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 2009
- All new features were justified by use cases and examples
- Most OWL software supports OWL now

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 2 of the classes in the 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
- Will be 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) and also either a minor or adult (but not both)

  foaf:Person
  owl:disjointUnionOf (:MalePerson :FemalePerson);
  owl:disjointUnionOf (:Minor :Adult) .

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" ] .
```

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:
  - 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
  - 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*
- 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 (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 chain inclusion

- Properties can be defined as a composition of other properties
- The brother of your parent is your uncle
  :uncle owl:propertyChainAxion (:parent :brother) .
- Your parent’s sister’s spouse is your uncle
  :uncle owl:propertyChainAxion (:parent :sister :spouse) .

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 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 .
```

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

- 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 inverse of another property
- For example, partOf and hasPart
- The ObjectInverseOf(*:partOf*) 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

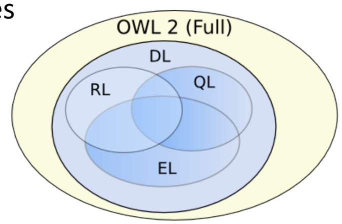
OWL 2 Profiles

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:

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

- 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
- 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

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

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:

  ans(Q, O, A) = ans(Q^0, ;, A)
- Resolution based query rewriting
  - **Clausify** ontology axioms
  - **Saturate** (clausified) ontology and query using resolution
  - **Prune** redundant query clauses

Query Rewriting Technique (basics)

- Example:

  $\text{Doctor} \sqsubseteq \exists \text{treats}.\text{Patient}$

  $\text{Consultant} \sqsubseteq \text{Doctor}$

$$Q(x) \leftarrow \text{treats}(x, y) \wedge \text{Patient}(y)$$

Q(x) is our query: Who treats people who are patients?

Query Rewriting Technique (basics)

- Example:

Doctor ⊑ ∃treats.Patient
Consultant ⊑ Doctor

$\text{treats}(x, f(x)) \leftarrow \text{Doctor}(x)$
$\text{Patient}(f(x)) \leftarrow \text{Doctor}(x)$
$\text{Doctor}(x) \leftarrow \text{Consultant}(x)$

$Q(x) \leftarrow \text{treats}(x, y) \land \text{Patient}(y)$

Translate the DL expressions into rules.

Note the use of f(x) as a Skolem individual. If you are a doctor then you treat someone and that someone is a patient

Query Rewriting Technique (basics)

- Example:

Doctor ⊑ ∃treats.Patient
Consultant ⊑ Doctor

$\text{treats}(x, f(x)) \leftarrow \text{Doctor}(x)$
$\text{Patient}(f(x)) \leftarrow \text{Doctor}(x)$
$\text{Doctor}(x) \leftarrow \text{Consultant}(x)$

$Q(x) \leftarrow \text{treats}(x, y) \land \text{Patient}(y)$

For each rule in the rules version of the KB we want to enhance the query, so that we need not use the rule in the KB.

Query Rewriting Technique (basics)

- Example:

Doctor ⊑ ∃treats.Patient
Consultant ⊑ Doctor

$\text{treats}(x, f(x)) \leftarrow \text{Doctor}(x)$
$\text{Patient}(f(x)) \leftarrow \text{Doctor}(x)$
$\text{Doctor}(x) \leftarrow \text{Consultant}(x)$

$Q(x) \leftarrow \text{treats}(x, y) \land \text{Patient}(y)$
$Q(x) \leftarrow \text{Doctor}(x) \land \text{Patient}(f(x))$

Since Doctor(X) implies treats(x, f(x)) we can replace it, but we have to also unify f(x) with y, so we edn up with the second way of satisfying our query Q(x).

Query Rewriting Technique (basics)

- Example:

Doctor ⊑ ∃treats.Patient
Consultant ⊑ Doctor

$\text{treats}(x, f(x)) \leftarrow \text{Doctor}(x)$
$\text{Patient}(f(x)) \leftarrow \text{Doctor}(x)$
$\text{Doctor}(x) \leftarrow \text{Consultant}(x)$

$Q(x) \leftarrow \text{treats}(x, y) \land \text{Patient}(y)$
$Q(x) \leftarrow \text{Doctor}(x) \land \text{Patient}(f(x))$

Query Rewriting Technique (basics)

- Example:

Doctor ⊑ ∃treats.Patient
Consultant ⊑ Doctor

$\text{treats}(x, f(x)) \leftarrow \text{Doctor}(x)$ $\quad Q(x) \leftarrow \text{treats}(x, y) \wedge \text{Patient}(y)$
$\text{Patient}(f(x)) \leftarrow \text{Doctor}(x)$ $\quad Q(x) \leftarrow \text{Doctor}(x) \wedge \text{Patient}(f(x))$
$\text{Doctor}(x) \leftarrow \text{Consultant}(x)$ $\quad Q(x) \leftarrow \text{treats}(x, f(x)) \wedge \text{Doctor}(x)$

