

Prise en main de Protégé

Ce document porte sur la version 5 de Protégé <http://protege.stanford.edu>

Il est basé sur le tutoriel Protégé version 4

(http://mowl-power.cs.man.ac.uk/protegeowl/tutorial/resources/ProtegeOWLTutorialP4_v1_3.pdf)

Les phrases précédées de ☞ indiquent ce qui est à réaliser.

Avant de commencer...

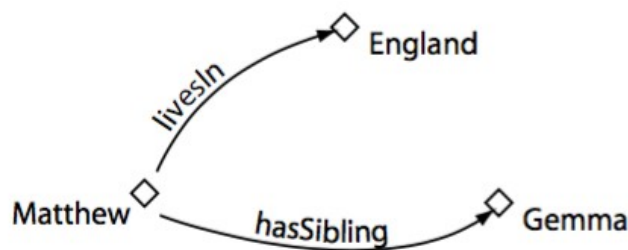
Rappel sur les objectifs des ontologies :

capturer la connaissance sur un domaine d'intérêt en décrivant les concepts du domaine et les relations entre ces concepts.

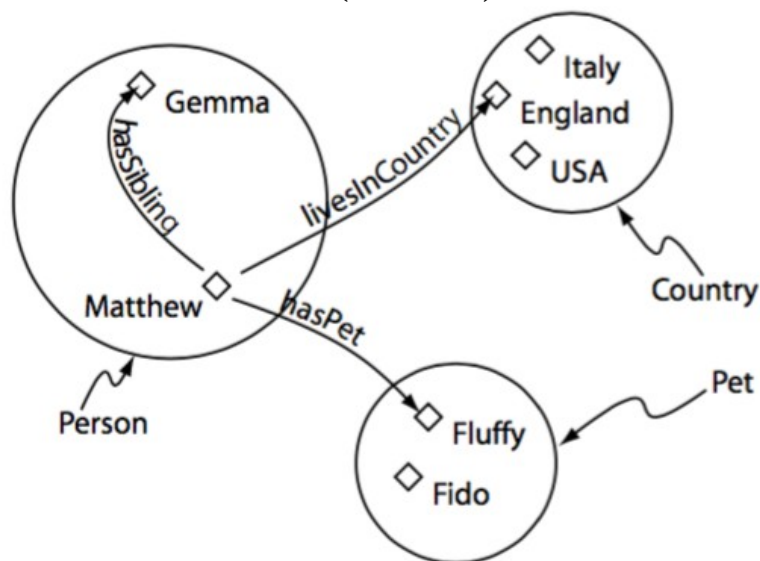
OWL est un langage d'ontologie du World Wide Web Consortium (W3C).

Composants d'une ontologie :

- Individus (instances)
- Propriétés (slots, roles en logique de description, attributs) : relations binaires, les instances de propriétés relient des individus



- Classes : ensembles d'individus. Représentations concrètes de concepts. Décrites formellement par les conditions d'appartenance d'un individu à la classe. Peuvent être organisées sous forme d'une hiérarchie (taxonomie).



C2 est une sous-classe de C1 signifie :

- tous les membres de C2 sont des membres de C1
- être membre de C2 implique être membre de C1
- C2 est subsumé par C1

Conventions de nommage :


- classes : les noms de classes commencent par une majuscule et ne contiennent pas d'espace (notation CamelCase ou CamelBack)
- propriétés : les noms de propriétés commencent par une minuscule, ne contiennent pas d'espace et les différents mots accolés commencent par une majuscule. Ils commencent par « has » ou « is » (ex. hasPart, isPartOf)

Construction de l'ontologie Pizzas

- ☞ Lancer Protégé.
- ☞ Créer une nouvelle ontologie : File/New...

Informations générales sur l'ontologie

L'onglet **Active Ontology** contient les informations générales sur l'ontologie. On peut notamment préciser son IRI.

- ☞ Remplacer l'IRI par <http://www.pizza.com/ontologies/pizza.owl>
- ☞ Ajouter le commentaire « A pizza ontology that describes various pizzas based on their toppings », pour cela il faut cliquer sur  puis sélectionner comment dans la liste des annotations possibles.
- ☞ Sauver l'ontologie sous le nom pizza.owl, au format OWL/XML.

Classes/Hiérarchie de classes

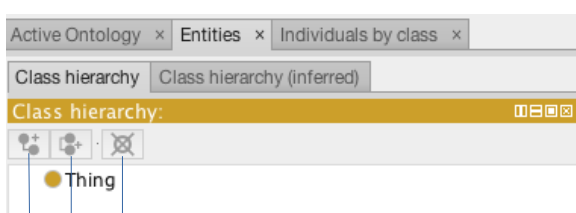
Les classes sont gérées à partir de l'onglet Entities.

