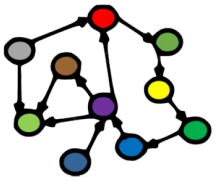
Welcome to INFO216:
Knowledge Graphs
Spring 2022

Andreas L Opdahl
<Andreas.Opdahl@uib.no>

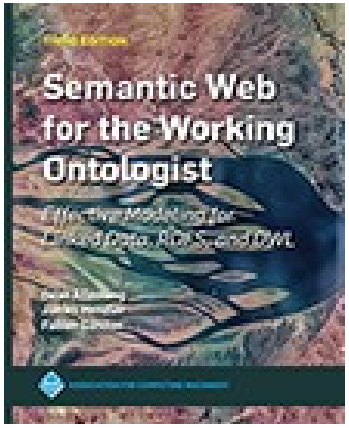
Session 11: Formal ontologies (OWL-DL)

- Themes:
  - restriction classes
  - anatomy of OWL
  - more examples of *Turtle* (+ *Manchester Syntax*)
  - *a little about the owlready2 API and Protegè reasoners*
  - builds on *S08: Ontologies (OWL, "RDFS Plus")*
    - what and why?
    - basic OWL constructs
    - complex classes
  - builds on *S10: Description Logic (DL)*
    - description logic, decision problems

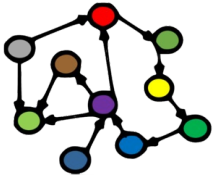


Readings

- Sources:
  - Allemang, Hendler & Gandon (2020): Semantic Web for the Working Ontologist, 3rd edition (chapters 12-13, but chapters 11-12 in the 2nd edition)
  - Blumauer & Nagy (2020): Knowledge Graph Cookbook – Recipes that Work (e.g., pages 105-109, 123-124, *supplementary*)
- Material at http://wiki.uib.no/info216:
  - OWL 2 Overview (http://www.w3.org/TR/owl-overview/)
  - OWL 2 Primer (http://www.w3.org/TR/owl-primer/):
    - show: Turtle and Manchester syntax
    - hide: other syntaxes

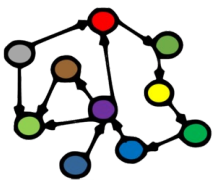


Web Ontology Language (OWL)



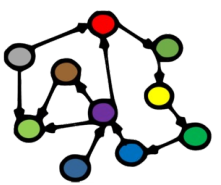
RDFS is a useful starting point... (S07)

- But there's lots of simple stuff it cannot express, e.g.:
  - “every ancestor of an ancestor is an ancestor too”
  - “the BirthNumber of a Person is unique”
  - “a Republic has exactly one President”
  - “a FootballTeam has 11 players, a VolleyballTeam only 6”
  - “a StringQuartet has two violins but only one viola and one cello”
  - “classes with different URIs actually represent the same class”
  - “resources with different URIs represent the same resource”
  - “properties with different URIs are actually the same”
  - “two individuals are different”, “two classes are disjoint”
  - “a class is a union (or intersection) of other classes”
  - “a class is a negation of another class”
- *OWL expresses all this and more!*

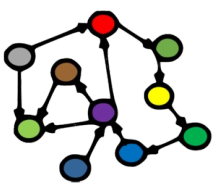


What does OWL offer?

- Extensions of RDFS, e.g.:
  - more *specific types* of properties
  - *identical and different* classes, properties, individuals
  - *defining new classes*:
    - complex classes (union, intersection, complement)
    - property restrictions, enumeration of individuals
  - *defining new properties* based on existing ones
  - *mathematical formality* (for large parts of OWL)
    - certain OWL ontologies are also logical systems
      - *description logic (DL)*
      - *OWL DL* has good computational behaviours
    - *(appearance of)* more powerful *entailments*

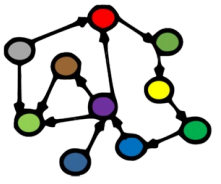


The Core OWL Concepts



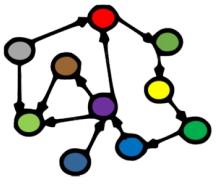
Classes, properties, and individuals

- Web Ontology Language (OWL):
  - builds on RDF and RDFS (but not SKOS)
  - uses classes and properties from RDF and RDFS
  - adds precision and formality
- Basic OWL-concepts:
  - owl:Class rdfs:subClassOf rdfs:Class .
  - “owl:Property” rdfs:subClassOf rdf:Property .
  - “owl:Individual” rdfs:subClassOf rdfs:Resource .
    - good practice: keep these three *disjoint*, i.e., no resource has more than one of them as *rdf:type*
    - *in OWL DL, this is a requirement...*



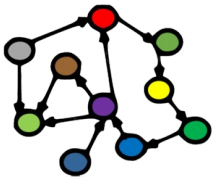
Building blocks

- OWL 2 has three building blocks:
  - *entities:*
    - refer to real-world entities using URIs
    - owl:NamedClass, owl:NamedIndividual
    - owl:ObjectProperty, owl:DatatypeProperty, owl:AnnotationProperty, owl:ObjectProperty
  - *axioms:*
    - basic statements the OWL ontology expresses
    - every triple in the RDF graph is an axiom
  - *expressions:*
    - combining simpler entities (classes, individuals, or properties) to define more complex ones
    - based on *constructors*



