Knowledge Engineering with Semantic Web Technologies

Lecture 4: Knowledge Representation

01: Ontology One on One

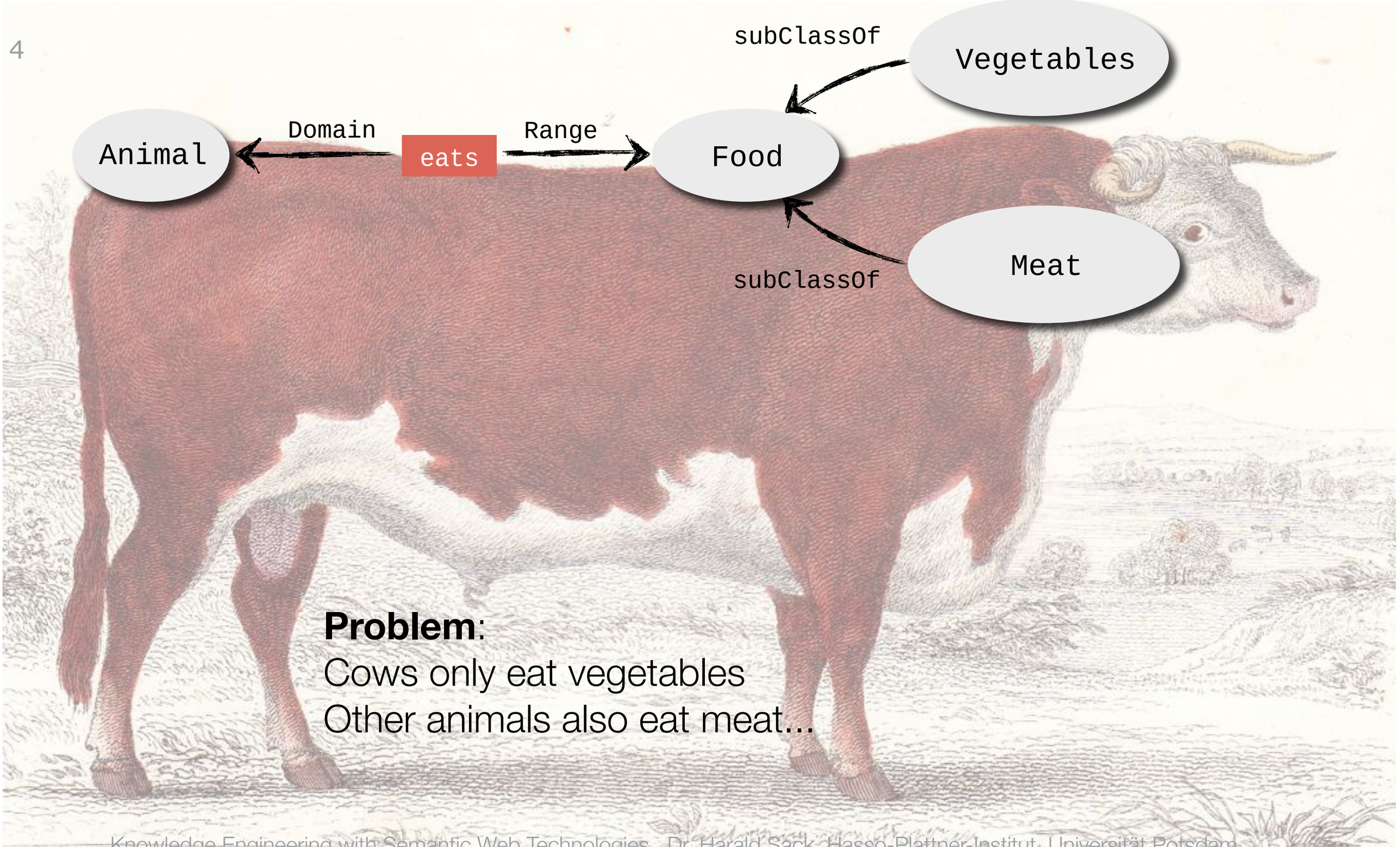
Dr. Harald Sack
Hasso-Plattner-Institut for IT Systems Engineering
University of Potsdam
Spring 2014

Lecture 4: Knowledge Representations I

Open HPI Course: Knowledge Engineering with Semantic Web Technologies

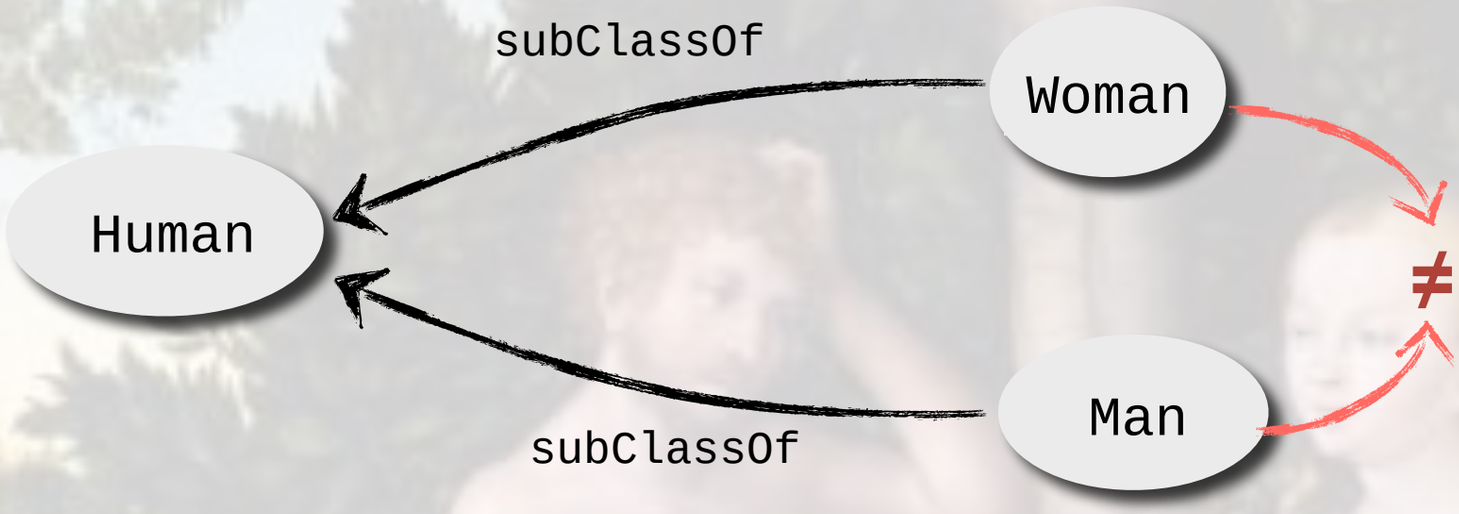
Is RDF(S) sufficient?

Just one example...



Problem:
Cows only eat vegetables
Other animals also eat meat...

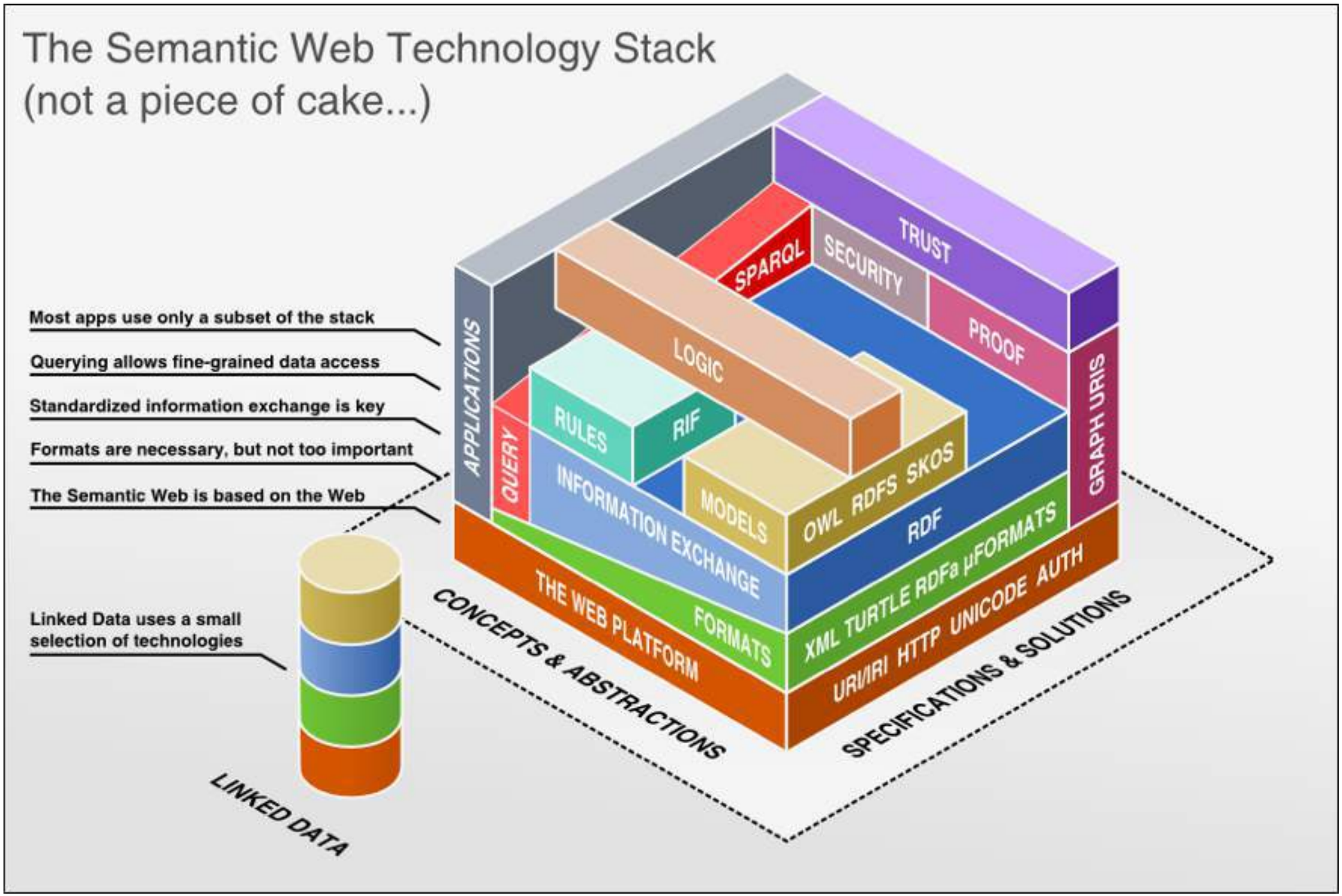
Just another example...



Problem:
Subclass relation cannot express disjunctive class (subclass) membership

4. Knowledge Representations

Ontology One on One



01 Ontology One on One

Open HPI Course: Knowledge Engineering with Semantic Web Technologies
Lecture 04: Knowledge Representations Pt.2

„People can't share knowledge if they don't speak a common language“

Thomas Davenport (1997)

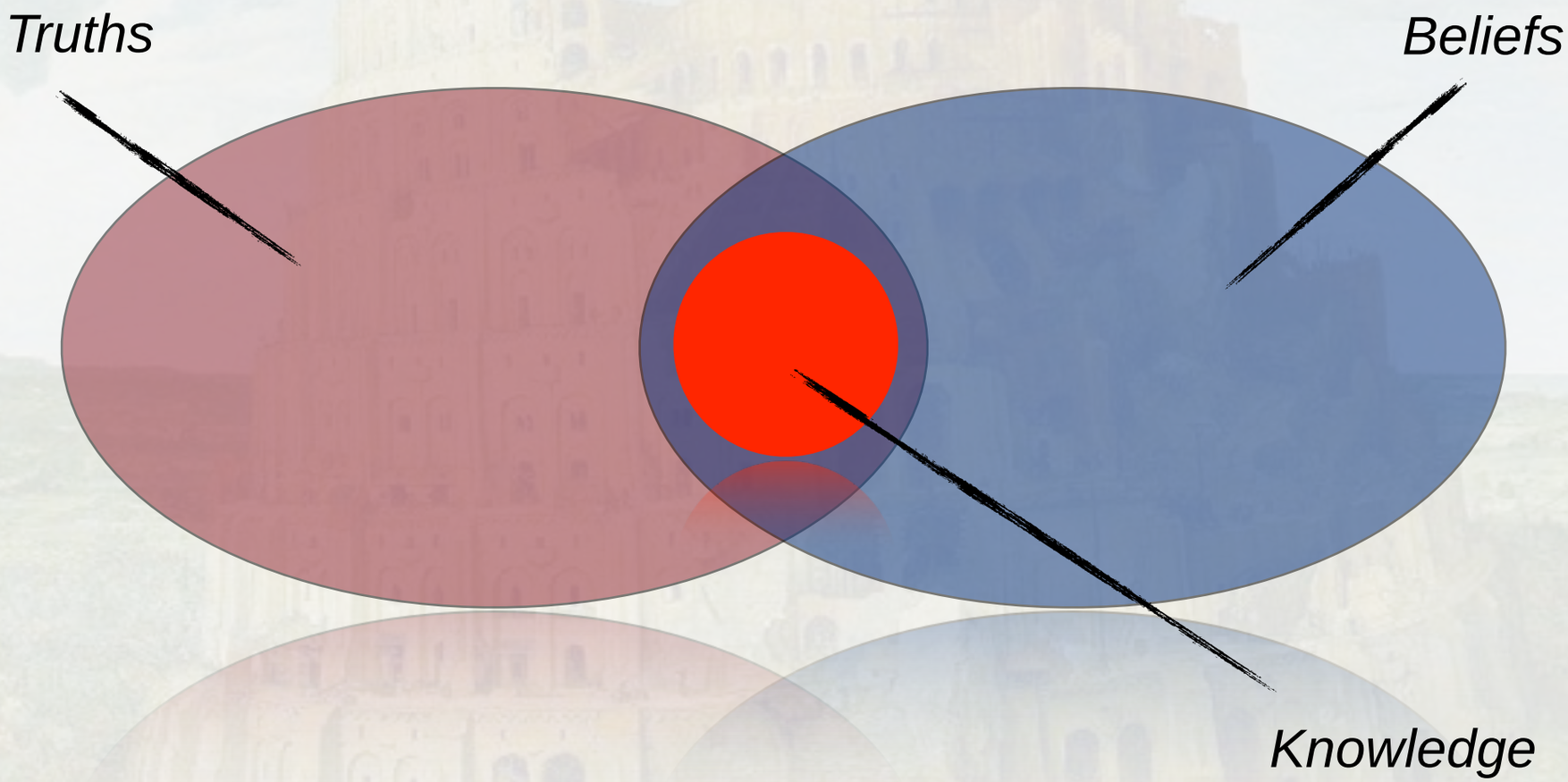
Turmbau zu Babel, Pieter Brueghel, 1563

To speak a common language...

- common symbols and concepts (**Syntax**)
- agreement about their meaning (**Semantics**)
- classification of concepts (**Taxonomy**)
- associations and relations of concepts (**Thesauri**)
- rules and knowledge about which relations are allowed and make sense (**Ontologies**)

Turmbau zu Babel, Pieter Brueghel, 1563

What is Knowledge?



Traditional Definition: „Knowledge is a subset of all true beliefs“

Turmbau zu Babel, Pieter Brueghel, 1563

To represent knowledge, we need a formal knowledge representation:

⇒ Ontologies

Turmbau zu Babel, Pieter Brueghel, 1563

Ontology is the
philosophical study of the nature of being, existence, or reality, as well as the basic categories of being and their relations...

Fundamental Questions of Ontology

1. What does it mean for a being to be?

- When are two things identical?
- Is everything that exists also real?
- Does something exist, if it is only possible?
- Are there non-existing things?

2. What categories of objects do exist?

- Do things exist that are only unique or only multiple (*Universalia*)?
- Do things exist that are unilaterally dependent of others (*Substances*)?
- Of which sort is this dependency (*Causality*)?
- Do necessary properties exist (*Essences*)?
- How do composed things relate to their components?

The Concept of Ontology in Computer Science

"An ontology is an explicit, formal specification of a shared conceptualization. The term is borrowed from philosophy, where an Ontology is a systematic account of Existence. For AI systems, what 'exists' is that which can be represented."

Thomas R. Gruber: A Translation Approach to Portable Ontology Specifications.Knowledge Acquisition, 5(2):199-220, 1993.

conceptualization:	abstract model (domain, identified relevant concepts, relations)
explicit:	meaning of all concepts must be defined
formal:	machine understandable
shared:	consensus about ontology

How should we represent Ontologies?

Ontologies - Components and Models

- Classes, Relations and Instances

  - **Classes** represent concepts
  - Classes are described via attributes
  - Attributes are Name Value pairs

„The address contains the name, title, and place of residence of the person addressed"

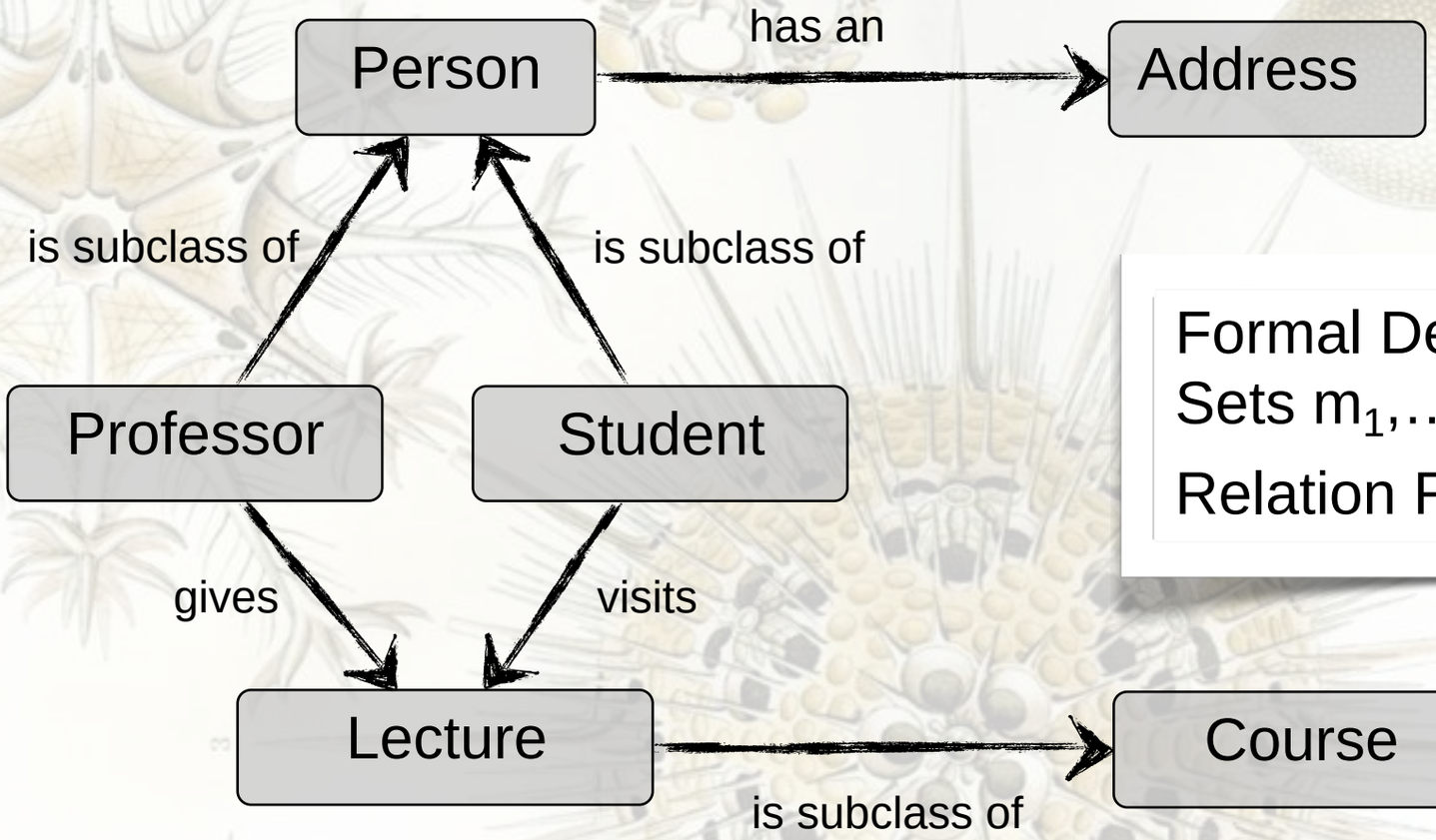
Informal Description

Address
- given name *<string>*
- family name *<string>*
- street *<string>*
- ZIP code *<int>*
- city *<string>*
- ...

