Chapter 4
OWL

Based on slides from Grigoris Antoniou and Frank van Harmelen

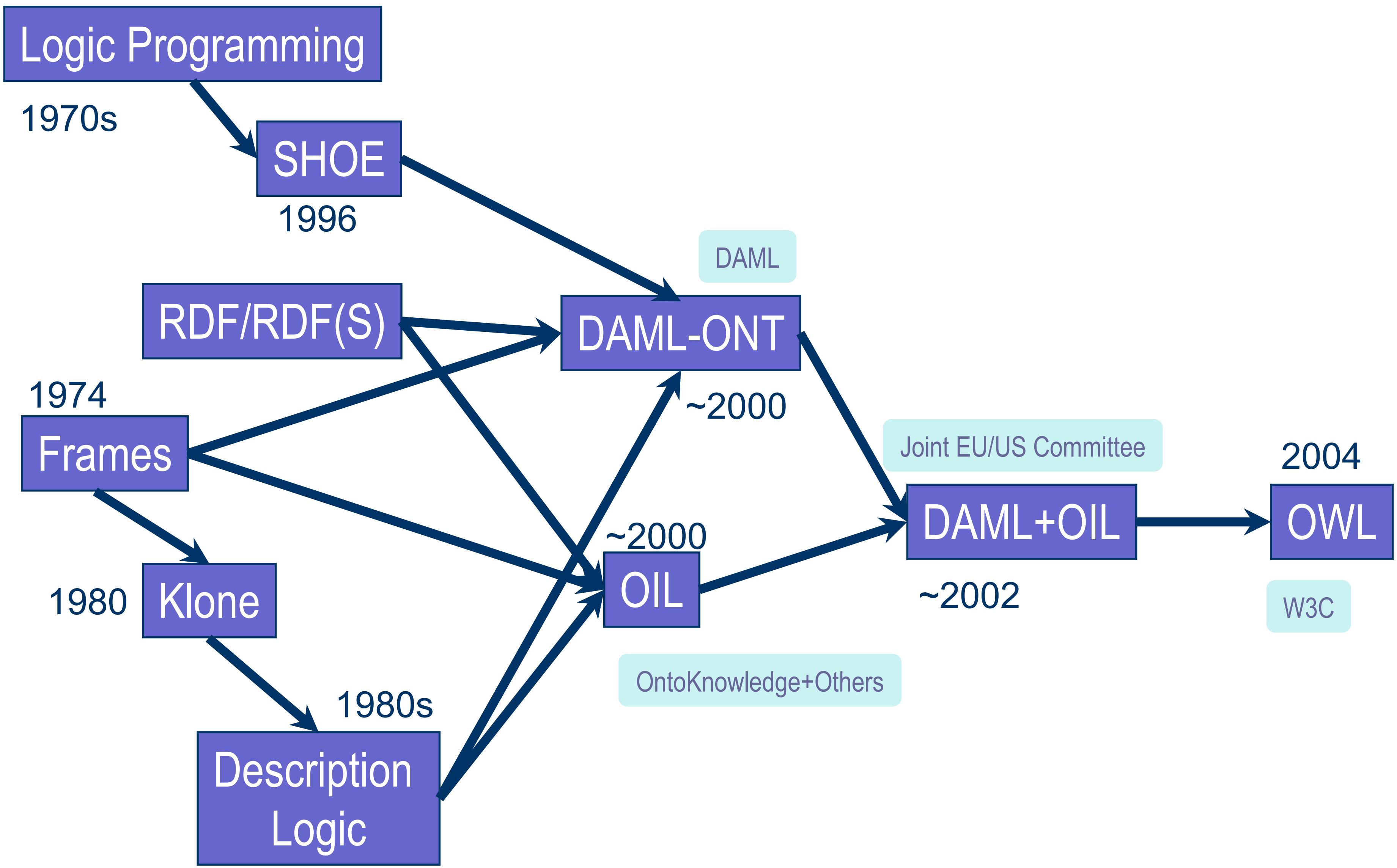
TL;DR: What is OWL

OWL uses the syntax of RDF but defines new classes and properties, making it more expressive as knowledge representation language

Outline

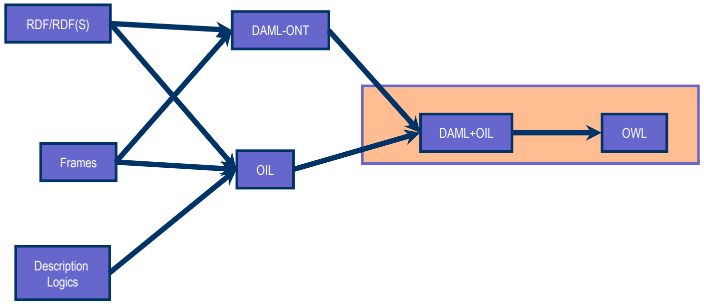
1. **A bit of history**
2. Basic Ideas of OWL
3. The OWL Language
4. Examples
5. The OWL Namespace
6. OWL 2

The OWL Family Tree



A Brief History of OWL: OWL

- W3C Recommendation (February 2004)
- Based on March 2001 DAML+OIL specification
- Well defined RDF/XML serializations
- Formal semantics
  - First order logic
  - Relationship with RDF
- Comprehensive test cases for tools/implementations
- Growing industrial take up