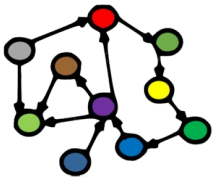
Building blocks

- OWL 2 has three building blocks:
  - *entities:*
    - refer to real-world entities using URIs
    - owl:NamedClass, owl:NamedIndividual
    - owl:ObjectProperty, owl:DatatypeProperty, owl:AnnotationProperty, owl:ObjectProperty
  - *axioms:* ← *can be true or false!*
    - basic statements the OWL ontology expresses
    - every triple in the RDF graph is an axiom
  - *expressions:*
    - combining simpler entities (classes, individuals, or properties) to define more complex ones
    - based on *constructors*



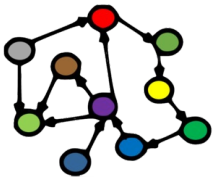
Things and named individuals

- owl:Thing:
  - is equivalent to *rdfs:Resource*
  - similar to the top concept in DL
- owl:Nothing
  - is the empty set
  - no resource has it as its *rdf:type*
  - similar to the bottom concept in DL
- owl:NamedIndividual
  - is an *owl:Thing* with an URI
  - defined in OWL2 DL



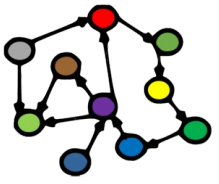
Named and constructed classes

- owl:NamedClass (with an URI):
  - semantics are given by:
    - URI-s, labels and other annotations
    - domain, range, subClassOf and other relationships
- *Constructed* (or *complex*) owl:Class:
  - built from existing classes, properties, individuals
    - which can be named *or anonymous*
  - constructed classes are *anonymous upon declaration,*
    - but can be *named* later
  - *unions, intersections* and *negations* of existing classes
  - *restrictions* on existing properties
  - *enumeration* of existing individuals



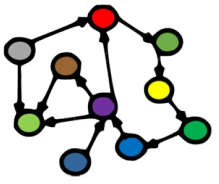
Object and datatype properties

- RDF triples: object is either a resource or a literal
  - OWL has two corresponding types of predicates
- owl:ObjectProperty:
  - rdfs:range (“verdiområde”) is an OWL-class of individuals
  - corresponds to RDF triples with a *resource* object
- owl:DatatypeProperty:
  - rdfs:range is an RDFS-datatype
  - corresponds to RDF triples with a *literal* object
- rdfs:domain (“definisjonsmengden”) for OWL properties is always an OWL-class of individuals



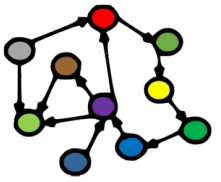
Annotation and ontology properties

- Annotation properties are used to annotate
  - whole *ontologies* (e.g., version)
  - ontology *entities* (*classes, individuals, properties*)
  - ontology *axioms* (*triples*)
  - for example: *rdfs:comment...*
- Ontology properties are used to manage ontologies
  - for example: *owl:imports...*
- They have *RDFS-semantics*
    - but no specific *description logic* (DL) semantics
    - often not “counted” alongside object and datatype properties



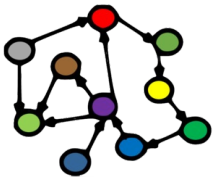
Summary: basic OWL types

- owl:Thing, owl:Nothing, owl:NamedIndividual
- owl:NamedClass, owl:Class
- owl:ObjectProperty, owl:DatatypeProperty
- owl:AnnotationProperty, owl:OntologyProperty



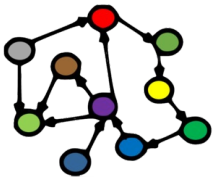
Summary: more precise properties (S06)

- owl:inverseOf
- owl:SymmetricProperty, owl:AsymmetricProperty
- owl:ReflexiveProperty, owl:IrreflexiveProperty
- owl:TransitiveProperty
- owl:FunctionalProperty, owl:InverseFunctionalProperty
- owl:hasKey
- Also:
  - negated properties *(today!)*
  - chained properties, e.g.:
    fam:hasGrandparent
    owl:propertyChainAxiom (:hasParent :hasParent) .

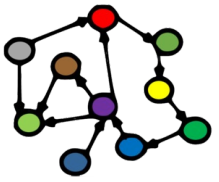


Summary: sameness and difference (S06)

- Individuals:
  - pairwise: owl:sameAs, owl:differentFrom
  - groupwise difference: owl:AllDifferent
- Classes:
  - pairwise: owl:equivalentClass, owl:disjointWith
  - groupwise difference: owl:AllDisjointClasses
- Properties:
  - pairwise: equivalentProperty, propertyDisjointWith
  - groupwise difference: owl:AllDisjointProperties
- Membership in the groups:
  - owl:distinctMembers *(preferred)* or owl:members