L'onglet Entities permet de gérer à la fois les classes et les propriétés. Il décompose la fenêtre en 4 cadres, à gauche on a en haut un cadre concernant la hiérarchie de classes et en bas un cadre concernant les propriétés. Les cadres de droite affichent les informations sur la classe ou la propriété sélectionnée à gauche.

La hiérarchie de classes est visible dans le sous-onglet Class hierarchy, l'onglet Class hierarchy (inferred) correspond à la hiérarchie qui peut être inférée à partir des descriptions des classes.

La classe Thing est la racine de la hiérarchie de classes et représente l'ensemble qui contient tous les individus.

Pour créer la hiérarchie on utilise les 3 boutons sous le titre Class hierarchy (Ajouter une sous-classe devient actif après sélection de Thing) :

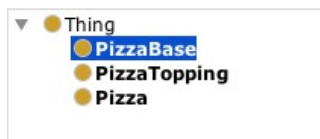


Supprimer la classe sélectionnée

Ajouter une classe sœur

Ajouter une sous-classe

☞ Créer la hiérarchie suivante

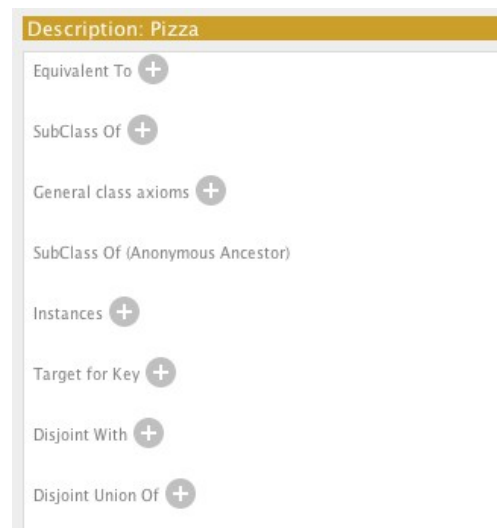


L'axiome de disjonction.

Sauf spécification contraire, il est supposé que les classes OWL peuvent avoir une intersection non vide. On ne peut pas supposer qu'un individu n'est pas membre d'une classe juste parce que cela n'a pas été déclaré.

La disjonction de classe permet de le poser. Lorsqu'on spécifie que des classes sont disjointes, un individu ne peut être instance de plus d'une de ces classes.

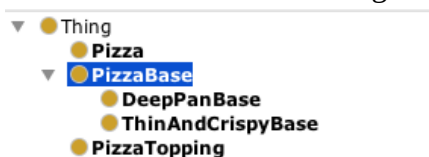
Sous Protégé, pour spécifier des classes disjointes, on sélectionne une des classes puis on clique sur Disjoint With dans le cadre Description et on indique l'ensemble des classes qui doivent être disjointes de la classe sélectionnée.



☞ Spécifier que les classes Pizza, PizzaTopping et PizzaBase sont disjointes. Vérifier que cela apparaît dans la description des 3 classes.

Il est possible de créer directement des portions de la hiérarchie avec l'outil Tools/Create class hierarchy...

☞ Utiliser cet outil pour ajouter des sous-classes à la classe PizzaBase comme sur la figure ci-dessous (pour le moment en utilisant seulement la grande zone de texte, il suffit de taper les noms les uns à la suite des autres en allant à la ligne à chaque fois). Ces classes doivent être disjointes.



On peut également avec cet outil créer une hiérarchie de classes avec suffixe ou préfixe commun.

☞ Créer la hiérarchie de Topping comme sur la figure ci-dessous en utilisant l'outil de création de hiérarchie. Pour différencier les niveaux de hiérarchie dans la zone de texte, on utilise la tabulation. Utilisez le fait que toutes ces classes ont le suffixe Topping. Toutes les classes sœurs sont disjointes.

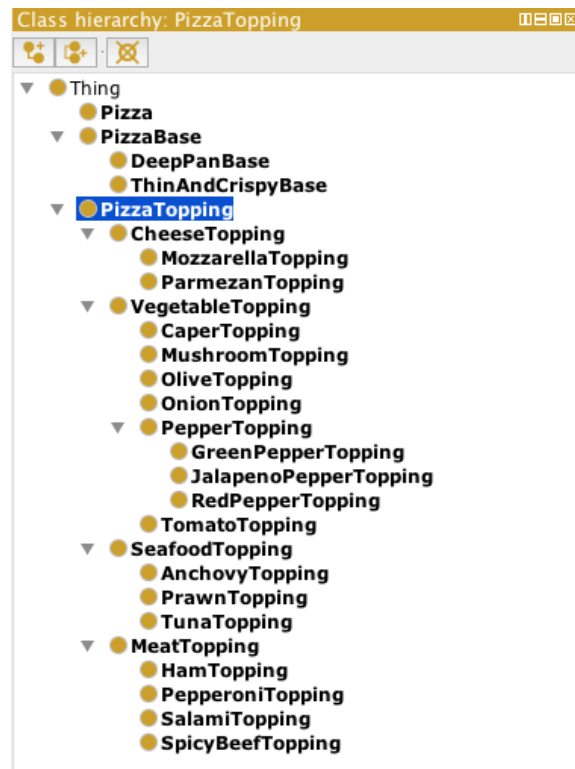
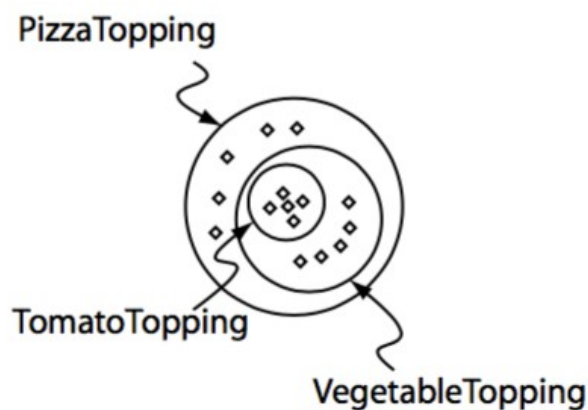


