## Knowledge Representation in Protégé –OWL
## ***Please install from CDs or USB pens provided:***

- http://www.co-ode.org/resources/tutorials/iswc2005
- Protégé 3.2 Beta – complete installation
- See instructions for other software on web site
  - You will need
  - At least one classifier - Racer, FaCT++ and/or Pellet
  - Graphviz
  - The example ontologies
  - The CO-ODE plugins not bundled with 3.2 beta (a single zip on web site)

# Ontology Design Patterns and Problems: Practical Ontology Engineering using Protege-OWL

Alan Rector[1], Natasha Noy[2], Nick Drummond[1], Mark Musen[2]

[1]University of Manchester
[2]Stanford University

rector@cs.man.ac.uk
{noy, holger}@smi.stanford.edu
musen@smi.stanford.edu

## Program

I Ontologies and “Best Practice”

II Creating an ontology – useful patterns

III Hands on examples

IV Patterns: n-ary relations

V Patterns: classes as values

VI Patterns: part-whole relations

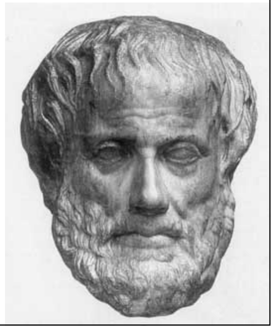
VII Summary

## Part I: Ontologies & “Best Practice”

- What are Ontologies & a review of History
- Semantic Web
- OWL
- “Best Practice”
  - Semantic Web Best Practice & Deployment Working Group (SWBP)

# What Is An Ontology?

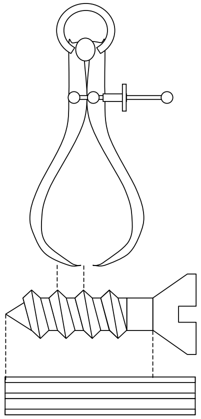
- Ontology (Socrates & Aristotle 400-360 BC)
- The study of being
- Word borrowed by computing for the explicit description of the conceptualisation of a domain:
  - concepts
  - properties and attributes of concepts
  - constraints on properties and attributes
  - Individuals (often, but not always)
- An ontology defines
  - a common vocabulary
  - a shared understanding



# Why Develop an Ontology?

- To share common understanding of the structure of descriptive information
  - among people
  - among software agents
  - between people and software
- To enable reuse of domain knowledge
  - to avoid “re-inventing the wheel”
  - to introduce standards to allow interoperability

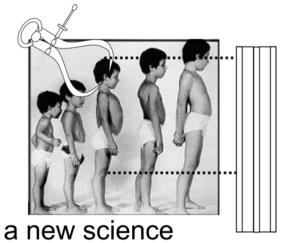
# Measure the world…*quantitative models* *(not ontologies)*



- Quantitative
  - Numerical data:
    - 2mm, 2.4V, between 4 and 5 feet
  - Unambiguous tokens
  - Main problem is accuracy at initial capture
  - Numerical analysis (e.g. statistics) well understood
- Examples:
  - How big is this breast lump?
  - What is the average age of patients with cancer ?
  - How much time elapsed between original referral and first appointment at the hospital ?

# describe the our understanding of the world - *ontologies*

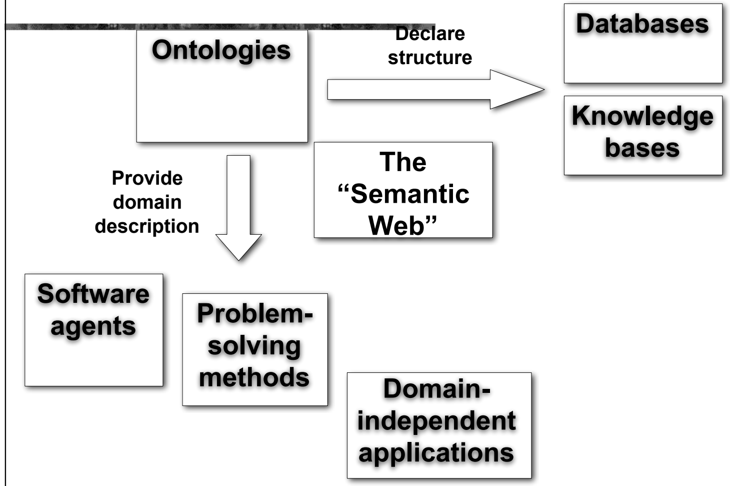
- Qualitative
  - Descriptive data
    - Cold, colder, blueish, not pink, drunk
  - Ambiguous tokens
    - What’s wrong with being drunk ?
      - Ask a glass of water.
  - Accuracy poorly defined
  - Automated analysis or aggregation is a new science
- Examples
  - Which animals are dangerous ?
  - What is their coat like?
  - What do animals eat ?



## More Reasons

- To make domain assumptions explicit
  - easier to change domain assumptions (consider a genetics knowledge base)
  - easier to understand and update legacy data
- To separate domain knowledge from the operational knowledge
  - re-use domain and operational knowledge separately (e.g., configuration based on constraints)
- To manage the combinatorial explosion

## An Ontology should be just the Beginning



## Outline

- What are Ontologies
- Semantic Web
- OWL
- Best Practice

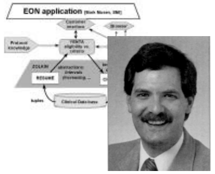
## The semantic web

- Tim Berners-Lee's dream of a computable meaningful web
  - Now critical to Web Services and Grid computing
- Metadata with everything
  - Machine understandable!
    - Ontologies are one of the keys

## Understanding rather than text matching

- Google image results for
  - Charlie Safran

  - Mark Musen



  - Alan Rector

## Ontology Examples

- Taxonomies on the Web
  - Yahoo! categories
- Catalogs for on-line shopping
  - Amazon.com product catalog
- Dublin Core and other standards for the Web
- Domain independent examples
  - Ontoclean
  - Sumo

## Ontology Technology

- “Ontology” covers a range of things
  - Controlled vocabularies – e.g. MeSH
    - Linguistic structures – e.g. WordNet
  - Hierarchies (with bells and whistles) – e.g. Gene Ontology
  - Frame representations – e.g. FMA
  - Description logic formalisms – Snomed-CT, GALEN, OWL-DL based ontologies
  - Philosophically inspired e.g. Ontoclean and SUMO

## Outline

- What are Ontologies
- Semantic Web
- OWL
- Best Practice

# OWL
# The Web Ontology Language

- W3C standard
- Collision of DAML (frames) and Oil (DLs in Frame clothing)
- Three ‘flavours’
  - OWL-Lite –simple but limited
  - OWL-DL – complex but deliverable (real soon now)
  - OWL-Full – fully expressive but serious logical/computational problems
    - Russel Paradox etc etc
  - All layered (awkwardly) on RDF Schema
- Still work in progress – see Semantic Web Best Practices & Deployment Working Group (SWBP)

# Note on syntaxes for OWL

- Three official syntaxes + Protégé-OWL syntax

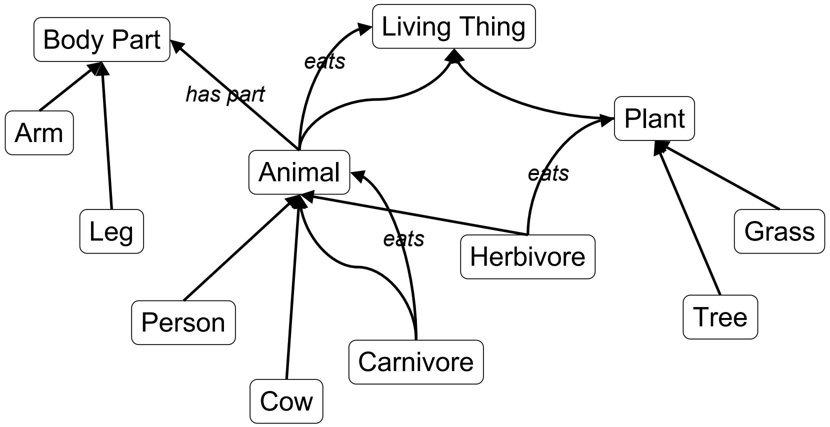
| | |
|---|---|
| Abstract syntax | -Specific to OWL |
| N3 | -OWL & RDF<br>-used in all SWBP documents |
| XML/RDF | -very verbose |
| Protégé-OWL | -Compact, derived from DL syntax |

- This tutorial uses simplified abstract syntax

| | |
|---|---|
| someValuesFrom → | *some* |
| allValuesFrom → | *only* |
| intersectionOf → | AND |
| unionOf → | OR |
| complementOf → | not |

- Protégé/OWL can generate all syntaxes

# A simple ontology: Animals



# Description Logics

- What the logicians made of Frames
  - Greater expressivity and semantic precision
    - Compositional definitions
      - “Conceptual Lego” – define new concepts from old
- To allow automatic classification & consistency checking
  - The mathematics of classification is tricky
    - Some seriously counter-intuitive results
      - The basics are simple – devil in the detail

# Description Logics

- Underneath:
  - computationally tractable subsets of first order logic
- Describes relations between Concepts/Classes
  - Individuals secondary
    - ***DL Ontologies are NOT databases!***

# Description Logics: A brief history

- Informal Semantic Networks and Frames (pre 1980)
  - Wood: *What's in a Link*; Brachman *What IS-A is and IS-A isn't*.
- First Formalisation (1980)
  - Bobrow *KRL*, Brachman: *KL-ONE*
- All useful systems are intractable (1983)
  - Brachman & Levesque: *A fundamental tradeoff*
    - Hybrid systems: T-Box and A-Box
- All tractable systems are useless (1987-1990)
  - Doyle and Patel: *Two dogmas of Knowledge Representation*

