

# Um guia prático de ologias

is using the Protege-OWL  
net:  
al.pdf>. Access: 20 June

ogias OWL, plugin  
na Internet

juino de 2008

# SUMÁRIO

## 1.INTRODUÇÃO

### 1.1)CONVENÇÕES

## 2.REQUISITOS

## 3.O QUE SÃO AS ONTOLOGIAS OWL?

### 3.1)AS TRÊS ESPÉCIES DE OWL

#### 3.1.1)OWL-Lite

#### 3.1.2)OWL-DL

#### 3.1.3)OWL-Full

#### 3.1.4)A escolha da sub-linguagem

### 3.2)COMPONENTES DA ONTOLOGIA OWL

#### 3.2.1) Individuals (Indivíduos)

#### 3.2.2)Properties (Propriedades)

#### 3.2.3)Classes (Classes)

## 4.CONSTRUÇÃO DE UMA ONTOLOGIA OWL

### 4.1)CRIAÇÃO DE CLASSES

### 4.2)CLASSES DISJUNTAS

### 4.3)USO DO ASSISTENTE OWL PARA CRIAR CLASSES

### 4.4)PROPRIEDADES OWL

### 4.5)PROPRIEDADES INVERSAS

### 4.6)CARACTERÍSTICAS DAS PROPRIEDADES OWL

#### 4.6.1)Propriedades funcionais

#### 4.6.2)Propriedades funcionais inversas

#### 4.6.3)Propriedades transitivas

#### 4.6.4)Propriedades simétricas

### 4.7)DOMAINS (DOMÍNIOS) E RANGES (ESCOPO) DE UMA PROPRIEDADE

### 4.8)DESCRIÇÃO E DEFINIÇÃO DE CLASSES

#### 4.8.1) Restrições de propriedades

#### 4.8.2)Restrições existenciais

### 4.9)USO DE UM MI-MECANISMO DE INFERÊNCIA (REASONER)

#### 4.9.1)Determinação a sub-linguagem OWL

#### 4.9.2)Uso do RACER

#### 4.9.3)Funcionamento do MI

#### 4.9.4)Classes inconsistentes

### 4.10)CONDIÇÕES NECESSÁRIAS E SUFICIENTES (CLASSES PRIMITIVAS E DEFINIDAS)

#### 4.10.1)Classes Primitivas e definidas

### 4.11)CLASSIFICAÇÃO AUTOMÁTICA

#### 4.11.1)Resultados da classificação

### 4.12)RESTRICÇÕES UNIVERSAIS

### 4.13)CLASSIFICAÇÃO AUTOMÁTICA E OWR-OPEN WORLD REASONING (RACIOCÍNIO DE MUNDO ABERTO)

#### 4.13.1)Axiomas de fechamento

### 4.14)PARTIÇÕES DE VALOR

#### 4.14.1)Axiomas de Cobertura

### 4.15)O PROPERTY MATRIX WIZARD (ASSISTENTE MATRIZ DE PROPRIEDADE)

### 4.16)RESTRICÇÕES DE CARDINALIDADE

## 5. MAIS SOBRE OWR-OPEN WORLD REASONING (RACIOCÍNIO DE MUNDO ABERTO)

## 6.OUTRAS CONSTRUÇÕES OWL NO PROTEGE-OWL

### 6.1)CRIAÇÃO DE INDIVÍDUOS

### 6.2)RESTRICÇÕES HASVALUE (TEMVALOR)

### 6.3)CLASSES ENUMERADAS

### 6.4)PROPRIEDADES DE ANOTAÇÃO (ANNOTATION PROPERTIES)

### 6.5)CONJUNTOS MÚLTIPLOS DE CONDIÇÕES NECESSÁRIAS E SUFICIENTES

## 7.OUTROS TÓPICOS

[7.1\) PERFIL DA LINGUAGEM](#)

[7.2\) NAMESPACES E IMPORTAÇÃO DE ONTOLOGIAS](#)

[7.2.1\) Namespaces](#)

[7.2.2\) Criação e edição de namespaces no Protege-OWL](#)

[7.2.3\) Importação de ontologias em OWL](#)

[7.2.4\) Importação de ontologias no Protege-OWL](#)

[7.2.5\) Importação da ontologia Dublin Core](#)

[7.2.6\) Protege-OWL Metadata Ontology \(Ontologia de metadados do Protege-OWL\)](#)

[7.3\) TESTES EM ONTOLOGIAS](#)

[7.4\) TODO LIST \(LISTA DE TAREFAS A FAZER\)](#)

## **APÊNDICE A**

[A.1 RESTRIÇÕES DE QUANTIFICAÇÃO](#)

[A.1.1 someValuesFrom – Restrições Existenciais](#)

[A.1.2 allValuesFrom – Restrições Universais](#)

[A.1.3 Combinação de restrições Existenciais e Universais na descrição de classes](#)

[A.2 RESTRIÇÕES HASVALUE](#)

[A.3 RESTRIÇÕES DE CARDINALIDADE](#)

[A.3.1 Restrições de cardinalidade mínima](#)

[A.3.2 Restrições de cardinalidade máxima](#)

[A.3.3 Restrições de cardinalidade exata](#)

[A.3.4 UNA-Unique Name Assumption \(Presunção de nome único\) e cardinalidades](#)

## **APÊNDICE B**

[B.1 CLASSES INTERSEÇÃO \(É\)](#)

[B.2 CLASSES UNIÃO \(É\)](#)

# LISTA DE EXERCÍCIOS

Exercício 1: faça o seguinte:

Exercício 2: criar um exemplo de projeto

Exercício 3: criação das classes *pizza*, *pizzatopping* e *pizzabase*

Exercício 4: tornar disjuntas as classes *pizza*, *pizzatopping* e *pizzabase*

Exercício 5: use o assistente create multiple classes (criar múltiplas classes) para criar *thinandcrispy* (finaecrocante) e *deeppan* (basegrossa) como subclasses de *pizzabase*

Exercício 6: criação de recheios para *pizzas*

Exercício 7: criar uma propriedade de objeto chamada *hasingredient*

Exercício 8: criação de subpropriedades de *hasingredient*: *hastopping* e *hasbase*

Exercício 9: criação de propriedades inversas

Exercício 10: tornar *hasingredient* (*temingrediente*) uma propriedade transitiva

Exercício 11: tornar funcional a propriedade *hasbase*

Exercício 12: especificar *range* (*escopo*) da relação *hastopping*

Exercício 13: especificar domínio da propriedade *hastopping* como *pizza*

Exercício 14: especificar *domain* (*domínio*) e *range* (*escopo*) para *istoppingof*

Exercício 15: especifique *domain* (*domínio*) e *range* (*escopo*) para a propriedade *hasbase* (*tembase*) e a sua propriedade inversa *isbaseof* (*ébasede*)

Exercício 16: adicionar restrição a *pizza* de forma que *pizza* tenha uma *pizzabase*

Exercício 17: adicione restrição a *pizza* especificando que *pizza* deve ter uma *pizzabase*

Exercício 18: criar uma subclasse de *pizza* chamada *namedpizza*, e uma subclasse de *namedpizza* chamada *margheritapizza*

Exercício 19: criar uma restrição existencial (E) para a classe *margheritapizza*, a propriedade *hastopping* e o *filler* *mozzarellatopping*, para que uma *margheritapizza* tenha pelo menos um *mozzarellatopping*.

Exercício 20: criar uma restrição existencial (E) para a classe *margheritapizza*, a propriedade *hastopping* e o *filler* *tomatotopping*, para que uma *margheritapizza* tenha pelo menos um *tomatotopping*

Exercício 21: clonar e modificar a descrição de *margheritapizza*

Exercício 22: criar classes *americanhotpizza* e *sohopizza*

Exercício 23: tornar disjuntas as subclasses de *namedpizza*.

Exercício 24: adicionar uma *probe class* denominada *probeinconsistenttopping* como subclasse de *cheesetopping* e de *vegetable*

Exercício 25: verificação de inconsistência de *probeinconsistenttopping*

Exercício 26: remoção de declaração disjunta de *cheesetopping* e *vegetabletopping*

Exercício 27: correção da ontologia com a disjunção entre *cheesetopping* e *vegetable*

Exercício 28: criação de subclasse de *pizza* chamada *cheesypizza*, com pelo menos um recheio que é um tipo de *cheesetopping*

Exercício 29: conversão de condições necessárias de *cheesypizza* em condições necessárias e suficientes

Exercício 30: uso do *mi* para computar automaticamente a subclasse de *cheesypizza*

Exercício 31: criação de classe para descrever *vegetarianpizza* (*pizza vegetariana*)

Exercício 32: conversão das condições necessárias de *vegetarianpizza* em condições necessárias e suficientes

Exercício 33: uso do *mi* para classificar a ontologia

Exercício 34: adição de axioma de fechamento à propriedade *hastopping* para *margheritapizza*

Exercício 35: adição de axioma de fechamento a propriedade *hastopping* para *sohopizza*

Exercício 36: criação automatica de axioma de fechamento na propriedade *hastopping* para *americanapizza*

Exercício 37: criação automatica de axioma de fechamento para a propriedade *hastopping* de *americanhotpizza*

Exercício 38: uso do *mi* para classificar a ontologia

Exercício 39: criação de um *valuepartition* para representar o *spiciness* ("tanto" de pimenta) em recheios de *pizza*

Exercício 40: uso do *assistente matriz de propriedade* para especificar o tempero de recheios de *pizza*

Exercício 41: criação de uma classe *spicypizza* como uma subclasse de *pizza*

Exercício 42: uso do *mi* para classificar a ontologia

Exercício 43: criação de *interestingpizza* (*pizza interessante*) com pelo menos 3 recheios.

Exercício 44: uso do *mi* para classificar a ontologia

Exercício 45: criação de *nonvegetarianpizza*, subclasse de *pizza* e disjunta de *vegetarianpizza*

Exercício 46: *nonvegetarianpizza* complemento de *vegetarianpizza*

Exercício 47: adição de *pizza* a condições necessárias e suficientes *nonvegetarianpizza*

Exercício 48: uso do *mi* para classificar a ontologia

Exercício 49: criação de subclasse de *namedpizza* com recheio de *mozzarella*

Exercício 50: uso do *mi* para classificar a ontologia

Exercício 51: criação da classe *country* (*país*) e inserção de indivíduos

Exercício 52: criação da restrição *hasvalue* para especificar que *mozzarellatopping* tem *italy* como país de origem.

[Exercício 53: conversão da classe \*country\* em uma classe enumerada](#)

[Exercício 54: criação de classe para definir um \*triangle\* \(\*triângulo\*\) usando múltiplos conjuntos de condições necessárias e suficientes](#)

[Exercício 55: criação de \*namespace\* e \*prefixo\* para se referir a classes, a propriedades e a indivíduos na ontologia de vinhos](#)

[Exercício 56: importação da ontologia \*koala\* para outra ontologia](#)

[Exercício 57: especificação de local alternativo para uma ontologia importada](#)

[Exercício 58: importar a ontologia \*dublin core meta data elements\*](#)

# 1.Introdução

O presente guia descreve a criação de ontologias utilizando o editor de ontologias *Protege* associado ao *Plug-in Protege-OWL*. Apresenta-se brevemente a linguagem *OWL-Ontology Web Language*, uma linguagem baseada em Lógica Descritiva, e enfatiza-se a construção de ontologias *OWL-DL*. Utiliza-se um *MI-Mecanismo de Inferência (reasoner)* baseado em Lógica Descritiva para verificar a consistência da ontologia e para computar automaticamente a hierarquia de classes. Além disso, descrevem-se constructos *OWL*, tais como a restrição *temValor* e *Classes Enumeradas*, descrevem-se *namespaces*, importação de ontologias, características e funcionalidades da ferramenta *Protege-OWL*.

## 1.1)Convenções

Os exercícios são apresentados da seguinte forma:

### **Exercício 1: Faça o seguinte:**

1. Faça isso;
2. Em seguida, faça isso
3. Em seguida, faça isso.

Outras convenções utilizadas:

DICA, SIGNIFICADO, ATENÇÃO, OBSERVAÇÃO, VOCABULÁRIO

## 2.Requisitos

Para seguir este tutorial é necessário instalar no mínimo o *Protege 3.4*, o *plug-in Protege-OWL* e também o *plug-in OWL-Wizards*, disponíveis na Web em <http://protege.stanford.edu/>.

Recomenda-se (opcional) o *plug-in OWLViz*, que permite visualizar as declarações e inferências obtidas. Os passos para instalação são documentados. Finalmente, é necessário ter um *DIG-Description Logics Implementaters Group* compatível com o *MI* instalado, o que permite computar as relações de subsunção entre as classes e detectar inconsistências. Recomenda-se o uso do *MI* denominado *RACER*, o qual está disponível na Internet.

## 3.0 que são as ontologias OWL?

Ontologias são utilizadas para capturar conhecimento sobre um domínio de interesse. Uma ontologia descreve os conceitos de um domínio e também as relações que existem entre esses conceitos. As diversas linguagens para construção de ontologias fornecem diferentes funcionalidades. O padrão mais recente de linguagens para ontologias é o OWL, desenvolvido no âmbito do W3C-World Wide Web Consortium. O *Protege-OWL* possui um conjunto de operadores (por exemplo, o **AND**, o **OR** e o **NOT**) e é baseado em um modelo lógico que torna possível definir conceitos da forma como são descritos. Conceitos complexos podem ser constituídos a partir de definições de conceitos simples. Além disso, o modelo lógico permite a utilização de um MI, o qual pode verificar se as declarações e as definições da ontologia são mutuamente consistentes entre si e reconhecer se conceitos são adequados a definições. O MI pode, portanto, ajudar a manter a hierarquia, o que é útil quando existem casos em que uma classe tem mais de um pai.

### 3.1)As três espécies de OWL

As ontologias OWL podem ser classificadas em três espécies, de acordo com a sub-linguagem utilizada: *OWL-Lite*, *OWL-DL* e *OWL-Full*. A característica principal de cada sub-linguagem é a sua expressividade: a *OWL-Lite* é a menos expressiva; a *OWL-Full* é a mais expressiva; a expressividade da *OWL-DL* está entre a duas, entre a *OWL-Lite* e a *OWL-Full*.

#### 3.1.1)OWL-Lite

A *OWL-Lite* é a sub-linguagem sintaticamente mais simples. Destina-se a situações em que apenas são necessárias restrições e uma hierarquia de classe simples. Por exemplo, o *OWL-Lite* pode fornecer uma forma de migração para tesouros existentes, bem como de outras hierarquias simples.

#### 3.1.2)OWL-DL

A *OWL-DL* é mais expressiva que a *OWL-Lite* e baseia-se em lógica descritiva, um fragmento de Lógica de Primeira Ordem, passível portanto de raciocínio automático. É possível assim computar automaticamente a hierarquia de classes e verificar inconsistências na ontologia. Este tutorial utiliza a *OWL-DL*.

#### 3.1.3)OWL-Full

A *OWL-Full* é a sub-linguagem OWL mais expressiva. Destina-se a situações onde alta expressividade é mais importante do que garantir a decidibilidade ou completeza da linguagem. Não é possível efetuar inferências em ontologias *OWL-Full*.

#### 3.1.4)A escolha da sub-linguagem

Para maiores detalhes sobre as três sub-linguagens OWL ver informações no W3C. Embora muitos fatores interfiram na escolha da sub-linguagem adequada, existem algumas regras básicas:

- Entre *OWL-Lite* e *OWL-DL*, é necessário saber se os constructos da *OWL-Lite* são suficientes;
- Entre *OWL-DL* e *OWL-Full*, é preciso saber se é importante realizar inferências na ontologia, ou se é importante usar funcionalidades altamente expressivas ou funcionalidades de modelagem, tais como as

meta-classes (classes de classes).

O plug-in do *Protege-OWL* não faz distinção entre a edição de ontologias *OWL-Lite* e *OWL DL*. No entanto, não oferece a opção de restringir a ontologia sob edição para *OWL-DL*, nem permite a expressividade da *OWL-Full*.

### 3.2)Componentes da Ontologia OWL

As ontologias OWL têm componentes similares a estrutura do *Frame-based Protege*, mas a terminologia usada para descrever tais componentes é um pouco diferente da utilizada no *Protege-Frames*. A correspondência entre as duas nomenclaturas é apresentada na TAB. 1.

Protege-Frames	Protege-OWL
<b>Instances</b> ( <i>Instâncias</i> )	<i>Indivíduos</i> ( <i>individuals</i> )
<i>Slots</i> ( <i>Slots</i> )	<i>Propriedades</i> ( <i>Properties</i> )
<i>Classes</i> ( <i>Classes</i> )	<i>Classes</i> ( <i>Classes</i> )

**Tabela 1: correspondência entre nomenclaturas**

#### 3.2.1) *Individuals* (Indivíduos)

Indivíduos representam objetos no domínio de interesse (ou domínio do discurso). Uma diferença importante entre o *Protege* e o *OWL* é que este último não usa o *UNA-Unique Name Assumption*. Isto significa que dois nomes diferentes podem remeter ao mesmo indivíduo. Por exemplo, *Queen Elizabeth* (Rainha Elizabeth), *The Queen* (A rainha) e, *Elizabeth Windsor* podem ser referências ao mesmo indivíduo. Em OWL deve-se declarar explicitamente que os indivíduos são os mesmos, ou diferentes uns dos outros. A FIG. 3.1 mostra uma representação de alguns indivíduos em alguns domínios. Neste tutorial representam-se os indivíduos como "diamantes" em diagramas.



**Figura 3.1: representação de indivíduos**

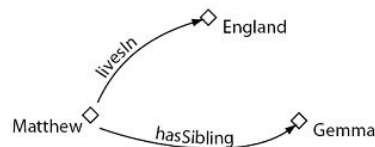
**VOCABULÁRIO** = Os *indivíduos* são também conhecidos como *instâncias*. Os indivíduos podem ser referenciados como *Instâncias de Classes*.

#### 3.2.2)*Properties* (Propriedades)

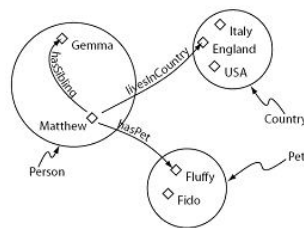
*Propriedades* são relações binárias (relações que contém duas coisas) entre indivíduos, ou seja, as propriedades ligam dois indivíduos. Por exemplo, a propriedade *hasSibling* (*temIrmão*) pode ligar o indivíduo *Matthew* ao indivíduo *Gemma*; ou a propriedade *hasChild* (*temCriança*) pode ligar o indivíduo *Peter* ao indivíduo *Matthew*. As *Propriedades* podem também ser inversas. Por exemplo, a propriedade inversa de *hasOwner* (*temDono*) é *isOwnedBy* (*éPropriedadeDe*). As propriedades podem limitar-se a um valor único: são as *Functional Properties* (*propriedades funcionais*). Elas também podem ser *Transitive Properties* (*Propriedades transitivas*) ou *Symetric Properties* (*Propriedades Simétricas*). Estas características das propriedades são detalhadas adiante. A FIG. 3.2 mostra propriedades que conectam indivíduos.



**VOCABULÁRIO** = *Propriedades* são equivalentes aos *slots* no *Protege-Frames*. Também são conhecidas como *papéis* (*roles*) em lógica descritiva, e *relações* (*relationships*) em UML-*Unified Modeling Language* e em outras abordagens de orientação a objeto. Em muitos formalismos, como no GRAIL, elas são denominadas de *atributos*.



**Figura 3.2:** representação de propriedades



**Figura 3.3:** representação de classes contendo indivíduos

### 3.2.3) Classes (Classes)

As classes OWL são conjuntos que contêm os indivíduos. Elas são descritas formalmente (descrições matemáticas) de forma que sejam apresentados os requisitos para a participação na classe. Por exemplo, a classe *Cat* (*gato*) pode conter todos os indivíduos que são *gatos*, no domínio de interesse. As classes podem ser organizadas em hierarquias superclasse-subclasse, também conhecidas como *taxonomias*. Subclasses são especializações de suas *superclasses*. Por exemplo, considere-se as classes *Animal* e *Cat*: *Cat* pode ser subclasse de *Animal*, e assim *Animal* é superclasse de *Cat*. Isso quer dizer que: *Todos os Gatos são Animais*; *Todos os membros da classe Cat são membros da classe Animal*; *Ser um Gato implica ser um Animal*; *Gato é subclasse de Animal*. Uma característica do *OWL-DL* é que o relacionamento superclasse-subclasse pode ser computado automaticamente por um MI. A FIG. 3.3 mostra uma representação de classes que contém indivíduos: as classes são representadas como círculos.

