Chapter 4 (2)
OWL

Based on slides from Grigoris Antoniou and Frank van Harmelen

Outline

1. A bit of history
2. Basic Ideas of OWL
3. **The OWL Language**
4. Examples
5. Beyond OWL

OWL Syntactic Varieties

- OWL builds on RDF and uses RDF’s serializations
- Other syntactic forms for OWL have also been defined:
  - Alternative, more readable serializations, e.g., Manchester syntax
  - These are often used in ontology editing tools, like Protege

OWL XML/RDF Syntax: Header in Turtle

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/ XLMSchema#> .
```

- OWL documents are RDF documents
- and start with a typical declaration of namespaces
- W3C owl recommendation has the namespace http://www.w3.org/2002/07/owl#"

owl:Ontology

```
<> a owl:Ontology ;
  rdfs:comment "Example OWL ontology" ;
  owl:priorVersion <http://example.org/uni-ns-old> ;
  owl:imports <http://example.org/persons> ;
  rdfs:label "University Ontology" .
```

- **owl:imports,** a transitive property, indicates that the document commits to all of the terms as defined in its target
- **owl:priorVersion** points to an earlier version of this document

OWL Classes

```
:AssociateProfessor a owl:Class ;
   owl:disjointWith (:Professor :AssistantProfessor) .