# A brief history of KR

- 'Maverick' incomplete/intractable logic systems (1985-90)
  - GRAIL, LOOM, Cyc, Apelon, …,
- Practical knowledge management systems based on frames
  - Protégé
- The German School: Description Logics (1988-98)
  - Complete decidable algorithms using tableaux methods (1991-1992)
  - Detailed catalogue of complexity of family – "alphabet soup of systems"
- Optimised systems for practical cases (1996-)
- Emergence of the Semantic Web
  - Development of DAML (frames), OIL (DLs) → DAML+OIL → OWL
    - **Development of Protégé-OWL**
    - **A dynamic field – constant new developments & possibilities**

# Outline

- What are Ontologies
- Semantic Web
- OWL
- "Best Practice"
  - Semantic Web Best Practice & Deployment Working Group (SWBP)

# Why the
# “Best Practice working Group”?

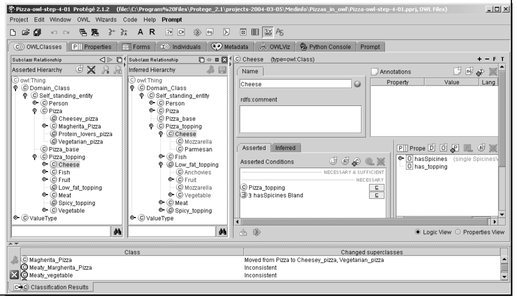
- There is no established “best practice”
  - It is new; We are all learning
  - A place to gather experience
  - A catalogue of things that work – Analogue of Software Patterns
    - Some pitfalls to avoid
      - ***…but there is no one way***
- Learning to build ontologies
  - Too many choices
    - Need starting points for gaining experience
- Provide requirements for tool builders

# Contributing to “best practice”

- Please give us feedback
  - Your questions and experience
    - On the SW in general: **semanticweb@yahoogroups.com**
    - For specific feedback to SWBP
      - Home & Mail Archive: **http://www.w3.org/2001/sw/BestPractices/** **public-swbp-wg@w3.org**

# Protégé OWL: New tools for ontologies

- Transatlantic collaboration
- Implement robust OWL environment within PROTÉGÉ framework
- Shared UI components
- Enables hybrid working



Asserted Hierarchy:

- owl:Thing
- Domain_Class
  - Self_standing_entity
    - Person
    - Pizza
      - Cheesey_pizza
      - Magherita_Pizza
      - Protein_lovers_pizza
      - Vegetarian_pizza
    - Pizza_base
    - Pizza_topping
      - Cheese
      - Fish
      - Fruit
      - Low_fat_topping
      - Meat
      - Spicy_topping
      - Vegetable
- ValueType

Inferred Hierarchy:

- owl:Thing
- Domain_Class
  - Self_standing_entity
    - Person
    - Pizza
    - Pizza_base
    - Pizza_topping
      - Cheese
        - Mozzarella
        - Parmesan
      - Fish
      - Low_fat_topping
        - Anchovies
        - Fruit
        - Mozzarella
        - Vegetable
      - Meat
      - Spicy_topping
- ValueType

Name: Cheese

Asserted Conditions:

- Pizza_topping ⊑
- ∃ hasSpicines Bland ⊑

Properties:

- hasSpicines (single Spicines\
- has_topping

| Class | Changed superclasses |
|---|---|
| Magherita_Pizza | Moved from Pizza to Cheesey_pizza, Vegetarian_pizza |
| Meaty_Margherita_Pizza | Inconsistent |
| Meaty_vegetable | Inconsistent |

# Protégé-OWL & CO-ODE

- Joint work: Stanford & U Manchester + Southampton & Epistemics
  - Please give us feedback on tools – mailing lists & forums at:
    - **protege.stanford.edu**
    - **www.co-ode.org**
- Don’t beat your head against a brick wall!
  - Look to see if others have had the same problem; If not…
  - ***ASK!***
    - ***We are all learning.***

# Part II – Creating an ontology Useful patterns

- *Upper ontologies & Domain ontologies*
- Building from trees and untangling
- Using a classifier
- Closure axioms
- Specifying Values
- n-ary relations
- Classes as values – using the ontology
- Part-whole relations

# Upper Ontologies

- Ontology Schemas
  - High level abstractions to constrain construction
    - e.g. There are “Objects” & “Processes”
  - Highly controversial
    - Sumo, Dolce, Onions, GALEN, SBU,…
  - Needed when you work with many people together
  - NOT in this tutorial – a different tutorial

# Domain Ontologies

- Concepts specific to a field
  - Diseases, animals, food, art work, languages, …
  - The place to start
    - Understand ontologies from the bottom up
      - Or middle out
- Levels
  - Top domain ontologies – the starting points for the field
    - Living Things, Geographic Region, Geographic_feature
  - Domain ontologies – the concepts in the field
    - Cat, Country, Mountain
  - Instances – the things in the world
    - Felix the cat, Japan, Mt Fuji

# Part II – Useful Patterns (continued)

- Upper ontologies & Domain ontologies
- *Building from trees and untangling*
- *Using a classifier*
- *Closure axioms & Open World Reasoning*
- Specifying Values
- n-ary relations
- Classes as values – using the ontology

# Example: Animals & Plants

- Dog
- Cat
- Cow
- Person
- Tree
- Grass
- Herbivore
- Male
- Female

- Carnivore
- Plant
- Animal
- Fur
- Child
- Parent
- Mother
- Father

- Dangerous
- Pet
- Domestic Animal
- Farm animal
- Draft animal
- Food animal
- Fish
- Carp
- Goldfish

# Example: Animals & Plants

- Dog
- Cat
- Cow
- Person
- Tree
- Grass
- Herbivore
- Male
- Female

- Carnivore
- Plant
- Animal
- Fur
- Child
- Parent
- Mother
- Father

- Healthy
- Pet
- Domestic Animal
- Farm animal
- Draft animal
- Food animal
- Fish
- Carp
- Goldfish

# Choose some main axes
# Add abstractions where needed; identify relations; Identify definable things, make names explicit

- Living Thing
  - Animal
    - Mammal
      - Cat
      - Dog
      - Cow
      - Person
    - Fish
      - Carp
      - Goldfish
  - Plant
    - Tree
    - Grass
    - Fruit

- Modifiers
  - domestic
    - pet
    - Farmed
      - Draft
      - Food
  - Wild
  - Health
    - healthy
    - sick
  - Sex
    - Male
    - Female
  - Age
    - Adult
    - Child

- Relations
  - eats
  - owns
  - parent-of
  - …
- Definable
  - Carinvore
  - Herbivore
  - Child
  - Parent
  - Mother
  - Father
  - Food Animal
  - Draft Animal

# Reorganise everything but “definable” things into pure trees – these will be the “primitives”

- Primitives
  - Living Thing
    - Animal
      - Mammal
        - Cat
        - Dog
        - Cow
        - Person
      - Fish
        - Carp
          Goldfish
    - Plant
      - Tree
      - Grass
      - Fruit

- Modifiers
  - **Domestication**
    - **Domestic**
    - **Wild**
  - **Use**
    - **Draft**
    - **Food**
    - **pet**
  - **Risk**
    - **Dangerous**
    - **Safe**
  - **Sex**
    - **Male**
    - **Female**
  - **Age**
    - **Adult**
    - **Child**

- Relations
  - eats
  - owns
  - parent-of
  - …
- Definables
  - Carnivore
  - Herbivore
  - Child
  - Parent
  - Mother
  - Father
  - Food Animal
  - Draft Animal

# Set domain and range constraints for properties

- Animal *eats* Living_thing
  - *eats* domain: Animal;
    range: Living_thing
- Person *owns* Living_thing except person
  - *owns* domain: Person
    range: Living_thing & not Person
- Living_thing *parent_of* Living_thing
  - *parent_of:* domain: Animal
    range: Animal

# Define the things that are definable from the primitives and relations

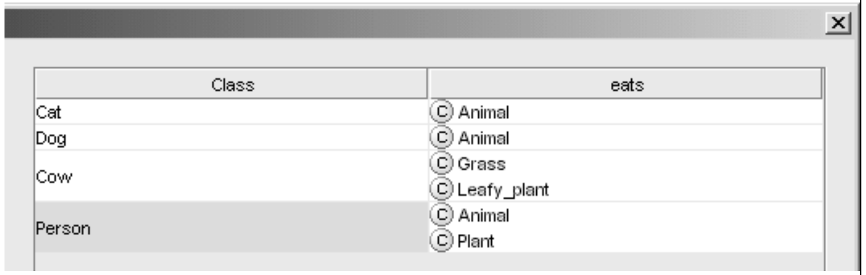
- Parent =
  Animal and *parent_of* some Animal
- Herbivore=
  Animal and *eats* only Plant
- Carnivore =
  Animal and *eats* only Animal

# Which properties can be filled in at the class level now?

- What can we say about *all* members of a class?
  - *eats*
    - *All cows eat some plants*
    - *All cats eat some animals*
    - *All dogs eat some animals* &
      *eat some plants*

# Fill in the details

(can use property matrix wizard)



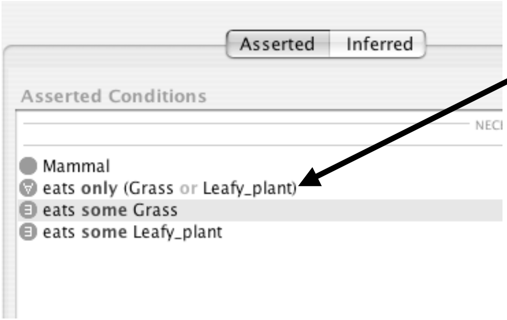
| Class | eats |
|---|---|
| Cat | Ⓒ Animal |
| Dog | Ⓒ Animal |
| Cow | Ⓒ Grass<br>Ⓒ Leafy_plant |
| Person | Ⓒ Animal<br>Ⓒ Plant |