**VOCABULÁRIO** = O termo *conceito* é às vezes usado no lugar de *classe*. As classes são representações concretas de conceitos.

Em OWL, as classes são construídas a partir de descrições, as quais especificam as condições que devem ser satisfeitas por um indivíduo para que ele possa ser um membro da classe. A formulação dessas descrições é explicada ao longo do tutorial.

## 4.Construção de uma ontologia OWL

Essa seção descreve a criação de uma ontologia de *Pizzas*. A idéia de utilizar *pizzas* reside no fato de que nesse domínio podem ser construídos bons exemplos.

### Exercício 2: criar um exemplo de projeto

1. Inicie o *Protege*. Caso já exista um projeto aberto, feche-o e reinicie o programa. Ao re-iniciar o *Protege*, a caixa de diálogo de boas vindas é apresentada, para que se possa criar um novo projeto, abrir um projeto recente ou obter ajuda.

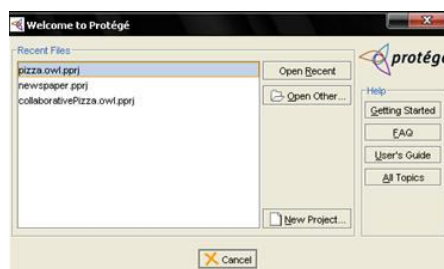


Figura Extra1: dialogo inicial do *Protege*

2. Clique em *New Project* (*Novo Projeto*) e uma nova caixa de diálogo será aberta: *Create New Project* (*Criar Novo Projeto*) permitindo a escolha do tipo de projeto. Selecione o tipo *OWL/RDF Files* (*Arquivos OWL/RDF*) e clique em *Finish* (*Terminar*).

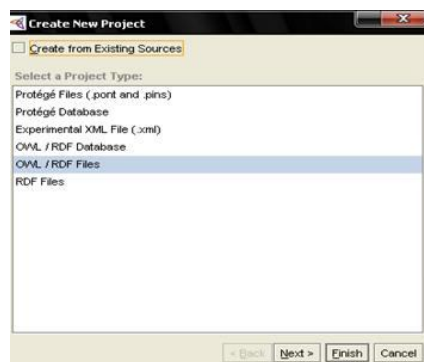
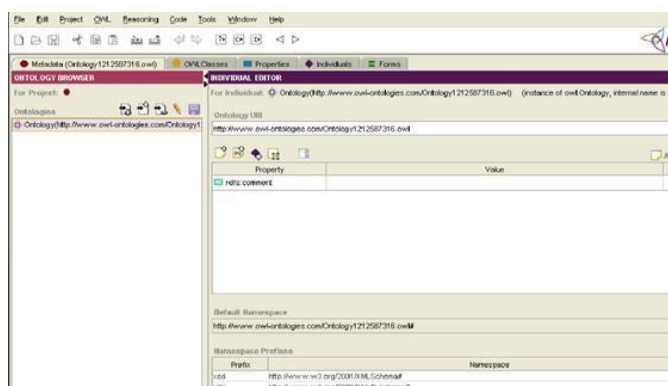


Figura Extra2: escolha do tipo de projeto

3. A janela do *Protege* é aberta e as *tabs* se tornam visíveis. Um novo projeto sempre é aberto na visão *Metadata* (*Metadados*).



### Figura Extra3: tela inicial (etiqueta *Metadata*) do Protege


4. Nomear e salvar o projeto.

É importante salvar o projeto, o que permite encerrar as atividades quando conveniente. Para salvar o projeto siga as instruções abaixo:

1. Clique no botão *Save Project* (*Salvar Projeto*), o terceiro da esquerda no menu superior do *Protege*. Pode-se escolher também *Save project* (*Salvar Projeto*) no menu *File* (*Arquivo*). A caixa de diálogo *Protege Files* (*Arquivos do Protege*) é aberta.



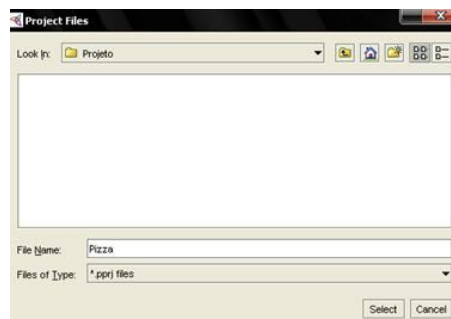
**Figura Extra4: dialogo para salvar projeto**

2. Escolha a local adequado para salvar o projeto, clique no botão  a direita da linha *Project* (*Projeto*). A caixa de diálogo de *Protege Files* (*Arquivos do Protege*) é aberta; navegue para selecionar ou para criar um diretório.



**Figura Extra5: dialogo para selecionar local a salvar projeto**

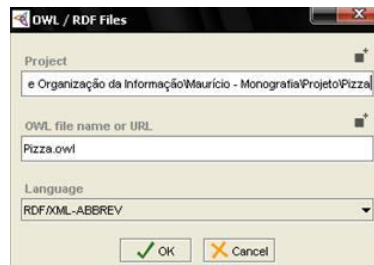
3. Entre com um nome de arquivo (por exemplo, *Pizza*).



**Figura Extra5: dialogo para selecionar local a salvar projeto**

4. Clique em *Select* (*Selecionar*).

5. Na caixa de diálogo de *Protege Files* (*Arquivos do Protege*) clique em *Ok* para salvar os arquivos e fechar a caixa de diálogo.



**Figura Extra6: dialogo para salvar projeto preenchido**

**Nota:** Pode-se também escolher uma localização digitando o caminho completo na linha *Project (Projeto)*. Os nomes dos outros arquivos serão preenchidos automaticamente.

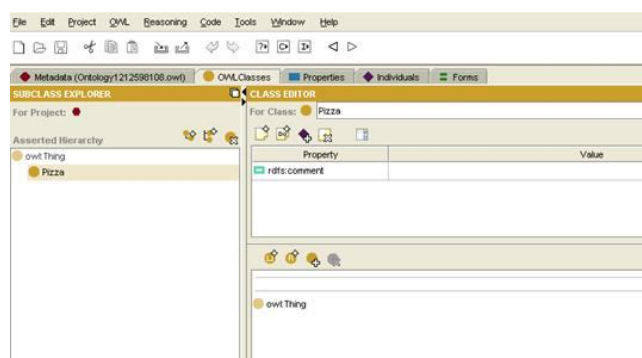
#### 4.1) Criação de classes

A janela principal do *Protege* consiste de *tabs* (etiquetas) que apresentam características do base de conhecimento. A etiqueta mais importante que surge ao se iniciar um projeto é a etiqueta *Classes*. Em geral, classes correspondem a objetos, ou a tipos de objetos no domínio. Por exemplo, em um jornal, as classes podem ser pessoas, tais como editores, repórteres e vendedores; componentes do *layout* do jornal, tais como seções; e conteúdo do jornal, tais como anúncios e artigos.

As classes no Protege são mostradas em uma hierarquia com heranças e apresentadas em um *Class Browser (Navegador de Classes)* do lado esquerdo da etiqueta *Classes*. As propriedades da classe selecionadas no momento são apresentadas no *Class Editor (Editor de Classes)*, à direita.

Nessa seção, o objetivo criar classes e subclasses, modificar a hierarquia de classes, criar classes abstratas, e adicionar superclasses adicionais a classes já existentes.




Uma nova ontologia contém uma classe chamada *owl:Thing*. Conforme mencionado, as classes *OWL* são interpretadas como conjuntos de indivíduos (ou conjunto de objetos). A classe *owl:Thing* é a classe que representa o conjunto que contém todos os indivíduos, uma vez que todas as classes são subclasses de *owl:Thing*.



**Figura 4.1: visão parcial da etiqueta Classes**



**Figura 4.2: o painel *Asserted Hierarchy (Hierarquia Declarada)***

	<i>Create subclass (Criar subclasse)</i>
	<i>Create sibling class (Criar classe irmã)</i>
	<i>Delete selected class (es) = (Apagar classes selecionadas)</i>

**Tabela 2: legenda dos botões do painel de Hierarquia de Classes**

### **Exercício 3: Criação das classes *Pizza*, *PizzaTopping* e *PizzaBase***

Para saber quais os tipos de pizza, os *PizzaTopping* (*Recheio de Pizza*) e as *PizzaBase* (*Bases de Pizza*) cria-se uma classe para cada um desses termos:

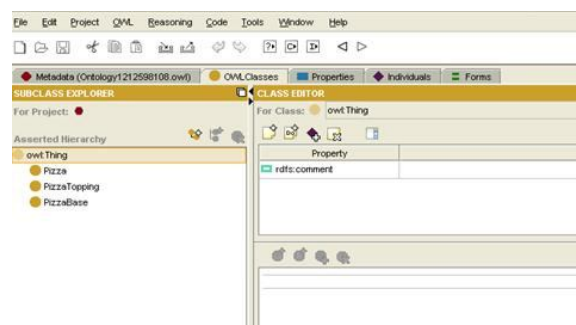
1. Selecione a etiqueta *Classes*.
2. Pressione o botão *Create subclass (Criar subclasse)*, conforme FIG.4.2. Este botão cria uma nova classe, como subclasse da classe selecionada (neste caso, está sendo criada uma subclasse de *owl:Thing*).
3. Renomeie a classe usando o campo *For class* (FIG. 4.3) localizado a direita da hierarquia de classes e pressione *Enter*.



**Figura 4.3: campo para nomear a classe (*For Class*)**

4. Repita o passo anterior para adicionar as classes *PizzaTopping* e também *PizzaBase*; a classe *owl:Thing* deve estar selecionada antes de pressionar *Create subclass (Criar subclasse)*, de forma que as classes sejam criadas como subclasses de *owl:Thing*.

A hierarquia de classes deve parecer como mostrada na FIG. 4.4:



**Figura 4.4: a hierarquia de classes inicial**

**VOCABULÁRIO** = A hierarquia de classe também pode ser chamada de *Taxonomia*.

**DICA** = Apesar de não existir uma regra obrigatória para nomear classes *OWL*, recomenda-se que todos os nomes de classes iniciem com letra maiúscula e não contenham espaços. Este tipo de notação é conhecida como *CamelBack*. Por exemplo: *Pizza*, *PizzaTopping*, *MargheritaPizza*. Pode-se usar o *underscore* ( *\_* ) para juntar palavras. Por exemplo, *Pizza\_Topping*. A regra é importante para a consistência da ontologia.

## 4.2)Classes disjuntas

Tendo adicionado as classes *Pizza*, *PizzaTopping* e *PizzaBase*, é preciso agora dizer que estas classes são *disjuntas*, de modo que um indivíduo (ou objeto) não poderá ser instância de mais de uma dentre as três classes. Para especificar as classes que serão disjuntas da classe selecionada, use a forma gráfica *Disjoints* (*Disjunção*), localizada no canto inferior direito da etiqueta *OWLClasses*.



Figura 4.5 Interface *Disjoints* (*Disjunções*)






	Create disjoint class from OWL expression (criar classe disjunta para expressão OWL)
	Add disjoint class (adicionar classe disjunta)
	Add all siblings (adicionar todas as classes irmãs)
	Remove all siblings (remover todas as classes irmãs)
	Delete selected row (apagar linha selecionada)

Tabela 3: botões da interface *Disjoints*

### Exercício 4: tornar disjuntas as classes *Pizza*, *PizzaTopping* e *PizzaBase*

1. Selecione a classe *Pizza* na hierarquia.
2. Clique no botão *Add all siblings* (adicione todas as irmãs), na interface *Disjoint*. Uma caixa de diálogo é aberta, como a mostrada na FIG.Extra7. Basta marcar a primeira opção e escolher Ok para finalizar. Isto tornará *PizzaBase* e *PizzaTopping* (as classes irmãs de *Pizza*) disjuntas de *Pizza*.

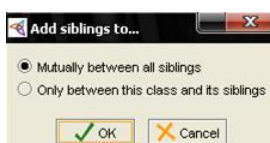


Figura Extra7: dialogo tipo de disjunção

Observe que a interface *Disjoint* (Disjunção) agora exibe as classes *PizzaTopping* e *PizzaBase*. Selecione a classe *PizzaBase* e note a interface *Disjoint* exibe agora as classes que disjuntas para *PizzaBase*, ou seja, *Pizza* e *PizzaTopping*.

**SIGNIFICADO** = Considera-se que as classes *OWL* se sobrepõem. Por isso, não se pode assumir que um indivíduo não é um membro de uma classe específica, simplesmente porque não se declarou que ele é um membro daquela classe. Para desconectar um grupo de classes é preciso torná-las disjuntas. Isto garante que um indivíduo que tenha sido declarado como sendo membro de uma das classes do grupo, não pode ser um membro de nenhuma outra classe naquele mesmo grupo. No exemplo (*Pizza*, *PizzaTopping* e *PizzaBase*) a disjunção foi realizada, o que significa que não é possível a um indivíduo ser membro de uma combinação destas classes. Não faz sentido um indivíduo ser uma *Pizza* e uma *PizzaBase*.

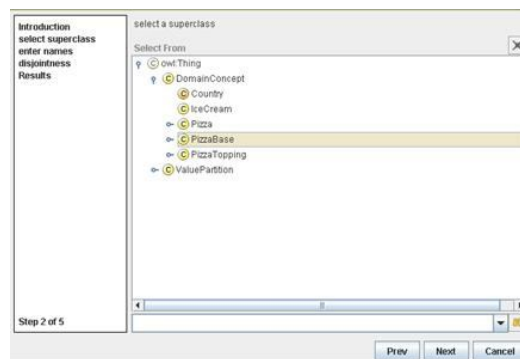
### 4.3)Uso do assistente OWL para criar classes

O *plug-in* *OWL Wizard* (*Assistente OWL*) consiste de um conjunto de assistentes, projetados para executar tarefas comuns, repetitivas e demoradas. Nesta seção usa-se o assistente *Create Multiple Classes* (*Criar múltiplas classes*) para adicionar subclasses a classe *PizzaBase*. Para utilizar o *Assistente OWL* é necessário que o *plug-in* correspondente esteja instalado.

#### **Exercício 5: use o Assistente *Create Multiple Classes* (Criar múltiplas classes) para criar *ThinAndCrispy* (FinaECrocante) e *DeepPan* (BaseGrossa) como subclasses de *PizzaBase***

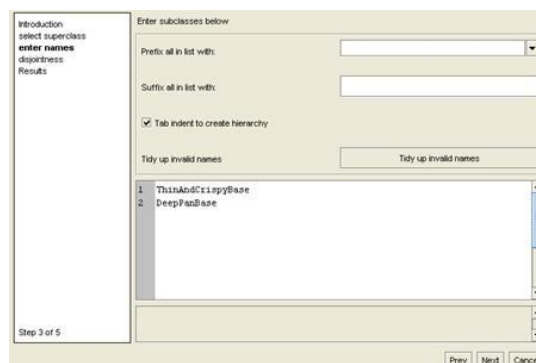
---

1. Selecione a classe *PizzaBase* na hierarquia de classes.
2. No menu *Tools*, selecione o comando *Quick OWL* e a opção *Create Multiple Subclasses* (*Criar múltiplas classes*). Pressione *Next*.
3. O assistente é apresentado como FIG.4.6. Selecione a classe *PizzaBase* (a qual é subclasse de *Domain Concept*).



**Figura 4.6:** tela do assistente para criar múltiplas classes

4. Pressione o botão *Next* no assistente. abaixo é exibida. Agora, é preciso dizer ao assistente qual subclasses de *PizzaBase* deve ser criada. Na área de texto maior, digite no nome da classe *ThinAndCrispyBase* (*PizzaDeBaseFina*) e tecle *enter*. Digite o nome de classe *DeepPanBase* (*BaseGrossa*). A tela deve estar parecida com a da FIG.4.7. a seguir.



**Figura 4.7:** tela do assistente para criar múltiplas classes

5. Clique no botão *Next* do Assistente, e os nomes digitados são verificados quanto ao estilo (letras maiúsculas/minúsculas, sem espaços, etc.). Também é verificada a unicidade das classes, ou seja, duas classes não podem ter o mesmo nome. Erros no nome das classes, são apresentados na tela, juntamente com sugestões para correção.

6. Clique no botão *Next* no assistente. Selecione a opção *Make all primitive siblings disjoint?* (*Marcar todas as irmãs primitivas como disjuntas?*). Ao invés de utilizar a interface das classes disjuntas, o assistente tornará as novas classes disjuntas automaticamente.

7. Clique no botão *Next* para visualizar e adicionar anotações. Em geral, as anotações são usadas para gravar dados sobre a edição da ontologia: quem e quando a criou, quando foi revisada, etc. As anotações básicas de propriedades OWL são selecionadas por *default*. Nenhuma dado será informado nesse momento. Pressione *Finish*.

**DICA** = caso tenha sido importada a ontologia *DC-Dublin Core* (mais detalhes adiante), as propriedades de anotação *DC* podem estar disponíveis para anotação das classes no passo 7 do exercício 5. O *DC* é um conjunto de metadados, que pode ser usado para anotações em ontologias de dados tais como *creator* (*criador*), *date* (*data*), *language* (*língua*), etc.

Depois de pressionar *Finish*, o assistente cria as classes, as torna disjuntas, e as seleciona na etiqueta *OWLClasses*. A ontologia tem *ThinAndCrispyBase* e *DeepPanBase* como subclasses de *PizzaBase*. Essas novas classes são disjuntas, e por isso, a *BasePizza* (*base da pizza*) não pode ser uma *ThinAndCrispyBase* (*base fina e torrada*) e uma *DeepPanBase* (*base grossa*) ao mesmo tempo. No caso de muitas classes a adicionar, o assistente acelera o processo.

**DICA** = na segunda tela do assistente (*Create Multiple Subclasses*), as classes são criadas e inseridas. Caso existam muitas classes a criar com o mesmo prefixo ou sufixo, é possível usar as opções de auto-anexar no início e auto-anexar no final em conjunto com os nomes de classes inseridos.

## Criação de recheios para *Pizza*

Com algumas classes básicas inseridas, vai-se criar recheios de *Pizza*. Os recheios são agrupados em categorias: *meat toppings* (*recheios de carne*), *vegetable toppings* (*recheios de vegetais*), *cheese toppings* (*recheios de queijo*) e *seafood toppings* (*recheios de frutos do mar*).

### Exercício 6: criação de recheios para *pizzas*

---

1. Selecione a classe *PizzaTopping* na hierarquia de classes.

2. Use o *OWL-Wizard* (*Assistente OWL*) e adicione as seguintes subclasses de *PizzaTopping*:

- MeatTopping* (*RecheioDeCarne*);
- VegetableTopping* (*RecheioDeVegetais*);
- CheeseTopping* (*RecheioDeQueijo*)
- SeafoodTopping* (*RecheioDeFrutosDoMar*).

As classes devem estar disjuntas.

3. Em seguida, adicione diferentes recheios de carne. Selecione a classe *MeatTopping* (*RecheioDeCarne*), e use o assistente *Create Multiple Subclasses* (*Assistente para criar múltiplas classes*) para adicionar as seguintes subclasses:

- SpicyBeefTopping* (*RecheioDeCarneApimentada*)
- PepperoniTopping* (*RecheioDeCalabresa*)
- SalamiTopping* (*RecheioDeSalame*)
- HamTopping* (*RecheioDePresunto*)

Certifique que as classes estão classes disjuntas.



4. Adicione alguns tipos de recheios de vegetais criando as seguintes subclasses disjuntas de *VegetableTopping* (*RecheioDeVegetais*):

*TomatoTopping* (*RecheioDeTomate*);  
*OliveTopping* (*RecheioDeAzeitona*);  
*MushroomTopping* (*RecheioDeCogumelo*);  
*PepperTopping* (*RecheioDePimenta*);  
*OnionTopping* (*RecheioDeCebola*)  
*CaperTopping* (*RecheioDeAlcaparras*).

Adicione seguintes classes como subclasses de *PepperTopping* (*RecheioDePimenta*):

*RedPepperTopping* (*RecheioDePimentaVermelha*)  
*GreenPepperTopping* (*RecheioDePimentaVerde*)  
*JalapenoPepperTopping* (*RecheioDePimentaMexicana*)  
As subclasses de *PepperTopping* devem ser disjuntas.

