Knowledge Engineering with Semantic Web Technologies

Lecture 5: Knowledge Representations Part 2

01: Web Ontology Language OWL

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

Lecture 5: Knowledge Representations II

Open HPI Course: Knowledge Engineering with Semantic Web Technologies

01 Web Ontology Language - OWL

Open HPI Course: Knowledge Engineering with Semantic Web Technologies
Lecture 05: Knowledge Representations Pt.2

OWL – Web Ontology Language

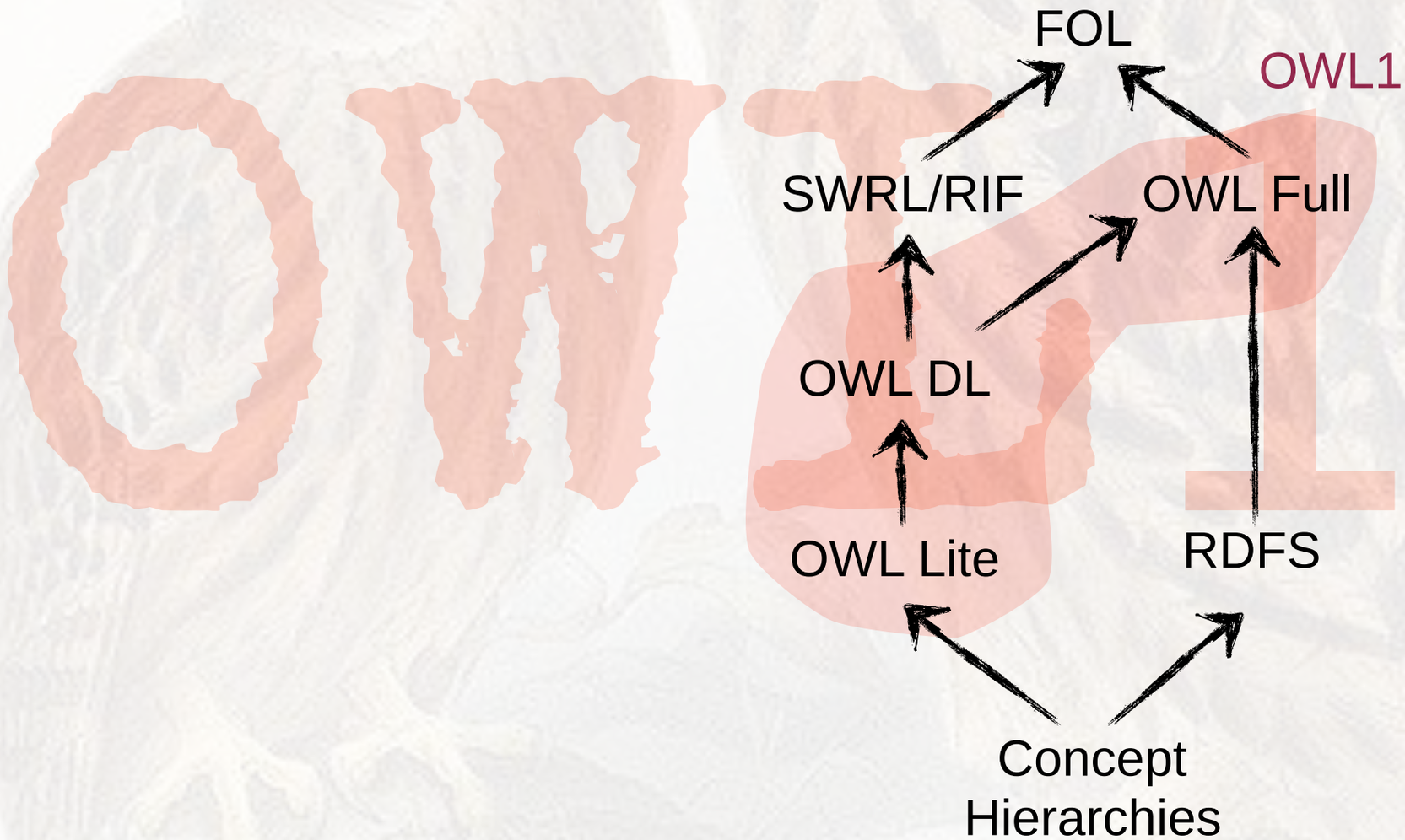
- OWL ($\mathcal{SHOIN}(\mathcal{D})$)- W3C Recommendation since 2004
- OWL 2 ($\mathcal{SHROIQ}(\mathcal{D})$)- W3C Recommendation since 2009

- an OWL Ontology consists of
  - classes / properties / individuals (instances of classes)

- **Open World Assumption**
  - „Absence of information must not be valued as negative information.“
    - E.g.: `sitsNextTo(PersonA, PersonB)`
      PersonA may also sit next to another person…
- **No Unique Name Assumption**
  - „Difference must be expressed explicitly“
    - E.g.: `PersonA` *possibly denotes the same individual as* `PersonB`

OWL – Web Ontology Language

- OWL is a semantic fragment of FOL
- OWL1 exists in different flavors
  - OWL Lite ⊆ OWL1 DL ⊆ OWL1 Full

