; thus it does not affect the soundness (and the completeness) of the algorithm.

Informal proof

- Suppose that SUBS?$[C, D]$ returns TRUE and consider one of the conjuncts of C – call it C_i.
- Either C_i is among the D_j, or it is of the form $\forall R.\, C'$.
- In the latter case, there is a $\forall R.\, D'$ among the D_j, where SUBS?$[C', D']$.
- Then, by induction, any extension of D' must be a subset of C', and so any extension of D_j must be a subset of C_i's.
- So, no matter what C_i is, the extension of D – which is the conjunction of all the D_j's – must be a subset of C_i.
- Since this is true for every C_i, the extension of D must also be a subset of the extension of C – which is the intersection of all the extensions of C_i.

- So, whenever SUBS?$[C, D]$ returns TRUE, C subsumes D, i.e., $\mathtt{D}^{\mathcal{I}} \subseteq \mathtt{C}^{\mathcal{I}}$.

Completeness

Remember: whenever an instance of the problem has a solution, a **complete** reasoning procedure computes the solution for that instance.

The structural subsumption algorithm is complete since, whenever it holds that $\mathtt{C}^{\mathcal{I}} \subseteq \mathtt{D}^{\mathcal{I}}$ for all interpretations, then the algorithm says that `C` subsumes `D`.

Observation: The proof is done by showing that anytime SUBS?$[C, D]$ returns FALSE, there exists an interpretation assigning an element to D but not to C, i.e., in that interpretation the extension on C is not a superset of the extension of D.

Idea of the proof

- The proof is done by showing that anytime SUBS?$[C, D]$ returns FALSE, there exists an interpretation assigning an element to D but not to C, i.e., in that interpretation the extension on C is not a superset of the extension of D.
- This shows a counter-example, i.e., anytime SUBS?$[C, D]$ returns FALSE it is not true that $\mathtt{D}^{\mathcal{I}} \subseteq \mathtt{C}^{\mathcal{I}}$ for that particular interpretation, and so $C \not\sqsubseteq D$.

Idea of the proof *(cont.)*

- The proof relies on the fact that anytime SUBS?$[C, D]$ returns FALSE it is possible to find a conjunct C_i of C which has no correspondent conjunct in D.
- It is shown that in this case there exists an interpretation which assigns an object to any primitive concept, but not to the factorized one C_i.
- Thus, it is not possible that $\mathtt{D}^{\mathcal{I}} \subseteq \mathtt{C}^{\mathcal{I}}$.

Structural algorithms or normalize-and-compare

What if we enrich the expressivity?

- CLASSIC
- BACK
- LOOM

These systems have incomplete algorithms, due to the interactions between constructors, which can not be taken into account by structural algorithms, which is based on syntactical comparisons between subexpressions of the concepts.

Example: $A \sqcup \neg A$ subsumes every concept, even if such a concept does not mention at all the atomic concept A in its definition.

Basic References

- H. J. Levesque and R. J. Brachman. Expressiveness and tractability in knowledge representation and reasoning. Computational Intelligence journal 3, 78-93 (1987).

- B. Nebel. Reasoning and Revision in Hybrid Representation Systems. Lecture Notes in Artificial Intelligence 422, Springer-Verlag, 1990. *Chapters 1 – 6.*

- Woods, W., Schmolze, J., 'The KL-ONE Family', Computers and Mathematics with Applications, special issue: Semantic Networks in Artificial Intelligence, Vol 2-5, pp 133-177, 1992.

Disjunction and Negation

$\texttt{Teaching-Assistant} \sqsubseteq \neg\texttt{Undergrad} \sqcup \texttt{Professor}$

$\forall x.\ \texttt{Teaching-Assistant}(x) \rightarrow$
$\neg\texttt{Undergrad}(x) \vee \texttt{Professor}(x)$

A necessary condition in order to be a teaching assistant is to be either not undergraduated or a professor. Clearly, a graduated student being a teaching assistant is not necessarily a professor; moreover, it may be the case that some professor is not graduated.

The "$\sqsubseteq$" symbol introduces a *primitive definition* – giving only necessary conditions – while the "$\doteq$" symbol introduces a real definition – with necessary and sufficient conditions.

$\texttt{Teaching-Assistant} \doteq \neg\texttt{Undergrad} \sqcup \texttt{Professor}$

$\forall x.\ \texttt{Teaching-Assistant}(x) \leftrightarrow$
$\neg\texttt{Undergrad}(x) \vee \texttt{Professor}(x)$

Syntax and Semantics of $\mathcal{ALC}$ the simplest propositional DL

A	$A^{\mathcal{I}} \subseteq \Delta$	primitive concept
R	$R^{\mathcal{I}} \subseteq \Delta \times \Delta$	primitive role
$\top$	Δ	top
$\bot$	$\emptyset$	bottom
$\neg C$	$\Delta \setminus C^{\mathcal{I}}$	complement
$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	conjunction
$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$	disjunction
$\forall R.\, C$	$\{x \mid \forall y.\, R^{\mathcal{I}}(x,y) \rightarrow C^{\mathcal{I}}(y)\}$	universal quant.
$\exists R.\, C$	$\{x \mid \exists y.\, R^{\mathcal{I}}(x,y) \ \wedge C^{\mathcal{I}}(y)\}$	existential quant.