```

- Classes are defined using **owl:Class**
  - **owl:Class** is a subclass of **rdfs:Class**
- Owl:Class is disjoint with datatypes (aka literals)
- Disjointness is defined using **owl:disjointWith**
  - Two disjoint classes are can share no instances

Another Example

:Man rdfs:subClassOf foaf:Person .

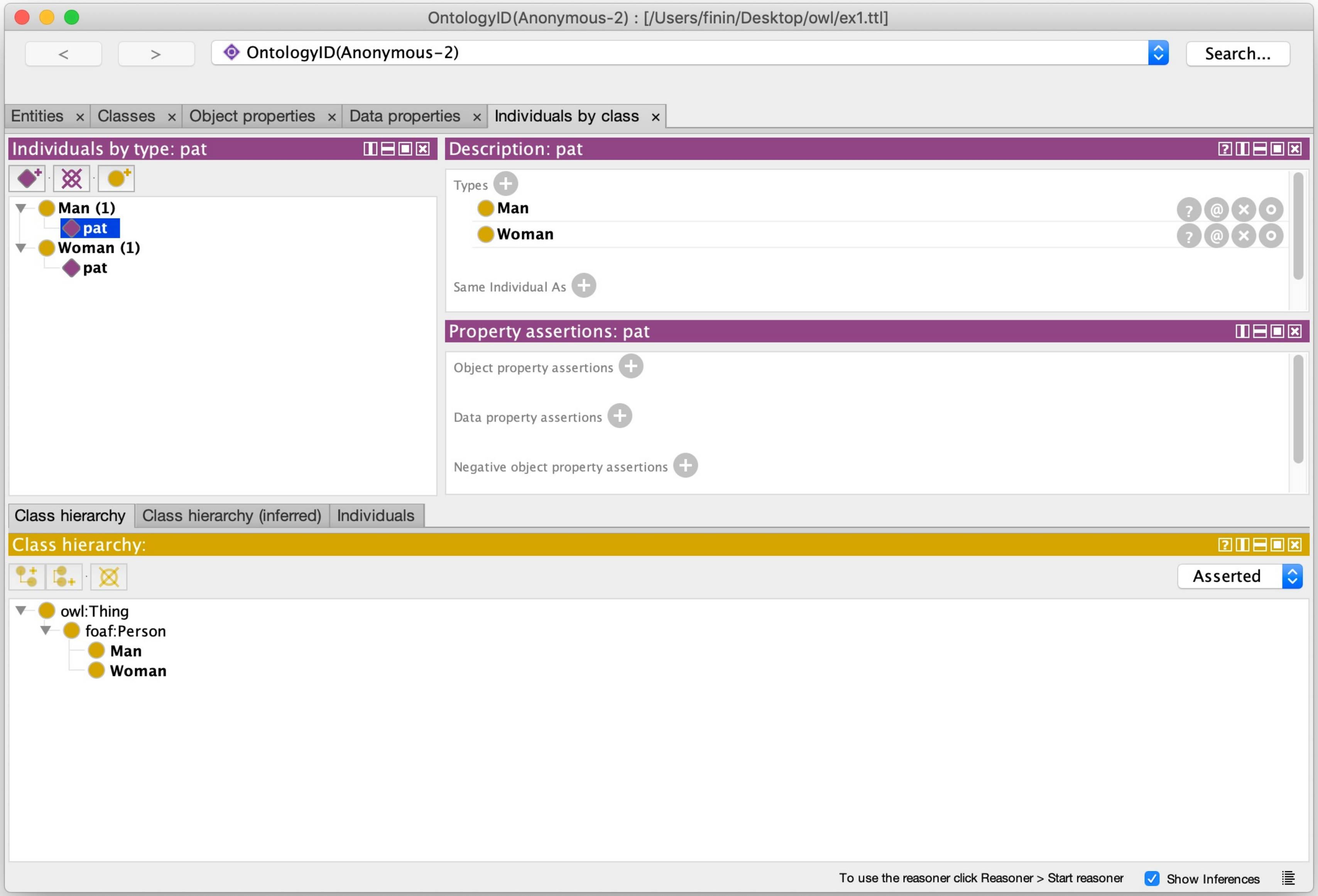
:Woman rdfs:subClassOf foaf:Person .

:Man owl:disjointWith :Woman .

Questions:

- Is :Man an rdfs:Class or a owl:Class?
- Why don’t we need to assert that :Man is some kind of class?
- Do we need to assert the disjointness both ways?
- What happens of we assert :pat a :Man; a :Woman?

Protégé



OntologyID(Anonymous-2) : [/Users/finin/Desktop/owl/ex1.ttl]

OntologyID(Anonymous-2)

Entities | Classes | Object properties | Data properties | Individuals by class

Individuals by type: pat

- Man (1)
  - pat
- Woman (1)
  - pat

Description: pat

Types
- Man
- Woman

Same Individual As

Property assertions: pat

Object property assertions

Data property assertions

Negative object property assertions

Class hierarchy | Class hierarchy (inferred) | Individuals

Class hierarchy:

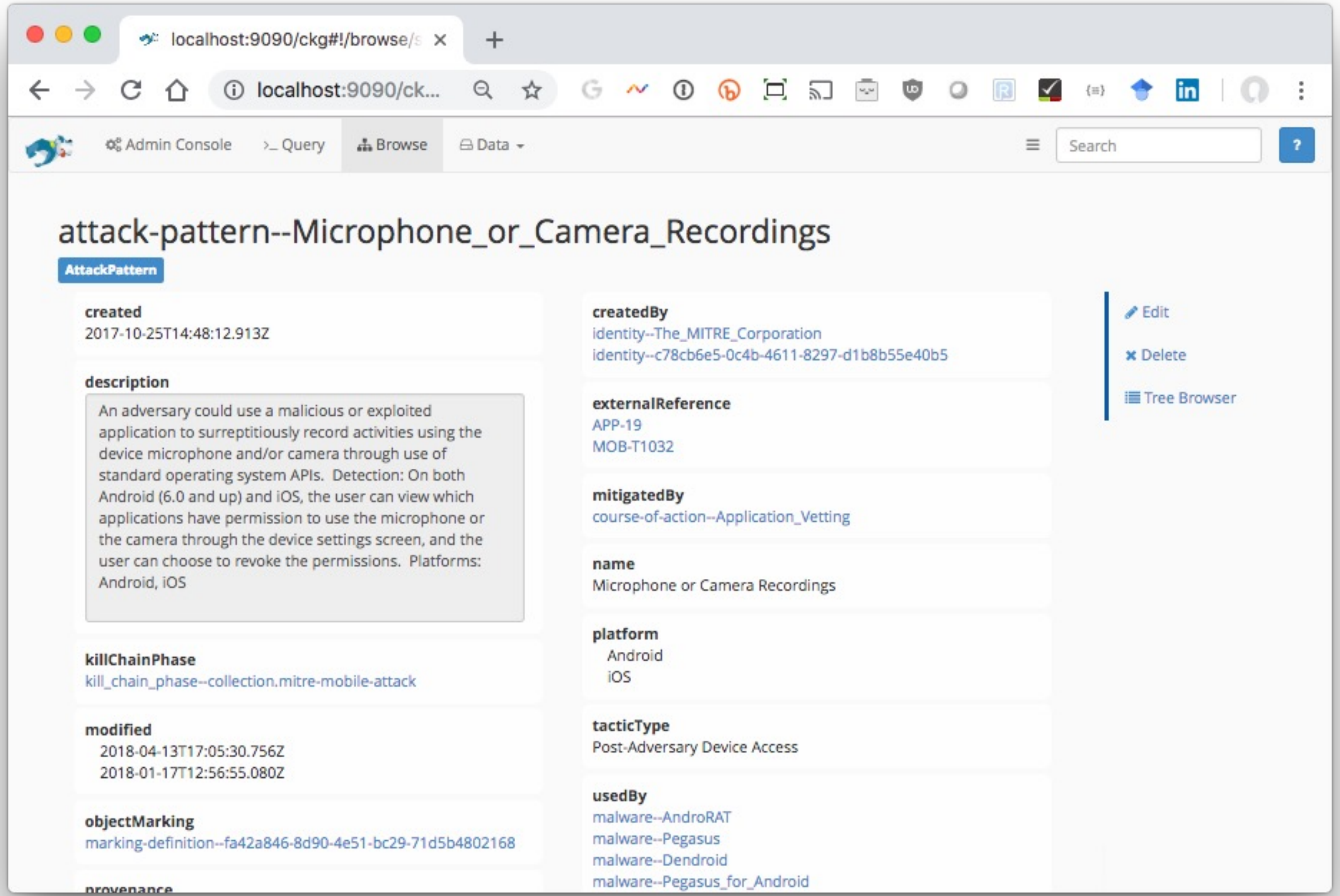
Asserted

- owl:Thing
  - foaf:Person
    - Man