Applying the KB second rule to the 1st query rule gives us another way to solve the Q(x)

Query Rewriting Technique (basics)

- Example:

Doctor ⊑ ∃treats.Patient
Consultant ⊑ Doctor

$\text{treats}(x, f(x)) \leftarrow \text{Doctor}(x)$ $\quad Q(x) \leftarrow \text{treats}(x, y) \wedge \text{Patient}(y)$
$\text{Patient}(f(x)) \leftarrow \text{Doctor}(x)$ $\quad Q(x) \leftarrow \text{Doctor}(x) \wedge \text{Patient}(f(x))$
$\text{Doctor}(x) \leftarrow \text{Consultant}(x)$ $\quad Q(x) \leftarrow \text{treats}(x, f(x)) \wedge \text{Doctor}(x)$

Query Rewriting Technique (basics)

- Example:

Doctor ⊑ ∃treats.Patient
Consultant ⊑ Doctor

$\text{treats}(x, f(x)) \leftarrow \text{Doctor}(x)$ $\quad Q(x) \leftarrow \text{treats}(x, y) \wedge \text{Patient}(y)$
$\text{Patient}(f(x)) \leftarrow \text{Doctor}(x)$ $\quad Q(x) \leftarrow \text{Doctor}(x) \wedge \text{Patient}(f(x))$
$\text{Doctor}(x) \leftarrow \text{Consultant}(x)$ $\quad Q(x) \leftarrow \text{treats}(x, f(x)) \wedge \text{Doctor}(x)$
$Q(x) \leftarrow \text{Doctor}(x)$

Since Doctor(x) implies treats(x, f(x)) we can derive Q(X) if Doctor(x) and Doctor(x), which reduces to the third query rule.

Query Rewriting Technique (basics)

- Example:

Doctor ⊑ ∃treats.Patient
Consultant ⊑ Doctor

$\text{treats}(x, f(x)) \leftarrow \text{Doctor}(x)$ $\quad Q(x) \leftarrow \text{treats}(x, y) \wedge \text{Patient}(y)$
$\text{Patient}(f(x)) \leftarrow \text{Doctor}(x)$ $\quad Q(x) \leftarrow \text{Doctor}(x) \wedge \text{Patient}(f(x))$
$\text{Doctor}(x) \leftarrow \text{Consultant}(x)$ $\quad Q(x) \leftarrow \text{treats}(x, f(x)) \wedge \text{Doctor}(x)$
$Q(x) \leftarrow \text{Doctor}(x)$

Query Rewriting Technique (basics)

- Example:

Doctor $\sqsubseteq$ ∃treats.Patient
Consultant $\sqsubseteq$ Doctor

$\mathsf{treats}(x, f(x)) \leftarrow \mathsf{Doctor}(x)$
$\mathsf{Patient}(f(x)) \leftarrow \mathsf{Doctor}(x)$
$\mathsf{Doctor}(x) \leftarrow \mathsf{Consultant}(x)$

$Q(x) \leftarrow \mathsf{treats}(x, y) \wedge \mathsf{Patient}(y)$
$Q(x) \leftarrow \mathsf{Doctor}(x) \wedge \mathsf{Patient}(f(x))$
$Q(x) \leftarrow \mathsf{treats}(x, f(x)) \wedge \mathsf{Doctor}(x)$
$Q(x) \leftarrow \mathsf{Doctor}(x)$
$Q(x) \leftarrow \mathsf{Consultant}(x)$

Query Rewriting Technique (basics)

- Example:

Doctor $\sqsubseteq$ ∃treats.Patient
Consultant $\sqsubseteq$ Doctor

$\mathsf{treats}(x, f(x)) \leftarrow \mathsf{Doctor}(x)$
$\mathsf{Patient}(f(x)) \leftarrow \mathsf{Doctor}(x)$
$\mathsf{Doctor}(x) \leftarrow \mathsf{Consultant}(x)$

$Q(x) \leftarrow \mathsf{treats}(x, y) \wedge \mathsf{Patient}(y)$
$Q(x) \leftarrow \mathsf{Doctor}(x) \wedge \mathsf{Patient}(f(x))$
$Q(x) \leftarrow \mathsf{treats}(x, f(x)) \wedge \mathsf{Doctor}(x)$
$Q(x) \leftarrow \mathsf{Doctor}(x)$
$Q(x) \leftarrow \mathsf{Consultant}(x)$

Query Rewriting Technique (basics)

- Example:

Doctor $\sqsubseteq$ ∃treats.Patient
Consultant $\sqsubseteq$ Doctor

$\mathsf{treats}(x, f(x)) \leftarrow \mathsf{Doctor}(x)$
$\mathsf{Patient}(f(x)) \leftarrow \mathsf{Doctor}(x)$
$\mathsf{Doctor}(x) \leftarrow \mathsf{Consultant}(x)$