# Check with classifier

- Cows should be Herbivores
  - Are they? why not?
    - What have we said?
      - Cows are animals and, *amongst other things*, eat *some* grass and eat some leafy_plants
    - What do we need to say: Closure axiom
      - Cows are animals and, *amongst other things,* eat *some* plants and eat *only* plants

# Closure Axiom

- Cows are animals and, *amongst other things,* eat *some* plants and eat *only* plants

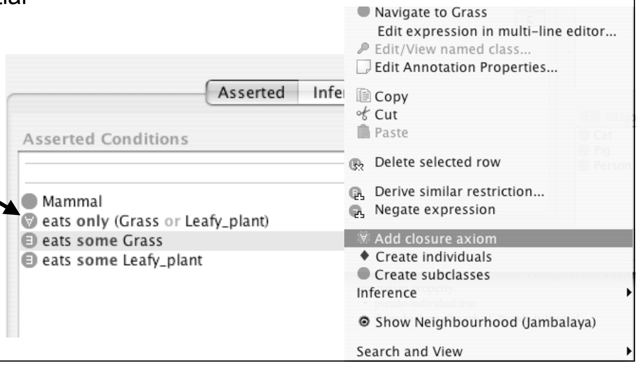


Mammal
eats only (Grass or Leafy_plant)
eats some Grass
eats some Leafy_plant

**Closure Axiom**

# In the tool

- Right mouse button short cut for closure axiom
  - for any existential restriction



Mammal
eats only (Grass or Leafy_plant)
eats some Grass
eats some Leafy_plant

adds closure axiom

# Open vs Closed World reasoning

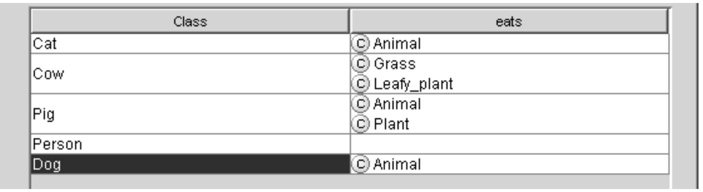
- Open world reasoning
  - Negation as contradiction
    - Anything might be true unless it can be proven false
      - Reasoning about *any world consistent with this one*
- Closed world reasoning
  - Negation as failure
    - Anything that cannot be found is false
      - Reasoning about *this world*

# Normalisation and Untangling

Let the reasoner do multiple classification

- Tree
  - Everything has just one parent
    - A ‘strict hierarchy’
- Directed Acyclic Graph (DAG)
  - Things can have multiple parents
    - A ‘Polyhierarchy’
- Normalisation
  - Separate primitives into disjoint trees
  - Link the trees with restrictions
    - Fill in the values

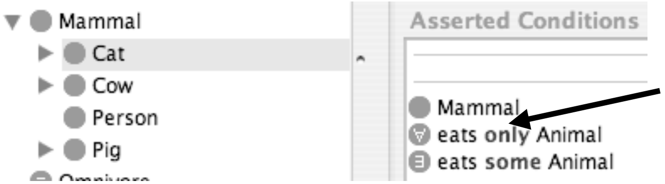
# Tables are easier to manage than DAGs / Polyhierarchies



| Class | eats |
|---|---|
| Cat | Ⓒ Animal |
| Cow | Ⓒ Grass<br>Ⓒ Leafy_plant |
| Pig | Ⓒ Animal<br>Ⓒ Plant |
| Person | |
| Dog | Ⓒ Animal |

…and get the benefit of inference:
Grass and Leafy_plants are both kinds of Plant

# Remember to add any closure axioms



▼ ● Mammal
▶ ● Cat
▶ ● Cow
● Person
▶ ● Pig
● Omnivore

Asserted Conditions

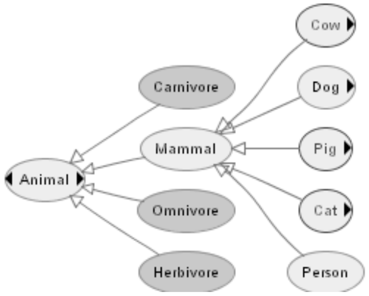
● Mammal
∀ eats only Animal
∃ eats some Animal

**Closure Axiom**

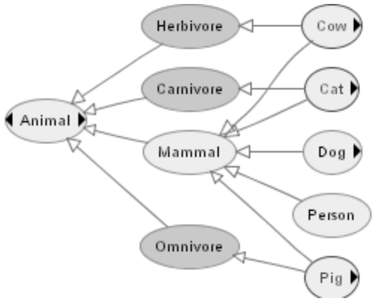
Then let the reasoner do the work

# Normalisation: From Trees to DAGs

- Before classification
  - A tree



- After classification
  - A DAG
    - **D**irected **A**cyclic **G**raph



# Part II – Useful Patterns (continued)

- Upper ontologies & Domain ontologies
- Building from trees and untangling
- Using a classifier
- Closure axioms & Open World Reasoning
- *Specifying Values*
- n-ary relations
- Classes as values – using the ontology

# Examine the modifier list

- Modifiers
  - **Domestication**
    - **Domestic**
    - **Wild**
  - **Use**
    - **Draft**
    - **Food**
  - **Risk**
    - **Dangerous**
    - **Safe**
  - **Sex**
    - **Male**
    - **Female**
  - **Age**
    - **Adult**
    - **Child**

- Identify modifiers that have mutually exclusive values
  - Domestication
  - Risk
  - Sex
  - Age
- Make meaning precise
  - Age → Age_group
- NB Uses are not mutually exclusive
  - Can be both a draft (pulling) and a food animal

# Extend and complete lists of values

- Modifiers
  - **Domestication**
    - **Domestic**
    - **Wild**
    - **Feral**
  - **Risk**
    - **Dangerous**
    - **Risky**
    - **Safe**
  - **Sex**
    - **Male**
    - **Female**
  - **Age**
    - **Infant**
    - **Toddler**
    - **Child**
    - **Adult**
    - **Elderly**

- Identify modifiers that have mutually exclusive values
  - Domestication
  - Risk
  - Sex
  - Age
- Make meaning precise
  - Age → Age_group
- NB Uses are not mutually exclusive
  - Can be both a draft and a food animal

# Note any hierarchies of values

- Modifiers
  - **Domestication**
    - **Domestic**
    - **Wild**
    - **Feral**
  - **Risk**
    - **Dangerous**
    - **Risky**
    - **Safe**
  - **Sex**
    - **Male**
    - **Female**
  - **Age**
    - **Child**
      - **Infant**
      - **Toddler**
    - **Adult**
      - **Elderly**

- Identify modifiers that have mutually exclusive values
  - Domestication
  - Risk
  - Sex
  - Age
- Make meaning precise
  - Age → Age_group
- NB Uses are not mutually exclusive
  - Can be both a draft and a food animal

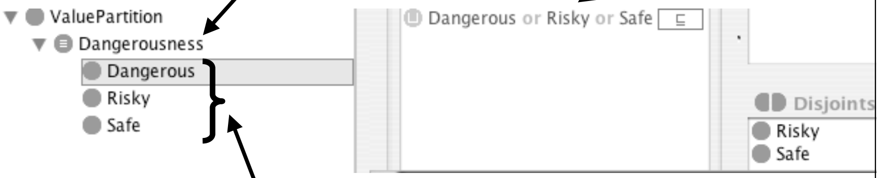
# Specify Values for each:
## Two methods

- Value partitions
  - Classes that partition a Quality
    - The disjunction of the partition classes equals the quality class
- Symbolic values
  - Individuals that enumerate all states of a Quality
    - The enumeration of the values equals the quality class

# Method 1: Value Partitions- example "Dangerousness"

- A parent quality – Dangerousness
- Subqualities for each degree
  - Dangerous, Risky, Safe
- All subqualities disjoint
- Subqualities 'cover' parent quality
  - Dangerousness = Dangerous OR Risky OR Safe
- A functional property has_dangerousness
  - Range is parent quality, e.g. Dangerousness
  - Domain must be specified separately
- Dangerous_animal = Animal *and* has_dangerousness *some* Dangerous

