OWL 2

Web Ontology Language

Some material adapted from presentations by Ian Horrocks and by Feroz Farazi

Introduction

- OWL 2 extends OWL 1.1 and is backward compatible with it
- The new features of OWL 2 based on real applications, use cases and user experience
- Adopted as a W3C recommendation in December 2012
- All new features were justified by use cases and examples
- Most OWL software supports OWL 2

Features and Rationale

- Syntactic sugar
- New constructs for properties
- Extended datatypes
- Punning
- Extended annotations
- Some innovations
- Minor features

Syntactic Sugar

- OWL 2 adds features that
  - Don’t change expressiveness, semantics, complexity
  - Makes some patterns easier to write
  - Allowing more efficient processing in reasoners
- New features include:
  - DisjointClasses
  - DisjointUnion
  - NegativeObjectPropertyAssertion
  - NegativeDataPropertyAssertion

Syntactic sugar: disJointClasses

- It's common to want to assert that a set of classes are pairwise disjoint
  - No individual can be an instance of two of the classes in set
- Faculty, staff and students are all disjoint

  [a owl:allDisjointClasses;

  owlmembers (:faculty :staff :students)]
- In OWL 1.1 we'd have to make three assertions
  - :faculty owl:disjointWith :staff
  - :faculty owl:disjointWith :student
  - :staff owl:disjointWith :staff
- Which gets cumbersome for large sets

Syntactic sugar: disJointUnion

- Need for disjointUnion construct
  - A *:CarDoor* is exclusively either
    - a *:FrontDoor*, a *:RearDoor* or a *:TrunkDoor*
    - and not more than one of them
- In OWL 2

  :CarDoor a owl:disjointUnionOf (:FrontDoor :RearDoor :TrunkDoor).
- In OWL 1.1

  :CarDoor owl:unionOf (:FrontDoor :RearDoor :TrunkDoor).

  :FrontDoor owl:disjointWith :ReadDoor .

  :FrontDoor owl:disjointWith :TrunkDoor .

  :RearDoor owl:disjointWith :TrunkDoor .

Syntactic sugar: disJointUnion

- It’s common for a concept to have more than one decomposition into disjoint union sets
- E.g.: every person is either male or female (but not both), either a minor or adult (but not both) and either living or dead (but not both)

```
foaf:Person
  owl:disjointUnionOf (:Male :Female);
  owl:disjointUnionOf (:Minor :Adult);
  owl:disjointUnionOf (:Living :Dead);
```

Syntactic sugar: negative assertions

- Asserts that a property doesn't hold between two instances or between an instance and a literal
- NegativeObjectPropertyAssertion
  - Barack Obama was not born in Kenya
- NegativeDataPropertyAssertion
  - Barack Obama is not 60 years old
- Encoded using a "reification style"

Syntactic sugar: negative assertions

```
@prefix dbr: <http://dbpedia.org/resource/> .
@prefix dbo: <http://dbpedia.org/ontology/> .

[a owl:NegativeObjectPropertyAssertion;
    owl:sourceIndividual  dbr:Barack_Obama ;
    owl:assertionProperty dbo:bithPlace ;
    owl:targetIndividual  dbr:Kenya] .

[a owl:NegativeDataPropertyAssertion;
    owl:sourceIndividual  dbo:Barack_Obama ;
    owl:assertionProperty dbo:age ;
    owl:targetIndividual  "60" ] .
```

Syntactic sugar: negative assertions

- Note that the negative assertions are about two **individuals**
- Suppose we want to say that :john has no spouse?
- Or to define the concept of an unmarried person?
- Can we use a negative assertion to do it?

Syntactic sugar: negative assertions

- Suppose we want to say that :john has no spouse?

  [a owl:NegativeObjectPropertyAssertion;

      owl:sourceIndividual :john ;

      owl:assertionProperty dbpo:spouse ;

      owl:targetIndividual ????????] .

