

**Welcome to INF0216:
Knowledge Graphs
Spring 2022**

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

Session 8: Ontologies (OWL)

- Themes:
 - what and why?
 - basic OWL constructs (“RDFS-Plus”):
 - more precise properties
 - sameness and difference
 - complex classes (→ more later)
 - Programming in RDFLib



Readings

- Sources:
 - Allemang, Hendler, Gandon (2020):
Semantic Web for the Working Ontologist, 3rd edition
chapter 9-10 (“RDFS Plus”, but chapters 8-9 in the 2nd ed.)
 - 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 Primer, sections 2-6:
<http://www.w3.org/TR/owl-primer/>
 - **show**: Turtle and Manchester syntax
 - VOWL: Visual Notation for OWL Ontologies

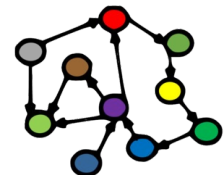


THE KNOWLEDGE GRAPH
COOKBOOK
RECIPES THAT WORK



ANDREAS BLUMAUER
AND HELMUT NAGY

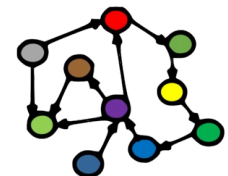
1st edition, 2020



Web Ontology Language (OWL)

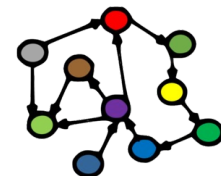
Why do we need vocabularies?

- Shared, well-defined terms (dereferencable URIs) for types, properties and some individuals that can be used to represent a domain
- Domains can be:
 - people, their friends and workplaces (FOAF, BIO)
 - electronic and other documents (DC, BIBO)
 - commerce (schema.org)
 - classification in libraries etc. (SKOS)
 - general encyclopedic information (DBpedia, Wikidata)
 - general time and place (OWL-Time, geo)
 - ...and *lots* of others



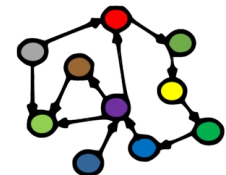
Why do we need vocabularies?

- To make knowledge graphs more precisely defined
- To make semantic data sets easier to use
 - encourage reuse
 - avoid misunderstandings and errors
 - easier to understand, recombine, enrich...
- To support computer processing
 - more powerful
 - more general



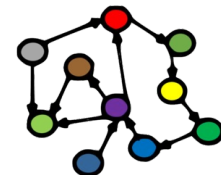
RDFS is a useful starting point...

- We can say:
 - “a pediatrician is a physician”
 - “Mary is a pediatrician” → “Mary is a physician”
 - “a physician is a health professional”
 - “a pediatrician is a health professional”
 - “having a patient” → “the subject is a health professional”
 - “treating a patient” → “the object is a person with health issues”
 - “treating a patient is a way of having a patient”
- if so:
 - “treating a patient” → “having a patient”
- *RDFS expresses this but not (so much) more...*



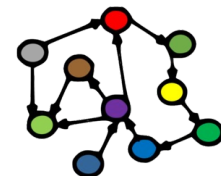
RDFS is a useful starting point...

- But 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 activePlayers, a VolleyballTeam 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!*



RDFS is a useful starting point...

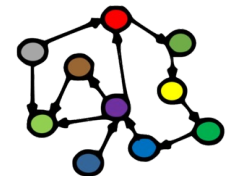
- But lots of simple stuff it cannot express, e.g.:
 - “every ancestor of an ancestor is an ancestor too”
 - “the BirthNumber of a Person is unique”
 - “a Republic has exactly one President”
 - “a FootballTeam has 11 activePlayers, a VolleyballTeam 6”
 - “a StringQuartet has two violins but only one 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”
- From vocabularies *to ontologies*

