Combining classes, property restrictions and intensional classes (Optional further reading)

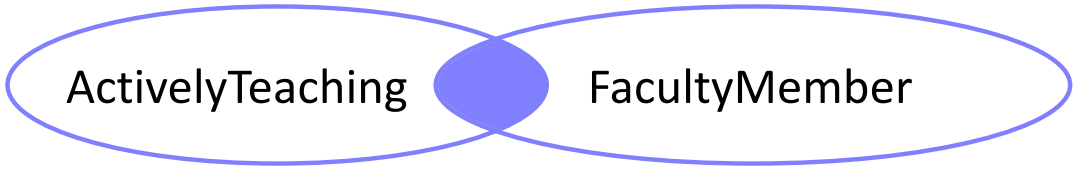
Class Union and Intersection, Enumeration, Complement

- Allow combining classes

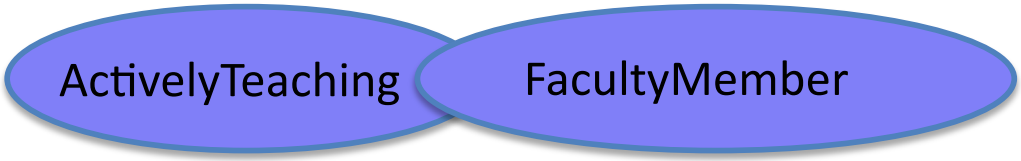
OWL notation	FOL translation
owl:intersectionOf $(C, D...)$	$C(X) \wedge D(X)...$
owl:unionOf $(C, D...)$	$C(X) \vee D(X)...$
owl:oneOf $(e, f...)$	$X = e \vee X = f...$
owl:complementOf(C)	$\neg C(X)$

Examples

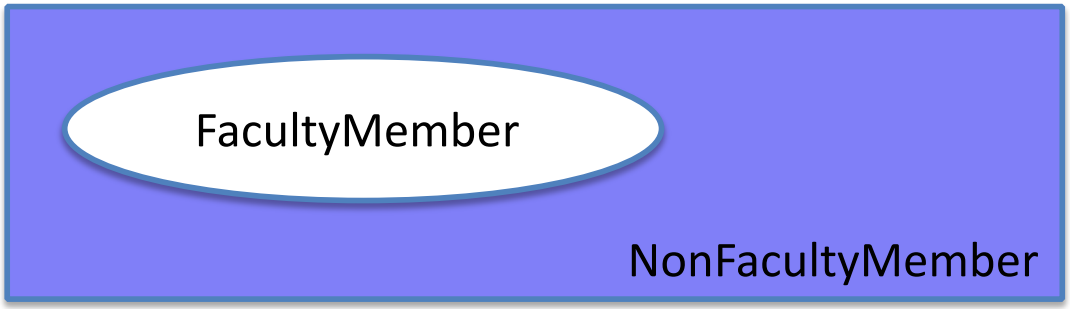
_:X owl:intersectionOf (ex:ActivelyTeaching ex:FacultyMember) .



_:X owl:unionOf (ex:ActivelyTeaching ex:FacultyMember) .



_:X owl:complementOf (ex:FacultyMember) .



Unnamed Classes

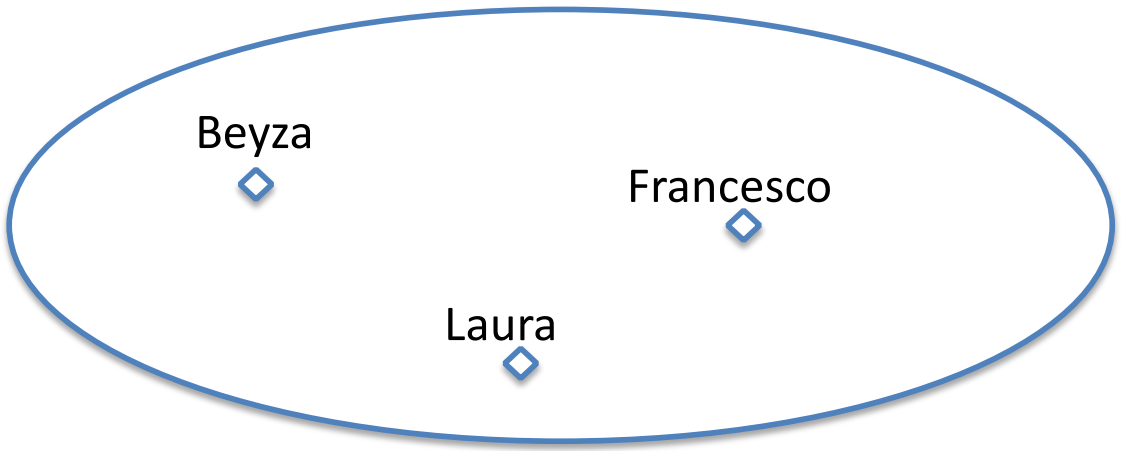
- The combination of classes with logical operators describes an anonymous (unnamed) class
- Other classes can be a declared to be a subclass or equivalent to this anonymous class

```
ex:NonFacultyMember rdfs:subClassOf _:X .
_:X rdf:type owl:Class .
_:X owl:complementOf ex:FacultyMember .
```

```
ex:TeachingFaculty owl:equivalentClass [
rdf:type owl:Class ;
owl:intersectionOf (ex:ActivelyTeaching, ex:FacultyMember) ].
```

Closed Classes

```
:PhDStudents owl:equivalentClass [
rdf:type owl:Class ;
owl:oneOf ( :Beyza :Laura :Francesco) ].
```



The OWL View of Life

- OWL is not like a database system
- no requirement that the only properties of an individual are those mentioned in a class it belongs to
- no assumption that everything is known
- classes and properties can have multiple “definitions”
- statements about individuals need not be together (in the same document)

Property Restrictions and Definition of Intensional Classes

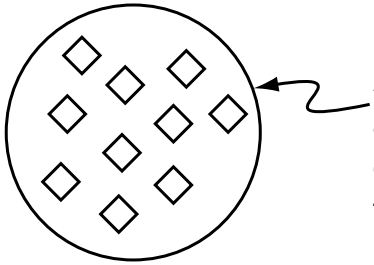
- Goal: allow expressing complex constraints such as:
  - departments can be lead only by professors
  - only professors or lecturers may teach to undergraduate students.
- The keyword `owl:Restriction` is used in association with a blank node class, and some specific restriction properties:
  - `owl:someValuesFrom`
  - `owl:allValuesFrom`
  - `owl:minCardinality`
  - `owl:maxCardinality`

