

Introduction to Semantic Web and Description Logics: Protégé and Pellet

Riccardo Zese

Outline

- 1 Where are we now?
- 2 The Semantic Web Cake
- 3 OWL Ontologies
- 4 Ontology Editors
- 5 Reasoners

Outline

- 1 Where are we now?
- 2 The Semantic Web Cake
- 3 OWL Ontologies
- 4 Ontology Editors
- 5 Reasoners

Summing up...

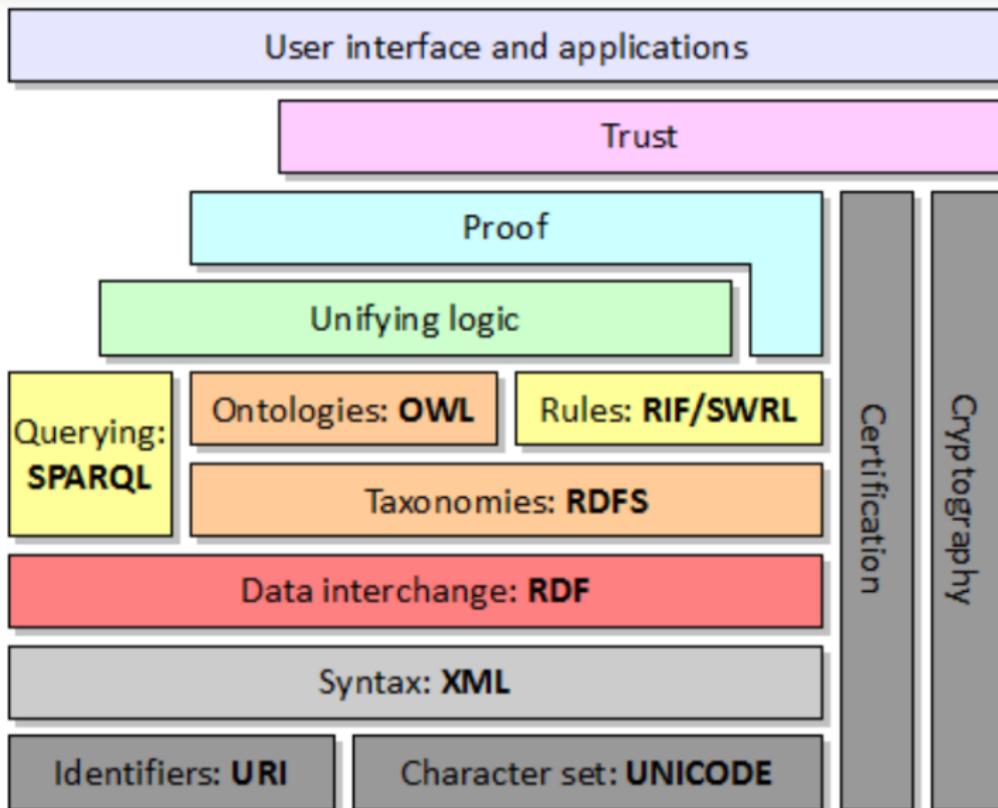
Until now, we have spoken about

- Description Logics, starting from a simple logic and finishing with an introduction of \mathcal{ALC}
- Semantic Web, with some applications and tools, standards, problems

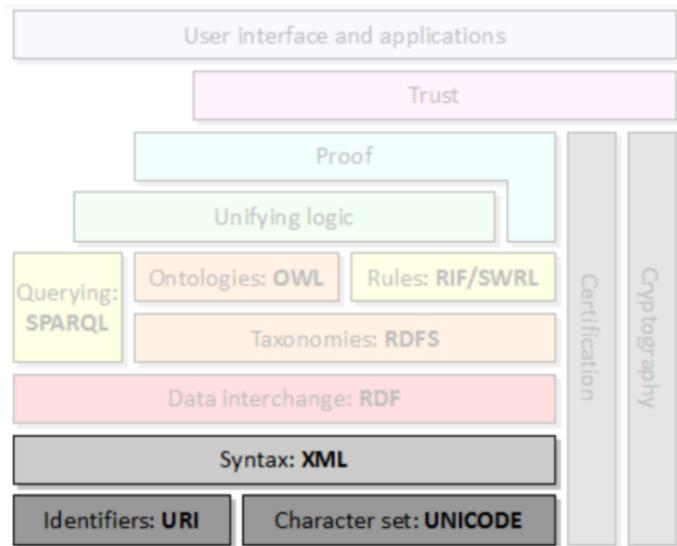
Outline

- 1 Where are we now?
- 2 The Semantic Web Cake**
- 3 OWL Ontologies
- 4 Ontology Editors
- 5 Reasoners

The Semantic Web Cake

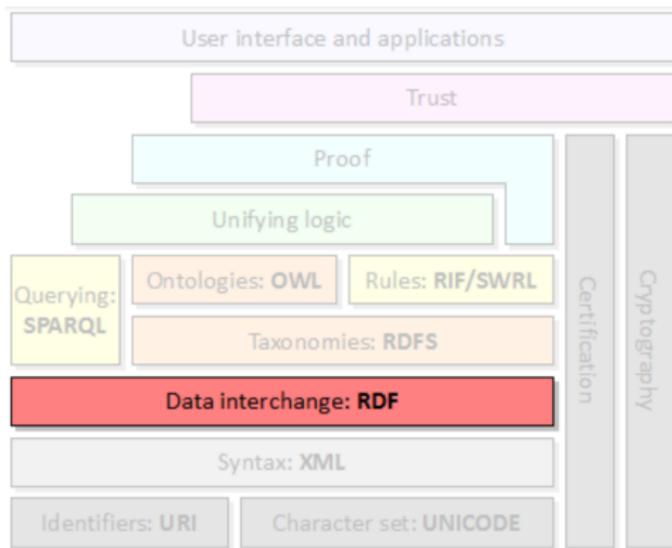


The Basis



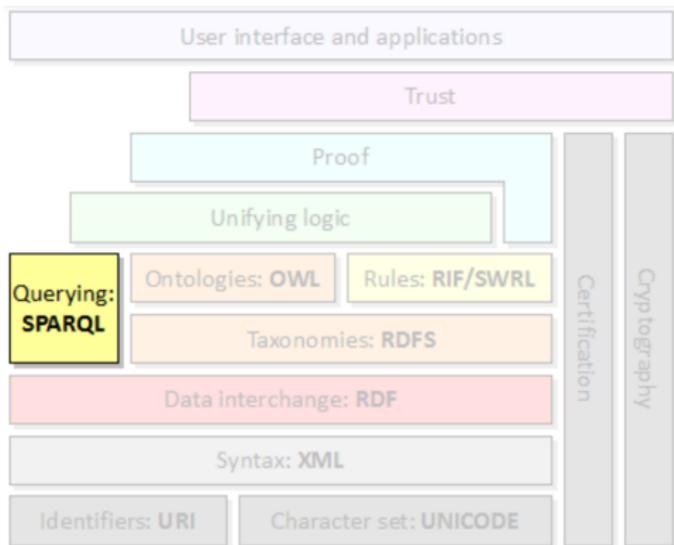
At the base there are the naming mechanism (URI) and the basic language, which specifies the elemental syntax (XML)