Illustration de la signification de sous-classe : tous les membres de la classe TomatoTopping sont des membres de la classe VegetableTopping et sont des membres de la classe PizzaTopping.



Propriétés/hiérarchie de propriétés.

Il existe plusieurs types de propriétés

- Object properties : relation entre deux individus
- Datatype properties : relation entre un individu et une valeur
- Annotation properties : pour ajouter des informations (métadonnées) sur les classes, individus, propriétés.

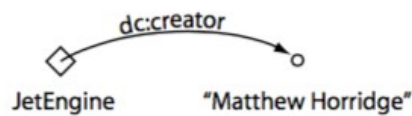
(voir les exemples ci-dessous)



An object property linking the individual Matthew to the individual Gemma



A datatype property linking the individual Matthew to the data literal '25', which has a type of an xsd:integer.

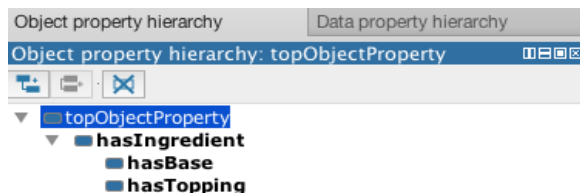


An annotation property, linking the class 'JetEngine' to the data literal (string) "Matthew Horridge".

Object properties. Elles peuvent être créées via le sous-onglet Object property hierarchy, cadre en bas à gauche lorsqu'on est dans l'onglet Entities.

Le sommet de la hiérarchie de propriétés est topObjectProperty.

Utiliser les boutons pour créer la hiérarchie de propriétés ci-dessous

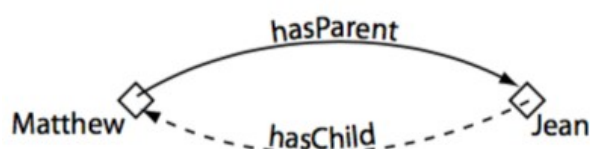



Caractéristiques des object properties :

Characteris

☐ Functional
☐ Inverse function
☐ Transitive
☐ Symmetric
☐ Asymmetric
☐ Reflexive
☐ Irreflexive

- propriété inverse : si une propriété lie un individu a à un individu b, sa propriété inverse lie b à a

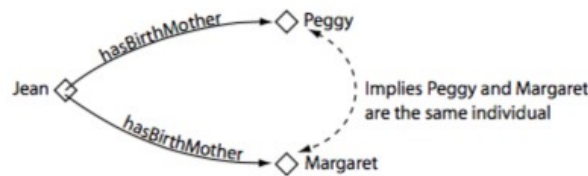


Une fois une object property sélectionnée dans la hiérarchie des objects properties, on peut spécifier une propriété comme inverse en cliquant sur le bouton  de Inverse Of dans la fenêtre Description de la propriété.

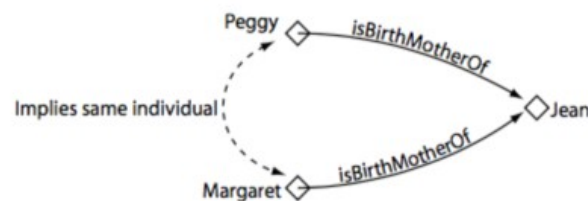
☞ Créer une propriété isIngredientOf, inverse de hasIngredient, une propriété isBaseOf inverse de hasBase et une propriété isToppingOf inverse de hasTopping.

NB : on peut éventuellement placer isBaseOf et isToppingOf comme sous-propriétés de isIngredientOf, mais cela n'est pas nécessaire car sera inféré par le raisonneur.