  `owl:hasValue`

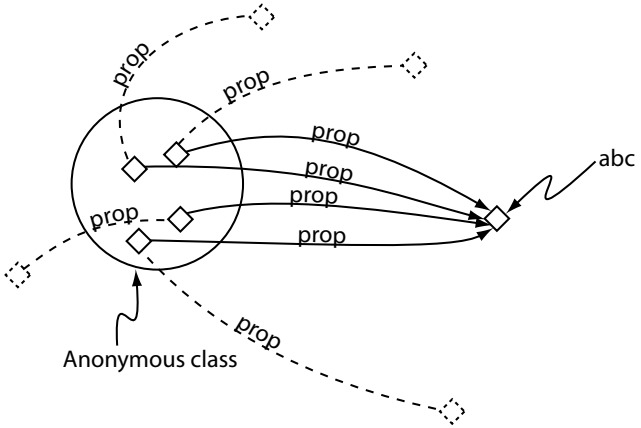
Property Restrictions

OWL notation	FOL translation
_:a owl:onProperty P _:a owl:allValuesFrom C	$\forall Y\,(P(X,Y) \Rightarrow C(Y))$
_:a owl:onProperty P _:a owl:someValuesFrom C	$\exists Y\,(P(X,Y) \wedge C(Y))$
_:a owl:onProperty P _:a owl:minCardinality n	$\exists Y_1 \ldots \exists Y_n (P(X,Y_1) \wedge \ldots \wedge P(X,Y_n) \wedge \bigwedge_{i,j \in [1..n], i \neq j} (Y_i \neq Y_j))$
_:a owl:maxCardinality n	$\forall Y_1 \ldots \forall Y_n \forall Y_{n+1}$ $(P(X,Y_1) \wedge \ldots \wedge P(X,Y_n) \wedge P(X,Y_{n+1}) \Rightarrow \bigvee_{i,j \in [1..n+1], i \neq j} (Y_i = Y_j))$
_:a owl:onProperty P _:a owl:hasValue v	P(X,v)

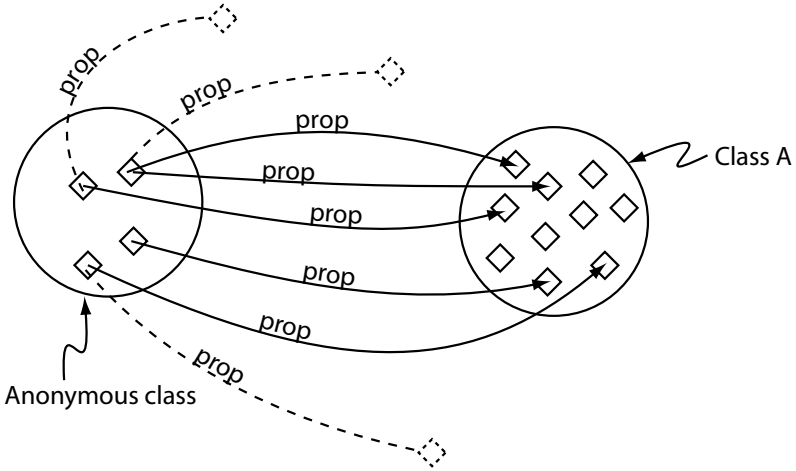
Restriction Classes



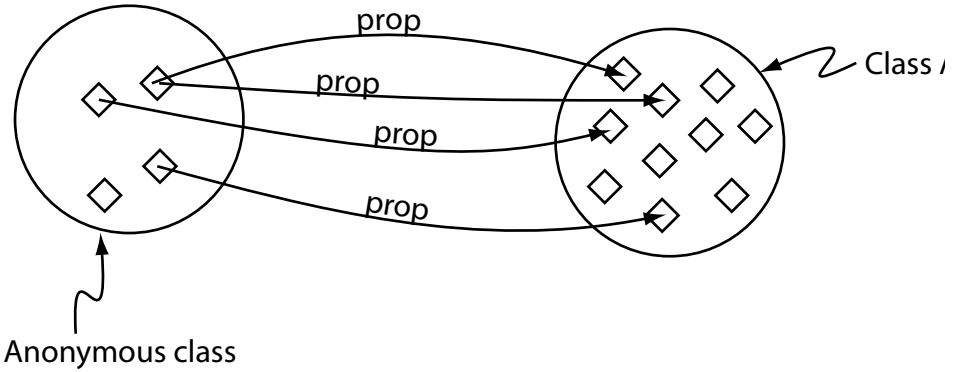
A set of individuals that satisfy a restriction - the restriction essentially describes an anonymous (unnamed) class that contains these individuals.