    - Woman

To use the reasoner click Reasoner > Start reasoner

Show Inferences

StarDog



attack-pattern--Microphone_or_Camera_Recordings

AttackPattern

created
2017-10-25T14:48:12.913Z

description
An adversary could use a malicious or exploited application to surreptitiously record activities using the device microphone and/or camera through use of standard operating system APIs. Detection: On both Android (6.0 and up) and iOS, the user can view which applications have permission to use the microphone or the camera through the device settings screen, and the user can choose to revoke the permissions. Platforms: Android, iOS

killChainPhase
kill_chain_phase--collection.mitre-mobile-attack

modified
2018-04-13T17:05:30.756Z
2018-01-17T12:56:55.080Z

objectMarking
marking-definition--fa42a846-8d90-4e51-bc29-71d5b4802168

provenance

createdBy
identity--The_MITRE_Corporation
identity--c78cb6e5-0c4b-4611-8297-d1b8b55e40b5

externalReference
APP-19
MOB-T1032

mitigatedBy
course-of-action--Application_Vetting

name
Microphone or Camera Recordings

platform
Android
iOS

tacticType
Post-Adversary Device Access

usedBy
malware--AndroRAT
malware--Pegasus
malware--Dendroid
malware--Pegasus_for_Android

OWL Classes