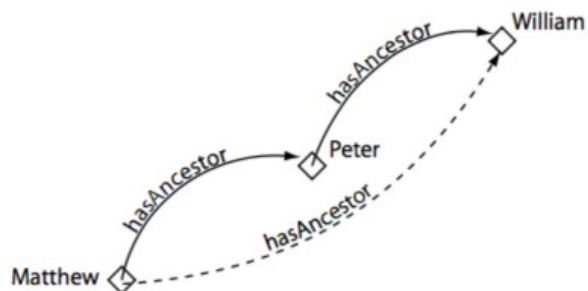
- propriété fonctionnelle : si une propriété est fonctionnelle, un individu a ne peut être relié qu'à au plus un individu b via la propriété



- propriété inverse fonctionnelle : si une propriété est inverse fonctionnelle, il ne peut y avoir qu'un seul individu b relié à un individu a via la propriété. La propriété inverse de cette propriété est fonctionnelle

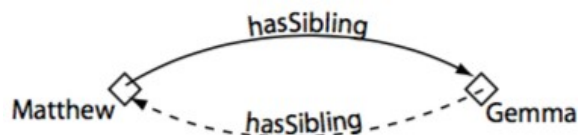


- propriété transitive : si une propriété est transitive, et cette propriété relie a à b d'une part et b à c d'autre part, alors a est relié à c via cette propriété

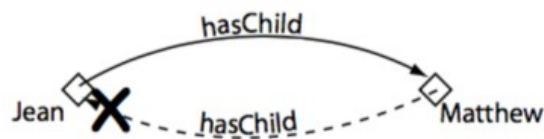


NB : une propriété transitive ne peut pas être fonctionnelle

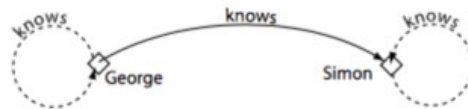
- propriété symétrique : si une propriété est symétrique, et cette propriété relie a à b, alors b est relié à a via cette propriété



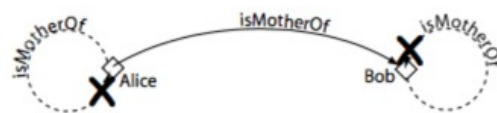
- propriété asymétrique : si une propriété est asymétrique, et cette propriété relie l'individu a à l'individu b, alors b ne peut être relié à a via cette propriété



- propriété réflexive : une propriété est dite réflexive si cette propriété doit relier a à lui-même



- propriété irreflexive : si une propriété est irreflexive et cette propriété relie un individu a à un individu b, a et b ne peuvent être le même individu

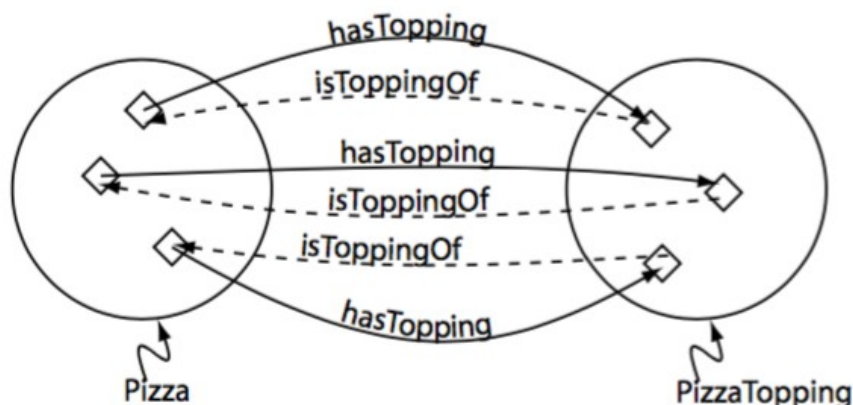


☞ Spécifier les caractéristiques suivantes (cadre caractéristiques) :

- hasIngredient est transitive
- hasBase est fonctionnelle (une pizza a une seule base)

Domain et range des propriétés : les propriétés lient des individus du domain à des individus du range.

NB : domain et range en OWL ne doivent pas être vus comme des contraintes à vérifier, ils sont utilisés comme des axiomes dans le raisonnement. Par exemple, si le domaine de la propriété hasTopping est mis à Pizza et que nous appliquons la propriété à IceCream, cela ne fera pas une erreur mais inférera que la classe IceCream doit être une sous-classe de Pizza !



☞ Spécifier (cadre de description de la propriété) :

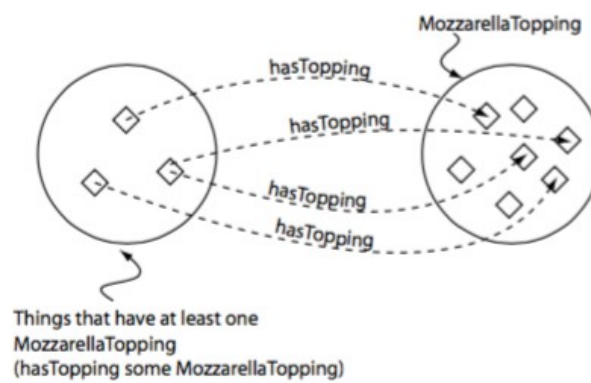
- hasTopping a pour range PizzaTopping
- hasTopping a pour domain Pizza
- hasBase a pour domain Pizza
- hasBase a pour range PizzaBase

Cela signifie que des individus utilisés à gauche de la propriété hasTopping seront inférés comme membres de la classe Pizza.