prop hasValue abc



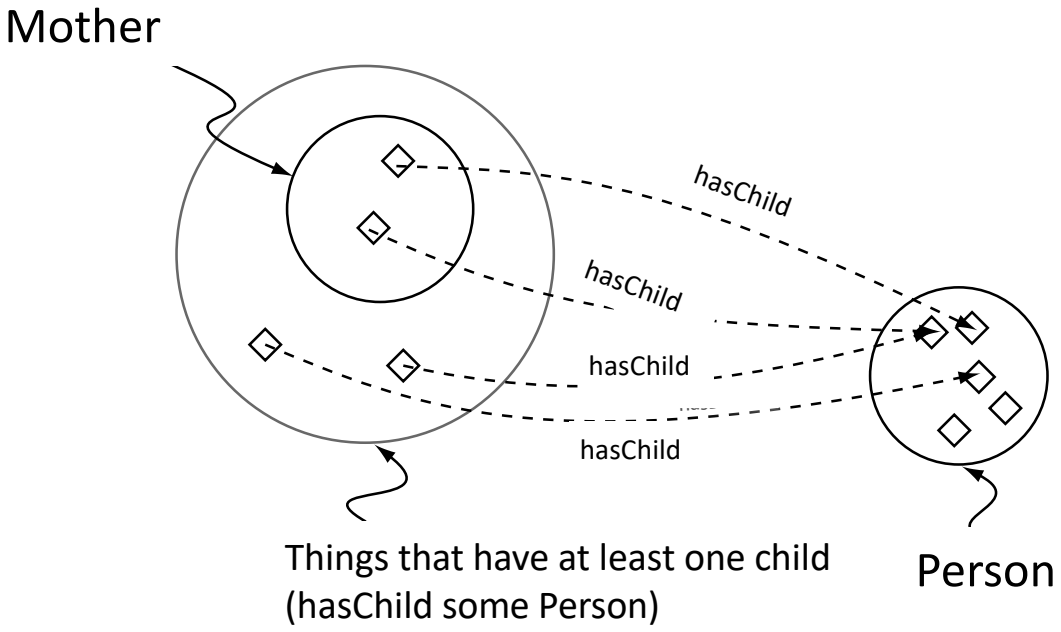
prop someValuesFrom A



prop allValuesFrom A

Property Restriction

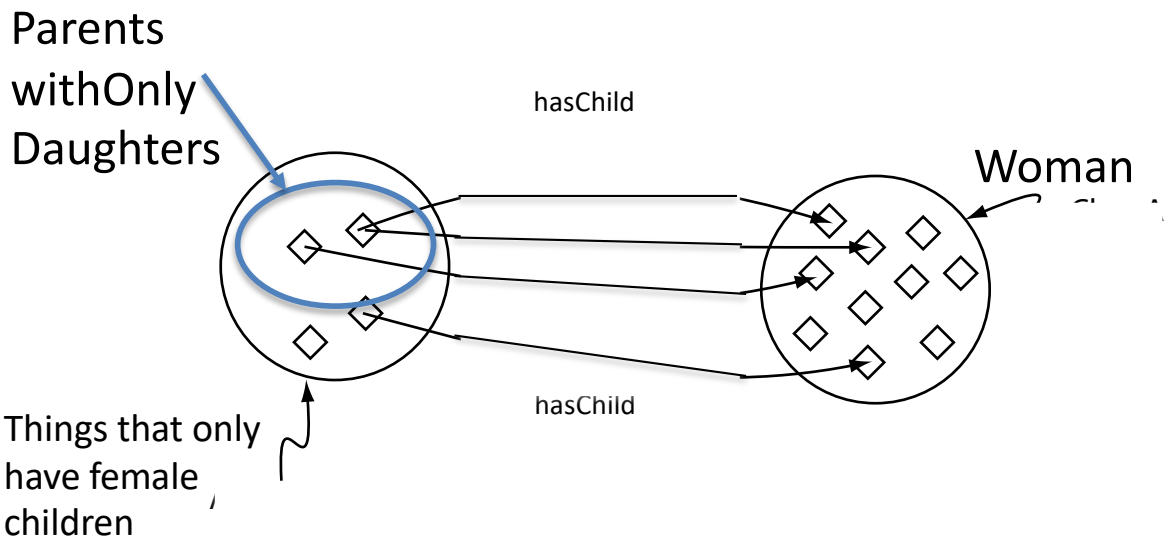
- A Mother is a Woman that has a child (some Person)
  - Set of objects that have a child (some Person)
  - Mother is a subclass




```
_:a rdfs:subclassOf owl:Restriction .
_:a owl:onProperty ex:hasChild .
_:a owl:someValuesFrom ex:Person .
ex:Mother rdfs:subclassOf _:a
```

Property Restriction

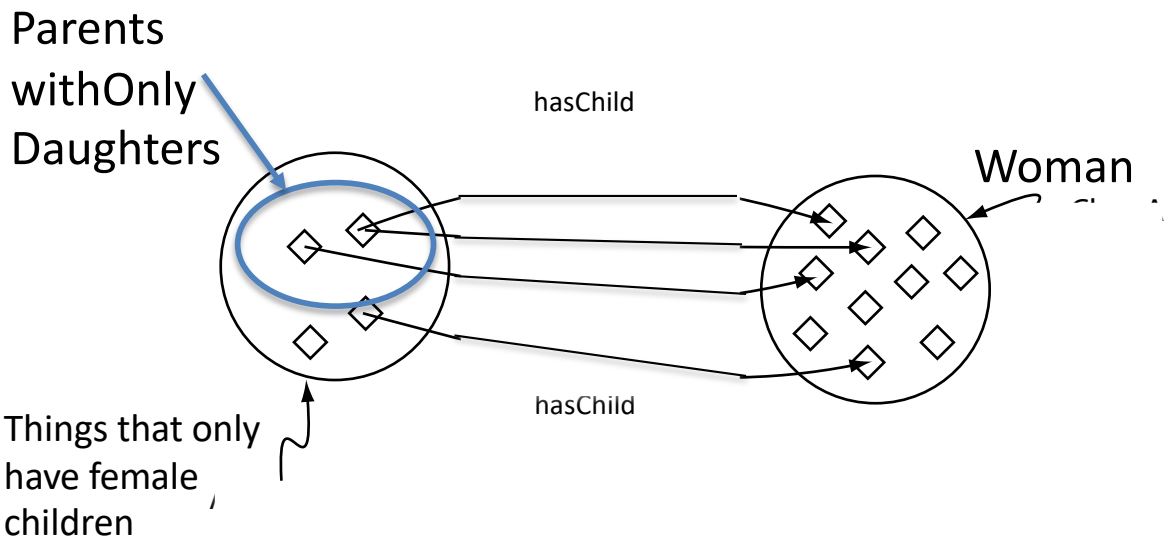
- Set of parents that only have female children (daughters)




```
_:a rdfs:subclass owl:Restriction .
_:a owl:onProperty ex:hasChild .
_:a owl:allValuesFrom ex:Woman .
ex:ParentwithOnlyDaughters rdfs:subclassOf _:a
```

Property Restriction

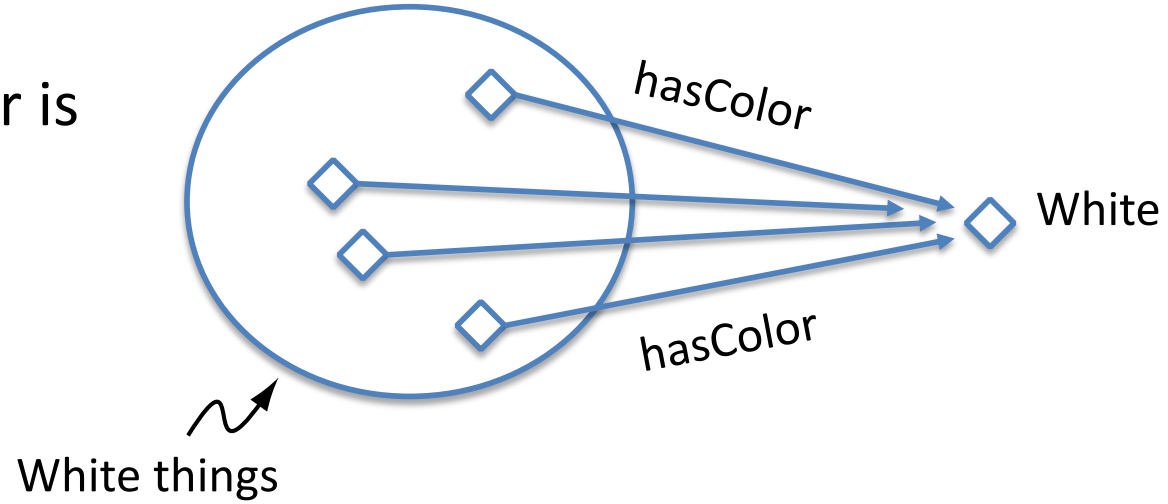
- Set of parents that only have female children (daughters)




```
_:a rdfs:subclass owl:Restriction .
_:a owl:onProperty ex:hasChild .
_:a owl:allValuesFrom ex:Woman .
ex:ParentwithOnlyDaughters rdfs:subclassOf _:a
```

