0.1 Expressive Ontologies

Why RDFS is not enough ?

- RDFS is rich enough for simple conceptual modeling, including resource annotations or property domains and ranges
- ... but it is not enough for describing complex domain relationships

  Examples

  - e.g. `ex:hasAncestor` being transitive, or
  - each `ex:Human` having exactly two biological parents
  - two `ex:Persons` sharing exactly one parent are step-siblings

- for these cases more powerful languages are needed. Here we sketch two of them:

  Web Ontology Language (OWL) is a standard for representing complex ontologies,

  Semantic Web Rule Language (SWRL) is one of possible rule languages built upon OWL.

0.1.1 OWL 2

OWL 2 Basics

- state-of the art ontology modeling language extending RDFS,
- describes *individuals*, their *classes* and *properties*,

Listing 1 : Example OWL ontology in Manchester syntax [2]

```
Prefix: : <http://ex.owl/>
Ontology: <http://ex.owl/o1>
 ObjectProperty: :hasChild
 Class: :Man
 Class: :FatherOfSons
  SubClassOf: :hasChild some owl:Thing and :hasChild only :Man
 Individual: :John
  Types: :FatherOfSons
```

classes – represent sets of objects (e.g. `ex:Man`, `ex:Employee`, etc.)

individuals – represent particular objects (e.g. `ex:John`)

properties – represent binary relations between objects (e.g. `ex:hasChild`), or attributes (e.g. `ex:hasName`)

OWL namespace is `http://www.w3.org/2002/07/owl#`, prefixed as `owl:`.

OWL (2) Language Family

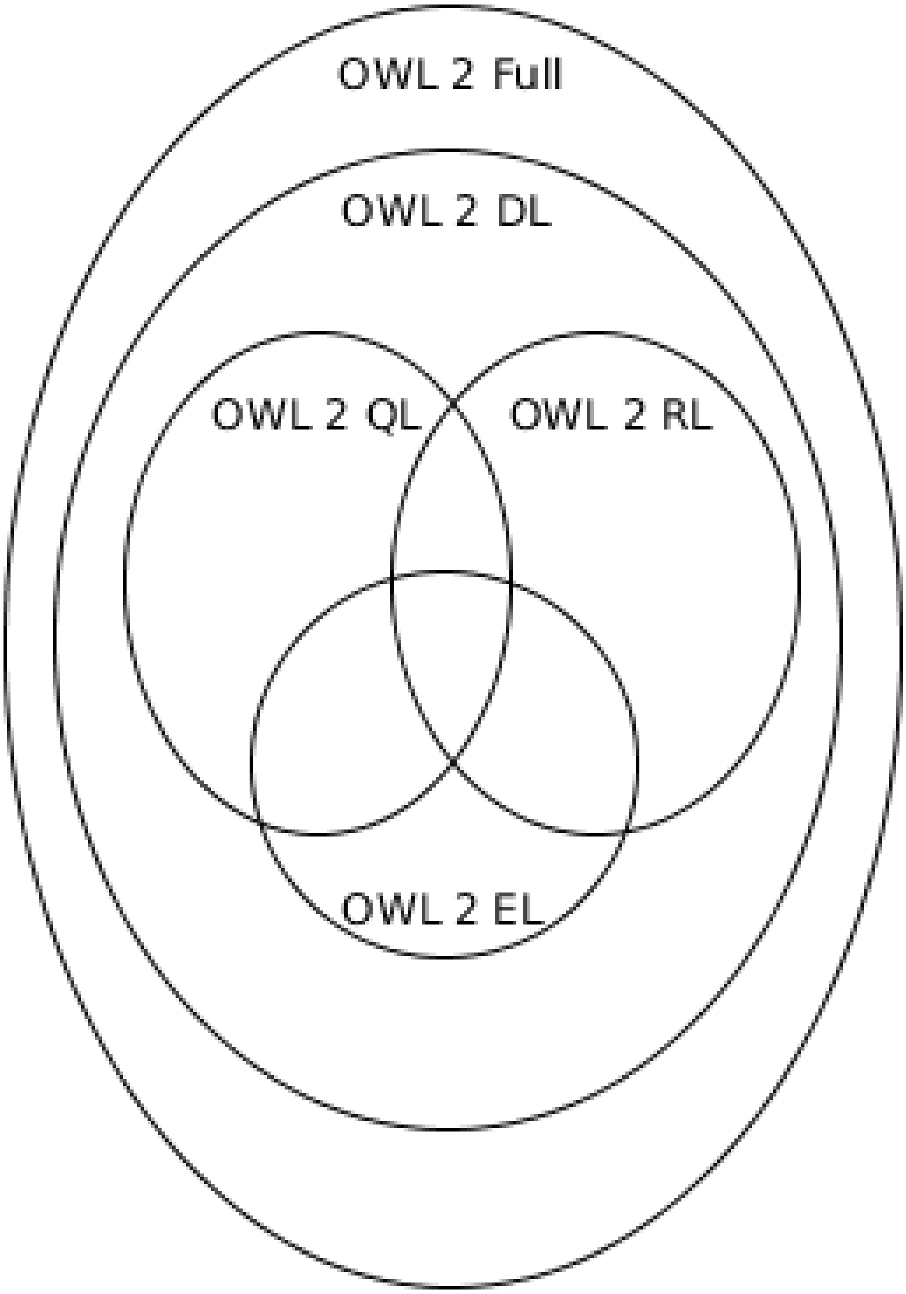
OWL 2 (Full) interprets any RDF graph under OWL-RDF entailment regime (undecidable).