NB : il est possible de spécifier plusieurs classes comme range d'une propriété, si tel est le cas, le range est interprété comme l'intersection des classes.

Décrire et définir des classes

Restrictions de propriétés : décrivent des classes anonymes contenant tous les individus satisfaisant la restriction. Il s'agit de décrire ou définir une classe d'individus par les relations auxquelles participent ces individus (ex 1 (figure ci-dessous). la classe des individus qui sont reliés à au moins un membre de la classe MozzarellaTopping via la propriété hasTopping, c'est-à-dire la classe des choses qui ont au moins une sorte de garniture mozzarella. Ex 2. la classe des individus qui ne sont reliés qu'à des membres de VegetableTopping via la propriété hasTopping).



Trois catégories de restrictions en OWL :

- quantifier restrictions (existentielles et universelles)
- cardinality restrictions
- hasValue restrictions

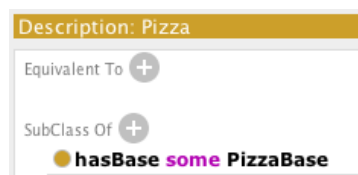
Rappel (logique de description) :

- les restrictions existentielles décrivent des classes d'individus qui ont au moins une relation avec un individu d'une classe donnée via une propriété donnée (ex. 1)
- les restrictions universelles décrivent des classes d'individus qui pour une propriété donnée ont uniquement des relations avec des membres d'une classe donnée (ex. 2)

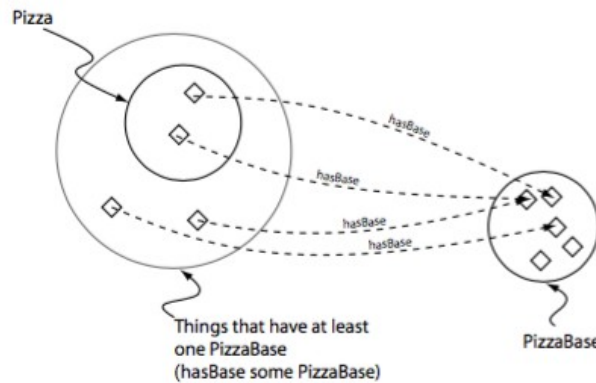
Les restrictions pour une classe sont visibles dans le cadre Description de la classe.

✎ Ajouter la restriction suivante (sélectionner la classe, puis bouton **+** de SubClassOf dans la description de la classe et Class expression editor pour écrire directement la restriction (auto-complétion disponible) ou Object restriction creator pour un assistant à la création) :

Une pizza doit avoir une base : hasBase some PizzaBase (ou avec une syntaxe Logique de description \exists hasBase PizzaBase)



Signification : Pour que quelque chose soit une pizza, il est nécessaire qu'il ait au moins une PizzaBase. Pizza est une sous-classe des choses qui ont au moins une PizzaBase.



☞ Ajouter différents types de pizzas (elles seront groupées à l'intérieur d'une classe NamedPizza, sous-classe de Pizza (ce seront des sous-classes de NamedPizza)) :

- MargheritaPizza : une pizza qui a les garnitures mozzarella et tomate. Ajoutez lui un commentaire A pizza that has Mozzarella and Tomato toppings (c'est une bonne habitude de commenter !)
- AmericanaPizza : une pizza qui a les garnitures pepperoni, mozzarella et tomate (NB : c'est une margherita avec une garniture de plus, on peut cloner Margherita en la sélectionnant puis « Duplicate selected class... » dans le menu Edit, puis repartir de ce clone et faire les ajouts nécessaires)
- AmericanHotPizza : quasiment identique à AmericanaPizza mais a comme garniture supplémentaire JalapenoPepper (vous pouvez encore cloner)
- SohoPizza : quasiment identique à MargheritaPizza mais avec des olives et du parmesan

Toutes ses classes doivent être disjointes (sélectionner une pizza et « Make primitive siblings disjoint » dans le menu Edit).

Raisonneur

Permet de tester si une classe est sous-classe d'une autre et ainsi de calculer la hiérarchie de classe inférée. Permet également de tester la consistance de l'ontologie (de tester si une classe peut avoir des instances ou si c'est impossible).

Dans le menu Reasoner, sélectionner un raisonneur.