Resource Description Framework



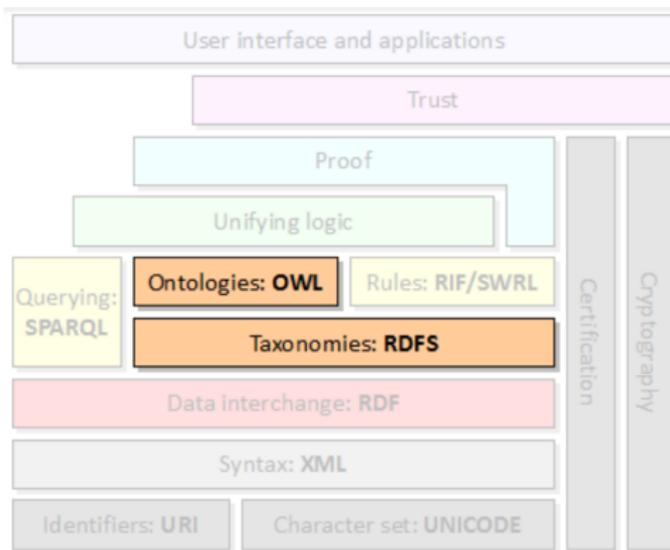
Describes the information of the domain by means of triples
 <subject, predicate, object> or <resource, attribute, value>

Querying at a low level



SPARQL can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. Extracts data, does not infer new information

Knowledge Representation



RDFS extends RDF with type, subclassOf, ...

OWL extends RDFS, with different level of expressivity

OWL

- Three level of expressivity/complexity
 - OWL-Lite, decidable, based on $SHIF(\mathbf{D})$
 - OWL DL, decidable and more expressive, based on $SHOIN(\mathbf{D})$
 - OWL Full, not decidable, highly expressive
- OWL 2 based on $SROIQ(\mathbf{D})$
- Permits the use of many features:
 - Classes (categories): subClassOf, intersectionOf, unionOf, complementOf, enumeration, equivalence, disjoint
 - Properties (Roles, Relations): symmetric, transitive, functional, inverse Functional, range, domain, subPropertyOf, inverseOf, equivalentProperty
 - Instances (Individuals): sameIndividualAs, differentFrom, allDifferent

Ontology

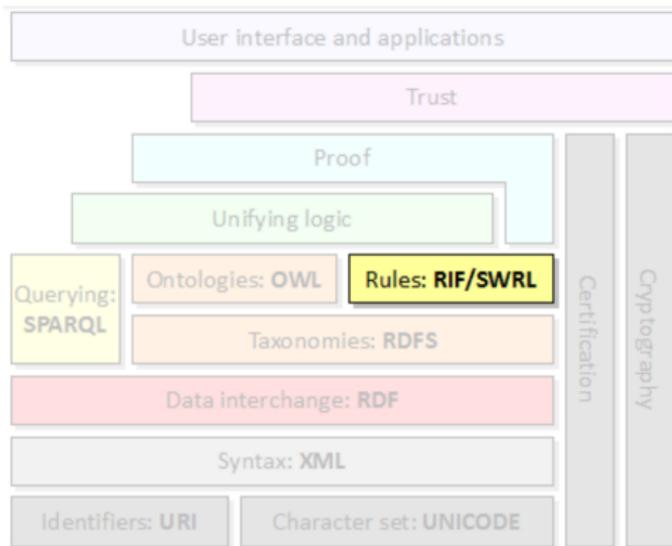
A **formal, explicit description** of a **domain** of interest

- Classes (Concepts)
- Semantic relation between classes (roles)
- Properties associated to a concept (restrictions, ...)
- Logic (axioms, inference rules)

Knowledge Base

Knowledge Base = Ontology + Instances

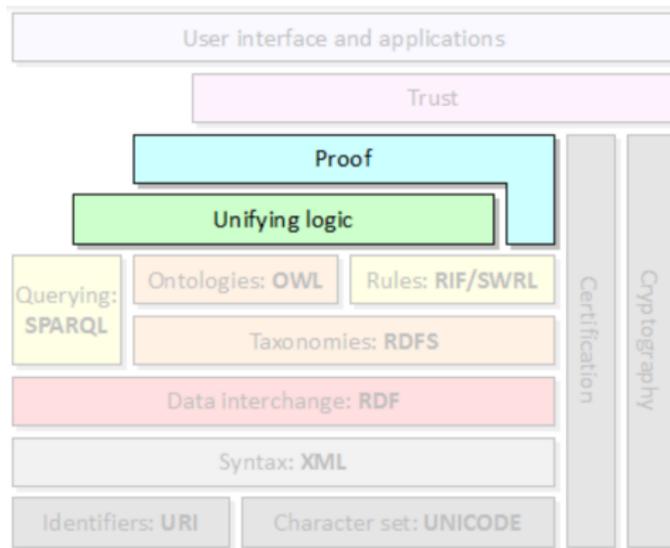
Knowledge Representation



Allows to add rules to data, e.g., **If-then**

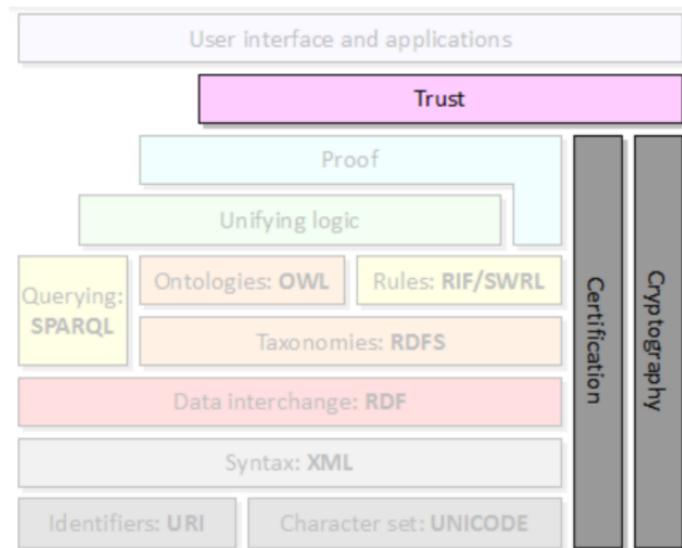
Note, one can use OWL 2 RL Profile

Reasoning



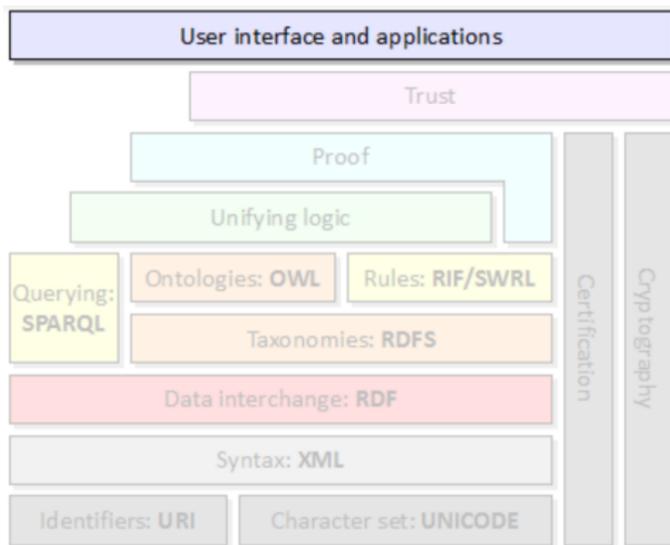
Allows to find new implicit information from the explicit ones, also providing proofs for the inferred new knowledge

