The Web Ontology Language OWL2

M. Koubarakis, G. Santipantakis

January 13, 2025

Table of Contents

The Web Ontology Language (OWL)

- An Ontology is an explicit, formal specification of a shared conceptualization. Ontologies are formal models that describe a certain domain and specify the definitions of terms by describing their relationships with other terms in the ontology.

The Web Ontology Language (OWL)

- An Ontology is an explicit, formal specification of a shared conceptualization. Ontologies are formal models that describe a certain domain and specify the definitions of terms by describing their relationships with other terms in the ontology.
- Web Ontology Language (OWL): a DL-based language to describe ontologies.

The Web Ontology Language (OWL)

- An Ontology is an explicit, formal specification of a shared conceptualization. Ontologies are formal models that describe a certain domain and specify the definitions of terms by describing their relationships with other terms in the ontology.
- Web Ontology Language (OWL): a DL-based language to describe ontologies.
- As a DL-based language:

The Web Ontology Language (OWL)

- An Ontology is an explicit, formal specification of a shared conceptualization. Ontologies are formal models that describe a certain domain and specify the definitions of terms by describing their relationships with other terms in the ontology.
- Web Ontology Language (OWL): a DL-based language to describe ontologies.
- As a DL-based language:
  - different language profiles based on expressiveness are possible

The Web Ontology Language (OWL)

- An Ontology is an explicit, formal specification of a shared conceptualization. Ontologies are formal models that describe a certain domain and specify the definitions of terms by describing their relationships with other terms in the ontology.
- Web Ontology Language (OWL): a DL-based language to describe ontologies.
- As a DL-based language:
  - different language profiles based on expressiveness are possible
  - ontologies comprise a TBox, an RBox and ABox

The Web Ontology Language (OWL)

- An Ontology is an explicit, formal specification of a shared conceptualization. Ontologies are formal models that describe a certain domain and specify the definitions of terms by describing their relationships with other terms in the ontology.
- Web Ontology Language (OWL): a DL-based language to describe ontologies.
- As a DL-based language:
  - different language profiles based on expressiveness are possible
  - ontologies comprise a TBox, an RBox and ABox
  - sound and complete algorithms for the reasoning tasks are available

OWL Overview

- OWL ($\mathcal{SHOIQ}(\mathcal{D})$) W3C Recommendation (2004)

OWL Overview

- OWL ($\mathcal{SHOIQ}(\mathcal{D})$) W3C Recommendation (2004)
- OWL2 ($\mathcal{SROIQ}(\mathcal{D})$) W3C Recommendation (2009)

OWL Overview

- OWL ($\mathcal{SHOIQ}(\mathcal{D})$) W3C Recommendation (2004)
- OWL2 ($\mathcal{SROIQ}(\mathcal{D})$) W3C Recommendation (2009)
- an OWL ontology consists of classes, properties and individuals

OWL Overview

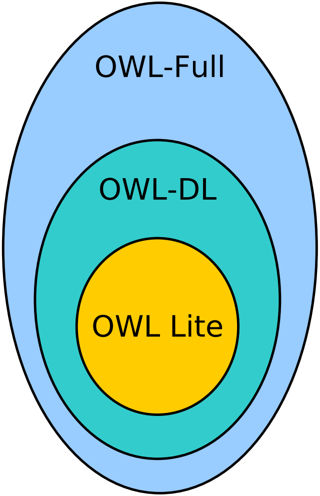
- OWL ($\mathcal{SHOIQ}(\mathcal{D})$) W3C Recommendation (2004)
- OWL2 ($\mathcal{SROIQ}(\mathcal{D})$) W3C Recommendation (2009)
- an OWL ontology consists of classes, properties and individuals
- No Unique Name Assumption:
  - e.g., `:Mary :hasChild :John ; :hasChild :kid1`
    :John and :kid1 may refer to the same entity

OWL Overview

- OWL ($\mathcal{SHOIQ}(\mathcal{D})$) W3C Recommendation (2004)
- OWL2 ($\mathcal{SROIQ}(\mathcal{D})$) W3C Recommendation (2009)
- an OWL ontology consists of classes, properties and individuals
- No Unique Name Assumption:
  - e.g., `:Mary :hasChild :John ; :hasChild :kid1`
    :John and :kid1 may refer to the same entity
- Open World Assumption:
  - absence of information must not be considered as negative information,
    e.g. `:Mary :hasChild :John`
    does not entail that Mary has only one child

OWL profiles (sublanguages) are syntactic restrictions:

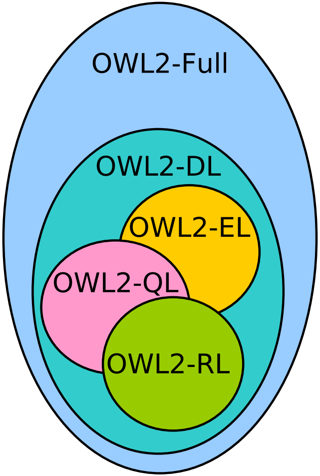
- **OWL Lite:** designed for easy implementation with a functional subset of OWL.
- **OWL DL:** designed to support the existing Description Logic and provide a language subset that has desirable computational properties for reasoning systems.
- **OWL Full:** relaxes some of the constraints on OWL DL for maximum expressiveness, but which violate the constraints of Description Logic reasoners.



