Introduction to OWL language
(Ontology Web Language)

Part. 2: OWL2
Bernard ESPINASSE
Aix-Marseille Université (AMU)
Polytech-Marseille
Nov. 2017

• From OWL1 to OWL2
• Declaration & Assertion Axioms in OWL2 DL
• Property Axioms & Class Expressions in OWL2 DL
• OWL2 Profiles: EL, QL and RL profiles
• Limits of OWL2

Outline

1. From OWL1 to OWL2
   • Aims and Overview of OWL2
   • OWL2 Syntaxes and Semantics
   • Ontology in OWL2: Axioms and Properties

2. Declaration & Assertions Axioms
   • Class and Individual declarations, Class and Subclass assertions
   • Property declaration and assertion, Negative property declaration
   • Annotation assertion

3. Property Axioms and Chains
   • Property axioms and Property chains

4. Class Expressions
   • Equivalent, Disjoint, Enumeration, Union, Intersection, Complement
   • Existential & universal quantification, individual value restriction, Datatype restriction

5. OWL2 Profiles
   • EL, QL and RL profiles

6. Conclusion
   • Limits of OWL1 and OWL2

References

• Books, articles and reports :
  • W3C, « OWL Web Ontology Language Semantics and Abstract Syntax »
  • ...

• Web W3C :
  • Page du W3C : http://www.w3.org/2004/OWL/
  • Référence : http://www.w3.org/TR/owl-ref/
  • Guide : http://www.w3.org/TR/owl-guide/
  • ...

• Course/tutorials :
  • Course of I. Horrocks and F. Farazi
  • Tutorial of M. Kuba, Institute of Computer Science,
    http://dior.ics.muni.cz/~makub/owl/
  • ...

1. From OWL1 to OWL2

   • Aims of OWL2
   • OWL2 Overview
   • Ontologies in OWL2
   • OWL2 Syntaxes
   • OWL2 Semantics
Aims of OWL2

- **OWL 2** is an extension and revision of the OWL Web Ontology Language developed by the W3C Web Ontology Working Group and published in 2004 (referred to hereafter as "OWL 1")
- **OWL 2** is developed by the W3C OWL Working Group (2009-...)
- **OWL 2** is compatible with OWL1
- Like OWL 1, **OWL 2** is designed to facilitate ontology development and sharing via the Web,
- With the ultimate goal of making Web content more accessible to machines.

2 Semantic views of an OWL2 Ontology

1. **Structural form**: is defined in the OWL2 Structural Specification document [OWL 2 Structural Specification]
   - **UML** is used to define the structural elements available in OWL2, explaining their roles and functionalities in abstract terms and without reference to any particular syntax.
   - **Functional-style syntax** defined following the structural specification and allowing OWL2 ontologies to be written in a compact form.

2. **RDF graph form**: Any OWL2 ontology can also be viewed as an RDF graph:
   - Mapping from the structural form to the RDF graph form, and vice versa are defined in [OWL 2 RDF Mapping]
   - **OWL 2 Quick Reference Guide** [OWL 2 Quick Guide] provides a simple overview of these two views of OWL 2, laid out side by side.

OWL2 Overview

- An **OWL2 ontology** can be thought as an abstract structure or an RDF graph: **2 Semantic specifications** define the meaning of OWL2 ontologies
- **Various concrete syntaxes** that can be used to serialize and exchange ontologies (OWL2 users need only one syntax and one semantics)

OWL2 Syntaxes (1)

- In practice, as in OWL1 a concrete syntax is needed in order to store OWL2 ontologies and to exchange them among tools and applications
- Primary exchange syntax for OWL 2 is **RDF/XML** [RDF Syntax], that is the only syntax that must be supported by all OWL 2 tools
- As for OWL1, other concrete syntaxes may also be used in OWL2 as:
  - **RDF serializations**, such as Turtle [Turtle];
  - an **XML serialization** [OWL 2 XML];
  - a more "readable" syntax, called the **Manchester Syntax** [OWL 2 Manchester Syntax], that is used in several ontology editing tools
  - **Functional-style syntax** can also be used for serialization, although its main purpose is specifying the structure of the language [OWL 2 Structural Specification].
### OWL2 Syntaxes (2)