# as created by Value Partition wizard



quality

covering axiom

ValuePartition

Dangerousness

Dangerous

Risky

Safe

Dangerous or Risky or Safe ⊑

Disjoints

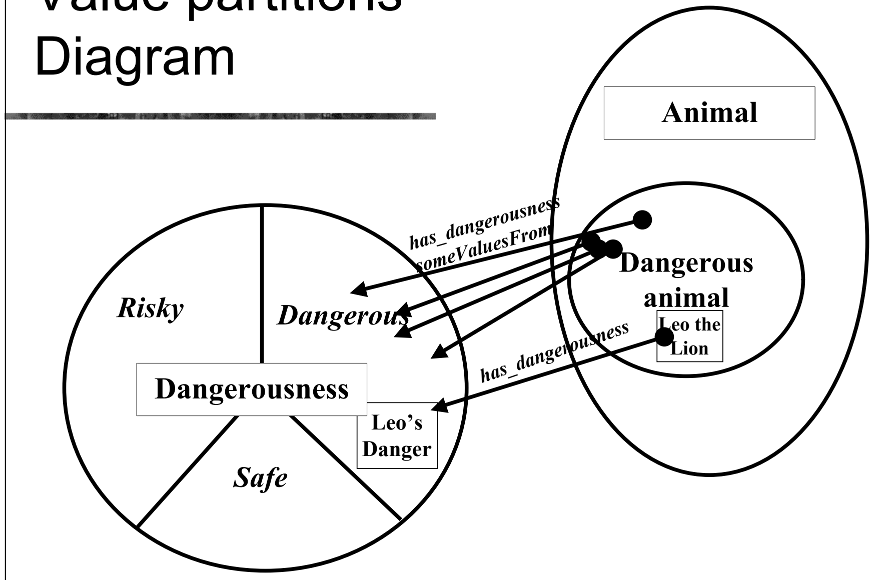
Risky

Safe

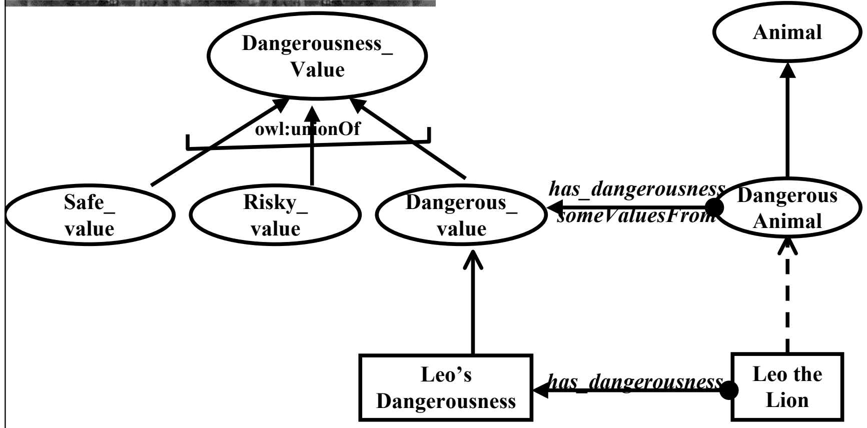
partitions

disjoints

# Value partitions Diagram



# Value partitions UML style



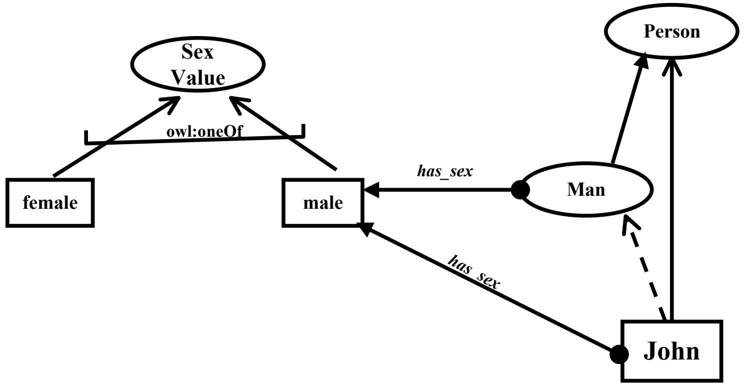
# Method 2: Value sets – Example Sex

- There are only two sexes
  - Can argue that they are things
    - "Administrative sex" definitely a thing
    - "Biological sex" is more complicated

# Method 2: Value sets- example Sex

- A parent quality – Sex_value
- Individuals for each value
  - male, female
- Values all different (NOT assumed by OWL)
- Value type is enumeration of values
  - Sex_value = {male, female}
- A functional property has_sex
  - Range is parent quality, e.g. Sex_value
  - Domain must be specified separately
- Male_animal =
  Animal *and* has_sex *is* male

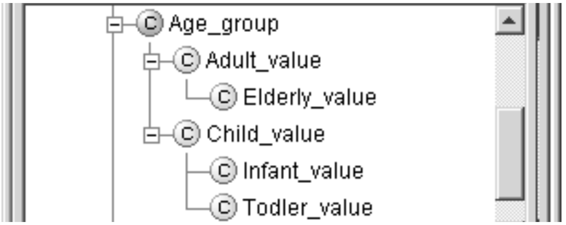
# Value sets UML style



# Issues in specifying values

- Value Partitions
  - Can be subdivided and specialised
  - Fit with philosophical notion of a quality space
  - Require interpretation to go in databases as values
    - **in theory but rarely considered in practice**
  - Work better with existing classifiers in OWL-DL
- **Value Sets**
  - Cannot be subdivided
  - Fit with intuitions
  - More similar to data bases – no interpretation
  - Work less well with existing classifiers

# Value partitions – practical reasons for subdivisions

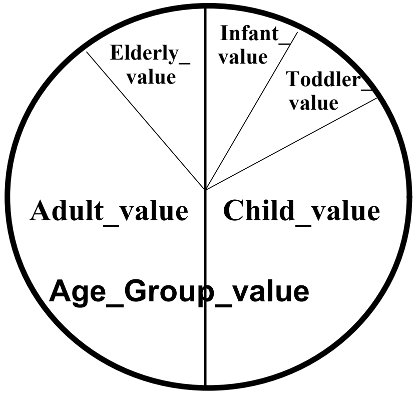


- Age_group
  - Adult_value
    - Elderly_value
  - Child_value
    - Infant_value
    - Todler_value

- “All elderly are adults”
- “All infants are children”
- etc.

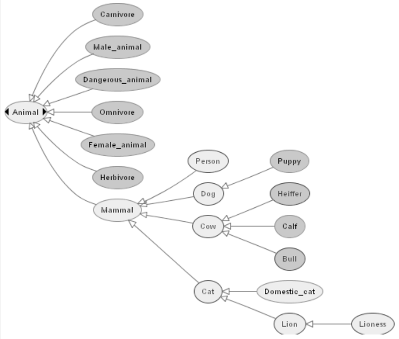
- See also “Normality_status” in http://www.cs.man.ac.uk/~rector/ontologies/mini-top-bio
  - One can have complicated value partitions if needed.

# Picture of subdivided value partition

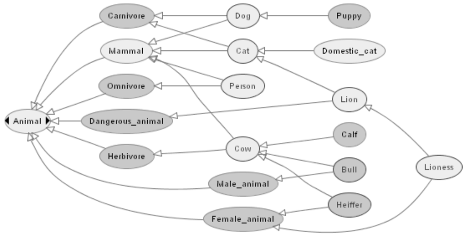


# More defined kinds of animals

- Before classification, trees



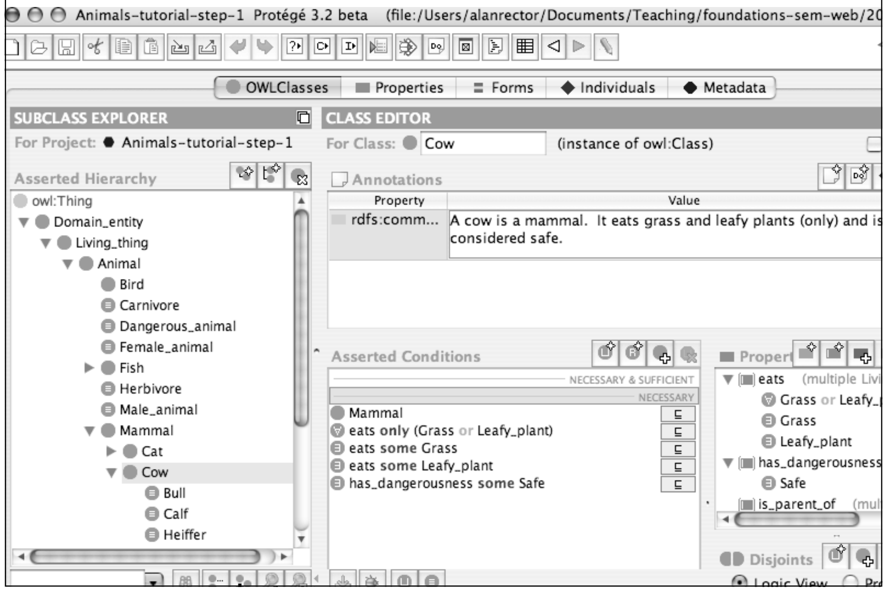
- After classification, DAGs



# Part III – Hands On

- Be sure you have installed the software
  - (See front page)
- Open Animals-tutorial-step-1

# Explore the interface



Animals-tutorial-step-1 Protégé 3.2 beta (file:/Users/alanrector/Documents/Teaching/foundations-sem-web/20

For Project: ● Animals-tutorial-step-1

For Class: ● Cow (instance of owl:Class)

owl:Thing
- Domain_entity
  - Living_thing
    - Animal
      - Bird
      - Carnivore
      - Dangerous_animal
      - Female_animal
      - Fish
      - Herbivore
      - Male_animal
      - Mammal
        - Cat
        - Cow
          - Bull
          - Calf
          - Heiffer

| Property | Value |
|---|---|
| rdfs:comm... | A cow is a mammal. It eats grass and leafy plants (only) and is considered safe. |

Mammal ⊑
eats only (Grass or Leafy_plant) ⊑
eats some Grass ⊑
eats some Leafy_plant ⊑
has_dangerousness some Safe ⊑

eats (multiple Livi
- Grass or Leafy_
- Grass
- Leafy_plant

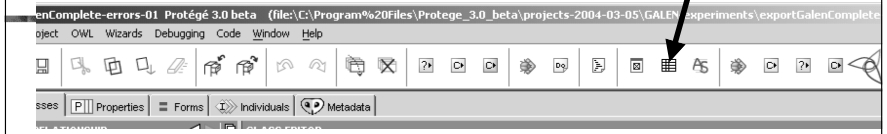
has_dangerousness
- Safe

is_parent_of (mul

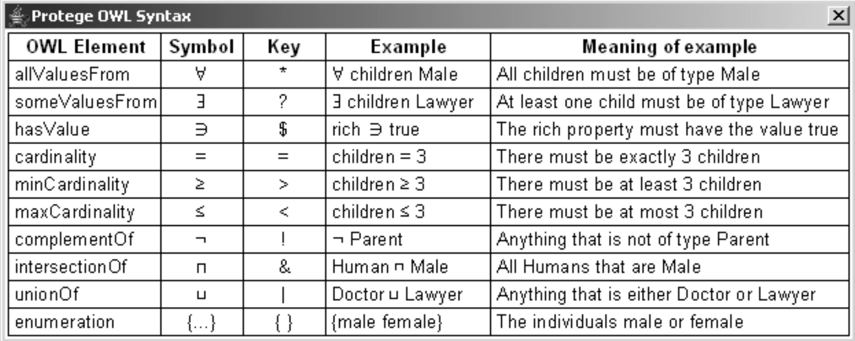
# Protégé - new abbreviated abstract syntax

| | | |
|---|---|---|
| some | someValuesFrom | ∃ |
| only | allValuesFrom | ∀ |
| has | hasValue | ∋ |
| …and… | intersectionOf(…) | ⊓ |
| …or… | unionOf(…) | ⊔ |
| not | complementOf() | ¬ |
| min | minCardinality | |
| max | maxCardinality | |
| exactly | cardinality | |
| =, ≤, ≥ | Numeric comparisons (coming soon) | |

