OWL 2  
Web Ontology Language

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

Introduction

• OWL 2 extends OWL 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 2009
• All new features were justified by use cases and examples

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
  – Allows some patterns easier to write
  – Allows more efficient processing in reasoners
• New features include:
  – disJointUnion
  – DisjointClasses
  – NegativeObjectPropertyAssertion
  – NegativeDataPropertyAssertion

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 turtle:
  :CarDoor a owl:Class;
  owl:disjointUnionOf (:FrontDoor, :RearDoor, :TrunkDoor).

Syntactic sugar: disJointClasses

• It’s common to want to assert that a set of classes are pairwise disjoint
• i.e., that no individual can be an instance of two of the classes in the set

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 dbp: <http://dbpedia.org/resource/>
@prefix dbpo: <http://dbpedia.org/ontology/>
[a owl:NegativeObjectPropertyAssertion;
 owl:sourceIndividual dbp:Barack_Obama;
 owl:assertionProperty dbpo:born_in;
 owl:targetIndividual dbp:Kenya].
[a owl:NegativeDataPropertyAssertion;
 owl:sourceIndividual dbp:Barack_Obama;
 owl:assertionProperty dbpo:age;
 owl:targetIndividual "60" ].

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
• For example, the class of processes that regulate themselves
• It is also called local reflexivity
• An example: Auto-regulating processes regulate themselves
• narcissists are people who love themselves

Qualified cardinality restrictions
• Qualifies the instances to be counted
• Six varieties: [{Data|Object}|Min|Exact|Max|Cardinality
• For example,
--People with exactly three children who are girls
--People with at least three names
--Each individual has at most one SSN

Qualified cardinality restrictions
• Done via new properties with domain owl:Restriction, namely (min|max)/QualifiedCardinality and onClass
• Example: people with exactly three children who are girls
  [a owl:restriction;
   owl:onProperty has_child;
   owl:onClass [owl:subClassOf FemalePerson;
   owl:subClassOf Minor].
   QualifiedCardinality "3".

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
--Deals with object properties
--Pairwise disjointness can be asserted
--E.g., connectedTo and contiguousWith
• DisjointDataProperties
--Deals with data properties
--Pairwise disjointness can be asserted
--E.g., startTime and endTime of a surgery

Property chain inclusion
• Properties can be defined as a composition of other properties
• The brother of your parent is your uncle
  uncle owl:propertyChainAxiom (parent brother)
**Keys**

- Individuals can be identified uniquely
- Identification can be done using
  - A data property
  - An object property or
  - A set of properties
- Example
  
  
  ```
  foaf:Person owl:hasKey (foaf:mbox);
  owl:hasKey (homePhone :foaf:name).
  ```


**Extended datatypes**

- Extra datatypes
  
  ```
  Examples: owl:real, owl:rationa1, xsd:pattern
  ```

- Datatype restrictions
  
  ```
  For example, adult has an age >= 18
  ```

- Datatype definitions
  
  ```
  New datatypes
  ```

- Extended datatypes
  
  ```
  Data range combinations
  ```

**An example**

```
:Teenager rdfs:subClassOf __x .
__x a rdfs:Class ;
owl:Restriction ;
owl:onProperty :age ;
owl:onDataRange (xsd:integer)

:_x1 a owl:Restriction ;
owl:equalTo "13"^^xsd:integer .
:_x2 a owl:Restriction ;
owl:equalTo "19"^^xsd:integer .
```

**Punning**

- An OWL 1 DL thing can’t be both a class and an 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**