- We can’t do this with a negative assertion ☹
- It requires a variable, e.g., there is no ?X such that (:john, dbpo:spouse, ?X) is true

Syntactic sugar: negative assertions

- The negative assertion feature is limited
- Can we define a concept :unmarriedPerson and assert that :john is an instance of this?
- We can do it this way in OWL:
  - An unmarried person is a kind of person
  - and a kind of thing with exactly 0 spouses

John is not married

```
:john a :unmarriedPerson .
:unmarriedPerson
   a Person;
   a [a owl:Restriction;
      onProperty dbpo:spouse;
      owl:cardinality "0"] .
```

New property Features

- Self restriction
- Qualified cardinality restriction
- Object properties
- Disjoint properties
- Property chain
- Keys

Self restriction

- Classes of objects that are related to themselves by a given property
  - E.g., the class of processes that regulate themselves
- It is also called *local reflexivity*
  - E.g., Auto-regulating processes regulate themselves
- Narcissists are things who love themselves

```
:Narcissist owl:equivalentClass
   [a owl:Restriction;
    owl:onProperty  :loves;
    owl:hasSelf "true"^^xsd:boolean] .
```

Qualified cardinality restrictions

- Qualifies the instances to be counted
- Six varieties: {Data|Object}{Min|Exact|Max} Type
- Examples
  - People with **exactly** 3 children who are girls
  - People with **at least** 3 names
  - Each individual has **at most** 1 SSN
  - E.g., pizzas with exactly four toppings all of which are cheeses

Qualified cardinality restrictions

- Done via new properties with domain owl:Restriction, namely *{min|max|} QualifiedCardinality* and *onClass*
- E.g.: people with exactly 3 children who are girls

  [a owl:restriction;

    owl:onProperty :hasChild;

    owl:onClass [owl:subClassOf :Female;
                            owl:subClassOf :Minor].

    QualifiedCardinality “3” .
- Or: hasChild exactly 3 Female and Minor

Object properties

- ReflexiveObjectProperty
  - Globally reflexive
  - Everything is part of itself
- IrreflexiveObjectProperty
  - Nothing can be a proper part of itself
- AsymmetricObjectProperty
  - If x is proper part of y, then the opposite does not hold

Disjoint properties

- E.g., you can't be both the *parent of* and *child of* the same person
- DisjointObjectProperties (for object properties)

  E.g., :hasParent owl:propertyDisjointWith :hasChild
- DisjointDataProperties (for data properties)

  E.g., :startTime owl:disjointWith :endTime
- AllDisjointProperties for pairwise disjointness

  [a owl:AlldisjointProperties ;

    owl:members (:hasSon :hasDaughter :hasParent)] .

A Dissertation Committee

Here is a relevant real-world example.

A dissertation committee has a candidate who must be a student and five members all of whom must be faculty. One member must be the advisor, another can be a co-advisor and two must be readers. The readers can not serve as advisor or co-advisor.

How can we model it in OWL?

A Dissertation Committee

A **dissertation committee** has a candidate who must be a student and five members all of whom must be faculty. One member must be the advisor, another can be a co-advisor and two must be readers. The readers can not serve as advisor or co-advisor.

- Define a DissertationCommittee class
- Define properties it can have along with appropriate constraints

A Dissertation Committee

:DC a owl:class; [a owl:Restriction;
owl:onProperty :co-advisor; owl:maxCardinality “1”] .

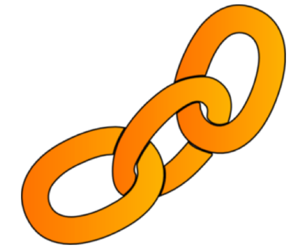
:candidate a owl:FunctionalProperty;
rdfs:domain :DC; rdfs:range :Student.

:advisor a owl:FunctionalProperty;
rdfs:domain :DC; rdfs:range :Faculty.

:co-advisor owl:ObjectProperty;
rdfs:domain :DC; rdfs:range :Faculty,
owl:propertyDisjointWith :advisor .

…

Property Chains



- A common pattern in a graph representation is a chain of properties, e.g. parent·parent
- Properties can be defined as a composition of other properties
- The brother of your parent is your uncle

  :uncle owl:propertyChainAxiom (:parent :brother).
- Your parent’s sister’s spouse is your uncle

  :uncle owl:propertyChainAxiom (:parent :sister :spouse).

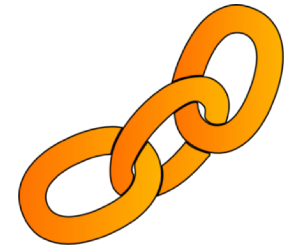
  .