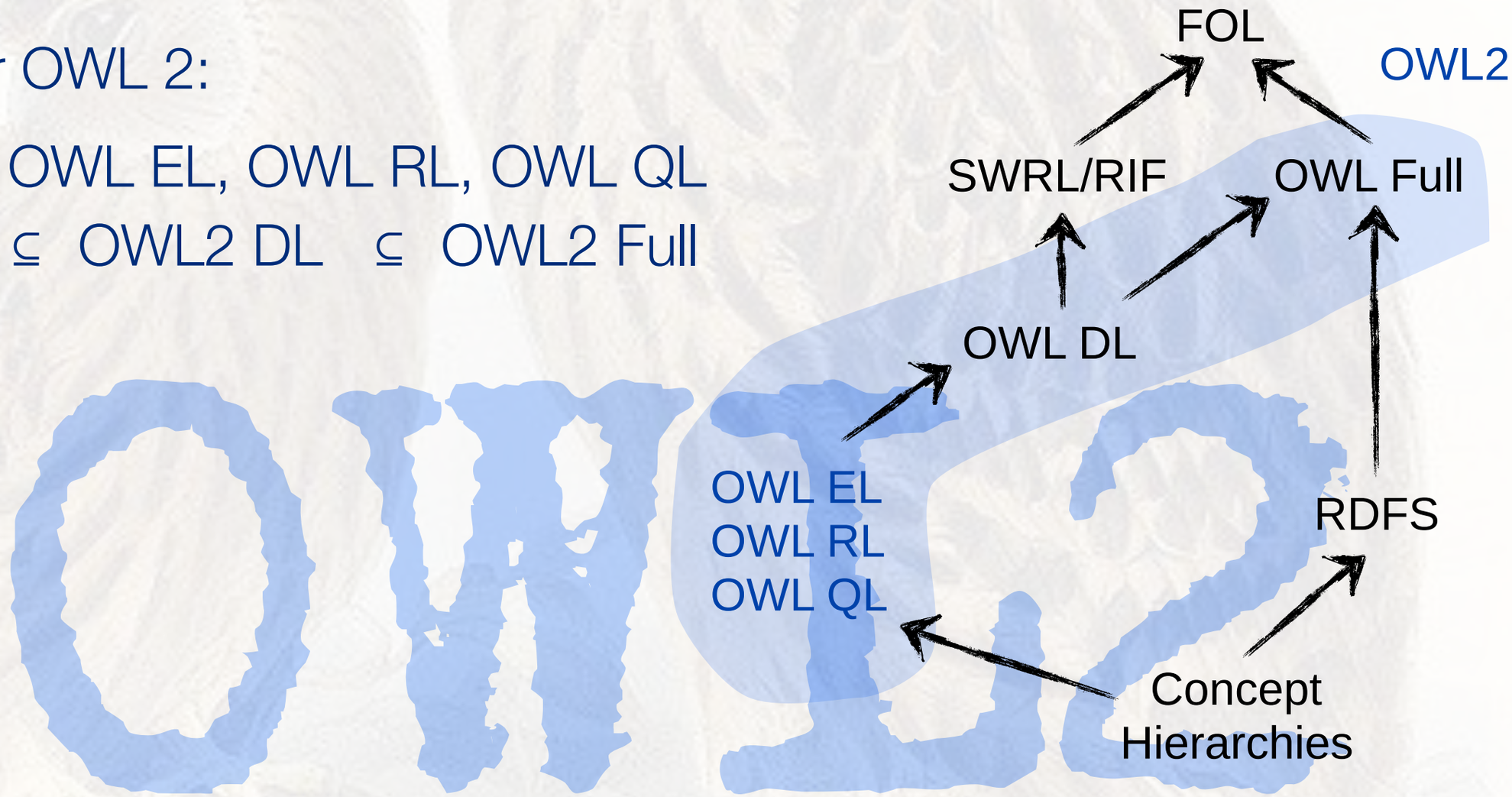


OWL1 DL is based on $\mathcal{SHOIN}(\mathcal{D})$

- **Axioms**
  - **TBox**: subclass relationships `C` $\sqsubseteq$ `D` $(\mathcal{H})$
  - **RBox**: subproperty relationships `R` $\sqsubseteq$ `S` $(\mathcal{H})$,
    inverse properties `R`$^-$ $(\mathcal{I})$, transitivity $\sqsubseteq^+$ $(\mathcal{S})$
  - **ABox**: facts for classes `C(a)`, properties `R(a,b)`,
    equality `a=b`, difference `a`$\neq$`b`
- **Class constructors**:
  - conjunction `C`$\sqcap$`D`, disjunction `C`$\sqcup$`D`, negation ¬`C` of classes
  - property restrictions: universal ∀`R.C` and existential ∃`R.C`
  - number restrictions: ≤`n` `R` and ≥`n` `R` $(\mathcal{N})$
  - closed classes (nominals): `{a}` $(\mathcal{O})$
- Datatypes $(\mathcal{D})$

OWL – Web Ontology Language

- OWL is a semantic fragment of FOL
- OWL1 exists in different flavors
  - OWL Lite ⊆ OWL1 DL ⊆ OWL1 Full
  - for OWL 2:
    - OWL EL, OWL RL, OWL QL ⊆ OWL2 DL ⊆ OWL2 Full



OWL2 DL is based on $\mathcal{SHROIQ}(\mathcal{D})$

Class Expressions

- Class names A,B
- Conjunction C ⊓ D
- Disjunction C ⊔ D
- Negation ¬C
- Exist. property restriction ∃R.C
- Univ. property restriction ∀R.C
- Self ∃S.Self
- Greater-than ≥n S.C
- Less-than ≤n S.C
- Enumerated classes {a}

Properties

- Property names R,S,T
- Simple properties S,T
- Inverse properties R^-
- Universal property U

Tbox (Class axioms)

- Inclusion C ⊑ D
- Equivalence C ≡ D

Rbox (Property Axioms)

- Inclusion $R_1 \sqsubseteq R_2$
- General Inclusion $R^{(-)}_1 \circ R^{(-)}_2 \circ \ldots \circ R^{(-)}_n \sqsubseteq R$
- Transitivity
- Symmetry
- Reflexivity
- Irreflexivity
- Disjunctiveness

Abox (Facts)

- Class membership C(a)
- Property relation R(a,b)
- Negated property relation ¬S(a,b)
- Equality a=b
- Inequality a≠b

OWL Syntax Variants

- OWL2 can be represented in different Syntax variants
  - **Functional Syntax**: substitutes abstract syntax of OWL1
  - **RDF-Syntax**: extension of existing OWL/RDF
  - **XML-Syntax**: independent XML serialisation
  - **Manchester-Syntax**: machine readable syntax, esp. for ontology editors (e.g. Protégé)
  - **Turtle**: optional

- Functional Syntax is easy to define, no RDF restrictions, more compact
- RDF-Syntax important for compatibility issues
- **Turtle: simple and efficient to write...**

OWL2 - Turtle Syntax - Example

$$\text{HappyPerson} \equiv \forall \text{hasChild}.\text{HappyPerson} \sqcap \exists \text{hasChild}.\text{HappyPerson}$$

```
:HappyPerson a owl:Class ;
    owl:equivalentClass  [
    a owl:Class ;
    owl:intersectionOf ([ a owl:Restriction ;
                          owl:onProperty :hasChild ;
                          owl:allValuesFrom :HappyPerson  ]
                        [ a owl:Restriction ;
                          owl:onProperty :hasChild ;
                          owl:someValuesFrom :HappyPerson ]
                        )
     ] .
```

OWL2 - RDF/XML Syntax - Example

$$HappyPerson \equiv \forall hasChild.HappyPerson \sqcap \exists hasChild.HappyPerson$$

```
<?xml version="1.0"?>
<rdf:RDF xmlns=...>
<owl:ObjectProperty rdf:about="http://example.org/turtle#hasChild"/>

<owl:Class rdf:about="http://example.org/turtle#HappyPerson">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:onProperty rdf:resource="http://example.org/turtle#hasChild"/>
          <owl:someValuesFrom rdf:resource="http://example.org/turtle#HappyPerson"/>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty rdf:resource="http://example.org/turtle#hasChild"/>
          <owl:allValuesFrom rdf:resource="http://example.org/turtle#HappyPerson"/>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
</rdf:RDF>
```

Standard Namespaces and Prefixes

Prefix name	Prefix IRI
rdf:	*<http://www.w3.org/1999/02/22-rdf-syntax-ns#>*
rdfs:	*<http://www.w3.org/2000/01/rdf-schema#>*
xsd:	*<http://www.w3.org/2001/XMLSchema#>*
owl:	*<http://www.w3.org/2002/07/owl#>*

(cf. http://www.w3.org/TR/owl2-syntax/)