Semi informal Description

Ontologies - Components and Models

- **Classes** are related to other classes

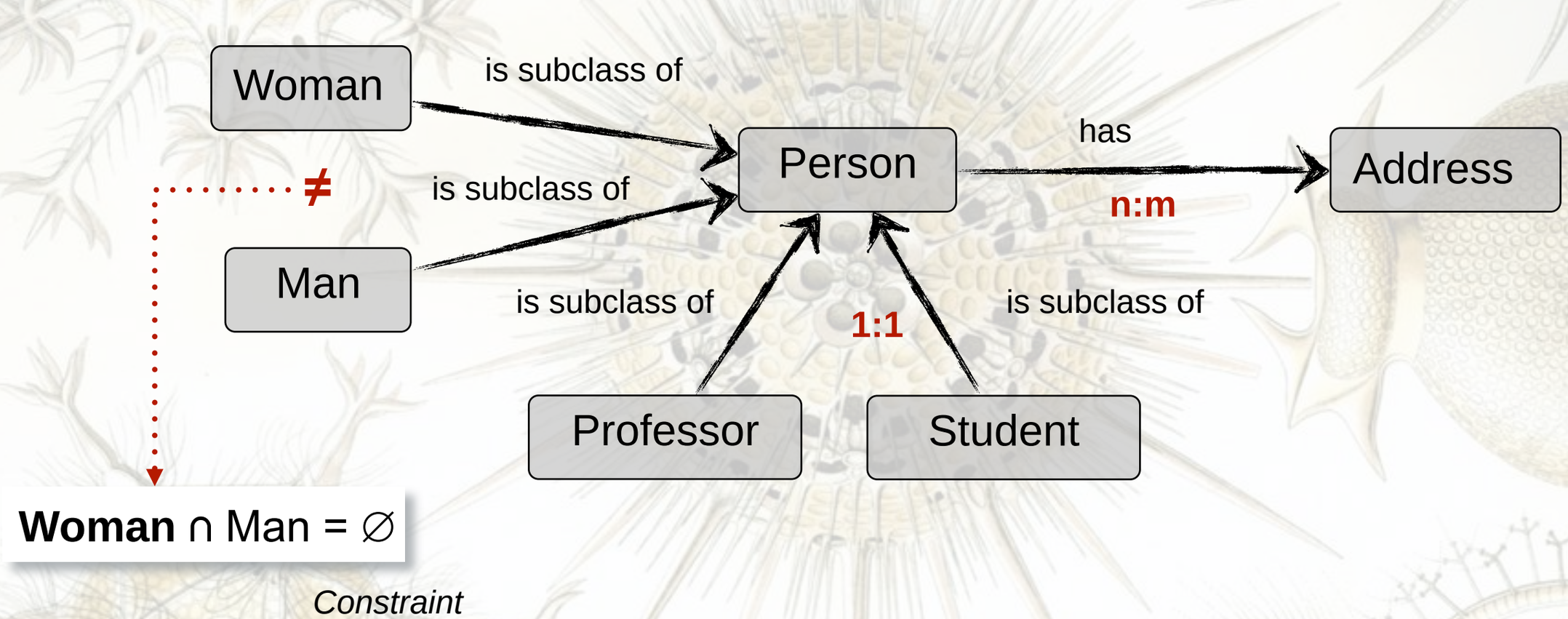


Formal Definition:
Sets $m_1,\ldots m_n$
Relation $R \subseteq m_1 \times \ldots \times m_n$

- **Relations** are special attributes, whose values are objects of (other) classes

Ontologies - Components and Models

- For Relations and Attributs **Constraints** (Rules) can be defined that determine allowed values



Ontologies - Components and Models

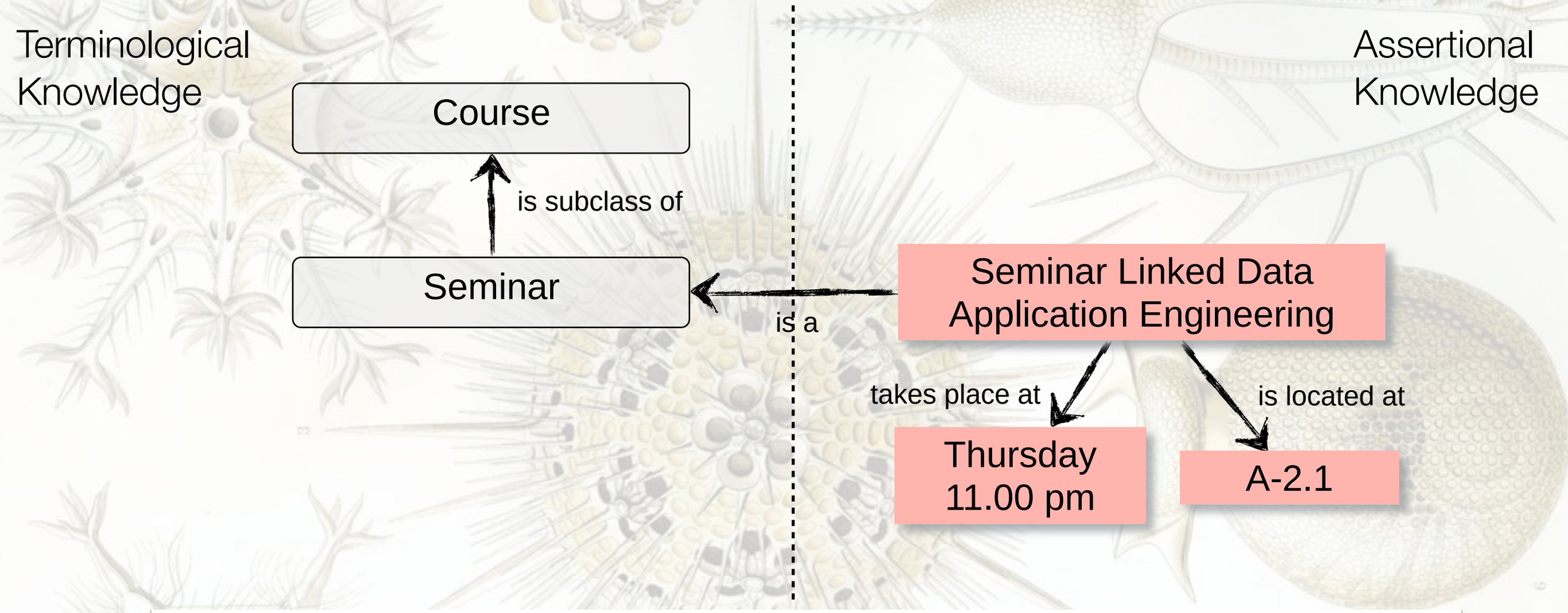
- Classes, relations, and constraints can be put together to form **statements / assertions**
- Special Case: **formal Axioms**

Example:
It is not possible to lecture two courses at the same time.

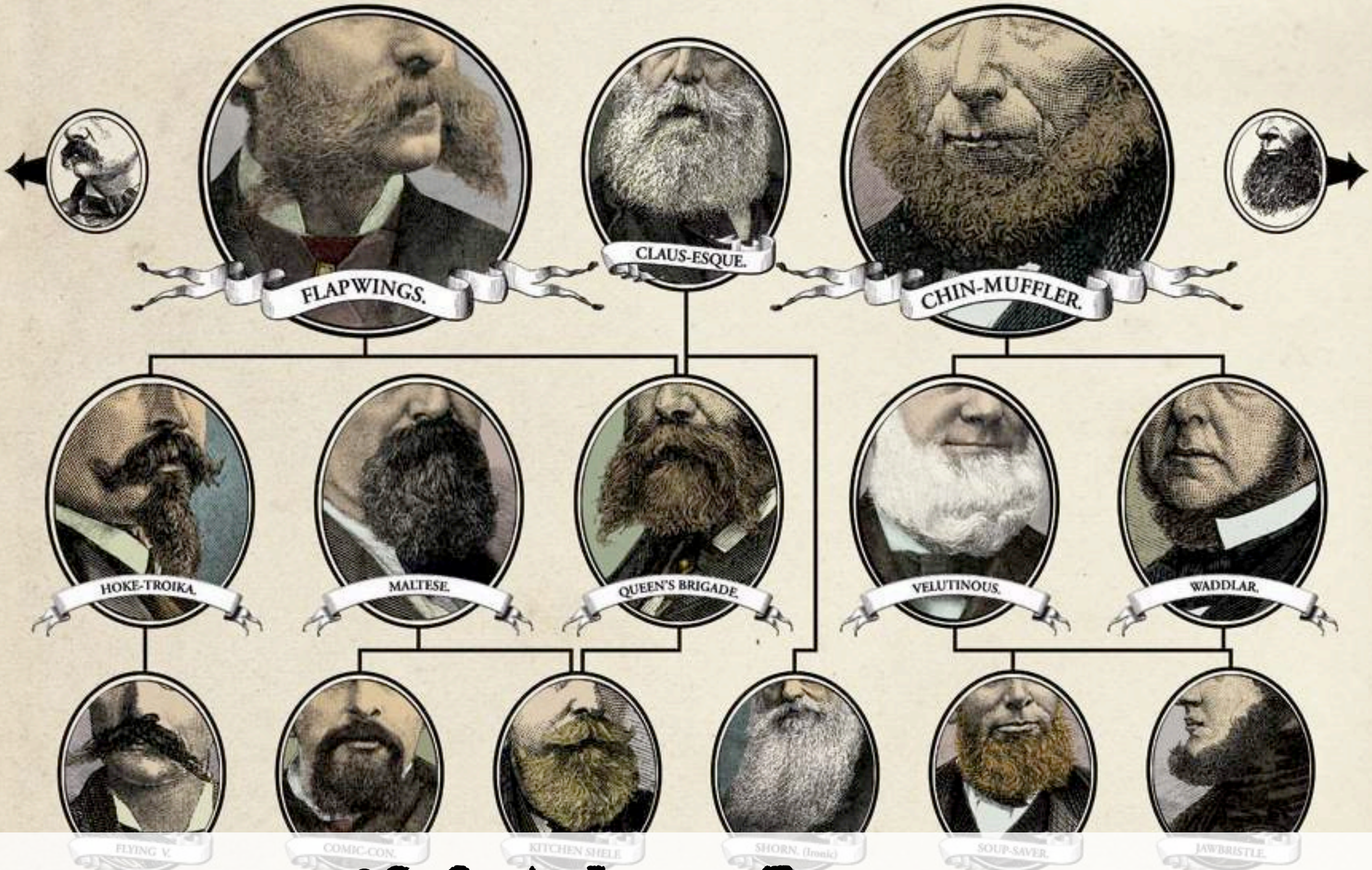
- **Axioms** describe knowledge that can‘t be expressed simply with the help of other existing components.

Ontologies - Components and Models

- **Instances** describe individuals of an ontology



Individuals *(instances) are the basic components of an ontology. The individuals in an ontology may include concrete objects such as people, animals, tables, automobiles, molecules, and planets, as well as abstract individuals such as numbers and words.*



02 Ontology Types

Open HPI Course: Knowledge Engineering with Semantic Web Technologies
Lecture 04: Knowledge Representations Pt.2

Knowledge Engineering with Semantic Web Technologies

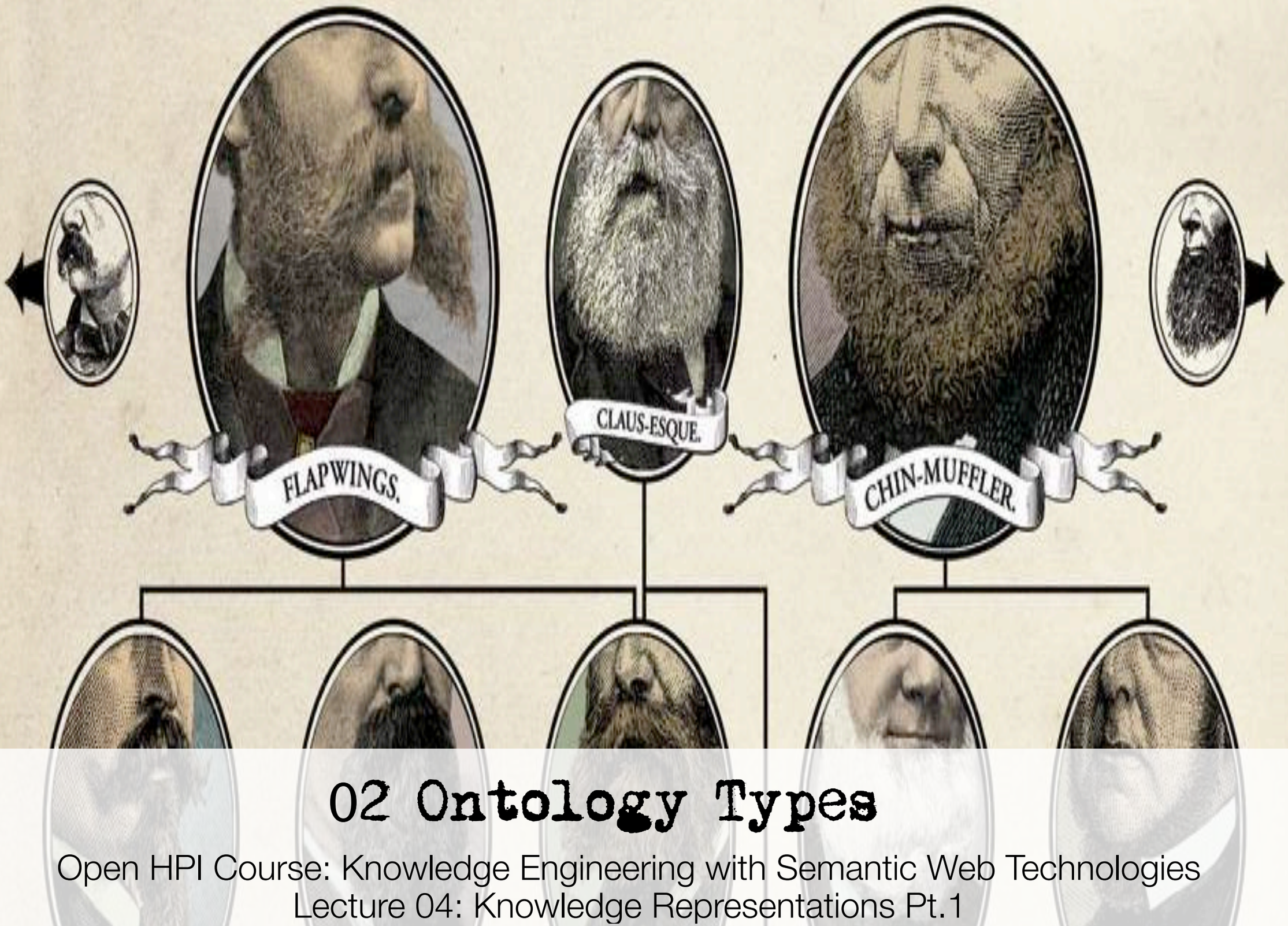
Lecture 4: Knowledge Representation

02: Ontology Types

Dr. Harald Sack
Hasso-Plattner-Institut for IT Systems Engineering
University of Potsdam
Spring 2014

Lecture 4: Knowledge Representations I

Open HPI Course: Knowledge Engineering with Semantic Web Technologies



02 Ontology Types

Open HPI Course: Knowledge Engineering with Semantic Web Technologies
Lecture 04: Knowledge Representations Pt.1

Ontology Types and Categories

general, cross domain ontologies
(represent very general concepts as e.g., Time, Space, Event independent of a specific domain or problem.)

Top-Level Ontology (Upper Ontology, Foundation Ontology)

Domain Ontology ⟷ **Task Ontology**

fundamental concepts according to a generic domain.

fundamental concepts according to a general activity or task.

Application Ontology

specialized ontology focussed on a specific task and domain

(according to Guarino,1998)

Ontology Types and Categories

general, cross domain ontologies
*(represent **very general concepts***
as e.g., Time, Space, Event
***independent of a specific domain or problem**.)*

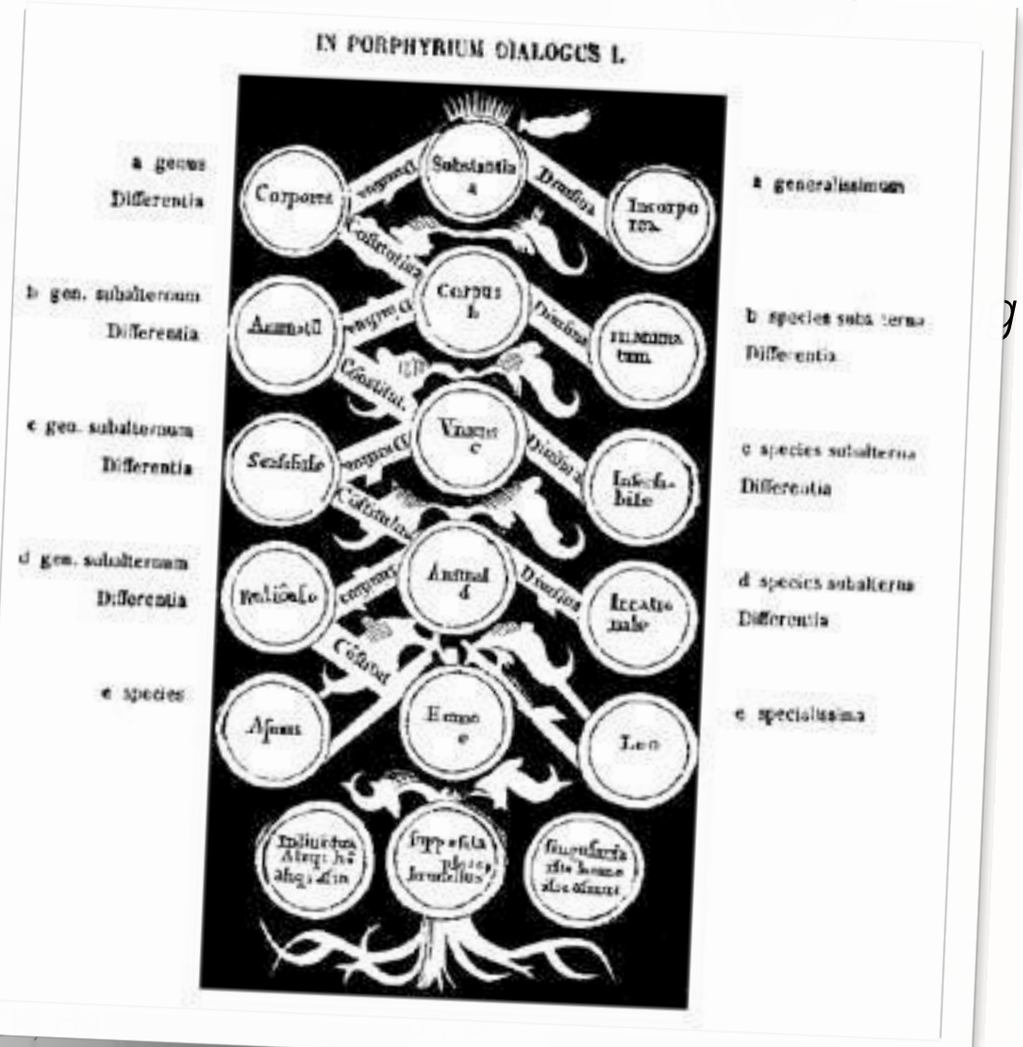
Top-Level Ontology (Upper Ontology, Foundation Ontology)

Domain Ontology ⟷ ***Ta***

fundamental concepts according to a generic domain.