Closed Propositional Language

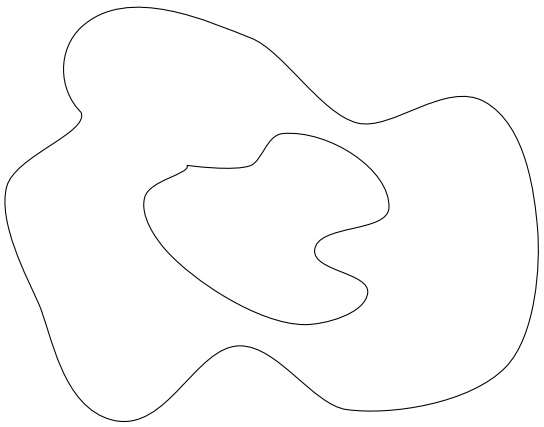
- **Conjunction** is interpreted as *intersection* of sets of individuals.
- **Disjunction** is interpreted as *union* of sets of individuals.
- **Negation** is interpreted as *complement* of sets of individuals.

- $\exists R.\top \Longleftrightarrow \exists R$
- $\neg(C \sqcap D) \Longleftrightarrow \neg C \sqcup \neg D$
- $\neg(C \sqcup D) \Longleftrightarrow \neg C \sqcap \neg D$
- $\neg(\forall R.\, C) \Longleftrightarrow \exists R.\neg C$
- $\neg(\exists R.\, C) \Longleftrightarrow \forall R.\neg C$

Negating Universal formulæ

- $\neg(\forall R.\, C) \Longrightarrow \exists R.\, \neg C$
- $\neg(\exists R.\, C) \Longrightarrow \forall R.\, \neg C$

Δ



(Compare with $\mathcal{FL}^-$ expressivity)

Formal Semantics

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

- a nonempty set Δ (the *domain*)
- a function $\cdot^{\mathcal{I}}$ (the *interpretation function*) that maps
  - every *concept* to a subset of Δ
  - every *role* to a subset of $\Delta \times \Delta$
  - every *individual* to an element of Δ

An interpretation function $\cdot^{\mathcal{I}}$ is an **extension** function if and only if it satisfies the semantic definitions of the language.

Knowledge Bases

$$\boxed{\Sigma = \langle \mathsf{TBox}, \mathsf{Abox} \rangle}$$

- **Terminological Axioms:** $C \sqsubseteq D$
  (where $C \doteq D$ iff $C \sqsubseteq D$ and $D \sqsubseteq C$)
  - $\texttt{Student} \doteq \texttt{Person} \sqcap$
    $\exists \texttt{NAME}.\texttt{String} \sqcap$
    $\exists \texttt{ADDRESS}.\texttt{String} \sqcap$
    $\exists \texttt{ENROLLED}.\texttt{Course}$
  - $\texttt{Student} \sqsubseteq \exists \texttt{ENROLLED}.\texttt{Course}$
  - $\exists \texttt{TEACHES}.\texttt{Course} \sqsubseteq$
    $\neg \texttt{Undergrad} \sqcup \texttt{Professor}$
- **Membership statements:** $C(a), R(a, b)$
  - $\texttt{Student}(\texttt{john})$
  - $\texttt{ENROLLED}(\texttt{john}, \texttt{cs415})$
  - $(\texttt{Student} \sqcup \texttt{Professor})(\texttt{paul})$

TBox: descriptive semantics

Different semantics have been proposed for the TBox, depending on the fact whether cyclic statements are allowed or not.

- An interpretation $\mathcal{I}$ *satisfies* the statement $C \dot{\sqsubseteq} D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.
- An interpretation $\mathcal{I}$ *satisfies* the statement $C \doteq D$ if $C^{\mathcal{I}} = D^{\mathcal{I}}$.

An interpretation $\mathcal{I}$ is a *model* for a TBox $\mathcal{T}$ if $\mathcal{I}$ satisfies all the statements in $\mathcal{T}$.

ABox

If $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ is an interpretation,

- $C(a)$ is satisfied by $\mathcal{I}$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$.
- $R(a, b)$ is satisfied by $\mathcal{I}$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$.

A set $\mathcal{A}$ of assertions is called an ABox.

An interpretation $\mathcal{I}$ is said to be a *model* of the ABox $\mathcal{A}$ if every assertion of $\mathcal{A}$ is satisfied by $\mathcal{I}$. The ABox $\mathcal{A}$ is said to be *satisfiable* if it admits a model.

An interpretation $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ is said to be a *model* of a knowledge base Σ if every axiom of Σ is satisfied by $\mathcal{I}$.
A knowledge base Σ is said to be *satisfiable* if it admits a model.

Logical Implication

$\Sigma \models \varphi$ if every model of Σ is a model of φ

Example:

TBox:

$\exists \texttt{TEACHES.Course} \sqsubseteq$

$\neg \texttt{Undergrad} \sqcup \texttt{Professor}$

ABox:

$\texttt{TEACHES}(\texttt{john}, \texttt{cs415}), \texttt{Course}(\texttt{cs415}),$

$\texttt{Undergrad}(\texttt{john})$

$\Sigma \models \texttt{Professor}(\texttt{john})$

Logical Implication

What if:

TBox:

$\exists \texttt{TEACHES.Course} \sqsubseteq$

$\texttt{Undergrad} \sqcup \texttt{Professor}$

ABox:

$\texttt{TEACHES}(\texttt{john}, \texttt{cs415}), \texttt{Course}(\texttt{cs415}),$

$\texttt{Undergrad}(\texttt{john})$

$\Sigma \stackrel{?}{\models} \texttt{Professor}(\texttt{john})$

$\Sigma \stackrel{?}{\models} \neg\texttt{Professor}(\texttt{john})$

Reasoning Services

- **Concept Satisfiability**

  $\Sigma \not\models C \equiv \bot$ $\qquad$ $\texttt{Student} \sqcap \neg\texttt{Person}$

  the problem of checking whether C is satisfiable w.r.t. Σ, i.e. whether there exists a model $\mathcal{I}$ of Σ such that $C^{\mathcal{I}} \neq \emptyset$

