## Syntaxes

<table>
<thead>
<tr>
<th>Name of Syntax</th>
<th>Specification</th>
<th>Status</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDF/XML</td>
<td>Mapping to RDF Graphs, RDF/XML</td>
<td>Mandatory</td>
<td>Interchange (can be written and read by all conformant OWL 2 software)</td>
</tr>
<tr>
<td>OWL/XML</td>
<td>XML Serialization</td>
<td>Optional</td>
<td>Easier to process using XML tools</td>
</tr>
<tr>
<td>Functional Syntax</td>
<td>Structural Specification</td>
<td>Optional</td>
<td>Easier to see the formal structure of ontologies</td>
</tr>
<tr>
<td>Manchester Syntax</td>
<td>Manchester Syntax</td>
<td>Optional</td>
<td>Easier to read/write DL Ontologies</td>
</tr>
<tr>
<td>Turtle</td>
<td>Mapping to RDF Graphs, Turtle</td>
<td>Optional, Not from OWL-WG</td>
<td>Easier to read/write RDF triples</td>
</tr>
</tbody>
</table>
MaxOneParent = parent max 1 Thing

```
<owl:Class rdf:about="&ex;MaxOneParent">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&ex;parent"/>
      <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>
```

Human ≡ Animal and (parent only Human)

```
<owl:Class rdf:about="&Cycles;Human">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <rdfs:Description rdf:about="&Cycles;Animal"/>
        <owl:Restriction>
          <owl:onProperty rdf:resource="&Cycles;parent"/>
          <owl:allValuesFrom rdf:resource="&Cycles;Human"/>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```
Functional syntax

Declaration(Class(:Human))

EquivalentClasses(:Human
    ObjectIntersectionOf(
        ObjectAllValuesFrom(:parent :Human)
        :Animal))

Declaration(Class(:MaxOneParent))

EquivalentClasses(:MaxOneParent
    ObjectMaxCardinality(1 :parent))

Metamodeling (by punning)

DL limitation: entities are either classes or individuals, not both.

In some situations this causes modeling problems

In OWL2, an IRI I can be used to refer to more than one type of entity.

Goal: To state facts about classes and properties themselves.

entities that share the same IRI should be understood as different "views" of the same underlying notion identified by the IRI.
Example

(1) ClassAssertion( x:Dog x:Brian ) Brian is a dog.
(2) ClassAssertion( x:Species x:Dog ) Dog is a species.

in (1) x:Dog is a class
in (2) x:Dog is an individual, member of x:Species

The individual x:Dog and the class x:Dog should be understood as two "views" of one and the same IRI — x:Dog.

The OWL 2 Direct Semantics treats the different uses of the same name as completely separate, as is required in DL reasoners.

Metamodelling and Annotations

Two means to associate additional information with classes and properties.

- **Metamodeling** should be used when the information attached to entities should be considered a part of the domain.
- **Annotations** should be used when the information attached to entities should not be considered a part of the domain and when it should not contribute to the logical consequences of an ontology.
Data Ranges

'One Of' Class Constructor

ObjectOneOf := 'ObjectOneOf' '(' Individual { Individual }')'

EquivalentClasses( a:GriffinFamilyMember
    ObjectOneOf( a:Peter a:Lois a:Stewie a:Meg a:Chris a:Brian )
)

- The Griffin family consists exactly of Peter, Lois, Stewie, Meg, Chris, and Brian.
DifferentIndividuals

DifferentIndividuals( a:Quagmire a:Peter a:Lois a:Stewie a:Meg a:Chris a:Brian )

- Quagmire, Peter, Lois, Stewie, Meg, Chris, and Brian are all different from each other.
**Self-Restriction**

ObjectHasSelf := 'ObjectHasSelf' '(' ObjectPropertyExpression ')' contains all those individuals that are connected by OPE to themselves.

- ObjectPropertyAssertion( a:likes a:Peter a:Peter )
- LTS ≡ ObjectHasSelf( a:likes )
  - those individuals that like themselves;
  - a:Peter is classified as an instance of LTS

**Literal Value Restriction**

DataHasValue( DPE lit ) contains all those individuals that are connected by DPE to lit.

a syntactic shortcut for the class expression

DataSomeValuesFrom( DPE DataOneOf( lit ) ).
Example

DataPropertyAssertion( a:hasAge a:Meg "17"^^xsd:integer )
- Meg is seventeen years old.

DataHasValue( a:hasAge "17"^^xsd:integer )
- contains all individuals that are connected by a:hasAge to the integer 17;
- a:Meg is classified as its instance:
### 9.1.4 Disjoint Union of Class Expressions

**DisjointUnion** (C CE₁ ... CEₙ)

- states that a class C is a disjoint union of the class expressions CEᵢ, 1 ≤ i ≤ n, all of which are pairwise disjoint.

- each instance of C is an instance of exactly one CEᵢ, and each instance of CEᵢ is an instance of C.

- a syntactic shortcut for the following two axioms:
  - `EquivalentClasses( C ObjectUnionOf( CE₁ ... CEₙ ) )`
  - `DisjointClasses( CE₁ ... CEₙ )`

### Object Subproperties

- analogous to subclass axioms

**SubObjectPropertyOf** (OPE₁ OPE₂).

- states that the object property expression OPE₁ is a subproperty of the object property expression OPE₂

- if an individual x is connected by OPE₁ to an individual y, then x is also connected by OPE₂ to y.
Object subproperty chains

SubObjectPropertyOf( ObjectPropertyChain( OPE₁ ... OPEₙ ) OPE ).

- if an individual x is connected by a sequence of object property expressions OPE₁, ..., OPEₙ with an individual y, then x is also connected with y by the object property expression OPE.

- also known as complex role inclusions.

Example

SubObjectPropertyOf( ObjectPropertyChain( a:hasMother a:hasSister ) a:hasAunt )
ObjectPropertyAssertion( a:hasMother a:Stewie a:Lois )
ObjectPropertyAssertion( a:hasSister a:Lois a:Carol )
entails
ObjectPropertyAssertion( a:hasAunt a:Stewie a:Carol )
Object property axioms

- disjoint properties
- inverse
- domain
- range
- (inverse) functional
- (ir)reflexive, transitive, (a)symmetric

9.5 Keys

HasKey( CE ( OPE₁ ... OPEₘ ) ( DPE₁ ... DPEₙ ) )

- states that each (named) instance of the class expression CE is uniquely identified by the object property expressions OPEᵢ and/or the data property expressions DPEⱼ.

- no two distinct (named) instances of CE can coincide on the values of all object property expressions OPEᵢ and all data property expressions DPEⱼ.
Key and InversFunctional

HasKey( owl:Thing ( OPE ) () )
is similar to
InverseFunctionalObjectProperty( OPE ),
difference
- HasKey is applicable only to individuals that are explicitly named in an ontology,
- InverseFunctionalObjectProperty is also applicable to individuals whose existence is implied by existential quantification.

Profiles

An OWL 2 profile (commonly called a fragment or a sublanguage in computational logic) is a trimmed down version of OWL 2 that trades some expressive power for the efficiency of reasoning.

EL : polynomial time reasoning (large ontologies)
QL : LOGSPACE reasoning ("database" applications)
RL : polynomial time reasoning with a rule-based (database) system
OWL 2 EL Profile

- for applications employing ontologies that define very large numbers of classes and/or properties,
- captures the expressive power used by many such ontologies,
- consistency, class expression subsumption, and instance checking can be decided in \textit{polynomial time}.

Constructs not supported in EL

- $R \text{ only } C$
- $R \text{ min } n \ C, \text{ R max } n \ C$
- $C \text{ or } D$
- $\text{not } C$
- $\{a_1, a_2, \ldots, a_n\}$ with $n > 1$

On properties:
- disjoint, irreflexive, inverse, functional and inverse functional,
- symmetric, asymmetric
**QL Profile**

- sound and complete query answering is in \text{LOGSPACE}
- many of the main features necessary to express conceptual models such as UML class diagrams and ER diagrams
- contains the intersection of RDFS and OWL 2 DL
- assertions stored in a standard relational database system can be queried through an ontology via a simple rewriting mechanism
  - rewriting the query into an SQL query

**Restrictions on axiom structures**

- a class
- \( R \ some \ Thing \)
- \( R \ some \ DataRange \)
- a class
- \( C \ and \ D \)
- \( \text{not} \ C \)
- \( R \ some \ C \)
- \( R \ some \ DataRange \)
Constructs not supported

- R only C
- R value V
- \{a, b, ...\}
- R min/max n C
- C or D
- ...

Axioms not supported

- P1 o P2 o ... subPropertyOf P
- sameIndividual(a, b)
- not P(a, b)
Rewriting examples

A sub R some C  
C sub D and E

Find all the instances of A:

```
select TA.id  
from TA, TR, TC, TD, TE  
where TA.id = TR.from and TR.to = C.id and C.id = D.id and C.id = E.id
```

OWL 2 RL

**OWL 2 RL** enables the implementation of polynomial time reasoning algorithms using rule-extended database technologies operating directly on RDF triples; it is particularly suitable for applications where relatively lightweight ontologies are used to organize large numbers of individuals and where it is useful or necessary to operate directly on data in the form of RDF triples.
Restrictions on axiom structures

- a class
- \{a, b, c, ...\}
- C and D
- C or D
- R some C
- R some DataRange
- R value a
- ...

Reasoning rules

- Semantics based on first order implication
- Rules to infer triples
- Provide basis for the implementation of rule-based reasoners
<table>
<thead>
<tr>
<th>T(?c₁, rdfs:subClassOf, ?c₂)</th>
<th>T(?x, rdf:type, ?c₁)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T(?c₁, owl:equivalentClass, ?c₂)</td>
<td>T(?x, rdf:type, ?c₂)</td>
</tr>
<tr>
<td>T(?c₁, owl:equivalentClass, ?c₂)</td>
<td>T(?x, rdf:type, ?c₂)</td>
</tr>
<tr>
<td>T(?c₁, owl:disjointWith, ?c₂)</td>
<td>T(?x, rdf:type, ?c₁)</td>
</tr>
<tr>
<td>T(?c₁, owl:disjointWith, ?c₂)</td>
<td>T(?x, rdf:type, ?c₁)</td>
</tr>
<tr>
<td>T(?x, rdf:type, owl:AllDisjointClasses)</td>
<td>false</td>
</tr>
<tr>
<td>T(?x, owl:members, ?y)</td>
<td>false</td>
</tr>
<tr>
<td>LIST[?y, ?c₁, ..., ?cₙ]</td>
<td></td>
</tr>
<tr>
<td>T(?z, rdf:type, ?c₁)</td>
<td></td>
</tr>
<tr>
<td>T(?z, rdf:type, ?c₁)</td>
<td></td>
</tr>
</tbody>
</table>

© U. de Genève - G. Falquet