OWL 2

- An **extension of OWL**
  - Addressed deficiencies identified by users and developers (at **OWLED workshop**)
- Based on more expressive DL subset: **SROIQ**
- W3C **working group** chartered
  - http://www.w3.org/2007/OWL/wiki/OWL_Working_Group
  - Became W3C recommendation Oct. **2009**
- **Supported** by popular OWL tools
  - Protégé, TopBraid, FaCT++, Pellet, …

Outline

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

Ontology and Data

- Philosophy: Ontologies are models of what exists in the world (kinds of things, relations, events, properties, etc.)
  - Information systems: a schema for info. or data
  - KR languages: model of classes & relations/properties & associated axioms, e.g., subPropertyOf is transitive
- Data is information about individual instances expressed with terms in the ontology
  - Some instances might be considered part of the ontology (e.g., God, George Washington, Baltimore)

Requirements for Ontology Languages

- **Ontology languages let users write explicit, formal conceptualizations of domain models**
- Requirements:
  - well-defined syntax
  - efficient reasoning support
  - formal semantics
  - sufficient expressive power
  - convenience of expression

Expressive Power vs. Efficient Reasoning

- Always a tradeoff between expressive power and efficient reasoning support
- The richer the language, the more inefficient the reasoning support becomes (in general)
- Reasoning can be undecidable or semi-decidable and even if decidable can be exponentially hard
- We need a compromise between:
  - Language supported by reasonably efficient reasoners
  - Language that can express large classes of ontologies and knowledge

Kinds of Reasoning about Knowledge

- **Class membership**

  If x is an instance of a class C, and C is a subclass of D, then we can infer that x is an instance of D

- **Equivalence of classes**

  If class A is equivalent to class B, and class B is equivalent to class C, then A is equivalent to C, too

- **Consistency**
  - X is an instance of classes A and B, but A and B are disjoint
  - This is an indication of an error in the ontology or data

- **Classification**

  Certain property-value pairs are a sufficient condition for membership in a class A; if an individual x satisfies such conditions, we conclude that x must be an instance of A

Uses for Reasoning

- **Reasoning support is important for**
  - Deriving new relations and properties
  - Automatically classifying instances in classes
  - Checking consistency of ontology and knowledge
  - checking for unintended relationships between classes
- **Checks like these are valuable for**
  - designing large ontologies, where multiple authors are involved
  - integrating and sharing ontologies from various sources

Reasoning Support for OWL

- Semantics is a prerequisite for reasoning support
- Formal semantics and reasoning support usually provided by
  - mapping an ontology language to known logical formalism
  - using automated reasoners that already exist for those formalisms
- OWL is (partially) mapped to a *description logic*

  DLs are a subset of logic for which efficient reasoning support is possible

RDFS's Expressive Power Limitations

- **Local scope of properties**
  - **rdfs:range** defines range of a property (e.g., eats) for **all** instances of a class
  - In RDF Schema we can't declare range restrictions that apply to only some
  - E.g., animals eat living_things but cows only eat plants
  - :eat rdfs:domain :animal; range :living_thing
    :eat rdfs:domain :cow; range :plant

RDFS’s Expressive Power Limitations

- **Disjointness of classes**
  - Sometimes we wish to say that classes are disjoint (e.g. **male** and **female**)
- **Boolean combinations of classes**
  - We may want to define new classes by combining other classes using union, intersection, and complement
  - E.g., **person** equals union of **male** and **female** classes
  - E.g., weekdays equals set {:Monday, … :Sunday}

RDFS's Expressive Power Limitations

- **Cardinality restrictions**
  - E.g., a person has **exactly two** parents, a course is taught by **at least one** lecturer
- **Special characteristics of properties**
  - Transitive property (like "greater than")
  - Unique property (like "is mother of")
  - A property is the inverse of another property (like "eats" and "is eaten by"

Combining OWL with RDF Schema

- Ideally, OWL would extend RDF Schema
  - Consistent with the layered architecture of the Semantic Web
- **But** simply extending RDF Schema works against obtaining expressive power and efficient reasoning
  - Combining RDF Schema with logic leads to uncontrollable computational properties
- OWL uses RDF and most of RDFS

Three Species of OWL 1

- W3C'sWeb Ontology Working Group defined OWL as three different sublanguages:
  - OWL Full
  - OWL DL
  - OWL Lite
- Each sublanguage geared toward fulfilling different aspects of requirements

OWL Full

- It uses all the OWL languages primitives
- It allows the combination of these primitives in arbitrary ways with RDF and RDF Schema
- OWL Full is fully upward-compatible with RDF, both syntactically and semantically
- OWL Full is so powerful that its reasoning is undecidable
  - No complete reasoning support

Soundness and completeness

- A **sound** reasoner only makes conclusions that logically follow from the input, i.e., all of its conclusions are correct
  - We typically require our reasoners to be sound
- A **complete** reasoner can make all conclusions that logically follow from the input
  - We cannot guarantee complete reasoners for full FOL and many subsets
  - So, we can’t do it for OWL Full