Application Ontology

specialized or
specific task a



(according to Guarino,1998)

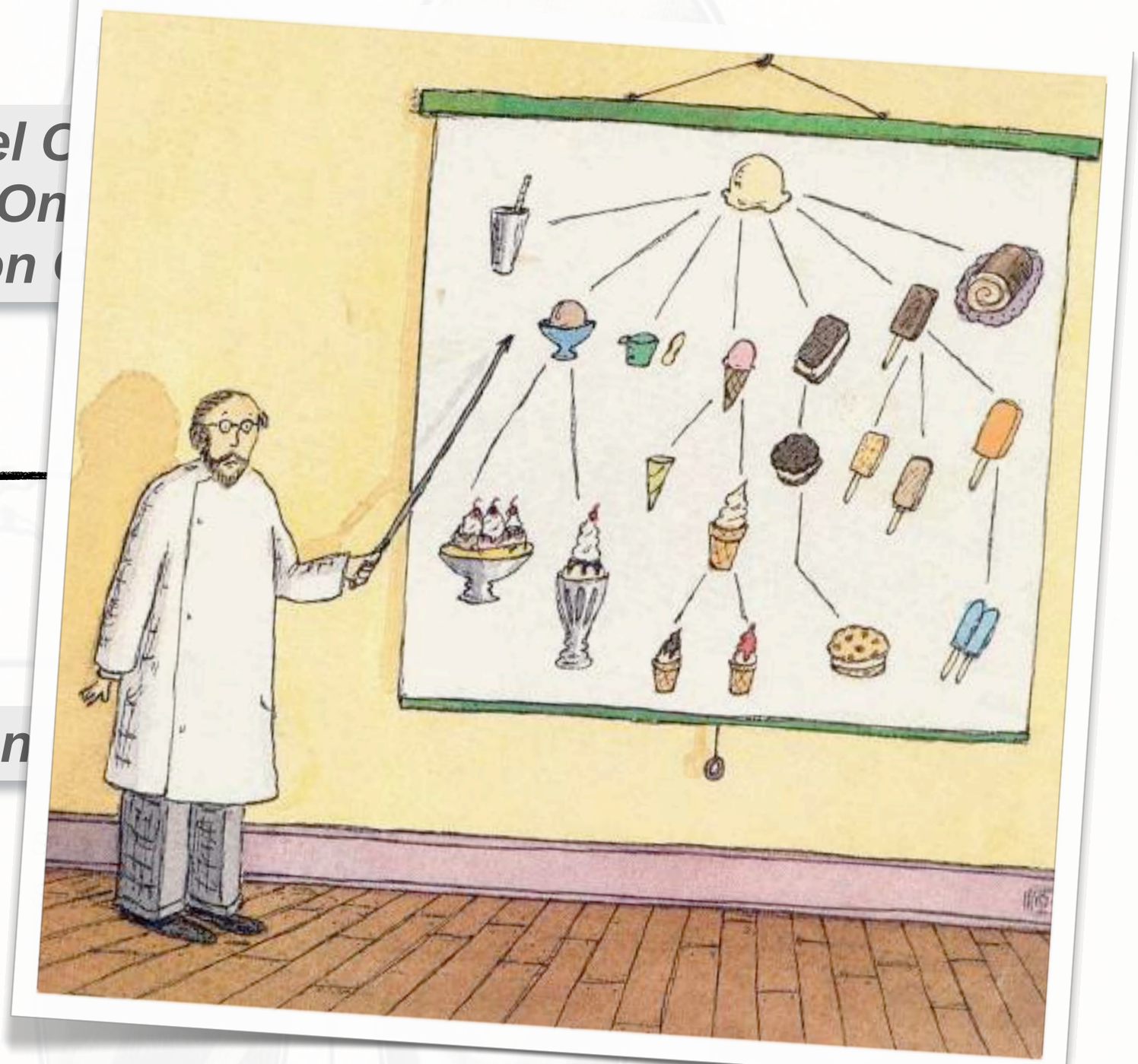
Ontology Types and Categories

Top-Level C
(Upper On
Foundation (

Domain Ontology

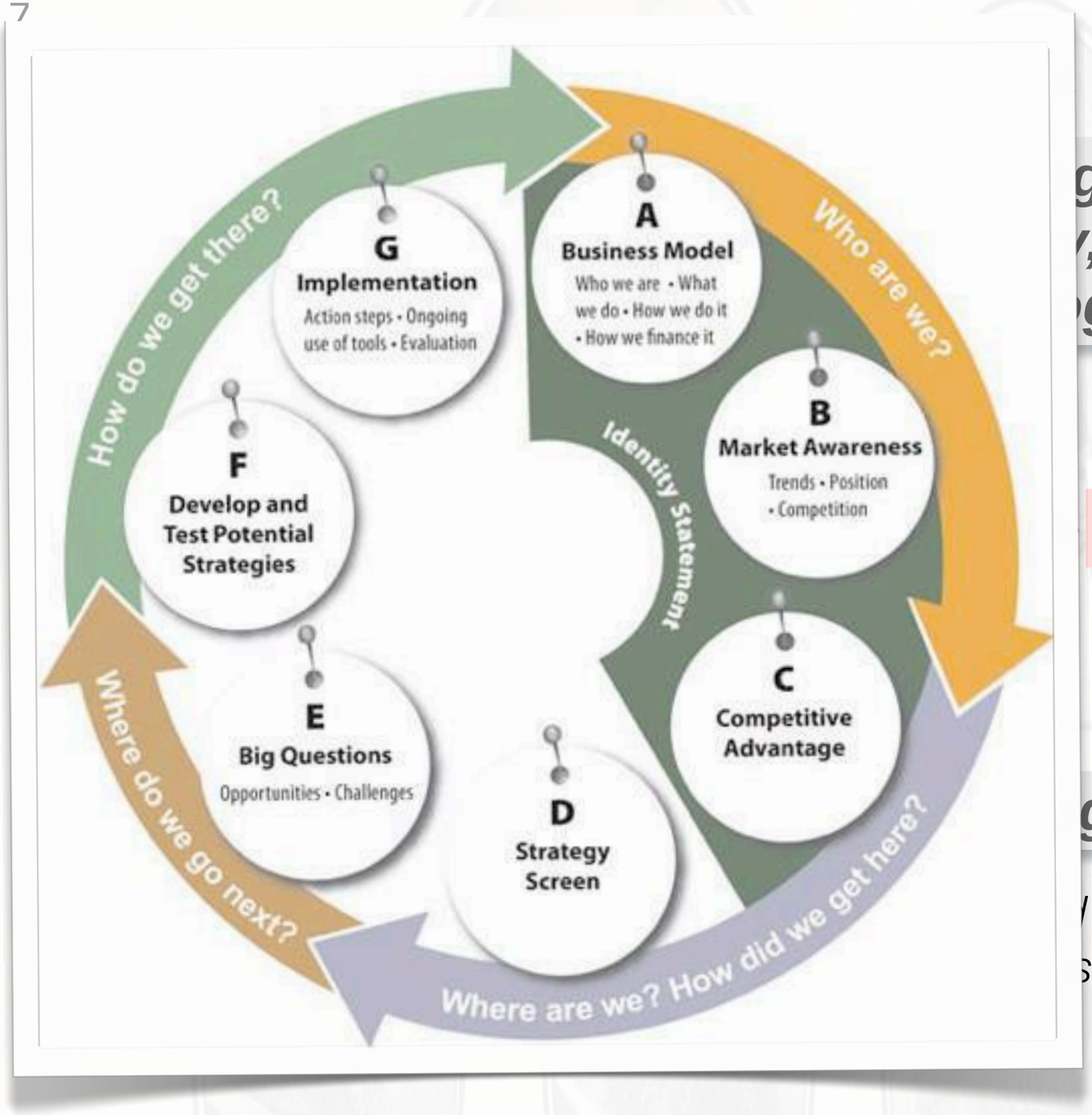
*fundamental concepts according to a generic **domain**.*

Application



(according to Guarino,1998)

Ontology Types and Categories



cross domain ontologies
(represent very general concepts as e.g., Time, Space, Event independent of a specific domain or problem.)

gy ,
gy)

Task Ontology

*fundamental concepts according to a **general activity or task**.*

gy

l ontology focussed on a sk and domain

(according to Guarino,1998)

Ontology Types and Categ

ge

neral concepts
ce, Event
pecific domain

Top-Level Onto
(Upper Ontolog
Foundation Onto

Domain Ontology

fundamental concepts
according to a
generic domain.

Application Ontology

specialized ontology focussed on a specific task and domain

(according to Guarino,1998)

Ontology Types and Categories

informal — Expressivity — formal

- +

controlled Vocabulary — ***Thesauri*** — ***formal IS-A*** — ***Frames*** — ***general logical Constraints***

Glossaries — ***informal IS-A*** — ***formal Instance*** — ***Value Restrictions*** — ***Disjunctness, Inversiveness, Part-of…***

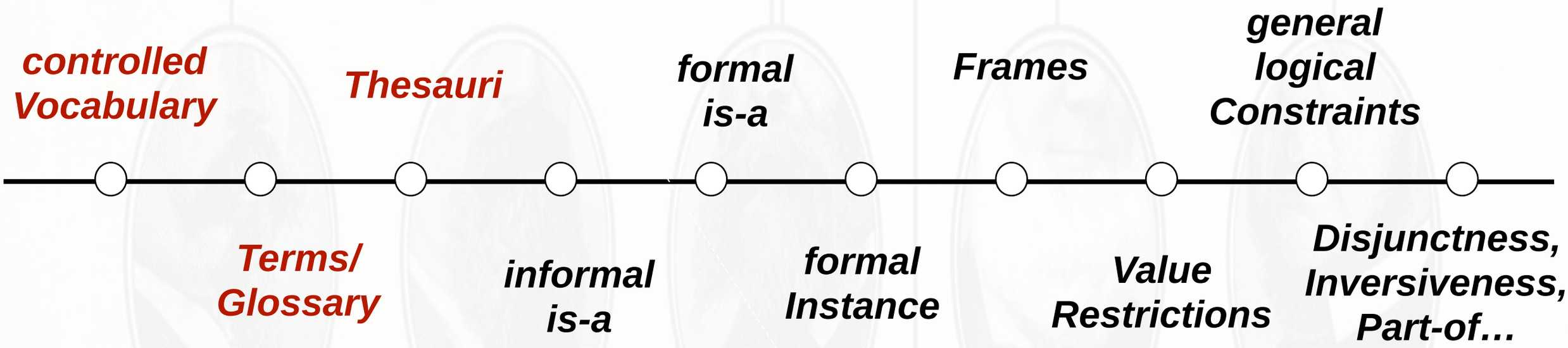
Terms — Data Dictionaries — Folksonomies — formal Taxonomies — Logic Programming — Description Logics — First Order Logics

lightweight ontologies — heavyweight ontologies

(according to Lassila/McGuinnes, 2001)

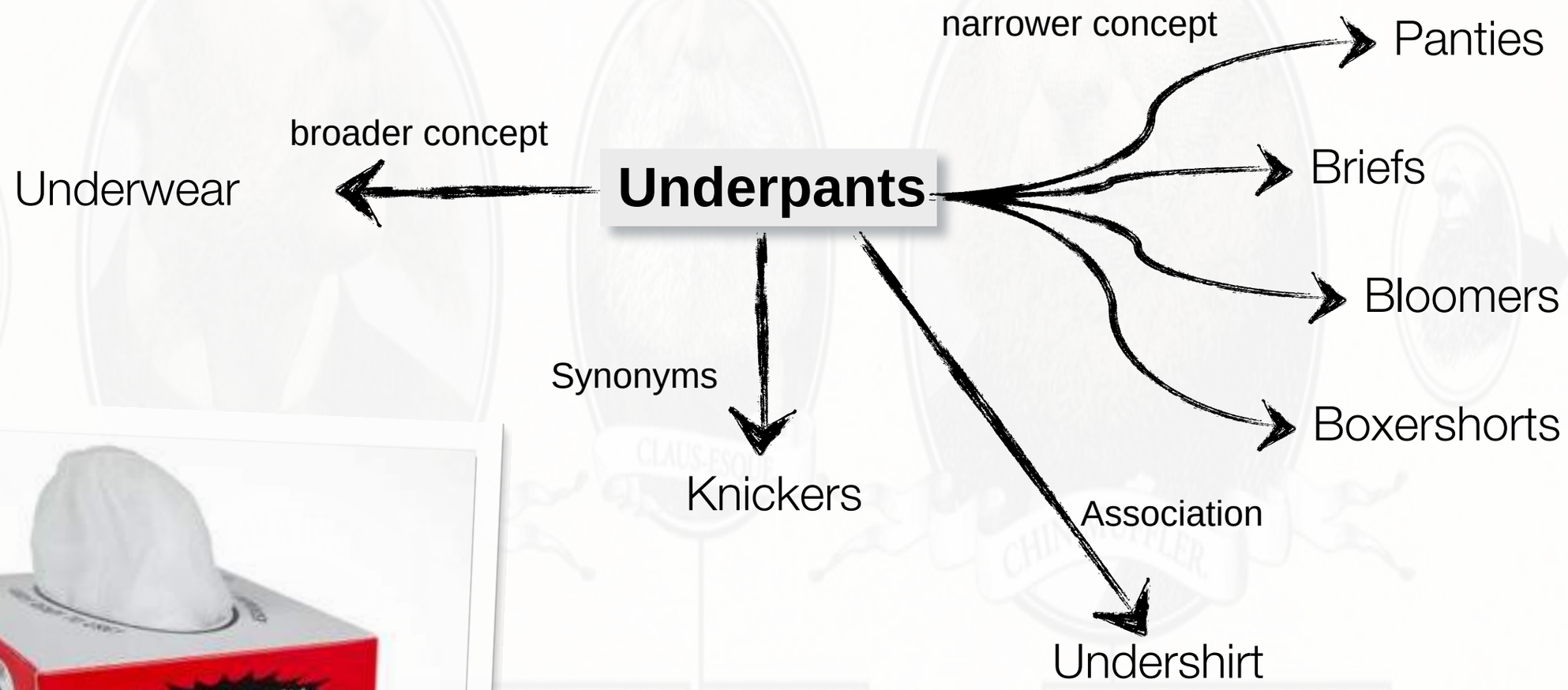
Ontology Types and Categories

- **Controlled Vocabulary:**
  finite list of terms (e.g. catalogue)
- **Glossary:**
  finite list of terms including an informal definition in natural language
- **Thesauri**: [greek. „treasure, treasure house“]
  controlled vocabulary, concepts are connected via relations.
  - Equivalency (synonyms)
  - Hierarchies (subclasses, superclasses)
  - Homographs (Homonyms)
  - Associations (similar concepts)



Example of a Thesaurus Entry...

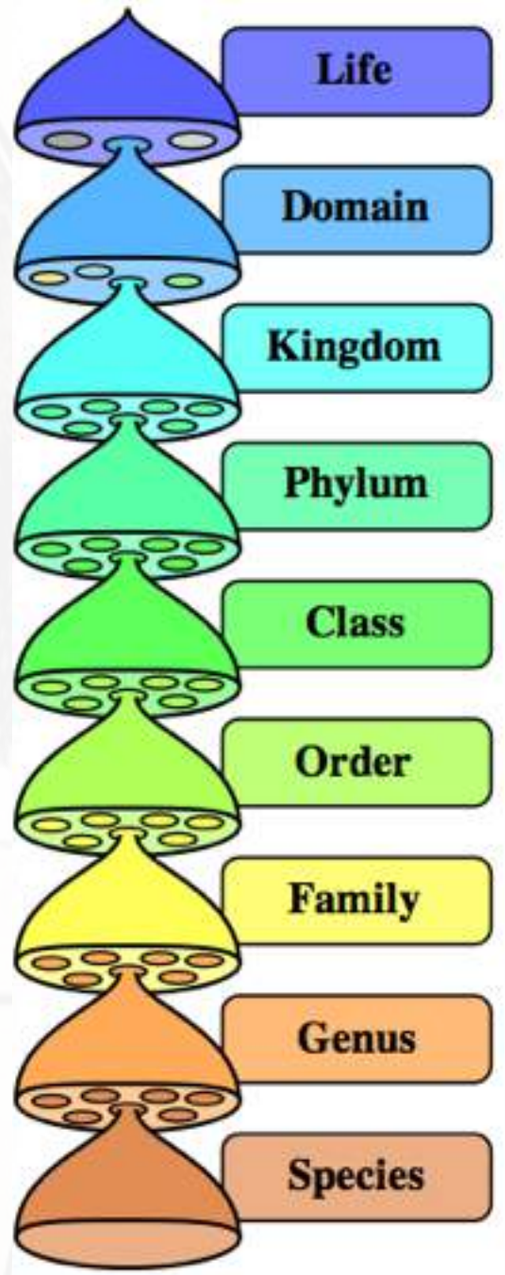
http://frat.manmosh.com/category/home/



Taxonomies

Taxonomy: Definition of a hierarchical system of groups (from [greek] *τάξις* (*taxis)* = order, arrangement and *νόμος* (*nomos)* = law, science) ...

- also **classification schema**, nomenclature, …
- in science most times classification into (mono-)hierarchical sets (classes, subclasses, ...)
- (also) subject of **biology**:
  - the arrangement of organisms into a classification according to similarities

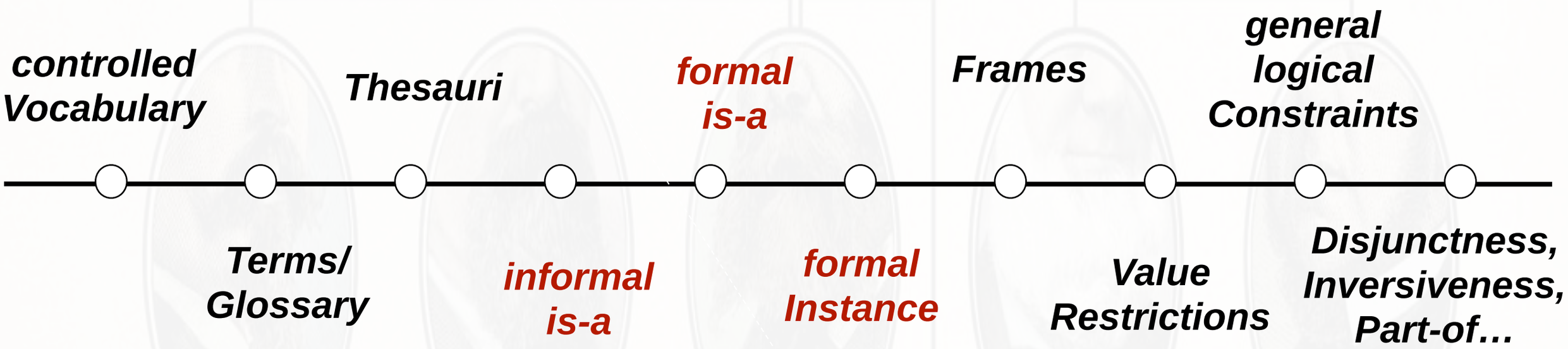


Taxonomies

- **informal IS-A-Hierarchy:**
  explicite hierarchy of classes, subclass relations are not strict
  (e.g. index of a library)

- **formal IS-A-Hierarchy:**
  explicite hierarchy of classes, subclass relations are strict

- **formale instance:**
  explicite class hierarchy, besides strict subclass relations also instance-of relations are allowed




But, beware....