Property Restriction

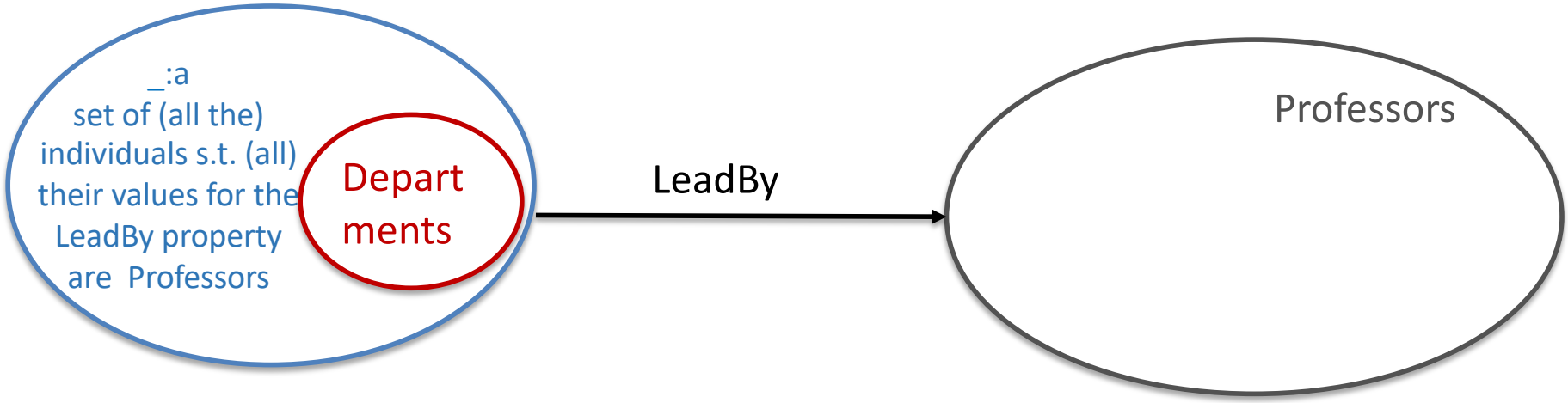
- Set of wines with color is white




```
_:a rdfs:subclass owl:Restriction .
_:a owl:onProperty ex:hasColor .
_:a owl:hasValue ex:White .
ex:WhiteWine rdfs:subclassOf _:a
```

Example

- Departments can be lead only by Professors



Example

- Departments can be lead only by professors

- Define the set of objects that are lead by professors

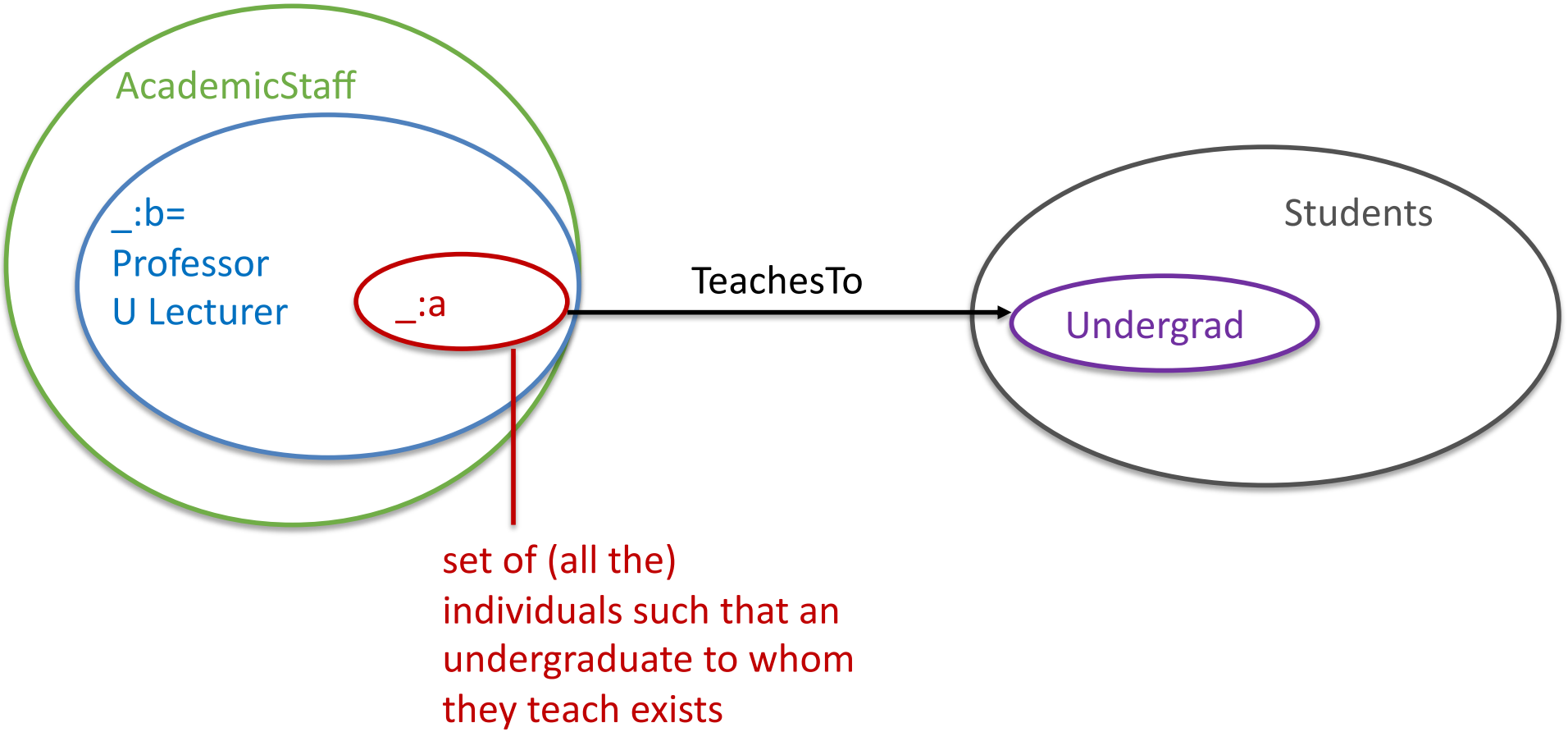
```
_a  rdfs:subClassOf    owl:Restriction
_a  owl:onProperty     LeadBy
_a  owl:allValuesFrom  Professor
```

- Now specify that all departments are lead by professors

```
Department  rdfs:subClassOf  _a
```