OWL DL

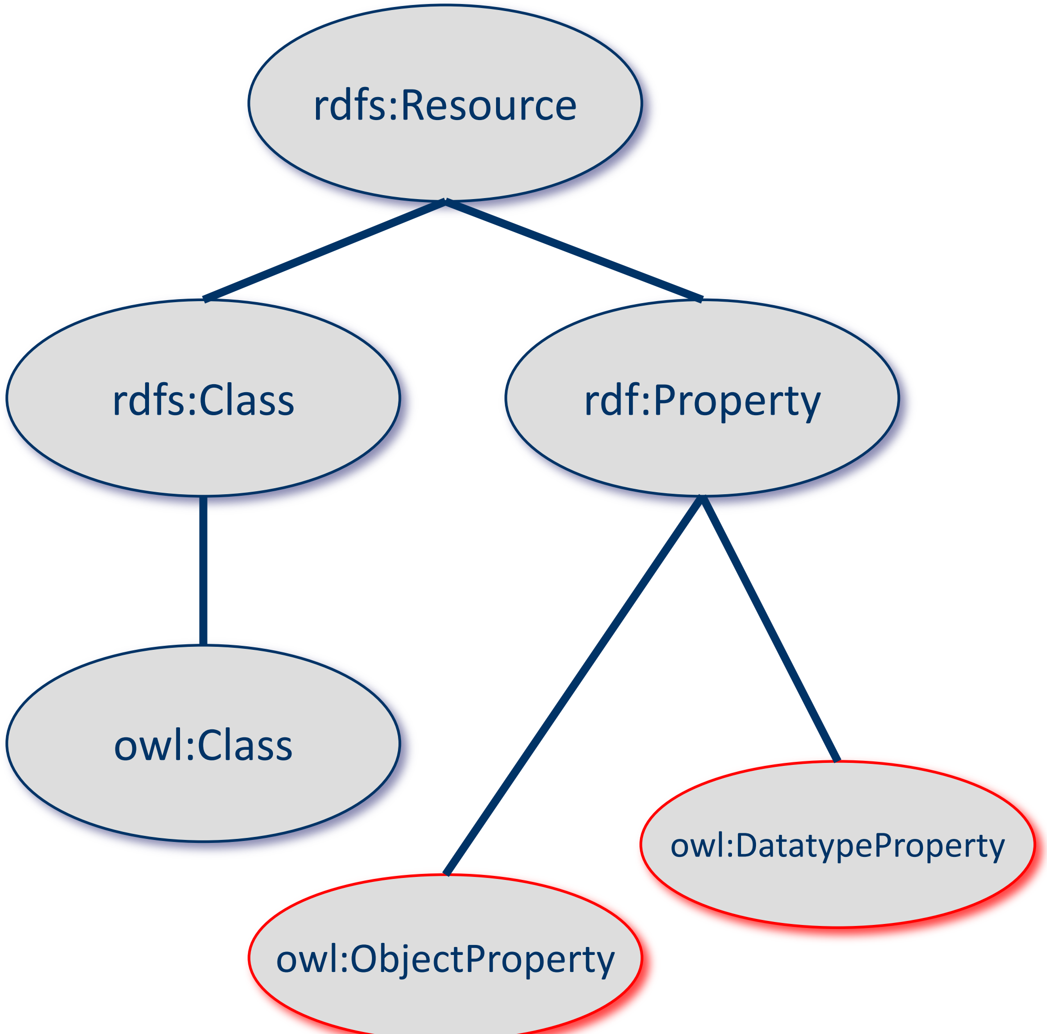
- OWL DL (Description Logic) is a sublanguage of OWL Full that restricts application of the constructors from OWL and RDF
  - Application of OWL's constructors to each other is disallowed
  - It corresponds to a well studied description logic
- OWL DL permits efficient reasoning support
- **But** we lose full compatibility with RDF
  - Not every RDF document is a legal OWL DL document
  - Every legal OWL DL document is a legal RDF document

OWL Lite

- An even further restriction limits OWL DL to a subset of the language constructors
  - E.g., OWL Lite excludes enumerated classes, disjointness statements, and arbitrary cardinality
- The advantage of this is a language that is easier to
  - grasp, for users
  - implement, for tool builders
- The disadvantage is restricted expressivity

OWL Compatibility with RDF Schema

- All varieties of OWL use RDF for their syntax
- Instances are declared as in RDF, using RDF descriptions
- OWL constructors are specialisations of theirRDF counterparts
- OWL classes and properties have additional constraints



Outline

1. A bit of history
2. Basic Ideas of OWL
3. **The OWL Language**
4. Examples
5. The OWL Namespace
6. Future Extensions

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

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 .

