

## Classes

Web Ontology Language



Classes can be subclasses of other classes (subClassOf): all the instances of the subclass are also instances of the superclass (But no the other way around)





Classes are equivalent if the extent of their sets is exactly the same (equivalentTo): all the instances of A are also instances of B and the other way around



#### A hierachy can be built combining different subClassOf axioms





In order to define the qualities that the individuals of a class must hold to be members of that class, *restrictions* on the number and type of binary relations are used

Thus, the restrictions define the conditions that must be fulfilled to be a member of a given class

For example, we can state (In our ontology!) that in order to be human something must eat plants

Eating plants is a *necessary condition* to be human: all the humans eat plants, but there are other organisms that also eat plants that are not humans

We can also define a *necessary and sufficient* condition: producing language is a unique quality of humans: if we find an individual (Organism) capable of producing language we can infer that is human, since no other organism does it



Conditions are anonymous classes: the named class we are defining with such conditions can be a subclass (Necessary) or equivalent class (Necessary and sufficient) to the anonymous class The class human is a subclass (N) of the anonymous class comprised of the individuals that have at least one eats binary relation with an individual of the class plant

**OWL** semantics





The class human is equivalent (N+S) to the anonymous class comprised of the individuals that have at least on relation with the property produces with and individual of the class language



#### **Existential restriction**

owl:someValuesFrom: the anonymous class comprised of the individuals that, ammongst other things, have at least one relation to an individual of a given class with a given property: human subClassOf <u>eats some plant</u>





#### **Universal restriction**

owl:allValuesFrom: the anonymous class comprised of the individuals that, if having a relation with a given property, must be to an individual of a concrete class or *none*: human subClassOf <u>eats only organism</u>



### hasValue

the anonymous class comprised of the individuals that have a relation to a concrete individual human subClassOf <u>eats value pig-peggy</u>



Cardinal restrictions:

Min: human subClassOf eats min 1

Max: human subClassOf eats max 5

Exactly: human subClassOf eats exactly 3





QCR (Qualified Cardinality Constraint): Min: human subClassOf eats min 1 plant Max: human subClassOf eats max 5 plant Exactly: human subClassOf eats exactly 3 plant



We can state that a class is different to other class (They don't have any individual in common) using disjointFrom: human disjointFrom plant

We can state that two classes are the same (They have the same extent of individuals) using equivalentTo: human equivalentTo person

### **Booleans**

Not: human subClassOf not (eats some stone)