Example

- Only professors or lecturers can teach to undergraduate students



Example

- only professors or lecturers may teach to undergraduate students

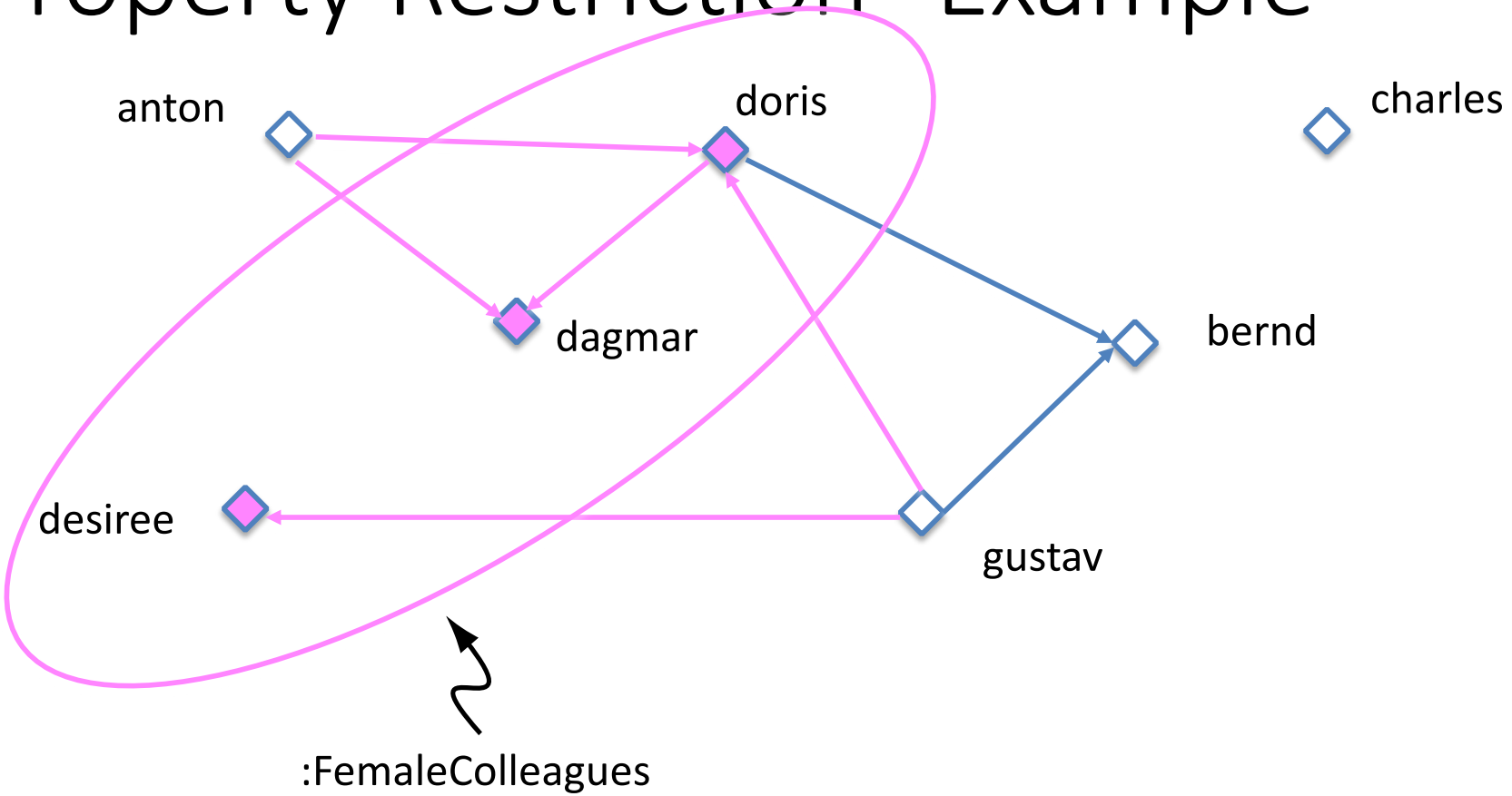
```
_a  rdfs:subClassOf     owl:Restriction
_a  owl:onProperty      TeachesTo
_a  owl:someValuesFrom  Undergrad
_b  owl:unionOf         (Professor, Lecturer)
_a  rdfs:subClassOf     _b
```

Property Restriction - Example

:anton a :Person;
:likesToWorkWith :doris, :dagmar .
:doris a :Person;
:likesToWorkWith :dagmar, :bernd .
:gustav a :Person;
likesToWorkWith :bernd, :doris, :desiree .
:charles a :Person .
:FemaleColleagues owl:equivalentClass [
rdf:type owl:Class ;
owl:oneOf (:dagmar :doris :desiree)].

+ :anton owl:differentFrom :doris .
… for all the person pairs …

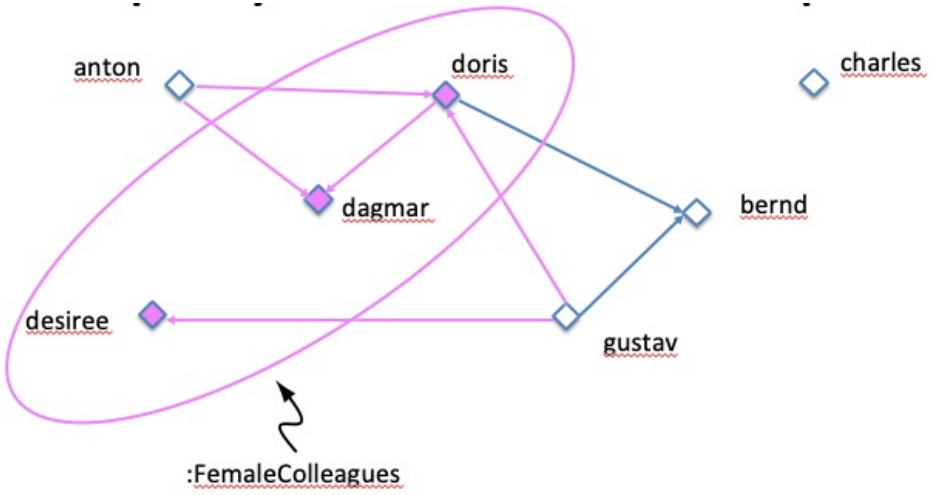
Property Restriction- Example



- All the properties are :likesToWorkWith properties

Property Restriction- Example

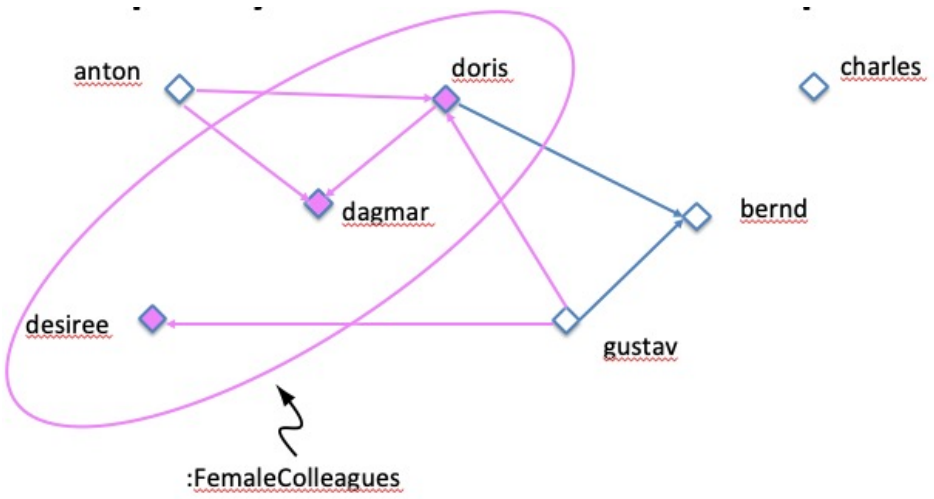
```
_:a1 rdfs:subclass owl:Restriction .
_:a1 owl:onProperty :likesToWorkWith .
_:a1 owl:someValuesFrom :FemaleColleagues
.
:Class1 owl:equivalentClass _:a1
```