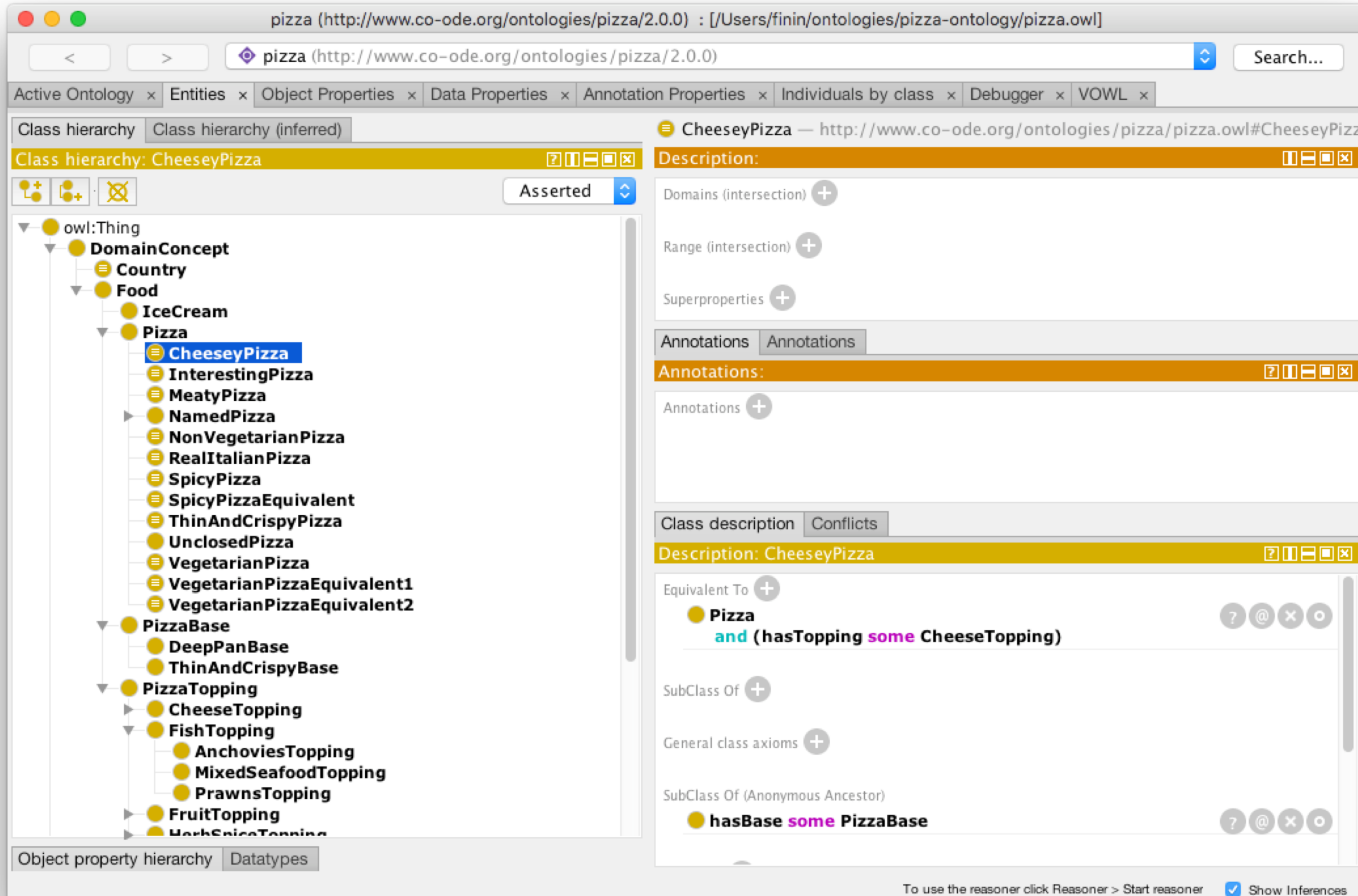
:Woman rdfs:subClassOf foaf:Person .

:Man owl:disjointWith :Woman .

Questions:

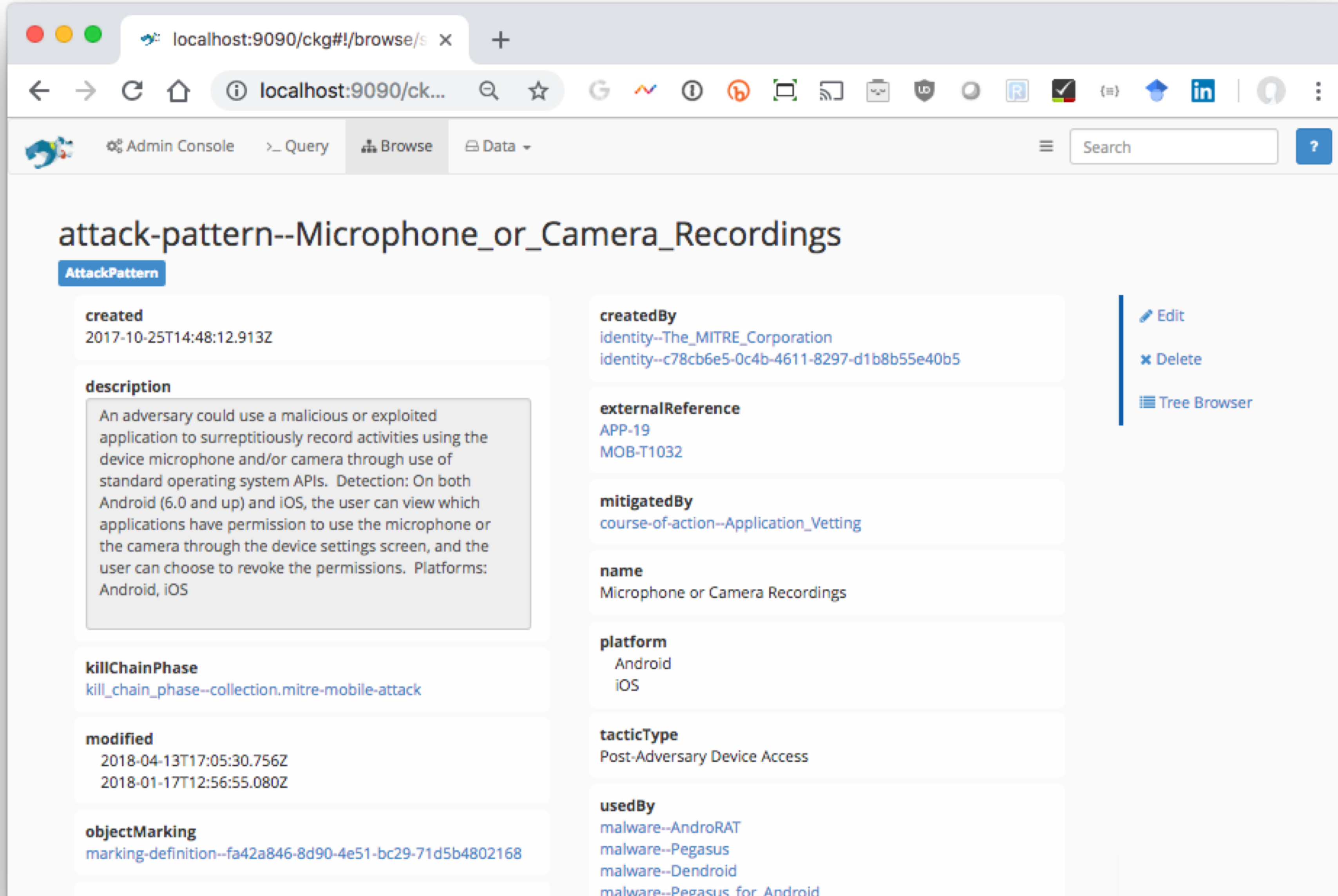
- Is :Man an rdfs:Class or a owl:Class?
- Why don’t we need to assert that :Man is some kind of class?
- Do we need to assert the disjointness both ways?
- What happens of we assert :bob a :Man; a :Woman?

