Knowledge Representation and Reasoning

OWL
Ontology Working Language
The WWW-Version of Description Logic

Prof. Dr. Hans Jürgen Ohlbach
Contents

• Historisches
• Syntaxes
• OWL-Full Elements
• OWL-DL
• OWL-Profiles
• OWL-Editor Protégé
• OWL as Spezifikation Language for Ontologies, based on Description Logics.
• Editors for OWL-Ontologies (Protégé,...)
• Reasoner for OWL-Ontologies (Fact, Racer, HermiT, KAON, ...)
• Java-API for loading OWL-Ontologies into programs.
• other APIs
• In the USA: since about 2000, The DARPA Agent Markup Language (DAML) XML/RDF-based (no formal Semantics)
• DAML+OIL (Ontology Inference Layer): Description Logic integrated into DAML
• Ab 2002: OWL 1: (Web Ontology Language) as successor of DAML+OIL:
  3 Versions: OWL-Lite, OWL-DL, OWL-FULL
  (satisfiability is not decidable for OWL-FULL!)
• Since 2009: OWL 2: OWL-DL, OWL-FULL, more language constructs
  OWL Profiles: OWL-EL, OWL-QL, OWL-RL
  restricted versions for special applications
## Example: Subcass Relation

### Functional-Style Syntax

SubClassOf( :Woman :Person )

### Turtle Syntax

:Woman rdfs:subClassOf :Person

### Manchester Syntax

Class: Woman  
SubClassOf: Person

### OWL/XML Syntax

```xml
<SubClassOf>
  <Class IRI="Woman"/>
  <Class IRI="Person"/>
</SubClassOf>
```

### RDF/XML Syntax

```xml
<owl:Class rdf:about="Woman">
  <rdfs:subClassOf rdf:resource="Person"/>
</owl:Class>
```
Overview
See

OWL 2 Web Ontology Language Primer
http://www.w3.org/TR/owl2-primer/

OWL 2 Web Ontology Language Document Overview
http://www.w3.org/TR/owl2-overview/

We use the Functional Style Syntax
Class Relationships

SubClassOf( :Woman :Person )
Woman is a subclass of Person

EquivalentClasses( :Person :Human )
Person and Human are equivalent

DisjointClasses( :Woman :Man )
Woman and Man are disjoint

`:` refers to a namespace (in this case empty)
Element Relationships

ClassAssertion( :Person :Mary )
  Mary is an Instance of the class Person

ObjectPropertyAssertion( :hasWife :John :Mary )
  Mary is John's Wife

NegativeObjectPropertyAssertion( :hasWife :Bill :Mary )
  Mary is not Bill's wife

DifferentIndividuals( :John :Bill )
  John and Bill are different

SameIndividual( :James :Teacher )
  James and the Teacher are identical
Features for Individuals

DataPropertyAssertion( :hasAge :John "51"^^xsd:integer )
John is 51 years of age

NegativeDataPropertyAssertion( :hasAge :Jack "53"^^xsd:integer )
Jack is no 53 Jahre years of age

xsd:integer refers to data types from XSL-Schema.

DataPropertyDomain( :hasAge :Person )
DataPropertyRange( :hasAge xsd:nonNegativeInteger )
hasAge maps Persons to non-negative Integers
Role Relationships

- **Role Hierarchies**
  SubObjectPropertyOf( :hasWife :hasSpouse )
  hasWife ist subrelation of hasSpouse

- **Domain and Range Restrictions**
  ObjectPropertyDomain( :hasWife :Man )
  Only men can have wifes
  ObjectPropertyRange( :hasWife :Woman )
  Only women can be wifes

- **Inverse Roles**
  InverseObjectProperties( :hasParent :hasChild )
  hasChild-1 = hasParent
Properties of Roles

- **Symmetry**
  SymmetricObjectProperty( :hasSpouse )
  \( \forall x,y \ r(x,y) \rightarrow r(y,x) \)

- **Asymmetry**
  AsymmetricObjectProperty( :hasChild )
  \( \forall x,y \ r(x,y) \rightarrow \neg r(y,x) \)

- **Reflexivity**
  ReflexiveObjectProperty( :hasRelative )
  \( \forall x \ r(x,x) \)

- **Irreflexivity**
  IrreflexiveObjectProperty( :parentOf )
  \( \forall x \ \neg r(x,x) \)
Further Properties of Roles

• **Disjointness**
  DisjointObjectProperties( :hasParent :hasSpouse )
  \[ \forall x,y \ r(x,y) \rightarrow \neg s(x,y) \]

• **Functionality**
  FunctionalObjectProperty( :hasHusband )
  \[ \forall x,y,z \ r(x,y) \land r(x,z) \rightarrow y = z \]

• **Inverse Functionality**
  InverseFunctionalObjectProperty( :hasHusband )
  \[ \forall x,y,z \ r(x,y) \land r(z,y) \rightarrow x = z \]

• **Transitivity**
  TransitiveObjectProperty( :hasAncestor )
  \[ \forall x,y,z \ r(x,y) \land r(y,z) \rightarrow r(x,z) \]
Role Chains