5. Agora adicione alguns recheio de queijo. Assim como feito anteriormente, adicione as seguintes subclasses de *CheeseTopping* (*RecheioDeQueijo*), assegurando-se de que as subclasses são disjuntas:

*MozzarellaTopping* (*RecheioDeMussarela*)  
*ParmezanTopping* (*RecheioDeParmesão*)

5. Finalmente, adicione subclasses de *SeafoodTopping* (*RecheioDeFrutosDoMar*):

*TunaTopping* (*RecheioDeAtum*)  
*AnchovyTopping* (*RecheioDeAnchova*)  
*PrawnTopping* (*RecheioDeCamarão*)

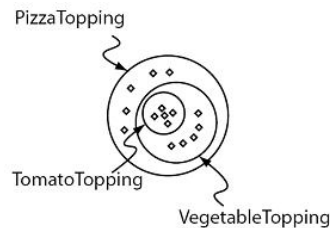
Nesse momento, a hierarquia de classes deve parecer com a apresentada na FIG.4.8 abaixo (o ordenamento das classes podem estar ligeiramente diferente).



Figura 4.8: hierarquia de classes

**SIGNIFICADO** = Até agora, foram criados *classes nomeadas* simples, algumas das quais são subclasses de outras. A construção da hierarquia de classes pode parecer intuitiva. Contudo, o que realmente significa ser subclasse de alguma coisa em OWL? Por exemplo, o que significa

para *VegetableTopping* (*RecheioDeVegetal*) ser subclasse de *PizzaTopping* (*RecheioDePizza*), ou para *TomatoTopping* (*RecheioDeTomate*) ser subclasse de *VegetableTopping* (*RecheioDeVegetais*)? Em OWL, ser uma subclasse significa uma *implicação necessária*. Em outras palavras, se *VegetableTopping* é uma subclasse de *PizzaTopping* então TODAS as instâncias de *VegetableTopping* são instâncias de *PizzaTopping*, sem exceção. Se alguma coisa é um *VegetableTopping*, isto implica que também é um *PizzaTopping*. Veja a FIG. 4.9.

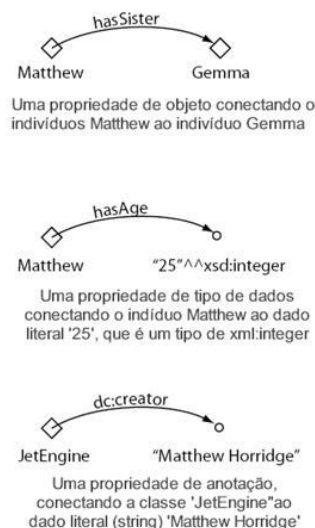


**Figura 4.9:** O significado de ser uma subclasse - todos os indivíduos que são membros da classe *TomatoTopping* são membros da classe *VegetableTopping* e *PizzaTopping*, uma vez que se estabeleceu que *TomatoTopping* é subclasse de *VegetableTopping*, que por sua vez é subclasse de *PizzaTopping*.

#### 4.4) Propriedades OWL

As propriedades OWL representam relacionamentos entre dois indivíduos. Existem dois tipos principais de propriedades: *Object Properties* (*Propriedades de Objeto*) e *DataType Properties* (*Propriedades de Tipos de Dados*). As *Object Properties* (*Propriedades de Objeto*) conectam um indivíduo a outro indivíduo. As *DataType Properties* (*Propriedades de Tipos de Dados*), por sua vez, conectam um indivíduo a um valor do *XML-Schema Datatype* (disponível em <http://www.w3.org/TR/xmlschema-2/>) ou a um literal do *RDF-Resource Description Framework* (disponível em <http://www.w3.org/TR/rdf-primer/>).

O OWL também tem um terceiro tipo de propriedade, denominada *Annotation Property* (*Propriedade de Anotação*), as quais são usadas para adicionar metadados as classes, aos indivíduos e as *Object Properties* (*Propriedades de Objeto*) e as *DataType Properties* (*Propriedades de Tipos de Dados*). A FIG. 4.10 apresenta um exemplo de cada tipo de propriedade.



**Figura 4.10:** diferentes tipos de propriedades OWL

As propriedades podem ser criadas usando a etiqueta *Properties (Propriedades)*, mostrada na FIG. 4.11. Para criar propriedades OWL a partir etiqueta *Properties* da utilize o botão localizado no canto superior esquerdo. Como se pode ver na FIG. 4.13, existem botões para criação de *propriedades de tipos de dados*, *propriedades de objeto* e *propriedades de anotação*. A maioria das propriedades criadas neste tutorial são *propriedades de objeto*. Pode-se também pode criar propriedades a partir da etiqueta *OWLClasses*, usando a interface *Properties*, apresentada na FIG.4.12.

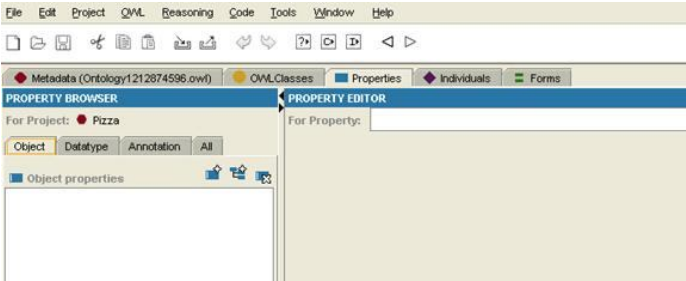


Figura 4.11: a etiqueta *Properties*



Figura 4.12: a interface *Properties* na etiqueta *OWL Classes*



Figura 4.13: botões para a criação *Properties (Propriedades)*

	Create datatype property (Criar propriedade de tipos de dados)
	Create object property (Criar propriedade de objeto)
	Create Subproperty (Criar subpropriedade)
	Create annotation datatype property (Criar propriedade de anotação tipo de dados)
	Create annotation object property (Criar propriedade anotação objeto)
	Create annotation property (Criar propriedade de anotação)
	Delete properties (apagar propriedades)

Tabela 5: botões para criação de *Properties (Propriedades)*

**DICA** = Embora não existam regras únicas para nomear propriedades, recomenda-se que os nomes das propriedades comecem com letra minúscula, sem espaço, e que a primeira letra da próxima palavra seja uma maiúscula. Recomenda-se também que as propriedades tenham como prefixo a palavra *has* (*tem*), ou a palavra *is* (*é*), por exemplo, *hasPart* (*temParte*), *isPartOf* (*éParteDe*), *hasManufacturer* (*temFabricante*), *IsProducerOf* (*éProdutoDe*). Essas convenções ajudam não só a facilitar o entendimento sobre as propriedades, como também permitem usar a *English Prose TooltipGenerator* (ferramenta geradora de dicas em inglês), a qual se utiliza dessa convenção para gerar expressões legíveis e descrever classes. A ferramenta exibe a descrição da classe em linguagem natural (inglês), facilitando o entendimento de uma descrição

de classe. Ela é ativada quando o cursor é colocado sobre a descrição da classe na interface do usuário.

Tendo adicionado a propriedade *hasIngredient* (*temIngrediente*), adicionam-se em seguida mais duas propriedades: *hasTopping* (*temRecheio*) e *hasBase* (*TemBase*). As propriedades podem ter subpropriedades, de modo que se formam hierarquias de propriedades. As subpropriedades especializam superpropriedades da mesma forma que subclasses se diferenciam das superclasses. Por exemplo, a propriedade *hasMother* (*temMãe*) é diferente da propriedade mais geral *hasParent* (*temPais*). No caso da ontologia de *pizza*, as propriedades *hasTopping* e *hasBase* devem ser criadas como subpropriedades de *hasIngredient* (*temIngrediente*). Se a propriedade *hasTopping* (ou *hasBase*) conecta dois indivíduos, isto implica que os dois indivíduos estão relacionados também pela propriedade *hasIngredient*.

## Exercício 7: criar uma propriedade de objeto chamada *hasIngredient*

1. Na etiqueta *Properties* (*Propriedades*), use o botão *Create Object Property* (*Criar Propriedade Objeto*) para criar uma nova *propriedade*. Uma *Object Property* (*propriedade de objeto*) com um nome genérico é criada.
2. Renomeie a propriedade para *hasIngredient* (*TemIngrediente*) no campo *For Property*, conforme apresentado na FIG. 4.14.



Figura 4.14: criação da propriedade *hasIngredient*

## Exercício 8: criação de subpropriedades de *hasIngredient*: *hasTopping* e *hasBase*

1. Para criar a propriedade *hasTopping* (*temRecheio*) como subpropriedade de *hasIngredient* (*temIngrediente*) clique com o botão direito do mouse sobre a propriedade *hasIngredient*, na hierarquia de propriedades da etiqueta *Properties*. O menu apresentado na FIG. 4.15 será aberto:

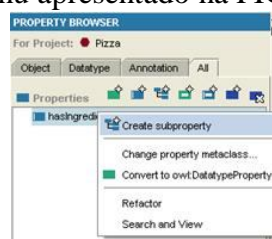


Figura 4.15: menu da hierarquia de propriedades

2. Selecione o item *Create subproperty* (*Criar subpropriedade*) do menu, criando uma nova *propriedade de objeto* como subpropriedade de *hasIngredient* (*temIngrediente*).

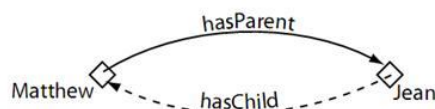
3. Renomeie a nova propriedade para *hasTopping* (*temRecheio*).

4. Repita o passo acima, usando agora o nome de propriedade *hasBase* (*temBase*).

Observe que é possível criar subpropriedades de *propriedades de tipos de dados*. Contudo, não é possível combinar *propriedades de objetos* e *propriedades de tipos de dados* nas subpropriedades. Por exemplo, não é possível criar uma *propriedade de objeto* que seja a subpropriedade de uma *propriedade de tipos de dados* e vice-versa.

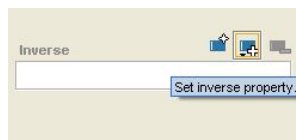
#### 4.5) Propriedades Inversas

Uma *propriedade de objeto* tem uma propriedade inversa correspondente. Se um propriedade liga um indivíduo "a" a um indivíduo "b", então a propriedade inversa correspondente liga o indivíduo "b" ao indivíduo "a". Por exemplo, a FIG. 4.16 mostra a propriedade *hasParent* (*temPais*) e sua propriedade inversa *hasChild* (*temFilho*): se *Matthew hasParent Jean*, da propriedade inversa pode-se inferir que *Jean hasChild Matthew*.



**Figura 4.16: exemplo de propriedade inversa: *hasParent* tem com inversa *hasChild***

Propriedades inversas são criadas na interface *Inverse Property* (*Propriedade Inversa*) conforme apresentado na FIG. 4.17.



**Figura 4.17: Interface de propriedade inversa**

Especifique as propriedades inversas das propriedades existentes na ontologia de *Pizza*.

### **Exercício 9: criação de propriedades inversas**

1. Use o botão *Create object property* na etiqueta *Properties* para criar uma nova *propriedade de objeto* chamada *isIngredientOf* (*éIngredientDe*), inversa de *hasIngredient* (*temIngredient*).

2. Pressione o botão *Set inverse property* (*Configurar propriedade inversa*) na interface da *Inverse Property* (*propriedade inversa*), conforme apresentado na mostrada na FIG. 4.17. Na caixa de diálogo apresentada, selecione a propriedade *hasIngredient* e pressione *Ok*. A propriedade *hasIngredient* (*temIngredient*) aparece agora na interface *Inverse Property* (*Propriedade Inversa*). A hierarquia de propriedades indica que *hasIngredient* e *isIngredientOf* são propriedades inversas.

3. Selecione a propriedade *hasBase* (*temBase*).

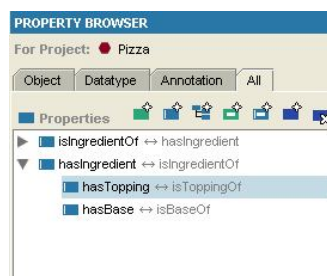
4. Pressione o botão *Create new inverse property* (*Criar nova propriedade inversa*) na interface *Inverse Property* (*Propriedade Inversa*). Uma caixa de diálogo apresenta dados sobre a propriedade mais recente. Use a caixa de diálogo para renomear a propriedade *isBaseOf* (*éBaseDe*) e então feche-a. Observe que a

propriedade *isBaseOf* (*éBaseDe*) foi criada como subpropriedade de *isIngredientOf*. Isto ocorre porque *hasBase* (*temBase*) é uma subpropriedade de *hasIngredient* (*temIngrediente*), e *isIngredientOf* (*éIngredienteDe*) é a propriedade inversa de *hasIngredient* (*temIngrediente*).

5. Selecione a propriedade *hasTopping* (*temRecheio*).

6. Pressione o botão *Create new inverse property* (*Criar uma nova propriedade inversa*) na interface *Inverse Property* (*Propriedade Inversa*). Na caixa de diálogo apresentada, renomeie a propriedade *isToppingOf* (*éRecheioDe*). Feche a caixa de diálogo e observe que *isToppingOf* (*éRecheioDe*) é sub-propriedade de *isIngredientOf* (*éIngredienteDe*).

A hierarquia de propriedades deve parecer com a FIG. 4.18. Observe a seta ‘bidirecional’ que indica a propriedade inversa.



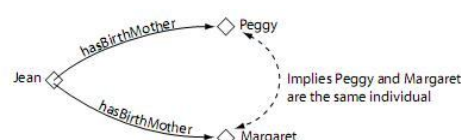
**Figura 4.18: propriedades inversas**

#### 4.6) Características das propriedades OWL

O OWL permite enriquecer o significado das propriedades, através do uso de *Property characteristics* (*características das propriedades*). As seções seguintes discutem as várias características que as propriedades podem ter.

##### 4.6.1) Propriedades funcionais

Se uma propriedade é *funcional*, para um determinado *indivíduo 1*, pode existir até no máximo um *indivíduo 2* que está relacionado ao *indivíduo 1* através dessa propriedade. A FIG. 4.19 apresenta um esquema de exemplo para a propriedade funcional *hasBirthMother* (*TemMãeBiológica*): alguém só pode nascer de uma única mãe. Se *Jean hasBirthMother Peggy*, e *Jean hasBirthMother Margaret*, então *hasBirthMother* é uma *propriedade funcional*, ou seja, *Peggy* e *Margaret* são mesma pessoa. Contudo, observe que se *Peggy* e *Margaret* são descritos explicitamente como duas pessoas diferentes, então as afirmações anteriores levam a uma inconsistência.

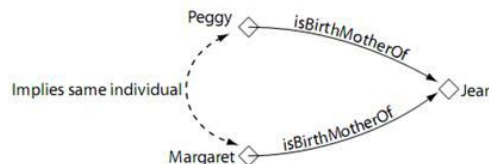


**Figura 4.19: exemplo da propriedade funcional *hasBirthMother* (*temMãeBiológica*)**

**VOCABULÁRIO** = *Propriedades Funcionais* também são conhecidas como *Single Value Properties* (*Propriedades de Valor Único*) ou *Features* (*Características*).

#### 4.6.2) Propriedades funcionais inversas

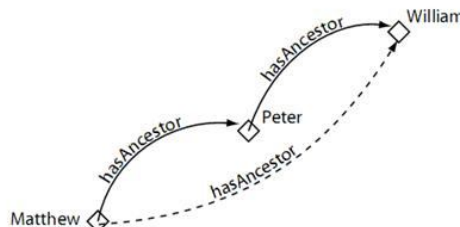
Se uma propriedade é uma *funcional inversa*, isto significa que a sua propriedade inversa é funcional. Para o *indivíduo1*, pode existir no máximo um indivíduo relacionado ao *indivíduo1* através da propriedade. A FIG. 4.20 mostra um exemplo da propriedade funcional inversa *isBirthMotherOf* (*éMãeBiológicaDe*), que é a propriedade inversa de *hasBirthMother* (*temMãeBiológica*). Se *hasBirthMother* (*temMãeBiológica*) é funcional, *isBirthMotherOf* (*éMãeBiológicaDe*) é funcional inversa. Se *Peggy* é a mãe natural de *Jean*, e *Margareth* é a mãe natural de *Jean*, infere-se que *Peggy* e *Margareth* correspondem a mesma pessoa.



**Fig. 4.20: Exemplo de uma propriedade funcional inversa**

#### 4.6.3) Propriedades transitivas

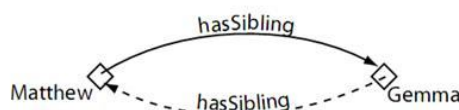
Se uma propriedade *P transitiva* relaciona o indivíduo "a" ao indivíduo "b", e também um indivíduo "b" ao indivíduo "c", infere-se que o indivíduo "a" está relacionado ao indivíduo "c" através da propriedade *P*. Por exemplo, a FIG. 4.21 mostra um exemplo da propriedade transitiva *hasAncestor* (*temAncestral*). Se o indivíduo *Matthew* tem o ancestral *Peter*, e *Peter* tem o ancestral *William*, então *Mathew* tem um ancestral que é *Willian*. Esse fato é indicado pela linha tracejada na FIG. 4.21.



**Figura: 4.21: exemplo de propriedade transitiva *hasAncestor* (*temAncestral*)**

#### 4.6.4) Propriedades simétricas

Se uma propriedade *P* é *simétrica*, e relaciona um indivíduo "a" ao indivíduo "b", então o indivíduo "b" também está relacionado ao indivíduo "a" através da propriedade *P*. A FIG. 4.22 mostra um exemplo. Se o indivíduo *Matthew* está relacionado ao indivíduo *Gemma* através da propriedade *hasSibling* (*temIrmão*), então *Gemma* também está relacionada a *Matthew* através da propriedade *hasSibling*. Em outras palavras, se *Matthew* tem uma irmã *Gemma*, então *Gemma* tem um irmão que é *Matthew*. Dito de outra forma, a propriedade é a própria inversa.



**Figura 4.22: exemplo da propriedade simétrica *hasSibling* (*temIrmão*)**



Deseja-se tornar *hasIngredient* (*temIngrediente*) uma propriedade transitiva, de modo que, por exemplo, se um recheio de *pizza* tem um ingrediente, então uma *pizza* do mesmo recheio deve que ter o mesmo ingrediente. Para definir as *características* da propriedade, utiliza-se o campo de *Property Characteristics* (*Característica da Propriedade*) conforme a FIG. 4.23, a qual está localizada no canto inferior direito da etiqueta *Property*.



**Figura 4.23: Interface *Property Characteristics* (*Característica da Propriedade*)**

### **Exercício 10: tornar *hasIngredient* (*temIngrediente*) uma propriedade transitiva**

---

1. Selecione a propriedade *hasIngredient* na hierarquia de propriedades, etiqueta *Properties*.
2. Assinale a opção *Transitive* no *Property Editor* (*Editor de Propriedades*), no canto direito acima da interface *Inverse*.
3. Selecione a propriedade *isIngredientOf*, que é a inversa de *hasIngredient*. Confirme que *Transitive* está marcado.

**OBSERVAÇÃO** = se uma propriedade é *transitiva*, então a propriedade *inversa* a ela também é *transitiva*. Essa operação é feita manualmente no *Protege-OWL*. Contudo, o MI assume que se a propriedade é transitiva, a propriedade inversa também é transitiva.

**ATENÇÃO** = se uma propriedade é transitiva ela não pode ser funcional, uma vez que a propriedade transitiva, por sua própria natureza, pode formar cadeias de indivíduos.

Deseja-se afirmar que uma *Pizza* pode ter apenas uma *base*. Existem várias formas para fazer isso. Escolhe-se tornar *hasBase* uma propriedade funcional, de modo que ela possa ter apenas um valor para um determinado indivíduo.

### **Exercício 11: tornar funcional a propriedade *hasBase***

---

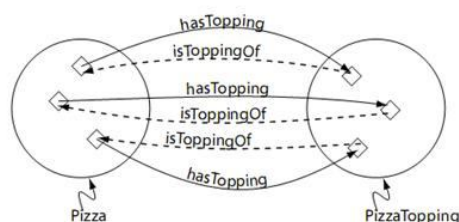
1. Selecione a propriedade *hasBase* (*temBase*).
2. Marque *Functional* na interface *Property Characteristics* (*Características da Propriedade*), caso esteja desmarcada.