Protégé



pizza (http://www.co-ode.org/ontologies/pizza/2.0.0) : [/Users/finin/ontologies/pizza-ontology/pizza.owl]
pizza (http://www.co-ode.org/ontologies/pizza/2.0.0)
Search...
Active Ontology
Entities
Object Properties
Data Properties
Annotation Properties
Individuals by class
Debugger
VOWL
Class hierarchy
Class hierarchy (inferred)
Class hierarchy: CheeseyPizza
Asserted
owl:Thing
DomainConcept
Country
Food
IceCream
Pizza
CheeseyPizza
InterestingPizza
MeatyPizza
NamedPizza
NonVegetarianPizza
RealItalianPizza
SpicyPizza
SpicyPizzaEquivalent
ThinAndCrispyPizza
UnclosedPizza
VegetarianPizza
VegetarianPizzaEquivalent1
VegetarianPizzaEquivalent2
PizzaBase
DeepPanBase
ThinAndCrispyBase
PizzaTopping
CheeseTopping
FishTopping
AnchoviesTopping
MixedSeafoodTopping
PrawnsTopping
FruitTopping
Object property hierarchy
Datatypes
CheeseyPizza — http://www.co-ode.org/ontologies/pizza/pizza.owl#CheeseyPizz
Description:
Domains (intersection)
Range (intersection)
Superproperties
Annotations
Annotations
Annotations:
Annotations
Class description
Conflicts
Description: CheeseyPizza
Equivalent To
Pizza
and (hasTopping some CheeseTopping)
SubClass Of
General class axioms
SubClass Of (Anonymous Ancestor)
hasBase some PizzaBase
To use the reasoner click Reasoner > Start reasoner
Show Inferences

StarDog


localhost:9090/ckg#!/browse/s
localhost:9090/ck...
Admin Console
Query
Browse
Data
Search
?
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
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
Edit
Delete
Tree Browser

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, supervises
- **Data type properties** relate objects to datatype values
  - owl:DatatypeProperty, e.g. phone, title, age, …
- These were made separate to make it easier to create sound and complete reasoners

Datatype Properties

- OWL uses XML Schema data types, exploiting the layered architecture of the Semantic Web

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

OWL Object Properties

Typically user-defined data types

```
: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
- Axioms

```
{ ?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)
- Example:
  - People whose sex is male and have at least one child whose sex is female and whose age is six
  - Things with exactly two arms and two legs

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:minCardinalityQ "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] ].
```

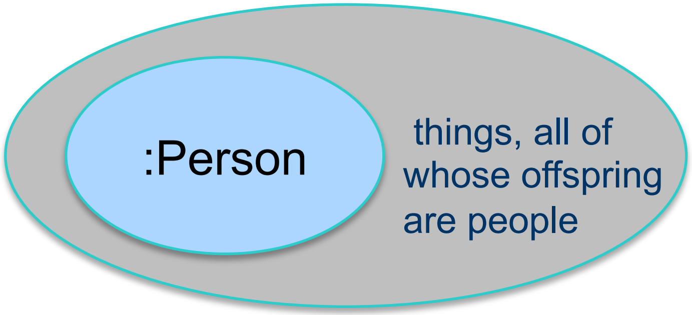
Offspring of people are people