Ontologies as Interpretation of Reality

15

various categories of animals from "a certain chinese encyclopedia" according to Jorge Luis Borges:

- THOSE THAT BELONG TO THE EMPEROR
- EMBALMED ONES
- THOSE THAT ARE TRAINED
- SUCKLING PIGS
- MERMAIDS (OR SIRENS)
- FABULOUS ONES
- STRAY DOGS
- THOSE THAT ARE INCLUDED IN THIS CLASSIFICATION
- THOSE THAT TREMBLE AS IF THEY WERE MAD
- INNUMERABLE ONES
- THOSE DRAWN WITH A VERY FINE CAMEL HAIR BRUSH
- ET CETERA
- THOSE THAT HAVE JUST BROKEN THE FLOWER VASE
- THOSE THAT, AT A DISTANCE, RESEMBLE FLIES

Jorge Luis Borges: The Analytical Language of John Wilkins (1942)

03 Foundations of Logic

Open HPI Course: Knowledge Engineering with Semantic Web Technologies
Lecture 04: Knowledge Representations Pt.1

Knowledge Engineering with Semantic Web Technologies

Lecture 4: Knowledge Representation

03: The Foundations of Logic

Dr. Harald Sack
Hasso-Plattner-Institut for IT Systems Engineering
University of Potsdam
Spring 2014

Lecture 4: Knowledge Representations I

Open HPI Course: Knowledge Engineering with Semantic Web Technologies

03 Foundations of Logic

Open HPI Course: Knowledge Engineering with Semantic Web Technologies
Lecture 04: Knowledge Representations Pt.1

Foundations of Logic

- Definition (for our lecture):
  Logic is the study of how to make formal correct deductions and inferences.

Turmbau zu Babel, Pieter Brueghel, 1563

„*The only way to rectify our reasonings is to make them as tangible as those of the Mathematicians, so that we can find our error at a glance, and when there are disputes among persons, we can simply say:* ***Let us calculate,*** *without further ado, to see who is right.*“

Gottfried Wilhelm Leibniz
(1646-1716)

Leibniz in a letter to Ph. J. Spener, Juli 1687

Foundations of Logic

- **Syntax**: *symbols without meaning*
  defines rules, how to construct well-formed and valid sequences of symbols (strings)
- **Semantic**: *meaning of syntax*
  defines rules how the meaning of complex sequences of symbols can be derived from atomic sequences of symbols.

Syntax

```
If (i<0) then display (“negative account!“)
```

assignment of meaning

print the message “negative account!“, if the account balance is negative

Variants of Semantics

- E.g. programming languages

```
FUNCTION f(n:natural):natural;
BEGIN
        IF n=0 THEN f:=1
        ELSE f:=n*f(n-1);
END;
```

Syntax

computation of the factorial

intentional semantics

- „*the meaning intended by the user*“
- restricts the set of all possible models (meanings) to the meaning intended by the (human) user

Variants of Semantics

- E.g. programming languages

```
FUNCTION f(n:natural):natural;
BEGIN
        IF n=0 THEN f:=1
        ELSE f:=n*f(n-1);
END;
```

Syntax

computation of the factorial

intentional semantics

$f : n \rightarrow n!$

formal semantics

- aims to express the meaning of symbol sequences (programs) in a **formal language**, in a way that assertions over the symbol sequences (programs) can be proven by the application of deduction rules.

Variants of Semantics

- E.g. programming languages

```
FUNCTION f(n:natural):natural;
BEGIN
        IF n=0 THEN f:=1
        ELSE f:=n*f(n-1);
END;
```

Syntax

computation of the factorial

intentional semantics

$f : n \rightarrow n!$

formal semantics

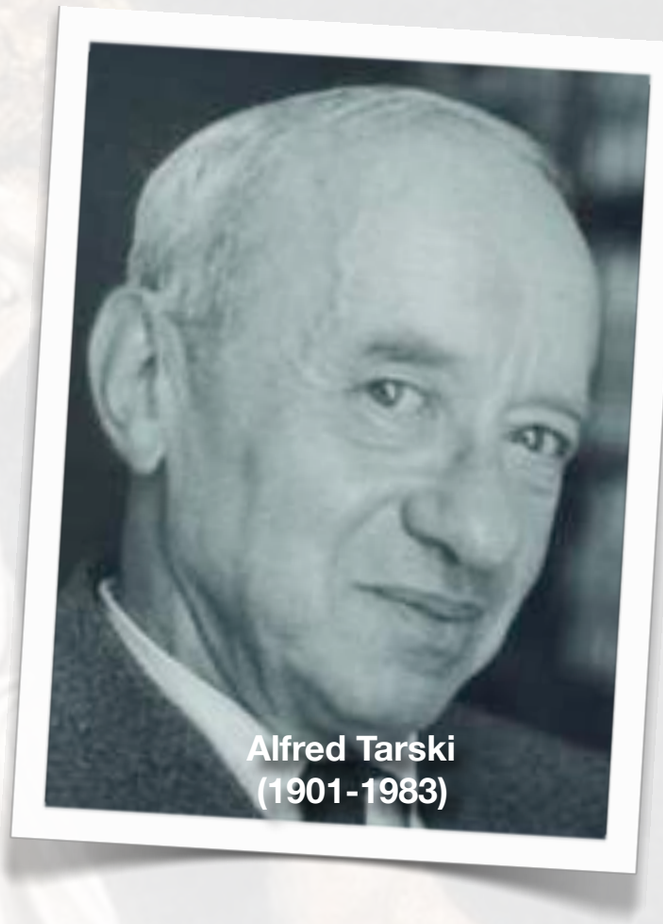
behaviour of the program at execution

procedural semantics

- the meaning of a language expression (program) is the procedure that takes place internally, whenever the expression does occur.

Model-theoretic Semantics

- **Model-theoretic semantics** performs the semantic interpretation of artificial and natural languages by „***identifying meaning with an exact and formally defined interpretation with a model***“
  - **= formal Interpretation with a model**


Alfred Tarski
(1901-1983)

- e.g. model-theoretic semantics of propositional logic
  - assignment of truth values „*true*“ and „*false*“ to atomic assertions and
  - description of logical connectives with *truth tables*

Model-theoretic Semantics

- **Example:** the language of arithmetics

- **Syntax:**
  - x+2 ≥ y is a formula
  - x2+y≥ is not a formula

- **Semantics:**
  - x+2 ≥ y is **true** under the interpretation x = 7, y = 1
  - x+2 ≥ y is **not true** under the interpretation x = 1, y = 7

- Inference/Entailment:
  - **Gaussian Elimination** is an algorithm for arithmetics.

Model-theoretic Semantics

- Any logic **L := (S , ⊨)** consists of
  (1) a ***set of statements* S** and
  (2) an ***entailment relation*** ⊨

- Let $\Phi \subseteq S$ and $\varphi \in S$: $\Phi \models \varphi$

- „ φ *is a logical consequence of* Φ“ or
  „*from the assertions of* Φ *follows the assertion* φ“

- If for 2 assertions $\varphi, \psi \in S$
  both $\{\varphi\} \models \psi$ and $\{\psi\} \models \varphi$,
  then both assertions φ and ψ are logically equivalent

$$\varphi \equiv \psi$$

Propositional Logic

logical connective	Name	intentional meaning
¬	Negation	„not“
∧	Conjunction	„and“
∨	Disjunction	„or“
→	Implication	„if – then“
↔	Equivalence	„if, and only if, then“

- **Logical connectives**: Op={ ¬,∧,∨,→,↔,(,) }, a set of symbols Σ with Σ∩Op=∅ and {`true`, `false`}
- Production rules for propositional formulae (propositions):
  - all atomic formulas are propositions (*all elements of* Σ)
  - if φ is a proposition, then also ¬φ
  - if φ and ψ are propositions, then also φ∧ψ, φ∨ψ, φ→ψ, φ↔ψ
- Priority: ¬ prior to ∧,∨ prior to →, ↔

Propositional Logic

- How to model facts?

Simple Assertions	**Modeling**
The moon is made of green cheese	g
It rains	r
The street is getting wet.	n

Composed Assertions	**Modeling**
if it rains, then the street will get wet.	$r \rightarrow n$
If it rains and the street does not get wet, then the moon is made of green cheese.	$(r \wedge \neg n) \rightarrow g$

PL - Model-theoretic Semantics

- **Interpretation `I`**:
  Mapping of all atomic propositions to {`t`,`f`}.
- If `F` is a formula and `I` an Interpretation, then `I(F)` is a truth value computed from `F` and `I` via truth tables.

I(p)	I(q)	I(¬p)	I(pvq)	I(p∧q)	I(p→q)	I(p↔q)
f	f	t	f	f	t	t
f	t	t	t	f	t	f
t	f	f	t	f	f	f
t	t	f	t	t	t	t

PL - Model-theoretic Semantics

- We write **I** ⊨ **F**, if **I(F)=t**,
  and call Interpretation I a **Model** of formula F.

- Rules of Semantics:
  - I is model of ¬φ, iff I is not a model of φ
  - I is model of (φ∧ψ), iff I is a model of φ AND of ψ
  - ...

- Basic concepts:
  - tautology
  - satisfiable
  - refutable
  - unsatisfiable (contradiction)

First Order Logic - FOL

Quantifier	Name	Intentional Meaning
∃	Existential Quantifier	„it exists“
∀	Universal Quantifier	„for all“

- **Operators (logical connectives)** as in propositional logic
- **Variables**, e.g., X,Y,Z,…
- **Constants**, e.g., a, b, c, …
- **Functions**, e.g., f, g, h, … (incl. arity)
- **Relations** / **Predicates**,e.g., p, q, r, … (incl. arity)

$$(\forall X)(\exists Y)\ ((p(X) \vee \neg q(f(X),Y)) \rightarrow r(X))$$

First Order Logic - FOL

FOL: Syntax

- „correct“ formulation of **Terms** from Variables, Constants and Functions:
  - f(X), g(a,f(Y)), s(a), .(H,T), x_location(Pixel)
- „correct“ formulation of **Atoms** from Relations with Terms as arguments
  - p(f(X)), q (s(a),g(a,f(Y))), add(a,s(a),s(a)), greater_than(x_location(Pixel),128)
- „correct“ formulation of **Formulas** from Atoms, Operators and Quantifiers:
  - (∀Pixel) (greater_than(x_location(Pixel),128) → red(Pixel))
- If in doubt, use brackets!
- All Variables should be quantified!

First Order Logic - FOL

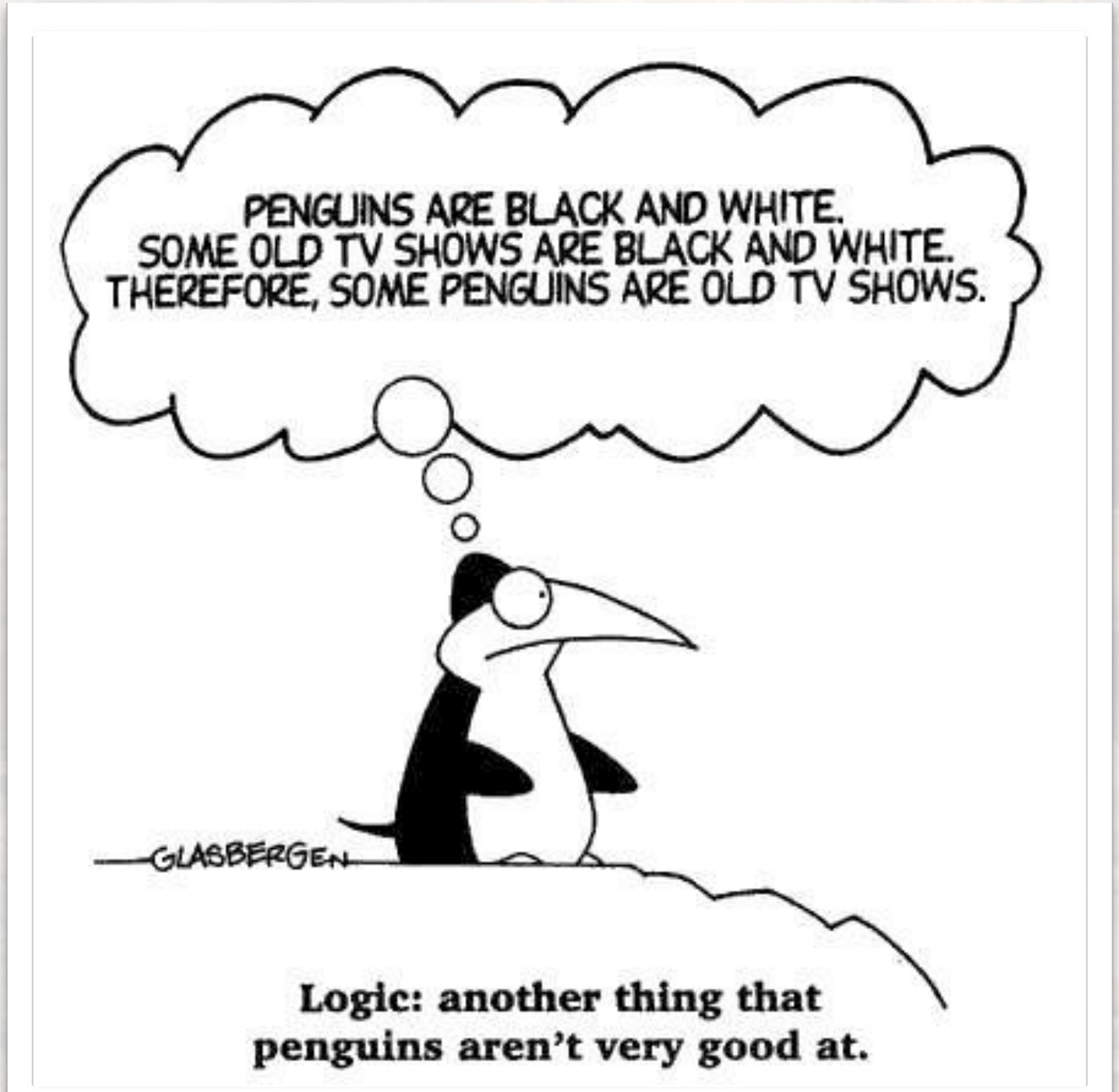
- How to model facts?
  - „All kids love icecream."
    ∀X: Child(X) → lovesIcecreme(X)
  - „the father of a person is a male parent."
    ∀X ∀Y: isFather(X,Y) ↔ (Male(X) ∧ isParent(X,Y))
  - „There are (one or more) interesting lectures."
    ∃X: Lecture(X) ∧ Interesting(X)
  - „The relation ‚isNeighbor' is symmetric."
    ∀X ∀Y: isNeighbor(X,Y) → isNeighbor(Y,X)

FOL - Model-theoretic Semantics

- **Structure**:
  - Definition of a **domain** `D`.
  - **Constant symbols** are mapped to elements of `D`.
  - **Function symbols** are mapped to funtions in `D`.
  - **Relation symbols** are mapped to relations over `D`.

- **Then**:
  - **Assertions** will become elements of `D`.
  - **Relation symbols** with arguments will become `true` or `false`.
  - **Logical connectives** and **quantifiers** are treated likewise.

The Penguin Example...



FOL - Model-theoretic Semantics

- The Penguin Example

$$((\forall X)(penguin(X) \rightarrow blackandwhite(X)) \wedge (\exists X)(oldTVshow(X) \wedge blackandwhite(X))) \rightarrow (\exists X)(penguin(X) \wedge oldTVshow(X))$$

- Intentional Semantics?

FOL - Model-theoretic Semantics

- The Penguin Example

$$((\forall X)(penguin(X) \rightarrow blackandwhite(X)) \wedge (\exists X)(oldTVshow(X) \wedge blackandwhite(X))) \rightarrow (\exists X)(penguin(X) \wedge oldTVshow(X))$$

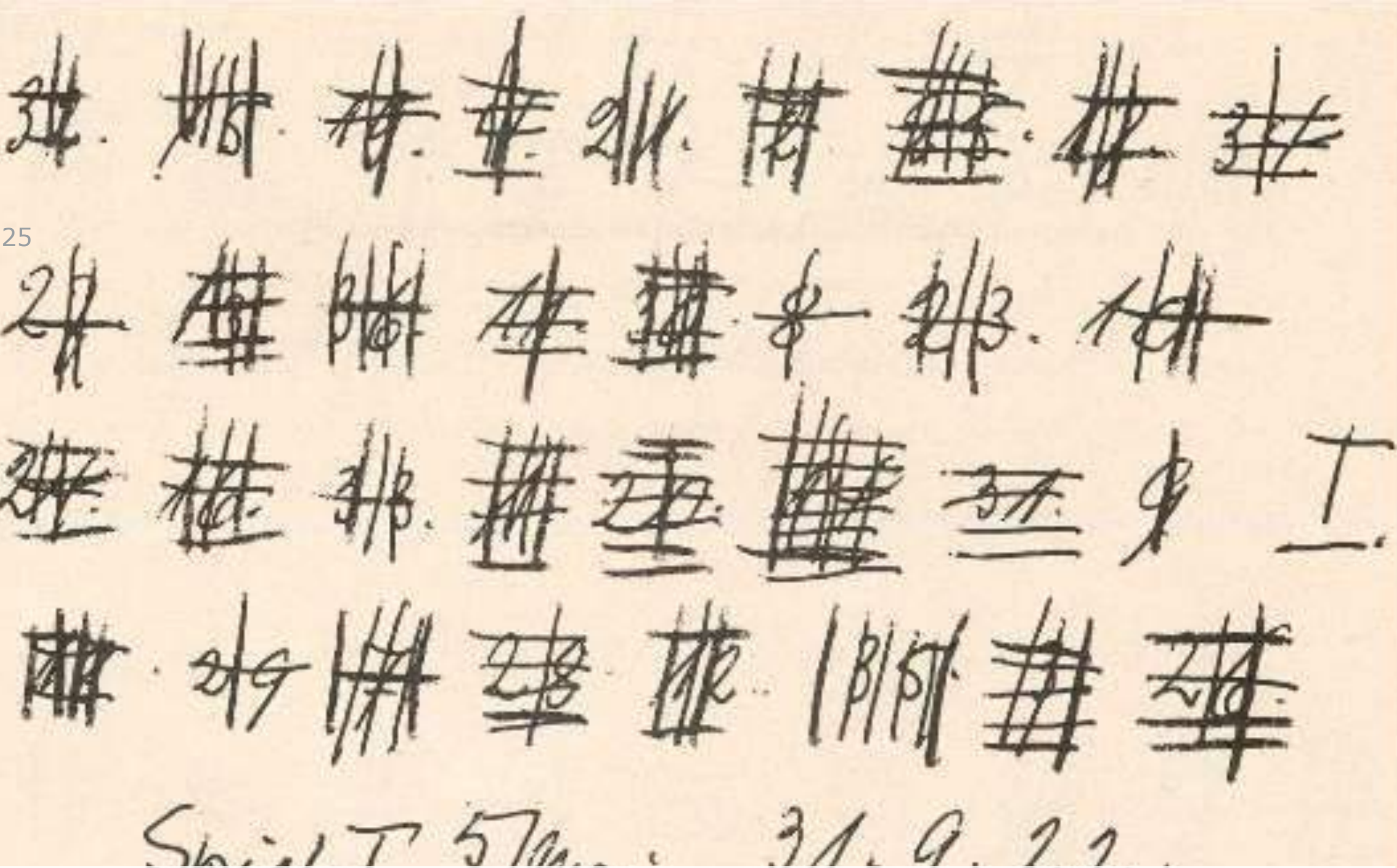
- Interpretation `I`:
  - **Domain**: a set `M`, containing elements `a,b`.
  - … no constant of function symbols …
  - We show: the formula is *refutable* (i.e. it is not a tautology):
  - If `I(penguin)(a), I(blackandwhite)(a), I(oldTVshow)(b), I(blackandwhite)(b)` is `true` , `I(oldTVshow)(a)` and `I(penguin)(b)` is `false`,
  - then the formula with Interpretation `I` is `false`, d.h. `I` $\not\models$ `F`

FOL - Logical Entailment

- a **theory** T is a set of formulas.
- an interpretation I is a **model** of T,
  iff I ⊨ G for all formulas G in T.
- a formula F is a **logical consequence** of T,
  iff all models of T are also models of F.
- then we write **T ⊨ F**.