Trust



Ensures privacy of the data

UI and Applications



Provides an environment to present application to final users

Outline

- 1 Where are we now?
- 2 The Semantic Web Cake
- 3 OWL Ontologies**
- 4 Ontology Editors
- 5 Reasoners

Ontologies and Semantic Web

As seen in the Semantic Web Cake, there are two ways for defining ontologies for the Semantic Web

- RDF Schema, that extends RDF with basic element for the description of ontologies (type, subclassOf, subPropertyOf, range, domain), good for taxonomies
- OWL, extends on RDFS, based on Description Logics. Defines three different sublanguages of increasing complexity:
 - OWL-Lite: limited support for certain features (ex.: cardinality), good for thesauri or hierarchies
 - OWL DL: good expressiveness, based on Description Logics, suited for modeling knowledge bases
 - OWL Full: minimal compatibility with RDFS

OWL and Description Logics

- Description Logics is a family of logics
- Each logic is distinguished from the other depending on which operators are supported
- The more supported operators, the higher the complexity

OWL and Description Logics

OWL DL supports the following operators

Axiom	DL Syntax	Example
subClassOf	$C_1 \sqsubseteq C_2$	<i>Human</i> \sqsubseteq <i>Animal</i> \sqcap <i>Biped</i>
equivalentClasses	$C_1 \equiv C_2$	<i>Man</i> \sqsubseteq <i>Human</i> \sqcap <i>Male</i>
disjointWith	$C_1 \sqsubseteq \neg C_2$	<i>Male</i> $\sqsubseteq \neg$ <i>Female</i>
sameIndividualAs	$\{x_1\} \equiv \{x_2\}$	$\{President_Bush\} \equiv \{G_W_Bush\}$
differentFrom	$\{x_1\} \sqsubseteq \neg \{x_2\}$	$\{robb\} \sqsubseteq \neg \{tony\}$
subPropertyOf	$R_1 \sqsubseteq R_2$	<i>hasDaughter</i> \sqsubseteq <i>hasChild</i>
equivalentPropertyOf	$R_1 \equiv R_2$	<i>cost</i> \equiv <i>price</i>
inverseOf	$R_1 \equiv R_2^-$	<i>hasChild</i> \equiv <i>hasParent</i> ⁻
transitiveProperty	R^+	<i>ancestor</i> ⁺
functionalProperty	$T \sqsubseteq \leq 1P$	$T \sqsubseteq \leq 1$ <i>hasMother</i>
inverseFunctionalProperty	$T \sqsubseteq \leq 1P^-$	$T \sqsubseteq \leq 1$ <i>hasSSN</i> ⁻

Note: all the operators that combines properties are applicable also to those that involve datatypes

A few of terminology

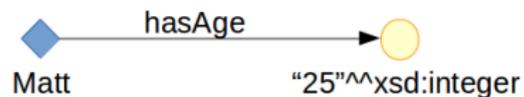
OWL	\Leftrightarrow	DL
Class	\Leftrightarrow	Concept
Property	\Leftrightarrow	Role
Instance	\Leftrightarrow	Individual

A few of terminology

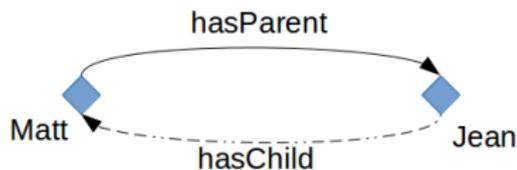
Object Property



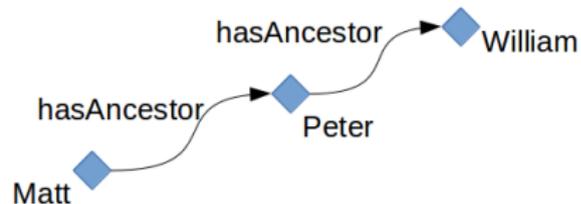
Datatype Property



Inverse Property

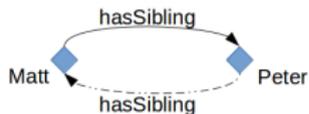


Transitive Property

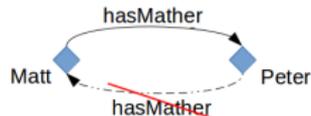


A few of terminology

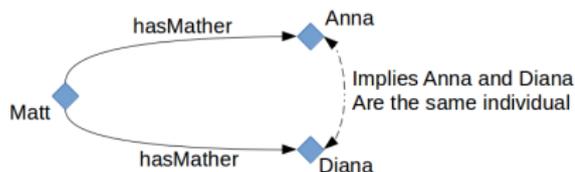
Symmetric Property



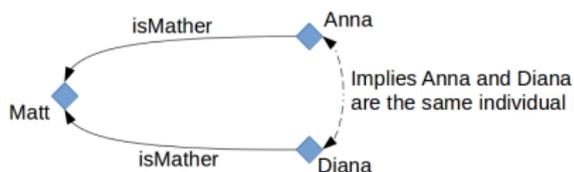
Antisymmetric Property



Functional Property



Inverse Functional Property



Reflexive Property

If r is reflexive, then

$$\forall a \in \text{Individuals} | r(a, a)$$

Irreflexive Property

If r is irreflexive, then

there cannot exist $r(a, a)$

Outline

- 1 Where are we now?
- 2 The Semantic Web Cake
- 3 OWL Ontologies
- 4 Ontology Editors**
- 5 Reasoners

Ontology Editors

There are many editor for developing ontologies

- **WebODE**, <http://mayor2.dia.fi.upm.es/oeg-upm/index.php/en/old-technologies/60-webode>
- **ICOM**, <http://www.inf.unibz.it/~franconi/icom/>
- **Protégé**, <http://protege.stanford.edu/>
- much more, some of them listed at <http://www.w3.org/2001/sw/wiki/Category:Editor>

Protégé

- Graphical editor
- Developed at the Stanford Center for Biomedical Informatics Research (US)
- Open Source, Java based, highly extensible
- Plug-in environment, with a very large number of available plug-ins
- Can export ontologies in several different formats (OWL, RDFS, Turtle, ...)
- A web interface is also available at <http://webprotege.stanford.edu/>