anton
gustav
doris
are instances of Class1

Property Restriction- Example

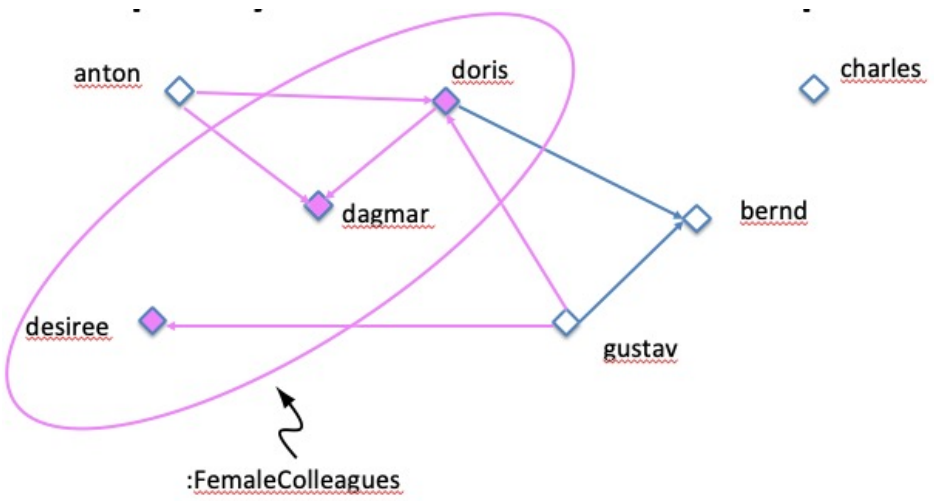
```
_:a2 rdfs:subclass owl:Restriction .
_:a2 owl:onProperty :likesToWorkWith .
_:a2 owl:allValuesFrom :FemaleColleagues .
:Class2 owl:equivalentClass _:a2
```



doris and gustav are NOT instances of Class2

Property Restriction- Example

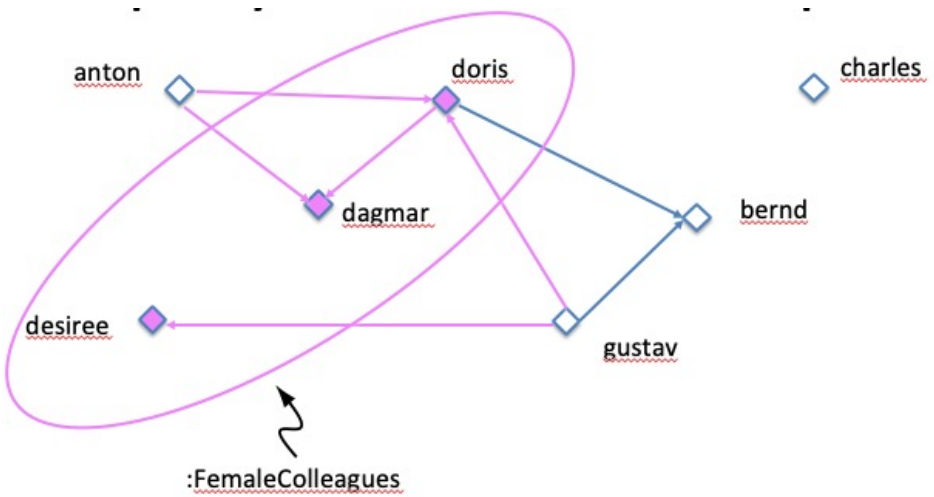
```
_:a3 rdfs:subclass owl:Restriction .
_:a3 owl:onProperty :likesToWorkWith .
_:a3 owl:hasValue :doris .
:Class3 owl:equivalentClass _:a3
```



anton and gustav are instances of Class2

Property Restriction- Example

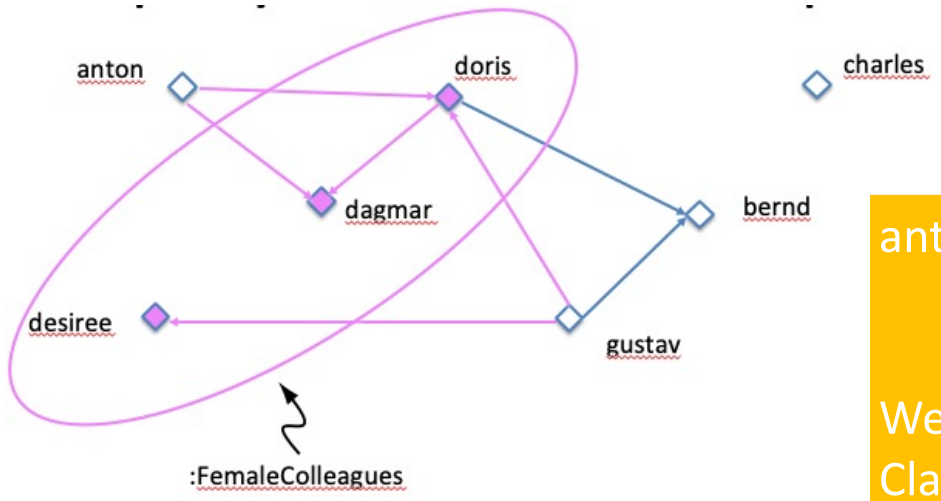
```
_:a4 rdfs:subclass owl:Restriction .
_:a4 owl:onProperty :likesToWorkWith .
_:a4 owl:minCardinality

     "3"^^xsd:nonNegativeInteger .
:Class4 owl:equivalentClass _:a4
```



gustav is an instances of Class2

Property Restriction - Example

```
_:a5 rdfs:subclass owl:Restriction .
_:a5 owl:onProperty :likesToWorkWith .
_:a5 owl:maxCardinality
     "0"^^xsd:nonNegativeInteger .
:Class5 owl:equivalentClass _:a5
```



anton, doris, gustav are NOT part of Class5

We cannot infer that charles belongs to Class5 because of OWA

Cardinality Restrictions and UNA

- OWL does not use the Unique Name Assumption (UNA)
  - This means that different names may refer to the same individual
- Cardinality restrictions rely on 'counting' distinct individuals
- Consider class6 identical to class5 but with maxcardinality 2
- If we add anton to class6
  - Since we are stating that the individual anton is a member of the class of individuals that work with at the most two other individuals (people) doris and dagmar are inferred to be the same individual
- Rather than being viewed as an error, it will be inferred that two of the names refer to the same individual

Some basic modelling guidelines

- X must be Y, X is an Y that...