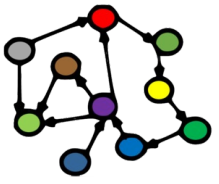


Complex OWL classes



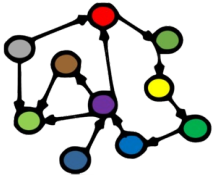
Enumeration classes

- An *enumeration class* is defined by exhaustively listing all its member individuals, e.g.:
  - [a owl:Class ;
    owl:oneOf (cal:Spring ... cal:Winter)] .
- An enumeration class is *closed*
  - there are no other member individuals
  - ensured by using *RDF Collections:*
    - rdf:List, rdf:first, rdf:rest, rdf:nil
- Does *not* imply that the individuals are distinct
  - this must be stated explicitly



Enumeration classes

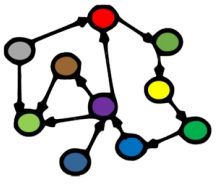
- An *enumeration class* is defined by exhaustively listing all its member individuals, e.g.:
  - [*a owl:Class ;*
    owl:oneOf (cal:Spring ... cal:Winter)] .
- An enumeration class is *closed*
  - there are no other member individuals
  - ensured by using *RDF Collections:*
    - rdf:List, rdf:first, rdf:rest, rdf:nil
- Does *not* imply that the individuals are distinct
  - this must be stated explicitly



Union classes

- A union class contains all the individuals in *either of* two or more other classes, e.g.,
  - fam:Parent
    a owl:Class;
    owl:unionOf (fam:Father fam:Mother) .
- Entailment rule:
  - if *C owl:equivalentClass owl:unionOf (C1... Cn)* then
    - C1 rdfs:subClassOf C . ... Cn rdfs:subClassOf C .
- why not say just, e.g.,:
  - fam:Father rdfs:subClassOf fam:Parent .
  - fam:Mother rdfs:subClassOf fam:Parent .

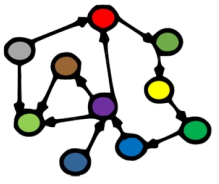
?



Intersection classes

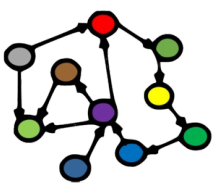
- An intersection class contains all the individuals in *all of* two or more other classes, e.g.
  - uib:StudentAssistant
    a owl:Class;
    owl:intersectionOf (uib:Student uib:Teacher) .
- Entailment rule:
  - if *C owl:equivalentClass owl:intersectionOf (C1... Cn)* then
    - C rdfs:subClassOf C1 . ... C rdfs:subClassOf Cn .
- why not say, e.g.:
  - uib:StudentAssistant rdfs:subClassOf uib:Student .
  - uib:StudentAssistant rdfs:subClassOf uib:Teacher .

?



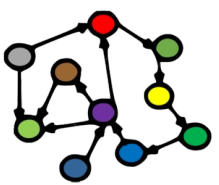
Complement classes

- A complement class contains all the individuals *that are not* in another class:
  - fam:Father owl:complementOf fam:Mother .



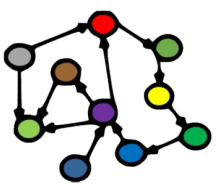
Complement classes

- A complement class contains all the individuals *that are not* in another class:
  - fam:Father owl:complementOf fam:Mother .
  - *...but is this correct?!*



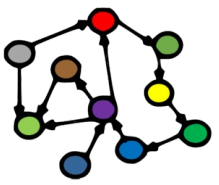
Complement classes

- A complement class contains all the individuals *that are not* in another class:
  - fam:Father
    a owl:Class;
    owl:complementOf fam:Mother .



Complement classes

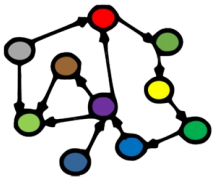
- A complement class contains all the individuals *that are not* in another class:
  - fam:Father
    owl:intersectionOf (
    fam:Parent
    owl:complementOf fam:Mother
) .



Complement classes

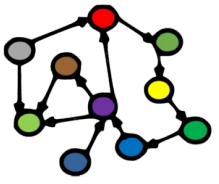
- A complement class contains all the individuals *that are not* in another class:
  - fam:Father

```
owl:intersectionOf (
        fam:Parent
        [   a owl:Class ;
            owl:complementOf fam:Mother
        ]
) .
```



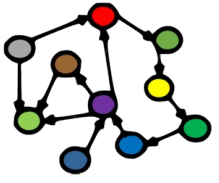
Complement classes

- A complement class contains all the individuals *that are not* in another class:
  - fam:Father
    owl:intersectionOf (
      fam:Parent
      [owl:complementOf fam:Mother]
) .



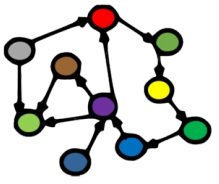
Closed World Assumption (CWA)

- Whenever something is not explicitly stated in the ontology, can we assume that the opposite is the case?
  - DBpedia only lists three James Dean movies – can we thus assume that he only played in three?
- Classical logic and many ICT languages assume so:
  - this is the *"Closed World Assumption" (CWA)*
- *In RDF and OWL, we do not assume that something is false just because it is not stated*
  - this is the *"Open World Assumption" (OWA)*

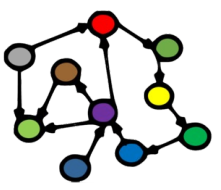


Summary: complex classes

- owl:oneOf
- owl:unionOf
- owl:intersectionOf
- owl:complementOf (and the *CWA*)
- owl:NegativePropertyAssertion, owl:sourceIndividual, owl:assertionProperty, owl:targetIndividual

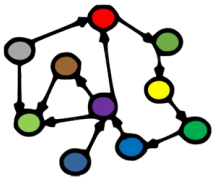


OWL restriction classes



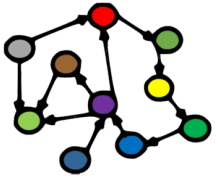
Property value restrictions

- Defining a class by a particular value on one of its properties, e.g.:
  - fam:Woman
    a owl:Restriction ;
    owl:onProperty fam:hasGender ;
    owl:hasValue fam:Female .