Property chains: OWL vs. SPARQL

- SPARQL also supports property chains (aka paths) and adds expressivity with a regex-like grammar
- Operators include ? (0 or 1), + (one or more), * (any number), ^ (inverse), # constraints, …

Syntax Form	Matches
uri	A URI or a prefixed name. A path of length one.
^*elt*	Inverse path (object to subject).
(*elt*)	A group path *elt*, brackets control precedence.
elt1 / *elt2*	A sequence path of *elt1*, followed by *elt2*
elt1 ^ *elt2*	Shorthand for *elt1* / ^*elt2*, that is *elt1* followed by the inverse of *elt2*.
elt1 | *elt2*	A alternative path of *elt1*, or *elt2* (all possibilities are tried).
*elt**	A path of zero or more occurrences of *elt*.
elt+	A path of one or more occurrences of *elt*.
elt?	A path of zero or one *elt*.
elt{n,m}	A path between n and m occurrences of *elt*.
elt{n}	Exactly *n* occurrences of *elt*. A fixed length path.
elt{n,}	*n* or more occurrences of *elt*.
elt{,n}	Between 0 and *n* occurrences of *elt*.

Property chains: OWL vs. SPARQL



- Common usecase: find all of an entities types

  SELECT DISTINCT ?class WHERE {

  dbr:Barack_Obama rdf:type/owl:subclassOf* ?class }

- Another: find all birth places using isPartOf

  SELECT DISTINCT ?place WHERE {

  dbr:Barack_Obama dbo:birthplace/dbo:isPartOf* ?place}

- Another: find all ancestors

  SELECT DISTINCT ?person WHERE {

  dbr:Barack_Obama ^dbo:child+ ?person}

Keys

- Individuals can be identified uniquely
- Identification can be done using
  - A data or object property (equivalent to inverse functional)
  - A set of properties
- Examples

```
foaf:Person
    owl:hasKey (foaf:mbox),
                   (:homePhone :foaf:name).
```

Extended datatypes

- Extra datatypes
  - Examples: owl:real, owl:rational, xsd:pattern
- Datatype restrictions
  - Range of datatypes
  - For example, a teenager has age between 13 and 18

Extended datatypes

- Data range combinations
  - Intersection of
    - DataIntersectionOf(*xsd:nonNegativeInteger xsd:nonPositiveInteger*)
  - Union of
    - DataUnionOf(*xsd:string xsd:integer*)
  - Complement of data range
    - DataComplementOf(*xsd:positiveInteger*)

An Example: Teenager

```
:Teenager a
   [owl:Restriction ;
      owl:onProperty :hasAge ;
      owl:someValuesFrom _:y .]
_:y a rdfs:Datatype ;
      owl:onDatatype  xsd:integer ;
      owl:withRestrictions ( _:z1 _:z2 ) .
_:z1 xsd:minInclusive     "13"^^xsd:integer .
_:z2 xsd:maxInclusive     "19"^^xsd:integer .
```

An Example: Teenager (2)

```
:Teenager a
  [owl:Restriction ;
     owl:onProperty :hasAge ;
     owl:someValuesFrom
       [a rdfs:Datatype ;
         owl:onDatatype  xsd:integer ;
         owl:withRestrictions
           ( [xsd:minInclusive    "13"^^xsd:integer]
             [xsd:maxInclusive    "19"^^xsd:integer ])]] .
```

Punning

- *OWL 1 DL* things can't be both a class and instance
  - E.g., :SnowLeopard can't be both a subclass of :Feline and an instance of :EndangeredSpecies
- OWL 2 DL offers better support for meta-modeling via *punning*
  - A URI denoting an owl thing can have two distinct views, e.g., as a **class** and as an **instance**
  - The one intended is determined by its **use**
  - A *pun* is often defined as a joke that exploits the fact that a word has two different senses or meanings

Punning Restrictions

- Some puns are not allowed ☹
- Classes and object properties also can have the same name
  - For example, :mother can be both a property and a class of people
- But classes and datatype properties can not have the same name
- Also datatype properties and object properties can not have the same name

Punning Example

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

foaf:Person a owl:Class.
:Woman a owl:Class.
:Parent a owl:Class.

:mother a owl:ObjectProperty;
  rdfs:domain foaf:Person;
  rdfs:range foaf:Person .