OWL Overview (cont'd)

OWL2 profiles (sublanguages), each profile is more restrictive than OWL DL:

- **OWL 2 EL** enables polynomial time algorithms for all the standard reasoning tasks
- **OWL 2 QL** enables conjunctive queries to be answered in LogSpace using standard relational database technology
- **OWL 2 RL** enables the implementation of polynomial time reasoning algorithms using rule-extended database technologies operating directly on RDF triples



OWL 1 is DL based on $\mathcal{SHOIN}(\mathcal{D})$

- $\mathcal{ALC}$ constructors ($\sqcap, \sqcup, \neg, \forall R.C, \exists R.C$) and property transitivity ($\mathcal{S}$),

OWL 1 is DL based on $\mathcal{SHOIN}(\mathcal{D})$

- $\mathcal{ALC}$ constructors $(\sqcap, \sqcup, \neg, \forall R.C, \exists R.C)$ and property transitivity ($\mathcal{S}$),
- **TBox:** concept subsumption e.g., $C \sqsubseteq D$ ($\mathcal{H}$)

OWL 1 is DL based on $\mathcal{SHOIN}(\mathcal{D})$

- $\mathcal{ALC}$ constructors $(\sqcap, \sqcup, \neg, \forall R.C, \exists R.C)$ and property transitivity $(\mathcal{S})$,
- **TBox:** concept subsumption e.g., $C \sqsubseteq D$ $(\mathcal{H})$
- **RBox:** property subsumption e.g., $R \sqsubseteq S$ $(\mathcal{H})$

OWL 1 is DL based on $\mathcal{SHOIN}(\mathcal{D})$

- $\mathcal{ALC}$ constructors $(\sqcap, \sqcup, \neg, \forall R.C, \exists R.C)$ and property transitivity $(\mathcal{S})$,
- **TBox:** concept subsumption e.g., $C \sqsubseteq D$ $(\mathcal{H})$
- **RBox:** property subsumption e.g., $R \sqsubseteq S$ $(\mathcal{H})$
- **TBox:** enumerated classes (nominals) e.g., $\{a\}$ $(\mathcal{O})$

OWL 1 is DL based on $\mathcal{SHOIN}(\mathcal{D})$

- $\mathcal{ALC}$ constructors ($\sqcap, \sqcup, \neg, \forall R.C, \exists R.C$) and property transitivity ($\mathcal{S}$),
- **TBox:** concept subsumption e.g., $C \sqsubseteq D$ ($\mathcal{H}$)
- **RBox:** property subsumption e.g., $R \sqsubseteq S$ ($\mathcal{H}$)
- **TBox:** enumerated classes (nominals) e.g., $\{a\}$ ($\mathcal{O}$)
- **RBox:** inverse properties e.g., R^- ($\mathcal{I}$)

OWL 1 is DL based on $\mathcal{SHOIN}(\mathcal{D})$

- $\mathcal{ALC}$ constructors ($\sqcap, \sqcup, \neg, \forall R.C, \exists R.C$) and property transitivity ($\mathcal{S}$),
- **TBox:** concept subsumption e.g., $C \sqsubseteq D$ ($\mathcal{H}$)
- **RBox:** property subsumption e.g., $R \sqsubseteq S$ ($\mathcal{H}$)
- **TBox:** enumerated classes (nominals) e.g., $\{a\}$ ($\mathcal{O}$)
- **RBox:** inverse properties e.g., R^- ($\mathcal{I}$)
- **TBox:** cardinality restrictions e.g., $\leq nR, \geq nR$ ($\mathcal{N}$)

OWL 1 is DL based on $\mathcal{SHOIN}(\mathcal{D})$

- $\mathcal{ALC}$ constructors $(\sqcap, \sqcup, \neg, \forall R.C, \exists R.C)$ and property transitivity $(\mathcal{S})$,
- **TBox:** concept subsumption e.g., $C \sqsubseteq D$ $(\mathcal{H})$
- **RBox:** property subsumption e.g., $R \sqsubseteq S$ $(\mathcal{H})$
- **TBox:** enumerated classes (nominals) e.g., $\{a\}$ $(\mathcal{O})$
- **RBox:** inverse properties e.g., R^{-} $(\mathcal{I})$
- **TBox:** cardinality restrictions e.g., $\leq nR, \geq nR$ $(\mathcal{N})$
- Datatypes $(\mathcal{D})$

OWL 1 is DL based on $\mathcal{SHOIN}(\mathcal{D})$

- $\mathcal{ALC}$ constructors ($\sqcap, \sqcup, \neg, \forall R.C, \exists R.C$) and property transitivity ($\mathcal{S}$),
- **TBox:** concept subsumption e.g., $C \sqsubseteq D$ ($\mathcal{H}$)
- **RBox:** property subsumption e.g., $R \sqsubseteq S$ ($\mathcal{H}$)
- **TBox:** enumerated classes (nominals) e.g., $\{a\}$ ($\mathcal{O}$)
- **RBox:** inverse properties e.g., R^{-} ($\mathcal{I}$)
- **TBox:** cardinality restrictions e.g., $\leq nR, \geq nR$ ($\mathcal{N}$)
- Datatypes ($\mathcal{D}$)
- **ABox:** class and property assertions e.g., $Student(ST001)$, $hasParent(MARIA, NIKOS)$, instance equality e.g., $ST001 = NIKOS$, and difference e.g., $MARIA \neq NIKOS$

OWL 2 additional features:

- **TBox:** qualified cardinality restrictions e.g., $\leq nR.C, \geq nR.C$ ($\mathcal{Q}$)

OWL 2 is DL based on $\mathcal{SROIQ}(\mathcal{D})$

OWL 2 additional features:

- **TBox:** qualified cardinality restrictions e.g., $\leq nR.C, \geq nR.C$ ($\mathcal{Q}$)
- **TBox:** Self $\exists S.Self$

OWL 2 is DL based on $\mathcal{SROIQ}(\mathcal{D})$

OWL 2 additional features:

- **TBox:** qualified cardinality restrictions e.g., $\leq nR.C, \geq nR.C$ ($\mathcal{Q}$)
- **TBox:** Self $\exists S.Self$
- **RBox:** General Role Inclusion e.g., $R_1 \circ R_2 \circ \ldots \circ R_n \sqsubseteq R$ ($\mathcal{R}$)

OWL 2 is DL based on $\mathcal{SROIQ}(\mathcal{D})$

OWL 2 additional features:

- **TBox:** qualified cardinality restrictions e.g., $\leq nR.C, \geq nR.C$ ($\mathcal{Q}$)
- **TBox:** Self $\exists S.Self$
- **RBox:** General Role Inclusion e.g., $R_1 \circ R_2 \circ ... \circ R_n \sqsubseteq R$ ($\mathcal{R}$)
- **RBox:** symmetry, reflexivity, irreflexivity, disjunctiveness

OWL 2 is DL based on $\mathcal{SROIQ}(\mathcal{D})$

OWL 2 additional features:

- **TBox:** qualified cardinality restrictions e.g., $\leq nR.C, \geq nR.C$ ($\mathcal{Q}$)
- **TBox:** Self $\exists S.Self$
- **RBox:** General Role Inclusion e.g., $R_1 \circ R_2 \circ \ldots \circ R_n \sqsubseteq R$ ($\mathcal{R}$)
- **RBox:** symmetry, reflexivity, irreflexivity, disjunctiveness
- **ABox:** negated property assertions e.g., $\neg hasPet(NIKOS, PLUTO)$

OWL in a nutshell

- Entities, expressions and axioms form the logical part of OWL 2. They can be given a precise semantics and inferences can be drawn from them.

OWL in a nutshell

- Entities, expressions and axioms form the logical part of OWL 2. They can be given a precise semantics and inferences can be drawn from them.
- Entities, axioms, and ontologies can be annotated e.g., a class can be given a human-readable rdfs:label, a rdfs:seeAlso reference or rdf:comment (annotations have no effect on the logical aspects of an ontology)

OWL in a nutshell

- Entities, expressions and axioms form the logical part of OWL 2. They can be given a precise semantics and inferences can be drawn from them.
- Entities, axioms, and ontologies can be annotated e.g., a class can be given a human-readable rdfs:label, a rdfs:seeAlso reference or rdf:comment (annotations have no effect on the logical aspects of an ontology)
- The ontology itself also has an IRI, it can be annotated (owl:version, rdfs:label, rdfs:seeAlso, etc) and reused (owl:import)

OWL in a nutshell (cont'd)

OWL 2 ontologies include the following:

- classes: representing sets of elements in the domain,

OWL in a nutshell (cont'd)

OWL 2 ontologies include the following:

- classes: representing sets of elements in the domain,
- properties: distinguished to ObjectProperty and DataProperty (the former relates instances to each other, the latter relates instances to data values),

OWL in a nutshell (cont'd)

OWL 2 ontologies include the following:

- classes: representing sets of elements in the domain,
- properties: distinguished to ObjectProperty and DataProperty (the former relates instances to each other, the latter relates instances to data values),
- instances (or individuals) representing entities in the domain.

OWL in a nutshell (cont'd)

OWL 2 ontologies include the following:

- classes: representing sets of elements in the domain,
- properties: distinguished to ObjectProperty and DataProperty (the former relates instances to each other, the latter relates instances to data values),
- instances (or individuals) representing entities in the domain.
- Expressions: describing complex classes of elements in the domain (i.e. complex concepts or roles in DL terms).

OWL in a nutshell (cont'd)

OWL 2 ontologies include the following:

- classes: representing sets of elements in the domain,
- properties: distinguished to ObjectProperty and DataProperty (the former relates instances to each other, the latter relates instances to data values),
- instances (or individuals) representing entities in the domain.
- Expressions: describing complex classes of elements in the domain (i.e. complex concepts or roles in DL terms).
- Axioms are statements that are asserted to be true (e.g., a subclass axiom)

OWL in a nutshell (cont'd)

OWL 2 ontologies include the following:

- classes: representing sets of elements in the domain,
- properties: distinguished to ObjectProperty and DataProperty (the former relates instances to each other, the latter relates instances to data values),
- instances (or individuals) representing entities in the domain.
- Expressions: describing complex classes of elements in the domain (i.e. complex concepts or roles in DL terms).
- Axioms are statements that are asserted to be true (e.g., a subclass axiom)
- DL reasoners can be employed to draw inferences from asserted knowledge

OWL Syntax Variants

Serializers and Parsers are available for the following syntax variants:

- Functional syntax (see also: https://www.w3.org/TR/owl2-primer/)

OWL Syntax Variants

Serializers and Parsers are available for the following syntax variants:

- Functional syntax (see also: https://www.w3.org/TR/owl2-primer/)
- An extension of existing RDF/XML

OWL Syntax Variants

Serializers and Parsers are available for the following syntax variants:

- Functional syntax (see also: https://www.w3.org/TR/owl2-primer/)
- An extension of existing RDF/XML
- An independent XML serialization (OWL/XML)

OWL Syntax Variants

Serializers and Parsers are available for the following syntax variants:

- Functional syntax (see also: https://www.w3.org/TR/owl2-primer/)
- An extension of existing RDF/XML
- An independent XML serialization (OWL/XML)
- Manchester syntax, also used in Protege (see also: https://www.w3.org/TR/owl2-manchester-syntax/)

OWL Syntax Variants

Serializers and Parsers are available for the following syntax variants:

- Functional syntax (see also: https://www.w3.org/TR/owl2-primer/)
- An extension of existing RDF/XML
- An independent XML serialization (OWL/XML)
- Manchester syntax, also used in Protege (see also: https://www.w3.org/TR/owl2-manchester-syntax/)
- Turtle