```
: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.
???
```

What follows?

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

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

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

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:

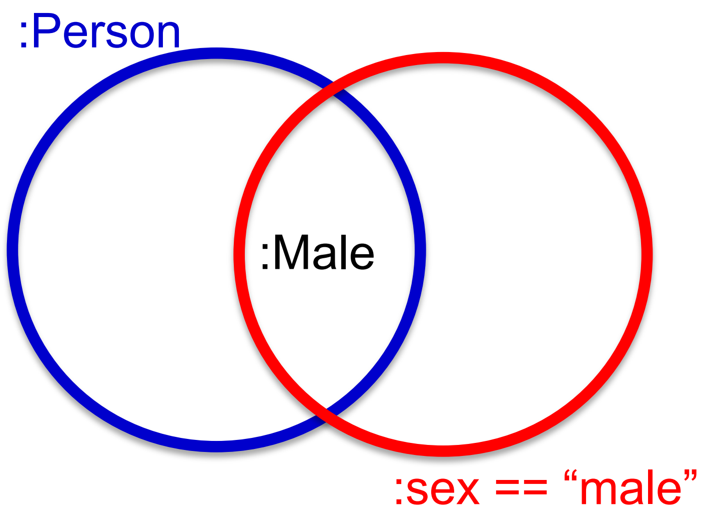
- Does this say all math courses are taught by :longhair?
- Does it say that there are some courses taught by :longhair?
- Can all classes, however defined, by paraphrased by a noun phrase in English?

A typical example

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

A typical example

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



Classes are sets in OWL

What follows?

:ed a :Male?

???

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

???

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

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

- 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 a precise 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. courses taught be at least two people

[a owl:Restriction;

owl:onProperty :isTaughtBy;

owl:minCardinality

“2”^^xsd;nonNegativeInteger] .

What does this say?

```
:Parent owl:equivalentClass
  [a owl:Restriction;
    owl:onProperty :hasChild;
    owl:minCardinality “1”^^xsd:integer] .
```

Questions:

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

Definition of a parent

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

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

If hasChild is defined as having Person as it’s 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 objects cannot have the same value

Special Properties

```
hasSameGradeAs
  a owl:ObjectProperty, owl:SymmetricProperty;
  rdfs:domain student;
  rdfs:range student .
```

Boolean Combinations

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

```
[ a :course,
    owl:Restriction;
        owl:onProperty :teaches;
        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) .
```

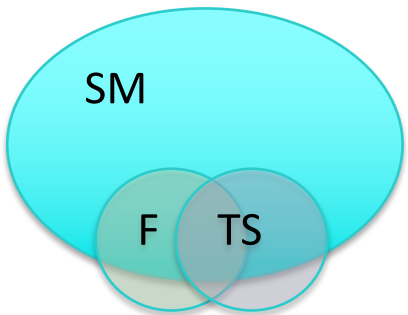
Boolean Combinations

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)]])
```

Enumerations with owl:oneOf

- *E.g., a thing that is either Monday, Tuesday, …*

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

Declaring Instances

Instances of OWL classes are declared as in RDF

```
:john
    a :academicStaffMember;
    uni:age 39^^xsd:integer .
```

No Unique-Names Assumption

- OWL does not adopt the unique-names assumption of database systems
  - That two instances have a different name or ID does not imply that they are 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

To ensure that different individuals are recognized as such, we must explicitly assert their inequality:

```
:john owl:differentFrom :mary .
```