- **Subsumption**

  $\Sigma \models C \sqsubseteq D$ $\qquad$ $\texttt{Student} \sqsubseteq \texttt{Person}$

  the problem of checking whether C is subsumed by D w.r.t. Σ, i.e. whether $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ in every model $\mathcal{I}$ of Σ

- **Satisfiability**

  $\Sigma \not\models$ $\qquad$ $\texttt{Student} \doteq \neg\texttt{Person}$

  the problem of checking whether Σ is satisfiable, i.e. whether it has a model

- **Instance Checking**

  $\Sigma \models C(a)$ $\qquad$ $\texttt{Professor}(\texttt{john})$

  the problem of checking whether the assertion $C(a)$ is satisfied in every model of Σ

Reasoning Services *(cont.)*

- **Retrieval**

  $\{a \mid \Sigma \models C(a)\}$ $\quad$ `Professor` $\Rightarrow$ `john`

- **Realization**

  $\{C \mid \Sigma \models C(a)\}$ $\quad$ `john` $\Rightarrow$ `Professor`

Reduction to satisfiability

- Concept Satisfiability

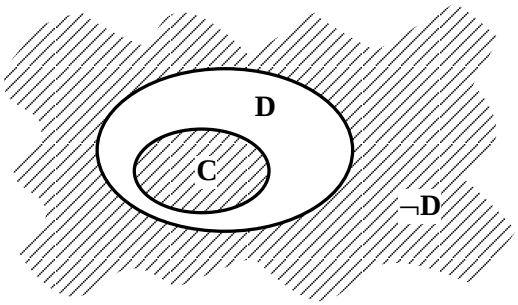
  $\Sigma \not\models C \equiv \bot \quad \leftrightarrow$

  exists x s.t. $\Sigma \cup \{C(x)\}$ has a model

- Subsumption

  $\Sigma \models C \sqsubseteq D \quad \leftrightarrow$

  $\Sigma \cup \{(C \sqcap \neg D)(x)\}$ has no models

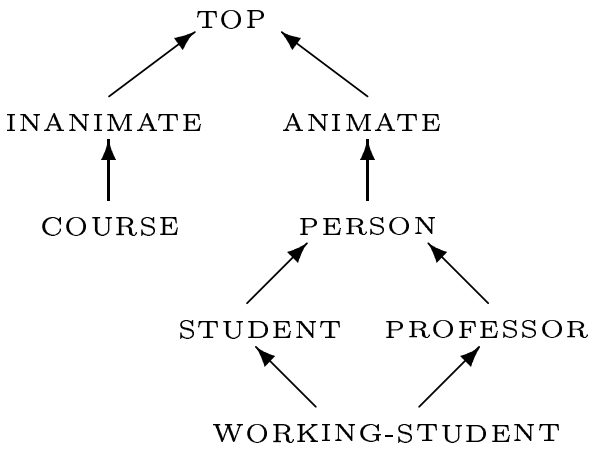


- Instance Checking

  $\Sigma \models C(a) \quad \leftrightarrow$

  $\Sigma \cup \{\neg C(a)\}$ has no models

The Taxonomy



- The subsumption relation is a *partial ordering* relation in the space of concepts.
- If we consider only *named* concepts, subsumption induces a taxonomy – i.e. a generalization/specialization hierarchy – where only direct subsumptions are explicitly drawn.
- A taxonomy is the minimal relation in the space of named concepts such that its rlflexive-transitive closure is the subsumption relation.

Classification

- Given a concept C and a TBox $\mathcal{T}$, for all concepts D of $\mathcal{T}$ determine whether D subsumes C, or D is subsumed by C.
- Intuitively, this amounts to finding the "right place" for C in the taxonomy implicitly present in $\mathcal{T}$.
- *Classification* is the task of inserting new concepts in a taxonomy. It is *sorting* in partial orders.

Reasoning procedures

- Terminating, efficient and complete algorithms for deciding **satisfiability** – and all the other reasoning services – are available.
- Algorithms are based on tableaux-calculi techniques.
- Completeness is important for the usability of description logics in real applications.
- Such algorithms are efficient for both average and real knowledge bases, even if the problem in the corresponding logic is in PSPACE or EXPTIME.

Tableaux Calculus

The Tableaux Calculus is a decision procedure solving the problem of satisfiability.

If a formula is satisfiable, the procedure will constructively exhibit a model of the formula.

The basic idea is to incrementally build the model by looking at the formula, by decomposing it in a top/down fashion. The procedure exhaustively looks at all the possibilities, so that it can eventually prove that no model could be found for unsatisfiable formulas.

Tableaux Calculus

1. Syntactically transform a theory Σ in a *Constraint System* S – also called *tableaux*. Every formula of Σ is transformed into a *constraint* in S.

2. Add constraints to S, applying specific *completion rules*.
   Completion rules are either deterministic – they yield a uniquely determined constraint system – or nondeterministic – yielding several possible alternative constraint systems (*branches*).

3. Apply the completion rules until either a contradiction (a *clash*) is generated in every branch, or there is a *completed* branch where no more rule is applicable.