```
:Faculty a owl:Class;
    owl:equivalentClass :AcademicStaffMember .
```

- **owl:equivalentClass** asserts two classes are equivalent
  - Each must have the same members
- **owl:Thing** is the most general class, which contains everything
  - i.e., every owl class is rdfs:subClassOf owl:Thing
- **owl:Nothing** is the empty class
  - i.e., owl:NoThing is rdfs:subClassOf every owl class

OWL Properties

- OWL has **two kinds** of properties
- **Object properties** relate objects to other objects
  - owl:ObjectProperty, e.g., isTaughtBy, employer
- **Data type properties** relate objects to Literals
  - owl:DatatypeProperty, e.g., age, phone, title…
- These were made separate to make it easier to implement **sound** and **complete** reasoners

Datatype Properties

- OWL uses XML Schema data types just like RDF nd RDFS, exploiting the layered architecture of the Semantic Web

```
:age a owl:DatatypeProperty;
  rdfs:domain foaf:Person;
  rdfs:range xsd:nonNegativeInteger .
```

OWL Object Properties

These connect to non-Literal objects, e.g.:

```
:isTaughtBy a owl:ObjectProperty;
    rdfs:domain :Course;
    rdfs:range :AcademicStaffMember;
    rdfs:subPropertyOf :involves .
```

Inverse Properties

```
:teaches a owl:ObjectProperty;
    rdfs:range :Course;
    rdfs:domain :AcademicStaffMember;
    owl:inverseOf :isTaughtBy .
```

Or just

```
:teaches owl:inverseOf :isTaughtBy .
```

A partial list of axioms:

```
owl:inverseOf rdfs:domain owl:ObjectProperty;
   rdfs:range owl:ObjectProperty;
   a owl:SymmetricProperty.
{?P  owl:inverseOf ?Q. ?S ?P ?O} => {?O ?Q ?S}.
{?P owl:inverseOf ?Q. ?P  rdfs:domain ?C} => {?Q rdfs:range ?C}.
{?A owl:inverseOf ?C. ?B owl:inverseOf ?C} => {?A rdfs:subPropertyOf ?B}.
```

Equivalent Properties

:lecturesIn owl:equivalentProperty :teaches .

- Two properties have the same *extension*
  - Intention vs. extension
  - Extension of a property is all of the subject-object pairs it holds between
- This could be defined by this axiom:

```
{ ?A rdfs:subPropertyOf ?B.
  ?B rdfs:subPropertyOf ?A.}
  <=> {?A owl:equivalentProperty ?B.}.
```

Property Restrictions

- Declare that class C satisfies certain conditions
  - All instances of C satisfy the conditions
- Equivalent to: C is subclass of a class C', where C' collects all objects that satisfy the conditions (C' can remain anonymous)
- Examples:
  - People whose age equals 20
  - Things with exactly two arms and two legs
  - People whose sex is male and have at least one child whose sex is female and whose age is six

Property Restrictions

- **owl:Restriction** element describes such a class
- Element has an **owl:onProperty** element and one or more **restriction declarations**
- One type defines **cardinality restrictions**

  A Parent must have at least one child

```
:Parent rdfs:subClassOf
      [a owl:Restriction;
        owl:onProperty :hasChild;
        owl:minCardinalityQ "1"] .
```

Property Restrictions

- This statement **defines** Parent as any Person who has at least one child

```
:Parent owl:equivalentClass
   owl:intersectionOf (:Person
      [a owl:Restriction;
         owl:onProperty :hasChild;
         owl:minCardinality "1” ])
```

- Note the Turtle syntax

  :C1 owl:intersectionOf **(:C2 :C3 :C4)** .

Property Restrictions

Other restriction types defines constraints on the kinds of values the property may take

- **owl:allValuesFrom** specifies universal quantification
- **owl:hasValue** specifies a specific value
- **owl:someValuesFrom** specifies existential quantification

owl:allValuesFrom

- Describe a class where all of the values of a property match some requirement
- E.g., Math courses taught by professors:

```
[a :mathCourse,
   [a owl:Restriction;
    owl:onProperty :isTaughtBy;
    owl:allValuesFrom :Professor] ].
```

Offspring of people are people

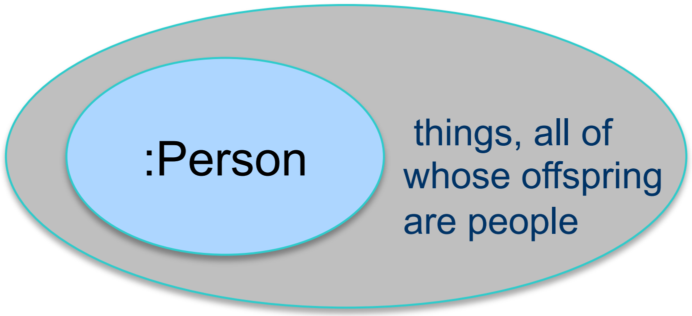
- This lets us solve the problem of an animal’s offspring must be the same type of animal
- i..e., people beget people, dogs beget dogs

```
:Person a owl:Class,
        rdfs:subClassOf
           [ a owl:Restriction;
             owl:onProperty bio:offspring;
             owl:allValuesFrom :Person] .
```

Offspring of people are people

```
:Person a owl:Class,
        rdfs:subClassOf
          [ a owl:Restriction;
           owl:onProperty   bio:offspring;
           owl:allValuesFrom :Person] .
```

“The class of things, all of whose offspring are people”



Offspring of people are people

```
:Person a owl:Class;
        rdfs:subClassOf
            [ a owl:Restriction;
              owl:allValuesFrom :Person;
              owl:onProperty bio:offspring ] .