Example (RDF/XML Syntax)

$$Parent \equiv \exists hasChild.Person$$

```
...
<owl:Class rdf:about="http://example.gr#Parent">
    <owl:equivalentClass>
        <owl:Restriction>
    <owl:onProperty rdf:resource="http://example.gr#hasChild"/>
            <owl:someValuesFrom rdf:resource="http://example.gr#Person"/>
        </owl:Restriction>
    </owl:equivalentClass>
</owl:Class>
...
```

Example (OWL/XML Syntax)

$$Parent \equiv \exists hasChild.Person$$

```
...
<Declaration>
    <Class IRI="#Parent"/>
</Declaration>
<Declaration>
    <Class IRI="#Person"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#hasChild"/>
</Declaration>
<EquivalentClasses>
    <Class IRI="#Parent"/>
    <ObjectSomeValuesFrom>
        <ObjectProperty IRI="#hasChild"/>
        <Class IRI="#Person"/>
    </ObjectSomeValuesFrom>
</EquivalentClasses>
...
```

Example (Functional/Manchester Syntax)

$$Parent \equiv \exists hasChild.Person$$

Functional Syntax:

```
Declaration(Class(<http://example.gr#Parent>))
Declaration(Class(<http://example.gr#Person>))
Declaration(ObjectProperty(<http://example.gr#hasChild>))
EquivalentClasses(<http://example.gr#Parent>
    ObjectSomeValuesFrom(<http://example.gr#hasChild>
        <http://example.gr#Person>))
```

Manchester Syntax:

```
ObjectProperty: <http://example.gr#hasChild>
Class: <http://example.gr#Person>
Class: <http://example.gr#Parent>
    EquivalentTo:
        <http://example.gr#hasChild> some
            <http://example.gr#Person>
```

Example (Turtle Syntax)

$$Parent \equiv \exists hasChild.Person$$

Turtle Syntax:

```
:Person rdf:type owl:Class .
:hasChild rdf:type owl:ObjectProperty .
:Parent rdf:type owl:Class ;
    owl:equivalentClass [ rdf:type owl:Restriction ;
            owl:onProperty :hasChild ;
            owl:someValuesFrom :Person
        ] .
```

OWL Classes

OWL classes represent sets of individuals.

Predefined classes in OWL are:

- `owl:Thing`, which represents the set of all individuals
- `owl:Nothing`, which represents the empty set

OWL ObjectProperty

OWL ObjectProperties connect pairs of individuals, e.g.,

```
:parentOf(:Homer :Bart)
```

Built-in object properties:

- `owl:topObjectProperty`, which connects all possible pairs of individuals.
- `owl:bottomObjectProperty`, which does not connect any pair of individuals.

Object properties can be used to form object property expressions.

OWL ObjectProperty Expression (Inverse)

An inverse object property expression `ObjectInverseOf(P)` connects an individual I1 with I2 if and only if the object property P connects I2 with I1.

Example: if an ontology contains:

```
ObjectPropertyAssertion(:parentOf :Homer :Bart)
```

then it also entails:

```
ObjectPropertyAssertion(ObjectInverseOf(:fatherOf)
      :Bart :Homer)
```

OWL ObjectProperty Expression (Symmetric/Asymmetric)

In some cases, a property and its inverse coincide, or in other words, the direction of a property doesn't matter.

Example: if an ontology contains:

```
SymmetricObjectProperty( :hasSpouse )
ObjectPropertyAssertion(:hasSpouse :Bart :Lisa)
```

then it also entails:

```
ObjectPropertyAssertion(:hasSpouse :Lisa :Bart)
```

A property can also be asymmetric, if it connects A with B, but it never connects B with A.

Example:

```
AsymmetricObjectProperty( :hasChild )
```

OWL ObjectProperty Expression (Reflexivity/Irreflexivity)

Reflexive properties relate everything to itself.
Example: Everyone is a relative to him/herself:
`ReflexiveObjectProperty( :hasRelative )`
Note: this does not necessarily mean that every two individuals which are related by a reflexive property are identical.
Irreflexive properties model the case where no individual can be related to itself by such a property.
Example:
`IrreflexiveObjectProperty( :parentOf )`
Nobody can be his own parent.

OWL ObjectProperty Expression (Functional)

Some properties relate a subject to at most one object. These properties are called functional.

Example: If an ontology contains

```
FunctionalObjectProperty( :hasHusband )
ObjectPropertyAssertion(:hasHusband :Marge :Homer)
ObjectPropertyAssertion(:hasHusband :Marge :HomerSimpson)
```

it also entails:

SameIndividual(:Homer :HomerSimpson)

Note: this expression does not require every individual to have a husband, it only states that there can be no more than one.

OWL ObjectProperty Expression (Inverse Functional)

It is also possible to indicate that the inverse of a given property is functional.

Example: If an ontology contains

```
InverseFunctionalObjectProperty( :hasHusband )
ObjectPropertyAssertion(:hasHusband :Marge :Homer)
ObjectPropertyAssertion(:hasHusband :MargeBouvier :Homer)
```

it also entails:

```
ObjectPropertyAssertion(owl:sameAs :Marge :MargeBouvier)
```

This indicates that if two or more individuals are related with the same individual via an inverse functional property, then these individuals refer to the same entity in the domain.

OWL ObjectProperty Expression (Transitive)

A transitive property R interlinks an individual i with j, whenever R relates i with k, and k with j.