4. The completed constraint system gives a model of Σ; it corresponds to a particular branch of the tableaux.

Complexity
with *non-cyclic* terminologies

Expressivity		$\Sigma \models C \sqsubseteq D$	$\Sigma \not\models$	$\Sigma \models C(a)$
$C \sqcap D$ $\forall R.C$ $\exists R$	$\mathcal{FL}^-$	P *(*)*	P	P *(*)*
$\neg A$	$\mathcal{AL}$	P *(*)*	P *(*)*	P *(*)*
$\exists R.C$	$\mathcal{ALE}$	NP	coNP	PSPACE
$\neg C$	$\mathcal{ALC}/K(n)$	PSPACE $\Longleftarrow$!!		
$\{a_1 \ldots\}$	$\mathcal{ALCO}$	PSPACE		
	$\mathcal{PDL}$	EXPTIME		
	KL-ONE	undecidable		

Note (*): with expanded terminologies

Extensions of $\mathcal{ALC}$: cardinality, enumeration, and functions

Constructor	Syntax	Semantics
concept name	A	$A^{\mathcal{I}} \subseteq \Delta$
top	$\top$	Δ
bottom	$\bot$	$\emptyset$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
disjunction ($\mathcal{U}$)	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
negation ($\mathcal{C}$)	$\neg C$	$\Delta \setminus C^{\mathcal{I}}$
universal	$\forall R.\, C$	$\{x \mid \forall y : R^{\mathcal{I}}(x, y) \rightarrow C^{\mathcal{I}}(y)\}$
existential ($\mathcal{E}$)	$\exists R.\, C$	$\{x \mid \exists y : R^{\mathcal{I}}(x, y) \wedge C^{\mathcal{I}}(y)\}$
cardinality ($\mathcal{N}$)	$\geq n\ R$	$\{x \mid \sharp\{y \mid R^{\mathcal{I}}(x, y)\} \geq n\}$
	$\leq n\ R$	$\{x \mid \sharp\{y \mid R^{\mathcal{I}}(x, y)\} \leq n\}$
enumeration ($\mathcal{O}$)	$\{a_1 \ldots a_n\}$	$\{a_1^{\mathcal{I}}, \ldots, a_n^{\mathcal{I}}\}$
selection ($\mathcal{F}$)	$f : C$	$\{x \in \mathrm{dom}\, f^{\mathcal{I}} \mid C^{\mathcal{I}}(f^{\mathcal{I}}(x))\}$

Concept-forming constructors.

($\mathcal{ALC}$ has same expressivity as $\mathcal{ALCUE}$)

Extending Description Logics

- Aggregation and Abstraction operators
- Epistemic queries
- Closed world assumption
- Negation as failure
- Default values
- Beliefs
- Probability- and similarity-based reasoning
- Generalized quantifiers and plural entities
- Ontological primitives
  - time
  - events
  - space
  - parts and wholes
  - topology

Understanding Knowledge Bases

$$\boxed{\Sigma = \langle \mathsf{TBox}, \mathsf{Abox} \rangle}$$

- **Terminological Axioms:** $C \sqsubseteq D$
- **Assertional Axioms:** $C(a), R(a, b)$

- An interpretation $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ *satisfies* the statement $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.
- $\mathcal{I}$ satisfies $C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$.
- $\mathcal{I}$ satisfies $R(a, b)$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$.

An interpretation $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ is said to be a *model* of Σ if every axiom of Σ is satisfied by $\mathcal{I}$. Σ is said to be *satisfiable* if it admits a model.

TBox statements

(1)	$A \sqsubseteq C$	Primitive concept definition
(2)	$A \doteq C$	Concept definition
(3)	$C \sqsubseteq D$	Concept inclusion
(4)	$C \doteq D$	Concept equation

Acyclic simple TBox

(1)	$A \sqsubseteq C$	Primitive concept definition
(2)	$A \doteq C$	Concept definition

Acyclic TBox: well-founded definitions.

A concept name A *directly uses* a concept name B in a TBox Σ iff the definition of A mentions B. A concept name A *uses* a concept name B_n iff there is a chain of concept names $\langle A, B_1, \ldots, B_n \rangle$ such that B_i directly uses B_{i+1}. A TBox is *acyclic* iff no concept name uses itself.

Free TBox

(3)	$C \sqsubseteq D$	Concept inclusion
(4)	$C \doteq D$	Concept equation

(There is no syntactic constraint on the left hand side of the axiom).

Concept inclusions make sense only with descriptive semantics – we will ignore here the (rather cumbersome) extensions of Descritpion Logics where it is possible to specify explicitly the semantics to be given to a knowledge base.

Theorem:

General concept definitions $A \doteq D$ and concept inclusions $C \sqsubseteq D$ have the same expressive power.

Simple TBoxes and Free TBoxes

Satisfiability in a knowledge base Σ with only concept inclusions $C_j \sqsubseteq D_j$ can be reduced into satisfiability in a knowledge base Σ' with only concept definitions $A_i \doteq D_i$.

$$A_i \doteq D_i \quad \Longrightarrow \quad A_i \sqsubseteq D_i,\ C_i \sqsubseteq A_i$$

$$C_j \sqsubseteq D_j \quad \Longrightarrow$$

$$A \doteq (\neg C_1 \sqcup D_1) \sqcap \cdots \sqcap (\neg C_n \sqcup D_n) \sqcap A^* \sqcap \forall R_1 . A \sqcap \cdots \sqcap \forall R_m . A$$

where A is a new concept name not appearing in Σ and R_i are all the role names appearing in Σ.

The process of eliminating general axioms is called *internalization*. The above simple TBox emulates the general axiom

$$(\neg C_1 \sqcup D_1) \sqcap \cdots \sqcap (\neg C_n \sqcup D_n) \doteq \top$$

Tests

- In order to allow *procedures* to be used in specifying concepts, the `TEST-C` operator is used.
- A test restriction requires that an individual must pass the test to satisfy the restriction.