Comparing various OWL2 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 conformat 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</td>
<td>Easier to read/write RDF triples</td>
</tr>
</tbody>
</table>

### OWL2 Semantics

- **2 alternative ways of assigning meaning to OWL 2 ontologies:**
  - **Direct Semantics** [OWL 2 Direct Semantics]
  - **RDF-Based Semantics** [OWL 2 RDF-Based Semantics]

  These 2 semantics are used by *reasoners* and *other tools*, e.g., to answer to:
  - **class consistency**
  - **subsumption**
  - **instance retrieval queries**.

### OWL2 Semantics : Direct Semantics

- **Assigns meaning directly to ontology structures**, resulting in a semantics compatible with the model theoretic semantics of the SROIQ Description Logic.
- **Advantage:** extensive DL literature and implementation experience can be directly exploited by OWL 2 tools.
- **Conditions:**
  - be placed on ontology structures in order to ensure that they can be translated into a SROIQ knowledge base (Ex: transitive properties cannot be used in number restrictions) – cf. Section 3 of « OWL 2 Structural Specification document »
  - Ontologies satisfying these conditions are called « OWL 2 DL » ontologies, and interpreted using the Direct Semantics.

### OWL2 Semantics : RDF-Based Semantics

- **Assigns meaning directly to RDF graphs** and so indirectly to ontology structures via the Mapping to RDF graphs
- **Is fully compatible with the RDF Semantics**, and extends the semantic conditions defined for RDF

  The RDF-Based Semantics can be applied to any OWL 2 Ontology **without restrictions**, as any OWL 2 Ontology can be mapped to RDF

  « OWL 2 Full » is informally used to refer to RDF graphs considered as OWL 2 ontologies and interpreted using the RDF-Based Semantics.
OWL2 : Correspondence theorem between Semantics

- The correspondence theorem from the RDF-Based Semantics Document defines a precise, close relationship between the Direct and RDF-Based Semantics.

- This theorem states, that:
  - given an « OWL 2 DL » ontology,
  - inferences drawn using the Direct Semantics will still be:
    - valid if the ontology is mapped into an RDF graph and interpreted using the RDF-Based Semantics.

OWL2 Ontology : Axioms and Properties

- OWL Ontology is a set of axioms, which provides explicit logical assertions about:
  - Classes
  - Individuals
  - Properties
    Same axiom can be written according to different syntaxes (XML/RDF, Turtle, Manchester, Functional, …)

There are 2 types of properties in OWL ontologies:
  - Data properties : binary relations that link an individual to a piece of typed data, like to a xsd:dateTime or xsd:string literal
  - Object properties : binary relations that link an individual to an individual.

OWL Ontology: a simple example

- A small ontology with 2 classes (Person and Student) and 2 individuals: Berny and Sabine (Graph obtained by OntoGraf in Protege-OWL 4.2 editor)

- By using a Reasoner we can infer other facts which are implicitly contained in the ontology:
  - if an individual Berny is in class Student, and
  - the class Student is a subclass of the class Person,
  - a reasoner will infer that Berny is a Person.