Ajout d'une classe inconsistante :

☞ Ajouter une classe ProbeInconsistentTopping qui est à la fois une sous-classe de CheeseTopping et VegetableTopping avec commentaire « This class should be inconsistent when the ontology is classified ».

Dans le menu Reasoner, cliquer sur Start reasoner. ProbeInconsistentTopping est en rouge, indiquant qu'elle est inconsistante. L'inconsistance vient du fait que VegetableTopping et CheeseTopping ont été spécifiées disjointes.

Tester ce qui se passe si on retire la spécification indiquant que VegetableTopping et CheeseTopping sont disjointes.

Puis remettre.

Jusqu'à présent nous avons utilisé uniquement des conditions nécessaires pour décrire les classes. Avec des conditions nécessaires uniquement, on ne peut pas dire que s'il remplit ces conditions, il

doit être un membre de la classe.

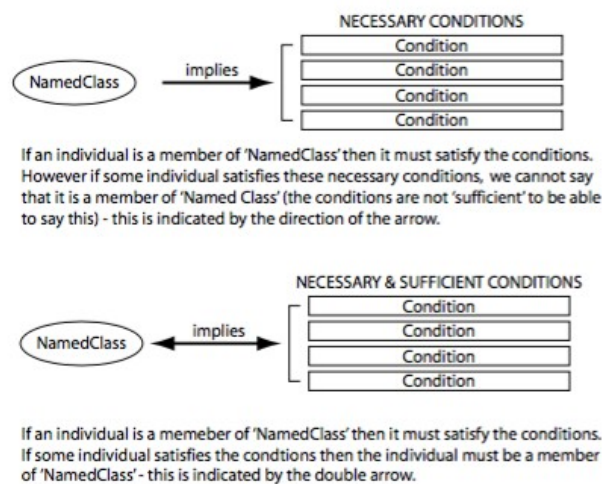
Conditions nécessaires et suffisantes, classes primitives et définies.

☞ Créer une classe CheesyPizza, sous-classe de Pizza, qui a au moins un CheeseTopping

Que pose la description de CheesyPizza ?

Si quelque chose est une CheesyPizza c'est nécessairement une Pizza (sous-classe) et il est nécessaire que cette chose ait au moins une garniture de type CheeseTopping.

Mais, si nous prenons un individu qui est membre de Pizza et qui a une garniture de type CheeseTopping, la description actuelle de CheesyPizza n'est pas suffisante pour déterminer que l'individu est membre de la classe CheesyPizza. Pour faire cela, il faut changer les conditions nécessaires en conditions nécessaires et suffisantes.



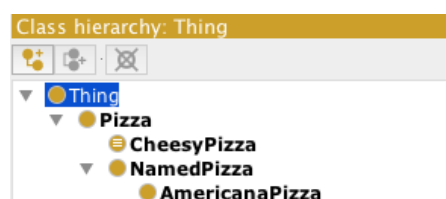
Sous Protégé 5, les conditions nécessaires sont visibles dans la partie SubClass Of du cadre de Description de la classe ; les conditions nécessaires et suffisantes sont visibles dans la partie Equivalent To du cadre de Description de la classe

- **Classe primitive** : a seulement des conditions nécessaires
- **Classe définie** : a au moins un ensemble de conditions nécessaires et suffisantes

Transformer une classe primitive en classe définie (un ensemble de conditions nécessaires en conditions nécessaires et suffisantes) : sélectionner la classe, puis dans le menu Edit, sélectionner « Convert to defined class ».

☞ Convertir CheesyPizza en classe définie.

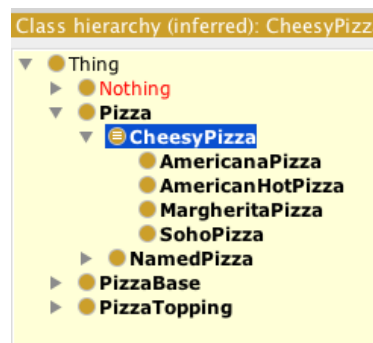
CheesyPizza doit alors apparaître comme ci-dessous



La description de CheesyPizza est maintenant une définition : si quelque chose est membre de CheesyPizza, il est nécessaire que ce quelque chose soit membre de la classe Pizza et ait au moins une garniture membre de la classe CheeseTopping. De plus si un individu est membre de la classe Pizza et a au moins une garniture membre de la classe CheeseTopping alors ces conditions sont suffisantes pour déterminer que l'individu doit être membre de la classe CheesyPizza.

En pratique, le raisonneur peut utiliser cela pour déterminer qu'une classe est sous-classe d'une autre et établir la hiérarchie inférée : supposons deux classes A définie et B, et nous savons que tout individu membre de B satisfait également les conditions qui définissent A, alors on peut déterminer que B est une sous-classe de A.