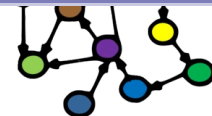
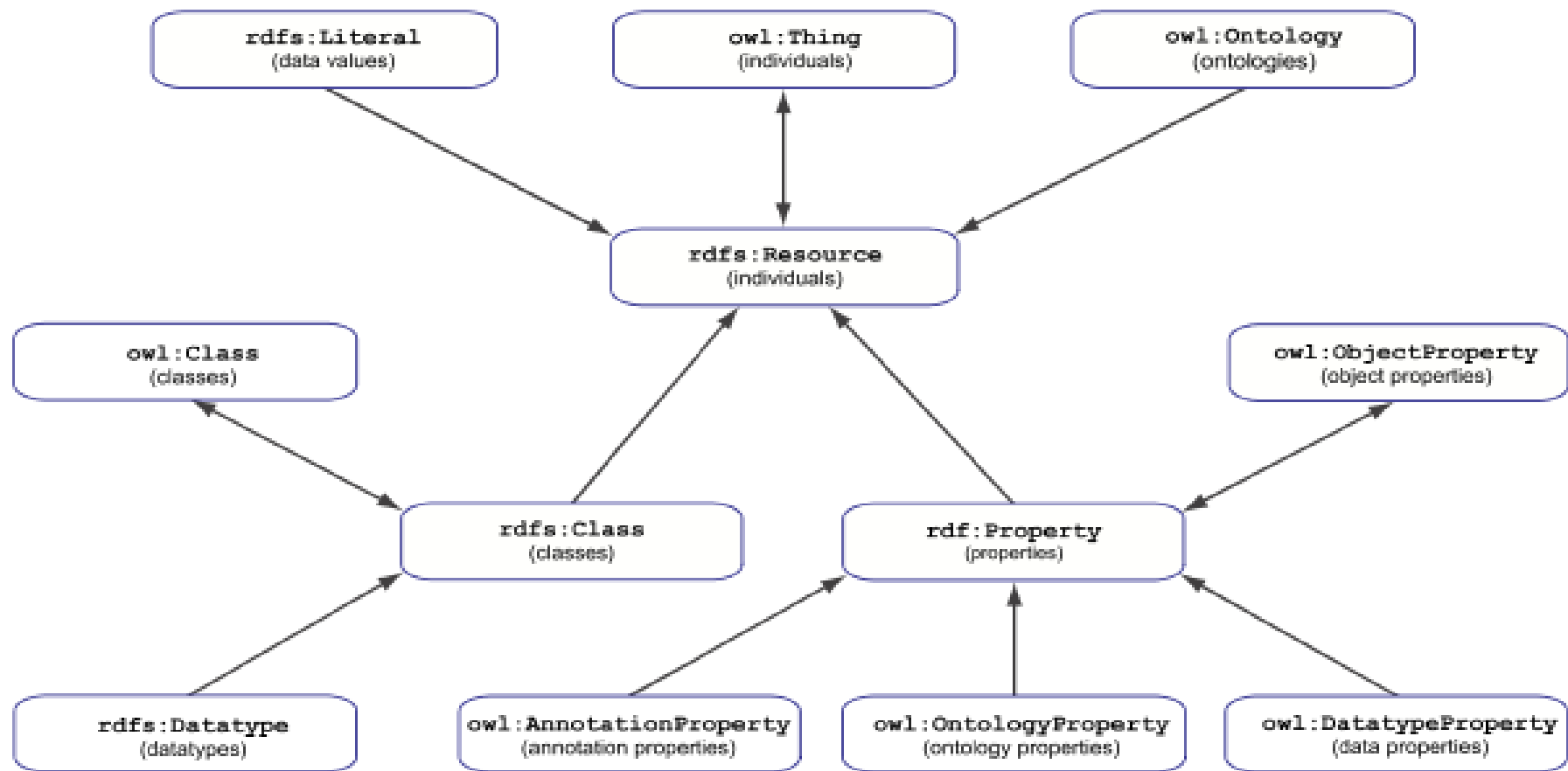


Basic idea

- Web Ontology Language (OWL):
 - builds on RDF and RDFS
 - uses classes and properties from RDF and RDFS
 - adds precision and formality
- Basic OWL-concepts:
 - `owl:Thing` owl:sameAs rdfs:Resource .
 - `owl:Class` owl:sameAs rdfs:Class .
 - *“owl:Property”* rdfs:subClassOf rdf:Property .
 - *“owl:Individual”* rdfs:subClassOf rdfs:Resource .

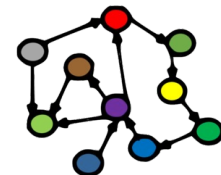
good practice: keep these three *disjoint*, i.e., no resource has more than one of them as *rdf:type*





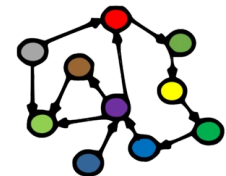
What does OWL offer?