2. Declaration & Assertion

Axioms in OWL2 DL

- Class declaration
- Individual declaration
- Class assertion
- Subclass assertion
- Property declaration
- Property assertion
- Negative property declaration
- Annotation assertion
Declaration & Assertion Axioms in OWL2 (1)

- **Class declaration** defines a class, that may contain individuals:
  
  Declaration(Class(:Person))

- **Individual declaration** defines a named individual:
  
  Declaration(NamedIndividual(:Berny))
  Declaration(NamedIndividual(:Sabine))

- **Class assertion** state that an individual belongs to a class:
  
  ClassAssertion(:Student :Berny)

- **Subclass assertion** declares that all individuals that belong to a class belong also to another class:
  
  SubClassOf(:Student :Person)

Declaration & Assertion Axioms in OWL2 (2)

- **Property declaration** defines:
  
  - a data property to link an individual to data
  
  Declaration(DataProperty(:hasEmail))
  Declaration(ObjectProperty(:hasSpouse))

- **Property assertion** states the relation of an individual to a data or an individual:
  
  DataPropertyAssertion(:hasEmail :Berny "berny@univ.fr"^^xsd:string)
  ObjectPropertyAssertion(:hasSpouse :Berny :Sabine)

Declaration & Assertion Axioms in OWL2 (3)

- **Negative property assertion** states that the relation of an individual to either data or individual does not exist.

- **OWL2 uses Open World Assumption**, so if an individual is not linked by some property with some value, it may be for 2 reasons:
  - either it really does not have the property with the value,
  - or it is unknown because the information missing from the ontology.

  A **negative property assertion** states that the individual cannot possibly have that property value:

  NegativeObjectPropertyAssertion(:hasSpouse :Berny :John)
  NegativeDataPropertyAssertion(:hasEmail :Berny "president@Elysee.gouv"")

Declaration & Assertions Axioms in OWL2 (4)

- **Annotation assertion** enables to annotate anything with some details

  - **Ex:** we may use "AMU" as the name of an individual because it is an abbreviation, and use annotation to label the abbreviation with the full meaning "Aix-Marseille University", idem for "LSIS" abbreviation:

    Declaration(NamedIndividual(Aix-Marseille_University))
    AnnotationAssertion(rdfs:label:Aix-Marseille_University "AMU")
    Declaration(NamedIndividual(LSIS_UMR_CNRS))
    AnnotationAssertion(rdfs:label:LSIS_UMR_CNRS "LSIS")

- Images generated by Protege then show the label instead of the name:
3. Property Axioms in OWL2 DL

- Main property axioms
- Infer others properties with a reasoner
- Property chains

Property Axioms (1)

We can define that a property is:
- transitive,
- symmetric,
- asymmetric,
- reflexive,
- irreflexive,
- functional (can have only one value),
- inverse-functional (its inverse is functional),
- inverse to some other property,
- subproperty of some other property,
- equivalent to some other property,
- disjoint with some other property (2 individuals cannot be linked by both properties in the same time).

Ex: `hasSpouse` property is symmetric, functional and irreflexive:

```owl
SymmetricObjectProperty( :hasSpouse )
FunctionalObjectProperty( :hasSpouse )
IrreflexiveObjectProperty( :hasSpouse )
```

Property Axioms (2)

1. We can define a new transitive property `isPartOf` is defined (in yellow), which connects DIMAG team to LSIS lab., and LSIS lab to AMU University:

```owl
Declaration( ObjectProperty( :isPartOf ))
TransitiveObjectProperty( :isPartOf )
ObjectPropertyAssertion( :isPartOf :DIMAG :LSIS )
ObjectPropertyAssertion( :isPartOf :LSIS :AMU )
```

Property Axioms (3)

2. We can define `hasPart` property as an inverse property of `isPartOf`:

```owl
Declaration( ObjectProperty( :hasPart ))
TransitiveObjectProperty( :hasPart )
ObjectPropertyAssertion( :hasPart :DIMAG :LSIS )
ObjectPropertyAssertion( :hasPart :LSIS :AMU )
```

Then we can use a reasoner to infer the other properties, because the `isPartOf` is transitive, LSIS isPartOf AMU, and because `hasPart` is inverse to `isPartOf`, we obtain the following complete inferred relations:

```owl
ObjectPropertyAssertion( :isPartOf :DIMAG :AMU )
ObjectPropertyAssertion( :hasPart :AMU :LSIS )
ObjectPropertyAssertion( :hasPart :AMU :DIMAG )
ObjectPropertyAssertion( :hasPart :LSIS :DIMAG )
```

It shows the power of reasoning, we do not have to declare all relations, just the needed minimum, and the rest can be inferred.
Property Chains (1)

- They allow to define some relationships among 3 individuals (the property uncle defined as chain of parent and brother properties)
- A property may be chained even with itself.