OWL 2 DL interprets OWL 2 ontologies (parsed only from **compliant** RDF graphs) by means of **decidable** $\mathcal{SROIQ}$ description logic semantics,

OWL 2 EL is a subset of OWL 2 DL for rich class taxonomies,

OWL 2 QL is a subset of OWL 2 DL for large data,

OWL 2 RL is a subset of OWL 2 DL with weaker rule-based semantic.



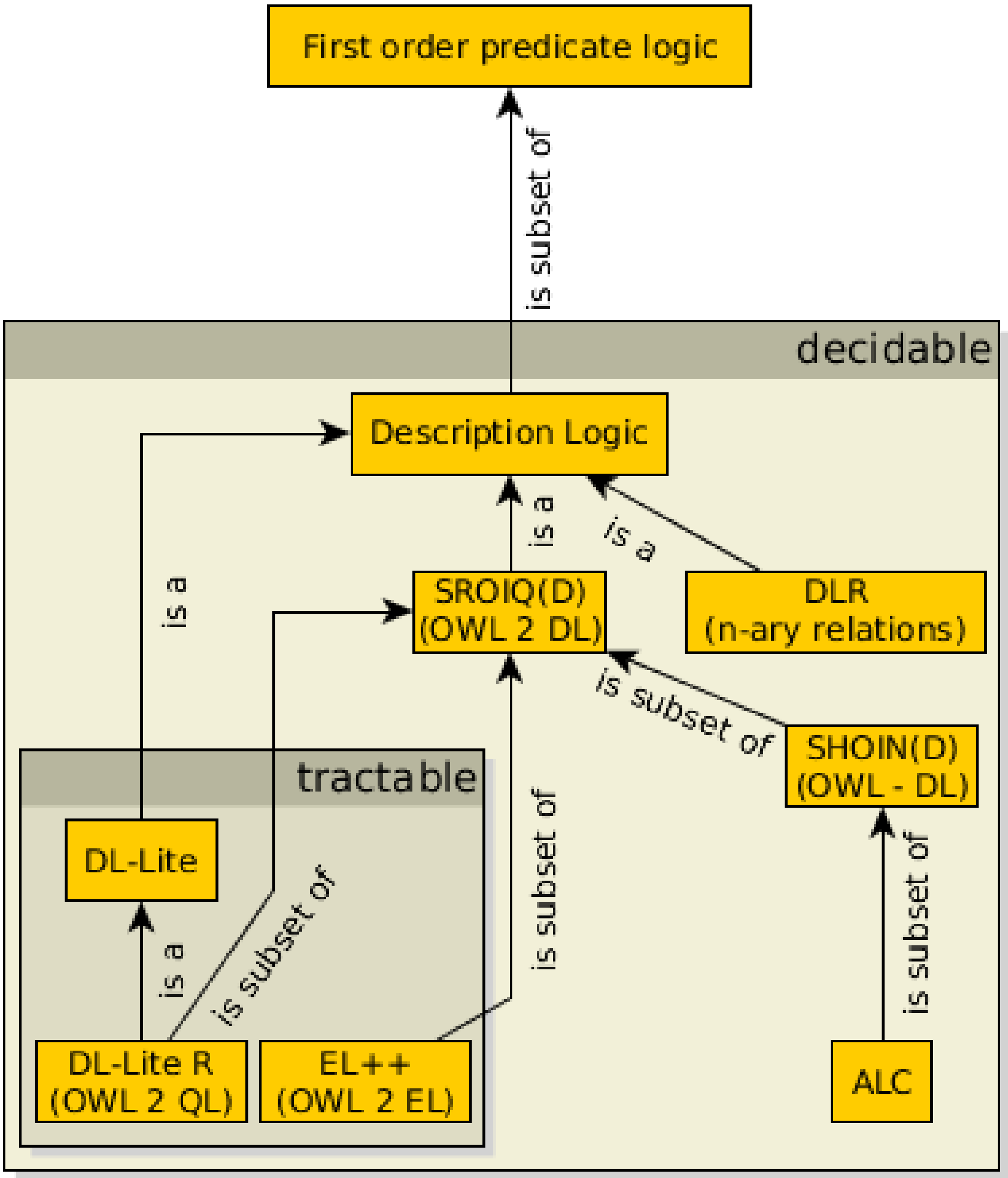
Description Logics

What are Description Logics (DLs) ?

- logics backing OWL languages

- *decidable subsets of first-order predicate logic (FOPL)*
- in 2004 description logic $\mathcal{SHOIN}(\mathcal{D})$ – OWL
- in 2009 description logic $\mathcal{SROIQ}(\mathcal{D})$ – OWL 2

$\mathcal{SROIQ}$
Currently, $\mathcal{SROIQ}(\mathcal{D})$ is one of the most expressive-yet-decidable description logic.



Characteristics of Description Logics

- suitable for descriptive tasks (conceptualization tasks),
- reasoning requires *satisfiability checking* (typically by means of a tableau algorithm),
- high complexity – even simple DLs are exponential, and thus poorly scalable on large datasets
- complexity of $\mathcal{SROIQ}(\mathcal{D})$ = N2ExpTime.
- complexity of $\mathcal{EL}++$, $DLLite_R$ = PTime.

Reasoning in Description Logics

Description logic reasoners take an ontology $\mathcal{O} = \{\alpha_i\}$, where α_i are axioms and are able to reason about

consistency , i.e. whether $\mathcal{O}$ is consistent (whether there exists at least one "realization" of the ontology that fullfills all α_i)

entailment , i.e. whether $\mathcal{O} \models \alpha$ for some axiom α

Basic Concepts

OWL Ontology Header

```
Prefix: : <http://ex.owl/>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Ontology: <http://ex.owl/o3> <http://ex.owl/o3-v1>
  Import: <http://ex.owl/o4>
  Import: <http://ex.owl/o5>
  Annotations: rdfs:comment "An example ontology"@en,
             :creator :John
  AnnotationProperty: :creator
  Individual: :John