http://prefix.cc/

02 Classes, Instances, and Relations in OWL

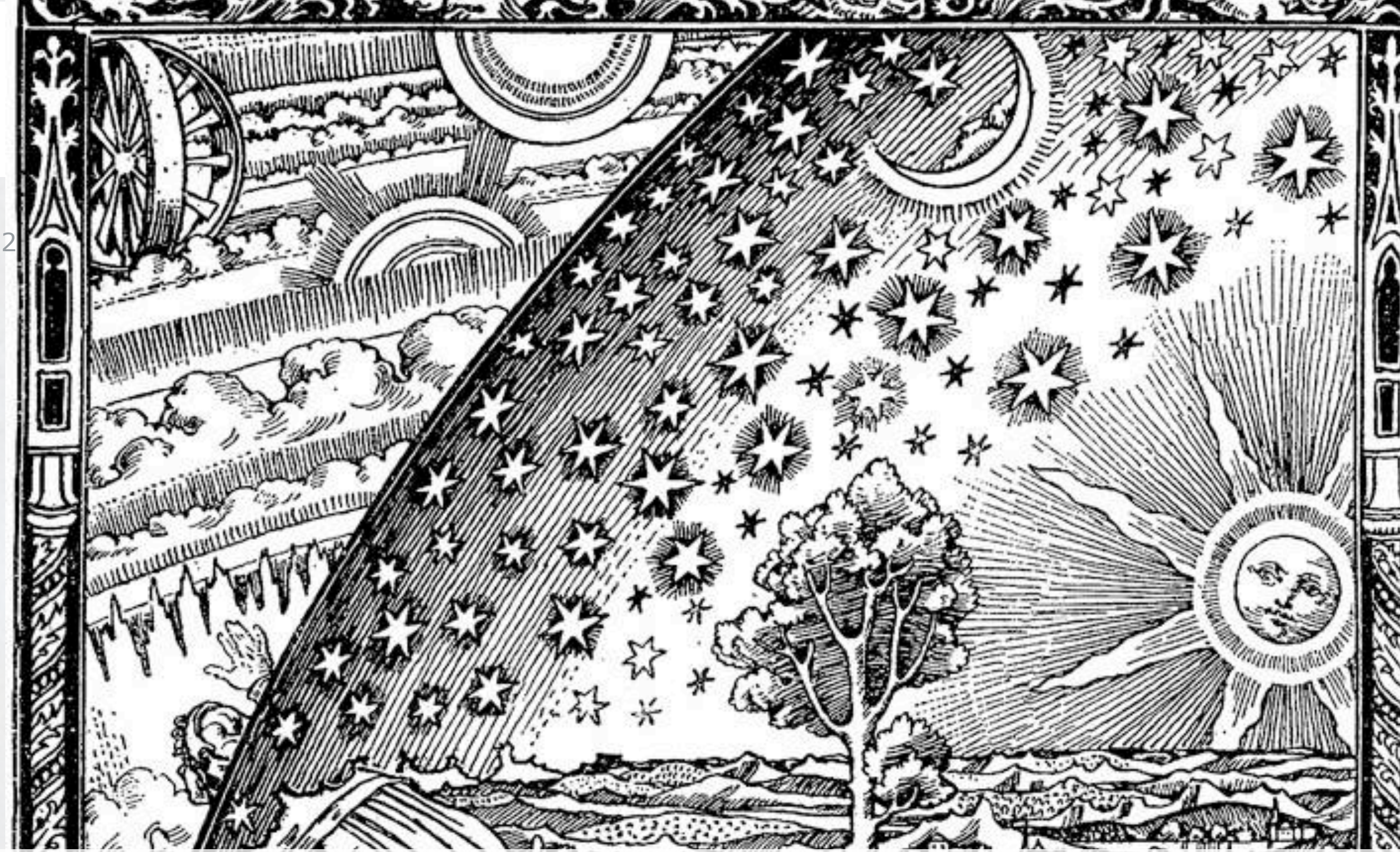
Open HPI Course: Knowledge Engineering with Semantic Web Technologies
Lecture 05: Knowledge Representations Pt.2

Knowledge Engineering with Semantic Web Technologies

Lecture 5: Knowledge Representations Part 2

02: Classes, Instances, and Properties in OWL

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



Lecture 5: Knowledge Representations II

Open HPI Course: Knowledge Engineering with Semantic Web Technologies

02 Classes, Instances, and Properties in OWL

Open HPI Course: Knowledge Engineering with Semantic Web Technologies
Lecture 05: Knowledge Representations Pt.2

OWL – Classes, Properties and Individuals

- Ontology axioms consist of the following three building blocks:
  - **Classes**
    - comparable with classes in RDFS
  - **Individuals**
    - comparable with objects in RDFS
  - **Properties**
    - comparable with properties in RDFS

OWL – Classes

- there exist two predefined classes
  - `owl:Thing` (class that contains all individuals) $\top$
  - `owl:Nothing` (empty class) $\bot$

- Definition of a class

```
:Book a owl:Class .
```

OWL – Individuals

- Definition of individuals via class membership

```
:NineteenEightyfour a :Book .
```

```
Book(NineteenEightyfour)
```

- Individuals can also be defined without direct class membership as **named entity**

```
:HaraldSack a owl:NamedIndividual .
```

OWL – Properties

- there exist two property variants:
  - object properties
  - datatype properties

- **Object properties** are defined like classes

```
:author  a owl:ObjectProperty .
```

- Domain and Range of object properties

```
:author  a owl:ObjectProperty ;
           rdfs:domain :Book ;
           rdfs:range  :Writer .
```

OWL – ObjectProperties

- Domain and Range of object properties

```
:author  a owl:ObjectProperty ;
           rdfs:domain :Book ;
           rdfs:range  :Writer .
```

- in DL domain and range can be defined in the following way:
  - **Domain**: $\exists author.\top \sqsubseteq Book$
  - **Range**: $\top \sqsubseteq \forall author.Writer$

OWL – DatatypeProperties

- Datatype properties have datatypes as range

```
:publicationYear  a owl:DatatypeProperty .
```

- Domain and range of datatype properties

```
:publicationYear   a owl:DatatypeProperty ;
            rdfs:domain :Book ;
            rdfs:range  xsd:integer .
```

- Many XML datatypes can be used.

$$\exists \text{publicationYear}.\top \sqsubseteq \text{Book}$$

$$\top \sqsubseteq \forall \text{publicationYear}.\text{Integer}$$

OWL – Properties and Individuals


```
:Book a owl:Class .
:Writer a owl:Class .
:GeorgeOrwell a Writer .
:author  a owl:ObjectProperty ;
            rdfs:domain :Book ;
            rdfs:range  :Writer .
:publicationYear  a owl:DatatypeProperty ;
            rdfs:domain :Book ;
            rdfs:range  xsd:integer .
:NineteenEightyFour a :Book ;
            :author :GeorgeOrwell ;
            :publicationYear 1948 .
```

OWL – Properties and Individuals

- Properties in general are not functional

```
:Genre a owl:Class .

:PoliticalFiction a :Genre .
:ScienceFiction a :Genre .
:DystopianFiction a :Genre .


:literaryGenre a owl:ObjectProperty ;
               rdfs:domain :Book ;
               rdfs:range  :Genre .

:NineteenEightyFour a :Book ;
                    :literaryGenre :PoliticalFiction ;
                    :literaryGenre :ScienceFiction ;
                    :literaryGenre :DystopianFiction .
```

OWL – Class Hierarchies


```
:Novel a owl:Class ;
        rdfs:subClassOf :Book .
:Book a owl:Class ;
        rdfs:subClassOf :Work .
:Work a owl:Class .
```

$$\texttt{Novel} \sqsubseteq \texttt{Book}$$
$$\texttt{Book} \sqsubseteq \texttt{Work}$$

- via inference it can be entailed that „`Novel`“ is also a subclass of „`Work`“

OWL – Class Hierarchies and Disjunctiveness

```
:Book a owl:Class .
:Writer a owl:Class .
:Novel a owl:Class ;
        rdfs:subClassOf :Book .
:Poet a owl:Class ;
        rdfs:subClassOf :Writer .

:Book owl:disjointWith :Writer .
```

$$\texttt{Novel} \sqsubseteq \texttt{Book}$$
$$\texttt{Poet} \sqsubseteq \texttt{Writer}$$
$$\texttt{Book} \sqcap \texttt{Writer} \sqsubseteq \bot$$

- via inference it can be entailed that „`Novel`“ and „`Poet`“ are also disjoint classes.