:john a :Person; bio:offspring :mary
```

What follows?

```
:Person rdfs:subClassOf
   [owl:allValuesFrom :Person;
    owl:onProperty bio:offspring] .
???
:bio:offspring rdfs:domain :animal;
               rdfs:range :animal.
???
:alice a foaf:Person;
       bio:offspring :bob.
???
:carol a foaf:Person.
:don bio:offspring :carol.
???
```

“people give birth to people”

What follows?

```
:Person rdfs:subClassOf
   [owl:allValuesFrom :Person;
    owl:onProperty bio:offspring] .
```

nothing

```
:bio:offspring rdfs:domain :animal;
                 rdfs:range :animal.
```

:Person rdfs:subClassOf :bio:animal.

:alice a foaf:Person; bio:offspring :bob.

:alice a :animal. :bob a :animal, :person.

:carol a foaf:Person.

:don bio:offspring :carol.

:don a :animal.

"people give birth to people"

Maybe other things can give birth to people, too (e.g., the ancient Greek gods)

What follows?

```
:Person rdfs:subClassOf
    [owl:allValuesFrom :Person;
     owl:onProperty bio:offspringOf] .

bio:offspringOf rdfs:domain :animal;
                rdfs:range :animal;
                owl:inverse bio:offspring.

:carol a foaf:Person.
:don bio:offspring :carol.
???
```

“people are born of people”

What follows?

```
:Person rdfs:subClassOf
   [owl:allValuesFrom :Person;
    owl:onProperty bio:offsringOf] .

bio:offspringOf rdfs:domain :animal;
                rdfs:range :animal;
                owl:inverse bio:offspring.

:carol a foaf:Person.
:don bio:offspring :carol.
```

“people are born of people”

:don a foaf:Person

owl:hasValue

- Describe a class with a particular value for a property
- E.g., Math courses taught by Professor Longhair

```
# Math courses taught by :longhair
[ rdfs:subclassOf :mathCourse;
  [ a owl:restriction;
      owl:onProperty :isTaughtBy;
      owl:hasValue :longhair] .
```

Questions:

- Does this say all math courses are taught by :longhair?
- Does it say that there are some courses taught by :longhair?
- If X is a :mathCourse & :isTaughtBy :longhair is it in this set?
- Can all classes, however defined, be paraphrased by a noun phrase in English?

owl:hasValue

- Describe a class with a particular value for a property
- E.g., Math courses taught by Professor Longhair

```
# Math courses taught by :longhair
[ rdfs:subclassOf :mathCourse;
  [ a owl:restriction;
      owl:onProperty :isTaughtBy;
      owl:hasValue :longhair] .
```

Questions:

- Does this say all math courses are taught by :longhair? **No**
- Does it say that there are some courses taught by :longhair? **No**
- If X is a :mathCourse & :isTaughtBy :longhair is it in this set? **Yes**
- Can all classes, however defined, be paraphrased by a noun phrase in English? **probably**

owl:hasValue

- Describe a class with a particular value for a property
- E.g., Math courses taught by Professor Longhair