Example: If an ontology contains

```
TransitiveObjectProperty( :hasAncestor )
ObjectPropertyAssertion(:hasAncestor :Marge :ClancyBouvier)
ObjectPropertyAssertion(:hasAncestor :Lisa :Marge)
```

it also entails:

```
ObjectPropertyAssertion(:hasAncestor :Lisa :ClancyBouvier)
```

OWL DataProperty

Data properties (e.g., :hasAge) connect individuals with literals.
Built-in properties:

- `owl:topDataProperty`, which connects all possible individuals with all literals.
- `owl:bottomDataProperty`, which does not connect any individual with a literal.

OWL Annotation Properties

Annotation properties can be used to provide an annotation for an ontology, axiom, or a resource. Users can define their own annotation properties or use the available built-in annotation properties:

- `rdfs:label`, `rdfs:comment`, `rdfs:seeAlso`, `rdfs:isDefinedBy`
- `owl:deprecated`, `owl:versionInfo`, `owl:priorVersion`, `owl:backwardCompatibleWith`, `owl:incompatibleWith`

OWL Individuals

Individuals represent actual objects in the domain. There are two types of individuals:

- Named individuals are given an explicit name (an IRI e.g., `:Peter`) that can be used in any ontology to refer to the same object.

OWL Individuals

Individuals represent actual objects in the domain. There are two types of individuals:

- Named individuals are given an explicit name (an IRI e.g., `:Peter`) that can be used in any ontology to refer to the same object.
- Anonymous individuals do not have a global name. They can be defined using a name (e.g., `_:somebody`) local to the ontology they are contained in. They are like blank nodes in RDF.

OWL Datatypes

Datatypes are entities that represent sets of data values.

- OWL 2 offers a rich set of data types: decimal numbers, integers, floating point numbers, rationals, reals, strings, binary data, IRIs and time instants.

OWL Datatypes

Datatypes are entities that represent sets of data values.

- OWL 2 offers a rich set of data types: decimal numbers, integers, floating point numbers, rationals, reals, strings, binary data, IRIs and time instants.
- In most cases, these data types are taken from XML schema. From RDF and RDFS, we have `rdf:XMLLiteral`, `rdf:PlainLiteral` and `rdfs:Literal`.

OWL Datatypes

Datatypes are entities that represent sets of data values.

- OWL 2 offers a rich set of data types: decimal numbers, integers, floating point numbers, rationals, reals, strings, binary data, IRIs and time instants.
- In most cases, these data types are taken from XML schema. From RDF and RDFS, we have `rdf:XMLLiteral`, `rdf:PlainLiteral` and `rdfs:Literal`.
- `rdfs:Literal` contains all the elements of other data types.

OWL Datatypes

Datatypes are entities that represent sets of data values.

- OWL 2 offers a rich set of data types: decimal numbers, integers, floating point numbers, rationals, reals, strings, binary data, IRIs and time instants.
- In most cases, these data types are taken from XML schema. From RDF and RDFS, we have `rdf:XMLLiteral`, `rdf:PlainLiteral` and `rdfs:Literal`.
- `rdfs:Literal` contains all the elements of other data types.
- There are also the OWL datatypes `owl:real` and `owl:rational`.

OWL Datatypes (cont'd)

- Formally, the data types supported are specified in the OWL 2 datatype map, where each datatype is identified by an IRI and is defined by the following components:

OWL Datatypes (cont'd)

- Formally, the data types supported are specified in the OWL 2 datatype map, where each datatype is identified by an IRI and is defined by the following components:
  - The value space is the set of values of the datatype. Elements of the value space are called data values.

OWL Datatypes (cont'd)

- Formally, the data types supported are specified in the OWL 2 datatype map, where each datatype is identified by an IRI and is defined by the following components:
  - The value space is the set of values of the datatype. Elements of the value space are called data values.
  - The lexical space is a set of strings that can be used to refer to data values. Each member of the lexical space is called a lexical form, and it is mapped to a particular data value.

OWL Datatypes (cont'd)

- Formally, the data types supported are specified in the OWL 2 datatype map, where each datatype is identified by an IRI and is defined by the following components:
  - The value space is the set of values of the datatype. Elements of the value space are called data values.
  - The lexical space is a set of strings that can be used to refer to data values. Each member of the lexical space is called a lexical form, and it is mapped to a particular data value.
  - The facet space is a set of pairs of the form (F,v) where F is an IRI called a constraining facet, and v is an arbitrary data value called the constraining value. Each such pair is mapped to a subset of the value space of the datatype.

OWL Datatypes (Example)

We can define a new datatype for a person's age by constraining the datatype integer to values between (inclusively) 0 and 120.

```
DatatypeDefinition( :personAge
    DatatypeRestriction(
    xsd:integer
        xsd:minInclusive "0"^^xsd:integer
        xsd:maxInclusive "120"^^xsd:integer
    )
)
```

OWL Literals

Literals represent data values such as particular strings or integers. They are analogous to RDF literals.

Examples:

`"1"^^xsd:integer` (typed literal)

`"Family Guy"` (plain literal, an abbreviation for `"Family Guy"^^rdf:PlainLiteral`)

`"Padre de familia"@es` (plain literal with language tag)

OWL Class Expressions

- Class names and property expressions can be used to construct class expressions.

OWL Class Expressions

- Class names and property expressions can be used to construct class expressions.
- These are essentially the complex concepts or descriptions that we can define in DLs.

OWL Class Expressions

- Class names and property expressions can be used to construct class expressions.
- These are essentially the complex concepts or descriptions that we can define in DLs.
- Class expressions represent sets of individuals by formally specifying conditions on the individuals' properties; individuals satisfying these conditions are said to be instances of the respective class expressions.

OWL Class Expressions (cont'd)

Class expressions can be formed by:

- Applying the standard Boolean connectives to simpler class expressions or by enumerating the individuals that belong to an expression.