OWL – Class Hierarchies and Disjunctiveness

- OWL provides a **shortcut** to define several classes to be disjunctive

```
[] a owl:AllDisjointClasses ;
     owl:members
     ( :Book
       :Writer
       :Vegetable
       :Furniture
       :Car ) .
```

OWL – Class Hierarchies and Equivalence

```
15  :Writer a owl:Class .
    :Author a owl:Class .
    :Poet a owl:Class ;
            rdfs:subClassOf :Writer .
    :Writer owl:equivalentClass :Author .
```

$$\texttt{Poet} \sqsubseteq \texttt{Writer}$$
$$\texttt{Writer} \equiv \texttt{Author}$$

- via inference it can be entailed that „`Poet`“ is also an „`Author`“

OWL – Individuals


```
:NineteenEightyfour a :Novel ;
                    :author :GeorgeOrwell ;
                    :publicationYear 1948 ;
                    owl:sameAs :ARX012345 .
:Novel a owl:Class ;
       rdfs:subClassOf :Book .
:Book a owl:Class.
```

- via inference it can be entailed that „`ARX012345`“ is a „`Book`“
- Difference of Individuals via **`owl:differentFrom`**

```
:ARX012345 a :Novel ;
           owl:differentFrom :ARX012346 .
```

OWL – Individuals

- OWL provides a shortcut to define several individuals to be different

```
[] a owl:AllDifferent ;
     owl:distinctMembers
          (:NineteenEightyfour
           :AnimalFarm
           :BraveNewWorld
           :Simulacron5) .
```

03 OWL Complex Classes and Properties

Open HPI - Course: Semantic Web Technologies - Lecture 5: Knowledge Representations II

Knowledge Engineering with Semantic Web Technologies

Lecture 5: Knowledge Representations Part 2

03: OWL Complex Classes

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

Lecture 5: Knowledge Representations II

Open HPI Course: Knowledge Engineering with Semantic Web Technologies

03 OWL Complex Classes

Open HPI - Course: Knowledge Engineering with Semantic Web Technologies
Lecture 5: Knowledge Representations II

OWL – Closed Classes (Nominals)


```
:Novel a owl:Class ;
:AnimalFarm a :Novel .
:NineteenEightyfour a :Novel .
:NovelsInStore a owl:Class ;
                owl:oneOf
               (:AnimalFarm
                :NineteenEightyfour) .
```

$$NovelsInStore \sqsubseteq \{AnimalFarm, NineteenEightyfour\}$$

- This says that there are only two novels available in the store.

OWL – Logical Class Constructors

- logical AND (conjunction): `owl:intersectionOf` ⊓
- logical OR (disjunction): `owl:unionOf` ⊔
- logical negation: `owl:complementOf` ¬
- Logical constructors are applied to create complex class descriptions from atomic classes.

OWL – Logical Class Constructors

$$BooksInStore \equiv ThingsInStore \sqcap Books$$

```
:Book a owl:Class .
:ThingsInStore a owl:Class .
:BooksInStore a owl:Class ;
      owl:intersectionOf (:Book :ThingsInStore) .
```

- The class “`BooksInStore`“ results from the intersection of all individuals of the classes “`ThingsInStore`“ and “`Book`“

OWL – Logical Class Constructors

$$Book \equiv Novel \sqcup Poetry \sqcup Nonfiction$$

```
:Book a owl:Class ;
      owl:equivalentClass [
         owl:unionOf ( :Novel
                       :Poetry
                       :NonFiction )
] .
```

- Novels, poetry, and non-fiction are also books

OWL – Logical Class Constructors

$$\text{Book} \sqsubseteq \neg \text{Writer}$$

```
:Book a owl:Class ;
      rdfs:subClassOf [
      owl:complementOf :Writer
] .
```

- semantically equivalent assertion:

```
:Book a owl:Class ;
      owl:disjointWith :Writer .
```

OWL – Property Restrictions

- are used to describe complex classes via properties

- restrictions on values:
  - `owl:hasValue`
  - `owl:allValuesFrom`
  - `owl:someValuesFrom`

- restrictions on cardinality:
  - `owl:cardinality`
  - `owl:minCardinality`
  - `owl:maxCardinality`

OWL – Property Restrictions with Constants

$$\mathtt{OrwellsBooks} \sqsubseteq \mathtt{author}.(\mathtt{GeorgeOrwell})$$


```
:OrwellsBooks a owl:Class ;
                rdfs:subClassOf
              [ a owl:Restriction ;
                owl:onProperty :author ;
                owl:hasValue :GeorgeOrwell
 ] .
```

- Class „`OrwellsBooks`“ is described via fixed value assignment (=constant) of the individual „`GeorgeOrwell`“ to the property „`author`“

```
:Writer a owl:Class .
:Book a owl:Class .

:GeorgeOrwell a :Person .
:author a owl:ObjectProperty ;
        rdfs:domain :Book ;
        rdfs:range :Writer .
```

OWL – Property Restrictions

- `owl:allValuesFrom`
  fixes all instances of a specific class `C` as allowed range for a property `P` → (Universal Quantification) `∀P.C`

`Poetry ⊑ ∀author.Poet`

```
:Poetry a owl:Class ;
        rdfs:subClassOf
      [ a owl:Restriction ;
        owl:onProperty :author ;
        owl:allValuesFrom :Poet
] .
```

```
:Poet a owl:Class .
      rdfs:subClassOf :Writer .
:Poetry a owl:Class
      rdfs:subClassOf :Book .
:author a owl:ObjectProperty ;
        rdfs:domain :Book ;
        rdfs:range :Writer .
```

OWL – Property Restrictions

- `owl:someValuesFrom`
  describes that there must exist an individual for property `P` and fixes its range to class `C`
  → (existential quantification) `∃P.C`

`Reader ⊑ ∃reads.Book`

```
:Reader a owl:Class ;
        rdfs:subClassOf
           [ a owl:Restriction ;
            owl:onProperty :reads ;
            owl:someValuesFrom :Book
           ] .
```

```
:Reader a owl:Class .
:Book a owl:Class .

:reads a owl:ObjectProperty ;
       rdfs:domain :Person ;
       rdfs:range  :ReadingMaterial .
```

OWL – Cardinality Restrictions

- `owl:cardinality` restricts to an exact number
- `owl:minCardinality, owl:maxCardinality` restricts to upper / lower bounds

`Tetralogy ⊑ (=4)hasVolumes`

```
:Tetralogy a owl:Class ;
        rdfs:subClassOf
        [ a owl:Restriction ;
          owl:onProperty  :hasVolumes ;
          owl:cardinality 4
        ] .
```

```
:hasVolumes a owl:DatatypeProperty ;
        rdfs:domain :Thing ;
        rdfs:range  xsd:integer .
```

04 OWL Property Restrictions

Open HPI - Course: Knowledge Engineering with Semantic Web Technologies
Lecture 5: Knowledge Representations II

Knowledge Engineering with Semantic Web Technologies

Lecture 5: Knowledge Representations Part 2

04: OWL Property Restrictions

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

Lecture 5: Knowledge Representations II

Open HPI Course: Knowledge Engineering with Semantic Web Technologies

04 OWL Property Restrictions

Open HPI - Course: Knowledge Engineering with Semantic Web Technologies
Lecture 5: Knowledge Representations II

OWL – Property Relationships

- Property hierarchies can be created via specializations: `rdfs:subPropertyOf`
- Inverse properties are defined via `owl:inverseOf`
- Identical properties are defined via `owl:equivalentProperty`

```
:isMadeOf a owl:ObjectProperty ;
          rdfs:subPropertyOf  :consistsOf .


:reads a owl:ObjectProperty ;
       owl:inverseOf  :isReadBy .
```

$$isMadeOf \sqsubseteq consistsOf$$

$$reads^{-} \equiv isReadBy$$

OWL – Property Relationships

- `owl:TransitiveProperty`
  - E.g.: if `isPartOf(a,b)` and `isPartOf(b,c)` then it holds that `isPartOf(a,c)`

- `owl:SymmetricProperty`
  - E.g.: if `isNeighborOf(a,b)`, then it holds that `isNeighborOf(b,a)`

- `owl:FunctionalProperty`
  - E.g.: if `hasMother(a,b)` and `hasMother(a,c)`, then it holds that `b=c`

- `owl:InverseFunctionalProperty`
  - E.g.: if `isMotherOf(b,a)` and `isMotherOf(c,a)` then it holds `that b=c`

OWL – Transitive Properties


```
:isPublishedBefore a owl:ObjectProperty ;
        a owl:TransitiveProperty ;
        rdfs:domain owl:Book ;
        rdfs:range owl:Book .

:AnimalFarm a owl:Book ;
        :isPublishedBefore :NineteenEightyfour .

:BraveNewWorld a :Book ;
          : isPublishedBefore  :AnimalFarm .
```

- via inference it can be entailed that „`BraveNewWorld`“ has also been published before „`NineteenEightyfour`“ .