$$X \subseteq Y$$

X rdfs:subClassOf Y

- X is exactly Y, X is the Y that...

$$X \equiv Y$$

X owl:equivalentClass Y

- X is not Y

$$X \subseteq \neg Y$$

X owl:complementOf Y

(*not the same as X is whatever it is not Y*)

- X and Y are disjoint

$$X \cap Y \subseteq \bot$$

X owl:disjointWith Y

- X is Y or Z

$$X \subseteq Y \cup Z$$

X rdfs:subclass [
rdf:type owl:Class ;
owl:unionOf (Y,Z)]

Some basic DL modelling guidelines

- X is Y for which property P has only instances of Z as values

$$X \subseteq Y \cap (\forall P.Z)$$

```
_:a rdfs:subclassOf owl:Restriction .
_:a owl:onProperty P .
_:a owl:allValuesFrom Z .
X rdfs:subclassOf [
   rdf:type owl:Class ;
   owl:intersectionOf (Y,_:a) ]
```

- X is Y for which property P has at least an instance of Z as a value

$$X \subseteq Y \cap (\exists P.Z)$$

- X is Y for which property P has at most 2 values

$$X \subseteq Y \cap (\leq 2.P)$$

- Individual X is a Y

$$X \in Y$$

```
X rdf:type Y
```

```
X a Y
```

)WL: Semantics and Reasoning

OWL Semantics

- Most of the OWL constructs come from Description Logics (DL)
- Therefore, we get for free all the positive and negative known results about reasoning in DLs

Ontologies and Description Logics

- First-order logic (FOL) is the formal foundation of the ontology languages for the Web
- First-order logic (also called predicate logic) is especially appropriate for knowledge representation and reasoning
  - ontologies are simply knowledge about classes and properties
  - from a logical point of view
    - classes are unary predicates
    - properties are binary predicates
    - constraints are logical formulas asserted as axioms on these predicates, i.e., asserted as true in the domain of interest

Ontologies and Description Logics

- Unfortunately, the implication problem in FOL is not decidable but only recursively enumerable:
  - an algorithm exists that given some formula F enumerates all the formulas G such that F implies G
  - no general algorithm exists that, when applied to two any input FOL formulas F and G, decides whether F implies G

Ontologies and Description Logics

- The simpler problem of deciding whether a FOL formula is satisfiable (i.e., there exists an assignment to the variables which make the formula true in the domain of interest) is also not decidable
- Said in other terms
  - an algorithm which takes a FOL formula F and returns yes if it is satisfiable and not if it not satisfiable does not exist

Ontologies and Description Logics

- **Issues**: exhibit fragments of FOL that are decidable, i.e., subsets of FOL formulas defined by some restrictions on the allowed formulas, for which checking logical entailment between formulas, possibly given a set of axioms, can be automatically performed by an algorithm

- Description logics (DLs) are decidable fragments of first-order logic allowing reasoning on complex logical axioms over unary and binary predicates
  - Exactly what is needed for handling ontologies

Ontologies and Description Logics

- DLs computational complexity depends on the set of constructs allowed in the language
- Research carried out on DLs provides a fine-grained analysis of the trade-off between expressive power and computational complexity of sound and complete reasoning

- The semantics of RDF, RDFS, OWL can be described in terms of DL

Ontologies in Description Logic Notation

- Classes and Instances
  - $C(x)$ ↔ `x a C .`
  - $R(x,y)$ ↔ `x R y .`
  - $C \sqsubseteq D$ ↔ `C rdfs:subClassOf D`
  - $C \equiv D$ ↔ `C owl:equivalentClass D`
  - $C \sqsubseteq \neg D$ ↔ `C owl:disjointWith D`
  - $C \equiv \neg D$ ↔ `C owl:complementOf D`
  - $C \equiv D \sqcap E$ ↔ `C owl:intersectionOf (D E) .`
  - $C \equiv D \sqcup E$ ↔ `C owl:unionOf (D E) .`
  - $\top$ ↔ `owl:Thing`
  - $\bot$ ↔ `owl:Nothing`

Ontologies in Description Logic Notation

- Domains, ranges, and restrictions
  - $\exists R.T \sqsubseteq C \leftrightarrow$ `R rdfs:domain C .`
  - $\forall R.C \leftrightarrow$ `R rdfs:range C .`
    $C \sqsubseteq \forall R.D \leftrightarrow$ `C owl:subClassOf [ a owl:Restriction; owl:onProperty R; owl:allValuesFrom D ] .`
  - $C \sqsubseteq \exists R.D \leftrightarrow$ `C owl:subClassOf [ a owl:Restriction; owl:onProperty R; owl:someValuesFrom D ] .`
  - $C \sqsubseteq \geq nR \leftrightarrow$ `C owl:subClassOf [ a owl:Restriction; owl:onProperty R; owl:minCardinality n ] .`

Basic Inference Tasks

- Inconsistency of one class – Class has to be empty?
  $C \equiv \bot$
- Subsumption – Is C a subclass of D?
  $C \sqsubseteq D$
- Equivalence of classes – Are two classes the same?
  $C \equiv D$
- Disjointness – Are two classes disjoint?
  $C \sqcap D \equiv \bot$
- Class membership – Is a an instance of C?
  $C(a)$

OWL – Reasoning Tasks Revisited

- Proof method: Reductio ad absurdum
  - "Invent" an instance i
  - Check for contradictions

- E.g. Subclass Relations
  - Student subclass of Person „Every student is a person“
  - Define Student(i) and ¬Person(i)
  - Check for contradictions
  - If there is one: Student subclass of Person has to hold
  - If there is none: Student subclass of Person cannot be derived
    - Note: it may still hold!

- All reasoning tasks can be reduced to the same basic tasks
  - i.e., consistency checking

- Tableaux algorithm

Tableaux Reasoning

- A tableaux reasoner shows the inconsistency (unsatisfiability) of a knowledge base (KB)
- Tableaux reasoners contain rules for handling all logical connectives and quantifiers
- Some rules cause the tableau to branch (divide into two alternatives)
- If any branch of a tableau leads to an evident contradiction, the branch closes
- If all branches close, the proof is complete, the KB is inconsistent (unsatisfiable)
- Else, if the algorithm finishes when no more rules can be applied, then the KB is consistent (satisfiable)