```
# Math courses taught by :longhair
[ rdfs:subclassOf :mathCourse;
  [ a owl:restriction;
      owl:onProperty :isTaughtBy;
      owl:hasValue :longhair] .
```

- OWL is based on Description Logic (DL)
- DL's notation was designed to be similar to the way we describe things in human languages

Questions:

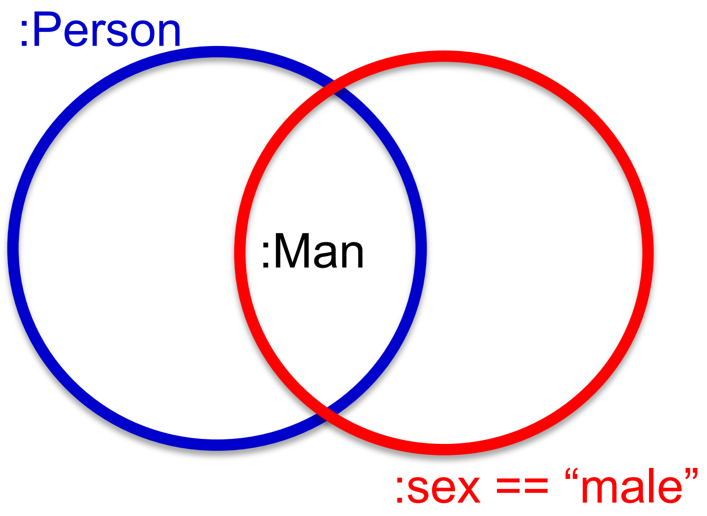
- Does this say all math courses are ta by :longhair? **No**
- Does it say that there are some co ses taught by :longhair? **No**
- If X is a :mathCourse & :isTaugh tBy :longhair is it in this set? **Yes**
- Can all classes, however defined, be paraphrased by a noun phrase in English? **probably**

A typical example

```
:Male owl:equivalentClass
  owl:intersectionOf
  (:Person,
    [a owl:Restriction;
      owl:onProperty :sex;
      owl:hasValue "male"] ).
```

A typical example

```
:Man owl:equivalentClass
  owl:intersectionOf
  (:Person,
   [a owl:Restriction;
     owl:onProperty :sex;
     owl:hasValue "male"] ).
```



Classes are sets in OWL

What follows?

```
:ed a :Man .

???

:frank a foaf:Person; :sex "male".

???

:pat a foaf:Person; :sex "male"; :sex "female" .

???
```

What follows?

:ed a :Man .

:ed a :Person; :sex “male”

:frank a foaf:Person; :sex "male".

:frank a :Man

:pat a foaf:Person; :sex "male"; :sex "female" .

:pat a :Man

We’ve not yet said that you can only have one :sex value

owl:someValuesFrom

- Describe class requiring it to have *at least one value* for a property matching a description
- E.g., Academic staff members who teach **an** undergraduate course

```
[ a :academicStaffMember;
   a [owl:onProperty :teaches;
      owl:someValuesFrom :undergraduateCourse] ]
```

Cardinality Restrictions

- A set's cardinality is its number of elements
- We can specify minimum and maximum number using **owl:minCardinality** & **owl:maxCardinality**
  - Courses with fewer than 10 students
  - Courses with between 10 and 100 students
  - Courses with more than 100 students
- Can specify an exact number by using the same minimum and maximum number
  - Courses with exactly seven students
- For convenience, OWL offers also **owl:cardinality**
  - E.g., exactly N

Cardinality Restrictions

E.g., This represents the courses taught by at least two instaructors:

```
[a owl:Restriction;
 owl:onProperty :isTaughtBy;
 owl:minCardinality
  “2”^^xsd:nonNegativeInteger] .
```

Since we’re using Turtle syntax, we can just say 2

What does this say?

```
:Parent owl:equivalentClass
   [a owl:Restriction;
     owl:onProperty :hasChild;
     owl:minCardinality 1] .
```

Questions:

- Must parents be humans?
- Must their children be humans?

What does this say?

```
:Parent owl:equivalentClass
   [a owl:Restriction;
     owl:onProperty :hasChild;
     owl:minCardinality 1] .
```

Questions:

- Must parents be humans? **No**
- Must their children be humans? **No**

Definition of a parent

The parent class is equivalent to the class of things that have at least one child

In logic, we might specify this as

All(x): Parent(x) ⇔ Exisits(y) hasChild(x, y)

If hasChild is defined as having Person as its domain, then Parents are also people.

Special Properties

- **owl:TransitiveProperty (**transitive property)
  - E.g. “has better grade than”, “is ancestor of”
- **owl:SymmetricProperty** (symmetry)
  - E.g. “has same grade as”, “is sibling of”
- **owl:FunctionalProperty** defines a property that has *at most* one value for each object
  - E.g. “age”, “height”, “directSupervisor”
- **owl:InverseFunctionalProperty** defines a property for which two different subjects cannot have the same value
  - e.g., “ssn”, “mobile phone number”

Defining Classes: Boolean Combinations

- We can combine classes using Boolean operations (union, intersection, complement)
- Negation is introduced by **complementOf**, *e.g.: courses taught by things that are not staffMembers*