```

- An ontology is identified by

  ontology IRI (`http://ex.owl/o3`) logically identifies an ontology (although it might be stored e.g. in a local file)

  version IRI (`http://ex.owl/o3-v1`) which is optional

- `Import:` allows importing other ontologies (for backward compatibility with OWL 1, the imported ontology is syntactically included in case it has no `Ontology:` header)
- `Annotations:` allows arbitrary ontology annotations (creators, comments, backward compatibility, etc.)

Ontology structure

Logically, an OWL 2 DL ontology is a set of axioms (see above). Each axiom represents a statement that must be valid in the domain, e.g

```
Class: :FatherOfSons
  SubClassOf: :hasChild some owl:Thing and :hasChild only :Man
```

says that a father of sons is someone who has at least one child and each his child is a man. These axioms are serialized as sets of RDF triples in RDF syntaxes, or as human-readable frames in Manchester syntax.

Listing 2 : Turtle version of the axiom above

```
:FatherOfSons rdf:type owl:Class ;
   rdfs:subClassOf [ rdf:type owl:Class ;
   owl:intersectionOf ( [ rdf:type owl:Restriction ;
                          owl:onProperty :hasChild ;
                          owl:someValuesFrom owl:Thing
                        ]
                        [ rdf:type owl:Restriction ;
                          owl:onProperty :hasChild ;
                          owl:allValuesFrom :Man
                        ]
                      )
```

Annotations

Each resource can be assigned a set of annotations (i.e. classes, properties, reified axioms, or even annotations themselves):

```
Class: :FatherOfSons
 Annotations:
      :creator :John,
      Annotations: :creator :Jack
            rdfs:label "Father of sons"@en
 SubClassOf:
      Annotations: :creator :Mary
            :hasChild some owl:Thing and :hasChild only :Man
```

Question

What do different creators refer to ?