**OBSERVAÇÃO** = se uma *datatype property* (*propriedade de tipos de dados*) está selecionada, a interface *Property Characteristics* (*Características da Propriedade*) é reduzida de modo que apenas as opções *Functional* e *Inverse Functional* sejam exibidas. O *OWL-DL* não permite que uma *datatype property* seja *transitiva*, *simétrica* ou tenha uma *propriedade inversa*.

#### 4.7) Domains (domínios) e Ranges (escopo) de uma propriedade

Uma propriedade possui *domain* (*domínio*) e *range* (*escopo*). As propriedades conectam indivíduos de um *domain* (*domínio*) a indivíduos de um *range* (*escopo*). Por exemplo, na ontologia de *Pizza*, a propriedade *hasTopping* (*temRecheio*) liga indivíduos pertencentes a classe *Pizza* a indivíduos pertencentes a classe *PizzaTopping* (*RecheioDePizza*). Neste caso, o *domain* (*domínio*) da propriedade *hasTopping* é *Pizza* e o *range* (*escopo*) é *PizzaTopping* (*RecheioDePizza*), conforme apresentado na FIG. 4.24.



**Figura 4.24:** o *domain* (*domínio*) e o *range* (*escopo*) para a propriedade *hasTopping* (*temRecheio*) e suas propriedades inversas *isToppingOf* (*éRecheioDe*). O *domain* para *hasTopping* é *Pizza* e a *range* para *hasTopping* é *PizzaTopping* (*RecheioDePizza*). O *domain* e a *range* para *isToppingOf* são o *domain* e a *range* para *hasTopping*.

**ATENÇÃO** = *Domains* (*domínios*) e *Ranges* (*escopos*) em OWL não são restrições sujeitas a verificação e são utilizados como axiomas em inferências. Por exemplo, se a propriedade *hasTopping* tem o conjunto *domínio* *Pizza* e aplica-se a propriedade *hasTopping* a *IceCream* (indivíduos membros da classe *IceCream*), o resultado pode ser um erro. É possível inferir que a classe *IceCream* é subclasse de *Pizza* (um erro é gerado através do MI, apenas se *Pizza* for disjunta de *IceCream*).

Deseja-se especificar que a propriedade *hasTopping* (*temRecheio*) tem um *range* (*escopo*) *PizzaTopping* (*RecheioDePizza*). Para tal, usa-se a interface *Range* (*Escopo*). O menu suspenso do *Protege* assume *Instance* (*Instância*) como padrão, indicando que a propriedade conecta instâncias de classes a instâncias de classes.

#### Exercício 12: especificar *range* (*escopo*) da relação *hasTopping*

1. Selecione a propriedade *hasTopping* (*temRecheio*) na hierarquia de propriedades da etiqueta *Properties*.
2. Pressione o botão *Specialise Range* (*Especializar escopo*) na interface *Range* (*Escopo*). Surge uma caixa de diálogo que permite selecionar a classe na hierarquia de classes, conforme FIG.4.25.
3. Selecione *PizzaTopping* (*RecheioDePizza*) e pressione *Ok*. A relação *PizzaTopping* deve ser exibida na lista de *range* (*escopo*).



Figura 4.25: interface para *Range* (escopo)

**ATENÇÃO** = também é possível, mas não recomendável, indicar que uma classe e não seus indivíduos são range de uma propriedade. É de um erro pensar que o *range* de uma propriedade é uma classe, quando um *range* corresponde na verdade aos indivíduos membros da classe. Ao especificar o *range* de uma propriedade como uma classe, trata-se tal classe como um indivíduo. Isto é um tipo de meta-declaração, e pode levar a ontologia para o *OWL-Full*.

**OBSERVAÇÃO** = é possível especificar várias classes como *range* de uma propriedade. Caso isso seja feito no *Protege-OWL*, o *range* da propriedade é interpretada como uma união das classes. Por exemplo, se uma propriedade tem as classes *Man* (homem) e *Woman* (mulher) listadas na interface *range*, isso significa que a *range* daquela propriedade será interpretada como *Man união com Woman*

### **Exercício 13: especificar domínio da propriedade *hasTopping* como *Pizza***

1. Selecione a propriedade *hasTopping* na hierarquia da etiqueta *Properties*.
2. Pressione *Specialise domain* (Especialize domínio) na interface *Domain*. Surge uma caixa de diálogo para seleção de uma classe da hierarquia, conforme FIG. 4.26.
3. Selecione *Pizza* e pressione *Ok*. A classe *Pizza* agora é exibida na lista de domínio.

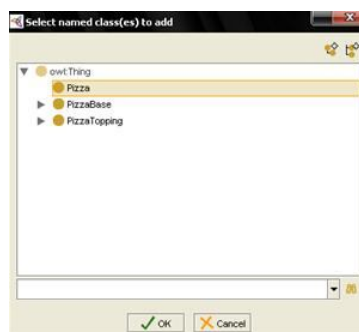


Figura 4.26: Interface *domain* (domínio) da propriedade

**SIGNIFICADO** = os indivíduos citados "do lado esquerdo" da propriedade *hasTopping* são membros da classe *Pizza*; os indivíduos citados "do lado direito" da propriedade *hasTopping* são membros da classe *PizzaTopping*. Por exemplo, sejam os indivíduos "a" e "b" e uma declaração

"a *hasTopping* b": infere-se que "a" é um membro da classe *Pizza* e que "b" é um membro da classe *PizzaTopping*.

**OBSERVAÇÃO** = quando diversas classes são especificadas como *domain* (*domínio*) de uma propriedade, o *Protege-OWL* interpreta o *domain* (*domínio*) da propriedade como a união dessas classes.

**ATENÇÃO** = embora a OWL permita o uso de *class expressions* (*expressões de classes*) subjetivas para o *domain* (*domínio*) de uma propriedade, isto não é permitido durante a edição de ontologias no *Protege-OWL*.

Deseja-se preencher o *domain* (*domínio*) e o *range* (*escopo*) para o inverso da propriedade *hasTopping* (*temRecheio*) e para *isToppingOf* (*éRecheioDe*).

#### **Exercício 14: especificar *domain* (*domínio*) e *range* (*escopo*) para *isToppingOf***

1. Selecione a propriedade *isToppingOf*.
2. Utilize os mesmos passos, explicados anteriormente, para definir o *domain* (*domínio*) da propriedade *isToppingOf* para *PizzaTopping*.
3. Defina *Pizza* como *range* (*escopo*) da propriedade *isToppingOf*.

Observe que o *domain* (*domínio*) da propriedade *isToppingOf* é o *range* (*escopo*) da propriedade inversa *hasTopping*, e que a *range* da propriedade *isToppingOf* é o *domain* da propriedade *hasTopping*.

#### **Exercício 15: especifique *domain* (*domínio*) e *range* (*escopo*) para a propriedade *hasBase* (*temBase*) e a sua propriedade inversa *isBaseOf* (*éBaseDe*)**

1. Selecione a propriedade *hasBase*.
2. Especifique *Pizza* como o domínio da propriedade *hasBase*.
3. Especifique *PizzaBase* como o escopo da propriedade *hasBase*.
4. Selecione a propriedade *isBaseOf*.
5. Defina *PizzaBase* como o domínio da propriedade *isBaseOf*.
6. Defina *Pizza* como o escopo da propriedade *isBaseOf*.

**DICA** = é preciso garantir que o domínio e o escopo para as propriedades estão também

configurados para as propriedades inversas de maneira correta. Em geral, o domínio para uma propriedade é o escopo de seu inverso, e o escopo para uma propriedade é o domínio de sua inversa.

**ATENÇÃO** = embora se tenha especificado domínios e escopos de várias propriedades para o presente tutorial, não se recomenda que esse procedimento seja rotineiro. As condições de domínio e de escopo não se comportam como restrições e, além disso, podem causar resultados inesperados na classificação. Esses problemas e seu efeitos indesejados são de difícil localização em uma grande ontologia.

## 4.8) Descrição e definição de classes

Após a criação de algumas propriedades pode-se agora utilizá-las para definir e descrever as classes da ontologia de *Pizza*.

### 4.8.1) Restrições de propriedades

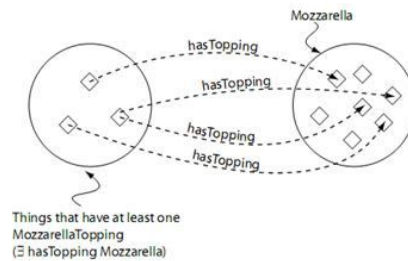
Em OWL, as propriedades são usadas para criar restrições. Tal como o nome sugere, restrições são utilizadas para restringir os indivíduos de uma classe. As restrições OWL são classificadas em três categorias principais:

- *Quantifier Restrictions* (*Restrições de Quantificador*)
- *Cardinality Restrictions* (*Restrições de Cardinalidade*)
- *Restrições hasValue*. (*Restrições temValor*)

As *Restrições de Quantificador* são compostas por um quantificador, uma propriedade e uma classe nomeada que contém indivíduos os quais atendem a restrição (denominada, *filler*). Os dois quantificadores disponíveis são:

- O *quantificador existencial* (E): lê-se como *pelo menos um*, ou *algum*; em OWL também pode ser lido como *someValuesFrom* (*algunsValoresDe*);
- O *quantificador universal* (A): lê-se como *apenas*; em OWL também pode ser lido como *allValuesFrom* (*todosValoresDe*).

Por exemplo, a restrição E *hasTopping MozzarellaTopping* é constituída pelo *quantificador existencial* E, pela propriedade *hasTopping*, e pelo *filler* *MozzarellaTopping*. Esta restrição descreve o conjunto, ou a classe, de indivíduos que tem *pelo menos um* recheio, e esse recheio é um indivíduo da classe *MozzarellaTopping*. Esta restrição é representada na FIG. 4.27: os símbolos em forma de diamante representam indivíduos. Note que a restrição descreve uma classe anônima de indivíduos que satisfazem a restrição.



**Figura4.27:** A restrição E *hasTopping MozzarellaTopping* descrevendo a classe de indivíduos que tem pelo menos um recheio que é *Mozzarella*

**SIGNIFICADO** = uma restrição descreve uma *classe anônima* (não nomeada), a qual é composta de indivíduos que satisfazem a restrição (vide o Apêndice A para detalhes sobre quantificação existencial e universal). Na realidade, quando restrições são usadas para descrever classes, elas especificam superclasses anônimas da classe que está sendo descrita. Por exemplo, pode-se dizer que *MargheritaPizza* é uma subclasse de *Pizza* (dentre outras coisas), e também uma subclasse das coisas que tem pelo menos um recheio que é *MozzarellaTopping*.

As restrições de uma classe são exibidas e editadas na interface *Asserted Conditions* (*Condições Declaradas*) apresentada na FIG. 4.28. A interface *Asserted Conditions* é a parte mais importante da etiqueta *Classes* do Protégé-OWL, uma vez que detém praticamente todas as informações para descrição de uma classe.



**Figura 4.28:** a interface *Conditions* da etiqueta *Classes*

As restrições são usadas em descrições de classes OWL, para especificar superclasses anônimas daquelas classes a serem descritas.

#### 4.8.2) Restrições existenciais

*Restrições Existenciais* (E) são o tipo mais comum de restrição em ontologias OWL. Para um conjunto de indivíduos, uma *restrição existencial* especifica a existência de um relacionamento (ou seja, *pelo menos um*) de um desses indivíduos com outro indivíduo, o qual é membro de uma classe específica, através da propriedade.

Por exemplo, E *hasBase PizzaBase* descreve todos os indivíduos que tem *pelo menos um* relacionamento com um indivíduo membro da classe *PizzaBase*, através da propriedade *hasBase*. Em linguagem natural: todos os indivíduos que tem pelo menos uma base de *pizza*.

**VOCABULÁRIO** = a *Restrição Existencial* também é conhecida como *Some Restrictions* (*Algumas Restrições*).

## Exercício 16: Adicionar restrição a *Pizza* de forma que *Pizza* tenha uma *PizzaBase*

---

1. Selecione *Pizza* na hierarquia de classes da etiqueta *Classes*.
2. Selecione *NECESSARY* na interface *Asserted Conditions*, de forma a criar uma condição necessária.
3. Pressione o botão *Create restriction* (*Criar Restrição*), mostrado na FIG. 4.29 Surge a caixa de diálogo *Create Restriction* (vide FIG. 4.30), utilizada para criar uma restrição.



Figura 4.29: botão *Criar Restrição*



Figura 4.30: caixa de diálogo *Criar Restrição*

A caixa de diálogo *Create Restriction* (*Criar Restrição*) tem quatro partes principais:

1. a lista de propriedades;
2. a lista de tipos de restrição;
3. a caixa de edição do *filler*;
4. o painel para construção de expressões.

Para criar uma restrição são necessários três procedimentos:

- Selecione o tipo de restrição na lista *Restriction* (*Restrição*) (num. 2 na FIG. 4.30); o padrão é uma *restrição existencial* (E).
- Selecione a propriedade a qual deseja aplicar a restrição na lista *Restricted Properties* (*Propriedade a ser Restrita*) (num. 1 na FIG. 4.30);.
- Especifique um *filler* para a restrição na caixa de edição de *fillers* (num. 3 na FIG. 4.30); possivelmente, será preciso usar o painel para construção de expressões (num. 4 na FIG. 4.30).

## Exercício 17: Adicione restrição a *Pizza* especificando que *Pizza* deve ter uma *PizzaBase*

---

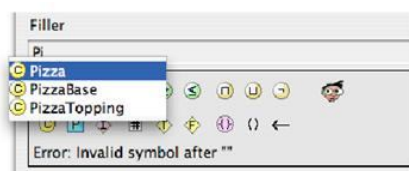
1. Selecione a declaração E *someValuesFrom* na lista de tipos de restrição (*someValuesFrom* é outro nome para a restrição existencial).
2. Selecione a propriedade *hasBase* da lista de propriedades.
3. Especifique que o *filler* é *PizzaBase*, o que pode ser feito de duas maneiras:  
 -digite *PizzaBase* na caixa de edição de *fillers*; ou...  
 -pressione o botão *Insert class* no painel de construção de expressões (FIG. 4.31) para exibir a hierarquia de classe, de onde *PizzaBase* pode ser selecionada.



**Figura 4.31: Painel *Expression Builder* (Construtor de Expressões) e botão *Insert Class***

4. Pressione o botão *Ok* para criar a restrição e feche a caixa diálogo. A restrição será exibida na interface *Asserted Conditions*. Caso algum erro seja detectado, a caixa de diálogo não é fechada e uma mensagem de erro é exibida no painel para construção de expressões; nesse caso, verifique se a restrição, a propriedade e o *filler* foram especificados corretamente.

**DICA** = um recurso útil para a construção de expressões é o auto-completar, utilizado para nomes de classes, nomes de propriedades e nomes de indivíduos. O recurso auto-completar é ativado pressionando-se *alt tab*. No exemplo acima, ao digitar "*Pi*", as opções iniciadas com "*Pi*" são apresentadas, conforme mostra a FIG. 4.32 abaixo. Com as teclas *up* e *down* seleciona-se *PizzaBase*.



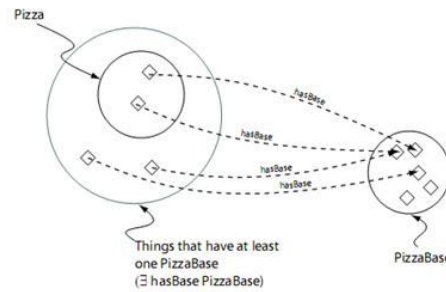
**Figura 4.32: o recurso auto-completar do construtor de expressões**

A interface *Asserted Conditions* aparece agora de forma similar a apresentada na FIG. 4.33:



**Figura 4.33: Interface *Asserted Conditions*: descrição de uma pizza**

**SIGNIFICADO** = até aqui descreveu-se a classe *Pizza* como uma subclasse de *owl:Thing* e como uma subclasse das "coisas" que possuem uma base, a qual, por sua vez, é algum tipo de *PizzaBase*. Note que tais condições são *necessárias*, ou seja, se alguma coisa é uma *Pizza* é necessário que tal coisa seja membro da classe *owl:Thing* (em OWL todas as coisas são membros de *owl:Thing*), e ainda é necessário que tal coisa tenha um tipo de *PizzaBase*. Em linguagem formal, diz-se que: alguma coisa é uma *Pizza* se é **necessário** existir um relacionamento entre tal coisa e um indivíduo membro da classe *PizzaBase*, através da propriedade *hasBase*. Essa situação é representada na FIG.4.34 a seguir.



**Figura 4.34:** para que algo seja uma *Pizza* é necessário que tenha pelo menos uma *PizzaBase*; uma *Pizza* é uma subclasse de coisas que tem pelo menos um *PizzaBase*

### Criação de tipos diferentes de Pizzas

Deseja-se adicionar diferentes *Pizzas* a ontologia. Adicione primeiro a *MargheritaPizza*, uma *Pizza* que tem recheio de mussarela e tomate. Para manter a ontologia organizada, agrupe os diferentes tipos de *Pizzas* na classe *NamedPizza* (*PizzasNomeadas*):

#### **Exercício 18: criar uma subclasse de *Pizza* chamada *NamedPizza*, e uma subclasse de *NamedPizza* chamada *MargheritaPizza***

1. Selecione a classe *Pizza* na hierarquia de classes da etiqueta *OWLClasses*.
2. Pressione *Create subclass* para criar uma nova subclasse de *Pizza*, e nomeie essa nova subclasse como *NamedPizza*.
3. Crie uma nova subclasse de *NamedPizza*, e nomeie essa subclasse como *MargheritaPizza*.
4. Adicione um comentário a classe *MargheritaPizza* usando a caixa de comentário localizada abaixo da campo *For class* no *Class Editor* (*Editor de Classes*): "uma pizza que tem apenas recheio de *Mozzarella* e *Tomato*". Adicionar um comentário é uma boa prática para documentar classes, propriedades, etc durante a edição da ontologia, comunicando o significado pretendido para outros desenvolvedores.

Tendo criado a classe *MargheritaPizza* é preciso especificar os recheios dessa *Pizza*. Adicionam-se então duas restrições para dizer que uma *MargheritaPizza* tem os recheios *MozzarellaTopping* e *TomatoTopping*.

#### **Exercício 19: criar uma restrição existencial (E) para a classe *MargheritaPizza*, a propriedade *hasTopping* e o filler *MozzarellaTopping*, para que uma *MargheritaPizza* tenha pelo menos um *MozzarellaTopping*.**

1. Certifique que *MargheritaPizza* está selecionada na hierarquia de classes.
2. Selecione *NECESSARY* no campo *Asserted Conditions* (*Condições Declaradas*), da interface *Class Editor* (*Editor de Classes*), de forma a criar uma condição necessária.
3. Use *Create restriction* (*Criar Restrições*) no campo *Asserted Conditions* (*Condições Declaradas*), da interface *Class Editor* (*Editor de Classes*), para exibir a caixa de diálogo.
4. Na caixa de diálogo *Create restrictions* faça com que a restrição criada seja um quantificador existencial selecionando o tipo de restrição tipo E *someValuesFrom*.



5. Selecione *hasTopping* como a propriedade a sofrer a restrição.
6. Entre com a classe *MozzarellaTopping* como o *filler* da restrição. Isto pode ser realizado digitando o nome da classe *MozzarellaTopping* dentro da caixa de edição *Filler*, ou usando o botão *Insert class* (*Inserir Classe*) para exibir a hierarquia de classes da ontologia.
7. Pressione o botão *Ok* na caixa de diálogo *Create Restriction* (*Criar Restrição*) para criar a restrição desejada. Caso exista algum erro, a restrição não será criada e uma mensagem de erro será exibida no painel de construção de expressões.

Agora especifique que *MargheritaPizza* também tem *TomatoTopping*.