Ex:

we define property isEmployedAt as a chain of itself and the transitive property isPartOf, meaning that if a person is employed at some organizational unit, the person is also employed at the bigger organizational units.

Declaration( ObjectProperty( :isEmployedAt ) )
ObjectPropertyAssertion( :isEmployedAt :Berny :DIMAG )
SubObjectPropertyOf( ObjectPropertyChain( :isEmployedAt :isPartOf ) :isEmployedAt )

Using a reasoner we may infer the complete relations:

ObjectPropertyAssertion( :isEmployedAt :Berny :LSIS )
ObjectPropertyAssertion( :isEmployedAt :Berny :AMU )

Property Chains (2)

- In Protege:
  Declaration( ObjectProperty( :isEmployedAt ) )
  ObjectPropertyAssertion( :isEmployedAt :Berny :DIMAG )
  SubObjectPropertyOf( ObjectPropertyChain( :isEmployedAt :isPartOf ) :isEmployedAt )

A reasoner infers the complete relations:

ObjectPropertyAssertion( :isEmployedAt :Berny :LSIS )
ObjectPropertyAssertion( :isEmployedAt :Berny :AMU )

(In Protege, in isEmployedAt object property declaration, you have to enter in dialog box of « Super Property Of (chain) » :isEmployedAt o isPartOf)

Class Expressions: Set Operations

- A common class definition is an assertion that a named class is equivalent to some (unnamed/anonymous) class defined by an expression.
- Classes can also be defined as disjoined with other class which means that they do not share any individuals.
- OWL 2 provides set operations as:
  - enumeration,
  - union,
  - intersection,
  - complement

(set operations in their usual mathematical meaning)
**Class Expressions: Different Individuals**

**Example:**

- We can define a new class Child which contains 4 new individuals named SmallBoy, BigBoy, SmallGirl, BigGirl.
- These individuals must be declared as different from each other, otherwise an OWL reasoner expects that they may be the same:

  - Declaration( Class( :Child ) )
  - Declaration( NamedIndividual( :BigBoy ) )
  - Declaration( NamedIndividual( :BigGirl ) )
  - Declaration( NamedIndividual( :SmallBoy ) )
  - Declaration( NamedIndividual( :SmallGirl ) )
  - ClassAssertion( :Child :BigBoy )
  - ClassAssertion( :Child :BigGirl )
  - ClassAssertion( :Child :SmallBoy )
  - ClassAssertion( :Child :SmallGirl )
  - DifferentIndividuals( :BigBoy :BigGirl :SmallBoy :SmallGirl )

**Class Expressions: Enumeration**

We define 2 base classes by enumerating their members:
- the Boy class containing boys and
- the Small class containing the small children.

(3) The individual SmallBoy is both in Boy and in Small classes)

**Functional syntax:**

- SubClassOf( :Boy :Child )
- SubClassOf( :Small :Child )
- EquivalentClasses( :Boy ObjectOneOf( :SmallBoy :BigBoy ) )
- EquivalentClasses( :Small ObjectOneOf( :SmallBoy :SmallGirl ) )

**Manchester syntax (Protege editor):**

- Class: Boy
  - EquivalentTo:
    - (BigBoy, SmallBoy)
    - Child
- Class: Small
  - EquivalentTo:
    - (SmallBoy, SmallGirl)
  - SubClassOf:
    - Child

A reasoner will find that the Girl class contains the individuals SmallGirl and BigGirl.