Punning

Should `ex:Dog` be considered a class (representing a set of dogs), or an individual (representing a particular species) ?

Punning is the mechanism of reusing the same IRI for entities of different type for the sake of metamodeling but certain typing constraints must be fulfilled to stay in OWL 2 DL.

OWL 2 DL Typing constraints

- All IRIs have to be declared to be either *class*, *datatype*, *object property*, *data property*, *annotation property*, *individual* in the *axiom closure of an ontology*

- Each IRI can be (declared/used as) only one of (object property, data property, annotation property)
- Each IRI can be (declared/used as) only one of (class, datatype)

Punning example

Listing 3 : A valid punning example – `ex:Dog` is used as both individual and class

```
Individual: ex:Dog
  Types: ex:Dog
```

Listing 4 : An invalid punning example – `ex:hasName` is used as both object and data property

```
Individual: ex:John
  Facts: ex:hasName ex:firstName
  Facts: ex:hasName "John"@en
```

Properties

Property Expressions

OWL 2 supports no data property expressions and the only object property expresion:

`inverse` means an inverse property (i.e. property going in the opposite direction),

```
inverse :hasChild
```

Inverse properties can be used in class frames, property frames as well as individuals frames.

Object Property Frames

```
ObjectProperty: :hasMother
    Characteristics: Functional, Irreflexive, Asymmetric
    Domain: :Person
    Range: :Woman
    SubPropertyOf: :hasParent
    EquivalentTo: inverse :isMotherOf
    DisjointWith: :hasFather
    InverseOf: :isMotherOf
    SubPropertyChain: :hasFather o :isWifeOf
```

`Characteristics` – selection of `Functional`,`InverseFunctional`, `Transitive`, `Reflexive`, `Irreflexive`, `Symmetric`, `Asymmetric` – interpreted in their mathematical sense

`Domain,Range` have the same meaning as in RDFS

`SubPropertyOf` specifies props representing supersets of the frame property

`EquivalentTo` specifies props semantically equivalent to the frame class

DisjointWith specifies props disjoint with the frame property

InverseOf specifies inverse props (like `inverse` property expression)

SubPropertyChain specifies a property composition

Data Property Frames

```
DataProperty: :hasBirthNumber
   Characteristics: Functional
   Domain: :Person
   Range: xsd:string
   SubPropertyOf: :hasIdentifyingNumber
```

The only **Characteristics** available is `Functional`. Other sections have the same meaning

Data Ranges (Datatype expressions)

Basic Data Ranges

OWL 2 supports basic modeling constructs for custom data ranges:

and,or,not have the meaning of standard set intersection, union and complement,

```
(xsd:nonNegativeInteger and xsd:nonPositiveInteger) or xsd:string
```

individual enumeration lists individuals belonging to a class expression.

```
{"true"^^xsd:boolean 1}
```

Facets

Facets restrict a particular datatype to a subset of its values.

```
xsd:integer[ >= 5, < 10 ]
```

Available facets

length, minLength, maxLength – string lengths

pattern – string regular expression

langRange – range of language tags

<=,<,>=,> – number comparison

New datatypes can be used by means of datatype frame axioms:

```
Datatype: :MyNumber
   EquivalentTo: xsd:integer[ >= 5, < 10 ]
```

Class Expressions

Boolean operators

OWL 2 supports many class modeling constructs including boolean connectives, individual enumeration, and object/data value restrictions.

owl:Thing, owl:Nothing are two predefined OWL classes containing all (resp. no) individuals,

and,or,not have the meaning of standard set intersection, union and complement,

```
(:FlyingObject and not :Bat) or :Pinguin
```

individual enumeration lists individuals belonging to a class expression.

```
{:John :Mary}
```

Object value Restrictions (1)

existential quantification says that a property filler exists (not necessarily in data !)