- two formulas F,G are called **logically equivalent**,
  iff {F}⊨G and {G}⊨F.
- then we write **F ≡ G**

Theory ≙ Knowledge Base



04 Tablaux Algorithm for Automated Reasoning

Open HPI Course: Knowledge Engineering with Semantic Web Technologies
Lecture 04: Knowledge Representations Pt.1

Knowledge Engineering with Semantic Web Technologies

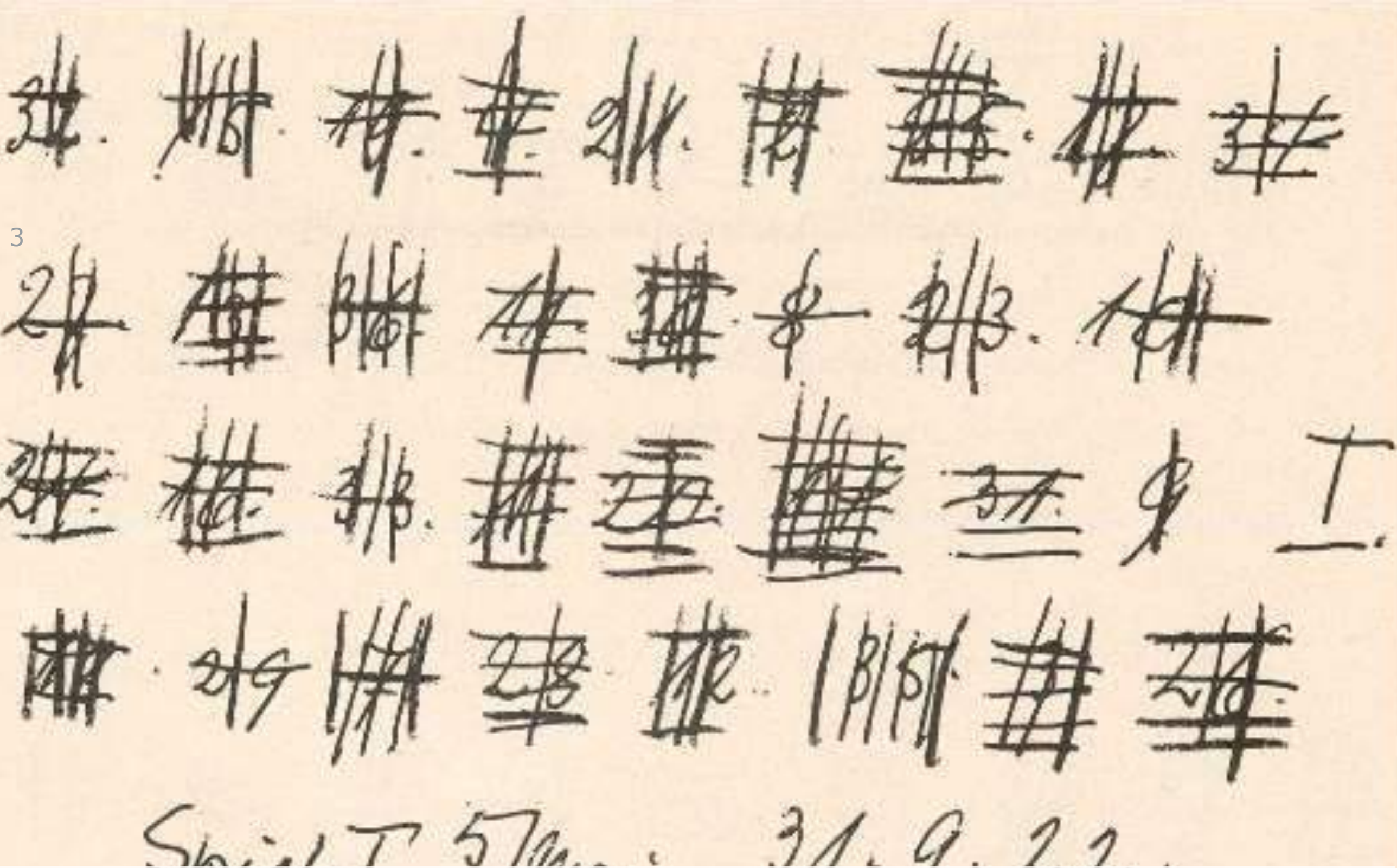
Lecture 4: Knowledge Representation

04: Automated Reasoning - Tableaux Algorithm

Dr. Harald Sack
Hasso-Plattner-Institut for IT Systems Engineering
University of Potsdam
Spring 2014

Lecture 4: Knowledge Representations I

Open HPI Course: Knowledge Engineering with Semantic Web Technologies



04 Automated Reasoning - Tablaux Algorithm

Open HPI Course: Knowledge Engineering with Semantic Web Technologies
Lecture 04: Knowledge Representations Pt.1

Logical Equivalences

Augustus De Morgan
(1806-1871)

$F \wedge G \equiv G \wedge F$
$F \vee G \equiv G \vee F$

$F \rightarrow G \equiv \neg F \vee G$
$F \leftrightarrow G \equiv (F \rightarrow G) \wedge (G \rightarrow F)$

$\neg(F \wedge G) \equiv \neg F \vee \neg G$
$\neg(F \vee G) \equiv \neg F \wedge \neg G$

DeMorgans' Laws

$\neg\neg F \equiv F$

$F \wedge t \equiv F$ $\quad$ $F \wedge f \equiv f$
$F \vee t \equiv t$ $\quad$ $F \vee f \equiv F$

$F \vee (G \wedge H) \equiv (F \vee G) \wedge (F \vee H)$
$F \wedge (G \vee H) \equiv (F \wedge G) \vee (F \wedge H)$

$\neg(\forall X)\, F \equiv (\exists X)\, \neg F$
$\neg(\exists X)\, F \equiv (\forall X)\, \neg F$

$(\forall X)(\forall Y)\, F \equiv (\forall Y)(\forall X)\, F$
$(\exists X)(\exists Y)\, F \equiv (\exists Y)(\exists X)\, F$

$(\forall X)\, (F \wedge G) \equiv (\forall X)\, F \wedge (\forall X)\, G$
$(\exists X)\, (F \vee G) \equiv (\exists X)\, F \vee (\exists X)\, G$

$F \wedge \neg F = f$ $\quad$ $F \vee \neg F = t$

Commutativity of Quantifiers

- Quantifiers (of the same sort) are commutative

  $(\forall X)(\forall Y)\ F \equiv (\forall Y)(\forall X)\ F$
  $(\exists X)(\exists Y)\ F \equiv (\exists Y)(\exists X)\ F$

- But

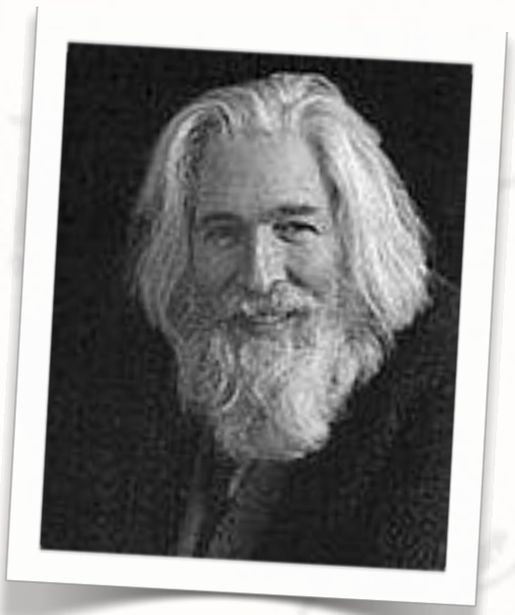
  $(\exists X)(\forall Y)\ F \not\equiv (\forall Y)(\exists X)\ F$

- Example:
  - $(\exists x)(\forall y)$: loves(x,y)
    „There exists somebody, who loves everybody.“
  - $(\forall y)(\exists x)$: loves(x,y)
    „Everybody is loved by somebody.“

Tableaux Algorithm

Evert Willem Beth
(1908-1964)

- For **automated reasoning**, we need syntactic algorithms to check the consistency of logical assertions
- The **method of analytical tableaux** is based on disjunctive normal form of logical expressions
  - invented by Dutch logician Evert Willem Beth in 1955 and simplified by Raymond Smullyan.

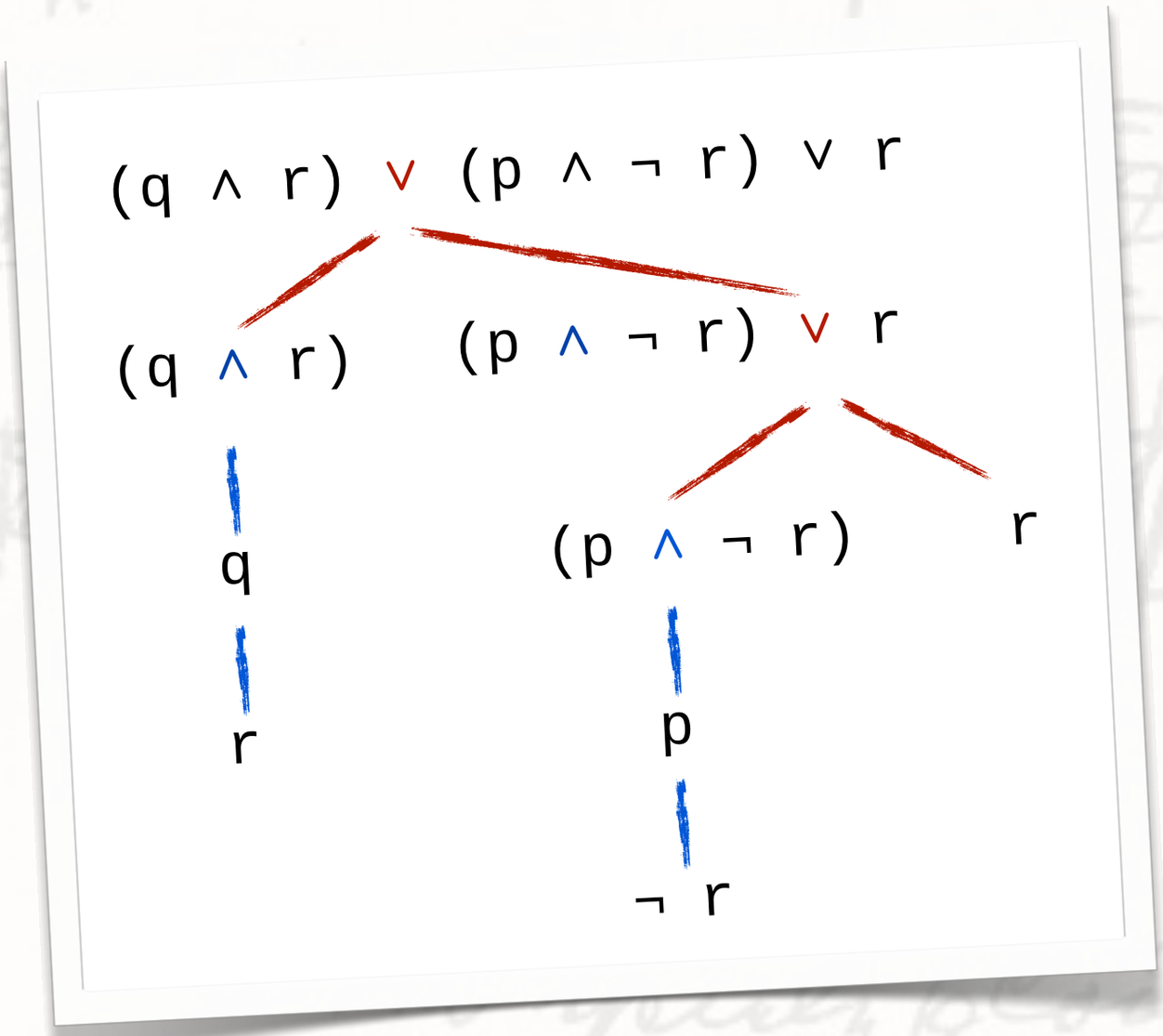


Raymond Merrill Smullyan

- **Basic Idea of Tableaux Algorithm**:
  - Proof algorithm (decision procedure) to check the consistency of a logical formula by inferring that its negation is a contradiction (***proof by refutation***).

Beth, Evert W., 1955. "Semantic entailment and formal derivability", *Mededlingen van de Koninklijke Nederlandse Akademie van Wetenschappen, Afdeling Letterkunde*, N.R. Vol 18, no 13, 1955, pp 309–42.

Tableaux Algorithm for PL



(1) Construct **Decision Tree**, where each node is marked with a logical formula.

- A **path from the root to a leaf** is the **conjunction** of all formulas represented within the nodes of the path;
- a **branch of the path** represents a **disjunction**.

Tableaux Algorithm for PL

(1) Construct **Decision Tree**, where each node is marked with a logical formula.

- A **path from the root to a leaf** is the **conjunction** of all formulas represented within the nodes of the path;
- a **branch of the path** represents a **disjunction**.

(2) The tree is created by successive application of the **Tableaux Extension Rules** (which are selected non deterministically).

(3) A **path** in the Tableaux is **closed**,

- if along the path as well X as ¬X for a formula X occurs (X doesn‘t have to be atomic) or
- if ***false*** occurs .

(4) A tableaux is **fully expanded**, if no more extension rules can be applied.

(5) A tableaux is called **closed**, if all its paths are closed.

(6) A **Tableaux Proof** for a formula X is a closed tableaux for ¬X.

Tableaux Extension Rules for PL

- for PL:

$\frac{\neg\neg X}{X}$	$\frac{\neg T}{F}$	$\frac{\neg F}{T}$

- for *conjunctive Formula* (α-Rules):

$\frac{\alpha}{\alpha_1}$	$\frac{X \wedge Y}{X}$	$\frac{\neg(X \vee Y)}{\neg X}$	$\frac{\neg(X \Rightarrow Y)}{X}$
α_2	Y	¬Y	¬Y

- for *disjunctive formula* (β-Rules):

$\frac{\beta}{\beta_1 \mid \beta_2}$	$\frac{X \vee Y}{X \mid Y}$	$\frac{\neg(X \wedge Y)}{\neg X \mid \neg Y}$	$\frac{(X \Rightarrow Y)}{\neg X \mid Y}$

Tableaux Algorithm (PL) -Example1

proof: $((q \wedge r) \Rightarrow (\neg q \vee r))$

α-Rule
$\neg(X \Rightarrow Y)$
X
$\neg Y$

(1) $\neg((q \wedge r) \Rightarrow (\neg q \vee r))$

(2) α,1: $(q \wedge r)$

(3) α,1: $\neg(\neg q \vee r) = q \wedge \neg r$

Tableaux Algorithm (PL) -Example1

proof: $((q \wedge r) \Rightarrow (\neg q \vee r))$

(1) $\neg((q \wedge r) \Rightarrow (\neg q \vee r))$

(2) α,1: $(q \wedge r)$

(3) α,1: $\neg(\neg q \vee r) = q \wedge \neg r$

(4) α,2: q

(5) α,2: r

(6) α,3: q

(7) α,3: ¬r

α-Rule

$$\frac{X \wedge Y}{\begin{array}{c} X \\ Y \end{array}}$$

- path is closed
- tableaux is closed

Tableaux Algorithm (PL) -Example2

proof: (p⇒(q⇒r)) ⇒ ((p⇒q)⇒(p⇒r))

α-Rule

$\frac{\neg(X \Rightarrow Y)}{X \quad \neg Y}$

(1) ¬((p⇒(q⇒r)) ⇒ ((p⇒q)⇒(p⇒r)))

(2|α from 1) (p⇒(q⇒r))

(3|α from 1) ¬((p⇒q)⇒(p⇒r))

(4|α from 3) (p⇒q)

(5|α from 3) ¬(p⇒r)

(6|α from 5) p

(7|α from 5) ¬r

Tableaux Algorithm (PL) -Example2

proof: (p⇒(q⇒r)) ⇒ ((p⇒q)⇒(p⇒r))

(1) ¬((p⇒(q⇒r)) ⇒ ((p⇒q)⇒(p⇒r)))

(2|α from 1) (p⇒(q⇒r))

(3|α from 1) ¬((p⇒q)⇒(p⇒r))

(4|α from 3) (p⇒q)

(5|α from 3) ¬(p⇒r)

(6|α from 5) p

(7|α from 5) ¬r

(8|β from 2) ¬p | (9|β from 2) (q⇒r)

(10|β from 9) ¬q | (11|β from 9) r

(12|β from 4) ¬p | (13|β from 4) q

β-Rule

(X⇒Y)

¬X | Y

Tableaux Algorithm Extension (FOL)

- as for propositional logic - X and Y stand for arbitrary (FOL) formulas
- Additional Rules for quantified formulas :

$$\frac{\gamma}{\gamma[t]} \qquad \frac{\delta}{\delta[c]}$$

- γ for universally quantified formulas, δ existentially quantified formulas, with:

γ	$\gamma[t]$
$\forall x.\Phi$	$\Phi[x \leftarrow t]$
$\neg\exists x.\Phi$	$\neg\Phi[x \leftarrow t]$

δ	$\delta[c]$
$\exists x.\Phi$	$\Phi[x \leftarrow c]$
$\neg\forall x.\Phi$	$\neg\Phi[x \leftarrow c]$

- t is an arbitrary ground term
  (i.e. doesn‘t contain variables that are bound in Φ),
- c is a „new“ constant

Tableaux Algorithm (FOL) -Example3

γ	γ[t]
∀x.Φ	Φ[x←t]
¬∃x.Φ	¬Φ[x←t]

δ	δ[c]
∃x.Φ	Φ[x←c]
¬∀x.Φ	¬Φ[x←c]

Proof: (∀x.P(x)∨Q(x))⇒(∃x.P(x))∨(∀x.Q(x))

(1) ¬((∀x.P(x)∨Q(x))⇒(∃x.P(x))∨(∀x.Q(x)))

(2|α from 1) (∀x.P(x)∨Q(x))

(3|α from 1) ¬((∃x.P(x))∨(∀x.Q(x)))

(4|α from 3) ¬(∃x.P(x))