# Protégé Old (≤v3.1) Syntax



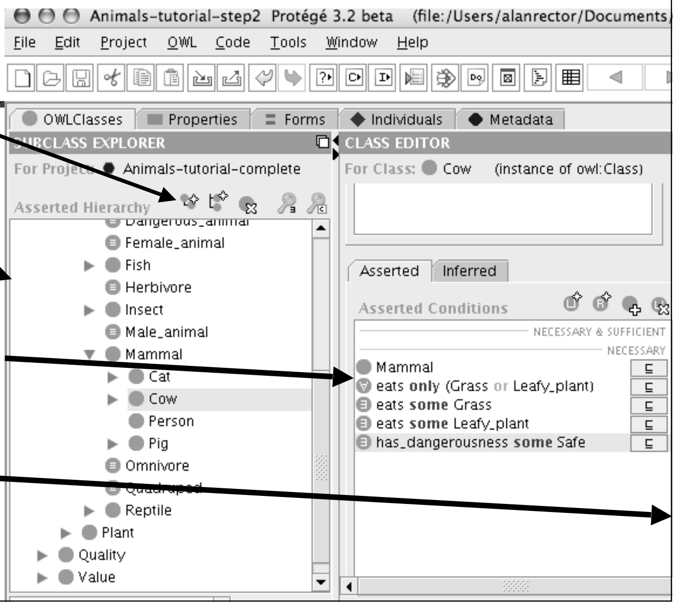
enComplete-errors-01 Protégé 3.0 beta (file:\C:\Program%20Files\Protege_3.0_beta\projects-2004-03-05\GALEN\experiments\exportGalenComplete



Protege OWL Syntax

| OWL Element | Symbol | Key | Example | Meaning of example |
|---|---|---|---|---|
| allValuesFrom | ∀ | * | ∀ children Male | All children must be of type Male |
| someValuesFrom | ∃ | ? | ∃ children Lawyer | At least one child must be of type Lawyer |
| hasValue | ∋ | $ | rich ∋ true | The rich property must have the value true |
| cardinality | = | = | children = 3 | There must be exactly 3 children |
| minCardinality | ≥ | > | children ≥ 3 | There must be at least 3 children |
| maxCardinality | ≤ | < | children ≤ 3 | There must be at most 3 children |
| complementOf | ¬ | ! | ¬ Parent | Anything that is not of type Parent |
| intersectionOf | ⊓ | & | Human ⊓ Male | All Humans that are Male |
| unionOf | ⊔ | \| | Doctor ⊔ Lawyer | Anything that is either Doctor or Lawyer |
| enumeration | {…} | { } | {male female} | The individuals male or female |

# Explore the interface



Mammal
eats only (Grass or Leafy_plant)
eats some Grass
eats some Leafy_plant
has_dangerousness some Safe

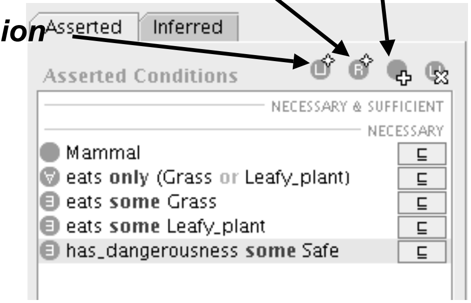
New Subclass icon

Asserted Hierarchy

Class Description

Disjoint Classes

# Explore the interface



Mammal
eats only (Grass or Leafy_plant)
eats some Grass
eats some Leafy_plant
has_dangerousness some Safe

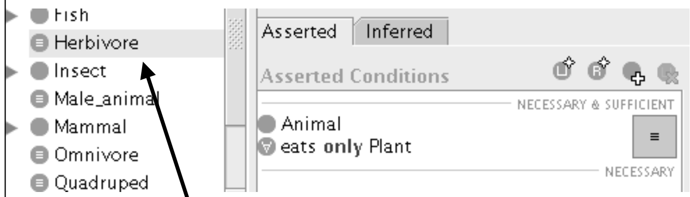
New *restriction*

Add *superclass*

New *expression*

Description "Necessary Conditions"

# Explore the interface

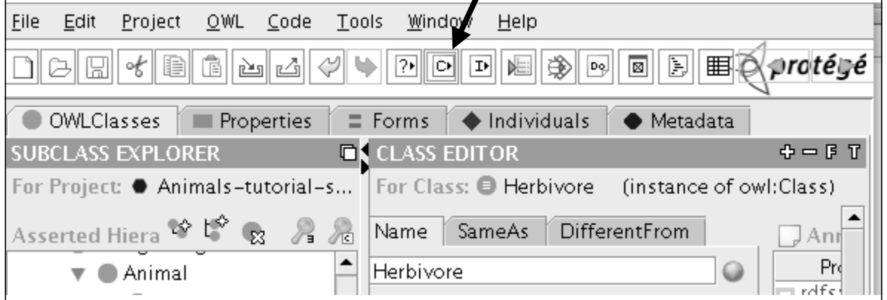


Animal
eats only Plant

Definition "Necessary & Sufficient Conditions"

"Defined class" has *necessary & sufficient conditions* ( ☰ )

# Explore the interface



Classify button (racer must be running*)

*Or some other DIG compliant classifier

## Exercise 1

- Create a new animal, an Elephant and an Ape
  - Make them disjoint from the other animals
  - Make the ape an omnivore
    - eats animals and eats plants
  - Make the sheep a herbivore
    - eats plants and only plants

## Exercise 1b: Classification

- Check it with the classifier
- Is Sheep classified under Herbivore
  - If not, have you forgot the closure axiom?
- Did it all turn red?
  - Do you have too many disjoint axioms?

## Exercise 1c: checking disjoints – make things that should be inconsistent

- Create a Probe_Sheep_and_Cow that is a kind of both Sheep and Cow
- Create a Probe_Ape_and_Man that is a kind of both Ape and Man
- Run the classifier
- Did both probes turn red?
  - If not, check the disjoints

## Exercise 2: A new value partition

- Create a new value partition
  - Size_partition
    - Big
    - Medium
    - Small
- Describe
  - Lions, Cows, and Elephants asBig
    domestic_cat as Small
    the rest Medium

# Exercise 2b

- Define Big_animal and Small_animal
  - Does the classification work
- Extra
  - Make a subdivision of Big for Huge and make elephants Huge
    - Do elephants still classify as “Big Animal

# Part IV – Patterns: n-ary relations

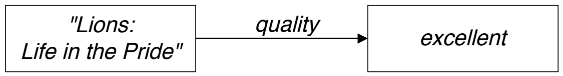
- Upper ontologies & Domain ontologies
- Building from trees and untangling
- Using a classifier
- Closure axioms & Open World Reasoning
- Specifying Values
- *n-ary relations*
- Classes as values – using the ontology

# Saying something about a restriction

- Not just
  - that an a book is good but who said so
  - And its price
  - And where to buy it
- But can say nothing about properties
  - except special thing
    - Super and subproperties
    - Functional, transitive, symmetric

# N-ary Relations

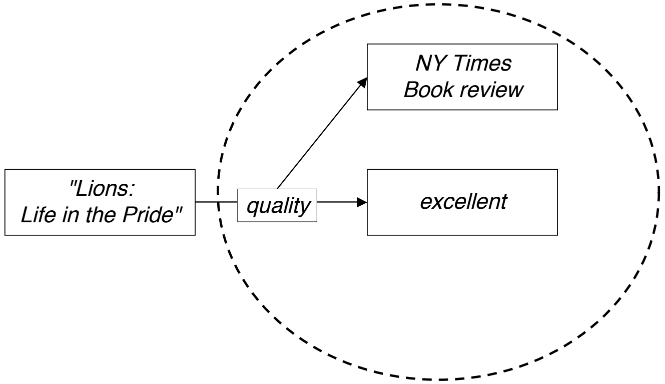
*Binary Relation*



*"Lions: Life in the Pride"* —*quality*→ *excellent*

- According to whom?

# Adding attributes to a  Relation



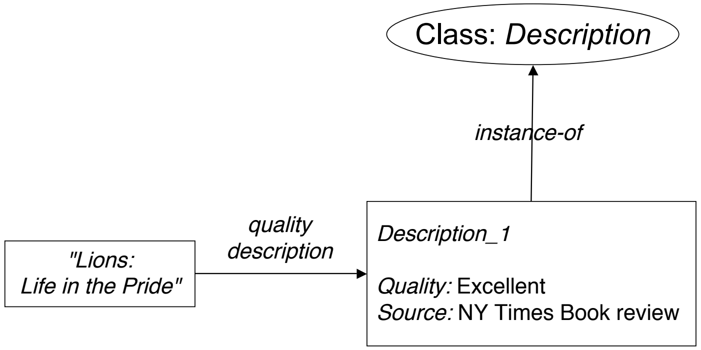
"Lions:
Life in the Pride"

quality

NY Times
Book review

excellent

# Define a class for a relation: Reification



Class: *Description*

*instance-of*

*quality description*

"Lions:
Life in the Pride"