Listing 5 : A set of objects having at least one son

```
:hasChild some :Man
```

universal quantification says that each property filler belongs to a class

Listing 6 : A set of objects having no child or only sons

```
:hasChild only :Man
```

cardinality restriction restricts the number of property fillers

Listing 7 : Sets of objects exactly two (min four/max one) wheels

```
:hasPart exactly 2 :Wheel
:hasPart min 4 :Wheel
:hasPart max 1 :Wheel
```

Object Value Restrictions (2)

individual value restriction restricts a property filler to a specified individual

Listing 8 : A set of objects having John as their child

```
:hasChild value :John
```

self restriction restricts a property filler to the same individual

Listing 9 : A set of objects trusting themselves

```
:trusts Self
```

Complex Value Restrictions

- analogous counterparts to the object value restrictions are available (except the Self restriction) as *data value restrictions*:

  ```
  :hasName some xsd:string[length 2]
  ```

What does this class expression describe ?

```
(:hasPart only (not :Tail))
and (:hasPart max 2 (:hasPart some :Knee))
and (:doesAssignmentWith Self)
and (:hasGrade only xsd:string[pattern "[AB]"])
```

Class frames

```
Class: :Father
  SubClassOf: :Parent
  EquivalentTo: :Man and :hasChild some :Person
  DisjointWith: :Mother
  DisjointUnionOf: :HappyFather :SadFather
  HasKey: :hasBirthNumber
```

SubClassOf section defines axioms specifying supersets of the frame class

EquivalentTo section defines axioms specifying classes semantically equivalent to the frame class

DisjointWith section defines classes sharing no individuals with the frame class

DisjointUnionOf section defines classes that are mutually disjoint and union of which is semantically equivalent to the frame class

HasKey section defines a set of properties that build up a *key* for the class – all instances of `Father` sharing the same value for the key (`:hasBirthNumber`) are semantically identical (`owl:sameAs`)

Individuals

Individual Frames

```
Individual: :John
  Types: :Person , :hasName value "Johnny"
  Facts: :hasChild :Jack, not :hasName "Bob"
  SameAs: :Johannes
  DifferentFrom: :Jack
```

Individual frames contain assertions, subject of which is the individual.

`Types` specifies class descriptions that are types (`rdf:type`) for the frame individual,

Facts specifies the object and data property assertions,

SameAs specifies individuals being semantically identical to the frame individual,

DifferentFrom specifies individuals being semantically different to the frame individual

Unique Name Assumption

OWL **does not accept** unique name assumption, i.e. it is not known whether two individuals `:John` and `:Jack` represent the same object, or not. By `SameAs` and `DifferentFrom`, either possibility can be enforced.

Listing 10 : This fragment does not cause ontology inconsistency as `:Jack` and `:Jim` might be interpreted as the same individual

```
Individual: :John
 Types: :hasChild exactly 1 owl:Thing
 Facts: :hasChild :Jack, :hasChild :Jim
```

OWL 2 DL Constraints

Global Constraints

We have discussed the typing constraints. Additionally, there are syntactic constraints that ensure decidability of reasoning. These constraints must be fulfilled for each OWL 2 DL ontology:

simple object property are properties that have no direct or indirect (through property hierarchy) subproperties that are transitive or defined by means of a property chain.

```
ObjectProperty: :hasChild
 SubPropertyOf: :hasDescendant
ObjectProperty: :hasDescendant
 Characteristics: Transitive
 SubPropertyOf: :hasRelative
ObjectProperty: :hasSon
 SubPropertyOf: :hasChild
ObjectProperty: :hasDaughter
 SubPropertyOf: :hasChild
ObjectProperty: :hasUncle
 SubPropertyOf: :hasRelative
 SubPropertyChain: :hasParent o :hasSibling
```

Global Constraints (2)

Formal specification is in [3], informally:

- `owl:topDataProperty` cannot be stated equal to any other data property (e.g. through `EquivalentTo` or `SubPropertyOf`).