OWL Class Expressions (cont'd)

Class expressions can be formed by:

- Applying the standard Boolean connectives to simpler class expressions or by enumerating the individuals that belong to an expression.
- Placing restrictions on object property expressions.

OWL Class Expressions (cont'd)

Class expressions can be formed by:

- Applying the standard Boolean connectives to simpler class expressions or by enumerating the individuals that belong to an expression.
- Placing restrictions on object property expressions.
- Placing restrictions on the cardinality of object property expressions.

OWL Class Expressions (cont'd)

Class expressions can be formed by:

- Applying the standard Boolean connectives to simpler class expressions or by enumerating the individuals that belong to an expression.
- Placing restrictions on object property expressions.
- Placing restrictions on the cardinality of object property expressions.
- Placing restrictions on data property expressions.

OWL Class Expressions (cont'd)

Class expressions can be formed by:

- Applying the standard Boolean connectives to simpler class expressions or by enumerating the individuals that belong to an expression.
- Placing restrictions on object property expressions.
- Placing restrictions on the cardinality of object property expressions.
- Placing restrictions on data property expressions.
- Placing restrictions on the cardinality of data property expressions.

OWL Class Expressions (cont'd)

An intersection class expression `ObjectIntersectionOf(CE_1 ... CE_n)` contains all individuals that are instances of all class expressions `CE_i` for $1 \leq i \leq n$.

Example:

`ObjectIntersectionOf(:Dog :CanTalk)`

OWL Class Expressions (cont'd)

A union class expression `ObjectUnionOf(CE_1 ... CE_n)` contains all individuals that are instances of at least one class expression `CE_i` for $1 \leq i \leq n$.

Example:

`ObjectUnionOf(:Man :Woman)`

OWL Class Expressions (cont'd)

A complement class expression `ObjectComplementOf(CE)` contains all individuals that are not instances of the class expression `CE`.

Example:

```
ObjectComplementOf( :Man )
```

OWL Class Expressions (cont'd)

We can define disjoint classes, i.e. those sets that cannot have a common element.

Example: If an ontology contains:

```
DisjointClasses(:Man :Woman)
ClassAssertion(:Woman :Marge)
```

it also entails:

```
ClassAssertion(ObjectComplementOf(:Man) :Marge)
```

OWL Class Expressions (cont'd)

An enumeration of individuals `ObjectOneOf(a_1 ... a_n)` contains exactly the individuals `a_i` with $1 \leq i \leq n$.

Example:

`ObjectOneOf(:Saturday :Sunday)`

Example Inference

From

```
EquivalentClasses(:GriffinFamilyMember
   ObjectOneOf(:Peter :Lois :Stewie :Meg :Chris :Brian))
DifferentIndividuals(:Quagmire :Peter :Lois :Stewie :Meg
    :Chris :Brian)
```

we can infer `ClassAssertion(`
` ObjectComplementOf(:GriffinFamilyMember) :Quagmire)`

Example Inference (cont'd)

From

```
ClassAssertion(:GriffinFamilyMember :Peter)
ClassAssertion(:GriffinFamilyMember :Lois)
ClassAssertion(:GriffinFamilyMember :Stewie)
ClassAssertion(:GriffinFamilyMember :Meg)
ClassAssertion(:GriffinFamilyMember :Chris)
ClassAssertion(:GriffinFamilyMember :Brian)

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

Can we infer this:

```
ClassAssertion(
ObjectComplementOf(:GriffinFamilyMember) :Quagmire) ?
```

OWL Class Expressions (cont'd)

An existential class expression `ObjectSomeValuesFrom(OPE CE)` consists of an object property expression `OPE` and a class expression `CE`, and it contains all those individuals that are connected by `OPE` to an individual that is an instance of `CE`.

Example:

`ObjectSomeValuesFrom(:hasChild :Person)`

If `OPE` is simple, the above class expression is equivalent with the class expression `ObjectMinCardinality(1 OPE CE)`

Example

From

```
ObjectPropertyAssertion(:hasChild :Peter :Stewie)
ClassAssertion(:Person :Stewie)
```

we can infer

```
ClassAssertion(
ObjectSomeValuesFrom(:hasChild :Person) :Peter)
```

OWL Class Expressions (cont'd)

A universal class expression `ObjectAllValuesFrom(OPE CE)` consists of an object property expression `OPE` and a class expression `CE`, and it contains all those individuals that are connected by `OPE` only to individuals that are instances of `CE`.

Example:

`ObjectAllValuesFrom(:fatherOf :Man)`

If `OPE` is simple, the above class expression is equivalent with the class expression `ObjectMaxCardinality(0 OPE ObjectComplementOf(CE))`

OWL Class Expressions (cont'd)

An individual value class expression `ObjectHasValue(OPE a)` consists of an object property expression `OPE` and an individual a, and it contains all those individuals that are connected by `OPE` to a.

Example: If an ontology contains:

```
EquivalentClasses(:SolarPlanet ObjectHasValue(:orbits :Sun))
ObjectPropertyAssertion(:orbits :Earth :Sun))
```

it can also entail:

```
ClassAssertion(:SolarPlanet :Earth)
```

The above class expression is equivalent to the class expression `ObjectSomeValuesFrom(OPE ObjectOneOf(a))`.

OWL Class Expressions (cont'd)

A self-restriction `ObjectHasSelf(OPE)` consists of an object property expression `OPE`, and it contains all those individuals that are connected by `OPE` to themselves.

Example: if an ontology contains:

```
EquivalentClasses(:Narcisist ObjectHasSelf(:likes))
ObjectPropertyAssertion(:likes :Peter :Peter)
```

it also infers:

```
ClassAssertion(:Narcisist :Peter)
```

OWL Class Expressions (cont'd)

Object property cardinality restrictions are distinguished into:

- Qualified: apply only to individuals that are connected by the object property expression and are instances of the qualifying class expression. (e.g. `>3hasChild.Male`)
- Unqualified: apply to all individuals that are connected by the object property expression (this is equivalent to the qualified case with the qualifying class expression equal to owl:Thing) (e.g. `>3hasChild`).

OWL Class Expressions (cont'd)

A minimum cardinality expression `ObjectMinCardinality(n OPE CE)` consists of a nonnegative integer `n`, an object property expression `OPE`, and a class expression `CE`, and it contains all those individuals that are connected by `OPE` to at least `n` different individuals that are instances of `CE`. If `CE` is missing, it is taken to be `owl:Thing`.

Example:

`ObjectMinCardinality(2 :fatherOf :Man)`

Example

From

```
ObjectPropertyAssertion(:fatherOf :Peter :Stewie)
ObjectPropertyAssertion(:fatherOf :Peter :Chris)
ClassAssertion(:Man :Stewie)
ClassAssertion(:Man :Chris)
DifferentIndividuals(:Chris :Stewie)
```

we can infer:

```
ClassAssertion(ObjectMinCardinality(2 :fatherOf :Man) :Peter)
```

OWL Class Expressions (cont'd)

A maximum cardinality expression `ObjectMaxCardinality(n OPE CE)` consists of a nonnegative integer `n`, an object property expression `OPE`, and a class expression `CE`, and it contains all those individuals that are connected by `OPE` to at most `n` different individuals that are instances of `CE`. If `CE` is missing, it is taken to be `owl:Thing`.

Example:

```
ObjectMaxCardinality(2 :hasPet)
```

OWL Class Expressions (cont'd)

An exact cardinality expression `ObjectExactCardinality(n OPE CE)` consists of a nonnegative integer `n`, an object property expression `OPE`, and a class expression `CE`, and it contains all those individuals that are connected by `OPE` to exactly `n` different individuals that are instances of `CE`.

Example:

`ObjectExactCardinality(1 :hasPet :Dog)`

The above expression is equivalent to

```
ObjectIntersectionOf(
   ObjectMinCardinality(n OPE CE)
   ObjectMaxCardinality(n OPE CE))
```

Data Property restrictions

- Data property restrictions are similar to the restrictions on object property expressions.
- The main difference is that the expressions for existential and universal quantification allow for n-ary data ranges.
- Given the syntax for data ranges given earlier, only unary data ranges are supported.
- However, the specification provide the syntactic constructs needed to have n-ary data ranges e.g., sets of rectangles defined by appropriate geometric constraints.

The "Data Range Extension: Linear Equations" W3C note proposes an extension to OWL 2 for defining n-ary data ranges in terms of linear (in)equations with rational coefficients. See `http://www.w3.org/TR/owl2-dr-linear/` .

OWL Axioms

A subclass axiom `SubClassOf(CE1 CE2)` states that the class expression `CE1` is a subclass of the class expression `CE2`.

Example:

`SubClassOf(:Child :Person)`

The properties known from RDFS for SubClassOf hold here as well (Reflexivity, Transitivity)

OWL Axioms

An equivalent classes axiom `EquivalentClasses(CE_1 ... CE_n)` states that all of the class expressions `CE_i`,$1 \leq i \leq n$, are semantically equivalent to each other.

Example:

`EquivalentClasses(:Boy ObjectIntersectionOf(:Child :Male))`

An axiom `EquivalentClasses(CE1 CE2)` is equivalent to the conjunction of the following two axioms: `SubClassOf(CE1 CE2)` `SubClassOf(CE2 CE1)`

OWL Axioms

A disjoint classes axiom `DisjointClasses(CE_1 ... CE_n)` states that all of the class expressions `CE_i`, $1 \leq i \leq n$, are pairwise disjoint.

Example:

`DisjointClasses(:Boy :Girl)`

An axiom `DisjointClasses(CE1 CE2)` is equivalent to the following axiom:

`SubClassOf(CE1 ObjectComplementOf(CE2))`

OWL Axioms

A disjoint union axiom `DisjointUnion(C CE_1 ... CE_n)` states that a class `C` is a disjoint union of the class expressions `CE_i`, $1 \leq i \leq n$, all of which are pairwise disjoint.

Such axioms are sometimes referred to as covering axioms, as they state that the extensions of all `CE_i` exactly cover the extension of `C`.

Example:

`DisjointUnion(:Child :Boy :Girl)`

Each such axiom is equivalent to the conjunction of the following two axioms: `EquivalentClasses(C ObjectUnionOf(CE1 ... CEn))`
`DisjointClasses(CE1 ... CEn)`

Example

From

```
DisjointUnion(:Child :Boy :Girl)
ClassAssertion(:Child :Stewie)
ClassAssertion(ObjectComplementOf(:Girl) :Stewie)
```

we can infer

```
ClassAssertion(:Boy :Stewie)
```

OWL Axioms

- Object subproperty axioms are analogous to subclass axioms.
- The basic form of an object subproperty axiom is `SubObjectPropertyOf(OPE1 OPE2)`.
- This axiom states that the object property expression `OPE1` is a subproperty of the object property expression `OPE2` i.e. if an individual x is connected by `OPE1` to an individual y, then x is also connected by `OPE2` to y.
- `SubObjectPropertyOf` is a reflexive and transitive relation.

OWL Axioms (Property Chain)

If `OPE1, ..., OPEn` are object properties then `OPE1 ... OPEn` is called an object property chain.

The more complex form of object subproperty axioms is
`SubObjectPropertyOf(`
`ObjectPropertyChain(OPE1 ... OPEn) OPE)`.

This axiom states that, if an individual x1 is connected by a sequence of object property expressions `OPE1, ..., OPEn` with an individual `xn`, then `x1` is also connected with `xn` by the object property expression `OPE`.

These axioms are known as complex role inclusions in the DL literature.

Example

From