OWL – Property Relationships

- In addition there are also
  - **Asymmetric** properties via **`owl:AsymmetricProperty`**
    - E.g.: if it holds that `isLeftOf(a,b)` then it is not possible that also `isLeftOf(b,a)`
  - **Reflexive** properties via **`owl:ReflexiveProperty`**
    - E.g.: `isRelatedTo(x,x)`
  - **Irreflexive** properties via **`owl:IrreflexiveProperty`**
    - E.g.: if `isParentOf(x,y)` then `x≠y`

      $\top \sqsubseteq \neg \exists R.Self$

OWL Properties and Relations

- OWL offers the Definition of **disjunctive properties**
- Two properties R and S are **disjunctive**, if two individuals x,y are never related via both properties

```
:hasParent a owl:ObjectProperty ;
   owl:propertyDisjointWith :hasChild .
```

- Shortcut for several disjunctive properties

```
[] rdf:type owl:AllDisjointProperties ;
   owl:members ( :hasParent :hasChild :hasGrandchild ) .
```

OWL Properties and Relations

- OWL defines a **universal and empty property** as counterparts for top and bottom classes:
  - **`owl:topObjectProperty`**
    relates each potential pair of individuals, superclass for object properties
  - **`owl:bottomObjectProperty`**
    connects no individuals, subclass of all object properties
  - **`owl:topDatatypeProperty`**
    relates all individuals with all typed literals, superclass for all datatype properties
  - **`owl:bottomDatatypeProperty`**
    connects no individual and no typed literal, subclass of all datatype properties

OWL 2 Qualified Number Restriction

- OWL enables class constructors with **number restrictions on properties connected with a range constraint**
- E.g.: SuccessfulAuthor ⊑ ≥1 notableWork.Bestseller

```
:SuccessfulAuthor a owl:Class;
   rdfs:subClassOf [
      a owl:Restriction ;
      owl:onProperty :notableWork ;
      owl:minQualifiedCardinality  “1“^^xsd:nonNegativeInteger;
      owl:onClass :Bestseller ] .
```

- owl:maxQualifiedCardinality, owl:minQualifiedCardinality, owl:qualifiedCardinality

OWL Reflexive Property Restriction

- OWL enables the definition of classes that contain individuals that are **related to themselves** for specific properties

```
:Philosopher a owl:Class ;
    rdfs:subClassOf [
        a owl:Restriction ;
        owl:onProperty :knows ;
        owl:hasSelf “true“^^xsd:boolean ] .
```

Philosopher ⊑ knows.self

OWL Negated Property Instantiation

- **OWL** also enables to express that two individuals are **NOT related via a given property**
- E.g..: ¬isBrother(GeorgeOrwell,AldousHuxley)

```
[] rdf:type owl:negativePropertyAssertion ;
   owl:sourceIndividual :GeorgeOrwell ;
   owl:assertionProperty :isBrother ;
   owl:targetIndividual :AldousHuxley .
```

05 OWL General Role Inclusion

Open HPI - Course: Knowledge Engineering with Semantic Web Technologies
Lecture 5: Knowledge Representations II

Knowledge Engineering with Semantic Web Technologies

Lecture 5: Knowledge Representations Part 2

05: OWL General Role Inclusion

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

Lecture 5: Knowledge Representations II

Open HPI Course: Knowledge Engineering with Semantic Web Technologies

05 OWL General Role Inclusion

Open HPI - Course: Knowledge Engineering with Semantic Web Technologies
Lecture 5: Knowledge Representations II

General Role / Property Inclusion

- Complex Roles / Properties can be constructed from simple roles / properties (R-Box)
- *„The friends of my friends are also my friends.“*
  - can be expressed as **transitive property**
- **But**: *„**The foes of my friends are also my foes.**“*
  - cannot be expressed in that way

- In FOL expressed as a rule (axiom):
  - `∀x,y,z:hasFriend(x,y)∧hasFoe(y,z) → hasFriendsFoe(x,z)`

General Role / Property Inclusion

- $\mathcal{SHROIQ}(\mathcal{D})$ enables the construction of arbitrary complex roles via General Role/Property Inclusion

General Property Inclusion

- R-Box expressions of the form $R_1 \circ R_2 \circ R_3 \circ \ldots \circ R_n \sqsubseteq S$
  `e.g.: hasFriend ∘ hasFoe ⊑ hasFriendsFoe`
- Semantics: if $(x_0, x_1) \in R_1^I, (x_1, x_2) \in R_2^I \ldots (x_{n-1}, x_n) \in R_n^I$, then it also holds that $(x_0, x_n) \in S^I$

E.g.: $(x_0, x_1) \in \texttt{hasFriend}^I$ and $(x_1, x_2) \in \texttt{hasFoe}^I$, then it also holds $(x_0, x_2) \in \texttt{hasFriendsFoe}^I$

BUT: OWL with general property inclusion in UNDECIDABLE !

Restriction to Regular RBoxes

- Can property inclusion be restricted in some way to stay decidable?
  - Rboxes are like *grammars for context-free languages*
  - Intersection of context free languages is problematic (undecidable)
  - Therefore **restriction to regular languages!**

Regular RBoxes

- **Property names** are ordered with $\prec$ **(*strict partial Ordering*)**.
- Each **RBox Inclusion** must must be formed like:
  - $R \circ R \sqsubseteq R$
  - $R^- \sqsubseteq R$
  - $S_1 \circ S_2 \circ S_3 \circ \ldots \circ S_n \sqsubseteq R$
  - $R \circ S_1 \circ S_2 \circ S_3 \circ \ldots \circ S_n \sqsubseteq R$
  - $S_1 \circ S_2 \circ S_3 \circ \ldots \circ S_n \circ R \sqsubseteq R$
- Where: $\mathbf{S_i \prec R}$ for all i=1,2,...,n
- An RBox is called **regular**, if such an (strict) Ordering $\prec$ exists.