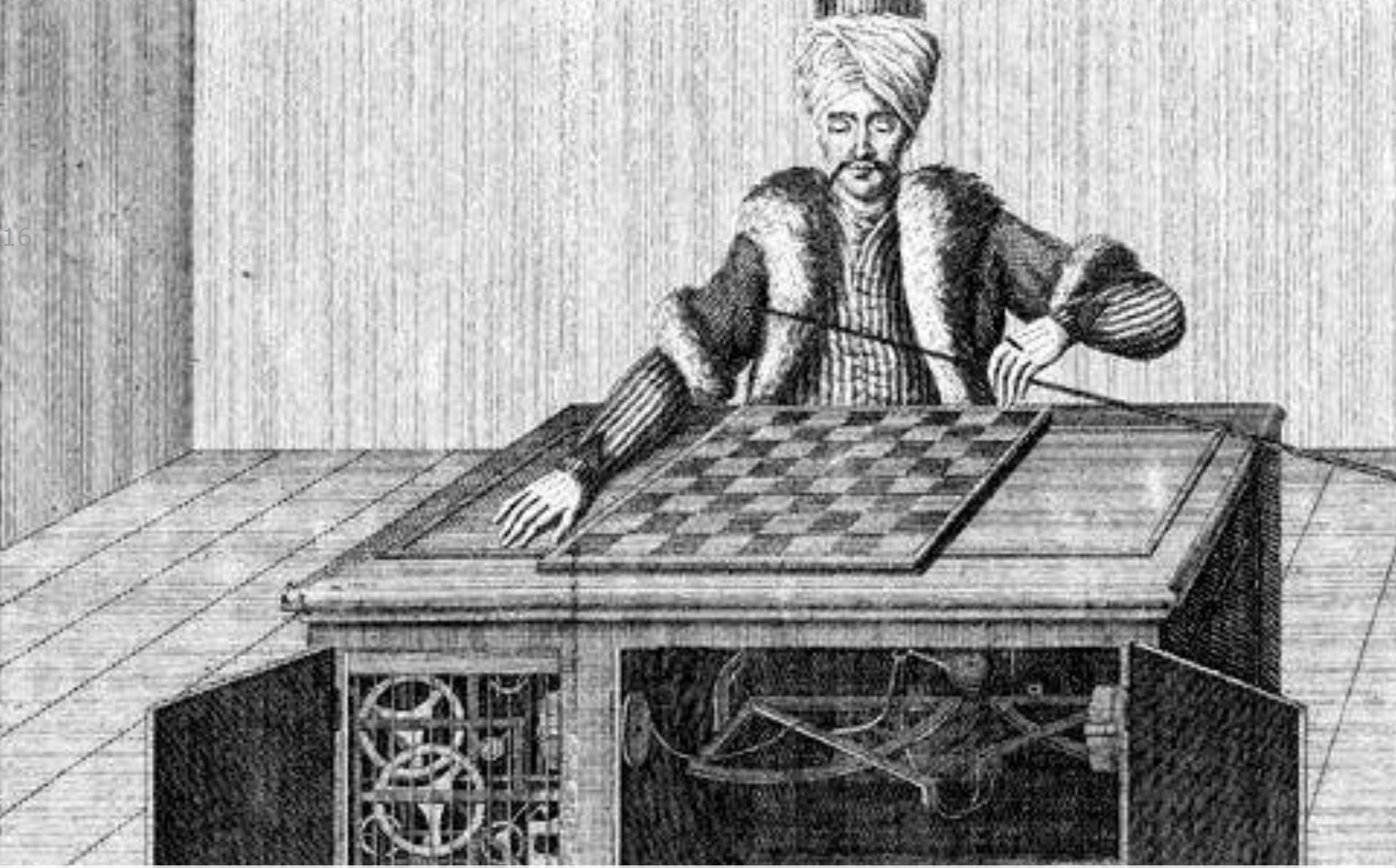
(5|α from 3) ¬(∀x.Q(x))

(6|δ from 5) ¬Q(c)

(7|γ from 4) ¬P(c)

(8|γ from 2) P(c)∨Q(c)

(9|β from 8) P(c) | (10|β from 8) Q(c)



05 Description Logics - ALC

Open HPI Course: Knowledge Engineering with Semantic Web Technologies
Lecture 04: Knowledge Representations Pt.1

Knowledge Engineering with Semantic Web Technologies

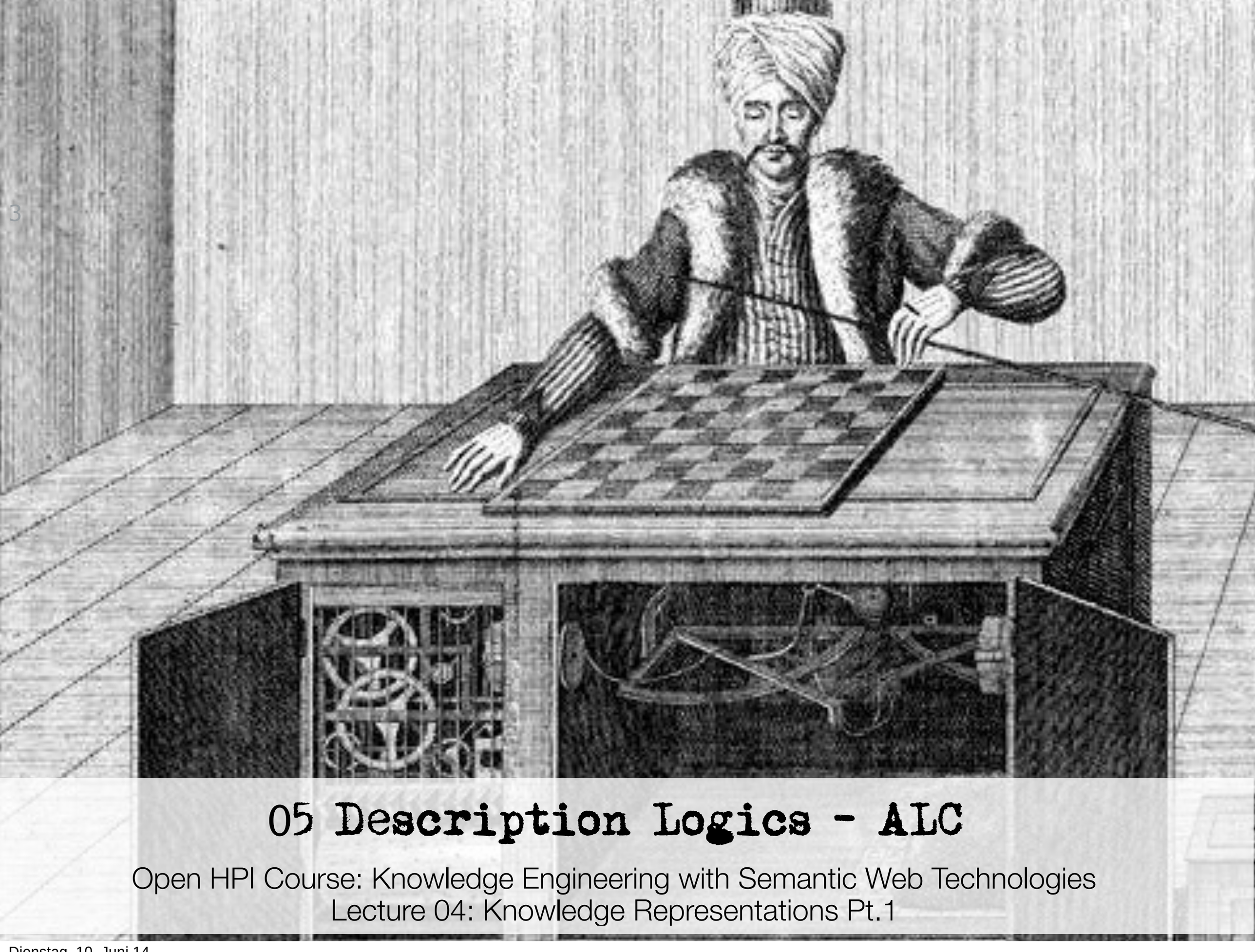
Lecture 4: Knowledge Representation

05: Description Logics - ALC

Dr. Harald Sack
Hasso-Plattner-Institut for IT Systems Engineering
University of Potsdam
Spring 2014

Lecture 4: Knowledge Representations I

Open HPI Course: Knowledge Engineering with Semantic Web Technologies



05 Description Logics - ALC

Open HPI Course: Knowledge Engineering with Semantic Web Technologies
Lecture 04: Knowledge Representations Pt.1

FOL as Semantic Web Language?

- Why not simply take FOL for Ontologies?
- FOL can do everything...
  - *you could also program everything with assembler instead of with higher programming languages*
- FOL has
  - high **expressivity**
  - too **bulky** for modelling
  - not appropriate to find consensus in modelling
  - proof theoretically very **complex** (semi-decidable)
- *FOL is of course not a Markup Language for the Web*

▶ Look for an appropriate fragment of FOL....

▶ *...and then make it a vocabulary for RDF(S)*

Description Logics (DLs)

- DLs are Fragments of FOL
- In DL from *simple descriptions* more *complex descriptions* are created with the help of ***Constructors***.
- DLs differ in the applied constructors (Expressivity)
- DLs have been developed from „semantic Networks“
- DLs are **decidable** (most times)
- DLs posess **sufficient expressivity** (most times)
- DLs are related to modal logics

- e.g.,
  W3C Standard OWL 2 DL is based on description logics
  $\mathcal{SHROIQ}(\mathcal{D})$

General DL Architecture

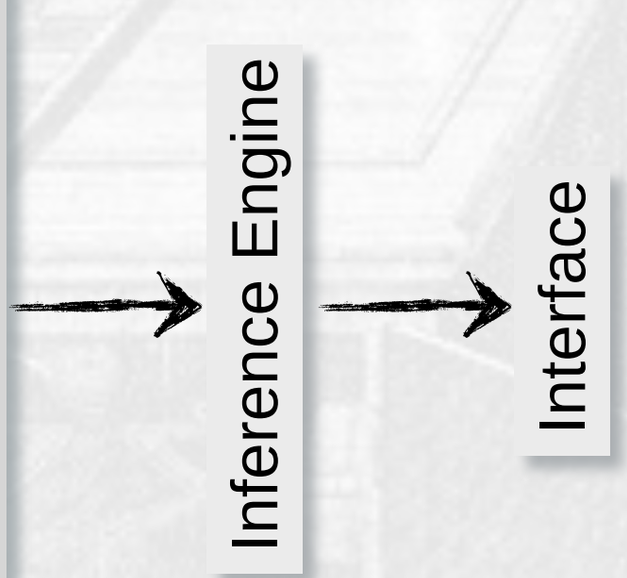
Knowledge Base

TBox **Terminological Knowledge**
Knowledge about concepts of a domain (classes, attributes, properties,..)

```
Writer ≡ Person ⊓ ∃author.Book
```

ABox **Assertional Knowledge**
Knowledge about Individuals / Entities

```
Writer(GeorgeOrwell)
```



Description Logics (DLs)

- DLs are a **family** of logic-based formalisms applied for knowledge representation
- Special languages characterized by:
  - ***Constructors*** for complex concepts and roles from simpler concepts and roles
  - Set of ***Axioms*** to express Facts about concepts, roles and individuals
  - $\mathcal{ALC}$ (*Attribute Language with Complement*) is the smallest deductively complete DL
    - Conjunction, Disjunction, Negation are class constructors, denoted as $\sqcap$, $\sqcup$, $\neg$
    - Quantifiers restrict domain and range of roles

Attributive Language with Complements $\mathcal{ALC}$

Building Blocks:

- Classes
- Roles / Properties
- Individuals

- `Writer(GeorgeOrwell)`
  Individual GeorgeOrwell is of class Writer

- `Book(NineteenEightyfour)`
  Individual NineteenEightyfour is of class Book

- `author(NineteenEightyfour, GeorgeOrwell)`
  The book NineteenEightyfour has the author GeorgeOrwell

$\mathcal{ALC}$ - Building Blocks

- **Atomic Types**
  - Concept names `A, B, ...`
  - Special concepts
    - `⊤` - Top (universal concept)
    - `⊥` - Bottom concept
  - Role names `R, S, ...`

- **Constructors**
  - Negation: `¬C`
  - Conjunction: `C ⊓ D`
  - Disjunction: `C ⊔ D`
  - Existential quantifier: `∃R.C`
  - Universal quantifier: `∀R.C`

$\mathcal{ALC}$ - Building Blocks

- **Class Inclusion**
  - `Novel ⊑ Book`
    - *every novel is also a book*
    - equals (∀x)(Novel(x) → Book(x))

- **Class Equivalence**
  - `Novel ≡ Prose`
    - *all Prose are exactly Novels*
    - equals (∀x)(Novel(x) ↔ Prose(x))

$\mathcal{ALC}$ - Complex Class Relations

- Conjunction ⊓
- Disjunction ⊔
- Negation ¬

```
Novel  ⊑ (Book ⊓ Fiction)
         ⊔ (Paperback ⊓ ¬Poetry)
```

$$(\forall x)(Novel(x) \rightarrow ((Book(x) \wedge Fiction(x)) \vee (Paperback(x) \wedge \neg Poetry(x))))$$

$\mathcal{ALC}$ - Quantifiers on Roles

- **Strict Binding** of the Range of a Role to a Class
  - `Book ⊑ ∀author.Writer`
  - *A Book must be authored by a Writer*
  - (∀x)(Book(x) → (∀y)(author(x,y) → Writer(y)))

- **Open Binding** of the Range of a Role to a Class
  - `Book ⊑ ∃author.Person`
  - *Every Book has at least one author (who is a person)*
  - (∀x)(Book(x) → (∃y)(author(x,y) ∧ Person(y)))

$\mathcal{ALC}$ - Formal Syntax

- Production rules for creating classes in $\mathcal{ALC}$:

*(**A** is an atomic class, **C** and **D** are complex Classes and **R** a Role)*

- $C, D ::= A \mid \top \mid \bot \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \exists R.C \mid \forall R.C$

- An $\mathcal{ALC}$ **TBox** contains assertions of the form $C \sqsubseteq D$ and $C \equiv D$, where C, D are complex classes.

- An $\mathcal{ALC}$ **ABox** contains assertions of the form $C(a)$ and $R(a,b)$, where C is a complex Class, R a Role and a, b Individuals.

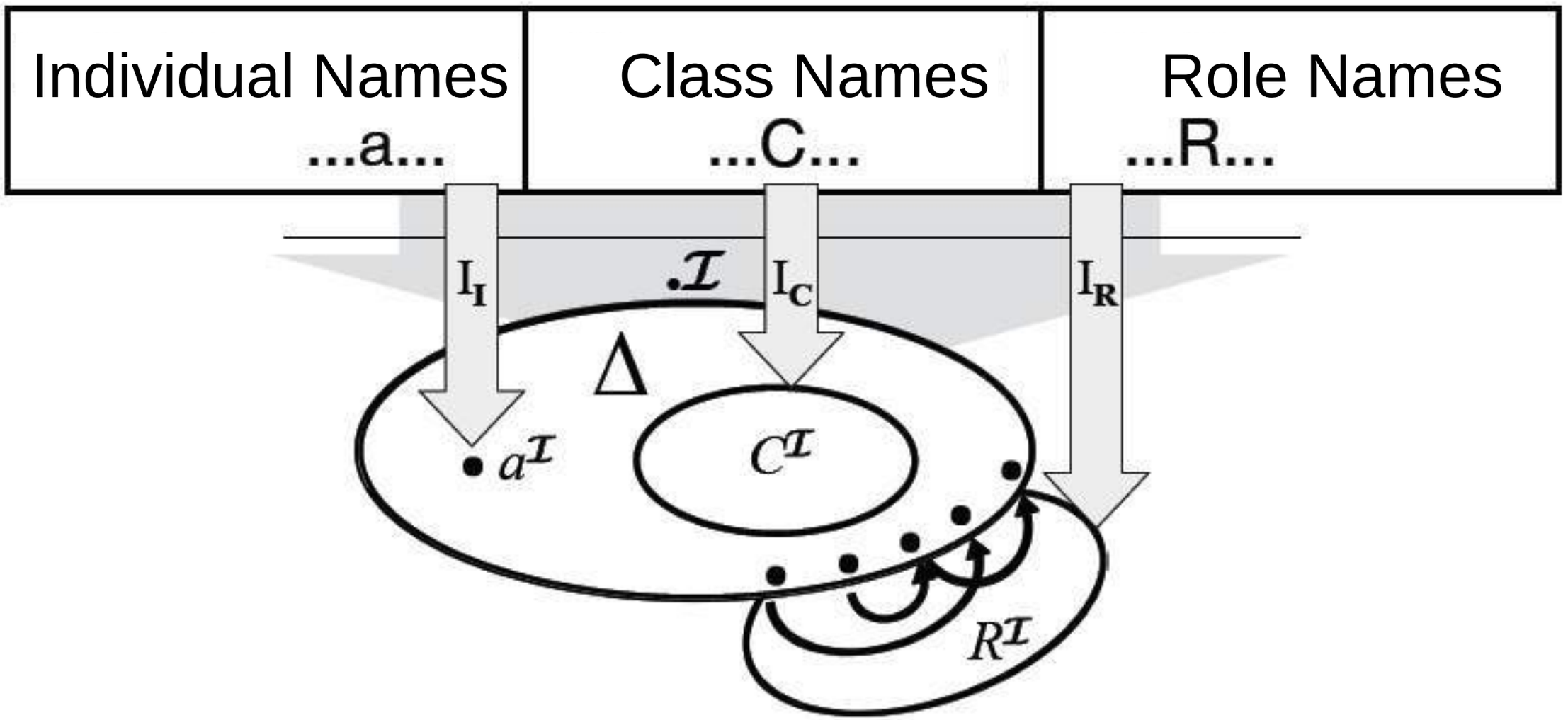
- An $\mathcal{ALC}$-**Knowledge Base** contains an ABox and a TBox.