```
[ a :course,
    owl:Restriction;
      owl:onProperty :taughtBy;
      owl:allValuesFrom [a owl:Class;
                                       owl:complementOf :staffMember]
 ] .
```

Boolean Combinations

- The new class is not a subclass of the union, but rather equal to the union
  - We have stated an equivalence of classes
- E.g., *university people is the union of staffMembers and Students*

```
:peopleAtUni
   owl:equivalentClass
      owl:unionOf (:staffMember :student) .
```

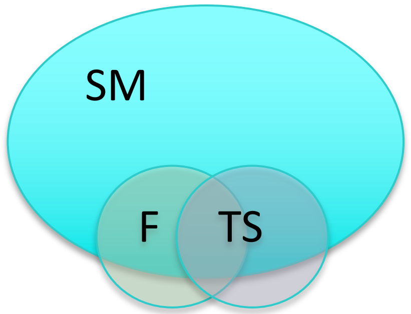
Boolean Combinations

E.g., CS faculty is the intersection of faculty and things that belongTo the CS Department.

```
:facultyInCS owl:equivalentClass
  owl:intersectionOf
     (:faculty
       [ a owl:Restriction;
         owl:onProperty :belongsTo;
         owl:hasValue :CSDepartment ]
     ) .
```

Nesting of Boolean Operators

E.g., administrative staff are staff members who are not faculty or technical staff members



```
:adminStaff owl:equivalentClass
  owl:intersectionOf
    (:staffMember
     [a owl:Class;
      owl:complementOf [a owl:Class;
                 owl:equivalentClass
                 owl:unionOf (:faculty :techSupportStaff)]])
```

Declaring Instances

Instances of OWL classes are declared as in RDF

```
:john
   a :academicStaffMember;
   uni:age 39 .

:Monday a owl:DayOfWeek .
:Tuesday a owl:DayOfWeek .
…
```

What if we want to say that DayOfWeek has a fixed number of instances?

Enumerations with owl:oneOf

- Sometimes we want to define a class that has a fixed set of known instances
- *E.g., a thing that's either Monday, Tuesday, …*

```
owl:DayOfWeek a owl:Class;
    owl:oneOf (:Monday :Tuesday :Wednesday
              :Thursday :Friday :Saturday :Sunday) ]
```

- Note the list syntax: a sequence of elements in parentheses with whitespace separators

No Unique-Names Assumption

- OWL does not adopt the unique name assumption used in many database systems
  - Two instances having different names or IDs doesn’t imply that they’re different individuals
- Suppose we state that each course is taught by at most one staff member, and that a given course is taught by #949318 and is taught by #949352
  - An OWL reasoner does not flag an error
  - Instead, it **infers** that the two resources are equal

Distinct Objects (1)

- An **owl:irreflexiveProperty** is one where the subject and object must differ
- Given:

  fam:hasChild a owl:IrreflexiveProperty

  :alice fam:hasChild :bob, :carol.
- Owl infers that :alice is a different individual that both :bob and :carol

Are :bob and :carol necessarily different?

Distinct Objects (2)

Assert that two instances are distinct like

:bob owl:differentFrom :carol .

OWL provides a shorthand notation to assert the pairwise inequality of all individuals in a given list