*Description_1*

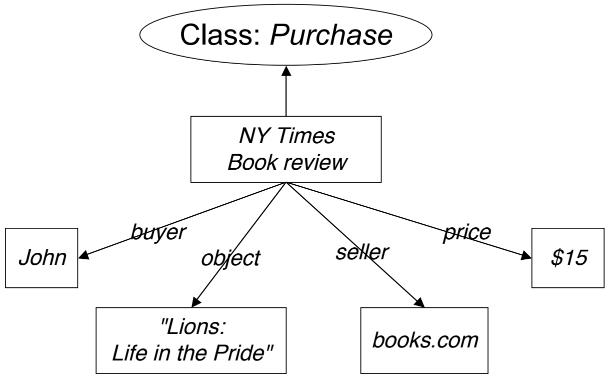
*Quality:* Excellent
*Source:* NY Times Book review

# A Relation Between Multiple Participants

*John buys "Lions:Life in the Pride" from books.com for $15*

- Participants in this relation:
  - John
  - "Lions: Life in the Pride"
  - books.com
  - $15
- No clear "originator"

# Network of Participants



Class: *Purchase*

NY Times
Book review

buyer

object

seller

price

John

"Lions:
Life in the Pride"

books.com

$15

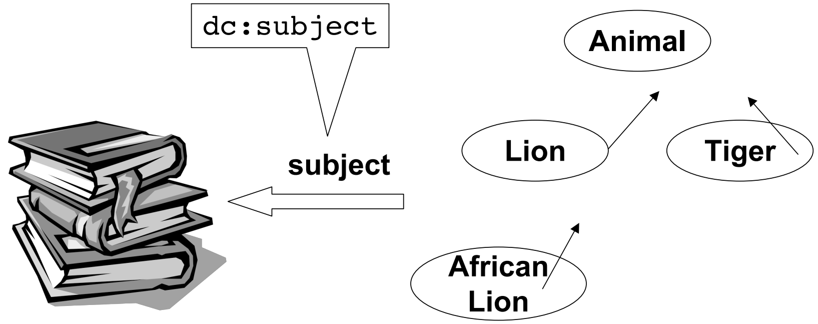
# Considerations

- Choosing the right pattern: often subjective
  - Pattern 1: additional attributes for a relation
  - Pattern 2: a network of participants
- Instances of reified relations usually don't have meaningful names
- Defining inverse relations is more tricky

# Part V – Patterns: Classes as values

- Upper ontologies & Domain ontologies
- Building from trees and untangling
- Using a classifier
- Closure axioms & Open World Reasoning
- Specifying Values
- n-ary relations
- *Classes as values – using the ontology*
- Part-whole relations

# Using Classes as Property Values


dc:subject

subject

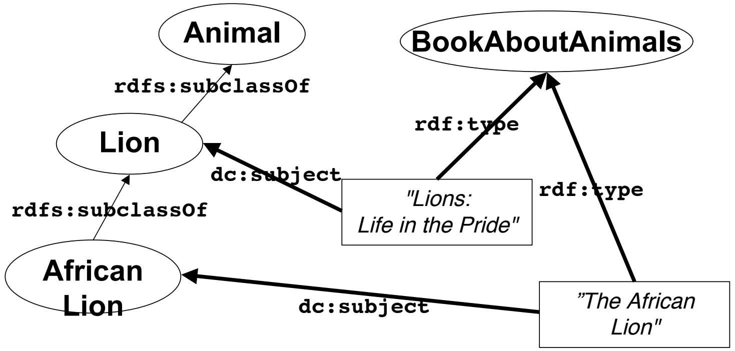
Animal

Lion

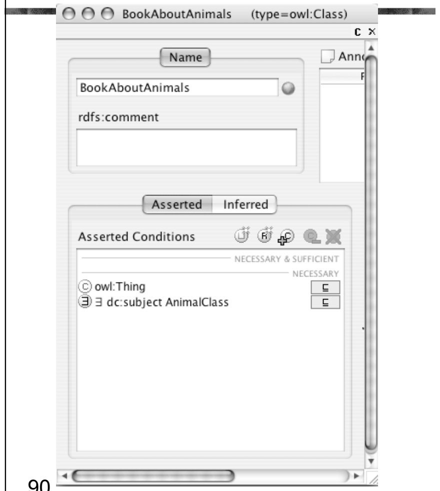
Tiger

African Lion

# Using Classes Directly As Values



# Representation in Protégé

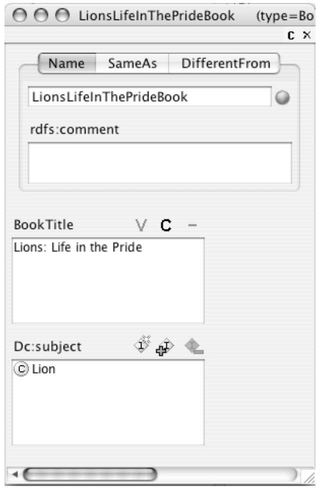


Name: BookAboutAnimals

rdfs:comment

Asserted Conditions

- owl:Thing ⊑
- ∃ dc:subject AnimalClass ⊑



Name: LionsLifeInThePrideBook

rdfs:comment

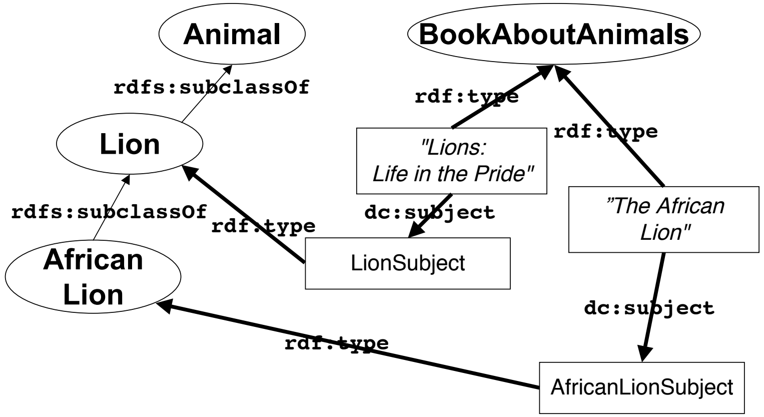
BookTitle: Lions: Life in the Pride

Dc:subject: Lion

# Approach 1: Considerations

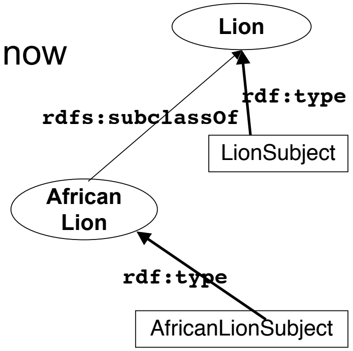
- Compatible with OWL Full and RDF Schema
- Outside OWL DL
  - Because classes cannot be values in OWL-DL
    - Nothing can be both a class and and instance

# Approach 2: Hierarchy of Subjects

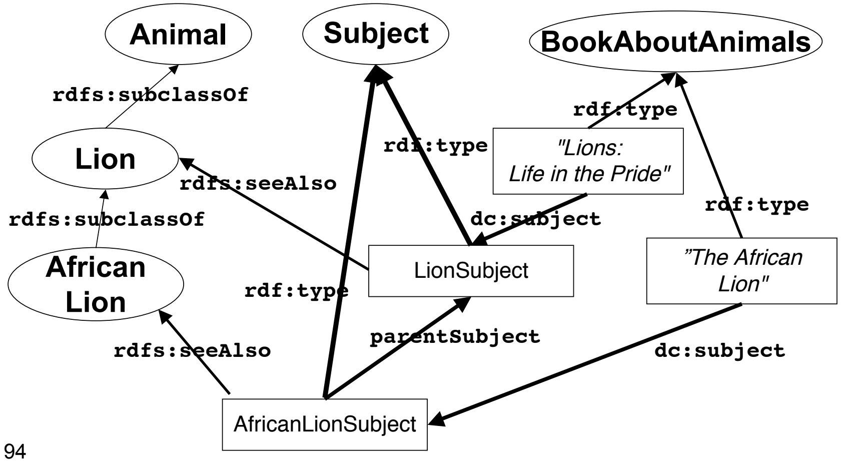


# Hierarchy of Subjects: Considerations

- Compatible with OWL DL
- Instances of class Lion are now subjects
- No direct relation between LionSubject and AfricalLionSubject
- Maintenance penalty

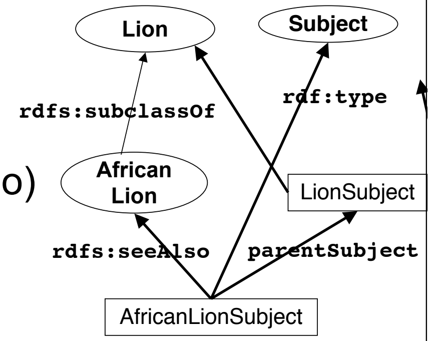


# Hierarchy of Subjects

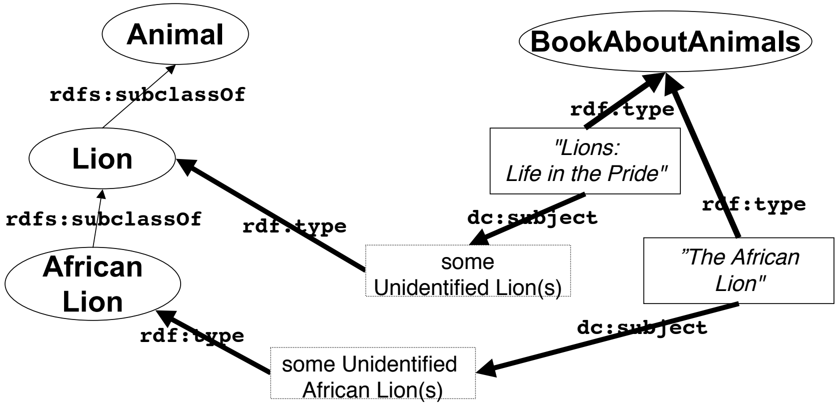


# Hierarchy of Subjects: Considerations

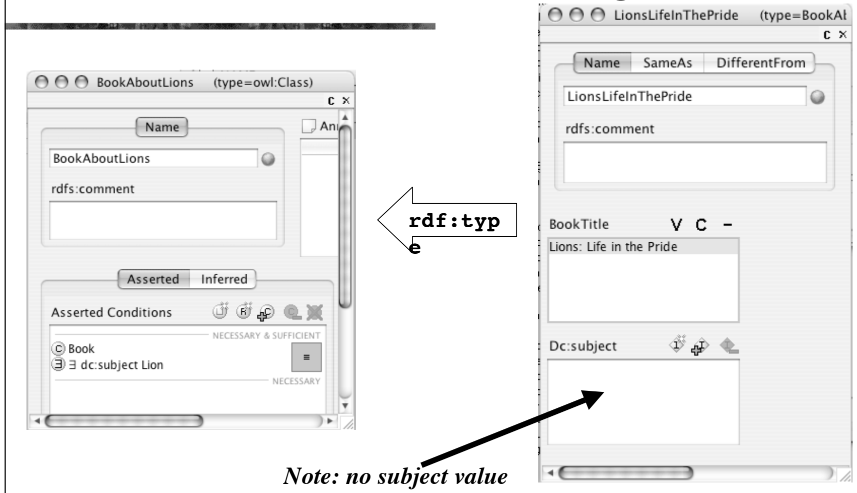
- Compatible with OWL DL
- Subject hierarchy (terminology) is independent of class hierarchy (rdfs:seeAlso)
- Maintenance penalty