Example:

$\texttt{Even-integer} \doteq \texttt{Integer} \sqcap (\texttt{:TEST-C}\ \textsc{evenp})$

where EVENP is a function in the host language.

The reasoning procedures are changed in a way that individuals *passing* some test function "f" will be in the extension of the concept (`:TEST-C f`). The individual to be tested is passed as an argument to the function.

Test functions

Test functions return one of three values when applied to an individual:

- FALSE: the individual is inconsistent with the restriction.
- UNKNOWN: the individual is consistent with the restriction, but if information is added to the individual, the individual may become either inconsistent with or provably described by the restriction.
- TRUE: the individual definitely passes the test, i.e., it provably satisfies it.

Test functions must be monotonic; that is, it should not be possible for the same test function to return TRUE (or FALSE) for an individual at one time, and FALSE (or TRUE) at a later time, when the information regarding the individual has been refined.

Forward-chaining Rules

- A rule consists of an antecedent and a consequent.
- An antecedent is always a concept name.
- A consequent is a generic concept description.

As soon as an individual is known to be in the extension of the antecedent concept, the rule is triggered, and the individual is also known to satisfy the consequent concept.

Forward-chaining Rules

- Intuitively, a rule $A \Longrightarrow C$ seems to have the same semantics as $A \sqsubseteq C$, in the sense that every individual in A should be also in C.
- However, a careful formal analysis shows that a rule is an autoepistemic statement, with a nonmonotonic behaviour.
- Rules play a role only for individuals, and not for concept definitions.
- They are mostly used for expressing contingent properties.

Integrity Checking

- The aim of integrity checking is to support the activity of populating the KB with additional checks on the structures of the individuals w.r.t. to the concepts they are instances of.
- This can not be obtained by adding extra restrictions (tests) to the definition of the concepts, because this would modify it (with problems for the classification).
- The solution: Rule + Test.

Example:

`Late−harvest−grape` $\Longrightarrow$ (`:TEST-C` SUGAR>30)

Where the test function SUGAR>30 provides (if it is the case) to remove the candidate instance from the `Late-harvest-grape` concept, too.

Rule + Test can also emulate *methods*.

Building Knowledge Bases

In order to build good KBs some choices must be done during its design. It is important to well understand some subtle distinctions:

- Primitive vs. Defined.
- Definitional vs. Incidental.
- Concept vs. Individual.
- Concept vs. Role.

When to Use Primitive Concepts?

- some concepts can not be completely defined (e.g. natural kinds);
- it can be not convenient/useful to completely define a concept;
- sooner or later we must end up with something not completely defined (*encyclopedic knowledge* cannot be given).

Thus, primitive concepts must be used when:

- there is no other way;
- even if it were defined, no (automatic) classification below it will be never required by the application.

Typically primitive concepts lie in the top region of the taxonomy.

When to Use Defined Concepts?

- ontological reason: it is easy and natural (in the context of the application) to give a complete definition of the concept;
- organization of the antecedents of rules;
- capturing complete descriptions used by rules for populating primitive concepts.

Definitional vs. Incidental

Are *incidental* all the properties that are contingent features for a concept, and thus must not be part of its definition.

Examples:

(∀SUGAR.Dry)

is incidental for the concept RED-BORDEAUX-WINE, while

(∀COLOR.Red) and (∀REGION.Bordeaux)

are not.

(∀INTELLIGENCE.Stupid)

is incidental for CHICKEN, while

(∀REPRODUCES−WITH.Egg)

is not.

Concept vs. Individual

- the set of individuals is a countable, discrete set;
- the concept space is ideally continuous and infinite;
- each individual has a clear identity: even if two individual have the same properties, they are distinct;
- if two concepts have equivalent descriptions, they **denote the same** concept;
- individual descriptions can be modified;
- concept descriptions can not be modified;
- individual update does not (usually) change the concept hierarchy;
- rules applies only to individuals.

Nevertheless, it is not always easy to decide whether an object should be a concept or an individual. The main issue to deal with is the "granularity" level.

Example:

Consider the KB describing courses in a Computer Science department : is "Introduction To Data Structures And Algorithms (503)" course a concept or an individual?

Individual vs. Concept

Another example: if we have `Wine` and `White-wine`, what about:

- chardonay-wine
- forman-chardonay
- 1981-forman-chardonay
- 1981-forman-chardonay-from-vineyard32
- 1981-forman-chardonay-from-in-cask18
- 1981-forman-chardonay-bottle#1576

A key to solve the problem could be asking the domain/application expert: "how many wines do you have?", in order to understand the needed granularity.

Concept vs. Role

Is not always easy to decide what must be a concept and what a role.

E.g.:

- PERSON: it is a concept.
- MOTHER:
  - consider "Sue is a new mother" and "Sue is the mother of Tom"
  - `Mother` as a concept does not exist if we don't consider the "role she plays" in a parental relation, i.e., if "Sue is a new mother" she must be the `MOTHER` of somebody!

Thus:

$\texttt{Mother} \doteq (\texttt{Woman} \sqcap (\exists \texttt{MOTHER.Person}))$

- But when the role is not the only important component of the definition, this dual use is not so neat (consider `VINTAGE`, `GRAPE`).
- Another problem is the *reading direction*: in the above example the role could be, `MOTHER` or `CHILD`.
- A clear convention must be stated, possibly creating long, non ambiguous names for roles, as, e.g.: `HAS-CHILD`, `IS-THE-PARENT-OF`, `HAS-VINTAGE`, `HAS-GRAPE`.
- As an alternative, the adoption of long names for concepts can be also suggested: e.g., `WINE-GRAPE`.