Regular RBoxes - Examples

- Example:

  $R \circ S \sqsubseteq R \qquad S \circ S \sqsubseteq S \qquad R \circ S \circ R \sqsubseteq T$

  is **regular** with ordering $S < R < T$

- Example:

  $R \circ T \circ S \sqsubseteq T$

  is **not regular** (*not allowed inclusion*)

- Example:

  $R \circ S \sqsubseteq S \qquad S \circ R \sqsubseteq R$

  is not regular (*no valid ordering possible*)

Regular RBoxes - Examples (2)

- Example:
  $\texttt{hasParent} \circ \texttt{hasHusband} \sqsubseteq \texttt{hasFather}$
  $\texttt{hasFather} \sqsubseteq \texttt{hasParent}$

- is **not regular** because regularity would enforce
  both
  $\texttt{hasParent} < \texttt{hasFather}$
  and
  $\texttt{hasFather} < \texttt{hasParent}$
  which is impossible because $<$ must be strict

Restrictions for simple Properties

- Simple properties in $\mathcal{SHOIN}(\mathcal{D})$ are properties without transitive subproperties
- In $\mathcal{SHROIQ}(\mathcal{D})$ general property inclusion has to be considered

Simple Properties

- are all properties that
  - **are not on the right side** of a property inclusion,
  - **are the inverse of** other simple properties,
  - are **only on the right side** of property inclusions $R \sqsubseteq S$, where **on the left side is a simple property**

- Non-simple properties are properties that are directly (or indirectly) dependent of property chains ($\circ$)

Restrictions for simple Properties

- The following expressions are permitted **ONLY for simple properties**:
  - `≤n R.C` and `≥n R.C` (qualified number restriction)
  - Irreflexive properties
  - Disjunctive properties
  - `∃R.Self`
  - `¬R(a,b)`
- Reason: Saving **Decidability**

General Role / Property Inclusion

Summary

- To ensure **decidability** the following **structural restrictions** must hold or $\mathcal{SHROIQ}(\mathcal{D})$:
  - **Regularity**:
    Restriction of potential interaction of RBox axioms
  - **Simplicity** of properties:
    Restrictions of how to apply properties in number restrictions

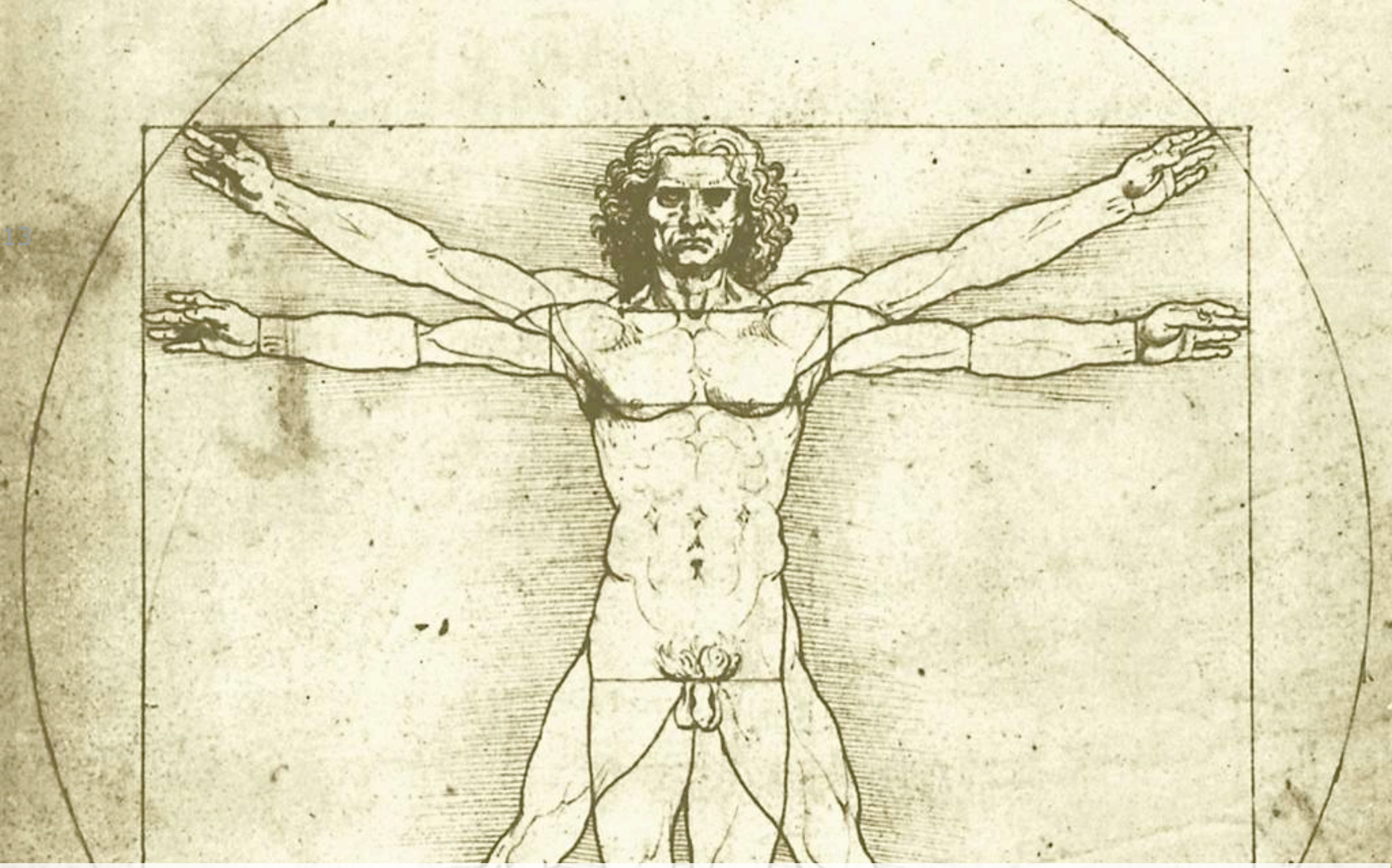
- Therefore a number of restrictions arise for the overall structure of the knowledge base that have to be considered for all axioms.

OWL 2 General Property Inclusion

- OWL 2 enables **property chaining (general property inclusion)**

```
:hasFriendsFoe  a owl:ObjectProperty ;
   owl:PropertyChainAxiom  ( :hasFriend :hasFoe ) .
```

- Not allowed for datatype properties



06 Rules - There's more than OWL

Open HPI - Course: Knowledge Engineering with Semantic Web Technologies
Lecture 5: Knowledge Representations II

Knowledge Engineering with Semantic Web Technologies

Lecture 5: Knowledge Representations Part 2

06: Rules - There‘s more than OWL

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

Lecture 5: Knowledge Representations II

Open HPI Course: Knowledge Engineering with Semantic Web Technologies

short recap:

OWL

OWL2

SHOIN(D)

SHROIQ(D)

OWL Ontologies

complex classes

property restrictions

general property inclusion

OWL profiles

Extension

OWL 2 Overview

Class Expressions

- Class names A,B
- Conjunction C ⊓ D
- Disjunction C ⊔ D
- Negation ¬C
- Exist. property restriction ∃R.C
- Univ property restriction ∀R.C
- Self ∃S.Self
- Greater-than ≥n S.C
- Less-than ≤n S.C
- Enumerated classes {a}

Properties

- Property names R,S,T
- Simple properties S,T
- Inverse properties R^-
- Universal property U

Tbox (Class axioms)

- Inclusion C ⊑ D
- Equivalence C ≡ D

Rbox (Property Axioms)

- Inclusion $R_1 \sqsubseteq R_2$
- General Inclusion $R^{(-)}_1 \circ R^{(-)}_2 \circ \ldots \circ R^{(-)}_n \sqsubseteq R$
- Transitivity
- Symmetry
- Reflexivity
- Irreflexivity
- Disjunctiveness

Abox (Facts)

- Class membership C(a)
- Property relation R(a,b)
- Negated property relation ¬S(a,b)
- Equality a=b
- Inequality a≠b