# Using members of a class as values



## Representation in Protege



BookAboutLions

Book
∃ dc:subject Lion

LionsLifeInThePride

Lions: Life in the Pride

*Note: no subject value*

## Considerations

- Compatible with OWL DL
- Interpretation: the subject is one or more specific lions, rather than the Lion class
- Can use a DL reasoner to classify specific books

## Part VI – Patterns: Part-whole relations

- Upper ontologies & Domain ontologies
- Building from trees and untangling
- Using a classifier
- Closure axioms & Open World Reasoning
- Specifying Values
- n-ary relations
- Classes as values – using the ontology
- *Part-whole relations*

## Part-whole relations
*One method: NOT a SWBP draft*

- How to represent part-whole relations in OWL is a commonly asked question
- SWBP will put out a draft.
- This is one approach that will be proposed
  - It has been used in teaching
  - It has no official standing
  - It is presented for information only

## Part Whole relations

- OWL has no special constructs
  - But provides (some of) the building blocks
- Transitive relations
  - Finger is_part_of Hand
    Hand is_part_of Arm
    Arm is_part_of Body
    - →
  - Finger is_part_of Body

## Many kinds of part-whole relations

- Physical parts
  - hand-arm
- Geographic regions
  - Hiroshima - Japan
- Functional parts
  - cpu – computer

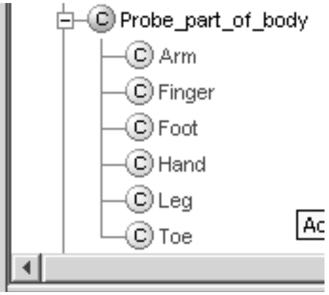
- See Winston & Odell
  Artale
  Rosse

## Simple version

- One property *is_part_of*
  - transitive
    - Finger is_part_of *some* Hand
      Hand is_part_of *some* Arm
      Arm is_part_of some Body

## Get a simple list

- Probe_part_of_body =
  Domain_category
  is_part_of some Body



- Logically correct
  - But may not be what we want to see

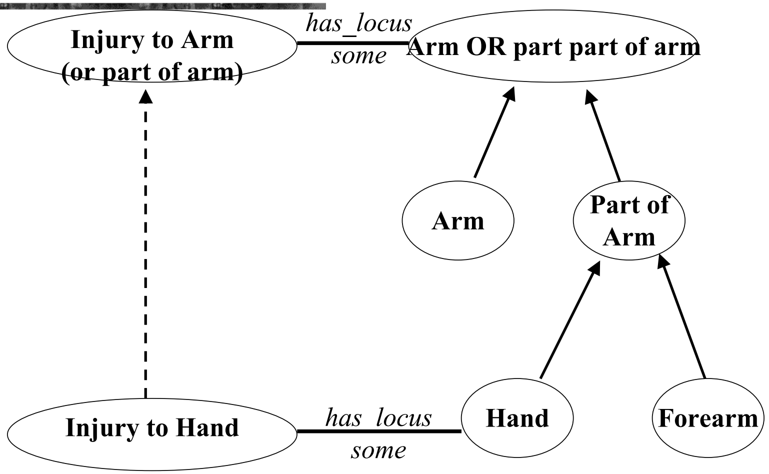
# Injuries, Faults, Diseases, Etc.

- A hand is not a *kind of* a body
  - … but an injury to a hand is a kind of injury to a body
- A motor is not a *kind of* automobile
  - … but a fault in the motor is a kind of fault in the automobile
- And people often expect to see partonomy hierarchies

# Being more precise: “Adapted SEP Triples”

- Body (‘as a whole’)
  - Body
- The Body’s parts
  - is_part_of *some* Body
- The Body and it’s parts
  - Body OR is_part_of *some* Body
- Repeat for all parts
  - Use ‘Clone class’ or
  - NB: ‘JOT’ Python plugin is good for this

# Adapted SEP triples: UML like view



Injury to Arm (or part of arm)

*has_locus some*

Arm OR part part of arm

Arm

Part of Arm

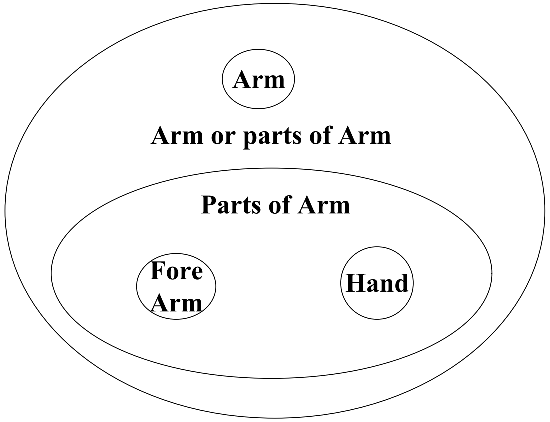
Injury to Hand

*has_locus some*

Hand

Forearm

# Adapted SEP triples: Venn style view



Arm

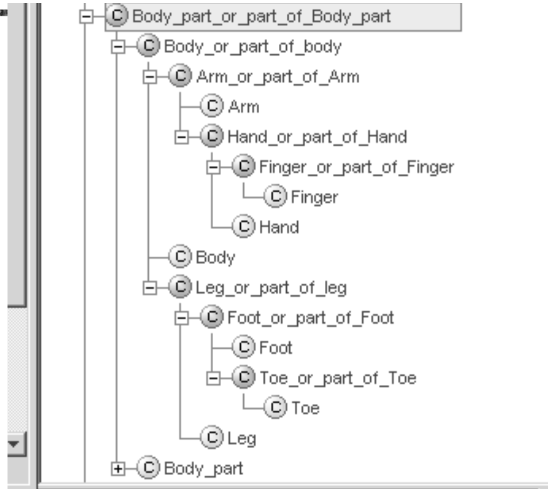
Arm or parts of Arm

Parts of Arm

Fore Arm

Hand

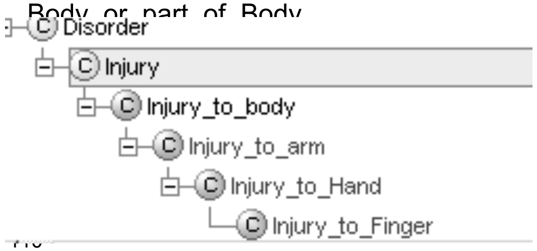
# Resulting classification: Ugly to look at, but correct



- Body_part_or_part_of_Body_part
  - Body_or_part_of_body
    - Arm_or_part_of_Arm
      - Arm
      - Hand_or_part_of_Hand
        - Finger_or_part_of_Finger
          - Finger
        - Hand
    - Body
    - Leg_or_part_of_leg
      - Foot_or_part_of_Foot
        - Foot
        - Toe_or_part_of_Toe
          - Toe
      - Leg
  - Body_part

# Using part-whole relations: Defining injuries or faults

- Injury_to_Hand =
  Injury has_locus *some*
  Hand_or_part_of_hand
- Injury_to_Arm =
  Injury has_locus *some* Arm_or_part_of_Arm
- Injury_to_Body =
  Injury has_locus *some*
  Body_or_part_of_Body



- Disorder
  - Injury
    - Injury_to_body
      - Injury_to_arm
        - Injury_to_Hand
          - Injury_to_Finger

- The expected hierarchy from point of view of anatomy

# Caution with part of

- Motor is_part_of *some* Car
  - Means "All motors are part of some car"
    - Obviously false!
  - But convenient to get:
    Car_part =
    is_part_of some Car
    subsumes
    Motor
  - To be correct must use
    "Car_motor =
    Motor *and* is_part_of *some* Car

# Geographical regions and individuals

- Similar representation possible for individuals but more difficult
  - and less well explored

# Simplified view: Geographical_regions

- Class: Geographical_region
  - Include countries, cities, provinces, …
    - A detailed ontology would break them down
- Geographical features
  - Include Hotels, Mountains, Islands, etc.
- Properties:
  - Geographical_region *is_subregion_of* Geographical_Region
  - Geographical_feature *has_location* Geographical_Region
  - Features located in subregions are located in the region.
    *is_subregion_of* is transitive

# Geographical regions & features are represented as individuals

- Japan, Honshu, Hiroshima, Hiroshima-ken,…
- Mt_Fuji, Hiroshima_Prince_Hotel, …

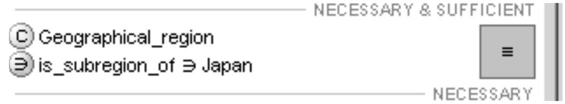
# Facts*

- Honshu is_subregion_of *hasValue* Japan
  Hiroshima-ken is_subregion_of *hasValue* Honshu
  Hiroshima is_subregion_of hasValue Hiroshima-ken
- Mt_Fuji has_location *hasValue* Honshu
  **Hiroshima_prince_hotel** has_location *hasValue* Hiroshima-ken