:mother a owl:Class;
  owl:intersectionOf (:Woman :Parent).
```

validate via http://owl.cs.manchester.ac.uk/validator/

Annotations

- In OWL *annotations* comprise information that carries no official meaning
- Some properties in OWL 1 are annotation properties, e.g., owl:comment, rdf:label and rdf:seeAlso
- OWL 1 allowed RDF reification as a way to say things about triples, again w/o official meaning

```
[a rdf:Statement;
  rdf:subject :Barack_Obama;
  rdf:predicate dbpo:born_in;
  rdf:object :Kenya;
  :certainty “0.01” ].
```

Annotations

- OWL 2 has native support for annotations, including
  - Annotations on owl axioms (i.e., triples)
  - Annotations on entities (e.g., a Class)
  - Annotations on annotations
- The mechanism is again reification

Annotations

```
:Man rdfs:subClassOf :Person .
_:x  rdf:type       owl:Axiom ;
    owl:subject    :Man ;
    owl:predicate  rdfs:subClassOf ;
    owl:object     :Person ;
    :probability “0.99"^^xsd:integer;
    rdfs:label     ”Every man is a person.” .
```

Inverse object properties

- Some object property can be the inverse of another property
- For example, partOf and hasPart
- ObjectInverseOf(*:partOf*) expression represents the inverse property of *:partOf*
- Makes writing ontologies easier by avoiding the need to explicitly name an inverse

OWL Sub-languages

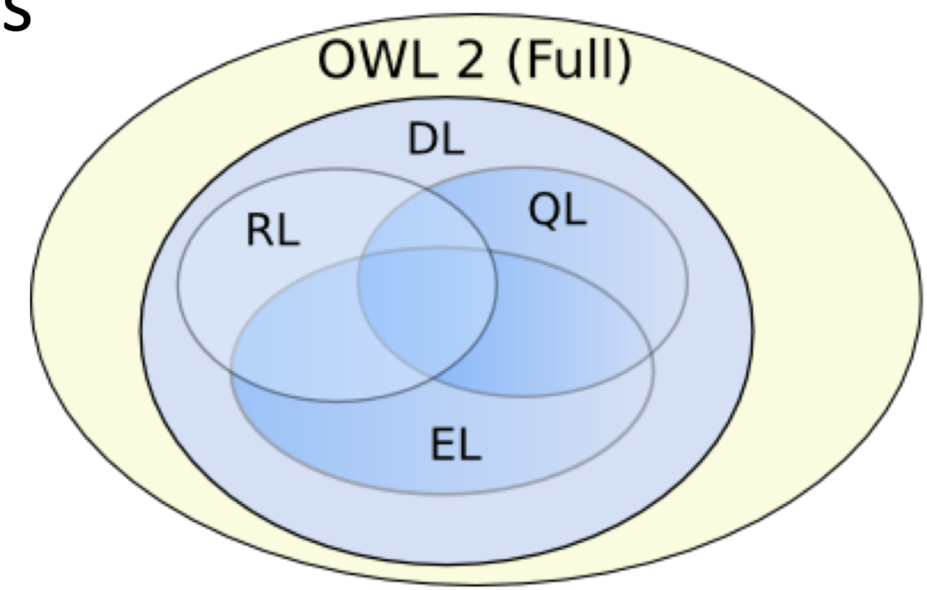
- OWL 1 had sub-languages: OWL FULL, OWL DL and OWL Lite
  - OWL FULL is undecidable
  - OWL DL is worst case highly intractable
  - Even OWL Lite turned out to be not very tractable (EXPTIME-complete)
- OWL 2 introduced three sub-languages (*profiles)* designed for different use cases

OWL 2 Profiles

- **EL**: polynomial time reasoning for schema & data
  - Useful for ontologies with large conceptual part
- **QL**: fast (logspace) query answering using RDBMs via SQL
  - Useful for large datasets already stored in RDBs
- **RL**: fast (polynomial) query answering using rule-extended DBs
  - Useful for large datasets stored as RDF triples

OWL Profiles

- Profiles considered
  - Useful computational properties, e.g., reasoning complexity
  - Implementation possibilities, e.g., using RDBs
- There are three profiles
  - OWL 2 EL
  - OWL 2 QL
  - OWL 2 RL



OWL 2 EL

- A (near maximal) fragment of OWL 2 such that
  - Satisfiability checking is in PTime (**PTime-Complete**)
  - Data complexity of query answering is PTime-Complete
- Based on **EL** family of description logics
  - Existential (someValuesFrom) + conjunction
- Does not allow disjunction or *universal restrictions*
- *Saturation* is an efficient reasoning technique
- It can capture the expressive power used by many large-scale ontologies, e.g., SNOMED CT

Basic Saturation-based Technique

Normalise ontology axioms to standard form:

$$A \sqsubseteq B \quad A \sqcap B \sqsubseteq C \quad A \sqsubseteq \exists R.B \quad \exists R.B \sqsubseteq C$$

- Saturate using inference rules:

$$\frac{A \sqsubseteq B \quad B \sqsubseteq C}{A \sqsubseteq C} \qquad \frac{A \sqsubseteq B \quad A \sqsubseteq C \quad B \sqcap C \sqsubseteq D}{A \sqsubseteq D}$$

$$\frac{A \sqsubseteq \exists R.B \quad B \sqsubseteq C \quad \exists R.C \sqsubseteq D}{A \sqsubseteq D}$$

- Extension to Horn fragment requires (many) more rules

Saturation is a general reasoning technique in which you first compute the deductive closure of a given set of rules and add the results to the KB. Then run your prover.

Saturation-based Technique

Performance with large bio-medical ontologies

	GO	NCI	Galen v.0	Galen v.7	SNOMED
Concepts:	20465	27652	2748	23136	389472
FaCT++	15.24	6.05	465.35	—	650.37
HermiT	199.52	169.47	45.72	—	—
Pellet	72.02	26.47	—	—	—
CEL	1.84	5.76	—	—	1185.70
CB	1.17	3.57	0.32	9.58	49.44
Speed-Up:	1.57X	1.61X	143X	∞	13.15X

Galen and Snomed are large ontologies of medical terms; both have OWL versions. NCI is a vocabulary of cancer-related terms. GO is the gene ontology.

OWL 2 QL

- The QL acronym reflects its relation to the standard relational **Query Language**
- It does not allow *existential* and *universal restrictions* to a class expression or a data range
  - enable a tight integration with RDBMSs
  - reasoners can be implemented on top of standard relational databases
- Can answer complex queries (in particular, unions of conjunctive queries) over the instance level (ABox) of a DL knowledge base

OWL 2 QL

We can exploit **query rewriting** based reasoning technique

- Computationally optimal
- Data storage and query evaluation can be delegated to standard RDBMS
- Can be extended to more expressive languages (beyond AC^0) by delegating query answering to a Datalog engine

What is Datalog?

- Truly declarative logic programming language that's a subset of Prolog
  - Just rules and facts
  - No data structures, cut
  - Rule ordering unimportant
- Used as a query language for deductive databases
- Queries on finite sets sets guaranteed to terminate