Distinct Objects

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) ].
```

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**.

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 :ssn "249148660" . :pat2 :ssn "482962271" .

  :ssn a InverseFunctionalProperty .

- Have different values for a functional property

  :pat1 :ssn "249148660" . :pat2 :ssn "482962271" .

  :ssn a InverseFunctionalProperty .

- Are connected with an irreflexive relation

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

Combination of Features in OWL Profiles

- In different OWL languages there are different sets of restrictions regarding the application of features
- In **OWL Full**, all the language constructors may be used in any combination as long as the result is legal RDF
- **OWL DL** removes or restricts some features to ensure that complete reasoning is *tractable* or to make reasoning implementations easier

Restriction of Features in OWL DL

- **Vocabulary partitioning**

  Any resource is allowed to be only a class, a data type, a data type property, an object property, an individual, a data value, or part of the built-in vocabulary, and not more than one of these

- **Explicit typing**

  The partitioning of all resources must be stated explicitly (e.g., a class must be declared if used in conjunction with **rdfs:subClassOf**)

Restriction of Features in OWL DL

- **Property Separation**
  - The set of object properties and data type properties are disjoint
  - Therefore the following can never be specified for data type properties:
    - **owl:inverseOf**
    - **owl:FunctionalProperty**
    - **owl:InverseFunctionalProperty**
    - **owl:SymmetricProperty**

Restriction of Features in OWL DL

- **No transitive cardinality restrictions**
  - No cardinality restrictions may be placed on transitive properties
  - e.g., people with more than 5 descendants
- **Restricted anonymous classes**

  Anonymous classes are only allowed to occur as:
  - the domain and range of either **owl:equivalentClass** or **owl:disjointWith**
  - the range (but not the domain) of **rdfs:subClassOf**

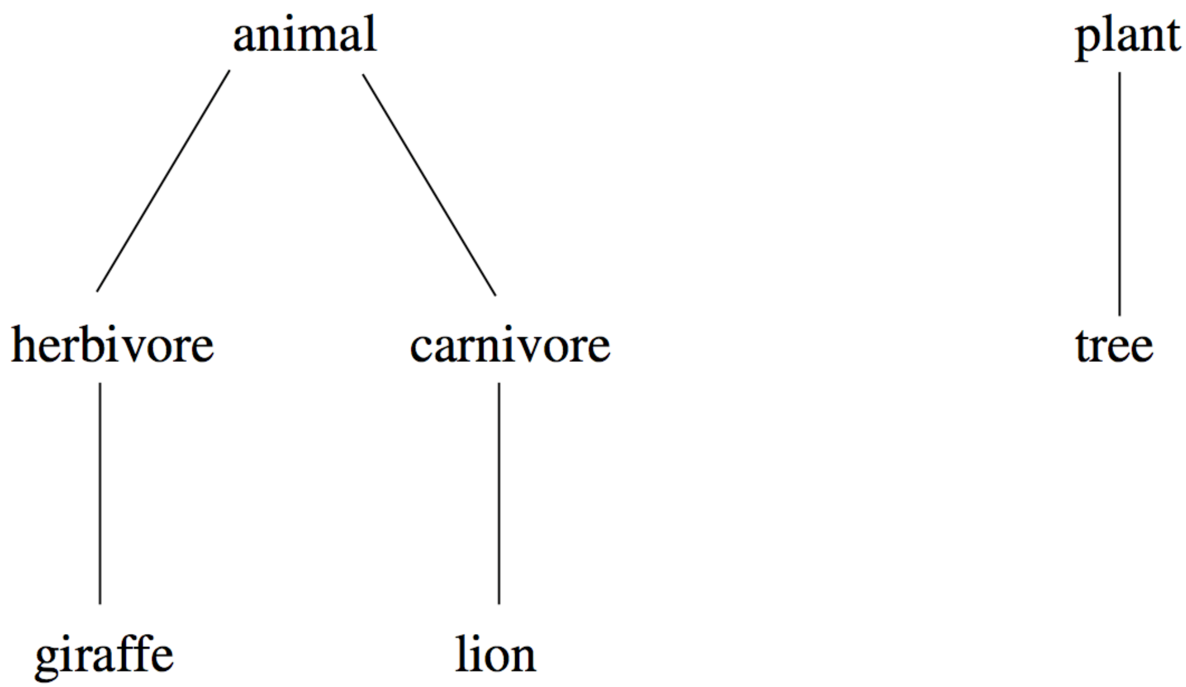
Restriction of Features in OWL Lite

- Restrictions of OWL DL and more
- **owl:oneOf**, **owl:disjointWith**, **owl:unionOf**, **owl:complementOf**, **owl:hasValue** not allowed
- Cardinality statements (minimal, maximal, exact cardinality) can only be made on values 0 or 1
- **owl:equivalentClass** statements can no longer be made between anonymous classes but only between class identifiers

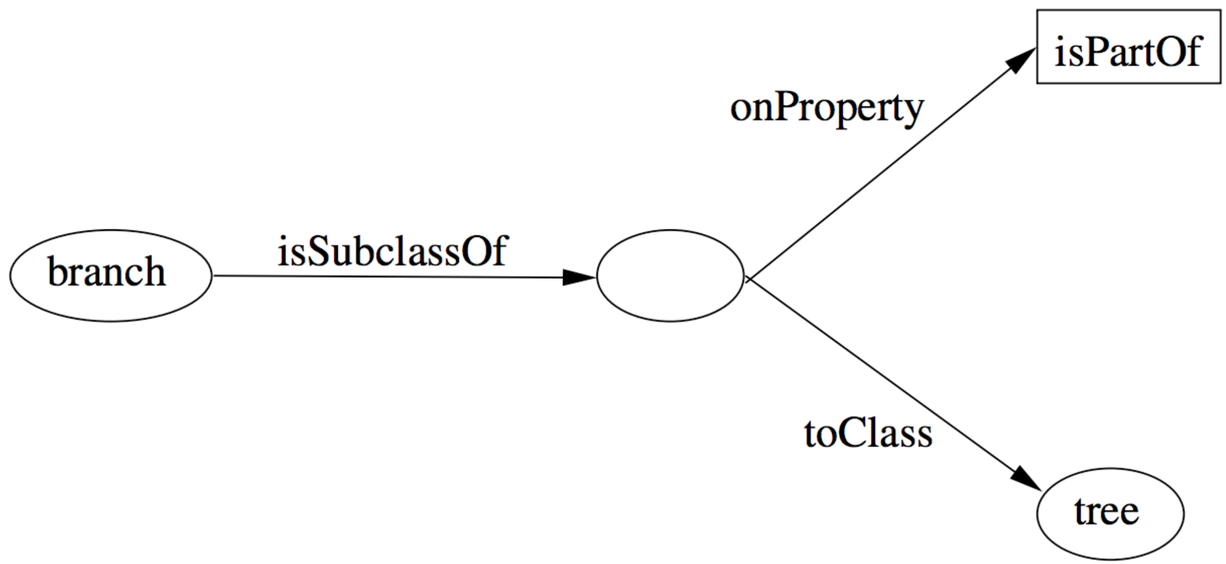
Outline

1. A bit of history
2. Basic Ideas of OWL
3. The OWL Language
4. **Examples**
5. The OWL Namespace
6. Future Extensions

African Wildlife Ontology: Classes



Branches are parts of trees



African Wildlife: Properties

```
# e.g, hand part of arm, arm part of body
:isPartOf a owl:TransitiveProperty .