## **Exercício 20: criar uma restrição existencial (E) para a classe *MargheritaPizza*, a propriedade *hasTopping* e o filler *TomatoTopping*, para que uma *MargheritaPizza* tenha pelo menos um *TomatoTopping***

1. Selecione *MargheritaPizza* na hierarquia de classes.
2. Selecione *NECESSARY* no campo *Asserted Conditions* (*Condições Declaradas*) do *Class Editor* (*Editor de Classes*) para adicionar uma condição necessária.
3. Use *Create Restriction* (*Criar Restrição*), no campo *Asserted Conditions* (*Condições Declaradas*) da interface *Class Editor* (*Editor de Classes*), para exibir o diálogo.
4. Na caixa de diálogo *Create Restriction* (*Criar Restrição*) faça com que a restrição criada seja um quantificador existencial selecionando o tipo de restrição tipo E *someValuesFrom*.
5. Selecione *hasTopping* como a propriedade a sofrer restrição.
6. Entre com a classe *TomatoTopping* como o *filler* para a restrição.
7. Clique no botão *Ok* para criar restrição.

O campo *Asserted Conditions* (*Condições Declaradas*) deve estar semelhante com a imagem da FIG. 4.35.



**Figura 4.35: o campo *Asserted Conditions* com a descrição de *MargheritaPizza***

**SIGNIFICADO** = adicionaram-se restrições a *MargheritaPizza* para informar que uma *MargheritaPizza* é uma *NamedPizza* que tem pelo menos um tipo de *MozzarellaTopping* e pelo menos tipo de *TomatoTopping*. Formalmente, lê-se: para que alguma coisa seja membro da classe *MargheritaPizza* é necessário que seja membro da classe *NamedPizza*; é necessário que seja também um membro da classe anônima de coisas que estão ligadas a pelo menos um membro da classe *MozzarellaTopping* via a propriedade *hasTopping*, e ainda é necessário que seja um membro da classe *TomatoTopping* via propriedade *hasTopping*.

Crie uma classe para representar a *AmericanaPizza*, que tem recheio de *Pepperoni*, *Mozzarella* e *Tomato*.

A classe *AmericanaPizza* é similar a classe *MargheritaPizza*: uma *AmericanaPizza* é quase uma *MargheritaPizza* com a diferença de que tem um recheio a mais (*Pepperoni*). Dessa forma, faça um clone da classe *MargheritaPizza* e então adicione uma restrição extra para informar que ela tem um recheio *Pepperoni*.

### **Exercício 21: clonar e modificar a descrição de *MargheritaPizza***

1. Selecione *MargheritaPizza* na hierarquia de classes, e clique com o botão direito do *mouse* dentro do campo *Asserted Hierarchy (Hierarquia Declarada)* para exibir um novo menu.
2. No menu exibido selecione *Create clone (Criar clone)*, o que cria uma cópia da classe *MargheritaPizza*, com o nome *MargheritaPizza2*, a qual tem as mesmas características (restrições, etc) de *MargheritaPizza*.
3. Renomeie *MargheritaPizza 2* para *AmericanaPizza* usando o campo *For class* do *Class Editor (Editor de Classes)*.
4. Selecione *AmericanaPizza*, escolha o cabeçalho *NECESSARY* no campo *Asserted Conditions (Condições Declaradas)* do *Class Editor (Editor de Classes)*, para adicionar a nova restrição as condições necessárias de *AmericanaPizza*.
5. Pressione *Create restrictions (Criar Restrições)* no campo *Asserted Conditions (Condições Declaradas)* do *Class Editor (Editor de Classes)*, para exibir a caixa de diálogo.
6. Selecione *E someValuesFrom* como tipo de restrição para criar a restrição de quantificador existencial.
7. Selecione a propriedade *hasTopping* como a propriedade a sofrer restrição.
8. Especifique no campo de restrições a classe *PepperoniTopping*, digitando o termo na caixa de edição do *filler*, ou usando o botão *Insert Class (Inserir Classe)* para apresentar o diálogo o qual permite a seleção de *PepperoniTopping*.
9. Pressione o botão *Ok* para criar a restrição.

A forma gráfica *Asserted Conditions (Condições Declaradas)* deve agora parecer como a imagem da FIG4.46.



**Figura 4.36: o campo *Asserted Conditions* mostrando a descrição para *AmericanaPizza***

### **Exercício 22: criar classes *AmericanHotPizza* e *SohoPizza***

1. Uma *AmericanHotPizza* é quase uma *AmericanaPizza*, mas possui *Jalapeno* (*PimentaDiferente*). Crie o novo tipo clonando a classe *AmericanaPizza*, e adicionando uma restrição existencial a propriedade *hasTopping* com um *filler JalapenoPepperTopping*.
2. Uma *SohoPizza* é quase uma *MargheritaPizza*, mas tem recheios adicionais de *Olive* (*Azeitona*) e *Parmezan* (*Parmesão*). Crie este novo tipo clonando *MargheritaPizza* e adicionando duas restrições existenciais junto a propriedade *hasTopping*, uma com o *filler OliveTopping*, e outra com o *ParmezanTopping*.

Para a *AmericanHotPizza* a forma gráfica *Asserted Conditions* (*Condições Declaradas*) deve agora parecer como a imagem da FIG4.37:



**Figura 4.37:** o campo *Asserted Conditions* com a descrição para *AmericanaHotPizza*

Para *SohoPizza*, a forma gráfica *Asserted Conditions* (*Condições Declaradas*) deve parecer como a imagem da FIG4.38:



**Figura 4.38:** o campo *Asserted Conditions* mostrando a descrição para *SohoPizza*

Após criar essas *Pizzas* é preciso torná-las disjuntas.

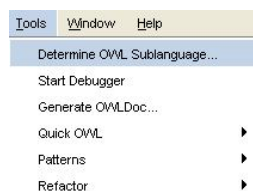
## **Exercício 23: tornar disjuntas as subclasses de *NamedPizza*.**

1. Selecione a classe *MargheritaPizza* na hierarquia da etiqueta *OWLClasses*.
2. Pressione o botão *Add all siblings* (*Adicionar todas as irmãs*) no campo *Disjoints* (*Disjunções*) para tornar as *Pizzas* disjuntas.

## **4.9) Uso de um MI-Mecanismo de Inferência (*reasoner*)**

### **4.9.1) Determinação a sub-linguagem OWL**

Conforme já mencionado, a OWL possui três sub-linguagens: *OWL-Lite*, *OWL-DL* e *OWL-Full* (a definição exata dessas sub-linguagens pode ser encontrada no site do W3C-World Wide Web Consortium). O *Protege-OWL* possui um mecanismo de validação que é capaz de determinar a sub-linguagem da ontologia em edição. Para usar o mecanismo de validação, na opção do menu *Tools* (*Ferramentas*), selecione o comando *Determine OWL Sublanguage...* (*Determinar a sub-linguagem OWL...*) conforme apresentado na FIG.4.39.



**Figura 4.39:** o menu *Tools* (*Ferramentas*)

Uma importante característica de ontologias descritas com o *OWL-DL* é a possibilidade processamento por um *reasoner* (*mecanismo de inferência*). Um relevante serviço oferecido por mecanismos de inferência é o teste para saber se uma classe é, ou não é, uma subclasse de outra classe (*subsumption test*), ou seja, as

descrições das classes (condições) são utilizadas para determinar se existe dentre elas uma relação superclasse/subclasse. Através de tais testes em todas as classes de uma ontologia é possível inferir automaticamente a hierarquia de classes da ontologia. Outro serviço padrão oferecido pelo mecanismo de inferência é o de *consistency checking* (*verificação da consistência*). Baseado na descrição (condições) de uma classe, o mecanismo de inferência pode verificar se é possível, ou não, que uma classe possua instâncias. Uma classe é considerada inconsistente se não é possível a ela ter instâncias.

**VOCABULÁRIO** = MI-Mecanismos de inferência (*reasoners*) são também chamados de *classificadores*.

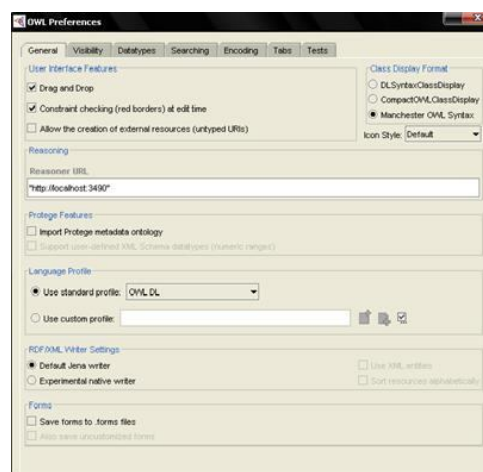
#### 4.9.2) Uso do RACER

Para executar inferências em ontologias no *Protege-OWL* é preciso instalar um MI compatível com o *DIG-Description Logic Implementers Group* (*Grupo de Desenvolvedores de Lógica Descritiva*). Um compatível DIG possibilita comunicação via um protocolo padrão para acesso ao MI. Antes disso, o MI deve ser instalado, configurado e iniciado. Neste tutorial utiliza-se o RACER, disponível na Internet em uma variedade de plataformas. Depois de instalado, o RACER deve ser iniciado com a configuração padrão (o padrão de comunicação é o HTTP ativado na porta 8080). A FIG. 4.40 apresentar uma visão dos dados exibidos na inicialização do RACER; a segunda linha de baixo para cima indica o estabelecimento da comunicação HTTP, especifica o endereço IP e o número da porta. Caso deseje usar uma porta diferente, é preciso configurar o *Protégé-OWL* através do comando *Preferences...* (*Preferências...*) do menu *OWL* (vide FIG.4.41)

```
/// RACER Version 1.7.12
/// RACER: Reasoner for ABoxes and Concept Expressions Renamed
/// Supported Description Logic: ALCQIIR+DI-
/// Copyright (C) 1998-2003, Volker Haarslev and Ralf Möller.
/// RACER comes with ABSOLUTELY NO WARRANTY; use at your own risk.
/// Commercial use is prohibited; contact the authors for licensing.
/// RACER is running on Mac OS Darwin computer as node Unknown

[2004-04-16 10:22:47] HTTP service enabled for: http://130.88.195.45:8080/
[2004-04-16 10:22:47] TCP service enabled on port 8088
```

**Figura 4.40:** tela inicial do RACER






**Figura 4.41:** menu OWL, comando *Preferences*

#### 4.9.3) Funcionamento do MI

Após iniciar o RACER (ou outro MI) a ontologia pode ser enviada ao mecanismo para execução dos

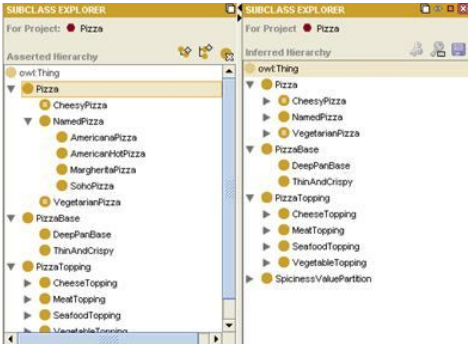
serviços. No *Protege-OWL*, a hierarquia de classe construída manualmente é chamada de *asserted hierarchy* (*hierarquia declarada*); a hierarquia de classes calculada automaticamente pelo MI é chamada *inferred hierarchy* (*hierarquia inferida*). Para classificar automaticamente a ontologia (e verificar sua consistência) utiliza-se o comando *Classify Taxonomy* (*Classificar Taxonomia*), disponível através de botão no menu superior do *Protege-OWL*, ou através de comando no menu OWL. Para verificar a consistência da ontologia, usa-se o comando *Check consistency...* (*Verificar Consistência*), disponível através de botão no menu superior do *Protege-OWL*, ou através de comando no menu OWL. Esses botões são apresentados na FIG. 4.42.

		
?	►	<i>Check consistency...</i> ( <i>Verificar consistência</i> )
C	►	<i>Classify taxonomy...</i> ( <i>Classificar taxonomia</i> )
I	►	<i>Compute inferred types...</i> ( <i>Computar tipos inferidos</i> )

**Figura 4.42: fragmento da barra de ferramentas do Protege-OWL**

Quando a *inferred hierarchy* (*hierarquia inferida*) for calculada, o resultado é apresentado ao lado da *asserted hierarchy* (*hierarquia declarada*) (vide FIG. 4.43). Se alguma classe foi reclassificada (ou seja, se a superclasse foi alterada) o seu nome aparece em azul na *hierarquia inferida*. Se uma classe inconsistente foi encontrada, o ícone correspondente aparece circulado em vermelho.

**VOCABULÁRIO** = a tarefa de calcular a hierarquia de classe inferida é também conhecida como *classificação da ontologia*.



**Figura 4.43: O painel *Inferred Hierarchy* (*Hierarquia Inferida*), aberto ao lado do painel *Asserted Hierarchy* (*Hierarquia Declarada*)**

#### 4.9.4)Classes inconsistentes

Para demonstrar o uso do MI na detecção de inconsistências foi criada uma classe, subclasse de *CheeseTopping* (*RecheioDeQueijo*) e de *MeatTopping* (*RecheioDeCarne*). Está estratégia é normalmente usada para verificar se a ontologia foi construída corretamente. As classes adicionadas para testar a integridade da ontologia são conhecidas como *ProbeClasses* (*Classes de Investigação*).

### Exercício 24: adicionar uma *Probe Class* denominada *ProbeInconsistentTopping* como subclasse de *CheeseTopping* e de *Vegetable*

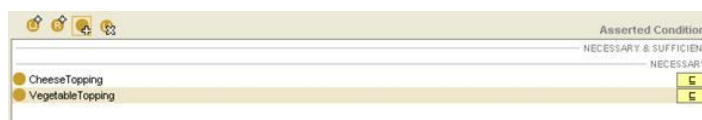
1. Selecione a classe *CheeseTopping* na hierarquia da etiqueta *OWLClasses*.
2. Crie uma subclasse de *CheeseTopping* com o nome *ProbeInconsistentTopping* (*Classe de investigação inconsistente*).
3. Adicione um comentário a classe *ProbeInconsistentTopping*, por exemplo: "essa classe será marcada

como inconsistente quando a ontologia for classificada". Isso possibilita que outra pessoa, ao analisar a ontologia, entenda que a classe foi deliberadamente criada como inconsistente.

4. Selecione a classe *ProbeInconsistentTopping* na hierarquia de classes, e escolha o *NECESSARY* no campo *Asserted Conditions (Condições Declaradas)*.

5. Clique no botão *Add Named Class (Adicionar Classes Nomeadas)* no campo *Asserted Conditions (Condições Declaradas)*. Será exibido um diálogo com a hierarquia de classes, permitido que uma delas seja selecionada. Selecione a classe *VegetableTopping* e pressione o botão Ok. A classe *VegetableTopping* será adicionada como uma condição necessária (como superclasse).

O campo *Asserted Conditions* deve estar parecido com FIG.4.44.



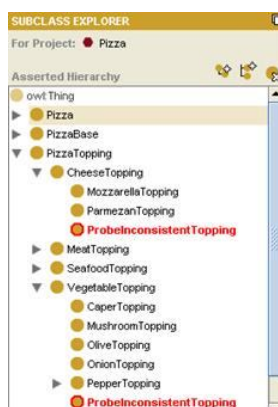
**Figura 4.44:** campo *Asserted Conditions (Condições Declaradas)* mostrando a classe *ProbeInconsistentTopping*

**SIGNIFICADO** = ao examinar a hierarquia, a classe *ProbeInconsistentTopping* aparece como subclasse de *CheeseTopping* e subclasse de *VegetableTopping*. Isso quer dizer que *ProbeInconsistentTopping* é um *CheeseTopping* e um *VegetableTopping*, ou seja, todos os indivíduos que são membros de *ProbeInconsistentTopping* são também (necessariamente) membros da classe *CheeseTopping* e (necessariamente) membros de *VegetableTopping*. Isto é incorreto visto que algo não pode ser queijo e vegetal ao mesmo tempo.

## **Exercício 25: verificação de inconsistência de *ProbeInconsistentTopping***

1. Pressione *Classify Taxonomy (Classificar Taxonomia)* na barra de ferramenta OWL.

Depois de alguns segundos, a *inferred hierarchy (hierarquia inferida)* é calculada e a janela correspondente é apresentada ao lado da janela *Asserted Hierarchy (Hierarquia Declarada)*, conforme apresentado na FIG. 4.45. Note-se que a classe *ProbeInconsistentTopping* está circulada em vermelho, indicando que o MI definiu a classe como inconsistente (ou seja, ela não pode ter indivíduos como membros).



**Figura 4.45:** a classe *ProbeInconsistentTopping* marcada como inconsistente pelo MI

**SIGNIFICADO** = analise a situação: sabe-se que uma coisa não pode ser queijo e vegetal ao mesmo tempo. Uma coisa não pode ser instância de *CheeseTopping (RecheioDeQueijo)* e instância de *VegetableTopping (RecheioDeVegetal)* simultaneamente. Entretanto, o nome para



as classes é escolhido pelo desenvolvedor. Para um MI, os nomes não tem significado, e ele não pode determinar se alguma coisa é inconsistente baseando-se em nomes. O motivo de a classe *ProbeInconsistentTopping* ser detectada como inconsistente é que suas superclasses *VegetableTopping* e *CheeseTopping* são disjuntas. Assim, indivíduos que são membros de *CheeseTopping* não podem ser membros de *VegetableTopping* e vice-versa.

**DICA** = Para fechar a *Inferred Hierarchy (Hierarquia Inferida)* use o sinal X na parte superior da janela da *hierarquia inferida*.

## Exercício 26: remoção de declaração disjunta de *CheeseTopping* e *VegetableTopping*

---

1. Selecione a classe *CheeseTopping* na hierarquia de classes.
2. No campo *Disjoints (Disjunções)* estão as classes irmãs a *CheeseTopping*, as quais são: *VegetableTopping (RecheioDeVegetais)*, *SeafoodTopping (RecheioDeFrutosDoMar)* e *MeatTopping (RecheioDeCarne)*. Selecione *VegetableTopping* no campo *Disjoints*.
3. Pressione o botão *Delete selected row (Apagar linha selecionada)* no campo *Disjoints* para remover o axioma de disjunção que mantém disjuntas *CheeseTopping* e *MeatTopping*.
4. Pressione *Classify Taxonomy (Classificar Taxonomia)* na barra de ferramentas OWL para enviar a ontologia ao MI. A ontologia é classificada e resultados são exibidos.

**SIGNIFICADO** = observe que *ProbeInconsistentTopping* não é mais inconsistente. Isto significa que indivíduos que são membros da classe *ProbeInconsistentTopping* são também membros da classe *CheeseTopping* e de *VegetableTopping*. Ou seja, estabeleceu-se (de forma errônea) que alguma coisa pode ser um queijo e um vegetal ao mesmo tempo. Isto ilustra a importância do cuidado no uso de axiomas de disjunção. As classes OWL se sobrepõem até que sejam declaradas disjuntas. Se algumas classes não são disjuntas, resultados inesperados podem surgir. Se certas classes são marcadas como disjuntas de forma incorreta, novamente resultados inesperados podem surgir.

## Exercício 27: correção da ontologia com a disjunção entre *CheeseTopping* e *Vegetable*

---

1. Selecione a classe *CheeseTopping* na hierarquia de classes.
2. O campo *Disjoints (Disjunções)* deve conter *MeatTopping* e *SeafoodTopping*.
3. Pressione *Add disjoint class (Adicionar classes disjuntas)* no campo *Disjoints* para exibir um diálogo onde se pode escolher classes. Selecione a classe *VegetableTopping* e pressione o botão Ok. A classe *CheeseTopping* deve ser disjunta de *VegetableTopping*.
4. Verifique se o axioma de disjunção foi adicionado corretamente: pressione *Classify Taxonomy*

(Classificar Taxonomia) na barra de ferramentas OWL. A ontologia é classificada, e a classe *ProbeInconsistentTopping* é marcada em vermelho, indicando inconsistência.

#### 4.10) Condições necessárias e suficientes (classes primitivas e definidas)

As classes criadas até agora foram descritas apenas com *condições necessárias*. A condição necessária pode ser lida: *se alguma coisa é um membro da classe então é necessário que preencha essas condições*. O uso de condições necessárias não permite dizer: *se alguma coisa preenche essas condições então deve ser um membro dessa classe*.

**VOCABULÁRIO** = se uma classe tem apenas condições necessárias ela é conhecida como uma *classe primitiva*.

Exemplo: uma subclasse de *Pizza* chamada *CheesyPizza* (*PizzaDeQueijo*) é uma *Pizza* com pelo menos um tipo de *CheeseTopping* (*RecheioDeQueijo*).