```
parent(bill,mary).
parent(mary,john).

ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(X,Z),ancestor(Z,Y).
```

Query Rewriting Technique (basics)

- Given ontology O and query Q, use O to rewrite Q as Q^{0} such that, for any set of ground facts A:

  $$\mathrm{ans}(Q, O, A) \;=\; \mathrm{ans}(Q^{0}, ;, A)$$

- Resolution based query rewriting
  - **Clausify** ontology axioms
  - **Saturate** (clausified) ontology and query using resolution
  - **Prune** redundant query clauses

OWL 2 RL

- RL acronym reflects relation to ***Rule Languages***
- OWL 2 RL designed to accommodate
  - OWL 2 applications that trade full expressivity for efficiency
  - RDF(S) applications needing added expressivity from OWL 2
- Not allowed: *existential quantification* to a class, *union* and *disjoint union* to class expressions
- It can be implemented using rule-based technologies such Datalog, Jess, Prolog, etc.

Profile Selection…

Depends on

- Expressiveness required by the application
- Priority given to reasoning on classes or data
- Size of the datasets

Conclusion

- Most of the new features of OWL 2 in comparing with the initial version of OWL have been discussed
- Rationale behind the inclusion of the new features have also been discussed
- Three profiles – EL, QL and RL – are provided that fit different use cases and implementation strategies