Protégé 5.0

Protégé presents several tabs

The screenshot displays the Protégé 5.0 web interface. At the top, the browser address bar shows the URL: `untitled-ontology-85 (http://www.semanticweb.org/riccardo/ontologies/2014/9/untitled-ontology-85) : [http://www.semanticweb.org/riccardo...`. Below the browser window, the Protégé application has a menu bar with `File Edit View Reasoner Tools Refactor Window Help`. A navigation bar contains tabs for `Active Ontology`, `Entities`, `Classes`, `Object Properties`, `Data Properties`, `Annotation Properties`, `Individuals`, `OWLviz`, `DL Query`, `OntoGraf`, `SPARQL Query`, and `Ontology Differences`. The `Active Ontology` tab is selected, showing the `Ontology header` section. This section includes fields for `Ontology IRI` (set to `http://www.semanticweb.org/riccardo/ontologies/2014/9/untitled-ontology-85`) and `Ontology Version IRI` (set to `e.g. http://www.semanticweb.org/riccardo/ontologies/2014/9/untitled-ontology-85/1.0.0`). Below this is an `Annotations` section with a plus sign. At the bottom of the header section are tabs for `Ontology imports`, `Ontology Prefixes`, and `General class axioms`. The `Ontology imports` tab is active, showing `Imported ontologies:` with sub-sections for `Direct Imports` and `Indirect Imports`, both with plus signs. The status bar at the bottom indicates `No Reasoner set. Select a reasoner from the Reasoner menu` and has a checked `Show Inferences` option.

Protégé 5.0

- Active Ontology, shows information about the open ontology
- Entities, gives a centralized place where the user can modify quite all the information of the ontology
- Classes, classes editor
- Object Properties
- Data Properties
- Annotation Properties (metadata...)
- Individuals

- SPARQL Query, SPARQL engine

Protégé - Classes Editor

untitled-ontology-85 (http://www.semanticweb.org/riccardo/ontologies/2014/9/untitled-ontology-85) : [http://www.semanticweb.org/riccardo... - + x]

File Edit View Reasoner Tools Refactor Window Help

untitled-ontology-85 (http://www.semanticweb.org/riccardo/ontologies/2014/9/untitled-ontology-85) Search for entity

Active Ontology Entities Classes Object Properties Data Properties Annotation Properties Individuals OWLViz DL Query OntoGraf SPARQL Query Ontology Differences

Class hierarchy Class hierarchy (inferred)

Class hierarchy: Thing

Annotations Usage

Annotations: Thing

Annotations +

Class Annotations

Description: Thing

Equivalent To +

SubClass Of +

General class axioms +

SubClass Of (Anonymous Ancestor)

Members +

Target for Key +

Disjoint With +

Necessary & Sufficient Class Restrictions

Necessary Class Restrictions

Disjointness & Definitions

No Reasoner set. Select a reasoner from the Reasoner menu Show Inferences

Protégé - Classes Editor

For each class, is possible to specify

- The hierarchy
- General information (annotations)
- Necessary and Sufficient conditions (\equiv)
- Necessary conditions (\sqsubseteq)
- Disjointness conditions with other classes (by default, in OWL all the classes can overlap with each others)

Class Definition

A class can be modeled by defining its necessary and sufficient conditions, by specifying expressions (subClassOf relation with conjunction/disjunction between concepts) or by specifying restrictions on properties

Class Definition Using Restrictions

- **Quantifier Restrictions**

- **Existential Restriction:** $\exists r.C$, `:someValuesFrom`, keyword **some**
class whose individuals are r -related with at least one individuals of C
- **Universal Restriction:** $\forall r.C$, `:allValuesFrom`, keyword **only**
class whose individuals are r -related with only individuals of C

- **Cardinality Restrictions (possibly qualified)**

- **Minimum Cardinality Restriction:** $\geq nr(.C)$, `:minCardinality`, keyword **min**
class whose individuals are r -related with at least n individuals (of C)
- **Exact Cardinality Restriction:** $= nr(.C)$, `:exactCardinality`, keyword **exactly**
class whose individuals are r -related with exactly n individuals (of C)
- **Maximum Cardinality Restriction:** $\leq nr(.C)$, `:maxCardinality`, keyword **max**
class whose individuals are r -related with at most n individuals (of C)

Protégé - Object Properties Editor

untitled-ontology-85 (http://www.semanticweb.org/riccardo/ontologies/2014/9/untitled-ontology-85) : [http://www.semanticweb.org/riccard... - + x]

File Edit View Reasoner Tools Refactor Window Help

untitled-ontology-85 (http://www.semanticweb.org/riccardo/ontologies/2014/9/untitled-ontology-85) Search for entity

Active Ontology Entities Classes Object Properties Data Properties Annotation Properties Individuals OWLViz DL Query OntoGraf SPARQL Query Ontology Differences

Object property hierarchy

topObjectProperty

Object Property Explorer

add/remove
rearrange properties

Annotations Usage

Annotations:

Annotations +

Object Property Annotations

Characteristics

Functional
 Inverse functional
 Transitive
 Symmetric
 Asymmetric
 Reflexive
 Irreflexive

Object Property Axioms

Description:

Equivalent To +
SubProperty Of +
Inverse Of +
Domains (intersection) +
Ranges (intersection) +
Disjoint With +
SuperProperty Of (Chain) +

Necessary & Sufficient
Object Property Restrictions

Necessary
Object Property Restrictions

Disjointness, Range,
Domain & Definitions

No Reasoner set. Select a reasoner from the Reasoner menu Show Inferences

Protégé - Object Properties Editor

For each object property, is possible to specify

- The hierarchy
- General information (annotations)
- Characteristics (Functional Property, Transitive Property, ...)
- Necessary and Sufficient conditions (\equiv)
- Necessary conditions (\sqsubseteq)
- Disjointness conditions with other properties
- Domain
- Range

Protégé - Data Properties Editor

untitled-ontology-85 (http://www.semanticweb.org/riccardo/ontologies/2014/9/untitled-ontology-85) : [http://www.semanticweb.org/riccardo... - + x]

File Edit View Reasoner Tools Refactor Window Help

untitled-ontology-85 (http://www.semanticweb.org/riccardo/ontologies/2014/9/untitled-ontology-85) Search for entity

Active Ontology Entities Classes Object Properties Data Properties Annotation Properties Individuals OWLViz DL Query OntoGraf SPARQL Query Ontology Differences

Data property hierarchy: topDataProperty

Data Property Explorer
add/remove
rearrange properties

Annotations Usage

Annotations: Annotations +

Data Property Annotations

Characteristics: Characteristics
 Functional
for data properties, only **Functional** is available

Description: Description +
Equivalent To +
SubProperty Of +
Domains (Intersection) +
Ranges +
Disjoint With +