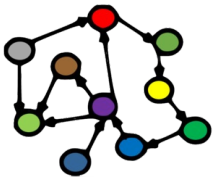
Property value restrictions

- Defining a class by a particular value on one of its properties, e.g.:
  - ~~fam:Woman~~
    ~~a owl:Restriction ;~~
    ~~owl:onProperty fam:hasGender ;~~
    ~~owl:hasValue fam:Female .~~
  - fam:Woman owl:intersectionOf (
    fam:Person
    [a owl:Restriction ;
    owl:onProperty fam:hasGender ;
    owl:hasValue fam:Female]
) .



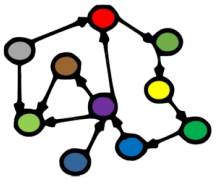
Existential property restrictions

- Defining a class by the existence of a relation (object property) to an individual in (another or the same) class, e.g.:
  - fam:Brother owl:intersectionOf (
    fam:Male
    [a owl:Restriction ;
    owl:onProperty fam:hasSibling ;
    owl:someValuesFrom owl:Thing]
) .
- *owl:someValuesFrom:* each individual in the defined class has *at least one* object property (given by owl:onProperty) to an individual in the other class (given by owl:someValuesFrom)



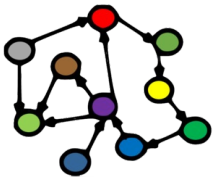
Existential property restrictions

- Defining a class by the existence of a relation (object property) to an individual in (another or the same) class, e.g.:
  - fam:Uncle owl:intersectionOf (
    fam:Male
    [a owl:Restriction ;
    owl:onProperty fam:hasSibling ;
    owl:someValuesFrom fam:Parent]
) .
- *owl:someValuesFrom:* each individual in the defined class has *at least one* object property (given by owl:onProperty) to an individual in the other class (given by owl:someValuesFrom)



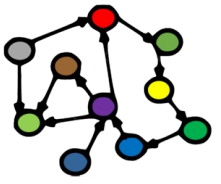
Universal property restrictions

- Defining a class by the necessity of a relation (object property) only to individuals in (another or the same) class, e.g.:
  - fam:HappyFather owl:intersectionOf (
    fam:Male
    [a owl:Restriction ;
    owl:onProperty fam:hasChild ;
    owl:allValuesFrom fam:HappyPerson]
) .



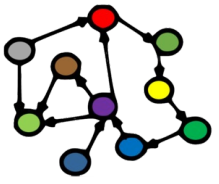
Universal property restrictions

- Defining a class by the necessity of a relation (object property) only to individuals in (another or the same) class, e.g.:
  - fam:HappyFather owl:intersectionOf (
    fam:Male
    [a owl:Restriction ;
    owl:onProperty fam:hasChild ;
    owl:allValuesFrom fam:HappyPerson]
) .
  - *...but is this correct?!*



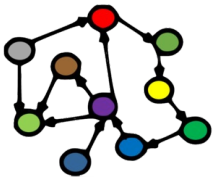
Universal property restrictions

- Defining a class by the necessity of a relation (object property) only to individuals in (another or the same) class, e.g.:
  - fam:HappyFather owl:intersectionOf (
    fam:Father
    [a owl:Restriction ;
    owl:onProperty fam:hasChild ;
    owl:allValuesFrom fam:HappyPerson]
) .



Universal property restrictions

- Defining a class by the necessity of a relation (object property) only to individuals in (another or the same) class, e.g.:
  - fam:HappyFather owl:intersectionOf (
    fam:Male
    [a owl:Restriction ;
    owl:onProperty fam:hasChild ;
    owl:allValuesFrom fam:HappyPerson]
    [a owl:Restriction ;
    owl:onProperty fam:hasChild ;
    owl:someValuesFrom fam:HappyPerson]
) .

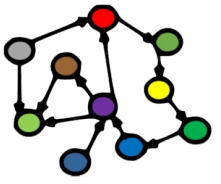


Property restriction

- Using an anonymous property, e.g.:
  - fam:Orphan owl:intersectionOf (

```
fam:Person
[ a owl:Restriction ;
  owl:onProperty [ owl:inverseOf  :hasChild ] ;
  owl:allValuesFrom  fam:Dead
]
[ a owl:Restriction ;
  owl:onProperty [ owl:inverseOf :hasChild ] ;
  owl:someValuesFrom  owl:Thing
]
```

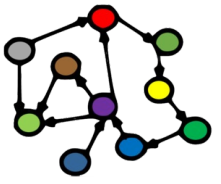
) .