EquivalentObjectProperties(
  :hasGrandparent
  \textbf{ObjectPropertyChain}( :hasParent :hasParent ))

hasGrandparent = hasParent \circ hasParent
Set Operations on Classes

**Intersection of** Classes:

EquivalentClasses(
  :Mother
  ObjectIntersectionOf( :Woman :Parent))

Mothers are Women $\cap$ Parents

**Union of** Classes:

EquivalentClasses(
  :Parent
  ObjectUnionOf( :Mother :Father ))

Parents are Mothers $\cup$ Fathers
Further Set Operations

Complement

EquivalentClasses(
    :ChildlessPerson
    ObjectIntersectionOf(:Person
        ObjectComplementOf( :Parent )))

Childless Persons are Persons $\cap$ Complement of Parent

or

ClassAssertion(
    ObjectIntersectionOf(:Person
        ObjectComplementOf( :Parent ))
    :Jack)

Jack is childless
Quantifiers

∃-Quantifier

\[
\text{EquivalentClasses( :Parent} \\
\text{ObjectSomeValuesFrom( :hasChild :Person ))} \\
\text{Parent} = \exists \text{ hasChild Person}
\]

∀-Quantifier

\[
\text{EquivalentClasses( :HappyPerson} \\
\text{ObjectAllValuesFrom( :hasChild :HappyPerson ))} \\
\text{HappyPerson} = \forall \text{ hasChild HappyPerson}
\]
Nominals

EquivalentClasses(
  :JohnsChildren
  ObjectHasValue( :hasParent :John ))

JohnsChildren = ∀ hasParent {John}

EquivalentClasses(
  :NarcisticPerson
  ObjectHasSelf( :loves ))

∀x NarcisticPerson(x) ⇔ loves(x,x)
Narcists love themselves.
Number Restrictions

ClassAssertion(
   ObjectMaxCardinality( 4 :hasChild :Parent ) :John)
John has at most 4 children which are parents

ClassAssertion(
   ObjectMinCardinality( 2 :hasChild :Parent ) :John)
John has at least 2 children which are parents.

ClassAssertion(
   ObjectExactCardinality( 3 :hasChild :Parent ) :John)
John has exactly 3 children which are parents

One can omit the 3rd argument
(unqualified number restrictions)
Enumerations

EquivalentClasses(
   :MyBirthdayGuests
   ObjectOneOf( :Bill :John :Mary))

MyBirthdayGuests are just Bill, John and Mary,
And nobody else.
SubClassOf(
  **Annotation**( rdfs:comment "States that every man is a person." )
  :Man
  :Person)

An annotation has no logical content. It is a meta-information.
Ontology Definitions are enclosed in
   Ontology(URL, ...)
and equipped with an URL.

**Example:**

Ontology(<http://workingontologist.org/Examples/Chapter7/Ancestry.owl>  
   ...  
   )

These may be imported by other ontologies

Import( <http://workingontologist.org/Examples/Chapter7/Ancestry.owl> )
Prefix(:=<http://example.com/ontology#>)
Prefix(otherOnt:=<http://example.org/otherOntologies/>)
Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>)
Prefix(owl:=<http://www.w3.org/2002/07/owl#>)

With such prefixes one can refer to other ontologies.
Entity-Declarations

All names must be declared first.

Example:

Declaration( NamedIndividual( :John ) )
Declaration( Class( :Person ) )
Declaration( ObjectProperty( :hasWife ) )
Declaration( DataProperty( :hasAge ) )
OWL-DL

OWL-DL in OWL 1 realises the Description Logic $SHOIN(D)$, which is $ALC$ plus role hierarchies, nominals, inverse roles, restricted number restrictions with the concrete Domain $D$.

OWL-2 extended this to the still decidable $SHOIQ(D)$. New are:

- Role Inclusion Axioms of the form $R ^{\circ} S \subseteq R$ or $S ^{\circ} R \subseteq R$
  (Ex.: “the friends of my enimies are also my enimies: is-enimy $^{\circ}$ has-friend $\subseteq$ is-enimy)

- reflexive, symmetric, transitive, irreflexive, disjunct and universal roles

- $\exists R$.Self, (see the Narcist Example).

- Negated relations in A-Boxes, as well as

- quantified number restrictions.

• OWL 2 EL was developed for large medical ontologies.
• It implements the Description Logic EL++
• The most important restrictions are
  – no negation and disjunction
  – no universal Quantifier
  – no role inverse

No branching in the tableau!
**OWL 2 QL** can be seen as extensions of relational databases. It contains constructs which can be mapped to SQL.

**OWL 2 RL** is an extension of RDF. OWL 2 RL A-Boxes correspond to RDF-graphs. In particular one cannot generate further individuals.
Summary

• OWL is standardised by W3C.
• Most of the ontologies are now developed in OWL. They have tens of thousands of classes (e.g. http://protegewiki.stanford.edu/wiki/Protege_Ontology_Library)
• The more operators one can handle the more expressive becomes OWL (OWL 1, OWL 2, ...)
• Restricted versions of OWL allow for more efficient algorithms. Which operators are necessary is usually determined by the application.
• There are comfortable editors for OWL, one of them is Protégé.
• More extensions are possible, but they require more research.