Necessary & Sufficient
Data Property Restrictions

Necessary
Data Property Restrictions

Disjointness, Range & Domain

No Reasoner set. Select a reasoner from the Reasoner menu Show Inferences

Protégé - Data Properties Editor

For each object property, is possible to specify

- The hierarchy
- General information (annotations)
- Characteristics (Functional Property only)
- Necessary and Sufficient conditions (\equiv)
- Necessary conditions (\sqsubseteq)
- Disjointness conditions with other properties
- Domain
- Range

Domain & Range

Properties link individuals from a **domain** with individuals from a **range**

The definition of a domain or a range is used during the inference process to infer new knowledge

E.g.: given the classes Pizza and PizzaTopping, the relation hasTopping has:

- Pizza as domain
- PizzaTopping as range

If I model that iceCream hasTopping chocolate, then the fact that iceCream is a Pizza can be inferred

Note: domain and range are inverted with inverse properties

Protégé - Individuals Editor

untitled-ontology-85 (http://www.semanticweb.org/riccardo/ontologies/2014/9/untitled-ontology-85) : [http://www.semanticweb.org/riccardo... - + x]

File Edit View Reasoner Tools Refactor Window Help

untitled-ontology-85 (http://www.semanticweb.org/riccardo/ontologies/2014/9/untitled-ontology-85) Search for entity

Active Ontology Entities Classes Object Properties Data Properties Annotation Properties Individuals OWLviz DL Query OntoGraf SPARQL Query Ontology Differences

Class hierarchy (inferred)
Class hierarchy
Class hierarchy: Thing
Thing

Class Explorer

Individuals:
add/remove rearrange individuals

Instance Explorer

Annotations Usage
Annotations
Individual Annotations

Description:
Types
Same Individual As
Different Individuals
Individual Class Instantiations

Property assertions:
Object property assertions
Data property assertions
Negative object property assertions
Negative data property assertions
Individual Property Instantiations

No Reasoner set. Select a reasoner from the Reasoner menu Show Inferences

Protégé - Individuals Editor

For each individual, it is possible to specify

- The type (classes)
- Relationship with other individuals (equality, disequality)
- Property assertions

Inferred Knowledge Base

Inferred ontology

- Protégé infers the **inferred hierarchy** or classes and properties by means of the subsumption mechanism (use of reasoners, e.g., FACT++)

Inferred classification of individuals

Consistency checking: check whether each class can have at least one individual that belongs to it

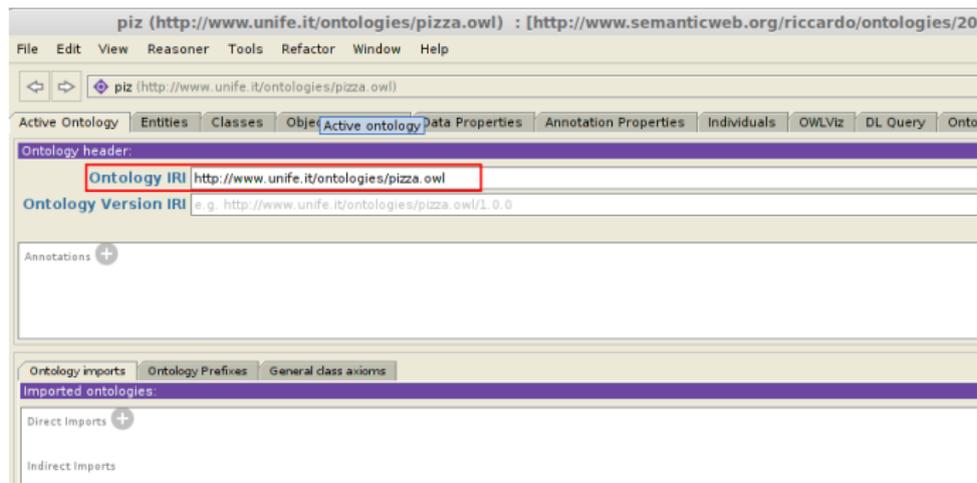
Exercise

A simplified and updated version of the step-by-step exercise from “Protégé OWL Tutorial”

http://mowl-power.cs.man.ac.uk/protegeowltutorial/resources/ProtegeOWLTutorialP4_v1_3.pdf

Exercise - Step 1

- Start Protégé and you will see the *main tab*. Replace the *IRI* with `http://www.unife.it/ontologies/pizza.owl`



- Now our ontology has a significant IRI. In *Annotation* we can add a comment (e.g., what the ontology models)

Exercise - Step 2

- Go to *Classes* tab and create a subclass of *Thing*, called *Pizza*

The screenshot shows the Protégé ontology editor interface. The main window displays the 'Classes' tab for the ontology 'piz (http://www.unife.it/ontologies/pizza.owl)'. The 'Class hierarchy' panel shows 'Thing' as the root class. A red box highlights the 'Create new class' icon (a circle with three dots) in the class hierarchy toolbar. A dialog box titled 'Create a new OWLClass' is open in the foreground, with the following fields:

- Name: Pizza
- IRI: https://sites.google.com/a/unife.it/ml/bundle#Pizza
- New entity options... button
- OK and Cancel buttons

Below the dialog, the 'SubClass Of (Anonymous Ancestor)' property is visible.

Exercise - Step 3

- Create two sibling classes of `Pizza`, called `PizzaTopping` and `PizzaBase` TIP: You can create a subclass of `Thing` or a sibling of the other classes

The screenshot shows the Protégé interface for the ontology `piz (http://www.unife.it/ontologies/pizza.owl)`. The **Class hierarchy** panel displays a tree structure with `Thing` as the root, and `PizzaTopping` and `Pizza` as its children. A red box highlights the class hierarchy icons, and a red arrow points to the `PizzaTopping` class with the text "Delete the selected class".

The **Create a new OWLClass** dialog box is open, showing the following fields:

- Name: `PizzaBase`
- IRI: `https://sites.google.com/a/unife.it/ml/bundle#PizzaBase`
- New entity options... button
- OK and Cancel buttons