#### **Exercício 28: criação de subclasse de *Pizza* chamada *CheesyPizza*, com pelo menos um recheio que é um tipo de *CheeseTopping***

1. Selecione *Pizza* na hierarquia de classes da etiqueta *OWLClasses*.
2. Pressione *Create Subclass* (*Criar Subclasse*) para criar uma subclasse de *Pizza*; renomeie-a como *CheesyPizza*.
3. Selecione *CheesyPizza* na hierarquia de classes; selecione *NECESSARY* na interface *Asserted Conditions* (*Condições Declaradas*). Ao selecionar *Asserted* (*Declarada*), a opção *Inferred* (*Inferida*) aparece automaticamente após a execução da classificação.
4. Pressione *Create restriction* (*Criar Restrição*) na interface *Asserted Conditions* (*Condições Declaradas*) para exibir o diálogo correspondente.
5. Selecione *SomeValuesFrom* como o tipo de restrição a criar.
6. Selecione *hasTopping* como propriedade a sofrer restrição.
7. Na caixa de edição do *filler* digite *CheeseTopping*, ou use o botão *Insert class* (*Inserir classe*) para exibir um diálogo do qual *CheeseTopping* pode ser selecionado. Pressione *Ok* para fechar o diálogo e criar a restrição.

A forma gráfica *Asserted Conditions* (*Condições Declaradas*) deve agora estar parecida como a apresentada na FIG.4.46.



**Figura 4.46: a descrição de *CheesyPizza*, usando *Necessary Conditions* (Condições Necessárias)**

**SIGNIFICADO** = a descrição de *CheesyPizza* indica que se alguma coisa é membro da classe *CheesyPizza* é necessário que seja membro da classe *Pizza*, e é necessário que tenha pelo menos um recheio que seja membro da classe *CheeseTopping*.



A atual descrição de *CheesyPizza* indica que se alguma coisa é um *CheesyPizza* necessariamente é uma *Pizza*, e além disso, ainda precisa ter pelo menos um recheio que é um tipo de *CheeseTopping*. Para dizer isto usam-se condições necessárias. Considere agora um indivíduo qualquer. Sabe-se que esse indivíduo é membro da classe *Pizza*; sabe-se também que tem pelo menos um tipo de *CheeseTopping*.

Entretanto, pela atual descrição de *CheesyPizza* essas informações não são suficientes para determinar que o indivíduo é membro da classe *CheesyPizza*. Para que isto seja possível é preciso mudar as condições de *CheesyPizza*, de *condições necessárias* para *condições necessárias e suficientes*. As condições não vão ser apenas necessárias para associação a classe *CheesyPizza*, elas vão ser também suficientes para determinar que qualquer indivíduo que as satisfaça é um membro de *CheesyPizza*.

**VOCABULÁRIO** = uma classe que tem pelo menos um conjunto de *condições necessárias e suficientes* é conhecida como *Defined Class (Classe Definida)*.

**VOCABULÁRIO** = classes que tem apenas condições necessárias são também conhecidas como *Partial Classes (Classes Parciais)*. Classes que tem pelo menos um conjunto de condições necessárias e suficientes são também conhecidas como *Complete Classes (Classes Completas)*.

Para converter *condições necessárias* para *condições necessárias e suficientes*, movem-se as condições do cabeçalho *NECESSARY* no campo *Asserted Conditions (Condições Declaradas)* para o cabeçalho *NECESSARY & SUFFICIENT*. Isto pode ser feito arrastando e soltando as condições, uma a uma.

### **Exercício 29: conversão de condições necessárias de *CheesyPizza* em condições necessárias e suficientes**

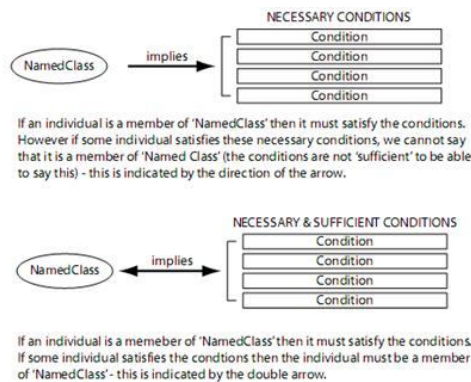
1. Selecione *CheesyPizza* na hierarquia de classes.
2. Em *Asserted Conditions (Condições Declaradas)* selecione E *hasTopping CheeseTopping*.
3. Arraste a restrição E *hasTopping CheeseTopping* de *NECESSARY* para *NECESSARY & SUFFICIENT*.
4. Selecione a classe *Pizza*.
5. Arraste a classe *Pizza* de *NECESSARY* para cima de E *hasTopping CheeseTopping*.

O campo *Asserted Conditions* deve parecer com a FIG.4.41:



**Figura 4.41: a descrição de *CheesyPizza*, usando condições *Necessary and Sufficient***

**SIGNIFICADO** = a descrição de *CheesyPizza* foi convertida em uma *definição*. Se alguma coisa é uma *CheesyPizza* é *necessário* que seja uma *Pizza*, e também é *necessário* que *pelo menos um* de seus recheios seja membro da classe *CheeseTopping*. Além disso, se um indivíduo é um membro da classe *Pizza* e ele tem pelo menos um recheio que é membro da classe *CheeseTopping* então essas condições são *suficientes* para determinar que o indivíduo deve ser um membro da classe *CheesyPizza*. A noção de condições necessárias e suficientes é ilustrada na FIG. 4.48 e 4.48a.



**Figura 4.48: Necessary and Sufficient conditions**

<b>Condições necessárias</b>	Se um indivíduo é membro de <i>NamedClass</i> ( <i>ClasseNomeada</i> ) então é obrigatório que satisfaça as condições. Entretanto, se algum indivíduo satisfaz as condições necessárias, não se pode dizer que seja membro de <i>NamedClass</i> : as condições não são suficientes para que se possa dizer isso. Tal fato é indicado pela direção da seta.
<b>Condições necessárias e suficientes</b>	Se um indivíduo é membro de <i>NamedClass</i> ( <i>ClasseNomeada</i> ) então é obrigatório que satisfaça as condições. Se algum indivíduo satisfaz as condições então é obrigatório que seja membro de <i>NamedClass</i> . Tal fato é indicado pela seta bidirecional.

**Fig. 4.48 a: condições necessárias e suficientes**

**ATENÇÃO** = Se por acidente *Pizza* foi descrita como *NECESSARY & SUFFICIENT* (ao invés de *E hasTopping CheeseTopping*) no exercício 29, o campo *Asserted Conditions* (*Condições Declaradas*) vai aparecer como a imagem mostrada na FIG. 4.49. Neste caso, uma nova condição necessária e suficiente foi criada, e não é a desejada. Para corrigir este erro, arraste *Pizza* sobre a restrição *E hasTopping CheeseTopping*.



**Figura 4.49: Uma descrição incorreta de CheesyPizza**

**DICA** = condições podem também ser transferidas de *NECESSARY* para *NECESSARY & SUFFICIENT* (e vice versa) usando *Cut & Paste* (*Cortar e Colar*). Clique com o botão direito em uma condição e selecione *Cut & Paste* no menu.

Resumindo... se a classe **A** é descrita por condições *necessárias*, então pode-se dizer que se um indivíduo é membro de **A**, ele deve satisfazer as condições. Não se pode dizer que qualquer indivíduo que satisfaça tais condições é um membro da classe **A**. Entretanto, se a classe **A** é *definida* usando *condições necessárias e suficientes* pode-se dizer que se um indivíduo é membro da classe **A** ele deve satisfazer as condições, e pode-se dizer que qualquer indivíduo satisfaz essas condições então ele deve ser membro de **A**. As condições não são apenas necessárias para a associação com **A**, mas também suficientes de forma a determinar que, se alguma coisa satisfaz essas condições, é um membro de **A**.

Como isso é útil na prática? Suponha outra classe **B**, e que quaisquer indivíduos que são membros da classe **B** também satisfazem as condições que definem a classe **A**. Pode-se determinar que a classe **B** é subclasse de **A**. Verificar a relação classe/suprclasse (*subsumption relationship*) de classes é uma tarefa básica de um

MI de lógica descritiva, e é possível usá-lo para computar automaticamente a hierarquia.

**OBSERVAÇÃO** = Em *OWL* é possível ter conjuntos múltiplos de condições necessárias e suficientes.

#### 4.10.1) Classes Primitivas e definidas

As classes que tem pelo menos um conjunto de *condições necessárias e suficientes* são conhecidas como *Defined Classes (Classes Definidas)*: tais classes tem uma definição, e qualquer indivíduo que satisfaça tal definição pertence a classe. Classes que não tem nenhum conjunto de *condições necessárias e suficientes* (apenas *condições necessárias*) são conhecidas como *Primitive Classes (Classes Primitivas)*. No *Protege-OWL*, as *Defined Classes (Classes Definidas)* possuem um ícone com fundo alaranjado. As *Primitive Classes (Classes Primitivas)* possuem um ícone com fundo amarelo. É importante entender que o MI pode classificar automaticamente apenas as classes definidas, ou seja, classes com pelo menos um conjunto de *condições necessária e suficientes*.

#### 4.11) Classificação automática

Um dos benefícios em construir uma ontologia com a sublinguagem *OWL-DL* é a possibilidade computar automaticamente a hierarquia de classes através de um MI. No caso de ontologias muito grandes (milhares de classes), esse auxílio automático para computar relacionamentos subclasse-superclasse é vital. Sem um MI é muito difícil manter grandes ontologias em um estado logicamente correto. Em casos em que as ontologias tem classes com muitas superclasses (herança múltipla) é uma boa prática construir uma hierarquia de classes como uma árvore simples. Dessa forma, as classes na *Asserted Hierarchy (Hierarquia Declarada, aquela construída manualmente)* não têm mais do que uma superclasse. Computar e manter herança múltipla também é trabalho do MI. Esta técnica (as vezes chamada *normalização de ontologia*) ajuda a manter a ontologia em um estado sustentável e modular. Isso promove a reutilização da ontologia e minimiza erros humanos inerentes na manutenção de hierarquias com herança múltipla.

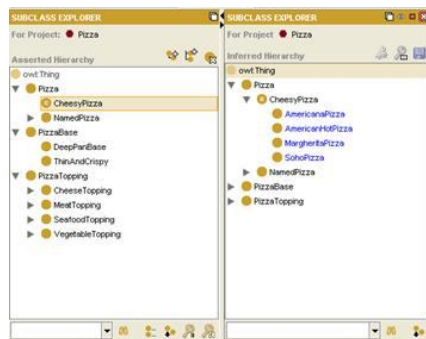
Uma vez criada uma definição para *CheesyPizza*, pode-se usar o MI para computar automaticamente subclasses de *CheesyPizza*.

### **Exercício 30: uso do MI para computar automaticamente a subclasse de *CheesyPizza***

---

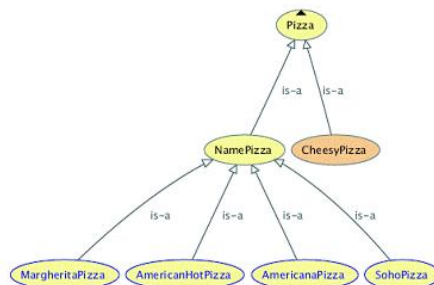
1. Certifique-se de que um MI (por exemplo, o RACER) está funcionando.
2. Pressione o botão *Classify Taxonomy (Classificar Taxonomia)* na barra de ferramentas.

A *Inferred Hierarchy (Hierarquia Inferida)* é calculada e abre-se a nova janela com os resultados, similar aos apresentados FIG.4.50. As FIG. 5.51 e FIG. 4.52 mostram uma visão *OWLViz (plug-in)* das hierarquias declarada e inferida, respectivamente. Observe que as classes cujas superclasses foram modificadas pelo MI são destacadas em azul.

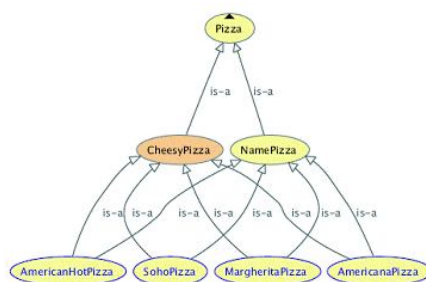


**Figura 4.50:** A *Asserted Hierarchy* (Hierarquia Declarada) e a *Inferred Hierarchy* (Hierarquia Inferida) apresentando os resultados da classificação para *CheesyPizza*

**SIGNIFICADO** = o MI determinou que *MargheritaPizza*, *AmericanaPizza*, *SohoPizza* e *AmericanHotPizza* são subclasses da *CheesyPizza*. Isto ocorreu porque definiu-se *CheesyPizza* usando condições necessárias e suficientes. Um indivíduo que é uma *Pizza*, e que tem pelo menos um recheio que é um *CheeseTopping* (*RecheioDeQueijo*), é membro da classe *CheesyPizza*. Como todos os indivíduos descritos pelas classes *MargheritaPizza*, *AmericanaPizza*, *AmericanHotPizza* e *SohoPizza* são *Pizzas* e tem pelo menos um recheio que é um *CheeseTopping* (ou recheios que pertencem a subclasses de *CheeseTopping*), o MI inferiu que essas classes devem ser subclasses de *CheeseTopping*.



**Figura 4.51:** vista do *plug-in OWLViz* apresentando a *Asserted Hierarchy* (Hierarquia Declarada) para *CheesyPizza*



**Fig.4.52:** vista do *plug-in OWLViz* apresentando a *Inferred Hierarchy* (Hierarquia Inferida) para *CheesyPizza*

**ATENÇÃO** = em geral, as classes não são classificadas como subclasses de *Primitive Classes* (*Classes Primitivas*) pelo MI. As *classes primitivas* são aquelas que possuem apenas condições necessárias. A exceção a esse fato ocorre quando uma propriedade tem um domínio que é uma *classe primitiva*. Isto pode fazer com que as classes sejam reclassificadas sob a *classe primitiva* que é o domínio da propriedade. Não se recomenda o uso do domínio de propriedade para causar tais efeitos.

#### 4.11.1) Resultados da classificação

Depois de finalizada a classificação, as classes inconsistentes são exibidas em um painel conforme

apresentado na FIG. 4.53: o painel *Classification Results (Resultados de Classificação)* é aberto na parte de baixo da tela do *Protege-OWL*. O ícone *Spanner* (uma pequena chave-inglesa) no lado esquerdo do painel corresponde ao botão *Assert Selected Changes (Aceitar Alterações Seleccionadas)*. Ao pressionar este botão, os relacionamentos superclasse-subclasse inferidos pelo MI são transportados para a *Asserted Hierarchy (Hierarquia Declarada)*, a qual foi manualmente construída. Por exemplo, se o botão *Assert Selected Changes (Aceitar Alterações Seleccionadas)* foi acionado com a seleção mostrada na FIG. 4.53, *CheesyPizza* é adicionado como superclasse de *AmericanaPizza*.



**Figura 4.53: o painel *Classification Results (Resultados de Classificação)***

**ATENÇÃO** = Apesar da existência desta facilidade, ela não é considerada uma boa prática para inserir relacionamentos inferidos em hierarquias construídas manualmente ou na hierarquia declarada enquanto a ontologia está sendo desenvolvida. Por isso aconselha-se evitar o uso desse botão durante o desenvolvimento de uma ontologia.

#### 4.12) Restrições Universais

Todas as restrições criadas até agora são *restrições existenciais (E)*. Esse tipo de restrição especifica a existência de *pelo menos um* relacionamento através de determinada propriedade com um indivíduo membro de uma classe, identificada pelo *filler*. Entretanto, a restrição existencial não obriga que *os únicos* relacionamentos através da propriedade que possa existir sejam obrigatoriamente com indivíduos membros de uma classe específica (*filler*).

Por exemplo, pode-se usar uma restrição existencial *E hasTopping MozzarellaTopping* para descrever os indivíduos que tem *pelo menos um* relacionamento através da propriedade *hasTopping*, com um indivíduo membro da classe *MozzarellaTopping*. Esta restrição não implica em que todos os relacionamentos *hasTopping* devam ser, obrigatoriamente, membros da classe *MozzarellaTopping*. Para restringir um relacionamento através de uma propriedade com indivíduos membros de uma classe específica, deve-se usar a *Universal Restriction (Restrição Universal)*.

As *Universals Restrictions (Restrições Universais)* são identificadas pelo símbolo *A*. Tais restrições condicionam o relacionamento através da propriedade com indivíduos que são membros de uma classe específica. Por exemplo, a restrição universal expressa pela declaração *A hasTopping MozzarellaTopping* descreve os indivíduos cuja totalidade dos relacionamentos *hasTopping* ocorre com membros da classe *MozzarellaTopping*. Os indivíduos não tem relacionamentos *hasTopping* com indivíduos que não são membros de *MozzarellaTopping*.

**VOCABULÁRIO** = *Universals Restrictions (Restrições Universais)* são também conhecidas como *All Restrictions (Todas Restrições)*.

**ATENÇÃO** = a *Restrição Universal* citada acima *A hasTopping MozzarellaTopping*, também descreve os indivíduos que não participam de nenhum relacionamento *hasTopping*. Um indivíduo que não participa de nenhum relacionamento *hasTopping*, por definição, nunca terá um relacionamento *hasTopping* com indivíduos que não são membros da classe *MozzarellaTopping*, e a restrição é assim satisfeita.

**ATENÇÃO** = para uma determinada propriedade, *Universals Restrictions (Restrições Universais)* não especificam a existência de relacionamento. Apenas indicam que, se existe um relacionamento para a propriedade, ele ocorre para indivíduos membros de uma classe.

Por exemplo, deseja-se criar a classe *VegetarianPizza (Pizza Vegetariana)*, de forma que os indivíduos membros dessa classe possam apenas ter recheios que são ou *CheeseTopping (RecheioDeQueijo)* ou *VegetableTopping (RecheioDeVegetais)*. Nesse caso, pode-se usar a *restrição universal*.

### Exercício 31: criação de classe para descrever *VegetarianPizza (Pizza Vegetariana)*

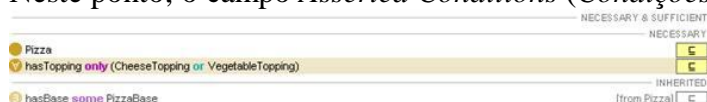
1. Crie uma subclasse de *Pizza*, e nomeá-la *VegetarianPizza*.
2. Selecione *VegetarianPizza*, clique em *NECESSARY* no campo *Asserted Conditions (Condições Declaradas)*.
3. Pressione *Create restriction (Criar Restrição)* no campo *Asserted Conditions (Condições Declaradas)* para exibir o diálogo *Create restriction (Criar Restrição)*.
4. Selecione o tipo de restrição *AllValuesFrom* para criar uma restrição de quantificador universal.
5. Selecione *hasTopping* como a propriedade a sofrer restrição.
6. O *filler* é *CheeseTopping (RecheioDeQueijo)* ou *VegetableTopping (RecheioDeVegetais)*. Digite *CheeseTopping* na caixa de edição do *filler* (ou use o botão *Insert class*). Agora é preciso usar o operador *unionOf (uniãoDe)* entre os nomes das classes: insira operador *unionOf* usando o botão correspondente no painel de construção de expressões, conforme apresentado na FIG. 4.54. Em seguida, digite a classe *VegetableTopping* (ou use o botão *Insert class*). A expressão *CheeseTopping* e *VegetableTopping* é então apresentada na caixa de edição de *filler*.



**Figura 4.54: Expression Builder Panel (Painel Construtor de Expressões) para inserir UnionOf (UniãoDe)**

7. Pressione o botão *Ok* para fechar o diálogo e criar a restrição. Caso exista algum erro o diálogo não é fechado e uma mensagem de erro aparece na parte de baixo do painel.

Neste ponto, o campo *Asserted Conditions (Condições Declaradas)* deve parecer como a FIG. 4.55.



**Figura 4.55: descrição de *VegetarianPizza*, usando *Necessary Conditions (Condições Necessárias)***



**SIGNIFICADO** = se alguma coisa é membro da classe *VegetarianPizza* é necessário que ela seja uma *Pizza*, e é *necessário* que tenha *apenas* ( *A* = *quantificador universal*) recheios que são tipos de *CheeseTopping* (*RecheioDeQueijo*) ou tipos de *VegetableTopping* (*RecheiosDeVegetais*). Em outras palavras, todos os relacionamentos *hasTopping* (*temRecheio*) dos quais os indivíduos membros de *VegetarianPizza* participam, devem ser com indivíduos que são membros das classes *CheeseTopping* ou *VegetableTopping*. A classe *VegetarianPizza* também contém indivíduos que são *Pizzas* e não participam em nenhum relacionamento *hasTopping*.