```
SubObjectPropertyOf(
ObjectPropertyChain(:hasMother :hasSister) :hasAunt)

ObjectPropertyAssertion(:hasMother :Stewie :Lois)
ObjectPropertyAssertion(:hasSister :Lois :Carol)
```

we can infer

```
ObjectPropertyAssertion(:hasAunt :Stewie :Carol)
```

OWL Axioms

An equivalent object properties axiom `EquivalentObjectProperties(OPE_1 ... OPE_n)` states that all of the object property expressions `OPE_i`, $1 \leq i \leq n$, are semantically equivalent to each other.

The axiom `EquivalentObjectProperties(OPE1 OPE2)` is equivalent to the following two axioms:

```
SubObjectPropertyOf(OPE1 OPE2)
SubObjectPropertyOf(OPE2 OPE1)
```

A disjoint object properties axiom `DisjointObjectProperties(OPE1 ... OPEn)` states that all of the object property expressions `OPE_i`, $1 \leq i \leq n$, are pairwise disjoint.

Example:

`DisjointObjectProperties(:hasFather :hasMother)`

OWL Axioms

- An object property domain axiom `ObjectPropertyDomain(OPE CE)` states that the domain of the object property expression `OPE` is the class expression `CE` i.e. if an individual x is connected by `OPE` with some other individual, then x is an instance of `CE`.
- An object property range axiom `ObjectPropertyRange(OPE CE)` states that the range of the object property expression `OPE` is the class expression `CE` i.e. if some individual is connected by `OPE` with an individual x, then x is an instance of `CE`.

Assertions

OWL 2 supports a rich set of axioms for stating assertions about individuals:

- Individual equality

Assertions

OWL 2 supports a rich set of axioms for stating assertions about individuals:

- Individual equality
- Individual inequality

Assertions

OWL 2 supports a rich set of axioms for stating assertions about individuals:

- Individual equality
- Individual inequality
- Class assertion

Assertions

OWL 2 supports a rich set of axioms for stating assertions about individuals:

- Individual equality
- Individual inequality
- Class assertion
- Positive object property assertion

Assertions

OWL 2 supports a rich set of axioms for stating assertions about individuals:

- Individual equality
- Individual inequality
- Class assertion
- Positive object property assertion
- Negative object property assertion

Assertions

OWL 2 supports a rich set of axioms for stating assertions about individuals:

- Individual equality
- Individual inequality
- Class assertion
- Positive object property assertion
- Negative object property assertion
- Positive data property assertion

Assertions

OWL 2 supports a rich set of axioms for stating assertions about individuals:

- Individual equality
- Individual inequality
- Class assertion
- Positive object property assertion
- Negative object property assertion
- Positive data property assertion
- Negative data property assertion

Assertions

OWL 2 supports a rich set of axioms for stating assertions about individuals:

- Individual equality
- Individual inequality
- Class assertion
- Positive object property assertion
- Negative object property assertion
- Positive data property assertion
- Negative data property assertion

Assertions are often also called facts. They are part of the ABox in DLs.

Assertions

An individual equality axiom `SameIndividual(a_1 ... a_n)` states that all of the individuals `a_i`, $1 \leq i \leq n$, are equal to each other.

Example:

From `SameIndividual(:Meg :Megan)`

`ObjectPropertyAssertion(:hasBrother :Meg :Stewie)`

we can infer:

`ObjectPropertyAssertion(:hasBrother :Megan :Stewie)`

Assertions

An individual inequality axiom `DifferentIndividuals(a_1 ... a_n)` states that all of the individuals `a_i`, $1 \leq i \leq n$, are different from each other.

Example:

`DifferentIndividuals(:Peter :Meg :Chris :Stewie)`

Class Assertion

A class assertion `ClassAssertion(CE a)` states that the individual `a` is an instance of the class expression `CE`.

Example:

`ClassAssertion(:Dog :Brian)`

Object Property Assertion

A positive object property assertion `ObjectPropertyAssertion(OPE a_1 a_2)` states that the individual `a_1` is connected by the object property expression `OPE` to the individual `a_2`.

A negative object property assertion `NegativeObjectPropertyAssertion(OPE a_1 a_2)` states that the individual `a_1` is not connected by the object property expression `OPE` to the individual `a_2`.

Examples:

```
ObjectPropertyAssertion(:hasDog :Peter :Brian)
NegativeObjectPropertyAssertion(:hasSon :Peter :Meg)
```

Readings

- OWL 2 Web Ontology Language Primer (Second Edition)
  `https://www.w3.org/TR/owl2-primer/`
- OWL 2 Web Ontology Language Manchester Syntax (Second Edition)
  `https://www.w3.org/TR/owl2-manchester-syntax/`
- Krötzsch, M. (2012). OWL 2 Profiles: An Introduction to Lightweight Ontology Languages. In: Eiter, T., Krennwallner, T. (eds) Reasoning Web. Semantic Technologies for Advanced Query Answering. Reasoning Web 2012. Lecture Notes in Computer Science, vol 7487. Springer, Berlin, Heidelberg.
  `https://doi.org/10.1007/978-3-642-33158-9_4`
- An introduction to OWL 2 and DL $\mathcal{SROIQ}$, which also discusses various OWL tools/applications: Ian Horrocks and Peter F. Patel-Schneider. KR and Reasoning on the Semantic Web: OWL. In Handbook of Semantic Web Technologies, chapter 9. Springer, 2010.
  `http://www.cs.ox.ac.uk/people/ian.horrocks/Publications/download/2010/HoPa10a.pdf`