- 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
  ```

**Annotations**

- In OWL annotations comprise information that carries no official meaning
  
  ```
  Some properties in OWL 1 are considered as annotation properties, e.g., owl:comment, rdf:label and rdf:seeAlso
  ```

- OWL 2 allowed RDF reification as a way to say things about triples, again w/o official meaning
  
  ```
  [a rdf:Statement ;
  rdf:subject: Barack_Obama;
  rdf:property: dbpo:born_in;
  rdf:object: "Kenya";
  :certainty "0.051"
  ]
  ```

**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 ;
owl:range owl:Person .

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

Inverse object properties

- Some object property can be inverse of another property
- For example, partOf and hasPart
- ObjectInverseOf (partOf): this expression represents the inverse property of: part of
- This makes writing ontologies easier by avoiding the need to 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, called Profiles, designed for different use cases

OWL 2 Profiles

- OWL 2 defines three different tractable profiles:
  - EL: polynomial time reasoning for schema and 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
- It does not allow disjunction and 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:
  \[ A \sqsubseteq B \quad \forall \land \forall \sqsubseteq C \quad A \sqsubseteq B , B \sqsubseteq C \quad \exists B \land \exists C \sqsubseteq D \]
- Saturate using inference rules:
  \[ A \sqsubseteq C \quad \exists \forall \sqsubseteq D \]
- Extension to Horn fragment requires (many) more rules
Saturation-based Technique (basics)
Example: infer that a heart transplant is a kind of organ transplant
- HeartTransplant = Transplant || Skin Organ
- Heart = Organ

HeartTransplant = Transplant || Skin Organ
- Heart = Organ

HeartTransplant = Transplant || Skin Organ
- Heart = Organ

Heart = Organ

HeartTransplant = Transplant || Skin Organ
- Heart = Organ

HeartTransplant = Transplant || Skin Organ
- Heart = Organ

Heart = Organ

HeartTransplant = Transplant || Skin Organ
- Heart = Organ

HeartTransplant = Transplant || Skin Organ
- Heart = Organ

Heart = Organ

HeartTransplant = Transplant || Skin Organ
- Heart = Organ

HeartTransplant = Transplant || Skin Organ
- Heart = Organ

Heart = Organ
Saturation-based Technique (basics)

Example:

\[
\text{HeartTransplant} \subseteq \text{Transplant} \land \text{Heart} \subseteq \text{Heart} \\
\text{Heart} \subseteq \text{Organ} \\
\text{Heart} \subseteq \text{Heart} \\
\text{Transplant} \land \text{Heart} \subseteq \text{Heart} \land \text{Heart} \\
\text{Organ} \subseteq \text{Organ} \\
\end{align}
\]

Saturation-based Technique (basics)

Example:

\[
\text{HeartTransplant} \subseteq \text{Transplant} \land \text{Heart} \subseteq \text{Heart} \\
\text{Heart} \subseteq \text{Organ} \\
\text{Heart} \subseteq \text{Heart} \\
\text{Transplant} \land \text{Heart} \subseteq \text{Heart} \land \text{Heart} \\
\text{Organ} \subseteq \text{Organ} \\
\end{align}
\]

Saturation-based Technique (basics)

Performance with large bio-medical ontologies

<table>
<thead>
<tr>
<th>Concept</th>
<th>OBO</th>
<th>OMIM</th>
<th>Galen v0</th>
<th>Galen v7</th>
<th>SNOMED</th>
</tr>
</thead>
<tbody>
<tr>
<td>FACT++</td>
<td>15.24</td>
<td>6.05</td>
<td>465.35</td>
<td>---</td>
<td>60.37</td>
</tr>
<tr>
<td>RECENT</td>
<td>195.52</td>
<td>165.47</td>
<td>45.72</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>PELLET</td>
<td>72.02</td>
<td>26.47</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>CEL</td>
<td>1.84</td>
<td>5.78</td>
<td>---</td>
<td>---</td>
<td>1185.70</td>
</tr>
<tr>
<td>PB</td>
<td>1.17</td>
<td>3.57</td>
<td>0.32</td>
<td>8.56</td>
<td>40.64</td>
</tr>
<tr>
<td>Speed-Up</td>
<td>1.67X</td>
<td>1.61X</td>
<td>140X</td>
<td>---</td>
<td>13.15X</td>
</tr>
</tbody>
</table>

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
• These restrictions
  – 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 the DL knowledge base

Query Rewriting Technique (basics)

Given ontology O and query Q, use O to rewrite Q as O' such that, for any set of ground facts A:

\[\text{ans}(Q, O, A) = \text{ans}(Q', O, A)\]

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

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 ACF) by delegating query answering to a Datalog engine
Query Rewriting Technique (basics)

- Example:
   Doctor C SNo=P, Patient
   Consultant C Doctor
   \[\text{in}(x,y) \rightarrow \text{Doctor}(x)\]
   \[\text{Patient}(y) \rightarrow \text{Doctor}(x)\]
   \[\text{Doctor}(x) \rightarrow \text{Consultant}(x)\]

- For DL-Lite, result is a union of conjunctive queries (UCQ)

OWL 2 RL

- The RL acronym reflects its relation to Rule Languages
- OWL 2 RL is designed to accommodate
  - OWL 2 applications that can trade the full expressivity of the language for efficiency
  - RDF(S) applications that need some added expressivity from OWL 2
- Not allowed: existential quantification to a class, union and disjoint union to class expressions
- These restrictions allow OWL 2 RL to be implemented using rule-based technologies such as rule extended DBMSs, Jess, Prolog, etc.

Profiles

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

Data can be stored/left in RDBMS
- Relationship between ontology and DB defined by mappings, e.g.:
  - Doctor \(\rightarrow\) SELECT Name FROM Doctor
  - Patient \(\rightarrow\) SELECT Name FROM Patient
  - treat \(\rightarrow\) SELECT DName, PName FROM Treats
- ULC translated into SQL query:
  SELECT Name FROM Doctor UNION
  SELECT DName FROM Treats, Patient WHERE PName=Name
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 – OWL 2 EL, OWL 2 QL and OWL 2 RL, and their necessity have been presented

Thank you!

Questions?