$Q(x) \leftarrow \mathsf{treats}(x, y) \wedge \mathsf{Patient}(y)$
~~$Q(x) \leftarrow \mathsf{Doctor}(x) \wedge \mathsf{Patient}(f(x))$~~
~~$Q(x) \leftarrow \mathsf{treats}(x, f(x)) \wedge \mathsf{Doctor}(x)$~~
$Q(x) \leftarrow \mathsf{Doctor}(x)$
$Q(x) \leftarrow \mathsf{Consultant}(x)$

Remove useless redundant query rules

Query Rewriting Technique (basics)

- Example:

Doctor $\sqsubseteq$ ∃treats.Patient
Consultant $\sqsubseteq$ Doctor

$\mathsf{treats}(x, f(x)) \leftarrow \mathsf{Doctor}(x)$
$\mathsf{Patient}(f(x)) \leftarrow \mathsf{Doctor}(x)$
$\mathsf{Doctor}(x) \leftarrow \mathsf{Consultant}(x)$

$Q(x) \leftarrow \mathsf{treats}(x, y) \wedge \mathsf{Patient}(y)$
~~$Q(x) \leftarrow \mathsf{Doctor}(x) \wedge \mathsf{Patient}(f(x))$~~
~~$Q(x) \leftarrow \mathsf{treats}(x, f(x)) \wedge \mathsf{Doctor}(x)$~~
$Q(x) \leftarrow \mathsf{Doctor}(x)$
$Q(x) \leftarrow \mathsf{Consultant}(x)$

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

Query Rewriting Technique (basics)

- Data can be stored/left in **RDBMS**
- Relationship between ontology and DB defined by **mappings**, e.g.:

Doctor	↦	SELECT Name FROM Doctor
Patient	↦	SELECT Name FROM Patient
treats	↦	SELECT DName, PName FROM Treats

- UCQ translated into **SQL query**:

SELECT Name FROM Doctor UNION
SELECT DName FROM Treats, Patient WHERE PName=Name

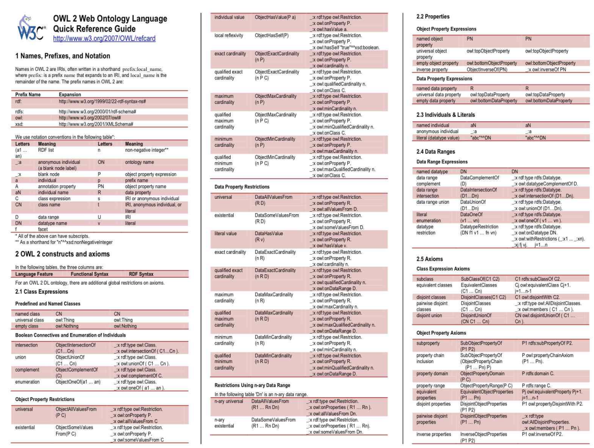
OWL 2 RL

- 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


OWL 2 Web Ontology Language
Quick Reference Guide
http://www.w3.org/2007/OWL/refcard

Key OWL 2 Documents

Part	Type	Document
1	For Users	Document Overview. A quick overview of the OWL 2 specification that includes a description of its relationship to OWL 1. This it the starting point and primary reference point for OWL 2.
2	Core Specification	Structural Specification and Functional-Style Syntax defines the constructs of OWL 2 ontologies in terms of both their structure and a functional-style syntax, and defines OWL 2 DL ontologies in terms of global restrictions on OWL 2 ontologies.
3	Core Specification	Mapping to RDF Graphs defines a mapping of the OWL 2 constructs into RDF graphs, and thus defines the primary means of exchanging OWL 2 ontologies in the Semantic Web.
4	Core Specification	Direct Semantics defines the meaning of OWL 2 ontologies in terms of a model-theoretic semantics.
5	Core Specification	RDF-Based Semantics defines the meaning of OWL 2 ontologies via an extension of the RDF Semantics.
6	Core Specification	Conformance provides requirements for OWL 2 tools and a set of test cases to help determine conformance.
7	Specification	Profiles defines three sub-languages of OWL 2 that offer important advantages in particular applications scenarios.
8	For Users	OWL 2 Primer provides an approachable introduction to OWL 2, including orientation for those coming from other disciplines.
9	For Users	OWL 2 New Features and Rationale provides an overview of the main new features of OWL 2 and motivates their inclusion in the language.
10	For Users	OWL 2 Quick Reference Guide provides a brief guide to the constructs of OWL 2, noting the changes from OWL 1.
11	Specification	XML Serialization defines an XML syntax for exchanging OWL 2 ontologies, suitable for use with XML tools like schema-based editors and XQuery/XPath.
12	Specification	Manchester Syntax (WG Note) defines an easy-to-read, but less formal, syntax for OWL 2 that is used in some OWL 2 user interface tools and is also used in the Primer.
13	Specification	Data Range Extension: Linear Equations (WG Note) specifies an optional extension to OWL 2 which supports advanced constraints on the values of properties.

http://w3.org/TR/2009/WD-owl2-overview-20090421/

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