☞ Utiliser le raisonneur pour inférer les sous-classes de CheesyPizza.
Vous devez obtenir le résultat ci-dessous :



On peut également voir la hiérarchie sous forme de graphe

☞ Ouvrir l'onglet OWLViz Window/Tabs/OWLViz (nécessite d'avoir Graphviz installé : <http://www.graphviz.org>)

Restrictions universelles

Attention une restriction avec quantificateur universel décrit également les individus qui n'ont aucune liaison via la propriété.

☞ Créer une classe **définie** VegetarianPizza, sous-classe de Pizza, qui n'a que des garnitures CheeseTopping ou VegetableTopping (NB : or correspond à \sqcup et only à \forall).

☞ Lancer le raisonneur.

Vous noterez que MargheritaPizza et SohoPizza n'ont pas été inférées sous-classes de VegetarianPizza. Pourtant les restrictions portent sur des garnitures CheeseTopping ou VegetableTopping. Cependant, rien ne dit que ce sont les seuls ingrédients. Il faut pour cela ajouter un **axiome de fermeture**.

Un **axiome de fermeture** consiste en une restriction universelle qui agit sur la propriété pour dire qu'elle ne peut être en lien qu'avec des membres de la classe précisée.

Par exemple, l'axiome de fermeture sur la propriété hasTopping de MargheritaPizza serait $\forall \text{hasTopping} (\text{MozzarellaTopping} \sqcup \text{TomatoTopping})$. Cela permet de dire **exactement** ce que sont les garnitures.

☞ Ajouter l'axiome de fermeture à MargheritaPizza et SohoPizza. Vous pouvez maintenant modifier le commentaire sur MargheritaPizza « A pizza that only has Mozzarella and Tomato toppings ».

On peut le faire plus facilement en sélectionnant la classe, puis une des restrictions portant sur la propriété, clic droit sur le rond jaune à gauche de la restriction et « Create closure axiom ».

☞ Ajouter l'axiome de fermeture à AmericanaPizza et AmericanHotPizza.

☞ Lancer le raisonneur.

Cette fois MargheritaPizza et SohoPizza sont des sous-classes de VegetarianPizza dans la hiérarchie inférée.

Partitions de valeurs.

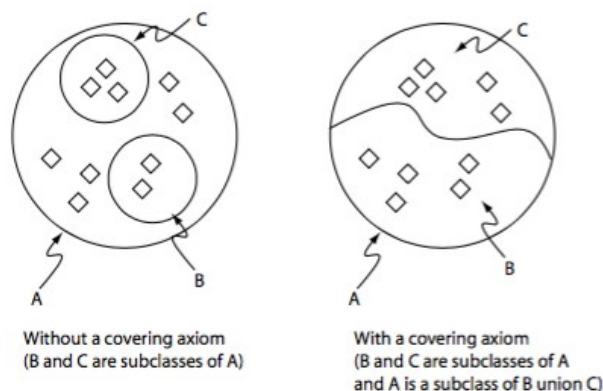
Elles ne font pas partie de OWL, ce sont des design patterns qui permettent de restreindre les valeurs possibles à une liste. Par exemple une partition de valeurs SpicinessValuePartition prendrait les valeurs Mild, Medium et Hot.

Pour créer une partition de valeurs :

- 1- créer une classe pour la partition
- 2- créer une sous-classe de la partition pour chaque option possible
- 3- spécifier la disjonction entre les sous-classes (on peut utiliser dans le menu Edit « Make primitive siblings disjoint »)
- 4- stipuler un axiome de couverture pour que la liste soit exhaustive sélectionner la classe de la partition, et ajouter dans la partie « Equivalent To » de la description Valeur1 or Valeur2 ...
- 5- créer une object property pour la partition, par exemple hasSpiciness
- 6- spécifier la propriété comme fonctionnelle
- 7- spécifier le range de la propriété comme la classe de la partition

L'axiome de couverture est composé de la classe couverte et des classes qui forment la couverture.

Si on n'utilise pas d'axiome de couverture :



☞ Créer une partition de valeurs SpicinessValuePartition, qui est une classe fille d'une classe ValuePartition.

Pour spécifier à quel point les garnitures sont épicées, on peut poser des restrictions comme auparavant ou utiliser le plugin Matrix (plus rapide).

- ☞ Spécifier à quel point les garnitures sont épicées (les garnitures qui n'ont pas de sous-classe) :
- JalapenoPepperTopping : Hot
 - Caper : Mild

- OliveTopping : Mild
- OnionTopping : Medium
- TomatoTopping : Mild
- Mushroom : Mild
- Pepperoni : Medium
- Mozzarella : Mild
- Parmezan : Mild
- Anchovy : Mild
- Prawn : Mild
- Tuna : Mild
- ...

☞ Créer une classe définie SpicyPizza, sous-classe de Pizza, pour les pizzas qui ont une garniture Hot.

☞ Lancer le raisonneur.