$\mathcal{ALC}$ - Semantic (Interpretation)

- we define a model-theoretic semantic for $\mathcal{ALC}$
  (*i.e. Entailment will be defined via Interpretations*)

- an **Interpretation** $\mathtt{I}=(\Delta^{\mathtt{I}}, .^{\mathtt{I}})$ contains
  - a set $\Delta^{\mathtt{I}}$ (**Domain**) of Individuals and
  - an **interpretation function** $.^{\mathtt{I}}$ that maps
    - Individual names $\mathtt{a}$
      to domain elements $\mathtt{a}^{\mathtt{I}} \in \Delta^{\mathtt{I}}$
    - Class names $\mathtt{C}$
      to a set of domain elements $\mathtt{C}^{\mathtt{I}} \subseteq \Delta^{\mathtt{I}}$
    - Role names $\mathtt{R}$
      to a set of pairs of domain elements $\mathtt{R}^{\mathtt{I}} \subseteq \Delta^{\mathtt{I}} \times \Delta^{\mathtt{I}}$

$\mathcal{ALC}$ - Semantic (Interpretation)



$\mathcal{ALC}$ - Semantic (Interpretation)

- Extension (of Interpretation $\mathcal{I}$) for complex classes:

- $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$ and $\bot^{\mathcal{I}} = \varnothing$
- $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$ and
  $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$
- $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
- $\forall R.C = \{a \in \Delta^{\mathcal{I}} \mid (\forall b \in \Delta^{\mathcal{I}})((a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}})\}$
- $\exists R.C = \{a \in \Delta^{\mathcal{I}} \mid (\exists b \in \Delta^{\mathcal{I}})((a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}})\}$

$\mathcal{ALC}$ - Semantic (Interpretation)

- ...and Axioms:

- $C(a)$ holds, iff $a^I \in C^I$
- $R(a,b)$ holds, iff $(a^I, b^I) \in R^I$
- $C \sqsubseteq D$ holds, iff $C^I \subseteq D^I$
- $C \equiv D$ holds, iff $C^I = D^I$

$\mathcal{ALC}$ - Knowledgebase

- **Terminological Knowledge (TBox)**
  Axioms that describe the Structure of the modelled domain (conceptional Schema):
  - `Writer ⊑ ∃author.Book`
  - `ScienceFictionNovel ≡ Novel ⊓ ∃literaryGenre.ScienceFiction`

- **Assertional Knowledge (ABox)**
  Axioms that describe specific situations (data):
  - `Writer(GeorgeOrwell)`
  - `author(NineteenEightyfour,GeorgeOrwell)`

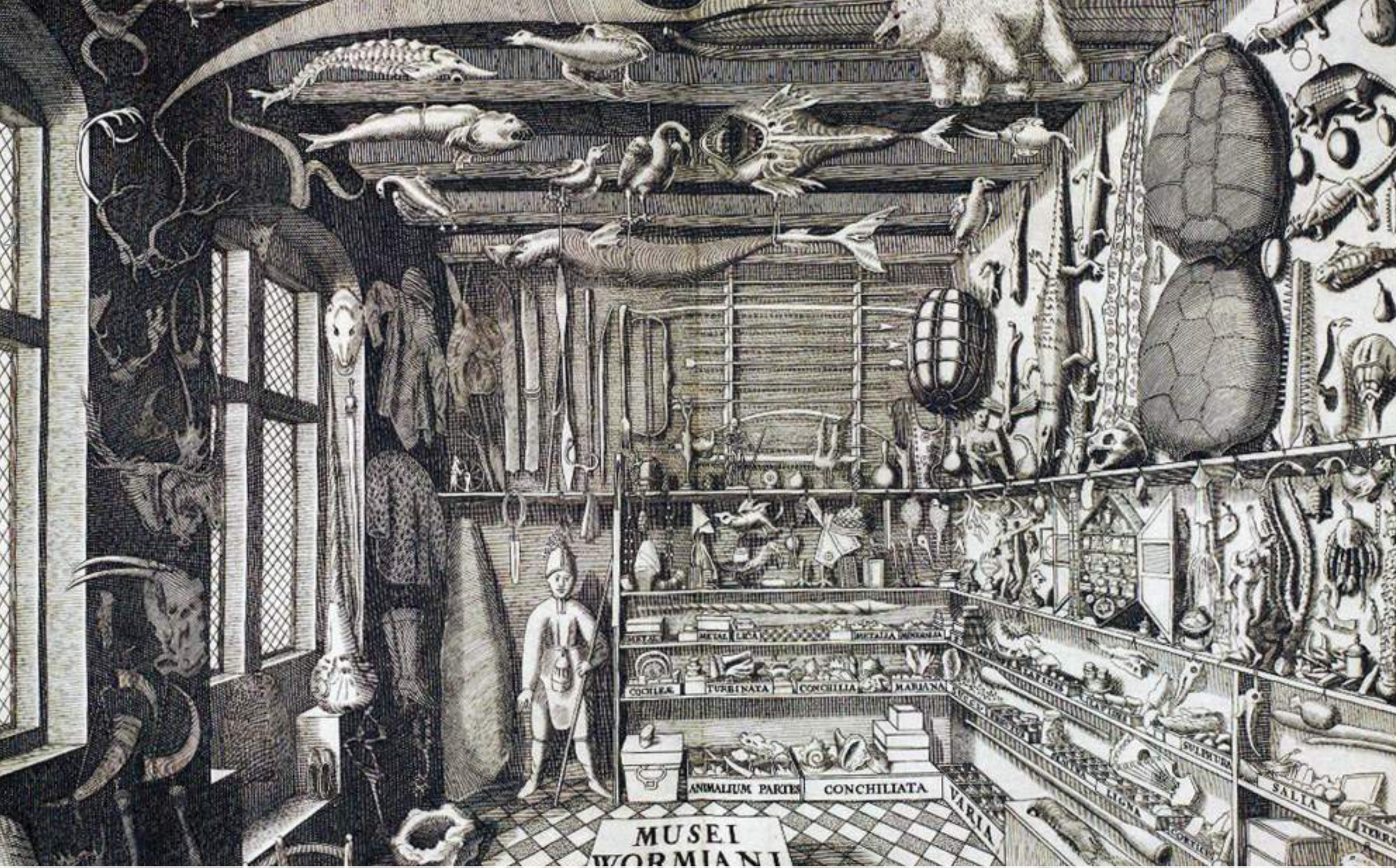
Description Logics

Operator/Constructor	Syntax	Language	
Conjunction	$A \sqcap B$	FL	S*
Value Restriction	$\forall R.C$		
Existential Quantifier	$\exists R$		
Top	$\top$	AL*	
Bottom	$\bot$		
Negation	$\neg A$		
Disjunction	$A \sqcup B$		
Existential Restriction	$\exists R.C$		
Number Restriction	$(\leq nR)\ (\geq nR)$		
Set of Inividuals	$\{a_1,...,a_2\}$		
Role Hierarchy	$R \sqsubseteq S$	H	
inverse Role	R^{-1}	I	
Qualified Number Restriction	$(\leq nR.C)\ (\geq nR.C)$	Q	

Description Logics

- $\mathcal{ALC}$: Attribute Language with Complement
- $\mathcal{S}$: $\mathcal{ALC}$ + Transitivity of Roles
- $\mathcal{H}$: Role Hierarchies
- $\mathcal{O}$: Nominals
- $\mathcal{I}$: Inverse Roles
- $\mathcal{N}$: Number restrictions ≤n R etc.
- $\mathcal{Q}$: Qualified number restrictions ≤n R.C etc.
- $(\mathcal{D})$: Datatypes
- $\mathcal{F}$: Functional Roles
- $\mathcal{R}$: Role Constructors

- OWL 2 DL is $\mathcal{SHROIQ}(\mathcal{D})$



06 DL Inference and Reasoning

Open HPI Course: Knowledge Engineering with Semantic Web Technologies
Lecture 04: Knowledge Representations Pt.1

Knowledge Engineering with Semantic Web Technologies

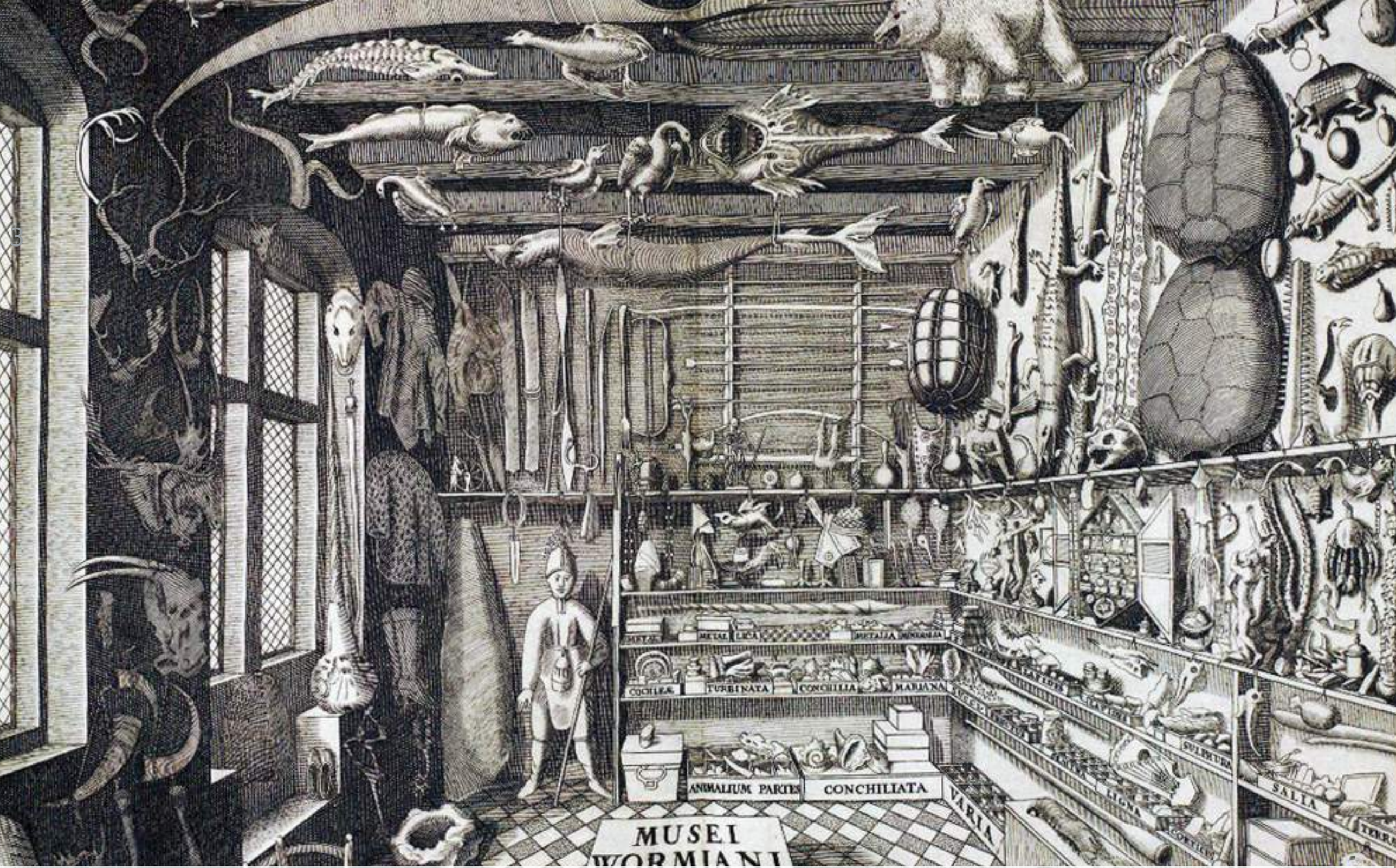
Lecture 4: Knowledge Representation

06: Description Logics - Inference and Reasoning

Dr. Harald Sack
Hasso-Plattner-Institut for IT Systems Engineering
University of Potsdam
Spring 2014

Lecture 4: Knowledge Representations I

Open HPI Course: Knowledge Engineering with Semantic Web Technologies

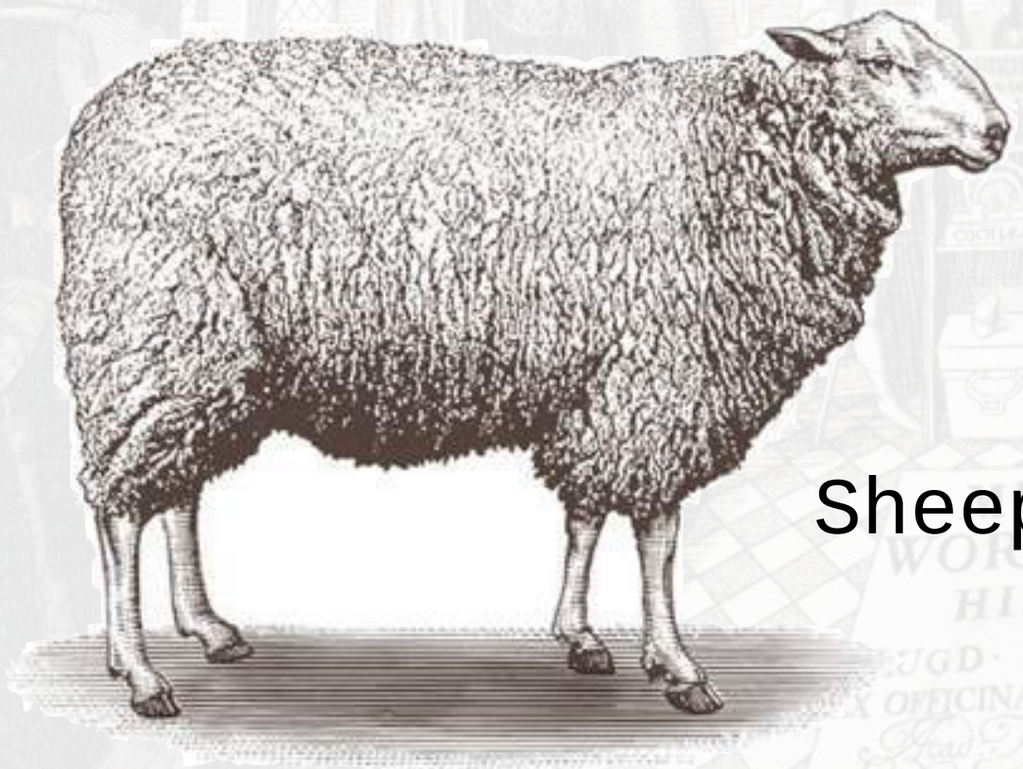


06 DL Inference and Reasoning

Open HPI Course: Knowledge Engineering with Semantic Web Technologies
Lecture 04: Knowledge Representations Pt.1

Open World Assumption - OWA

- When we have an **empty** DL ontology, **everything is possible**
- We then **constrain an ontology iteratively**, making it more restrictive as we go
- We state what is **not possible**, what is **forbidden** or **excluded**

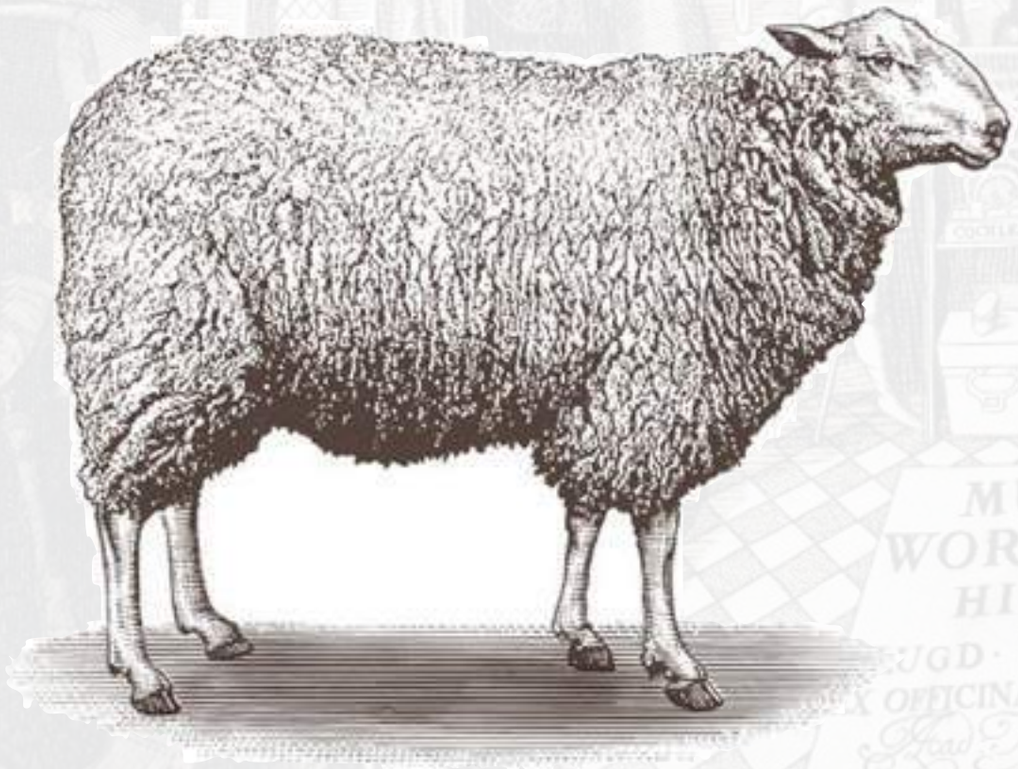


$$Sheep \sqsubseteq Animal \sqcap \forall hasLimbs.Leg$$

Open World Assumption - OWA

- Can Sheep fly? $\texttt{Sheep} \sqsubseteq \texttt{Animal} \sqcap \forall \texttt{hasLimbs.Leg}$

- No idea, but **probably yes** (according to our knowledge base)



- In the OWA, unless we have a statement (or we can infer) "*sheep can/cannot fly*" we return "***don't know***"
- In the real world, we are used to deal with incomplete information

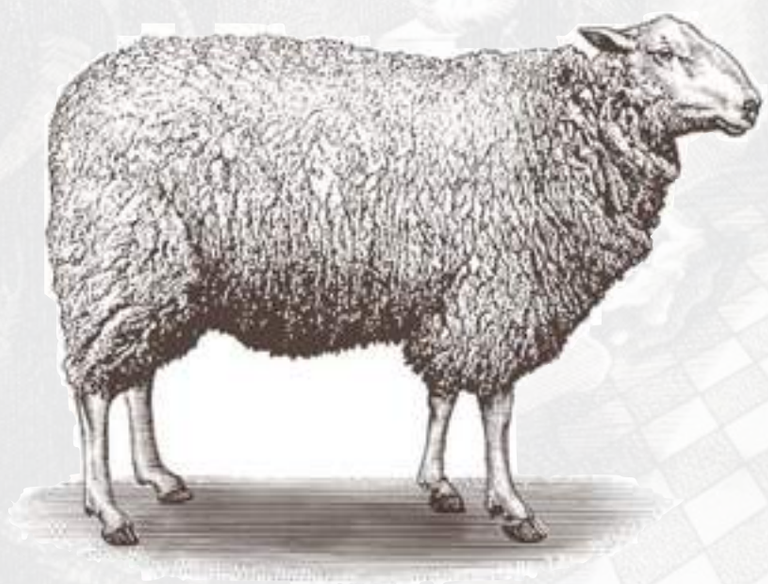
Open World Assumption - OWA

- In the Semantic Web we expect people to extend our own models
  (*but we don‘t worry in advance how*)
- The OWA assumes incomplete information by default
- Therefore, we can intentionally underspecify our models and allow others to reuse and extend

$$\texttt{Sheep} \sqsubseteq \texttt{Animal} \sqcap \forall \texttt{hasLimbs.Leg} \sqcap \texttt{canFly}$$

Closed World Assumption - CWA

- Closed World Systems require a place to put everything
- You can’t say anything until there’s somewhere to say it, as e.g. a slot on a frame, field on an OO class, column in a DB
- In Close World Systems, we state **what is possible** and **have to specify all knowledge**
- the CWA holds that **anything that cannot be shown to be true is false**; no explicit declaration of falsehood is needed.



$$Sheep \sqsubseteq Animal \sqcap \forall hasLimbs.Leg$$

Sheep can't fly!

Open World vs. Closed World Assumption

- **OWA: Open World Assumption**
  The existence of further individuals is possible, if they are not explicitly excluded.

- **CWA: Closed World Assumption**
  It is assumed that the knowledge base contains all individuals.

	are all children of Bill male?	*no idea since we do not know all children of Bill*	*if we assume that we know everything about Bill then all of his children are male*
`child(Bill,Bob)` `Man(Bob)`	? ⊨ ∀child.Man(Bill)	**DL** answers don‘t know	**PROLOG** answers yes
`≤1 child.⊤(Bill)`	? ⊨ ∀child.Man(Bill)	yes	*now we know everything about Bill‘s children*

Problems of Inference (1)

- Global (In)Consistency of the knowledge base
  - *Does the knowledge base make sense?* $KB \models \bot$?
- Class(in)consistency $C \equiv \bot$?
  - *Must class C be empty?*
- Class inclusion (Subsumption) $C \sqsubseteq D$?
  - *Structuring the knowledge base*
- Class equivalency $C \equiv D$?
  - *Are two classes the same?*

Problems of Inference (1)

- Class disjunctiveness — $C \sqcap D = \bot$?
  - *Are two classes disjunctive?*
- Class membership — C(a)?
  - *Is individual a contained in class C?*
- Instance generation (Retrieval)
  „find all x with C(x)“
  - *Find all (known!) Individuals of class C.*

Description Logics and Inference

- **Decidability:**
  for each inference problem there always exists an algorithm that terminates in finite time
  - **DLs** are fragments of **FOL**, therefore (in principle) **FOL** inference algorithms (e.g. Tableaux) can be applied.
  - But **FOL** algorithms do not always terminate!

- **Problem**: Find algorithms that always terminate!
  - There might be no „naive“ solutions!

Decidability and DL

- **FOL** inference algorithms (Tableaux) must be adapted for **DLs**
- *(for the lecture we will restrict to $\mathcal{ALC}$ tableaux algorithm)*

- Tableaux algorithm shows **unsatisfiability** of a theory (knowledge base)
- We have to adapt the entailment problems to the **detection of contradictions in the knowledge base**, i.e. proof of the unsatisfiability of the knowledge base!