Inference Tasks as Unsatisfiability Problems

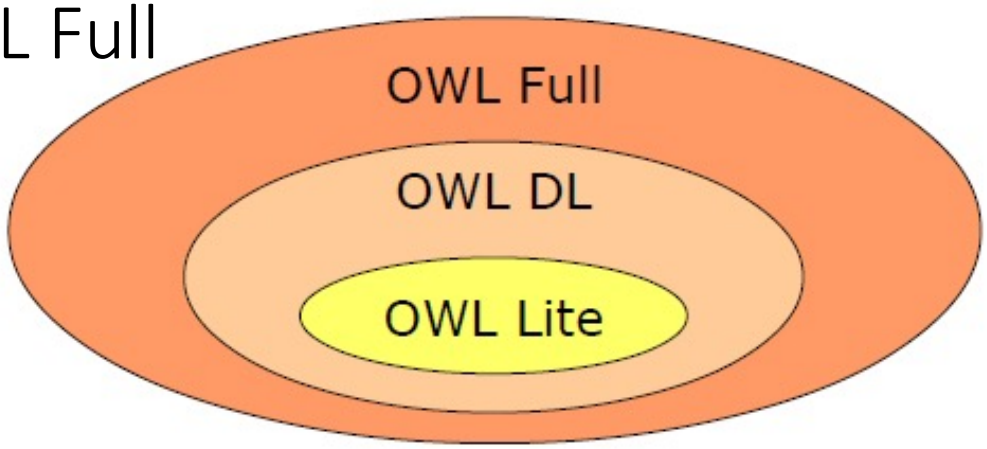
- Inconsistency of one class – Class has to be empty?
  $C \equiv \bot$ iff $KB \cup \{C(a)\}$ unsatisfiable (a is a new individual)
- Subsumption – Is C a subclass of D?
  $C \sqsubseteq D$ iff $KB \cup \{(C \sqcap \neg D)(a)\}$ unsatisfiable (a new)
- Equivalence of classes – Are two classes the same?
  $C \equiv D$ iff $C \sqsubseteq D$ and $D \sqsubseteq C$
- Disjointness – Are two classes disjoint?
  $C \sqcap D \equiv \bot$ iff $KB \cup \{(C \sqcap D)(a)\}$ unsatisfiable (a new)
- Class membership – Is a an instance of C?
  $C(a)$ iff $KB \cup \{\neg C(a)\}$ unsatisfiable

Tableaux Reasoners and Inference Tasks

- Our objective is to check whether a statement is true.
- To do this, we prove that the "reverse" of the statement cannot be satisfied.
- This means we take the KB we have and input the negation of the statement we want to prove.
- If the tableau ends with an inconsistent KB, the proof is complete and the original statement is true.
- If the tableau ends with a consistent KB, we have not been able to prove the truth of the statement.

OWL – Why so many variants?

- More expressive/powerful than RDF schema
- Trade-off:
  - –Expressive power
  - –Complexity of reasoning
  - –Decidability
- Solution: different variants of OWL, e.g.,
  - –OWL Lite, OWL DL, OWL Full
  - - OWL2 profiles



OWL Full

OWL 2 Full

- All OWL features added on top of RDF(S)
  - *Allows, e.g., redefining the meaning of RDF(S) and OWL primitives*
- Advantages
  - *Fully upward compatible with RDF*
    - Any RDF document is an OWL 2 Full document
    - Any RDF(S) conclusion is an OWL 2 Full conclusion
  - *RDF-based semantics*
- Disadvantages
  - *Undecidable, as RDFS already has some very powerful modeling primitives*
  - *Complete and efficient reasoning not possible*

OWL DL

OWL 2 DL (Description Logic)

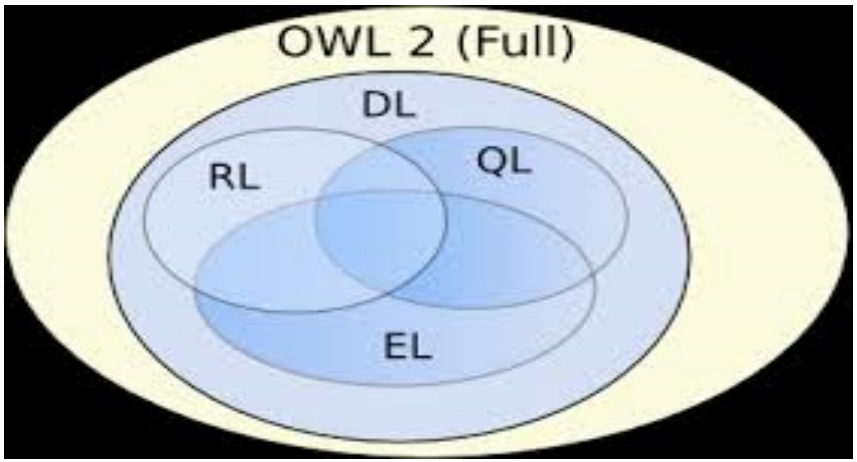
- Restricted form of OWL Full for which decidable, efficient support for reasoning is possible
  - *OWL 2 primitives cannot be applied to themselves*
  - *Only classes of non-literal resources considered*
  - *Strict separation between datatype and object properties*
  - *Strict separation between an individual, a class, or a property*
    - Using ”punning” the same name may be used for different purposes, but treated as different views on the same IRI, interpreted semantically as if they were distinct
- Direct semantics, based on Description logics (Terminology logics)
  - *Subsets of predicate logic*
  - *But also RDF-based semantics can be applied to OWL 2 DL ontologies*
- Reasoning engines are available for DL
  - *Pellet, FaCT, RACER, HermiT*

OWL 2 Profiles

OWL 2 DL includes three specific profiles for different use cases

- OWL 2 EL
- OWL 2 QL
- OWL 2 RL

Each profile includes a subset of OWL DL features



OWL 2 EL

- Good for ontologies with lots of classes and/or properties
- Polynomial complexity of standard inference types: satisfiability, classification, instance checking
- Used for large scale class ontologies, e.g., Snomed CT

- Limitations include:
  - *Negation and disjunction not supported*
  - *Universal quantification on properties*
    - *E.g., "all children of a rich person are rich" cannot be stated*
  - *All kinds of role inverses are not available*
    - *E.g., parentOf and childOf cannot be stated as inverses*

OWL 2 QL

- Good for querying large numbers of individuals
- Relational Query Languages (conjunctive queries)
  - *Can be implemented efficiently using relational databases*

- Limitations include:
  - *Existential quantification of roles to a class expression*
    - *E.g., it can be stated that every person has a parent, but not that every person has a female parent*
  - *Property chain axioms and equality are not supported*

OWL 2 RL (Rule Language)

- Good for rule-based reasoning, database focus
- Can be implemented using logic programming
  - *First-order implications: IF certain triples exist THEN add additional triples*
    - See partial axiomatization of the OWL 2 RDF-based semantics as rules in the OWL 2 Profiles specification (Section 4.3)

- Limitations include:
  - *Statements where the existence of an individual enforces the existence of another individual*
    - E.g., the statement "every person has a parent" is not expressible
  - *Restricts class axioms asymmetrically*
    - *Constructs for a subclass cannot necessarily be used as a superclass*

Suggested Reading

- Pascal Hitzler, Markus Krötzsch and Sebastian Rudolph. Foundations of Semantic Web Technologies. Chapman & Hall/CRC, 2009. (Chapter 4)
- Matthew Horridge. A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools. Edition 1.3.

- Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, Sebastian Rudolph. OWL 2 Web Ontology Language Primer (Second Edition), 2012