**DICA** = Em vez de usar o botão *Insert union* (*Inserir união*) no exercício anterior, pode-se simplesmente digitar *OR* (*OU*) na caixa de edição do *filler*, e a palavra é automaticamente convertida para o símbolo de união ( $\cup$ ).

**ATENÇÃO** = em situações como a do exemplo acima, um erro comum é usar a interseção ao invés da união. Por exemplo, *CheeseTopping* é *VegetableTopping*, é lido como *CheeseTopping AND VegetableTopping*. Embora *CheeseTopping AND VegetableTopping* seja uma frase comum em linguagem natural, em termos lógicos significa que algo é *simultaneamente* um tipo de *CheeseTopping* e de *VegetableTopping*. Isto é incorreto conforme demonstrado na seção 4.9.4. Se a classe *CheeseTopping* e *VegetableTopping* não são disjuntas, é possível dizer isso com legitimidade lógica. Dessa forma, a classe não é inconsistente e portanto não será destacada por um MI.

**ATENÇÃO** = no exemplo acima, uma opção é criar duas restrições universais, uma para *CheeseTopping* (*A hasTopping CheeseTopping*) e outra para *VegetableTopping* (*A hasTopping VegetableTopping*). Entretanto, quando restrições múltiplas são utilizadas (para qualquer tipo de restrição) a descrição total é considerada como a interseção das restrições individuais. Isso é equivalente a uma restrição com um *filler* que é a interseção de *MozzarellaTopping* e *TomatoTopping*. Conforme explicado, isto é incorreto do ponto de vista lógico.

No momento, *VegetarianPizza* é descrita usando *condições necessárias*. Entretanto, a descrição de *VegetarianPizza* poderia ser considerada como completa. Sabe-se que qualquer indivíduo que satisfaça a essas condições deve ser um *VegetarianPizza*. Pode-se portanto, converter as *condições necessárias* de *VegetarianPizza* em *condições necessárias e suficientes*. Isto vai permitir usar o MI para determinar a subclasse de *VegetarianPizza*.

### **Exercício 32: conversão das condições necessárias de *VegetarianPizza* em condições necessárias e suficientes**

---

1. Selecione *VegetarianPizza* na hierarquia de classes.
2. No campo *Asserted Conditions* (*Condições Declaradas*) selecione a restrição universal (*A*) na propriedade *hasTopping*.
3. Arraste *hasTopping* de *NECESSARY* para *NECESSARY & SUFFICIENT*.
4. Selecione a classe *Pizza*.
5. Arraste a classe *Pizza* que está em *NECESSARY* para cima de *hasTopping*.

O campo *Asserted Conditions* (*Condições Declaradas*) deve estar semelhante a FIG.4.56.



**Figura 4.56: o campo *Asserted Conditions* e a definição de *VegetarianPizza* com *Necessary and Sufficient Conditions* (*Condições Necessárias e Suficientes*)**

**SIGNIFICADO** = converteu-se a descrição de *VegetarianPizza* em uma *definição*. Se alguma coisa é uma *VegetarianPizza*, então é necessário que seja uma *Pizza* e também é necessário que todos os recheios pertençam a classe *CheeseTopping* ou *VegetableTopping*. Além disso, se alguma coisa é membro da classe *Pizza* e todos os recheios são membros da classe *CheeseTopping* ou *VegetableTopping*, essas condições são suficientes para reconhecer que tal coisa é ser um membro da classe *VegetarianPizza*.

#### 4.13) Classificação automática e OWR-Open World Reasoning (Raciocínio de Mundo Aberto)

Deseja-se usar o MI para computar automaticamente o relacionamento superclasse-subclasse (*subsumption relationship*) entre *MargheritaPizza* e *VegetarianPizza*, e entre *SohoPizza* e *VegetarianPizza*. Acredita-se que *MargheritaPizza* e *SohoPizza* devem ser pizzas vegetarianas (subclasses de *VegetarianPizza*) porque tem recheios vegetarianos: pela definição, recheios vegetarianos são membros das classes *CheeseTopping* ou *VegetableTopping* e de suas subclasses. Tendo anteriormente criado uma definição para *VegetarianPizza* usando um conjunto de *condições necessárias e suficientes*, pode-se usar o MI para classificação automática e para determinar as pizzas que são *VegetarianPizza* na ontologia.

### Exercício 33: uso do MI para classificar a ontologia

1. Certifique-se de que um MI (por exemplo, o RACER) está em execução. Pressione o botão *Classify taxonomy* (*Classificar Taxonomia*).

Observe que *MargheritaPizza* e *SohoPizza* não foram classificadas como subclasses de *VegetarianPizza*. Pode parecer estranho, pois *MargheritaPizza* e *SohoPizza* tem ingredientes vegetarianos, ou seja, ingredientes que são tipos de *CheeseTopping* ou de *VegetableTopping*. No entanto, pode-se observar que *MargheritaPizza* e *SohoPizza* perderam algo da definição, e assim não podem ser classificadas como subclasses de *VegetarianPizza*.

As inferências em OWL (Lógica Descritiva) se baseiam na OWA-Open World Assumption (*Suposição de Mundo Aberto*), também conhecida como OWR-Open World Reasoning (*Raciocínio de Mundo Aberto*). A *Suposição de Mundo Aberto* significa que não se pode assumir que alguma coisa não existe, até que seja estabelecido explicitamente que ela não existe. Em outras palavras, porque alguma coisa não foi definida como verdade, não significa que ela é falsa: presume-se que tal conhecimento apenas não foi adicionado a base de conhecimento.

No caso da ontologia de *Pizza*, indica-se que *MargheritaPizza* tem recheios que são tipos de *MozzarellaTopping* e também tipos de *TomatoTopping*. Por causa da *Suposição de Mundo Aberto*, até que se diga explicitamente que uma *MargheritaPizza* tem apenas esses dois tipos de recheios, presume-se (pelo MI) que uma *MargheritaPizza* pode ter outros recheios. Para especificar explicitamente que uma *MargheritaPizza* tem recheios que são tipos de *MozzarellaTopping* ou tipos de *MargheritaTopping*, e apenas estes, deve-se adicionar um *Closure Axiom* (*Axioma de Fechamento*) à propriedade *hasTopping*. Um *closure axiom* também é chamado de *Closure Restriction* (*Restrição de Fechamento*).



#### 4.13.1) Axiomas de fechamento

Um *Closure Axiom* (Axioma de Fechamento) consiste em uma restrição universal que atua na propriedade informando que ela pode apenas ser preenchida por *fillers* específicos. A restrição tem um *filler* que é a união dos *fillers* que ocorrem nas restrições existenciais da propriedade. Por exemplo, o *Closure Axiom* (Axioma de Fechamento) da propriedade *hasTopping* para *MargheritaPizza* é uma restrição universal que atua na propriedade *hasTopping*, com um *filler* que é a união de *MozzarellaTopping* e de *TomatoTopping*, ou seja, a declaração:

A *hasTopping* (*MozzarellaTopping*  $\dot{\cup}$  *TomatoTopping*).

### Exercício 34: adição de axioma de fechamento à propriedade *hasTopping* para *MargheritaPizza*

1. Selecione *MargheritaPizza* na hierarquia de classe da etiqueta *OWLClasses*.
2. Selecione *NECESSARY* no campo *Asserted Conditions* (Condições Declaradas).
3. Pressione o botão *Create Restriction* (Criar Restrição) no campo *Asserted Conditions* (Condições Declaradas) para exibir o diálogo.
4. Selecione o tipo de restrição A *allValuesFrom* (restrição universal).
5. Selecione *hasTopping* como a propriedade a sofrer restrição.
6. Na caixa de edição de *fillers* digite *MozzarellaTopping*  $\dot{\cup}$  *TomatoTopping*. Isto também pode ser feito digitando *MozzarellaTopping* OR *TomatoTopping* e nesse caso, o OR (OU) é convertido automaticamente para o símbolo  $\dot{\cup}$ . Pode-se também usar o botão *Insert class* (Inserir Classe) e o botão *Insert unionOf* (Inserir UniãoDe): insere-se a classe *MozzarellaTopping*, depois insere-se o símbolo *unionOf* e por fim a classe *TomatoTopping*.
7. Pressione o botão *Ok* para criar a restrição e adicioná-la a classe *MargheritaPizza*.

A forma gráfica *Asserted Conditions* (Condições Declaradas) deve estar similar a mostrada na FIG.4.57.



**Figura 4.57: campo *Asserted Conditions* (Condições Declaradas); *MargheritaPizza* com o axioma de fechamento para a propriedade *hasTopping***

**SIGNIFICADO** = isto significa que um indivíduo membro da classe *MargheritaPizza* deve ser membro da classe *Pizza*, deve ter pelo menos um recheio que é um tipo de *MozzarellaTopping*, deve ter pelo menos um recheio que é membro de *TomatoTopping*, e os recheios devem ser apenas tipos de *MozzarellaTopping* ou *TomatoTopping*.

**ATENÇÃO** = um erro comum é usar apenas restrições universais nas descrições. Por exemplo, descreve-se *MargheritaPizza* como subclasse de *Pizza*, usando apenas a declaração *A hasTopping* (*MozzarellaTopping* è *TomatoTopping*) sem nenhuma restrição existencial. Entretanto, pela semântica da restrição universal, a declaração significa realmente: coisas que são *Pizzas* e somente tem recheios que são *MozzarellaTopping* ou *TomatoTopping*, OU, coisas que são *Pizzas* e não tem qualquer recheio.

### **Exercício 35: adição de axioma de fechamento a propriedade *hasTopping* para *SohoPizza***

---

1. Selecione *SohoPizza* na hierarquia de classes da etiqueta *OWLClasses*.
2. Selecione *NECESSARY* no campo *Asserted Conditions* (*Condições Declaradas*).
3. Pressione o botão *Create restriction* (*Criar Restrição*) para exibir o diálogo.
4. Selecione o tipo de restrição *A allValuesFrom*, para criar uma restrição de quantificador universal.
5. Selecione *hasTopping* como a propriedade a sofrer restrição.
6. Na caixa de edição dos *fillers* insira a união dos recheios para *SohoPizza*, digitando *ParmesanTopping* OR *MozzarellaTopping* OR *TomatoTopping* OR *OliveTopping*. A palavra-chave OR (OU) é automaticamente convertida para o símbolo *unionOf* (è).
7. Pressione Ok para criar a restrição e fechar o diálogo. Se o diálogo não fechar devido a erros, verifique se os nomes das classes foram escritos corretamente.

Para complementar, adicione um *Closure Axiom* (*Axioma de Fechamento*) a propriedade *hasTopping* em *AmericanaPizza* e em *AmericanHotPizza*. A atividade de inserir manualmente axiomas de fechamento parece trabalhosa, mas o Protege-OWL possui recursos para a criação facilitá-la.

### **Exercício 36: criação automatica de axioma de fechamento na propriedade *hasTopping* para *AmericanaPizza***

---

1. Selecione *AmericanaPizza* na hierarquia de classes da etiqueta *OWLClasses*.
2. No campo *Asserted Conditions* (*Condições Declaradas*) clique com o botão direito sobre uma das restrições existenciais de *hasTopping*. Selecione *Add closure axiom* (*Adicionar um axioma de fechamento*). Uma restrição de fechamento (restrição universal) é criada na propriedade *hasTopping*, a qual contém a união dos *fillers* existenciais de *hasTopping*.

### **Exercício 37: criação automatica de axioma de fechamento para a propriedade *hasTopping* de *AmericanHotPizza***

---

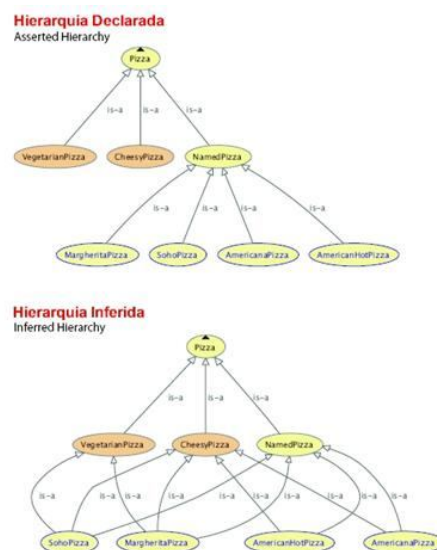
1. Selecione *AmericanHotPizza* na hierarquia de classes da etiqueta *OWLClasses*.
2. No campo *Asserted Conditions* (*Condições Declaradas*) clique com o botão direito sobre uma das restrições existenciais de *hasTopping*. Selecione *Add closure axiom* (*Adicionar um axioma de fechamento*).

Tendo adicionado axiomas de fechamento na propriedade *hasTopping* para as *Pizzas*, usa-se agora o MI para computar automaticamente a classificação.

### Exercício 38: uso do MI para classificar a ontologia

1. Pressione o botão *Classify taxonomy (Classificar Taxonomia)* na barra de ferramentas *OWL* para chamar o MI.

A ontologia é classificada e os resultados são apresentados no painel *Inferred Hierarchy (Hierarquia Inferida)*. Desta vez, *MargheritaPizza* e *SohoPizza* estão classificadas como subclasses de *VegetarianPizza*. Isto ocorreu porque especificou-se a propriedade *hasTopping* como *Closed (Fechada)* nas *Pizzas*, para dizer exatamente quais recheios elas tem. Além disso, *VegetarianPizza* foi definida para ser uma *Pizza* apenas com tipos de *CheeseTopping* e de *VegetableTopping*. A FIG.4.58 apresenta as hierarquias atuais, a declarada e a inferida. Observe que a *Asserted Hierarchy (Hierarquia Declarada)* é mais simples e mais limpa do que a *Inferred Hierarchy (Hierarquia Inferida)*. Embora a ontologia em questão é uma estrutura simples neste estágio, o uso do MI pode auxiliar, especialmente no caso de grandes ontologias, a manter a hierarquia de herança múltipla.



**Figura 4.58:** As hierarquias inferida e declarada, mostrando o antes e o depois da classificação de *Pizzas* em *CheesyPizzas* e *VegetarianPizzas*.

#### 4.14) Partições de Valor

Nessa seção criam-se *Value Partitions (Partições de Valor)*, as quais são usadas para refinar as descrições de classes. *Value Partitions (Partições de Valor)* não são parte da linguagem OWL, ou de outra linguagem de ontologia, são um *padrão de projeto*.

*Padrões de projetos* em ontologias são similares a *padrões de projeto* em programação orientada a objetos: são soluções desenvolvidas por especialistas e reconhecidos como soluções para problemas comuns de modelagem. Conforme mencionado, as *Value Partitions (Partições de Valor)* podem ser criadas para refinar descrições de classe. Por exemplo, cria-se uma *Value Partition* chamada *SpicinessValuePartition (PartiçãoDeValorApimentada)* para descrever o *spiciness* ("o tanto de pimenta") de *PizzaToppings (RecheiosDePizza)*. As *Value Partitions* restringem a faixa de valores possíveis para uma lista exhaustiva, por exemplo, a *SpicinessValuePartition* restringe a faixa para *Mild (Levemente apimentado)*, *Medium (Médio)*, e *Hot (Muito Apimentado)*.

Para criar uma *Value Partition* em OWL são necessários alguns passos:

1. Criar uma classe para representar a *Value Partition*. Por exemplo, para representar a partição de valor *spiciness* ("o tanto de pimenta"), cria-se a classe *SpicinessValuePartition* (*PartiçãoDeValorApimentada*).
2. Criar subclasses da partição para representar as opções. Por exemplo, pode-se criar as classes *Mild* (*Levemente apimentado*), *Medium* (*Médio*), e *Hot* (*Muito Apimentado*) como subclasses da classe *SpicinessValuePartition*.
3. Tornar essas subclasses disjuntas.
4. Fornecer um *Covering Axiom* (*Axioma de Cobertura*), de forma a que a lista de valores seja exaustiva.
5. Criar uma *Object Property* (*Propriedade Objeto*) para a *Value Partition*. Por exemplo, para a *SpicinessValuePartition* pode-se criar a propriedade *hasSpiciness* (*TemPimenta*).
6. Torne essa propriedade *funcional*.
7. Defina o *range* (*escopo*) da propriedade como a classe *ValuePartition*. Por exemplo, para a propriedade *hasSpiciness* o escopo é definido como *SpicinessValuePartition*.

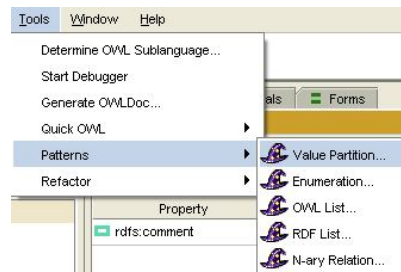
Devido ao maior número e a complexidade de passos que é mais fácil cometer erros. Despende-se mais tempo para criar algumas poucas *ValuePartitions*. Felizmente, o *Protege OWL* possui um assistente para criar *ValuePartitions*, o qual é denominado *Create ValuePartition* (*Criar Partição de Valor*).

Deseja-se criar um *ValuePartition* para descrever o tempero dos recheios de *Pizza*. Então será possível classificar as *Pizzas* em pizzas apimentadas e em pizzas não-apimentadas. Será possível também dizer que os recheios das pizzas têm um tanto de pimenta que as classifique como: *Mild* (*Levemente apimentado*), *Medium* (*Médio*), e *Hot* (*Muito Apimentado*). Note-se que tais opções são mutuamente exclusivas: alguma coisa não pode ser ao mesmo tempo *mild* (*Levemente apimentado*), e *hot* (*Muito Apimentado*), ou uma combinação de opções.

### **Exercício 39: criação de um *ValuePartition* para representar o *spiciness* ("tanto"de pimenta) em recheios de *Pizza***

---

1. Selecione *Create Value Partition* (*Criar Partição de Valor*) no menu do Protege (*Tools à Patterns à Value Partition...*) para chamar o *ValuePartition Wizard* (*Assistente de Partição de Valor*). Vide FIG. Extra 8.
2. Na primeira página do assistente digite *SpicinessValuePartition* como nome da classe *ValuePartition* e pressione o *Next*.
3. Em seguida, digite *hasSpiciness* como o nome da propriedade *ValuePartition*, e pressione o *Next*.



**FIG. Extra 8: acesso ao menu ValuePartition no Protege 3.4**

4. Agora é preciso especificar os valores para o *valueType* (*tipo de valor*). Na área de texto digite *Mild* e pressione *enter*, digite *Medium* e pressione *enter*, e digite *Hot* e pressione *enter*. Isto irá criar as subclasses *Mild*, *Medium* e *Hot* da classe *SpicinessValuePartition*. Pressione *Next* para continuar.

5. O nome do *ValuePartition* é verificado. Pressione *Next*.

6. A página de anotações está disponível: pode-se adicionar anotações para o *ValuePartition* caso desejado. Para continuar, pressione *Next*.

7. A última página do assistente solicita que se especifique uma classe para funcionar como um *root* (*raiz*), abaixo da qual todas as *ValuePartitions* serão criadas. Recomenda-se que as *ValuePartitions* sejam criadas sob uma classe nomeada, a qual é a opção padrão. Pressione *Finish* para criar a *ValuePartition*.

Analise as tarefas do assistente que reduzem o trabalho manual do usuário (FIG. 4.59 e 4.60):

1. Cria-se uma classe *ValuePartition* como subclasse de *owl:Thing*.
2. Cria-se uma classe *SpicinessValuePartition* como subclasse de *ValuePartition*.
3. Criam-se as classes *Mild*, *Medium*, *Hot* como subclasses de *SpicinessValuePartition*.
4. Aplica-se a disjunção as classes *Mild*, *Medium* e *Hot*.
5. Cria-se a classe união de *Mild*, *Medium* e *Hot* como subclasse de *SpicinessValuePartition*.
6. Cria-se uma *Object Property* (*propriedade de objeto*) *hasSpiciness*.
7. A propriedade *hasSpiciness* se torna funcional.
8. Define-se *SpicinessValuePartition* como o escopo da propriedade *hasSpiciness*.