*with apologies for any errors in Japanese geography

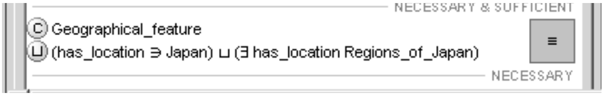
# Definitions

- Region_of_Japan =
  Geographical_region *AND*
  is_subregion_of *hasValue* Japan



NECESSARY & SUFFICIENT
Ⓒ Geographical_region
∋ is_subregion_of ∋ Japan
≡
NECESSARY

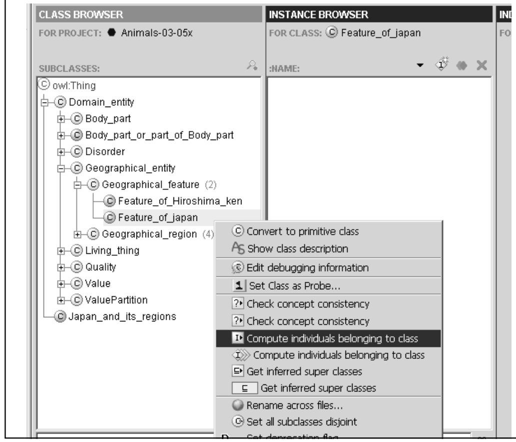
- Feature_of_Japan =
  Geographical_feature *AND*
  ( hasLocation *hasValue* Japan *OR*
  hasLocation *hasValue* Region_of_Japan )

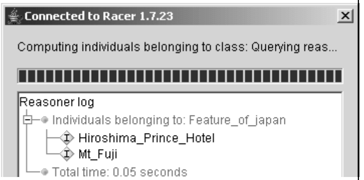


NECESSARY & SUFFICIENT
Ⓒ Geographical_feature
⊔ (has_location ∋ Japan) ⊔ (∃ has_location Regions_of_Japan)
≡
NECESSARY

# In tools at this time

- Must ask from right mouse button menu in Individuals tab


CLASS BROWSER
FOR PROJECT: ● Animals-03-05x
SUBCLASSES:
owl:Thing
Domain_entity
Body_part
Body_part_or_part_of_Body_part
Disorder
Geographical_entity
Geographical_feature (2)
Feature_of_Hiroshima_ken
Feature_of_japan
Geographical_region (4)
Living_thing
Quality
Value
ValuePartition
Japan_and_its_regions
INSTANCE BROWSER
FOR CLASS: Feature_of_japan
:NAME:
Convert to primitive class
Show class description
Edit debugging information
Set Class as Probe...
Check concept consistency
Check concept consistency
Compute individuals belonging to class
Compute individuals belonging to class
Get inferred super classes
Get inferred super classes
Rename across files...
Set all subclasses disjoint



Computing individuals belonging to class: Querying reas...

```
Reasoner log
  Individuals belonging to: Feature_of_japan
    Hiroshima_Prince_Hotel
    Mt_Fuji
  Total time: 0.05 seconds
```

- better integration under development

# Warning: Individuals and reasoners

- Individuals only partly implemented in reasoners
  - If results do not work, ask someone if they should!
    - **Open World reasoning with individuals is very difficult to implement**
  - If it doesn't work, try simulating individuals by classes
  - Large sets of individuals better in "Instance Stores", RDF triple stores, databases, etc that are restricted or closed world
- ***Ontologies are mainly about classes***
  - ***Ontologies are NOT databases***

# Part-whole in OWL

- Note - the only aspect of the part whole relation represented in OWL is transitivity
  - "Mereologists" (those who study parts-whole relations) define other axioms
    - Antisymmetry (nothing can be part of itself)
    - Reflexive (everything is a part of itself)
    - Weak supplementation principle -
      - When you take away a part (except the whole), you leave something behind

# Qualified cardinality constraints

- Use with partonomy
- Use with n-ary relations

# Cardinality Restrictions

- "All mammals have four limbs"
  - "All Persons have two legs and two arms"
  - "(All mammals have two forelimbs and two hind limbs)"

# What we would like to say: Qualified cardinality constraints

- Mammal
  has_part cardinality=4 Limb
- Mammal
  has_part cardinality = 2 Forelimb
  has_part cardinality = 2 Hindlimb
- Arm = Forelimb AND is_part_of some Person

**Glossary: "Forelimb" = front leg or arm**
**"Hindlimb" = back leg**

# What we have to say in OWL

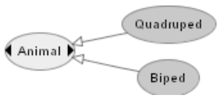
- The property *has_part* has subproperties:
  *has_limb*
  *has_leg*
  *has_arm*
  *has_wing*

| | | |
|---|---|---|
| Mammal, Reptile, Bird | *has_limb* | cardinality=4 |
| Person | *has_leg* | cardinality=2 |
| Cow, Dog, Pig… | *has_leg* | cardinality=4 |
| Bird | *has_leg* | cardinality=2 |

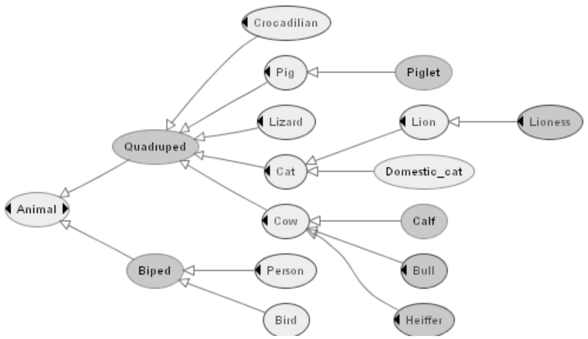
- Biped = Animal AND
  *has_leg* cardinality=2

# Classification of bipeds and quadrupeds

- Before classification



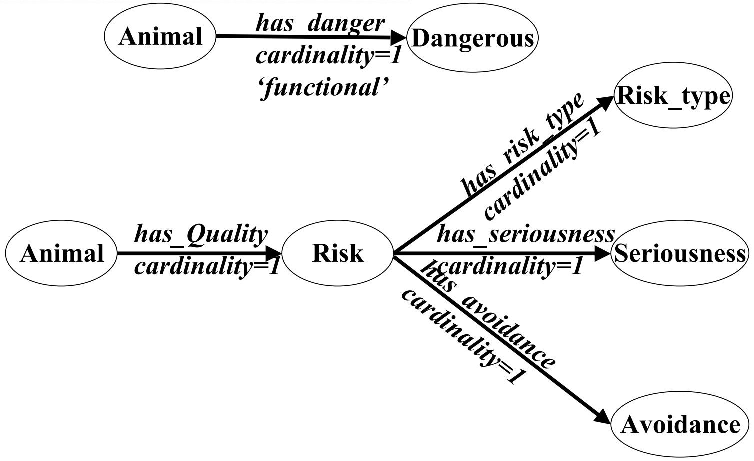
- After classificaiton



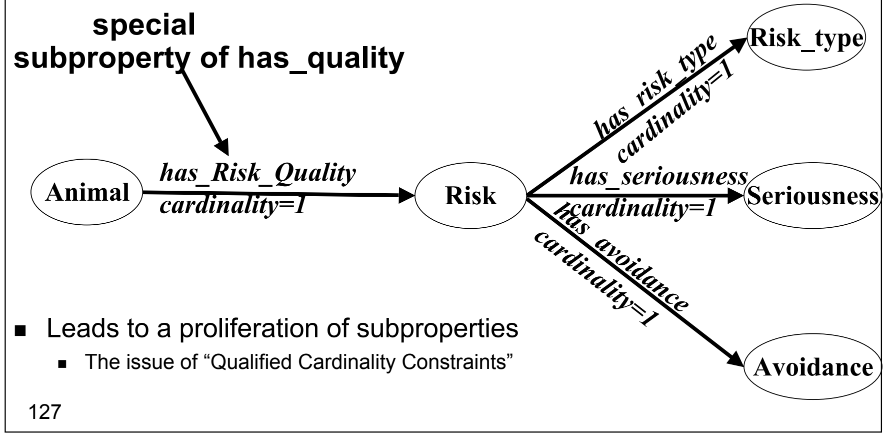
# Cardinality and n-ary relations

- Need to control cardinality of relations represented as classes
  - An animal can have just 1 “dangerousness”
    - Requires a special subproperty of quality:
      - has_dangerousness_quality cardinality=1

# Re-representing the property has_danger as the class Risk



# In OWL must add subproperty for each quality to control cardinality, e.g. *has_risk_quality*



- Leads to a proliferation of subproperties
  - The issue of “Qualified Cardinality Constraints”

# Part VII – Summary

- Upper ontologies & Domain ontologies
- Building from trees and untangling
- Using a classifier
- Closure axioms & Open World Reasoning
- Specifying Values
- n-ary relations
- Classes as values – using the ontology
- Part-whole relations
  - Transitive properties
  - Qualified cardinality restrictions

# End

- To find out more:
  - **http://www.co-ode.org**
    - Comprehensive tutorial and sample ontologiesxz
  - **http://protege.stanford.org**
    - Subscribe to mailing lists; participate in forums
  - On the SW in general: **semanticweb@yahoogroups.com**
  - For specific feedback to SWBP
    - Home & Mail Archive: **http://www.w3.org/2001/sw/BestPractices/ public-swbp-wg@w3.org**

# Part VI – Hands On supplement

- Open Animals-tutorial-step-2

# Exercise 3: (Advanced supplement)

- Load Animals-Tutorial-complete.pprj
- Define a new kind of Limb – Wing
- Describe birds as having 2 wings
- Define a Two-Winged_animal
- Does bird classify under Two-Winged_animal?