Property self-reflexion

- Defining a class by a *Self* value on one of its properties, e.g.:
  - fam:NarcissisticPerson

```
a                    owl:Restriction ;
owl:onProperty  fam:loves ;
owl:hasSelf       "true"^^xsd:boolean .
```



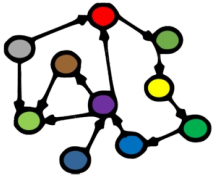
Datatype property restriction

- Restrictions on data range, e.g.:
  - fam:personAge rdfs:range

```
[     a rdfs:Datatype;
      owl:onDatatype  xsd:integer;
      owl:withRestrictions (
            [ xsd:minInclusive  "0"^^xsd:integer ]
            [ xsd:maxInclusive  "150"^^xsd:integer ]  )
] .
```

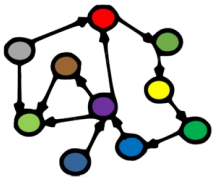
  - :toddlerAge rdfs:range

```
[     a rdfs:Datatype;
      owl:oneOf ( "1"^^xsd:integer  "2"^^xsd:integer)
] .
```



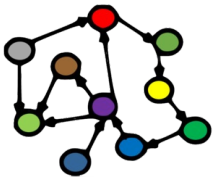
Cardinality restriction

- Defining a class by the number of object values its individuals have for some property, e.g.:
  - music:Quartet owl:intersectionOf (
    music:Ensemble
    [a owl:Restriction ;
    owl:onProperty music:hasInstrument ;
    owl:cardinality 4]
) .
- owl:cardinality gives the *exact cardinality*
  owl:minCardinality gives the *least cardinality*
  owl:maxCardinality gives the *greatest cardinality*



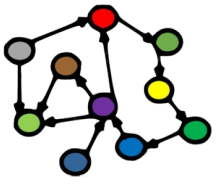
Qualified cardinality restriction

- Defining a class by the number of object values its individuals have *of a given class* for some property, e.g.:
  - pol:Triumvirate owl:intersectionOf (
    pol:PoliticalLeadership
    [a owl:Restriction ;
    owl:onProperty pol:hasMember ;
    owl:qualifiedCardinality 3 ;
    owl:onClass pol:PoliticalLeader]
) .
- owl:qualifiedCardinality gives the *exact cardinality*
  owl:minQualifiedCardinality gives the *least cardinality*
  owl:maxQualifiedCardinality gives the *greatest cardinality*
- *Perhaps the most important addition in OWL2!*



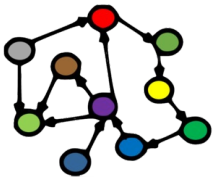
Qualified cardinality restriction

- music:StringQuartet owl:intersectionOf (
  music:MusicalQuartet
  [a owl:Class ;
  owl:onProperty music:hasInstrument ;
  owl:qualifiedCardinality “2” ;
  owl:onClass music:Violin]
  [a owl:Class ;
  owl:onProperty music:hasInstrument ;
  owl:qualifiedCardinality “1” ;
  owl:onClass music:Viola]
  [a owl:Class ;
  owl:onProperty music:hasInstrument ;
  owl:qualifiedCardinality “1” ;
  owl:onClass music:Cello]
) .