```
[a owl:allDifferent;
  owl:distinctMembers
       (:alice :bob :carol :don) ].
```

Inferring Distinctness

An ontology may provide **many** ways to infer that individuals as distinct from what's known about them, e.g., they

- Belong to sets known to be disjoint (e.g., :Man, :Woman)

  :pat1 a :Man. :pat2 a :Woman. :Man owl:disjointWith :Woman.

- Have inverse functional properties with different values

  :pat1 :mobile "4105618733 . :pat2 :mobile "2154729983" .

  :mobile a owl:InverseFunctionalProperty .

- Have different values for a functional property

  :pat1 :hasMother :p1 . :pat2 :hasMother :p2 .

  :hasMother a owl:FunctionalProperty . :p1 owl:differentFrom :p2

- Are connected with an irreflexive relation

  :pat1 :hasChild :pat2. :hasChild a owl:IrreflexiveProperty .

Inferring Distinctness

An ontology may provide **many** ways to infer that individ-uals as distinct from what's known about them, e.g., they

- Belong to sets known to be disjoint (e.g., :Man, :Woman)

  :pat1 a :Man. :pat2 a :Woman. :Man owl:disjointWith :Woman.

- **Have inverse functional properties with different values**

  :pat1 :mobile "4105618733 . :pat2 :mobile "2154729983" .

  :mobile a owl:InverseFunctionalProperty .

- Have different values for a fu   tional property

  :pat1 :hasMother :p1 . :pat2   ther :p2 .

  :hasMother a owl:Fun   2

- Are connected with a

  :pat1 :hasChild :pat2.

- An **owl:inverseFunctionalProperty** means that its object uniquely defines the subject.
- i.e., at most one subject can have it as the value of the property

Inferring Distinctness

An ontology may provide **many** ways to infer that individuals as distinct from what's known about them, e.g., they

- Belong to sets known

  :pat1 a :Man. :pat2 a :

- Have inverse function

  :pat1 :mobile "4105618733 . :p obile "2154729983" .

  :mobile a owl:InverseFunctiona perty .

- **Have different values for a functional property**

  :pat1 :hasMother :p1 . :pat2 :hasMother :p2 .

  :hasMother a owl:FunctionalProperty. :p1 owl:differentFrom :p2

- Are connected with an irreflexive relation

  :pat1 :hasChild :pat2. :hasChild a owl:IrreflexiveProperty .

- An **owl:functionalProperty** means that the subject can only have one value
- If two instances have different values, they must be different

Inferring Distinctness

An ontology may provi
uals as distinct from w

- Belong to sets known t

  :pat1 a :Man. :pat2 a :W

- Have inverse functiona

  :pat1 :ssn “249148660” . :pat2 962271” .

  :ssn a owl:InverseFunctionalPr

- Have different values for a f tional property

  :pat1 :ssn “249148660” . :pat2 sn “482962271” .

  :ssn a owl:FunctionalProperty .

- Are connected with an **irreflexive** relation

  :pat1 :hasChild :pat2. :hasChild a owl:IrreflexiveProperty .

- An **owl:reflexiveProperty** always holds between an object and itself, e.g.:

  owl:sameAge a owl:reflexiveProperty .

- An owl:irreflexiveProperty never holds between an object and itself, e.g.,

  owl:sibling a owl:irreflexiveProperty .

Data Types in OWL

- XML Schema provides a mechanism to construct user-defined data types
  - E.g., the data type of **adultAge** includes all integers greater than 18
- Such derived data types can’t be used in OWL
  - The OWL reference document lists all the XML Schema data types that can be used
  - These include the most frequently used types such as **string**, **integer**, **Boolean**, **time**, and **date**.

Combination of Features in OWL Profiles

- Different OWL profiles have different sets of restrictions regarding the application of features
- In **OWL Full**, all language constructors may be used in any combination if the result is legal RDF
- **OWL DL** removes or restricts some features to ensure that **complete reasoning** is possible and ***tractable*** or to make implementations easier
  - tractable computations can be done in polynomial time
- The **OWL RL** profile further restricts features to those that can be inferred by simple rules.
- Most practicle systems limit themselves to OWL RL

OWL Conclusions (1)

- OWL is a standard for Web ontologies that builds upon RDF and RDF Schema and provides inference and consistency checking
- Formal semantics and reasoning support is provided through the mapping of OWL to logic
  - Predicate logic and description logics have been used for this purpose
- While OWL is sufficiently rich to be used in practice and most practical systems use OWL DL or OWL RL subsets

OWL Conclusions (2)

- OWL is not used for many useful very large or inconsistent KGs like Wikidata
- These are often based on RDF and RDFS
- We'll cover important components that work on these and provide some alternatives to inference
  - SPARQL (SPARQL Protocol and RDF Query Language) is an efficient query language that can be used for inference and query answering
  - SHACL (SHApe Constraint Language) is a system that defines consistency constraints