Exercise - Step 4

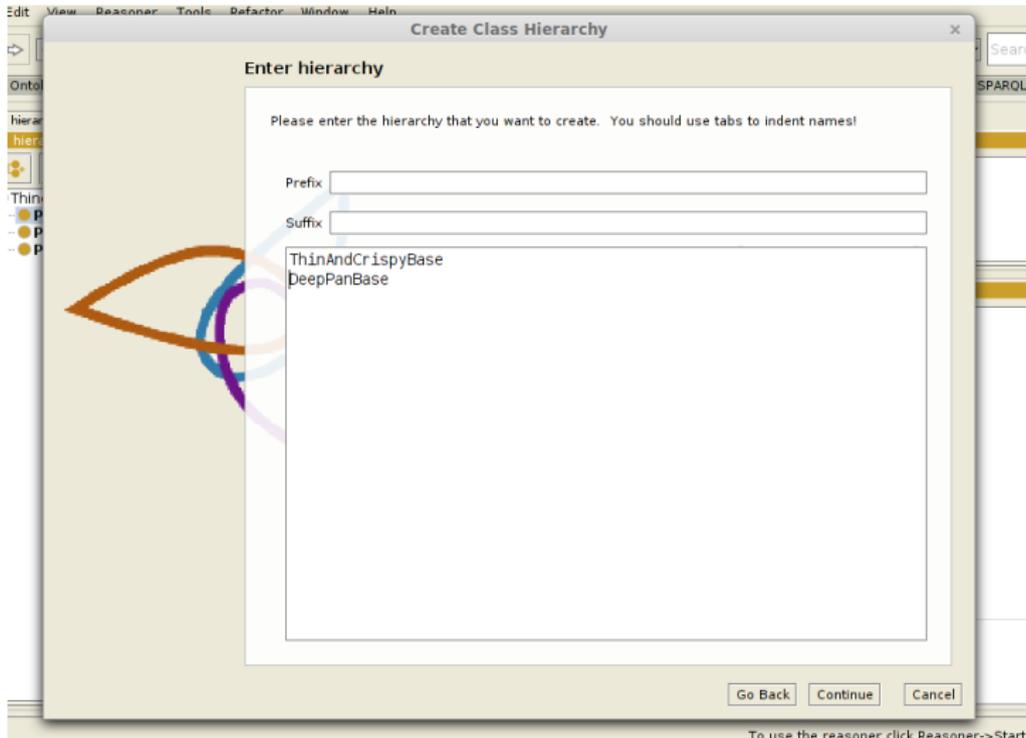
- Make the created classes disjoint from each other. Select `Pizza`, press *Disjoint With* in *Description* and select the other classes

The screenshot shows the Protégé ontology editor interface. The main window displays the class hierarchy for the 'Pizza' ontology, with 'Pizza' selected. The 'Description' panel on the right shows the 'Disjoint With' button highlighted with a red box. A dialog box titled 'Pizza' is open, showing the class hierarchy with 'PizzaBase' and 'PizzaTopping' selected, also highlighted with a red box. The dialog box has 'OK' and 'Cancel' buttons at the bottom.

Exercise - Step 5

- Create a class hierarchy for `PizzaBase`
- From the *Tools* menu select *Create Class Hierarchy...*
- Select `PizzaBase`
- Type in the class name the two names `ThinAndCrispyBase` and `DeepPanBase`, and click *Continue*
- After the tool have checked the entered names, tick *Make sibling classes disjoint* and click *Finish*

Exercise - Step 5



To use the reasoner click Reasoner->Start

Exercise - Step 6

- Create a class hierarchy for `PizzaTopping`, usign *Create Class Hierarchy...* tool
- Select `PizzaTopping`
- Type in the class name text you find in the next step, set `Topping` in *Suffix* (the tool automatically appends `Topping` at the end of all the created classes) and click *Continue*
- After the tool have checked the entered names, tick *Make sibling classes disjoint* and click *Finish*

Exercise - Step 6

The tool allows a hierarchy of classes to be entered using a *tab indented tree*. Class names must be indented using tabs, so for example `SpicyBeef`, which we want to be a subclass of `Meat` is entered under `Meat` and indented with a tab.

```
Cheese
  Mozzarella
  Parmezan
Meat
  Ham
  Salami
  SpicyBeef
Seafood
  Tuna
  Anchovy
  Prawn
Vegetable
  Caper
  Mushroom
  Olive
  Onion
  Pepper
  RedPepper
  GreenPepper
Tomato
```

Exercise - Step 6

Create Class Hierarchy

Enter hierarchy

Please enter the hierarchy that you want to create. You should use tabs to indent names!

Prefix

Suffix

- Cheese
 - Mozzarella
 - Parmezan
- Meat
 - Ham
 - Salami
 - SpicyBeef
- Seafood
 - Tuna
 - Anchovy
 - Prawn
- Vegetable
 - Caper
 - Mushroom
 - Olive
 - Onion
 - Pepper
 - RedPepper
 - GreenPepper
 - Tomato

Go Back Continue Cancel

Exercise - Step 6



Exercise - Step 7

- Go to *Object Properties* tab and create a property, called `hasIngredient`
- Create two subproperty of `hasIngredient`, called `hasTopping` and `hasBase`

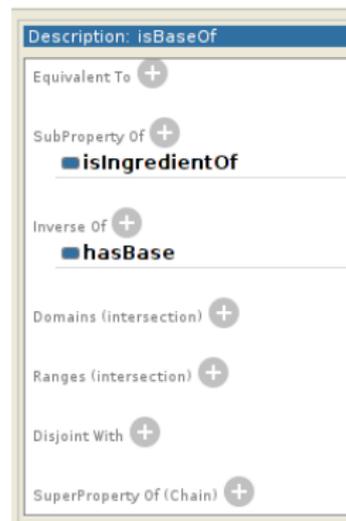


Exercise - Step 8

- Create inverse properties of the properties just defined
- Select the property and click on *InverseOf*, here create the corresponding property

hasIngredient	↔	isIngredientOf
hasTopping	↔	isToppingOf
hasBase	↔	isBaseOf

- You can optionally place the new `isBaseOf` property as a sub-property of `isIngredientOf` (N.B This will get inferred later anyway when you use the reasoner).



Exercise - Step 9

- Make `hasIngredient` property *transitive*
- Make `hasBase` property *functional*

The screenshot shows the Protégé ontology editor interface. On the left, a tree view displays the ontology structure:

- topObjectProperty
 - isIngredientOf
 - isToppingOf
 - isBaseOf
 - hasIngredient
 - hasBase
 - hasTopping

On the right, the 'Annotations: hasIngredient' panel is visible, showing a list of characteristics for the selected property:

- Functional
- Inverse functional
- Transitive
- Symmetric
- Asymmetric
- Reflexive
- Irreflexive

The 'Functional' and 'Transitive' checkboxes are highlighted with red boxes. The 'Transitive' checkbox is checked, indicating that the property is being set to transitive.

Exercise - Step 10

- Specify the the domain (`Pizza`) and range (`PizzaTopping`) for `hasTopping` property
- Specify the the domain (`Pizza`) and range (`PizzaBase`) for `hasBase` property
- Do the inverse for `isToppingOf` and `isBaseOf`

The screenshot displays the Protégé ontology editor interface. On the left, a tree view shows the ontology structure with properties like `isIngredientOf`, `isToppingOf`, `isBaseOf`, `hasIngredient`, `hasBase`, and `hasTopping`. The main window is titled "hasTopping" and shows the configuration for this property. The "Object restriction creator" tab is active, displaying a list of classes: `Pizza`, `PizzaBase`, and `PizzaTopping`. The "Ranges (intersection)" section is highlighted with a red box, indicating the configuration of the range for the property. The "Description" section shows the property is a subproperty of `hasIngredient` and the inverse of `isToppingOf`. The "Domains (intersection)" section shows the domain is `Pizza`. The "Characteristics" section includes checkboxes for Functional, Inverse functional, Transitive, Symmetric, Asymmetric, Reflexive, and Irreflexive. The "Disjoint With" section is also visible.