**Class Expressions: Complement & intersection**

We define a new class Girl as the intersection of the class Child with an unnamed class that is the complement of the class Boy

**Functional syntax:**

- Declaration( Class( :Girl ) )
- SubClassOf( :Girl :Child )
- EquivalentClasses( :Girl ObjectIntersectionOf( ObjectComplementOf( :Boy ) :Child ) )

**Manchester syntax (Protege editor syntax):**

- Class: Girl
  - SubClassOf:
    - Child
  - EquivalentTo:
    - Child and (not (Boy))

A reasoner will find that the Girl class contains the individuals SmallGirl and BigGirl.

**Class Expressions: Union**

- We define a new class Boy_or_Small as the union of the classes Small and Boy
- Then, the description is more intuitive in the Protege syntax as Boy or Small

**Protege syntax:**

- Class: Boy_or_Small
  - EquivalentTo:
    - Boy or Small

**Functional syntax:**

- Declaration( Class( :Boy_or_Small ) )
- EquivalentClasses( :Boy_or_Small ObjectUnionOf( :Small :Boy ) )

A reasoner will find that the class contains the individuals SmallBoy, SmallGirl, BigBoy.
### Introduction to OWL/Part 2

**Class expressions: Existential Quantification (1)**

- A class definition can say that the class contains only individuals that are connected by a given property with individuals from a given class.

- **Example:**
  
  We may define a new class `OrgUnit` which contains the 3 individuals AMU, LSIS, DIMAG:
  
  ```
  Declaration(Class(:OrgUnit))
  ClassAssertion(:OrgUnit :AMU)
  ClassAssertion(:OrgUnit :LSIS)
  ClassAssertion(:OrgUnit :DIMAG)
  ```

- Then we may define a new class `Employee`:
  
  - as a subclass of `Person`,
  - which is equivalent to an anonymous class of individuals that are linked by property `isEmployedAt` to individuals from the class `OrgUnit`.

  $\Rightarrow$ The meaning is that employees are the persons that are employed somewhere.

**Class expressions: Existential Quantification (2)**

**Protege syntax:**

```
Class: Employee
SubClassOf: Person
EquivalentTo: isEmployedAt some OrgUnit
```

**Functional syntax:**

```plaintext
Declaration(Class(:Employee))
SubClassOf(:Employee :Person)
EquivalentClasses(:Employee ObjectSomeValuesFrom(:isEmployedAt :OrgUnit))
```

In OWL2, the `ObjectSomeValuesFrom()` operator states: « individuals such that there exists some individual from the given class connected by the given property ».

**Class expressions: Existential Quantification (3)**

**A reasoner** we may infer that the class `Employee` contains the individual `Berny`, and produce an explanation for this:

```
<table>
<thead>
<tr>
<th>Explanation for: Berny Type: Employee</th>
</tr>
</thead>
<tbody>
<tr>
<td>Berny isEmployedAt DIMAG</td>
</tr>
<tr>
<td>DIMAG Type: OrgUnit</td>
</tr>
<tr>
<td>Employee EquivalentTo isEmployedAt some OrgUnit</td>
</tr>
</tbody>
</table>
```

**Class expressions: Individual Value Restriction**

- A class expression can also say that the class contains individuals that are connected by a given property with a given individual.

- **Example:**

  we define a new class `EmployeesOfAMU` as containing individuals that are connected by the property `isEmployedAt` with the individual `AMU - those who are employed at AMU`.