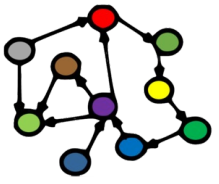


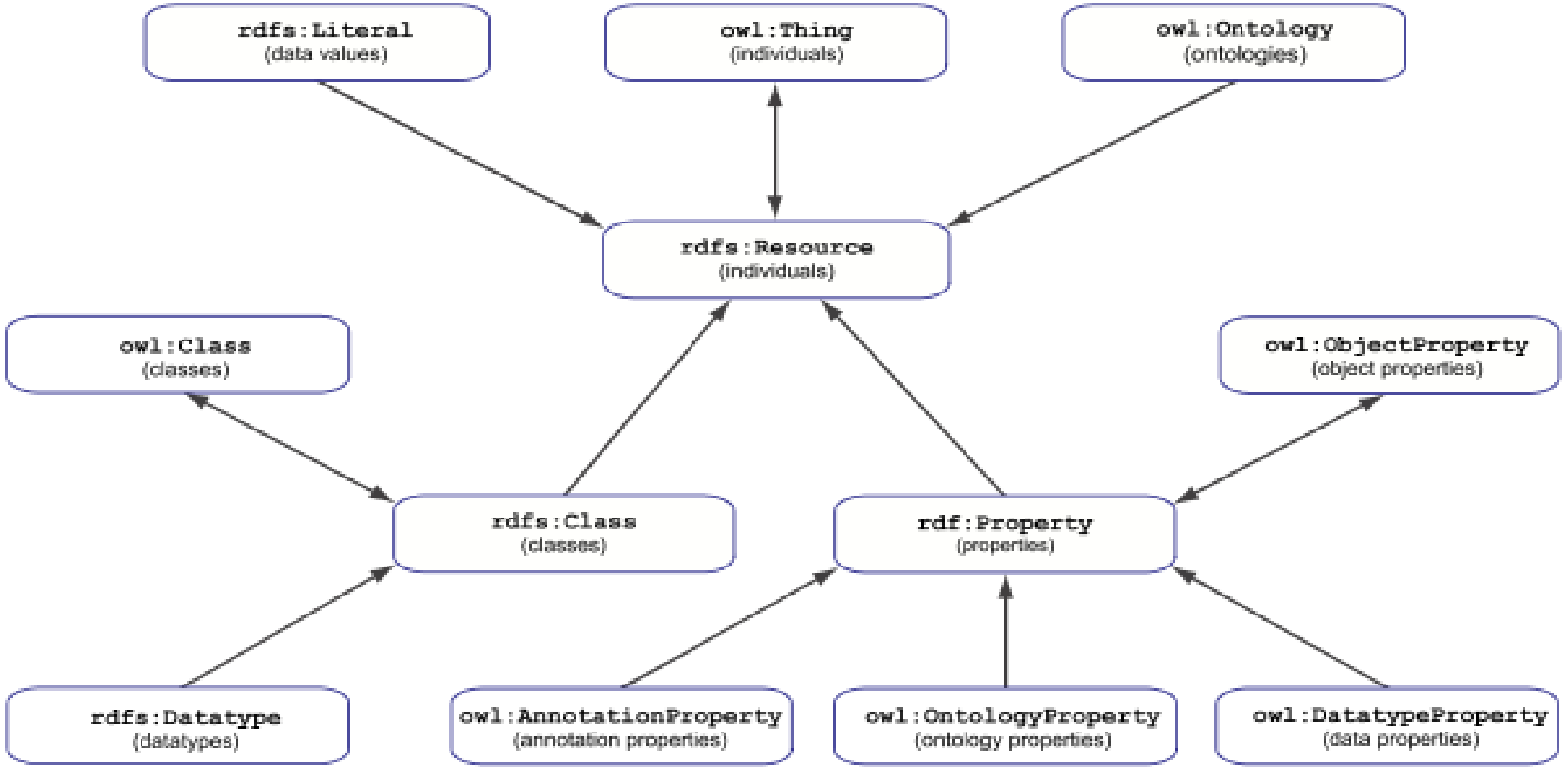
Summary: property restrictions

- owl:Restriction owl:onProperty
- owl:someValuesFrom, owl:allValuesFrom, owl:hasValue
- owl:cardinality, owl:minCardinality, owl:maxCardinality
- owl:onClass, owl:qualifiedCardinality, owl:minQualifiedCardinality, owl:maxQualifiedCardinality