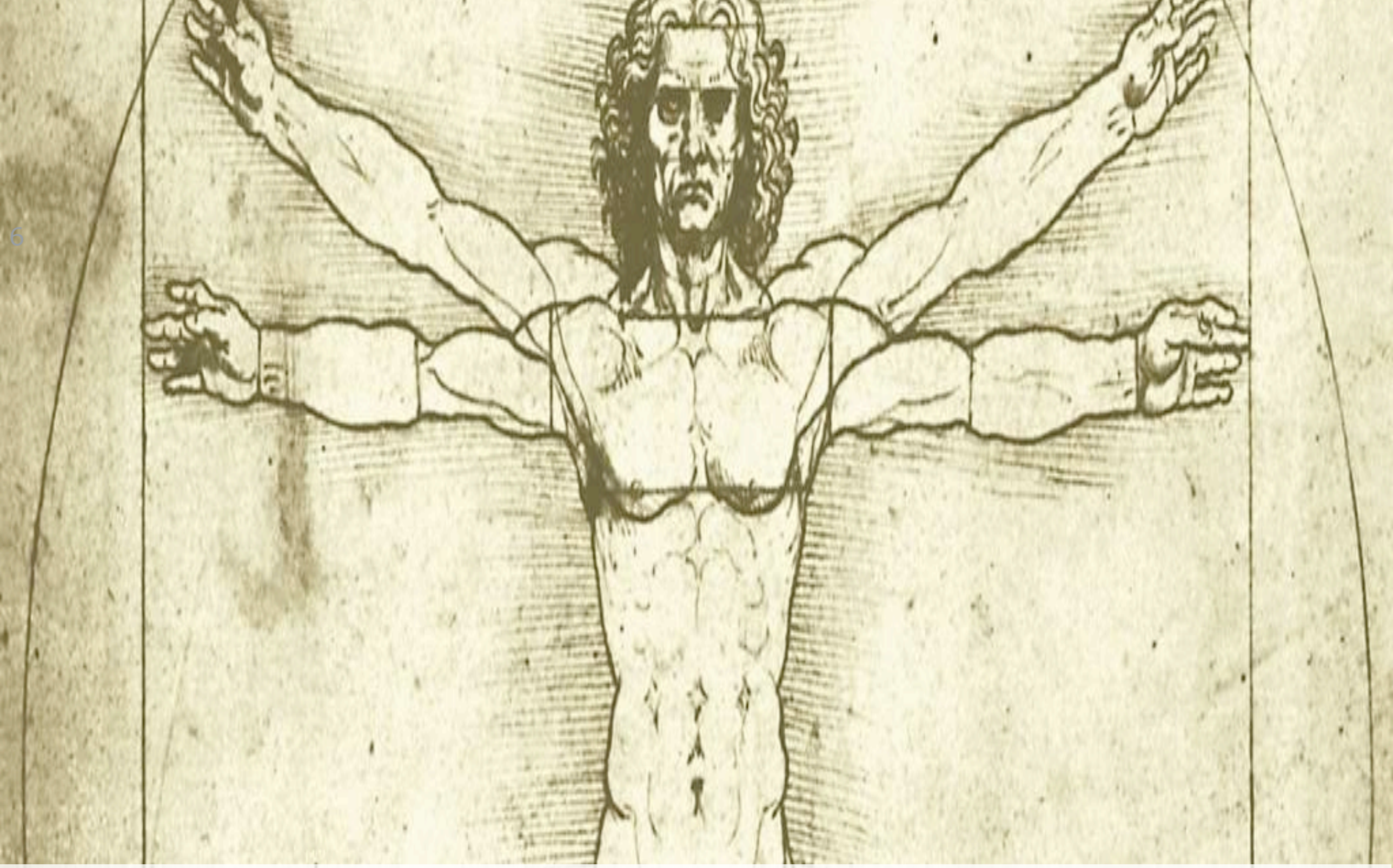
OWL – Web Ontology Language

- OWL is a semantic fragment of FOL
- OWL exists in different flavors
  - OWL2 EL, OWL2 RL, OWL2 QL ⊆ OWL2 DL ⊆ OWL2 Full

OWL2 Full		*undecidable*
OWL2 DL	$\mathcal{SHROIQ}(\mathcal{D})$	*2NExpTimeComplete*
OWL1 DL	$\mathcal{SHOIN}(\mathcal{D})$	*NExpTimeComplete*
OWL2 RL OWL2 EL	$\mathcal{EL}$++	*PTimeComplete*
OWL2 QL	DL-Lite	AC_0

Erweiterung

Turmbau zu Babel, Pieter Brueghel, 1563



06 Rules - There's more than OWL

Open HPI - Course: Knowledge Engineering with Semantic Web Technologies
Lecture 5: Knowledge Representations II

The Role of Rules

- The **Semantic Web** concentrates on **declarative** forms of knowledge representation
  - OWL, RDFS

- Rules are a common form of **procedural** knowledge representation in **Knowledge Engineering**
  - Expert Systems
  - Prolog, CLIPS, JESS, OPS, …

- Knowledge representation formalisms of the Semantic Web have **expressive limitations** which can be overcome by rule-based knowledge
  - e.g. composition of complex classes from classes and properties

What are Rules?


```
IF  A .... THEN  B ....

    A      →     B
```

- Interpretation of a rule depends on context
  - General Inference:
    Premise → Conclusion
  - Hypothesis:
    Cause → Effect
  - Production:
    Condition → Action

What are Rules?

- **Logical Rules** (FOL implication):
  - `F → G` is equivalent with `¬F ∨ G`
  - Logical extension of the KB (static)
  - Open World, declarative

- **Procedural Rules** (e.g. Production Rules):
  - `If X then Y else Z`
  - executable machine instructions (dynamic)
  - operational (semantics = effect at application)

- **Logic Programming Rules** (e.g. Prolog, F-Logic):
  - `„woman(X) <- person(X) AND NOT man(X)“`
  - Approximation of logical semantics with operational aspects
  - Closed World (mostly), semi-declarative

FOL as Rule Language

- Rules as FOL implications (***Horn Clause***)

$$\underbrace{A_1 \wedge A_2 \wedge \ldots \wedge A_n}_{\text{Body}} \rightarrow \underbrace{H}_{\text{Head}}$$

- semantically equivalent with

$$\neg A_1 \vee \neg A_2 \vee \ldots \vee \neg A_n \vee H$$

- where A_i, H are atomic formulas
- Quantification most times ommitted, free variables are considered to be universally quantified
  - i.e. the rule holds for all possible assignments

FOL as Rule Language

- Rules as FOL implications (*Horn Clause*)

$$\underbrace{H}_{\text{Head}} \leftarrow \underbrace{A_1 \wedge A_2 \wedge \ldots \wedge A_n}_{\text{Body}}$$

often written from right to left (← or :-)

- semantically equivalent with

$$\neg A_1 \vee \neg A_2 \vee \ldots \vee \neg A_n \vee H$$

- where A_i, H are atomic formulas
- Quantification most times ommitted, free variables are considered to be universally quantified
  - i.e. the rule holds for all possible assignments

Variants of FOL Rules

- **Lloyd-Topor Transformation**
  - Several atoms in the head are usually considered as **conjunction**

$$A_1\ A_2 \ldots A_n \rightarrow H_1\ H_2 \ldots H_m$$

  - is equal to

$$A_1\ A_2 \ldots A_n \rightarrow H_1$$
$$A_1\ A_2 \ldots A_n \rightarrow H_2$$
$$\ldots$$
$$A_1\ A_2 \ldots A_n \rightarrow H_m$$

  - this transformation is called **Lloyd-Topor Transformation**

Variants of FOL Rules

- **Disjunctive Rules**
  - Disjunction of several non-negated Atoms

$$\underbrace{A_1 \wedge A_2 \wedge \ldots \wedge A_n}_{\text{Body}} \rightarrow \underbrace{H_1 \vee H_2 \vee \ldots \vee H_m}_{\text{Head}}$$

- reverse implication, as e.g.