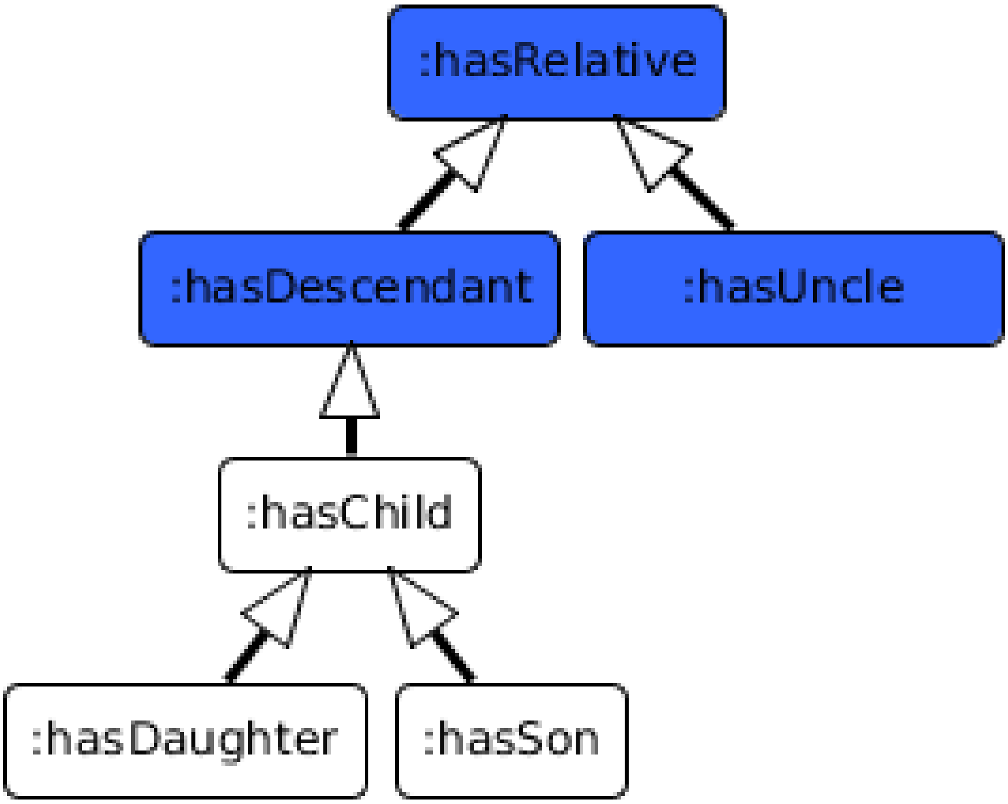


Figure 0.1: White properties are simple, blue ones are not.

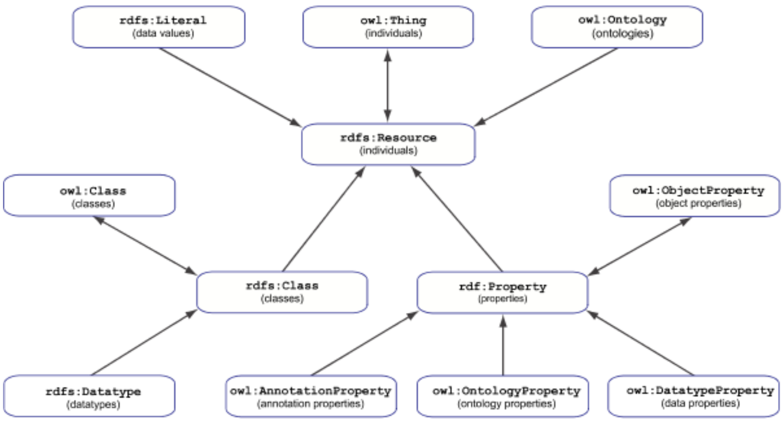


Figure 0.2: Informal relationships between OWL parts (taken from [3]). Arrows represent subparts (either `rdf:type` or `rdfs:subClassOf`)

- datatype definitions must be acyclic
- the following constructs are only allowed with *simple properties*:
  - cardinality restrictions (`min`, `max`, `exactly`),
  - self restriction ((Self)),
  - property characteristics `Functional`, `InverseFunctional`, `Irreflexive`,`Asymmetric`,
  - property axiom `DisjointWith`
- property chains must not be cyclic
- (restriction on anonymous individuals (that we haven't discussed))

OWL 2 – RDFS mapping

OWL 2 – RDFS mapping

The vocabulary for serializing OWL ontologies into RDF contains plenty of resource (e.g. `owl:Class`,`owl:someValuesFrom`,`owl:Axiom`), see [3] for details.