  **Protege syntax:**

  ```
  Class: EmployeesOfAMU
  SubClassOf: Person
  EquivalentTo: isEmployedAt value AMU
  ```

  **Functional syntax:**

  ```plaintext
  Declaration(Class(:EmployeesOfAMU))
  SubClassOf(:EmployeesOfAMU :Person)
  EquivalentClasses(:EmployeesOfAMU ObjectHasValue(:isEmployedAt :AMU))
  ```

  A reasoner will find that the class `EmployeesOfAMU` contains the individual `Berny`.
Class expressions: **Universal Quantification (1)**

- **Example:**
  - Every person has at most 2 parents. We suppose that if both parents are of the same nationality, then the person has the same nationality.
  - Let's create a new individual `Valentin`, which is connected by a new object property `hasParent` to the individuals `Berny` and `Sabine`.
  - `Valentin` is a child of `Berny` and `Sabine`.

  ```
  Declaration( NamedIndividual(:Valentin) )
  ClassAssertion( :Person :Valentin )
  Declaration( ObjectProperty( :hasParent ) )
  ObjectPropertyAssertion( :hasParent :Valentin :Berny )
  ObjectPropertyAssertion( :hasParent :Valentin :Sabine )
  ```

  We have:

  ![Diagram](image)

Class expressions: **Datatype restrictions**

For data properties, we can restrict the values of a data property.

**Example:**

the axiom to define that individuals in the `Person` class have exactly one value of the `hasAge` property that is an integer in the range between 0 and 130:

- **Functional syntax:**
  ```
  EquivalentClasses( :Person DataExactCardinality( 1 :hasAge DatatypeRestriction( xsd:integer xsd:minInclusive "0"^^xsd:integer xsd:maxInclusive "130"^^xsd:integer )))
  ```

- **Manchester syntax:**
  ```
  Class: Person
  EquivalentTo:
  hasAge exactly 1 xsd:integer[>= 0 , <= 130]
  ```

Class expressions: **Universal Quantification (2)**

- We can state that a `Person` has at most 2 parents:
  ```
  SubClassOf( :Person ObjectMaxCardinality( 2 :hasParent ) )
  ```

- We can define a new class `French` containing `Berny` and `Sabine`:
  ```
  Declaration( Class( :French ) )
  SubClassOf( :French :Person )
  ClassAssertion( :French :Sabine )
  ClassAssertion( :French :Berny )
  ```

  Last axiom: the universal quantification that individuals that have all parents French are (a subclass of) French.

  In OWL2 the `ObjectAllValuesFrom()` operator means: individuals such that they are always connected by the given property only to individuals from the given class.

  the conclusion made by a reasoner is `Valentin is in the class French.`

Class expressions: **Functional Data Property and keys (1)**

- The OWL's open world assumption says that ontology may not contain all information and thus some information may be unknown.

- This is why named individuals with different names may represent the same individual unless it is explicitly stated that they are different.

- This is also why we had to specify in the examples above (the set operations and the universal quantification) that the individuals are different.

- We can declare the named individuals different in 2 ways:
  - **1 - by the DifferentIndividuals() axiom:**
    ```
    DifferentIndividuals(:BigBoy :BigGirl
    :SmallBoy :SmallGirl :LSIS :AMU :DIMAG
    :Valentin :Sabine :Berny )
    ```

    -> this way cause performance problems for reasoner when the number of individuals is large.
Functional Data Property and keys (2)

2 - using a functional data property with unique values:
- Functional property can have only one value, and thus makes individuals different when their values are different
- If the property is also declared as a key, it makes individuals with the same value to be the same individual

Here:

\[\text{Declaration(DataProperty(:hasId))}\]
\[\text{FunctionalDataProperty(:hasId)}\]
\[\text{HasKey( :Person (:hasId) )}\]
\[\text{DataPropertyAssertion(:hasId :Sabine "1234")}\]
\[\text{DataPropertyAssertion(:hasId :Berny "5648")}\]

5. OWL2 Profiles

- OWL2 EL
- OWL2 QL
- OWL2 RL

OWL2 profiles (1)

- OWL 2 Profiles:
  - are sub-languages (syntactic subsets) of OWL2 (DL) that offer important advantages in particular application scenarios
  - are defined as a syntactic restriction of the OWL2 Structural Specification
  - are a subset of the structural elements that can be used in a conforming ontology
  - are more restrictive than OWL2 (DL)
  - they trade off different aspects of OWL’s expressive power in return for different computational and/or implementational benefits

- 3 profiles are defined in OWL2:
  - OWL2 EL
  - OWL2 QL
  - OWL2 RL

All these profiles can be interpreted using Direct or RDF-Based Semantics.