Modeling a red wine

- `Wine` is a concept.
- `Red` is a concept, denoting all red objects;
  thus, $\texttt{RED-WINE} \doteq \texttt{Wine} \sqcap \texttt{Red}$
- `red` is a specific color, instance of the
  concept `Color`;
  but `COLOR` is also a property of objects; thus,
  $\texttt{RED-WINE} \doteq \texttt{Wine} \sqcap \forall \texttt{COLOR.}\,(\{\,\texttt{red}\,\} \sqcap \texttt{Color})$
- `Red` is the concept of red color, possibly
  having several instances representing the
  different real red colors, ...

How to design a KB in 12 steps

1. **Enumerate Object**. As a bare list of elements of the KB; they will became individuals, concepts, or role.
2. **Distinguish Concepts from Roles**. Make a first decision about what object must be considered role; remember that some could have a "natural" concept associated. The remaining objects will be concepts (or maybe individuals). Also, try to distinguish roles from attributes.
3. **Develop Concept Taxonomy**. Try to decide a classification of all the concepts, imagining their extensions. This taxonomy will be used as a first reference, and could be revised when definition will be given. It will be used also to check if definition meet our expectations (sometime, interesting, unforeseen (re)classifications are found).

4. **Devise partitions**. Try to make explicit all the disjointness and covering constraints among classes, and reclassify the concepts.

5. **Individuals**. Try to list as many as possible *generally* useful individuals . Some could have been already listed in step 1. Try to describe them (classify).

6. **Properties and Parts**. Begin to define the internal structure of concepts (this process will continue in the next steps). For each concept list:

   - *intrinsic* properties, that are part of the very nature of the concept;
   - *extrinsic* properties, that are contingent or external properties of the object; they can sometime change during the time;
   - *parts*, in the case of structured or collective objects. They can be physical (e.g., "the components of a car", "the

casks of a winery", "the students of a class", "the members of a group", *"the grape of a wine"*) or abstract (e.g., *"the courses of a meal"*, "the lessons of a course", *"the topics of a lesson"*).

In some cases some relationships between individuals of classes can be considered too accidental to be listed above (e.g., "the employees of a winery"; but the matter could change if we consider `Winery` as a subconcept of `Firm`.

In general, the above distinctions depend on the level of detail adopted.

Some of the listed roles will be later considered definitional, and some incidental.

After this and the next steps check/revision of step 3 could be necessary.

7. **Cardinality Restrictions.** For the relevant roles for each concept.

8. **Value Restriction.** As above. Also, chose the right restriction.

9. **Propagate Value Restrictions.** If some value restrictions stated in the previous step does not correspond to already existing concepts, they must be defined.

10. **Inter-role Relationship.** Even if hardly definable in DL, they can be useful during the populating and debugging phases.

11. **Definitional and Incidental.** It is important distinguish between definitional and incidental properties, w.r.t. to the particular application.

12. **Primitive and Defined.** As above.

So what?

Why should we bother of using DL for querying databases, when there are much more expressive languages for that purpose – basically without the "three variables" limit?

- Reasoning over the query is decidable:
  - Query containment,
  - Query satisfiability.
- Evaluation still tractable.
- Two natural extensions:
  - Incomplete information,
  - Querying with a conceptual schema.

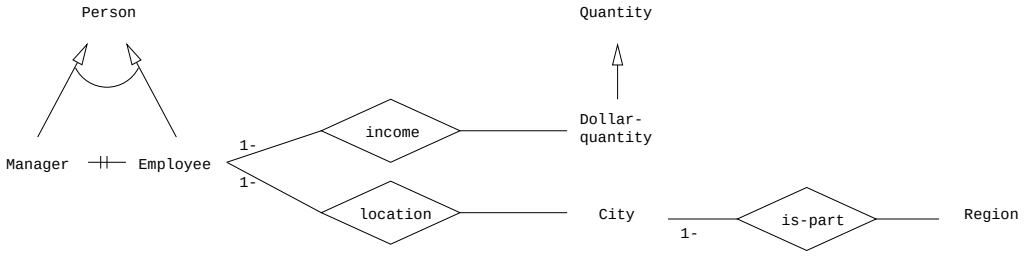
Incomplete Information

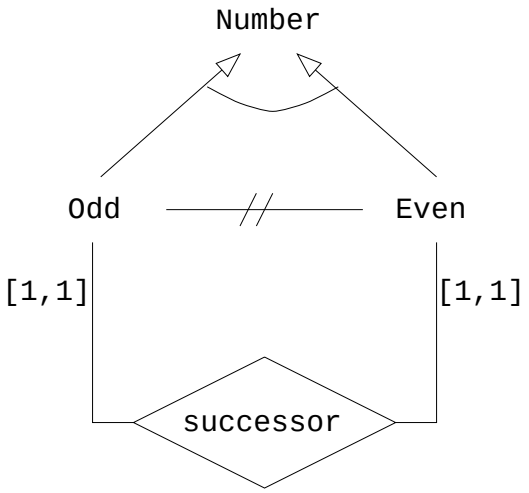
Handling incomplete knowledge is the ability to correctly reason without a complete specification of a situation, but with a under-specified description of a class of possible situations.

- FOL theories.
- KR theories.
  - Unique Name (UNA) assumption.
- Finite Domain theories.
  - Domain Closure (DO) assumption.
- Closed theories (e.g., null values).
  - DO + Completion (CO) assumptions.
- Extended Relational theories.
  - UNA + DO + CO assumptions.

The Entity-Relationship (ER) Conceptual Data Model

The Entity-Relationship (ER) model is the most common semantic data model for database design.