Exercise - Step 11

- Add that the class `Pizza` is a subclass of $\exists \text{hasBase.PizzaBase}$ (hasBase **some** PizzaBase)
- Add that the class `Pizza` is a subclass of $\exists \text{hasTopping.PizzaTopping}$ (hasTopping **some** PizzaTopping)

The screenshot displays the Protégé ontology editor interface. On the left, a class hierarchy tree shows the following structure:

- Thing
 - PizzaBase
 - ThinAndCrispyBase
 - DeepPanBase
 - PizzaTopping
 - MeatTopping
 - HamTopping
 - SalamiTopping
 - SpicyBeefTopping
 - CheeseTopping
 - MozzarellaTopping
 - ParmezanTopping
 - SeafoodTopping
 - AnchovyTopping
 - PrawnTopping
 - TunaTopping
 - VegetableTopping
 - CaperTopping
 - MushroomTopping
 - OliveTopping
 - OnionTopping
 - PepperTopping
 - GreenPepperTopping
 - RedPepperTopping
 - TomatoTopping
 - Pizza

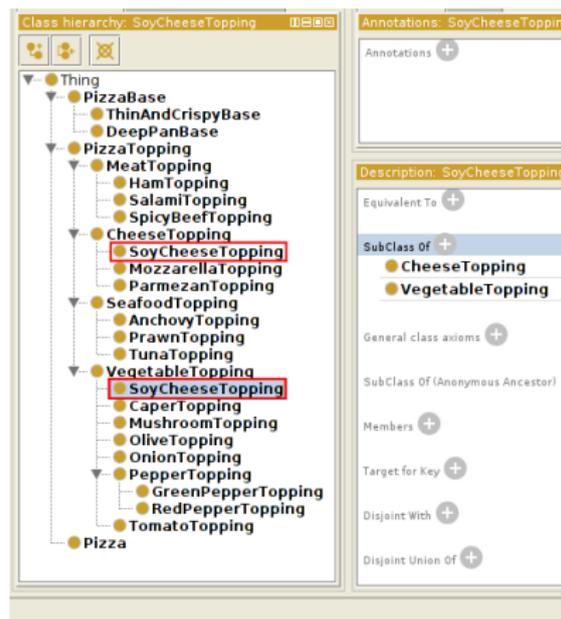
The central pane shows the 'Description: Pizza' section with the following content:

- Equivalent To +
- SubClass Of + (highlighted with a red box)
 - hasBase some PizzaBase
- General class axioms +
- SubClass Of (Anonymous Ancestor)
- Members +
- Target for Key +
- Disjoint With +
 - PizzaTopping, PizzaBase
- Disjoint Union Of +

The right pane is a dialog box titled 'Pizza' with four tabs: 'Data restriction creator', 'Class expression editor', 'Object restriction creator', and 'Class hierarchy'. The 'Class expression editor' tab is active, showing the expression `hasTopping some Pizza` and a list of classes: `Pizza`, `PizzaBase`, and `PizzaTopping`. The 'OK' and 'Cancel' buttons are visible at the bottom of the dialog.

Exercise - Step 12

- Add the class `SoyCheeseTopping` as a subclass of `VegetableTopping` and of `CheeseTopping`



Exercise - Step 13

Add some individuals

- chiliPepper
 - **types** RedPepperTopping
- spicyTomato
 - **types** TomatoTopping
 - **Object property assertions** hasIngredient chiliPepper
- spicyRedDeepPizza
 - **types** Pizza, hasBase some DeepPanBase
 - **Object property assertions** hasTopping spicyTomato
- tunaOnionThinPizza
 - **types** hasBase some ThinAndCrispyBase, hasTopping some OnionTopping, hasTopping some TunaTopping

Exercise - Step 13

Individuals: spicyRedDeepPizza

- ◆ chillPepper
- ◆ **spicyRedDeepPizza**
- ◆ spicyTomato
- ◆ tunaOnionThinPizza

Annotations: spicyRedDeepPizza

Annotations +

Description: spicyRedDeepPizza

Types +

- **hasBase some DeepPanBase**
- **Pizza**

Same Individual As +

Different Individuals +

Property assertions: spicyRedDeepPizza

Object property assertions +

- **hasTopping spicyTomato**

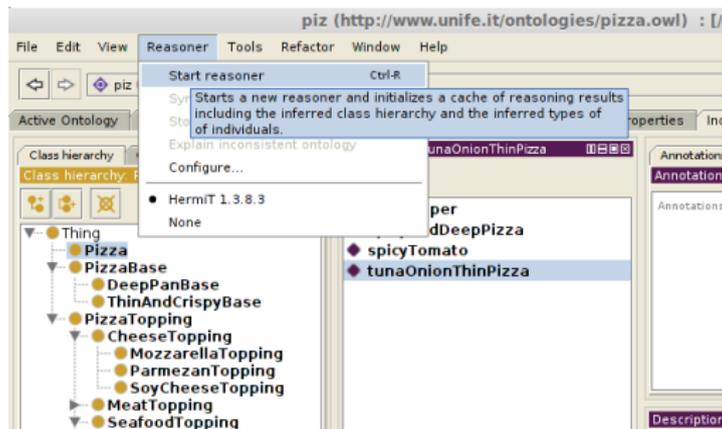
Data property assertions +

Negative object property assertions +

Negative data property assertions +

Exercise - Step 14

Start the internal reasoner and see what happens



Outline

- 1 Where are we now?
- 2 The Semantic Web Cake
- 3 OWL Ontologies
- 4 Ontology Editors
- 5 Reasoners**

Reasoners

There are many reasoners available

- **FaCT++** <http://owl.man.ac.uk/factplusplus/>
- **Hermit** <http://www.hermit-reasoner.com/>
- **Pellet** <https://github.com/stardog-union/pellet>
- **much more, some of them listed at**
<http://www.w3.org/2001/sw/wiki/Category:Reasoner>

Pellet

- Developed by Clark & Parsia LLC
- Open Source, Java based
- Can import ontologies in several different formats (OWL, RDFS, Turtle, ...)
- Can import ontologies from several different sources (local file, remote ontology, ...)

Pellet

“Pellet is an OWL 2 reasoner. [...] Pellet provides functionality to check consistency of ontologies, compute the classification hierarchy, explain inferences, and answer SPARQL queries.”