- 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)
 - (more on this later)



Reuses or specialises RDFS

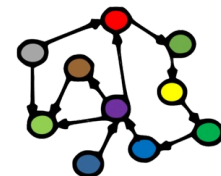
- *Reused* in OWL:
 - `rdf:type`, `rdf:Property`, `rdfs:subClassOf`, `rdfs:subPropertyOf`, `rdfs:domain`, `rdfs:range`
 - ...and lots of other stuff...
- *Renamed* by OWL:
 - `owl:Thing`, `owl:Class`, `owl:ObjectProperty`
- *Specialised* by OWL:
 - everything else in OWL *specialises* something in RDF / RDFS
 - but also introduces its own, and more powerful, formal underpinning



Basic OWL ("RDFS Plus")

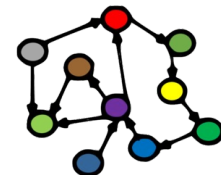
Inverse properties

- Properties can be each other's reverses (with subject and object swapped), e.g.,
 - `rex:HaakonMagnus fam:hasParent rex:Harald .`
 - `rex:Harald fam:hasChild rex:HaakonMagnus .`
- `P1 owl:inverseOf P2`:
 - `fam:hasParent owl:inverseOf fam:hasChild .`
 - `owl:inverseOf owl:inverseOf owl:inverseOf .`
 - `owl:inverseOf` a `owl:ObjectProperty` .
- “Entailment rules”:
 - if *`P1 owl:inverseOf P2`* then
 - *`P2 owl:inverseOf P1 .`*
 - if *`S P1 O . P1 owl:inverseOf P2`* then
 - *`O P2 S .`*



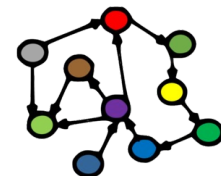
Symmetric properties

- Some properties are their own inverse, e.g.,
 - `rex:Harald fam:marriedTo rex:Sonja` .
 - `rex:Sonja fam:marriedTo rex:Harald` .
- `P rdf:type owl:SymmetricProperty`:
 - `fam:marriedTo a owl:SymmetricProperty` .
 - `owl:inverseOf a owl:SymmetricProperty` .
 - `owl:SymmetricProperty rdfs:subClassOf owl:ObjectProperty` .
- Entailment rules:
 - if *`P a owl:SymmetricProperty`* then
 - `P owl:inverseOf P` .
 - if *`S P O . P a owl:SymmetricProperty`* then
 - `O P S` .



Asymmetric, reflexive, irreflexive properties

- New in OWL2:
 - both *reflexive* and *irreflexive* properties:
 - owl:sameAs a owl:ReflexiveProperty .
 - “every resource is owl:sameAs itself”
 - fam:hasChild a owl:IrreflexiveProperty .
 - “no resource can be fam:hasChild of itself”
 - *many properties are neither!*
 - both *symmetric* and *asymmetric* properties:
 - fam:marriedTo a owl:SymmetricProperty .
 - “fam:marriedTo is always mutual (two-way)”
 - fam:hasChild a owl:AsymmetricProperty .
 - “no resources can be fam:hasChild of each other”
 - *many properties are neither!*



Transitive properties

- Some properties can form chains so that the result is the property itself, e.g.:
 - `rex:HaakonMagnus fam:hasAncestor rex:Harald .`
 - `rex:Harald fam:hasAncestor rex:Olav .`
 - `rex:HaakonMagnus fam:hasAncestor rex:Olav .`
- `P a owl:TransitiveProperty`:
 - `fam:hasAncestor a owl:TransitiveProperty .`
 - `rdfs:subClassOf a owl:TransitiveProperty .`
 - `rdfs:subPropertyOf a owl:TransitiveProperty .`
- Entailment rules:
 - “if *`S P X . X P O . P a owl:TransitiveProperty`* then
 - *`S P O .`*”



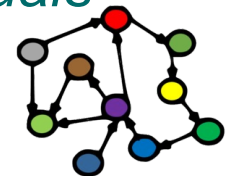
Functional properties

- Each subject *can only have one* object value for the functional property, e.g.,
 - fam:mother a owl:FunctionalProperty .
 - fam:birthdate a owl:FunctionalProperty .
 - owl:FunctionalProperty rdfs:subClassOf “owl:Property” .
- “Entailment rule”:
 - if $S P O1 . S P O2 . P$ a owl:FunctionalProperty then
 - $O1$ owl:sameAs $O2$.
 - This rule is for owl:ObjectProperties
 - There is a corresponding rule for owl:DatatypeProperties
 - but if two different literals become asserted as owl:sameAs one another, the ontology is inconsistent