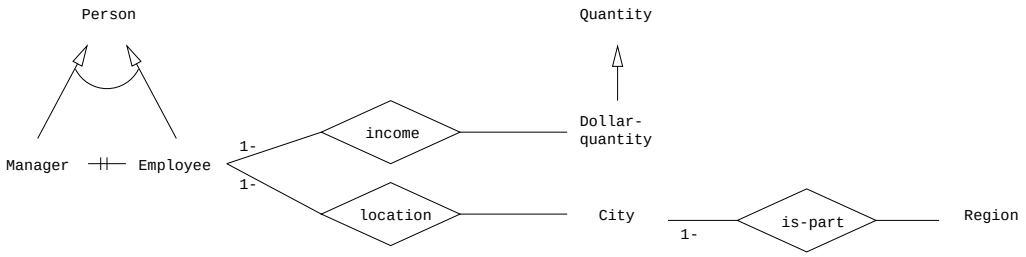




- An ER conceptual schema can be expressed in a suitable description logic theory.
- The models of the DL theory correspond with legal database states of the ER schemas.
- Reasoning services such as satisfiability of a schema or logical implication can be performed by the corresponding DL theory.
- A description logic allows for a greater expressivity than the original ER framework, in terms of full disjunction and negation, and entity definitions by means of both necessary and sufficient conditions.

Mapping an ER schema in a DL theory

- Relations are *reified* in the description logic theory, i.e., they become concepts with n special feature names denoting the n arguments of the n-ary relation.
- The relation `INCOME` becomes a concept with the two features:
  - `incomer` – relating to the first argument of the relation, i.e., an employee,
  - `incoming` – relating to the second argument of the relation, i.e., a dollar quantity.
- `incomer`, `incoming`, `locator`, `place`, `whole`, and `part` are functional roles.



$$\mathtt{INCOME} \sqsubseteq \mathtt{incomer} : \mathtt{Employee} \sqcap \mathtt{incoming} : \mathtt{Dollar\text{-}quantity}$$

$$\mathtt{LOCATION} \sqsubseteq \mathtt{locator} : \mathtt{Employee} \sqcap \mathtt{place} : \mathtt{City}$$

$$\mathtt{IS\text{-}PART} \sqsubseteq \mathtt{part} : \mathtt{City} \sqcap \mathtt{whole} : \mathtt{Region}$$

$$\mathtt{Employee} \sqsubseteq \mathtt{Person} \sqcap \exists \mathtt{incomer}^{-1}.\mathtt{INCOME} \sqcap \exists \mathtt{locator}^{-1}.\mathtt{LOCATION}$$

$$\mathtt{Manager} \sqsubseteq \mathtt{Person} \sqcap \neg \mathtt{Employee}$$

$$\mathtt{Person} \sqsubseteq \mathtt{Manager} \sqcup \mathtt{Employee}$$

$$\mathtt{Dollar\text{-}quantity} \sqsubseteq \mathtt{Quantity}$$

$$\mathtt{City} \sqsubseteq \exists \mathtt{part}^{-1}.\mathtt{IS\text{-}PART}$$

Object-Oriented Conceptual Data Models

- Is is intuitive to understand the relationship between a description logic and a generic Object-Oriented formalism.
- Unlike object-oriented systems, description logics do not stress the representation of the behavioral aspect of information, for which they are still considered inadequate.
- The translation of the structural part of an O-O schema into a description logic knowledge base is similar to the one sketched for ER schemas.

Advantages of DL for Conceptual Modeling

- *Ontological organization.* It is possible to capture important basic facets of data semantics, including the structure of complex entities.
- *Consistency checking.* It is possible to check whether the global information conveyed in a schema forces some specific class to be inconsistent. Moreover, one could check the consistency of the whole schema, also with respect to possible integrity constraints.

- *Data entry.* The user is supported in the phase of populating the data base, according to the knowledge of the schema and satisfying the integrity constraints. Then, the system could not only check the consistency of the data base itself, but also make some deductive inferences asserting new facts regarding the data. Moreover, the system supports the user in the refinement of the schema in a populated data base.

- *Views organization.* Views - i.e., pre-defined descriptions, grounded on the terms of the schema - are automatically organized into a hierarchy, which is a non-trivial task when there are many complex views. Taxonomic relations between views do explicit their meaning and their specificity, allowing for its retrieval and reuse.

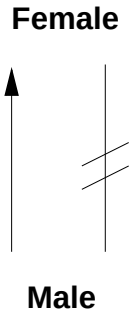
- *Schema refinement.* It is possible to reduce the redundancy of a schema, by discovering equivalent descriptions, by reusing descriptions, and by exploiting the description lattice.

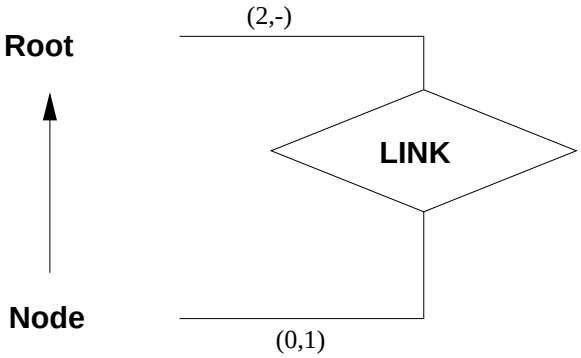
- *Inter-schema organization.* It is possible to define inter-schema knowledge describing the constraints among different databases, easing the task of managing multi-databases.