SOURCE: Pellet Documentation

<https://github.com/stardog-union/pellet>

Features

Pellet can execute several different tasks:

- **classify**: classifies the ontology and display the hierarchy
- **consistency**: checks the consistency of an ontology
- **entail**: checks if all axioms are entailed by the ontology
- **explain**: explains one or more inferences in a given ontology including ontology inconsistency
- **extract**: extract a set of (specified) inferences from an ontology
- **info**: displays information and statistics about 1 or more ontologies

Features

- **lint**: shows problems contained in the ontology (warnings, errors)
- **modularity**: extracts from the ontology information about given classes
- **query**: executes SPARQL queries
- **realize**: computes and displays the most specific instances for each class
- **trans-tree**: computes a transitive-tree closure, reporting the hierarchy of the classes which use the given transitive property
- **unsat**: finds the unsatisfiable classes in the ontology

Entail - Query Types

Entail computes one or more explanations for the given query

- **unsat**: Explain why the given class is unsatisfiable
- **all-unsat**: Explain all unsatisfiable classes
- **inconsistent**: Explain why the ontology is inconsistent
- **hierarchy**: Print all explanations for the class hierarchy
- **subclass**: Explain why C is a subclass of D , where C and D are given classes
- **instance**: Explain why i is an instance of C , where i is an individual and C a class
- **property-value**: Explain why s has value o for property p , where s is an individual, o an individual or a value of a certain datatype and p is a property

Explanation

Roughly: an explanation is a set of axioms from the KB which entail (is a model) for the given query

There could be many different explanations for a given query, depending on the KB

Tableau Algorithm

All the commands of Pellet exploit the Tableau Algorithm for doing inference
The algorithm builds a graph (also called **tableau**)

- A tableau is an ABox represented as a graph in which:
 - Each node represents an individual a and is labeled with the set of concepts it belongs to;
 - Each edge between two individuals a and b is labeled with the set of roles to which the couple (a, b) belongs.

Tableau Algorithm

- A tableau algorithm proves an axiom by refutation
 - Axiom E is entailed if $\neg E$ has no model in the KB.
 - Example 1: to test a class assertion axiom $C(a)$, it adds $\neg C$ to the label of a .
 - Example 2: to test the inconsistency of a concept C , it adds a new anonymous node a to the tableau and adds $\neg C$ to the label of a .

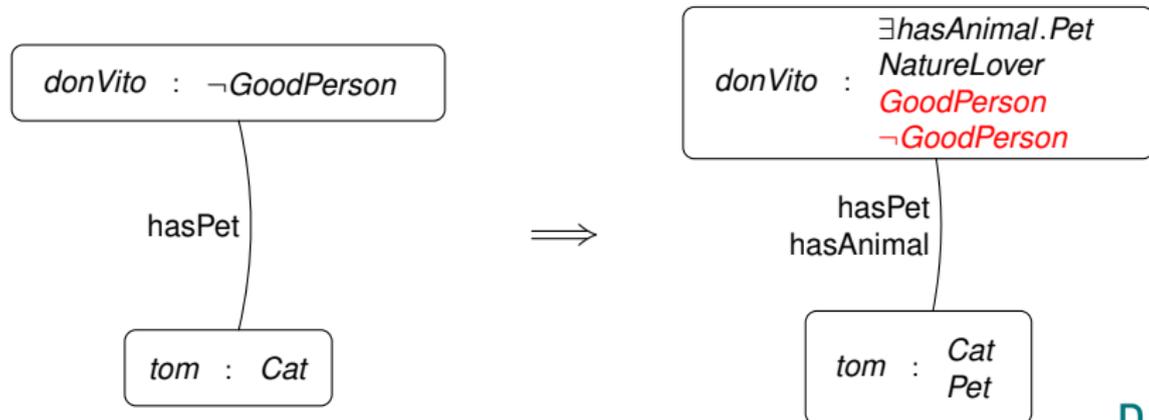
Tableau Algorithm

- A tableau algorithm repeatedly applies a set of consistency preserving tableau expansion rules until a clash is detected or a clash-free graph is found to which no more rules are applicable.
 - there are several expansion rules, often a rule correspond to one concept-forming operator
 - A clash (contradiction) is either:
 - a couple (C, a) where C and $\neg C$ are present in the label of a node;
 - a couple $(a = b, a \neq b)$, where a and b are individuals.
- If the expansion of the tableau with the query leads to at least one clash the query is entailed w.r.t. the KB.

The Tableau Algorithm

$tom : Cat$
 $(donVito, tom) : hasPet$
 $Cat \sqsubseteq Pet$
 $\exists hasAnimal.Pet \sqsubseteq NatureLover$
 $NatureLover \sqsubseteq GoodPerson$
 $hasPet \sqsubseteq hasAnimal$

$Q = donVito : GoodPerson$



Inference with Pellet

- The tableau algorithm finds a single explanation
- Pellet implement also a backtracking algorithm to find all the possible explanations
 - uses an hitting set algorithm that repeatedly removes an axiom from the KB and then computes again a new explanation.
- Some expansion rules are non-deterministic, the tableau algorithm has to handle non-determinism

Why use reasoners?

Reasoners such as Pellet can help also the debug of an knowledge base

The explanations can highlight part of the ontology that contains inconsistent information

Exercise

Try some commands:

- information regarding the knowledge base

```
info file:<path-to-pizza-owl>/pizza.owl
```

- hierarchy of the knowledge base

```
classify file:<path-to-pizza-owl>/pizza.owl
```

Note: SoyCheeseTopping \equiv Nothing!

- check if two individuals are linked together

```
explain -max 100 --property-value  
spicyRedDeepPizza,hasIngredient,chiliPepper  
file:<path-to-pizza-owl>/pizza.owl
```

Exercise

Run the following queries:

- check the unsatisfiability of a class

```
explain -max 100 --unsat SoyCheeseTopping
file:<path-to-pizza-owl>/pizza.owl
```

- check why an individual belongs to a class

```
explain -max 100 --instance spicyRedDeepPizza,Pizza
file:<path-to-pizza-owl>/pizza.owl
```

- check if two individuals are linked together

```
explain -max 100 --property-value
spicyRedDeepPizza,hasIngredient,chiliPepper
file:<path-to-pizza-owl>/pizza.owl
```

Exercise

```

Terminal
riccardo@riccardo ~/src/pellet-2.3.1 $ ./pellet.sh explain -max 100 --property-v
alue https://sites.google.com/a/unife.it/ml/bundle#spicyRedDeepPizza,https://sit
es.google.com/a/unife.it/ml/bundle#hasIngredient,https://sites.google.com/a/unif
e.it/ml/bundle#chiliPepper file:/home/riccardo/papers/seminariosw/Ontology/pizza
.owl
Axiom: spicyRedDeepPizza hasIngredient chiliPepper

Explanation(s):
1) Transitive hasIngredient
   spicyTomato hasIngredient chiliPepper
   spicyRedDeepPizza hasTopping spicyTomato
   hasTopping subPropertyOf hasIngredient

2) hasTopping inverseOf isToppingOf
   Transitive hasIngredient
   spicyTomato hasIngredient chiliPepper
   isIngredientOf inverseOf hasIngredient
   isToppingOf subPropertyOf isIngredientOf
   spicyRedDeepPizza hasTopping spicyTomato

riccardo@riccardo ~/src/pellet-2.3.1 $

```

Thanks.
Questions?