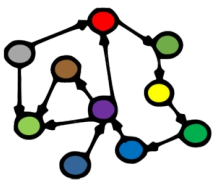
Anatomy of OWL

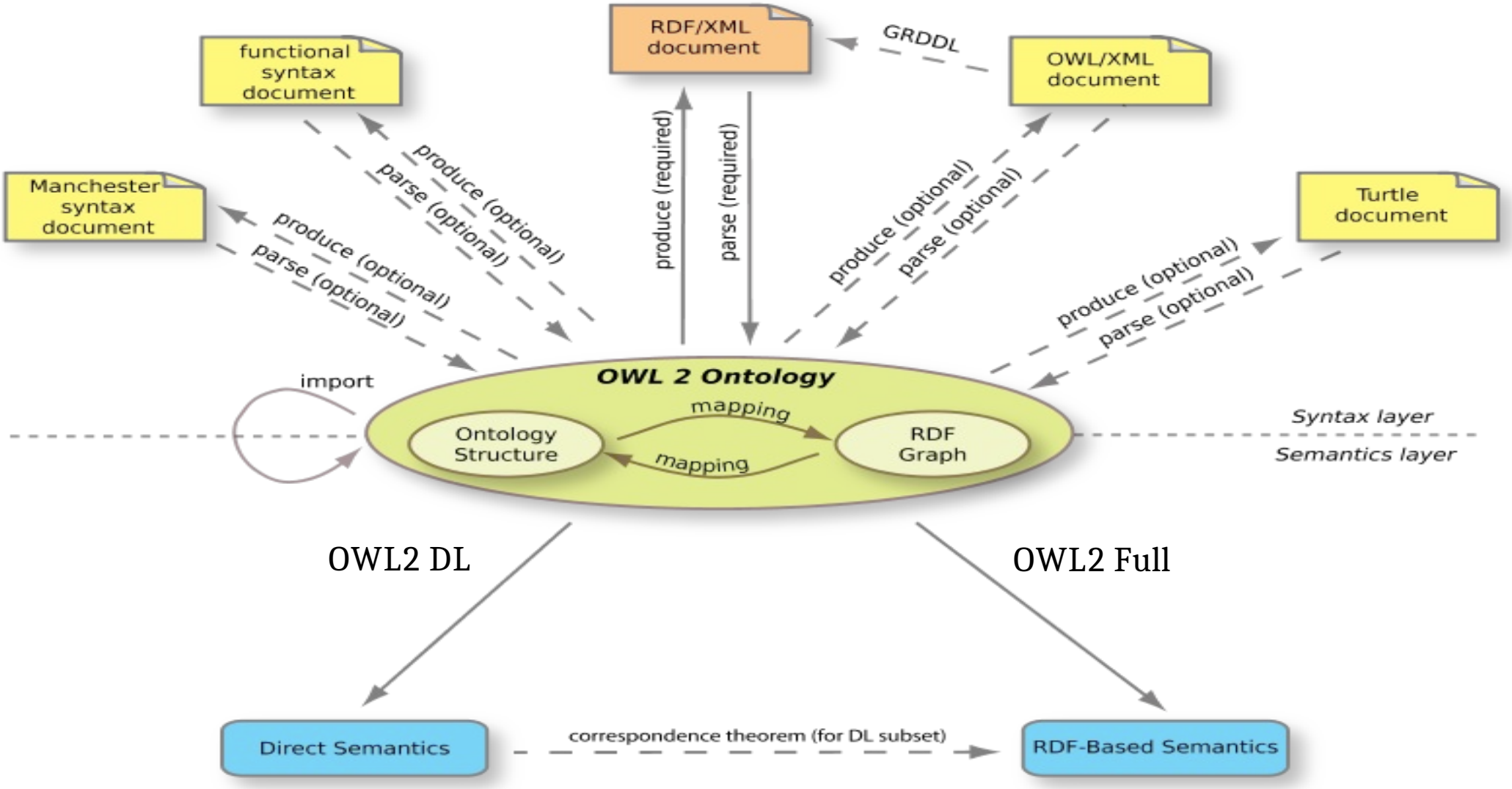



rdfs:Literal
(data values)
owl:Thing
(individuals)
owl:Ontology
(ontologies)
rdfs:Resource
(individuals)
owl:Class
(classes)
owl:ObjectProperty
(object properties)
rdfs:Class
(classes)
rdf:Property
(properties)
rdfs:Datatype
(datatypes)
owl:AnnotationProperty
(annotation properties)
owl:OntologyProperty
(ontology properties)
owl:DatatypeProperty
(data properties)

OWL versions

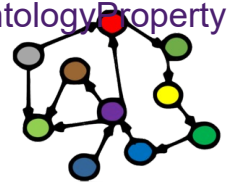
- OWL “1” (2002):
  - OWL Full – anything goes
  - OWL DL – fragment of OWL Full,
    - formal semantics through *description logic*
  - OWL Lite – simple fragment of OWL DL, not much used
- OWL 2 (2008):
  - *backwards compatible* with OWL “1”!
  - OWL2 DL – fragment of OWL2 full, extension of OWL DL
  - *OWL2 DL* – has three further fragments:
    - OWL2 EL – quick reasoning, fragment of OWL2 DL
    - OWL2 RL – rule language, fragment of OWL2 DL
      - OWL LD – for Linked Data
    - OWL2 QL – query language, fragment of OWL2 DL



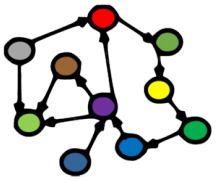

RDF/XML document
GRDDL
OWL/XML document
functional syntax document
Manchester syntax document
Turtle document
produce (optional)
parse (optional)
produce (required)
parse (required)
produce (optional)
parse (optional)
produce (optional)
parse (optional)
produce (optional)
parse (optional)
import
OWL 2 Ontology
mapping
mapping
Ontology Structure
RDF Graph
Syntax layer
Semantics layer
OWL2 DL
OWL2 Full
Direct Semantics
correspondence theorem (for DL subset)
RDF-Based Semantics

Summary of OWL terms

- owl:Ontology owl:Class owl:DatatypeProperty owl:ObjectProperty owl:NamedIndividual
- owl:Thing owl:Nothing owl:topObjectProperty owl:bottomObjectProperty owl:topDataProperty owl:bottomDataProperty
- owl:inverseOf owl:FunctionalProperty owl:InverseFunctionalProperty owl:TransitiveProperty owl:ReflexiveProperty owl:IrreflexiveProperty owl:SymmetricProperty owl:AsymmetricProperty owl:propertyChainAxiom
- owl:equivalentClass owl:disjointWith owl:equivalentProperty owl:propertyDisjointWith owl:sameAs owl:differentFrom owl:AllDifferent owl:AllDisjointClasses owl:AllDisjointProperties owl:members owl:distinctMembers owl:disjointUnionOf owl:NegativePropertyAssertion owl:assertionProperty owl:sourceIndividual owl:targetIndividual owl:targetValue
- owl:complementOf owl:intersectionOf owl:unionOf owl:oneOf owl:datatypeComplementOf owl:onDatatype owl:withRestrictions
- owl:Restriction owl:onProperty owl:onProperties owl:allValuesFrom owl:someValuesFrom owl:onDataRange owl:hasValue owl:hasSelf owl:cardinality owl:qualifiedCardinality owl:minCardinality owl:maxCardinality owl:onClass owl:minQualifiedCardinality owl:maxQualifiedCardinality
- owl:hasKey
- owl:annotatedProperty owl:annotatedSource owl:annotatedTarget owl:Annotation owl:AnnotationProperty owl:Axiom owl:imports owl:versionInfo owl:versionIRI owl:priorVersion owl:backwardCompatibleWith owl:OntologyProperty owl:incompatibleWith owl:deprecated owl:DeprecatedClass owl:DeprecatedProperty
- deprecated: owl:DataRange