Finite Model Reasoning

- *Simple* description logics do have the Finite Model property: if a formula is satisfiable, then it is finitely satisfiable.
- However, very expressive description logics do not have the finite model property anymore.
- Satisfiability (logical implication) and finite satisfiability (logical implication) may diverge. The theory may infer a property holding only in the finite structures, but classical reasoning may not reveal this fact.
- Finite model reasoning is relevant for database conceptual modeling: databases are always finite.
- In order to model ER schemas, we need a logic which does not have the finite model property.

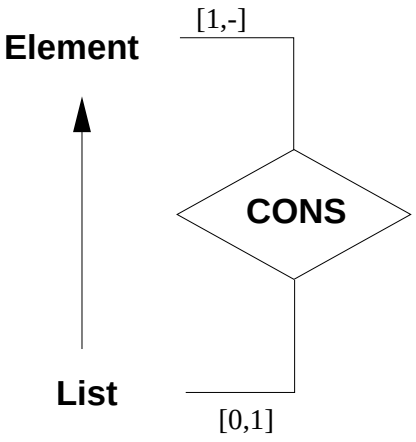
Examples





Unsatisfiable

Satisfiable, but not finitely satisfiable



Not logically implied, but finitely logically implied

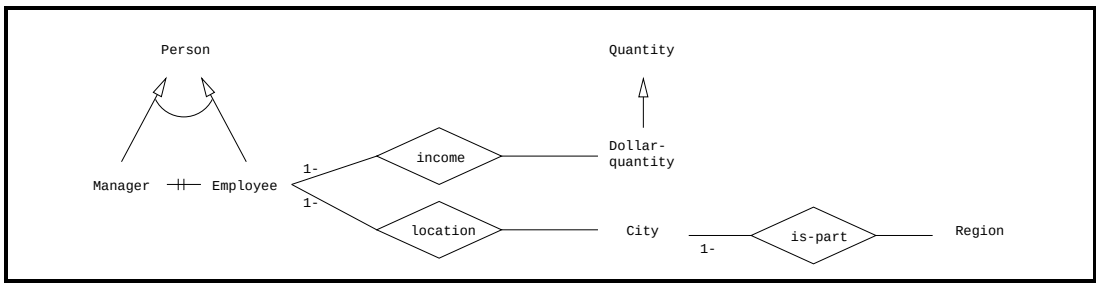
Querying with a conceptual schema

- Query containment of conjunctive queries (SPJ-queries) referring to predicates defined in a $\mathcal{ALCQI}_{reg}$ theory is EXPTIME-complete.
- It is possible to encode constraints over n-ary relations.
- The DL schema allows for recursive definitions, with full negation, disjunction, and universal quantification (compare with DATALOG).
- DL are able to fully encode many O-O and semantic data models, e.g., Entity-Relationship (ER), Object-Role Modeling (ORM), OMT static model. It can extend them in may ways, e.g., with negation, disjointness, covering constraints.

Querying with a conceptual schema

$$\mathtt{Q}(\mathtt{x},\mathtt{y},\mathtt{z}) \doteq \mathtt{Person}(\mathtt{x}) \wedge \neg\mathtt{Manager}(\mathtt{x}) \wedge \mathtt{INCOME}(\mathtt{x},\mathtt{y}) \wedge \mathtt{LOCATION}(\mathtt{x},\mathtt{z}).$$

$$\Downarrow$$



$$\Uparrow$$

$$\mathtt{Table}^{1}_{\mathtt{DB}_1}(\mathtt{x},\mathtt{y}) \doteq \mathtt{Employee}(\mathtt{x}) \wedge \mathtt{INCOME}(\mathtt{x},\mathtt{y}).$$

$$\mathtt{Table}^{2}_{\mathtt{DB}_1}(\mathtt{x},\mathtt{y}) \doteq \mathtt{Employee}(\mathtt{x}) \wedge \mathtt{LOCATION}(\mathtt{x},\mathtt{y}).$$

Advantages of DL

- *Query validation.* Incoherent queries - i.e., queries that can not return any value as answer, given their inconsistent meaning with respect to the schema - are detected, and the user is informed about its ill-formed request.

- *Query generalization.* In many situations, the query, even if it is consistent, can return an empty answer, since there is no actual object in the database satisfying it. In such cases, it is reasonable to generalize the query until a non empty answer is obtained; the description lattice is the obvious space where such generalizations can be searched for.

- *Query organization.* Data exploration may involve a great amount of queries, possibly submitted by different group of persons, in different periods of time, for different purposes. The system can organize the set of queries in a hierarchy, such that it is possible to retrieve already submitted similar or equivalent queries, together with the cached results. This is relevant if the queries need a substantial amount of time to be processed, or if the users associate comments or observations to the queries or to the answers.

- *Query refinement.* Queries can be specified through an iterative refinement process supported by the description lattice for the queries. This process is useful for data exploration tasks. The user may specify his/her request using generic terms; after the query classification, which makes explicit the meaning and the specificity of the query itself and of the terms composing the query, the user may refine some terms of the query or introduce new terms, and iterate the process.

- *Intensional query processing.* Users may explore and discover new generic facts without querying the whole information base, but by giving an explicit meaning to the queries through classification. The system has the ability of answering a query with synthetic concepts representing the general characteristics of the information that satisfy it, as opposed to answering with long sequences of detailed data. Moreover, if the query is classified in a taxonomy of descriptions and queries already computed and indexing the answers, then it can be processed with respect to the indexed objects only, rather than with respect to the whole information base.

- *Query optimization.* Given a schema and a set of views and already processed queries, a query can be optimized by computing an equivalent more efficient one. The optimized query can be obtained by using the cached results - maintained by a semantic indexing technique - retrieved by classification, and/or making more specific the single terms and the complex descriptions used within the query original formulation.