OWL profiles and Description Logics (DL)

Each profile is based on a specific DL, oriented towards specific purposes:

- OWL 1 and OWL2 languages and sub-languages:
  - OWL1 DL is based on the description logic SHOIN.
  - OWL1 Lite is based on the description logic SHIF.
  - OWL1 Full is based on an undecidable logic, superset of SHOIN
  - OWL2 (DL) is based on the description logic SROIQ, and is oriented towards enabling ontologies with a high degree of expressivity in the language.

- OWL 2 Profiles (are OWL2 (DL) restrictions):
  - OWL2 EL is based on the description logic EL+
  - OWL2-QL is based on DL-Lite
  - OWL2 RL is based on Description Logic Programs (DLP)
OWL2-EL Profile

- **OWL2 EL:**
  - is based on the description logic EL++
  - permitting scalable reasoning in the **TBox** : *polynomial-time reasoning* for most inference tasks such as classification

- particularly suitable for applications where:
  - very large ontologies are needed
  - expressive power can be traded for *performance guarantees*

---

OWL2-EL Profile: **Axiomes supported**

- class inclusion (SubClassOf)
- class disjointness (DisjointClasses)
- object property inclusion (SubObjectPropertyOf) with or without property chains, and data property inclusion (SubDataPropertyOf)
- property equivalence (EquivalentObjectProperties and EquivalentDataProperties),
- transitive object properties (TransitiveObjectProperty)
- reflexive object properties (ReflexiveObjectProperty)
- domain restrictions (ObjectPropertyDomain and DataPropertyDomain)
- range restrictions (ObjectPropertyRange and DataPropertyRange)
- assertions (SameIndividual, DifferentIndividuals, ClassAssertion, ObjectPropertyAssertion, DataPropertyAssertion, NegativeObjectPropertyAssertion, and NegativeDataPropertyAssertion)
- functional data properties (FunctionalDataProperty)
- keys (HasKey)

---

OWL2-QL Profile

- **OWL2 QL:**
  - is based on DL-Lite

  - permits scalable query answering in the **ABox** : *when dealing with lots of instance data and a relatively simple TBox*

  - enables *conjunctive queries to be answered in LogSpace* (more precisely, AC0) using standard relational database technology (SQL)

  - particularly suitable for applications where:
    - relatively lightweight ontologies are used to organize large numbers of individuals
    - needs access the data directly via relational queries (SQL)

---

OWL2-EL Profile: **Constructs NOT supported**

- universal quantification to a class expression (ObjectAllValuesFrom) or a data range (DataAllValuesFrom)
- cardinality restrictions (ObjectMaxCardinality, ObjectMinCardinality, ObjectExactCardinality, DataMaxCardinality, DataMinCardinality, and DataExactCardinality)
- disjunction (ObjectUnionOf and DisjointUnion)
- class negation (ObjectComplementOf)
- enumerations involving more than one individual (ObjectOneOf and DataOneOf)
- disjoint properties (DisjointObjectProperties and DisjointDataProperties)
- irreflexive object properties (IrreflexiveObjectProperty)
- inverse object properties (InverseObjectProperties)
- functional and inverse-functional object properties (FunctionalObjectProperty and InverseFunctionalObjectProperty)
- symmetric object properties (SymmetricObjectProperty)
- asymmetric object properties (AsymmetricObjectProperty)
OWL2-QL Profile: Axiomes supported