# only animals eat things
:eats :domain :animal.

# the inverse of :eats in :eatenBy
:eats owl:inverseOf :eatenBy.
```

An African Wildlife: Branches

```
# plants and animals are disjoint
:Plant owl:disjointWith :Animal

# trees are plants
:Tree rdfs:subClassOf :Plant

# branches are only parts of trees
:Branch rdfs:subClassOf
  [a owl:Restriction;
    owl:allValuesFrom :Tree
    owl:onProperty :isPartOf]
```

African Wildlife: Leaves

```
# leaves are only parts of branches
:Leaf rdfs:subClassOf
  [a owl:Restriction;
    owl:allValuesFrom :Branch
    owl:onProperty :isPartOf]
```

African Wildlife: Carnivores

```
# Carnivores are exactly those animals
# that eat animals
:Carnivore owl:intersectionOf
    (:Animal,
        [a owl:Restriction;
            owl:someValuesFrom :Animal
            owl:onProperty :eats]
    ) .
```

African Wildlife: Herbivores

How can we define Herbivores?

African Wildlife: Herbivores

Here is a start

:Herbivore a owl:Class;

rdfs:comment "Herbivores are exactly those animals that eat only plants or parts of plants” .

African Wildlife: Herbivores

```
:Herbivore owl:equivalentClass
 [a owl:Class;
  owl:intersectionOf
   (:Animal
    [a owl:Restriction
      owl:onProperty :eats;
      owl:allValuesFrom
        [a owl:Class;
         owl:equivalentClass
          owl:unionOf
           (:Plant
            [a owl:Restriction;
             owl:onProperty :isPartOf;
             owl:allValuesFrom :Plant])]]])]
```

African Wildlife: Giraffes

```
#Giraffes are herbivores, and eat only leaves
Giraffe rdfs:subClassOf
    :Herbavore,
    [owl:Restriction
        owl:onProperty :eats;
        owl:allValues:From :Leaf] .
```

African Wildlife: Lions

```
# Lions are animals that eat only herbivores.

:lion rdfs:subClassOf
   :Carnivore,
   [a Restriction
     owl:onProperty :eats;
     owl:allValuesFrom :Herbavore] .
```

African Wildlife: Tasty Plants

#tasty plants are eaten both by herbivores & carnivores

???????????????

African Wildlife: Tasty Plants

```
#tasty plants are eaten both by herbivores & carnivores
:TastyPlant
    rdfs:subClassOf
        :Plant,
        [a Restriction
            owl:onProperty :eatenBy;
            owl:someValuesFrom :Herbavore],
        [a Restriction
            owl:onProperty :eatenBy;
            owl:someValuesFrom :Carnivore .]
```

Outline

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

Extensions of OWL

- Modules and Imports
- Defaults
- Closed World Assumption
- Unique Names Assumption
- Procedural Attachments
- Rules for Property Chaining

Modules and Imports

- The importing facility of OWL is very trivial:
  - It only allows importing of an entire ontology, not parts of it
- Modules in programming languages based on **information hiding**: state functionality, hide implementation details
  - Open question how to define appropriate module mechanism for Web ontology languages

Defaults

- Many practical knowledge representation systems allow inherited values to be overridden by more specific classes in the hierarchy
  - treat inherited values as defaults
- No consensus has been reached on the right formalization for the nonmonotonic behaviour of default values

Closed World Assumption

- OWL currently adopts the **open-world assumption**:
  - A statement cannot be assumed true on the basis of a failure to prove it
  - On the huge and only partially knowable WWW, this is a correct assumption
- **Closed-world assumption**: a statement is true when its negation cannot be proved
  - tied to the notion of defaults, leads to nonmonotonic behaviour

Unique Names Assumption

- Typical database applications assume that individuals with different names are indeed different individuals
- OWL follows the usual logical paradigm where this is not the case
  - Plausible on the WWW
- One may want to indicate portions of the ontology for which the assumption does or does not hold

Procedural Attachments

- A common concept in knowledge representation is to define the meaning of a term by attaching a piece of code to be executed for computing the meaning of the term
  - Not through explicit definitions in the language
- Although widely used, this concept does not lend itself very well to integration in a system with a formal semantics, and it has not been included in OWL

Rules for Property Chaining

- OWL does not allow the composition of properties for reasons of decidability
- In many applications this is a useful operation
- One may want to define properties as general rules (Horn or otherwise) over other properties
- Integration of rule-based knowledge representation and DL-style knowledge representation is an area of research

OWL 2 adds

- Qualified cardinality
  - A hand has five digits, one of which is a thumb and four of which are fingers
- Stronger datatype/range support
- Additional property characteristics
  - E.g., reflexivity
- Role chains
  - E.g., hasParent.hasSibling.hasChild
- A better defined model for punning within DL
  - Allows a term to name both a concept and an individual
- More powerful annotations

Conclusions

- OWL is the proposed standard for Web ontologies
- OWL builds upon RDF and RDF Schema:
  - (XML-based) RDF syntax is used
  - Instances are defined using RDF descriptions
  - Most RDFS modelling primitives are used
- Formal semantics and reasoning support is provided through the mapping of OWL on logics
  - Predicate logic and description logics have been used for this purpose
- While OWL is sufficiently rich to be used in practice, extensions are in the making
  - They will provide further logical features, including rules