Reduction to Unsatisfiability (1)

- Class inconsistency
  - iff $KB \cup \{C(a)\}$ unsatisfiable (a new)

$C \equiv \bot$

- Class inclusion (Subsumption)
  - iff $KB \cup \{(C \sqcap \neg D)(a)\}$ unsatisfiable (a new)

$C \sqsubseteq D$

- Class equivalency
  - iff $C \sqsubseteq D$ and $D \sqsubseteq C$

$C \equiv D$

Reduction to Unsatisfiability (2)

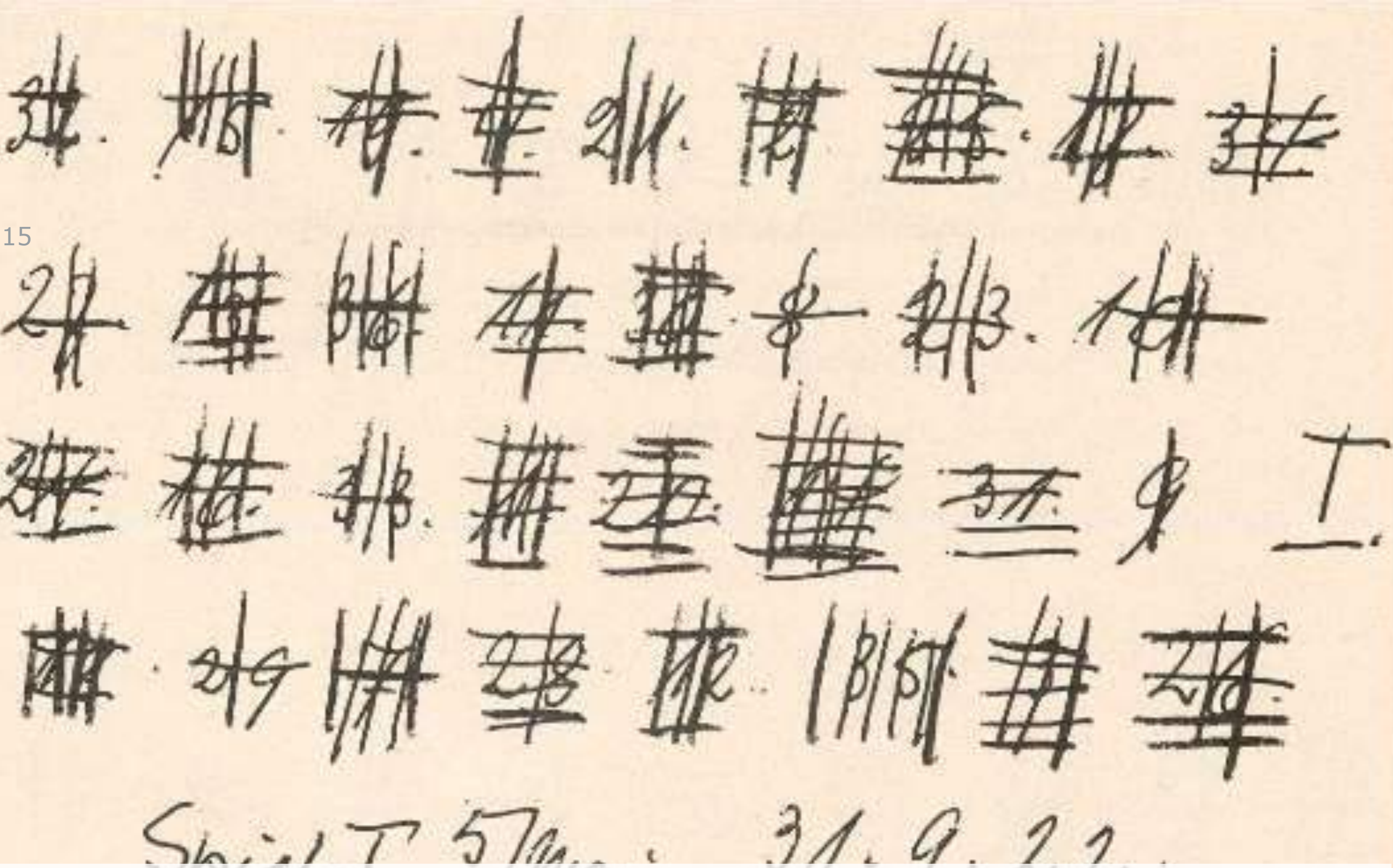
- Class disjunctness
  - iff KB∪{(C⊓D)(a)} unsatisfiable (a new)

$C \sqcap D = \bot$

- Class membership
  - iff KB∪{¬C(a)} unsatisfiable

$C(a)$

- Instance generation (Retrieval) „find all x with C(x)“
  - ***Check class membership for all individuals***



07 Tableaux Algorithm for ALC

Open HPI Course: Knowledge Engineering with Semantic Web Technologies
Lecture 04: Knowledge Representations Pt.1

Knowledge Engineering with Semantic Web Technologies

Lecture 4: Knowledge Representation

06: Tableaux Algorithm for ALC

Dr. Harald Sack
Hasso-Plattner-Institut for IT Systems Engineering
University of Potsdam
Spring 2014

Lecture 4: Knowledge Representations I

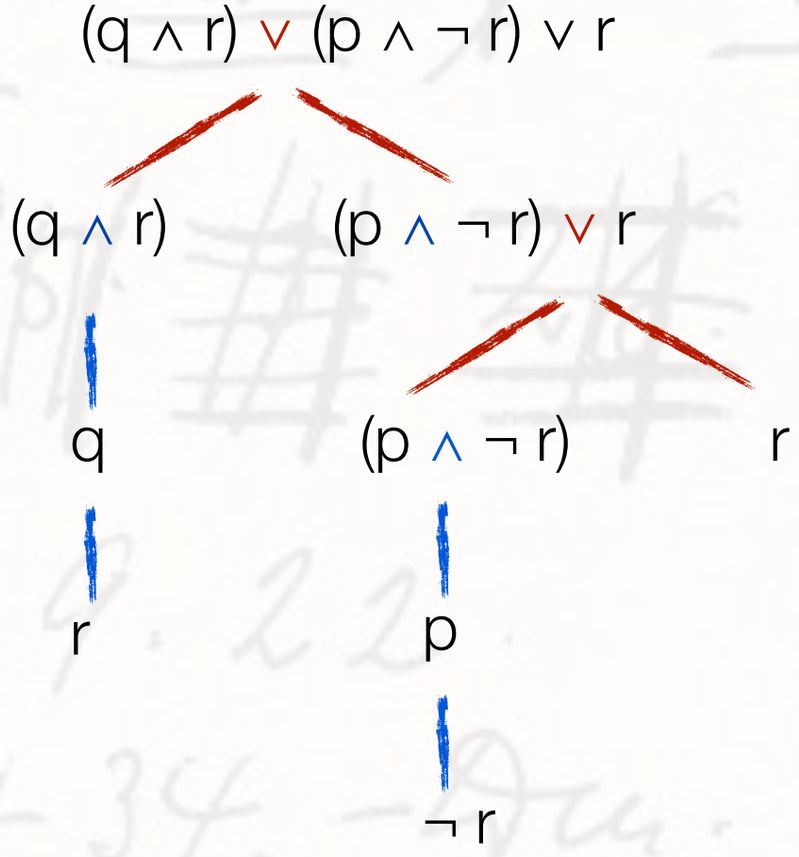
Open HPI Course: Knowledge Engineering with Semantic Web Technologies

07 Tableaux Algorithm for ALC

Open HPI Course: Knowledge Engineering with Semantic Web Technologies
Lecture 04: Knowledge Representations Pt.1

Short Recapitulation

- Proof algorithm to check the consistency of a logical formula by inferring that its negation is a contradiction (***proof by refutation***).
- Construct **Tree**, where each **node** is marked with a logical formula. A **path** from the root to a leaf is the **conjunction** of all formulas represented within the nodes of the path; a **branch** of the path represents a **disjunction**.
- The tree is created by successive application of the **Tableaux Extension Rules**.



Tableaux Algorithm for Description Logics

- Transformation to ***Negation normalform*** **NNF** necessary

- Let $\mathtt{W}$ be a knowledge base,
  - Substitute $\mathtt{C \equiv D}$ by $\mathtt{C \sqsubseteq D}$ and $\mathtt{D \sqsubseteq C}$
  - Substitute $\mathtt{C \sqsubseteq D}$ by $\mathtt{\neg C \sqcup D}$.
  - Apply the NNF Transformations from the next page

- Resulting knowledge base $\mathtt{NNF(W)}$
  - Negation Normalform of W.
  - Negation is placed directly in front of atomic classes.

Tableaux Transformation in Negation Normalform

- NNF Transformations

```
NNF(C) = C,  if  C  is atomic
NNF(¬C) = ¬C,  if  C  is atomic
NNF(¬¬C) = NNF(C)
NNF(C⊔D) = NNF(C) ⊔ NNF(D)
NNF(C⊓D) = NNF(C) ⊓ NNF(D)
NNF(¬(C⊔D)) = NNF(¬C) ⊓ NNF(¬D)
NNF(¬(C⊓D)) = NNF(¬C) ⊔ NNF(¬D)
NNF(∀R.C) = ∀R.NNF(C)
NNF(∃R.C) = ∃R.NNF(C)
NNF(¬∀R.C) = ∃R.NNF(¬C)
NNF(¬∃R.C) = ∀R.NNF(¬C)
```

- `W` and `NNF(W)` are logically equivalent.

Tableaux Transformation in Negation Normalform

- Example: $P \sqsubseteq (E \sqcap U) \sqcup \neg(\neg E \sqcup D)$
- In NNF: $\neg P \sqcup (E \sqcap U) \sqcup (E \sqcap \neg D)$

$C \sqsubseteq D = \neg C \sqcup D$

$\neg(C \sqcup D) = \neg C \sqcap \neg D$

Tableaux Extension Rules for DL

Selection	Action
$C(a) \in \mathcal{W}$ (ABox)	Add $C(a)$
$R(a,b) \in \mathcal{W}$ (ABox)	Add $R(a,b)$
$C \in \mathcal{W}$ (TBox)	Add $C(a)$ for a known Individual a
$(C \sqcap D)(a) \in \mathcal{A}$	Add $C(a)$ and $D(a)$
$(C \sqcup D)(a) \in \mathcal{A}$	Split the path. Add (1) $C(a)$ and (2) $D(a)$
$(\exists R.C)(a) \in \mathcal{A}$	Add $R(a,b)$ and $C(b)$ for a new Individual b
$(\forall R.C)(a) \in \mathcal{A}$	if $R(a,b) \in \mathcal{A}$, then add $C(b)$

- If the resulting tableaux is closed, the original knowledge base is unsatisfiable.
- Only select elements that lead to new elements within the tableaux. If this is not possible, then the algorithm terminates and the original knowledge base is satisfiable.

Tableaux Algorithm (DL) - Example

- P … Professor
- E … Person
- U … University Employee
- D … Student

- Knowledge Base: $P \sqsubseteq (E \sqcap U) \sqcup (E \sqcap \neg D)$
- We want to proof: Is $P \sqsubseteq E$ a logical consequence?

- Knowledge Base (with [*negated*] query) in NNF:
  $\{\neg P \sqcup (E \sqcap U) \sqcup (E \sqcap \neg D),\ (P \sqcap \neg E)(a)\}$

$C \sqsubseteq D = \neg C \sqcup D$

$\neg(C \sqcup D) = \neg C \sqcap \neg D$

Tableaux Algorithm (DL) - Example

- Knowledge Base $\mathcal{W}$:{¬P⊔(E⊓U)⊔(E⊓¬D),(P⊓¬E)(a)}
- Tableaux $\mathcal{A}$:

(1) (P⊓¬E)(a) (from knowledge base)

(2|α from 1) P(a)

(3|α from 1) ¬E(a)

(4) (¬P⊔(E⊓U)⊔(E⊓¬D))(a) (from knowledge base)

(5|β from 4)) ¬P(a) | (6)((E⊓U)⊔(E⊓¬D))(a)

(7|β from 6) (E⊓U)(a) | (8) (E⊓¬D)(a)

(9|α from 7) E(a) (10|α from 8) E(a)

(11|α from 7) U(a) (12|α from 8) ¬D(a)

Tableaux Algorithm (DL) - Problem

- Knowledge Base: ¬Person ⊔ ∃hasParent.Person
- infer: ¬Person(Bill)

{ ¬Person ⊔ ∃hasParent.Person, Person(Bill) }

Person(Bill)

(¬Person ⊔ ∃hasParent.Person)(Bill)

¬Person(Bill)

∃hasParent.Person(Bill) ← ⊔

hasParent(Bill,x_1)

Person(x_1) ← ∃

(¬Person ⊔ ∃hasParent.Person)(x_1)

¬Person(x_1)

∃hasParent.Person(x_1) ← ⊔

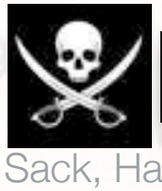
hasParent(x_1,x_2)

Person(x_2) ← ∃

(¬Person ⊔ ∃hasParent.Person)(x_2)

¬Person(x_2)

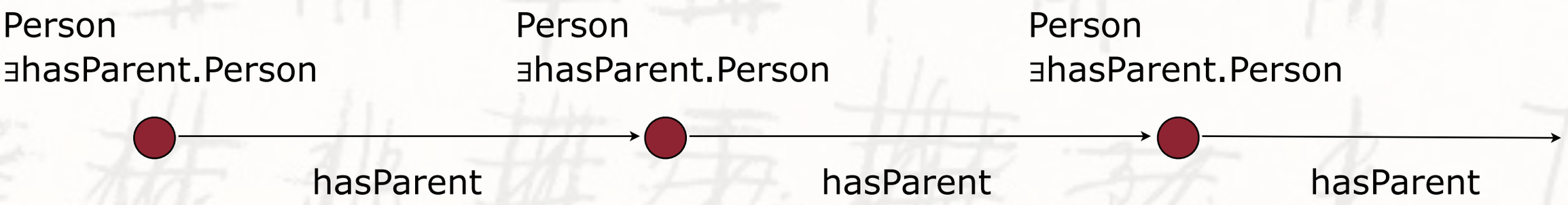
∃hasParent.Person(x_2) ← ⊔

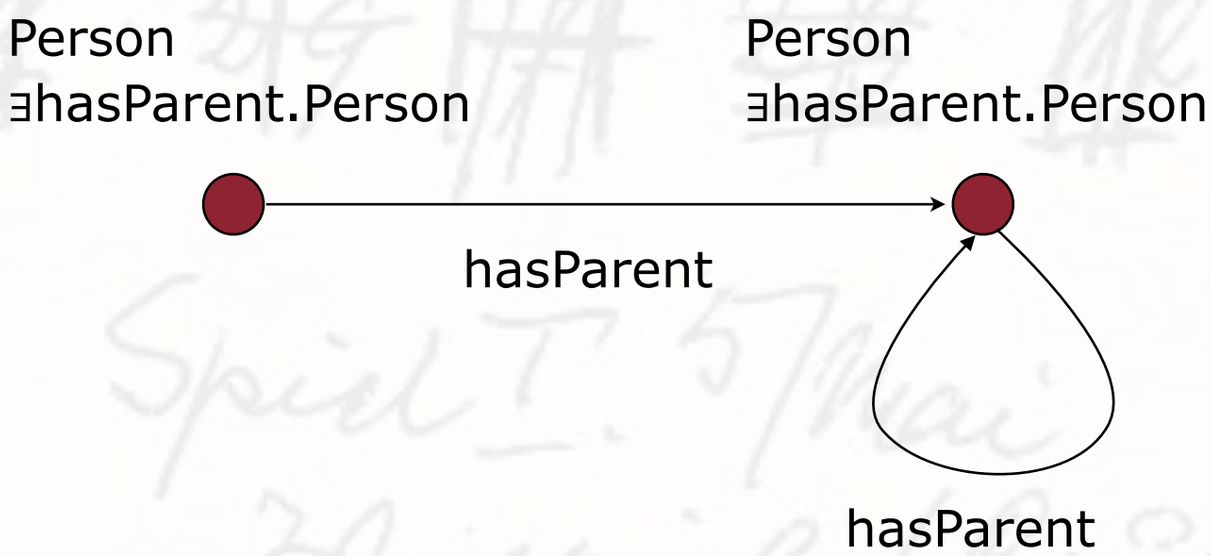


no termination possible

Problem with existential quantification
also for OWL:minCardinality

Idea of Blocking





Blocking

Correctness of this short cut must be proven!

Tableaux Algorithm (DL) with Blocking

- Knowledge Base: ¬Person ⊔ ∃hasParent.Person
- infer: ¬Person(Bill)

Person(Bill)

(¬Person ⊔ ∃hasParent.Person)(Bill)

¬Person(Bill)

STOP

∃hasParent.Person(Bill) ← ⊔

hasParent(Bill,x_1) ← ∃

Person(x_1)

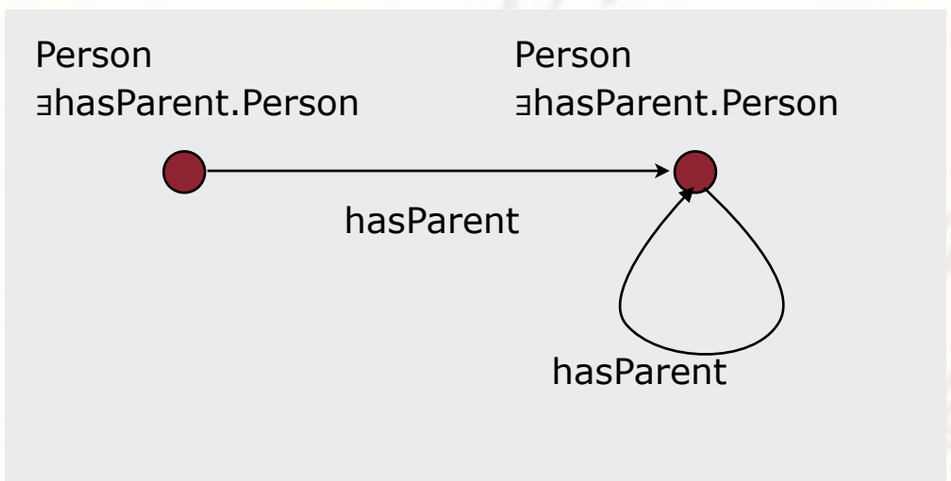
(¬Person ⊔ ∃hasParent.Person)(x_1)

¬Person(x_1)

STOP

∃hasParent.Person(x_1) ← ⊔

STOP termination



Tableaux Algorithm (DL) with Blocking

Person(Bill)

(¬Person ⊔ ∃hasParent.Person)(Bill)

¬Person(Bill)

∃hasParent.Person(Bill) ← ⊔

hasParent(Bill,x_1) ← ∃

Person(x_1)

(¬Person ⊔ ∃hasParent.Person)(x_1)

STOP

¬Person(x_1)

∃hasParent.Person(x_1) ← ⊔

STOP

σ(Individual) = {Classes}

σ is **Class Extension** of an Individual

σ(Bill) = {Person,
¬Person ⊔ ∃hasParent.Person,
∃hasParent.Person}

σ(x_1) = { Person,
¬Person ⊔ ∃hasParent.Person,
∃hasParent.Person}

σ(x_1) ⊆ σ(Bill), therefore Bill blocks x_1

STOP termination

Tableaux Algorithm (DL) with Blocking

- The Selection of $(\exists R.C)(a)$ in the tableaux path A is **blocked**, if there is already an individual b with $\{C|C(a)\in A\}\subseteq\{C|C(b)\in A\}$.

- Two possibilities of termination:

1. Closing the Tableaux.
   Knowledge Base is **unsatisfiable**.

2. All non blocked selections from the tableaux don‘t lead to an extension.
   Knowledge Base is **satisfiable**.

Lecture 5: Knowledge Representations II

Open HPI Course: Knowledge Engineering with Semantic Web Technologies