OWL 2 Semantics

OWL 2 RDF-Based Semantics

defines an entailment $\models_{OWL2-RDF}$) allowing to **interpret all RDF graphs** (called *OWL 2 Full*)

- is an extension of D-entailment (inteprets the whole RDF graph)
- undecidable, but *incomplete* entailment rules are provided [4]

Listing 11 : RDF graph G_1

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <http://example.org/2014-osw-14/>.
_:y a owl:Ontology .
_:x rdfs:subClassOf :Parent ;
   a owl:Restriction ;
:hasChild a owl:ObjectProperty .
:John :hasChild :Mary .
```

Listing 12 : RDF graph G_2

```
@prefix : <http://www.example.org/2014-osw-14/> .
:hasChild a rdf:Property .
:Mary a owl:NamedIndividual .
```

The following entailment holds:

$$G_1 \models_{OWL2-RDF} G_2$$

OWL 2 Direct Semantics

defines an entailment $\models_{OWL2-DL}$ in terms of the $\mathcal{SROIQ}(\mathcal{D})$ DL.

- interprets only "logically-backed" knowledge, while **ignoring the rest** (e.g. annotations, declarations, etc.)
- $F(G)$ is an OWL 2 DL ontology, for G sat. OWL 2 DL restrictions.

Listing 13 : RDF graph G_3

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <http://example.org/2014-osw-14/>.
_:y a owl:Ontology .
_:x rdfs:subClassOf :Parent ;
   a owl:Restriction ;
   owl:onProperty :hasChild ;
   owl:someValuesFrom owl:Thing .
:John :hasChild :Mary .
:John a owl:NamedIndividual .
:Mary a owl:NamedIndividual .
:hasChild a owl:ObjectProperty .
```

Listing 14 : RDF graph G_4

```
@prefix : <http://www.example.org/2014-osw-14/> .
:John a :Parent .
:John rdfs:label "john"@en .
```

The following entailment holds:

$$F(G_3) \models_{OWL2-DL} F(G_4)$$

(For the sake of brevity, $F(\bullet)$ is often omitted whenever G is a serialization of an OWL-DL ontology $F(G)$)

OWL 2 Correspondence Theorem (CT)

- direct and RDF-based semantics for OWL are different (i.e. there exist entailments valid for one semantic and not for the other one)
- CT says that **OWL RDF semantic can express anything that OWL DL semantics can**

  OWL 2 Correspondence Theorem – simplified version

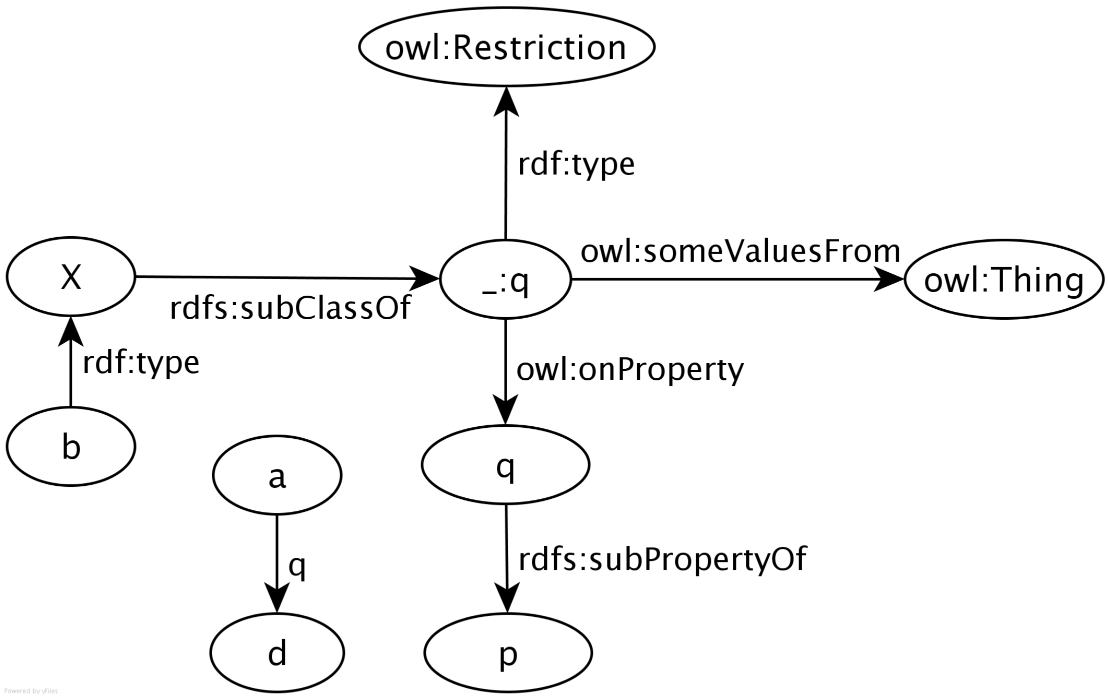
  For any two RDF graphs G_1 and G_2, there exist two RDF graphs G_1' and G_2', s.t. $F(G_1) \dashv\models_{OWL-DL} F(G_1')$ and $F(G_2) \dashv\models_{OWL2-DL} F(G_2')$, and