And (Intersection): man equivalentTo human and male

Or (Union): human equivalentTo woman or man





## Conditions can be very complex, combining different OWL elements

Class hierarchy: Hypothesis_MYB_AP1_IIIEIIC   Annotations: Hypothesis_MYB_AP1_UP   ECO_0000000   ECO_0000037   ECO_0000217   GO_0003674   GO_0005575   GO_0008150   Hypothesis_MYB_AP1_UP   MI_0001   MI_0002   MI_0003   MI_0116     Annotations: Hypothesis_MYB_AP1_UP	Class hierarchy Class hierarchy (inferred)	Annotations Usage
Image: Second constraints       Image: Second constraints         Image: Second constraints       Description: Hypothesis_MYB_AP1_UP         Image: Second constraints       Equivalent classes         Image: Second constraints       Image: Second constraints	Class hierarchy: Hypothesis_MYB_AP1_000000	Annotations: Hypothesis_MYB_AP1_UP
<ul> <li>Thing</li> <li>ECO_0000000</li> <li>ECO_0000037</li> <li>ECO_0000217</li> <li>GO_0003674</li> <li>GO_0005575</li> <li>GO_0008150</li> <li>Hypothesis_MYB_AP1_UP</li> <li>MI_0001</li> <li>MI_0002</li> <li>MI_0003</li> <li>MI_0116</li> <li>Description: Hypothesis_MYB_AP1_UP</li> </ul>	📽 😫 🕱	Appetations
<ul> <li>ECO_0000000</li> <li>ECO_0000037</li> <li>ECO_0000217</li> <li>GO_0003674</li> <li>GO_0005575</li> <li>GO_0008150</li> <li>GHypothesis_MYB_AP1_UP</li> <li>MI_0001</li> <li>MI_0002</li> <li>MI_0003</li> <li>MI_0116</li> </ul>	Thing ∎	Description: Hypothesis_MYB_AP1_UP
<ul> <li>ECO_0000217</li> <li>GO_0003674</li> <li>GO_0005575</li> <li>GO_0008150</li> <li>Hypothesis_MYB_AP1_UP</li> <li>MI_0001</li> <li>MI_0002</li> <li>MI_0003</li> <li>MI_0116</li> <li>transcription_factor exactly 1 (PRO_000009232 and (located_in_cellular_component some ((ECO_0000033 and GO_0005654) or (GO_0000790 and (evidence_code some ECO_0000203)))))</li> </ul>	ECO_0000000	Equivalent classes 🕒
GO_0003674 GO_0005575 GO_0008150 MI_0001 MI_0002 MI_0003 MI_0003 MI_0003 MI_0003 MI_0003 MI_0003 MI_0003 MI_0003 MI_0016 MI_0003 MI_00000 MI_00000 MI_000000 MI_00000	ECO 0000217	transcription_factor exactly 1 (PRO_000009232
GO_0005575       ((ECO_0000033         GO_0008150       and GO_0005654)         Hypothesis_MYB_AP1_UP       or (GO_0000790         MI_0001       and (evidence_code some ECO_0000203)))))         MI_0002       target_gene exactly 1 (PRO_000010799         MI_0116       (MI_0931	► • GO 0003674	and (located_in_cellular_component some
GO_0008150       and GO_0005654)         Hypothesis_MYB_AP1_UP       or (GO_0000790         MI_0001       and (evidence_code some ECO_0000203))))         MI_0002       etarget_gene exactly 1 (PRO_000010799)         MI_0116       (MI_0931)	► • GO 0005575	((ECO_0000033
<ul> <li>Hypothesis_MYB_AP1_UP</li> <li>MI_0001</li> <li>MI_0002</li> <li>MI_0003</li> <li>MI_0116</li> <li>Or (GO_0000790 and (evidence_code some ECO_0000203))))</li> <li>I (PRO_000010799 and (participates_in some (MI_0931)</li> </ul>	► • GO 0008150	and GO_0005654)
<ul> <li>MI_0001</li> <li>MI_0002</li> <li>MI_0003</li> <li>MI_0116</li> <li>and (evidence_code some ECO_0000203))))</li> <li>target_gene exactly 1 (PRO_000010799 and (participates_in some (MI_0931)</li> </ul>	Hypothesis_MYB_AP1_UP	or (GO_0000790
MI_0002         etarget_gene exactly 1 (PRO_000010799           MI_0003         and (participates_in some           MI_0116         (MI_0931	▶ ● MI_0001	and (evidence_code some ECO_0000203)))))
MI_0003 MI_0116 and (participates_in some) (MI_0931	▶ ● MI_0002	target_gene exactly 1 (PRO_000010799
MI_0116 (MI_0931	▶ ●MI_0003	and (participates_in some
	▶ ●MI_0116	(MI_0931
MI_0190 and (detected_by some MI_0438)	▶ ●MI_0190	and (detected_by some MI_0438)
MI_0300 and (has_participant only PRO_000009232)))	▶ ● MI_0300	and (has_participant only PRO_000009232))))
MI_0313	▶ ●MI_0313	hypothesis entity only
► ● MI_0333 (PRO_00009232	▶ ● MI_0333	(PRO 000009232
► ● MI_0346 or PRO 000010799)	▶ ●MI_0346	or PRO 000010799)
Pregulation some UD	▶ ●MI_0353	eregulation some UD
	MI_0444	



# **Object Properties**



## Property hierarchy:

### Sub/SuperProperties

If Tom talks to Martin, he interacts with him If Tom interacts with Michael, he doesn't talk to him (he might!)

### **Equivalent Properties**

**Disjoint Properties** 







## **Inverse** properties







#### Domain and Range:

Usually classes or class unions

But any anonymous expression class can be used

They are not constraints, they are axioms





## Individuals





An individual can be a member of one or more anonymous or named classes (Types)

An individual can be the same as other individual (SameAs)

An individual can be different from another individual (DifferentFrom)

Individuals can be related in binary relations (Object Properties):

my\_wheel part\_of my\_car my\_wheel not part\_of your\_car

Individuals can be related with data (Data Type properties):

my\_car has\_power "90"^^xsd:positiveInteger my\_car not has\_power "90"^^xsd:positiveInteger