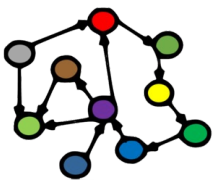


OWL DL



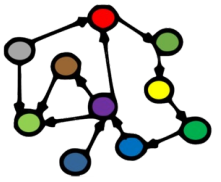
Relation to OWL

- OWL DL and description logic are closely matched
  - everything in OWL DL has a DL-counterpart
  - most everything in DL can be expressed in OWL DL
- DL is a family of logic systems:
  - some of them correspond to particular OWL profiles
  - OWL1 DL: $\mathcal{SHOIN}^{(D)}$
  - OWL2 DL: $\mathcal{SROIQ}^{(D)}$



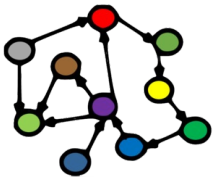
OWL profiles revisited

- OWL "1" (2002):
  - *OWL Full – "anything goes"*
  - OWL DL – fragment of OWL Full,
    - formal semantics through *description logic*
  - OWL Lite – simple fragment of OWL DL, not much used
- OWL 2 (2008):
  - *OWL2 Full – "anything goes"*
  - OWL2 DL – fragment of OWL2 full, extension of OWL DL
    - OWL2 EL – quick reasoning, fragment of OWL2 DL
    - OWL2 RL – rule language, fragment of OWL2 DL
      - OWL LD – linked data, fragment of OWL2 RL
    - OWL2 QL – query language, fragment of OWL2 DL

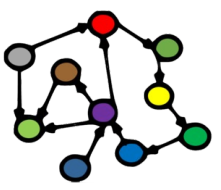


And there is more...

- A few other constructions
- Formal definitions of
  - syntax (rules for valid expressions, reasoning)
  - semantics (rules for interpreting expressions)
- Tools and techniques
- Lots of applications

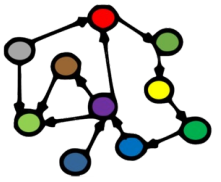


Manchester OWL syntax



Manchester OWL-syntax

- A simple DL notation without special symbols
  - used by Protege-OWL to construct classes
  - similar to DL syntax
- ```
  Class: Woman
         EquivalentTo: Human and Female
  ```
- ```
  Class: Man
         EquivalentTo: Human and not Female
  ```
- ```
  Class: Parent
         EquivalentTo: Mother or Father
  ```
- Can be used to *serialise* complete ontologies
  - ...we will look mostly at TBox expressions
- http://www.w3.org/TR/owl2-manchester-syntax/




  ```

Comparison

- DL:

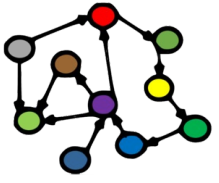
  Male ≐ Human ⊓ ¬Female

- Machester OWL:

```
Class: Man
    EquivalentTo: Human and not Female
```

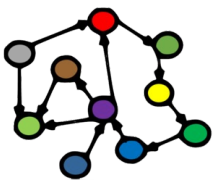
- TURTLE:

```
family:Man owl:equivalentClass
        owl:intersectionOf (
                family:Human
                [   a owl:Class ;
                    owl:complementOf family:Woman
                ]
        ) .
```



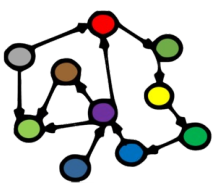
Roles in Manchester OWL syntax

- **Class: Mother**
  EquivalentTo:
  Female and hasChild some owl:Thing
- **Class: Bachelor**
  EquivalentTo:
  Male and not hasSpouse some owl:Thing
- **Class: Uncle**
  EquivalentTo:
  Male and hasSibling some Parent
  - universal concept (top): **owl:Thing**
  - existential restriction: **some**



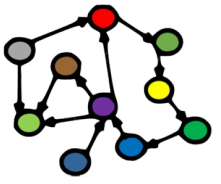
Null concept in Manchester OWL syntax

- **Class:** <class-name>
  EquivalentTo: Male **and** Female
  SubClassOf: **owl:Nothing**
  - null concept (bottom): **owl:Nothing**
  - subsumption (subconcept): **SubClassOf:**
  - equivalence: **EquivalentTo:**
    - ...used both for *definitions* and for *axioms*



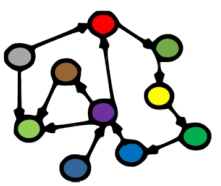
More roles in Manchester OWL syntax

- **Class: HappyFather**
  EquivalentTo:
  Father and hasChild only Happy
  - value restriction: **only**
- **Class: MotherOfOne**
  EquivalentTo: Mother and
  hasChild exactly 1
- **Class: Bigamist**
  EquivalentTo: hasSpouse min 2
  - number restriction: **exactly**, **min**, **max**
- **Class: Narcissist**
  EquivalentTo: loves some Self



Inverse, symmetric and transitive roles

- **Class:** Child
  EquivalentTo:
  Human **and inverse** hasChild **some** owl:Thing
- **Class:** hasParent
  EquivalentTo: inverse hasChild
- **ObjectProperty:** hasSibling
  Characteristics: Symmetric
- **ObjectProperty:** hasAncestor
  Characteristics: Transitive
- inverse role: **inverse**
  - symmetric role:
    Characteristics: SymmetricProperty
  - transitive role:
    Characteristics: TransitiveProperty



After Easter:
Graph Embeddings