- subclass axioms (SubClassOf)
- class expression equivalence (EquivalentClasses)
- class expression disjointness (DisjointClasses)
- inverse object properties (InverseObjectProperties)
- property inclusion (SubObjectPropertyOf not involving property chains and SubDataPropertyOf)
- property equivalence (EquivalentObjectProperties and EquivalentDataProperties)
- property domain (ObjectPropertyDomain and DataPropertyDomain)
- disjoint properties (DisjointObjectProperties and DisjointDataProperties)
- symmetric properties (SymmetricObjectProperty)
- assertions other than the equality assertions (DifferentIndividuals, ClassAssertion, ObjectPropertyAssertion, and DataPropertyAssertion)

OWL2-QL Profile: Constructs NOT supported

- existential quantification to a class expression or a data range (ObjectSomeValuesFrom and DataSomeValuesFrom in the subclass position)
- self-restriction (ObjectExistsSelf)
- existential quantification to an individual or a literal (ObjectHasValue, DataHasValue)
- nominals (ObjectOneOf, DataOneOf)
- universal quantification to a class expression or a data range (ObjectAllValuesFrom, DataAllValuesFrom)
- cardinality restrictions (ObjectMaxCardinality, ObjectMinCardinality, ObjectExactCardinality, DataMaxCardinality, DataMinCardinality, DataExactCardinality)
- disjunction (ObjectUnionOf, DisjointUnion)
- property inclusions (SubObjectPropertyOf involving property chains)
- functional and inverse-functional properties (FunctionalObjectProperty, InverseFunctionalObjectProperty, and FunctionalDataProperty)
- transitive properties (TransitiveObjectProperty)
- reflexive properties (ReflexiveObjectProperty)
- irreflexive properties (IrreflexiveObjectProperty)
- asymmetric properties (AsymmetricObjectProperty)
- keys (HasKey)

OWL2-RL Profile

- **OWL2 RL:**
  - is based on Description Logic Programs (DLP), which has an expressivity that subsets that of OWL2 DL (the fragment that can be handled using a description logic).
  - enables the implementation of polynomial time reasoning algorithms using rule-extended database technologies operating directly on RDF triples
  - particularly suitable for applications where:
    - relatively lightweight ontologies are used to organize large numbers of individuals
    - it is useful or necessary to operate directly on data in the form of RDF triples

OWL2-RL Profile specificities

- When using **OWL2 RL**, a rule-based implementation can operate directly on RDF triples and so can be applied to an arbitrary RDF graph, i.e., to any OWL2 ontology.
- In this case, reasoning:
  - will always be sound (that is, only correct answers to queries will be computed),
  - but it may not be complete (that is, it is not guaranteed that all correct answers to queries will be computed).
- However, in general, when the ontology is consistent with the structural definition of OWL2 RL, a suitable rule-based implementation performing ground atomic queries will be both sound and complete.
OWL2-RL Profile: Axiomes supported

OWL2 RL supports all axioms of OWL 2

Except:

- disjoint unions of classes (DisjointUnion)
- reflexive object property axioms (ReflexiveObjectProperty)
- negative object and data property assertions:
  - NegativeObjectPropertyAssertion
  - NegativeDataPropertyAssertion.

Limits of OWL1 and OWL2

- The OWL2 (DL) is a decidable fragment of first order predicate logic, with some decidable extensions that go beyond first order
- OWL1 cannot express the uncle relation, which is a chain of relations’ parent and sibling.
- OWL2 can express uncle relation using property chains, however it still cannot express relations between individuals referenced by properties.
- OWL 2 cannot express the child of married parents concept because it cannot express the relationship between parents of the individual (see A better uncle for OWL article).

6. Conclusion

- Limits of OWL1 and OWL2

OWL2 profiles

- A profile is a fragment or a sub-language in computational logic, is a trimmed down version of OWL 2 (DL) that trades some expressive power for the efficiency of reasoning
- 3 OWL2 profiles:
  - EL: polynomial time reasoning: for large ontologies
  - QL: LOGSPACE reasoning: for « database » applications
  - RL: polynomial time reasoning with a rule-based (database) system