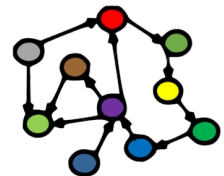
Inverse functional properties

- Two different subjects cannot have the same object for an inverse functional property, i.e.,
 - `fam:persNum` a `owl:InverseFunctionalObjectProperty` .
 - `fam:persNum` a `owl:FunctionalProperty` .
- Inverse functional properties are *unique* for each individual
 - used for *identifiers* in OWL 1
 - OWL 2 has a built-in *owl:hasKey* property for identifiers:
 - similar to inverse functional object properties
 - can only be used with OWL 2's *owl:NamedIndividuals*
 - ...not for anonymous “*owl:Individuals*”



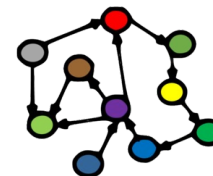
Summary: more precise properties

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



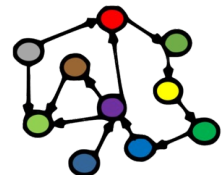
Individual equivalence

- Two individuals (with different URI-s) may represent the same thing:
 - http://dbpedia.org/resource/Amanda_Plummer
 - http://yago-knowledge.org/resource/Amanda_Plummer
 - <http://data.linkedmdb.org/resource/actor/34880>
- I1 owl:sameAs I2:
 - owl:sameAs a owl:ReflexiveProperty .
 - owl:sameAs a owl:SymmetricProperty .
 - owl:sameAs a owl:TransitiveProperty .
- owl:sameAs is an *equivalence relation*:
 - because it is *reflexive*, *symmetric* and *transitive*



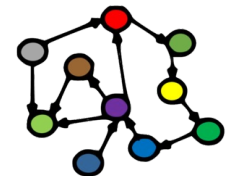
Unique Name Assumption (UNA)

- If two resources have different names, do they necessarily represent different things?
- RDF and OWL does *not* assume this!
 - *in RDF and OWL, we do not know whether resources with different names represent different things or not*
- We can use
 - `owl:sameAs` – two resources represent the same thing!
 - `owl:differentFrom` – they represent different things!
- Some ICT-languages and technologies use UNA, others do not!



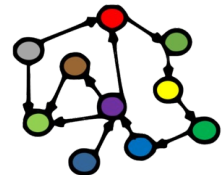
Individual difference

- A *pair* of individuals with different names (URI-s) must represent different things, e.g.,
 - `cal:Spring owl:differentFrom cal:Summer` .
- ...is *not* transitive



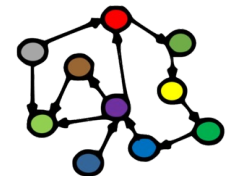
Individual difference

- A *pair* of individuals with different names (URI-s) must represent different things, e.g.,
 - `cal:Spring owl:differentFrom cal:Summer` .
- A *group* of individuals with different names (URI-s) must represent different things, e.g.,
 - [`a owl:AllDifferent`] `owl:distinctMembers (cal:Spring cal:Summer cal:Autumn cal:Winter)` .



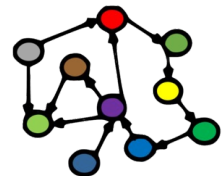
Individual difference

- A *pair* of individuals with different names (URI-s) must represent different things, e.g.,
 - `cal:Spring owl:differentFrom cal:Summer` .
- A *group* of individuals with different names (URI-s) must represent different things, e.g.,
 - `[a owl:AllDifferent] owl:distinctMembers (cal:Spring cal:Summer cal:Autumn cal:Winter)` .
 - *owl:AllDifferent* and *owl:distinctMembers* are special constructs in OWL
 - they must always be used together
 - ...corresponds to pairwise *owl:differentFrom* between *all* individuals in the *owl:distinctMembers*-list



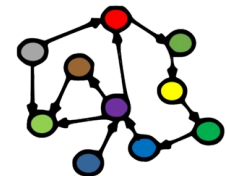
Equivalent classes

- Two classes (with different URI-s) represent the same class:
- C1 owl:equivalentClass C2:
 - owl:equivalentClass a owl:ReflexiveProperty .
 - owl:equivalentClass a owl:SymmetricProperty .
 - owl:equivalentClass a owl:TransitiveProperty .
- owl:equivalentClass is another *equivalence relation*:
 - it is *reflexive*, *symmetric* and *transitive*
- means the same as
 - C1 rdfs:subClassOf C2 and C2 rdfs:subClassOf C1



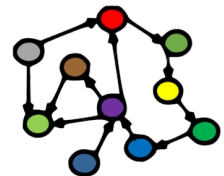
Disjoint classes

- Some classes cannot have the same individual as a member,
 - fam:Male owl:disjointWith fam:Female .
 - owl:disjointWith a owl:SymmetricProperty .
 - ...but it is *not* transitive
- I.e., no individual can have both classes as its rdf:type
 - ...corresponds to owl:differentFrom between *all* pairs of individuals in fam:Male and fam:Female
- Preferred in *formal* versions of OWL (no “punning”):
 - owl:Class owl:disjointWith “owl:Property” .
 - owl:Class owl:disjointWith “owl:Individual” .
 - “owl:Property” owl:disjointWith owl:Individual .