Vous devriez trouver AmericanHotPizza comme membre de SpicyPizza dans la hiérarchie inférée.

Continuer le tutoriel à partir de la page 73.

Corrigés

- Une pizza doit avoir une base : hasBase some PizzaBase (ou avec une syntaxe Logique de description \exists hasBase PizzaBase)
- MargheritaPizza : une pizza qui a les garnitures mozzarella et tomate

Description: MargheritaPizza

Equivalent To +

SubClass Of +

- hasTopping some MozzarellaTopping
- hasTopping some TomatoTopping
- NamedPizza

General class axioms +

SubClass Of (Anonymous Ancestor)

- hasBase some PizzaBase

- AmericanaPizza

Description: AmericanaPizza

Equivalent To +

SubClass Of +

- hasTopping some MozzarellaTopping
- hasTopping some PepperoniTopping
- hasTopping some TomatoTopping
- NamedPizza


General class axioms +


SubClass Of (Anonymous Ancestor)

- hasBase some PizzaBase


- AmericanHotPizza

Description: AmericanHotPizza

Equivalent To 

SubClass Of 

- hasTopping **some** JalapenoPepperTopping
- hasTopping **some** MozzarellaTopping
- hasTopping **some** PepperoniTopping
- hasTopping **some** TomatoTopping
- NamedPizza


General class axioms 


SubClass Of (Anonymous Ancestor)

- hasBase **some** PizzaBase


- SohoPizza

Description: SohoPizza

Equivalent To 

SubClass Of 

- hasTopping **some** MozzarellaTopping
- hasTopping **some** OliveTopping
- hasTopping **some** ParmezanTopping
- hasTopping **some** TomatoTopping
- NamedPizza


General class axioms 


SubClass Of (Anonymous Ancestor)

- hasBase **some** PizzaBase


- CheesyPizza primitive

Description: CheesyPizza

Equivalent To 

SubClass Of 

- hasTopping **some** CheeseTopping
- Pizza


General class axioms 

SubClass Of (Anonymous Ancestor)


- hasBase **some** PizzaBase


- CheesyPizza définie

Description: CheesyPizza

Equivalent To 

- Pizza
- and (hasTopping **some** CheeseTopping)

SubClass Of 


General class axioms 

SubClass Of (Anonymous Ancestor)


- hasBase **some** PizzaBase


- VegetarianPizza définie

Description: VegetarianPizza

Equivalent To 

- **Pizza** and (hasTopping only (CheeseTopping or VegetableTopping))

SubClass Of 


General class axioms 


SubClass Of (Anonymous Ancestor)

- hasBase some PizzaBase


- MargheritaPizza avec axiome de fermeture

Description: SohoPizza

Equivalent To 

SubClass Of 

- hasTopping only (MozzarellaTopping or OliveTopping or ParmezanTopping or TomatoTopping)
- hasTopping some MozzarellaTopping
- hasTopping some OliveTopping
- hasTopping some ParmezanTopping
- hasTopping some TomatoTopping
- NamedPizza


General class axioms 


SubClass Of (Anonymous Ancestor)

- hasBase some PizzaBase


- SohoPizza avec axiome de fermeture

Description: MargheritaPizza

Equivalent To 

SubClass Of 

- hasTopping only (MozzarellaTopping or TomatoTopping)
- hasTopping some MozzarellaTopping
- hasTopping some TomatoTopping
- NamedPizza

General class axioms 

SubClass Of (Anonymous Ancestor)

- hasBase some PizzaBase

- AmericanaPizza avec axiome de fermeture

Description: AmericanaPizza

Equivalent To

SubClass Of

- **hasTopping only** (MozzarellaTopping or PepperoniTopping or TomatoTopping)
- **hasTopping some** MozzarellaTopping
- **hasTopping some** PepperoniTopping
- **hasTopping some** TomatoTopping
- **NamedPizza**

General class axioms

SubClass Of (Anonymous Ancestor)

- **hasBase some** PizzaBase

- AmericanHotPizza avec axiome de fermeture

Description: AmericanHotPizza

Equivalent To

SubClass Of

- **hasTopping only** (JalapenoPepperTopping or MozzarellaTopping or PepperoniTopping or TomatoTopping)
- **hasTopping some** JalapenoPepperTopping
- **hasTopping some** MozzarellaTopping
- **hasTopping some** PepperoniTopping
- **hasTopping some** TomatoTopping
- **NamedPizza**

General class axioms

SubClass Of (Anonymous Ancestor)

- **hasBase some** PizzaBase

- SpicyPizza : Pizza and hasTopping some (PizzaTopping and (hasSpiciness some Hot))

Description: SpicyPizza

Equivalent To

- **Pizza** and (hasTopping some (PizzaTopping and (hasSpiciness some Hot)))

SubClass Of

General class axioms

SubClass Of (Anonymous Ancestor)

- **hasBase some** PizzaBase