  „if I see someting, then the light is on or the sun is shining“

Variants of FOL Rules

- FOL Rules
  - **Clause**: Disjunction of atomic formulas or negated atomic formulas
  - **Horn Clause**: Clause with **at most one** not negated atom

```
¬p ∨ ¬q ∨ . . . ∨ ¬t ∨ u      can be written as
 p ∧  q ∧ . . . ∧ t  → u
```

  - **Definite Clause**: Clause with **exactly one** not negated atom
  - **Fact**: Clause **of a single** not negated atom

Variants of FOL Rules

- Examples

$Person(x) \rightarrow Woman(x) \vee Man(x)$ (clause)

$Man(x) \wedge hasChild(x,y) \rightarrow Father(x)$ (definite clause)

$hasBrother(mother(x),y) \rightarrow isUncle(x,y)$
(with function symbol)

$Man(x) \wedge Woman(x) \rightarrow$ (horn clause)

$Woman(Anja)$ (fact)

- Semantics of rules complies to FOL semantics

Description Logics vs. Rules

- Rules are usually considered to apply only to **known** constants.

- No possibility to „create“ new things „on the flight“ by using existential quantification ∃

$$\texttt{Human} \sqsubseteq \exists \texttt{hasParent.Human}$$

- If rules are considered FOL formulas, then combining rules with $\mathcal{ALC}$ leads to undecidability.

- What about decidable FOL-Rules....?

DATALOG

DATALOG

- is a **logical rule language** that consists of
  - *horn clauses **without** function symbols*
  - *conjunction, constants, universally quantified variables, predicate symbols*
  - *no disjunction, no negation, no existential quantification, no function symbols*
- originally developed as foundation of deductive databases
- *Knowledge Bases* (Datalog Programs) are sets of horn clauses (without function symbols)
- DATALOG is decidable and computationally efficient, ExpTime

DATALOG - Syntax

- DATALOG **Term**: constant **c** or variable **v**
- DATALOG **Atom**: **$p(t_1,...,t_n)$**
  with predicate p,
  terms $t_1,...,t_n$
- DATALOG **Rule**: **$\forall x_1 ... \forall x_n\ (B_1 \wedge ... \wedge B_n \rightarrow H)$**
  with $B_1,...,B_n,H$ atoms and
  $x_1,...,x_n$ variables
- DATALOG **Program**: set of DATALOG rules

DATALOG Examples

19

(1)Vegetarian(x)∧FishProduct(y) → dislikes(x,y)

(2)orderedDish(x,y)∧dislikes(x,y) → Unhappy(x)

(3)orderedDish(x,y) → Dish(y)

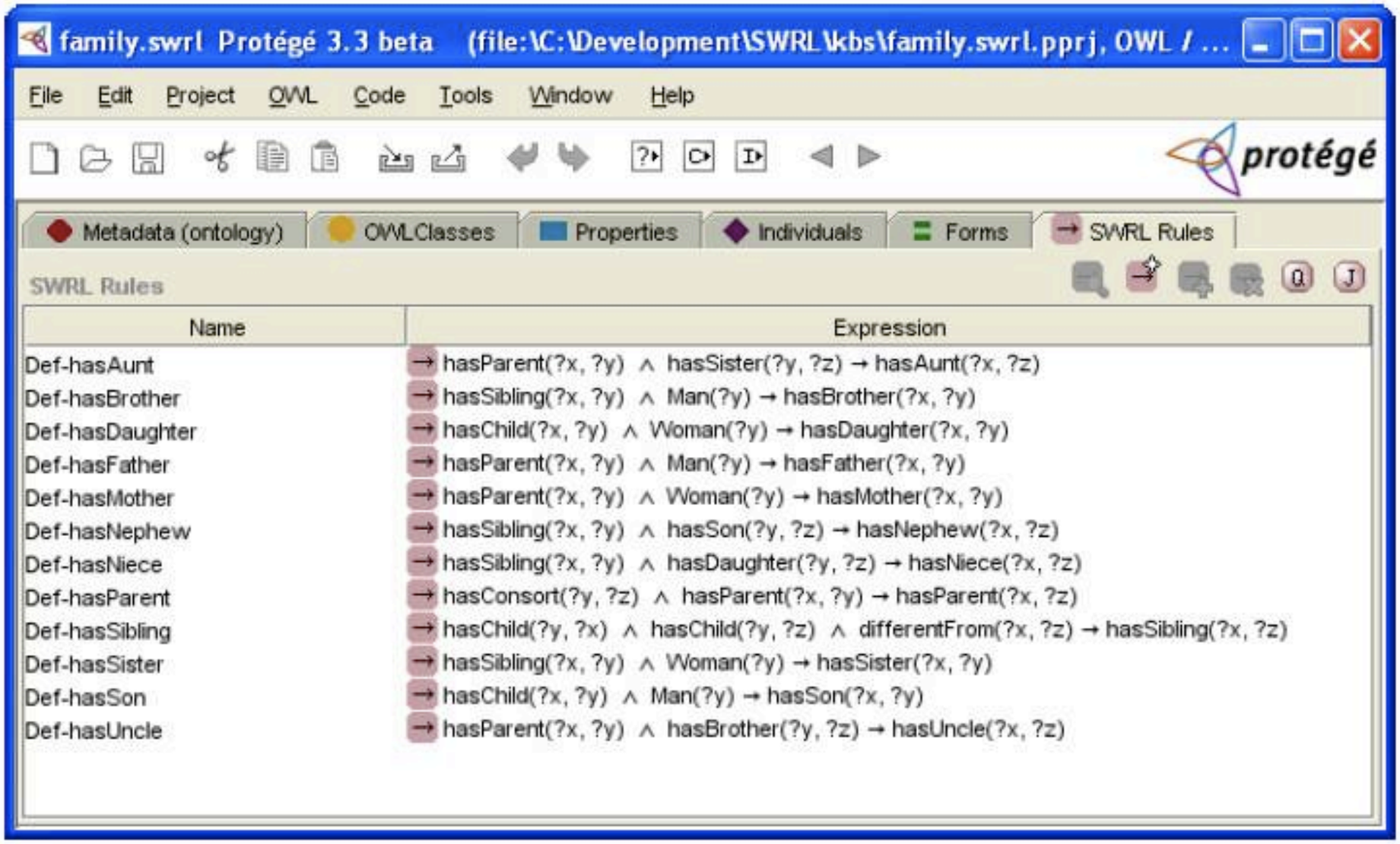
(4)dislikes(x,z)∧Dish(y)∧contains(y,z)→ dislikes(x,y)

(5)→ Vegetarian(Matthias)

(6)Happy(x) ∧ Unhappy(x) →

- DATALOG Rules allow mixing classes and relations (i.e. unary and binary predicates)
- therefore it can be **more expressive than DL**
- A combination of DATALOG and OWL is the **SWRL Language** (*not subject of this course*)

SWRL - Semantic Web Rule Language



- based on combination of parts of **OWL** and **RuleML/Datalog**
- **Idea**: Datalog Rules that apply on OWL ontologies
  - Symbols in rules can be OWL identifiers (or new Datalog identifiers)
- SWRL is undecidable

Lecture 6: Knowledge Engineering

Open HPI Course: Knowledge Engineering with Semantic Web Technologies