Equivalent properties

- Two properties (with different URI-s) represent the same property:
- P1 owl:equivalentProperty P2:
 - owl:equivalentProperty a owl:ReflexiveProperty .
 - owl:equivalentProperty a owl:SymmetricProperty .
 - owl:equivalentProperty a owl:TransitiveProperty .
- owl:equivalentProperty is another *equivalence relation*:
 - it is *reflexive*, *symmetric* and *transitive*
- Also *disjoint properties*:
 - :hasParent owl:propertyDisjointWith :hasSpouse .



Summary: sameness and difference

- 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`

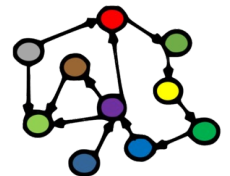


Basic OWL reasoning in Python and rdflib

RDFS inference in RDFLib

- `import owlrl.RDFSClosure`

```
rdfs = owlrl.RDFSClosure
      .RDFS_Semantics(g, False, False, False)
rdfs.closure()
rdfs.flush_stored_triples()
```



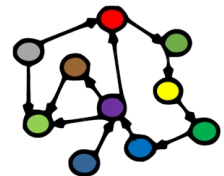
Basic OWL inference in RDFLib

- `import owlrl.RDFSClosure`

```
rdfs = owlrl.RDFSClosure
    .RDFS_Semantics(g, False, False, False)
rdfs.closure()
rdfs.flush_stored_triples()
```

- `import owlrl.CombinedClosure`

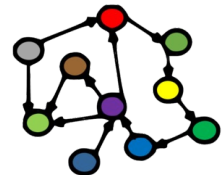
```
owl = owlrl.CombinedClosure
    .RDFS_OWLRL_Semantics(g, False, False, False)
owl.closure()
owl.flush_stored_triples()
```



Complex OWL classes
(most likely for later!)

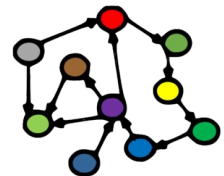
Union classes

- 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 .



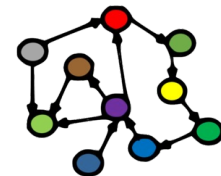
Intersection classes

- 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 .`



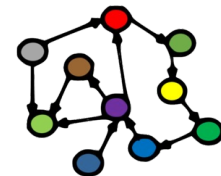
Complement classes

- 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?!*



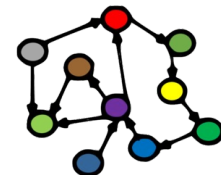
Complement classes

- 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*

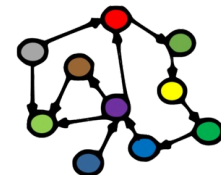


Complement classes

- 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  
  ]  
).
```



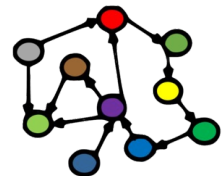
Complement classes

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



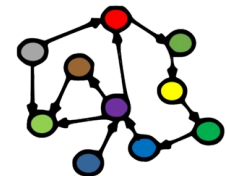
Closed World Assumption (CWA)

- 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)



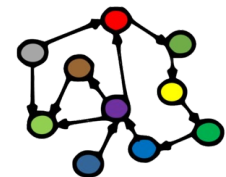
Enumeration classes

- 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



Other ways to write complex classes

- Why can also write:

```
cal:Season
```

```
  owl:oneOf ( cal:Spring ... cal:Winter ) .
```

or

```
cal:Season owl:equivalentClass [
```

```
  owl:oneOf ( cal:Spring ... cal:Winter ) ] .
```

- or (a weaker claim):

```
cal:Season owl:subClassOf [
```

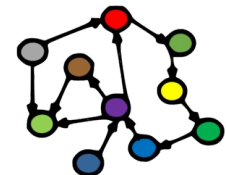
```
  owl:oneOf ( cal:Spring ... cal:Winter ) ] .
```

- Reason:

- sometimes we just need *rdfs:subClassOf*

- and it can be computationally more efficient

- *owl:equivalentClass* entails two-way *rdfs:subClassOf*



Summary: complex classes

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



**Next week:
Vocabularies**