  $$F(G_1') \models_{OWL2-DL} F(G_2') \quad \text{implies} \quad G_1' \models_{OWL2-RDF} G_2',$$

  where $F(G)$ is an OWL-DL ontology corresponding to the RDF graph G.

- For example $G_1 \not\models_{OWL2-DL} G_2$, while $G_3 \not\models_{OWL2-RDF} G_4$
- Removing last triple (label) from G_4, we get G_4', s.t. $F(G_4) \dashv\models_{OWL-DL} F(G_4')$ and $G_4 \models_{OWL-RDF} G_4'$

OWL 2 Entailment in SPARQL




```
PREFIX : <http://ex.org/e1>
SELECT ?x
WHERE { ?x :p _:y }
```

Simple-entailment No result.

RDF-entailment No result.

RDFS-entailment One result: ?x=a.

OWL 2 DL entailment Two results: ?x=a and ?x=b.

```
Individual: :b
  Types: :X
Class: :X
  SubClassOf: :q some owl:Thing
```

0.1.2 SWRL

SWRL Basics

- SWRL [1] is a rule language for OWL
- allows rules of the form (Protégé syntax)

$$a_1(?x^1_1, \ldots, ?x^1_{n_1}), \ldots, a_m(?x^m_1, \ldots, ?x^m_{n_m}) \rightarrow \\ c_1(?x^{m+1}_1, \ldots, ?x^{m+1}_{l_1}), \ldots, c_k(?x^{m+k}_1, \ldots, ?x^{m+k}_{l_k}) \quad (0.1)$$

where all $a^{(s)}_t$ are antecedents and $c^{(s)}_t$ consequents of the rule. Each $a^{(s)}_t$, $c^{(s)}_t$ is of the following form:

C(z) , where C is a class expression, or

P(z_1, z_2) , where P is a property expression,

SameAs(z_1, z_2) , or

DifferentFrom(z_1, z_2) , or

$built-in(z_1, \ldots, z_p)$, where *built−in* is one of the built-in atoms, e.g. add$(?x, 3, 4)$

where each $z_{(i)}$ is either a variable $?x$ or an individual, or a literal.

Example Rules

Listing 15 : Transitivity of hasAncestor

```
hasAncestor(?x,?y), hasAncestor(?y,?z) -> hasAncestor(?x,?z)
```

Listing 16 : One parent is a man

```
Man(?y), hasParent(?x,?y), hasParent(?x,?z), DifferentFrom(?y,?z) -> Woman(?z)
```

Listing 17 : Women share surnames of their husbands.

```
hasHusband(?x,?y), hasSurname(?y,?z) -> hasSurname(?x,?z), Man(?x)
```

SWRL Reasoning

- SWRL rules can be embedded into OWL ontologies
- unrestricted, they are *undecidable*
- their decidable subset is called **DL-safe rules** which shares their syntax but variables match only **known** data (not entailed)
- SWRL support (DL-safe rules) is implemented in Protégé, as well as major reasoners, like Pellet, HermiT, etc.

References

[1] Ian Horrocks et al. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. W3C Member Submission. World Wide Web Consortium, 2004. URL: `http://www.w3.org/Submission/SWRL`.

[2] Peter Patel-Schneider, Bijan Parsia, and Boris Motik. *OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition)*. W3C Recommendation. http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/. W3C, Dec. 2012.

[3] Peter Patel-Schneider, Bijan Parsia, and Boris Motik. *OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition)*. W3C Recommendation. http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/. W3C, Dec. 2012.

[4] Michael Schneider. *OWL 2 Web Ontology Language RDF-Based Semantics (Second Edition)*. W3C Recommendation. http://www.w3.org/TR/2012/REC-owl2-rdf-based-semantics-20121211/. W3C, Dec. 2012.