**Figura 4.59: classes adicionadas pelo assistente Create ValuePartition (Criar Partição)**

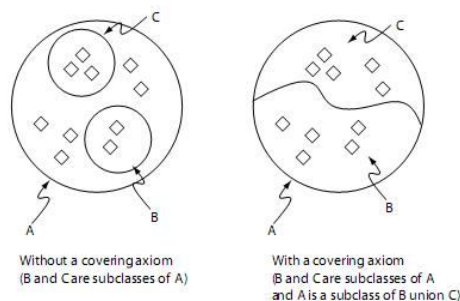


**Figura 4.60: o campo *Asserted Conditions (Condições Declaradas)* mostrando a descrição da classe *SpicinessValuePartition***

#### 4.14.1) Axiomas de Cobertura

Como parte do padrão *ValuePartition* usou-se um *Covering Axiom (Axioma de Cobertura)*, o qual consiste de duas partes: a classe que está sendo coberta, e as classes que formam a cobertura. Por exemplo, tem-se três classes A, B e C, e as classes B e C são subclasses de A. Tem-se um *Covering Axiom (Axioma de Cobertura)* que especifica que a classe A é coberta pela classe B e também pela classe C. Isto significa que um membro da classe A deve ser membro da classe B e/ou C. Se as classes B e C são disjuntas, então um membro de A deve ser um membro de B ou de C. Em geral, embora B e C sejam subclasses de A, um indivíduo pode ser um membro de A sem ser membro de uma das classes B ou C.

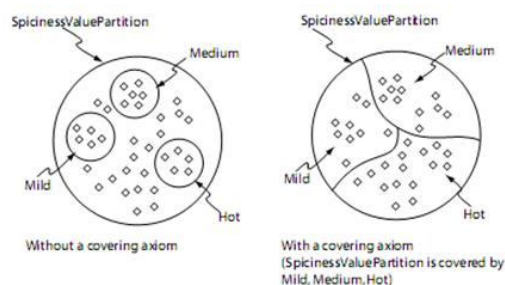
No *Protege-OWL* um *Covering Axiom (Axioma de Cobertura)* manifesta-se como uma classe que é a união das classes que estão sendo cobertas, as quais formam a superclasse da classe que está sendo coberta. No caso de A, B e C, a classe A pode ter uma superclasse de B e C. O efeito de um *axioma de cobertura* é representado na Fig. 4.61.



**Figura 4.61: um esquema que mostra o efeito de usar um *Covering Axiom* para cobertura da classe A com as classes B e C**

A *SpicinessValuePartition* tem uma *axioma de cobertura* para indicar sua *cobertura* pelas classes *Mild*, *Medium* e *Hot*, as quais são disjuntas para que um indivíduo não possa ser um membro de mais de uma delas. A classe *SpicinessValuePartition* tem uma superclasse que é *Mild* e *Medium* e *Hot*. A cobertura indica que um membro de *SpicinessValuePartition* deve ser membro de uma das classes: *Mild* ou *Medium* ou *Hot*.

A diferença entre usar ou não um *axioma de cobertura* é representada na Fig. 4.62. Em ambos os casos, as classes *Mild*, *Medium* e *Hot* são disjuntas, elas não se sobrepõem. No caso de não utilização do *axioma de cobertura*, um indivíduo pode ser um membro da classe *SpicinessValuePartition* e não ser membro de *Mild*, *Medium* ou *Hot*, uma vez que *SpicinessValuePartition* não é coberta por *Mild*, *Medium* e *Hot*. Compare com o caso em que a cobertura de *axioma* é usada. Se um indivíduo é membro da classe *SpicinessValuePartition*, ele *deve* ser membro de uma das três subclasses *Mild*, *Medium* ou *Hot*, uma vez que *SpicinessValuePartition* é coberta por *Mild*, *Medium* e *Hot*.



**Figura 4.62: o efeito de usar um *covering axiom* sobre *SpicinessValuePartition***

#### 4.15) O *Property Matrix Wizard* (Assistente Matriz de Propriedade)

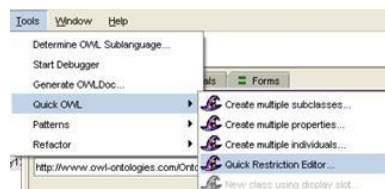
Pode-se usar o *SpicinessValuePartition* para descrever os temperos dos recheios de *Pizza*. Para fazer isto, adicione uma restrição existencial para cada tipo de *PizzaTopping* (*Recheio de Pizza*), de forma a indicar que ela é apimentada. A forma da restrição é **E** *hasSpiciness SpicinessValuePartition*, onde *SpicinessValuePartition* é *Mild*, *Medium* ou *Hot*. Como existem mais de vinte recheios na ontologia de pizza, o trabalho pode ser demorado. Para ajudar nesse trabalho existe o *Property Matrix Wizard* (*Assistente Matriz de Propriedade*), o qual é usado para adicionar restrições existenciais junto a propriedades de muitas classes.

N do T: Na versão 3.4 (Build 130) do Protege, o *Property Matrix Wizard* não está presente com esse nome. O recurso correspondente é o *Quick Restriction Editor*. Algumas funcionalidades são diferentes daquelas apresentadas no tutorial, como por exemplo, os botões de transferência (>> e <<) que não existem no *Quick Restriction Editor* (os objetos devem ser arrastados). Assim, os exercícios seguintes foram adaptados para que sua realização se tornasse possível na versão do Protege mencionada.

### **Exercício 40: uso do *Assistente Matriz de Propriedade* para especificar o tempero de recheios de *Pizza***

---

1. Inicie o assistente através do comando *Quick Restriction Editor* (*Editor de Restrição Rápido*), disponível no menu *Tools* à *Quick OWL* (ver FIG. Extra 9)



**Figura Extra 9: menu do *Editor de Restrição Rápido***

2. A primeira página do assistente solicita a seleção das classes. Selecione todas as classes de recheios de pizza e arraste-as para a lista do lado direito, conforme apresentado na FIG. 4.63. Selecione somente as classes que são recheios reais, e assim, classes como *CheeseTopping* não devem ser selecionadas. Pressione *Next*.
3. Selecione a propriedade *hasSpiciness* e arraste-a para a coluna da direita. Pressione *Next*.
4. Especifique os *fillers* das propriedades clicando no campo *Default* (*padrão*) de cada classe e selecionando um valor: *Mild*, *Medium* ou *Hot*. Selecione o *filler* de *Mild* para todas as propriedades, exceto para *PepperoniTopping* e *SalamiTopping*, que devem ter *fillers* *Medium*, e exceto para *JalapenoPepperTopping* e *SpicyBeefTopping*, os quais devem ter *fillers* *Hot*. A tela deve estar similar a apresentada na FIG. 4.65.
5. O próximo passo solicita que se atualize cada classe alterada para a dada propriedade. Deixe a caixa *Close* desmarcada.
6. Pressione *Finish* para fechar o assistente e criar as restrições sobre os recheios. Selecione alguns recheios e observe que possuem restrições através da propriedade *hasSpiciness*, com *fillers* de subclasses da partição *SpicinessValuePartition*.



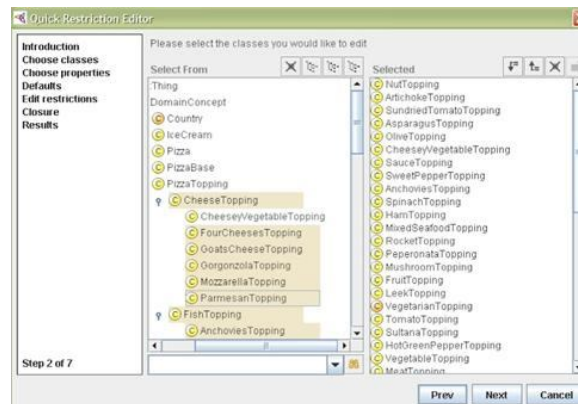


Figura 4.63: página de seleção de classes no *Quick Restriction Editor*

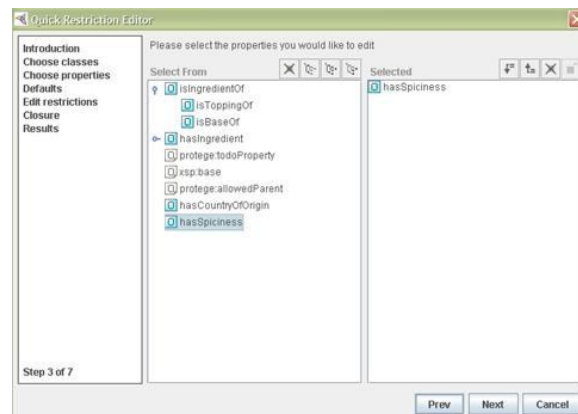


Figura 4.64: página de seleção de propriedades no *Quick Restriction Editor*

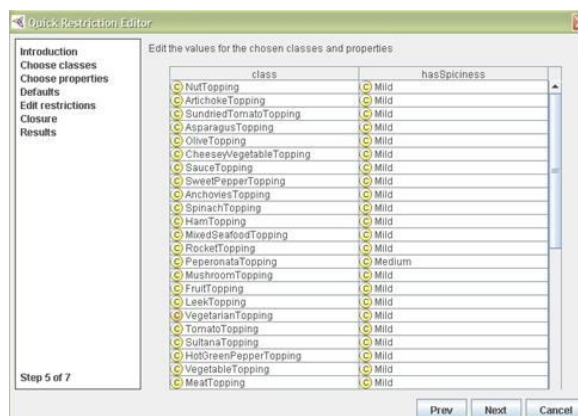


Figura: 4.65: painel de fillers de restrição no *Quick Restriction Editor*

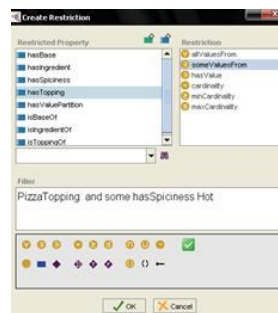
Para completar esta seção, cria-se uma nova classe *SpicyPizza*, que deve ter *Pizzas* com recheios apimentados como subclasses. Para fazer isto define-se a classe *SpicyPizza* como uma *Pizza* com pelo menos um recheio (*hasTopping*) e um tempero apimentado (*hasSpiciness*) que é *Hot*. Isto pode ser feito de mais de uma maneira: vai-se criar uma restrição na propriedade *hasTopping* que tem uma restrição sobre a propriedade *hasSpiciness* como seu filler.

### Exercício 41: criação de uma classe *SpicyPizza* como uma subclasse de *Pizza*

1. Criar uma subclasse de *Pizza* chamada *SpicyPizza*.
2. Selecione *SpicyPizza* na hierarquia de classes, escolha *NECESSARY & SUFFICIENT* em *Asserted Conditions* (Condições Declaradas).



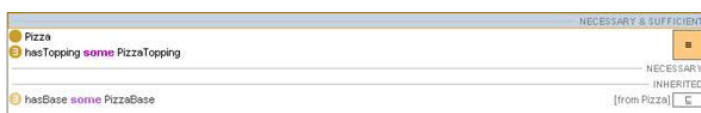
3. Pressione o botão *Create restriction* (*Criar Restrição*) no campo *Asserted Conditions* (*Condições Declaradas*) para mostrar o diálogo.
4. Selecione *E someValuesFrom* como o tipo de restrição.
5. Selecione *hasTopping* como a propriedade a sofrer restrição.
6. O *filler* deve ser *PizzaTopping* é **E** *hasSpiciness Hot*. Este *filler* descreve uma classe anônima, a qual contém os indivíduos que são membros da classe *PizzaTopping* e também membros da classe de indivíduos relacionados aos membros da classe *Hot* através da propriedade *hasSpiciness*. Em outras palavras, diz respeito as coisas que são *PizzaToppings* e tem um *spiciness* (*tempero apimentado*) que é *Hot*. Para entrar com esta restrição como um tipo de *filler*, digite no campo de edição de *fillers* a declaração: *PizzaTopping AND SOME hasSpiciness Hot*.  
A palavra AND é convertida para o símbolo de interseção é, a palavra SOME é convertida para o símbolo de quantificador existencial E.
7. O diálogo *Create Restriction* (*Criar Restrição*) deve agora estar similar a FIG. 4.67. Pressione Ok para fechar o diálogo e criar a restrição.



**Figura 4.67: Diálogo *Create Restriction*: uma restrição que descreve um *SpicyTopping***

8. Arraste *Pizza* de NECESSARY para a recém criada restrição: (*E hasTopping* (*PizzaTopping* é **E** *hasSpiciness Hot*)).

O campo *Asserted Conditions* (*Condições Declaradas*) deve estar agora similar a FIG.4.66.



**Fig. 4.66: a definição de *SpicyPizza***

**SIGNIFICADO** = a descrição apresentada para *SpicyPizza* diz que todos os membros de *SpicyPizza* são *Pizzas* e tem pelo menos um recheio com um *spiciness* (*Tempero Apimentado*) *Hot* (*muito apimentado*). Informa também que qualquer coisa que é uma *Pizza* e tem pelo menos um recheio com *spiciness* (*tempero apimentado*) *Hot* (*muito apimentado*) é um *SpicyPizza*.

**OBSERVAÇÃO** = no passo final do exercício anterior, criou-se uma restrição utilizando a expressão de classe (*PizzaTopping* é **E** *hasSpiciness Hot*) ao invés de uma classe nomeada com seu *filler*. Tal *filler* foi construído pela interseção entre a classe nomeada *PizzaTopping* e a restrição **E** *hasSpiciness Hot*. Uma outra forma para fazer isto é criar uma subclasse de *PizzaTopping* chamada *HotPizzaTopping* e defini-la com um recheio *hot* (*tempero apimentado*) a partir da condição necessária **E** *hasSpiciness Hot*. Pode-se então usar **E** *hasTopping HotPizzaTopping* na definição de *SpicyPizza*.

Agora vai-se chamar o MI e determinar quais pizzas são *spicy* (*apimentadas*) na ontologia.

## Exercício 42: uso do MI para classificar a ontologia

---

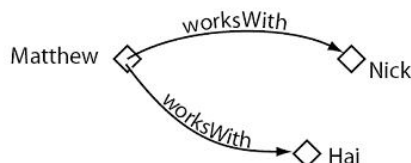
1. Pressione o botão *Classify Taxonomy* (*Classificar Taxonomia*) na barra de ferramentas do OWL para chamar o MI e classificar a ontologia.

Depois de realizada a classificação automática, surge o painel *Inferred Hierarchy* (*Hierarquia Inferida*). Observe que *AmericanHotPizza* foi classificada como subclasse de *SpicyPizza*: o MI computou automaticamente que qualquer indivíduo que é membro de *AmericanHotPizza* é também membro de *SpicyPizza*.

### 4.16) Restrições de Cardinalidade

Em OWL pode-se descrever a classe dos indivíduos que tem *pelo menos um*, ou *no máximo* ou *exatamente* um número específico de relacionamentos com outros indivíduos ou *datatype values* (*valores de tipos de dados*). As restrições que descrevem essas classes são conhecidas como *Cardinality Restrictions* (*Restrições de Cardinalidade*). Para uma dada propriedade P, uma *Minimum Cardinality Restriction* (*Restrição de Cardinalidade Mínima*) especifica o número mínimo de relacionamentos P dos quais um indivíduo deve participar. Uma *Restrição Cardinalidade Máxima* (*Maximum Cardinality Restriction*) especifica o número máximo de relacionamentos P dos quais um indivíduo pode participar. Uma *Cardinality Restriction* (*Restrição de Cardinalidade*) especifica o número exato de relacionamentos P dos quais um indivíduo participa.

Os relacionamentos (por exemplo, entre dois indivíduos) são considerados como relacionamentos separados quando se pode garantir que também são distintos os indivíduos que funcionam como *fillers* dos relacionamentos. Por exemplo, a FIG. 4.68 apresenta o indivíduo *Matthew* relacionado ao indivíduos *Nick* e *Hai*, através da propriedade *worksWith*. O indivíduo *Matthew* satisfaz uma restrição de *cardinalidade mínima 2* na propriedade *worksWith*, caso os indivíduos *Nick* e *Hai* sejam indivíduos distintos.



**Figura 4.68: Cardinality Restrictions (Restrições de Cardinalidade): Counting Relationships**

Adiciona-se uma *restrição de cardinalidade* a ontologia de *Pizza*. Vai-se criar uma nova subclasse de *Pizza* chamada *InterestingPizza* (*Pizza interessante*), a qual será definida com três ou mais recheios.

## Exercício 43: criação de *InterestingPizza* (*Pizza Interessante*) com pelo menos 3 recheios.

---

1. Selecione a classe *Pizza* na etiqueta *OWLClasses*.
2. Crie uma subclasse de *Pizza* chamada *InterestingPizza*.
3. Selecione o *NECESSARY & SUFFICIENT* no campo *Asserted Conditions* (*Condições Declaradas*).

4. Pressione o botão *Create restriction (Criar Restrição)* para abrir o diálogo correspondente.
5. Selecione  $\geq \text{minCardinality}$  como o tipo de restrição.
6. Selecione *hasTopping* como propriedade a sofrer a restrição.
7. Especifique uma *Minimum Cardinality Restriction (Restrição de Cardinalidade Mínima)* de 3, digitando o número 3 na caixa de edição do *filler*.
8. Pressione Ok para fechar o diálogo e criar a restrição.
9. O campo *Asserted Conditions (Condições Declaradas)* deve estar com uma condição *NECESSARY* de *Pizza*, e uma condição *NECESSARY & SUFFICIENT* de *hasTopping  $\geq 3$* . É preciso que *Pizza* seja parte das condições *NECESSARY & SUFFICIENT*. Arraste *Pizza* e solte sobre condição *hasTopping  $\geq 3$* .

O campo *Asserted Conditions* deve estar similar a imagem apresentada na FIG. 4.69.



**Figura 4.69:** o campo *Asserted Conditions* mostrando a descrição de *InterestingPizza*

**SIGNIFICADO** = O que significa isso? A definição de *InterestingPizza* (*Pizza Interessante*) descreve o conjunto de indivíduos que são membros da classe *Pizza* e que tem *pelo menos* três relacionamentos *hasTopping* com outros indivíduos (distintos um do outro).

## Exercício 44: uso do MI para classificar a ontologia

1. Pressione *Classify Taxonomy (Classificar Ontologia)* na barra de ferramentas *OWL*.

Após a classificação automática da ontologia, surge a janela *Inferred Hierarchy (Hierarquia Inferida)*. Abra a hierarquia e observe que *InterestingPizza* agora tem como subclasses *AmericanaPizza*, *AmericanHotPizza* e *SohoPizza*. Observe ainda que *MargheritaPizza* não foi classificada sob *InterestingPizza* porque tem apenas dois tipos distintos de recheio.

## 5. Mais sobre OWR-*Open World Reasoning* (Raciocínio de Mundo Aberto)

Os exemplos dessa seção demonstram novas nuances do *Raciocínio Aberto de Mundo*.

Vai-se criar uma classe *NonVegetarianPizza* (*PizzaNãoVegetariana*) para complementar a categorização de *Pizzas* de *VegetarianPizzas*. A *NonVegetarianPizza* deve conter todas as *Pizzas* que não são *VegetarianPizzas*. Para isso, cria-se uma classe que é o complemento de *VegetarianPizza*.

Uma *complement class* (*classe complemento*) contém todos os indivíduos que não estão contidos na classe da qual ela é complemento. Portanto, ao criar *NonVegetarianPizza* (subclasse de *Pizza*) como complemento de *VegetarianPizza*, ela deve conter todas as *Pizzas* que não são membros de *VegetarianPizza*.

### **Exercício 45: criação de *NonVegetarianPizza* , subclasse de *Pizza* e disjunta de *VegetarianPizza***

---

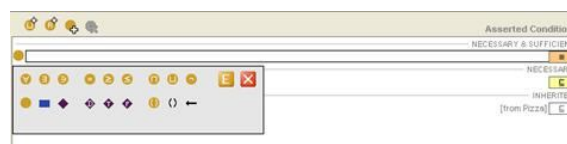
1. Selecione *Pizza* na hierarquia de classes da etiqueta *OWLClasses*. Pressione o *Create subclass* (*Criar Subclasse*) para criar uma nova subclasse de *Pizza*.
2. Renomeie a nova classe como *NonVegetarianPizza*.
3. Torne disjuntas a classes *NonVegetarianPizza* e a classe *VegetarianPizza*. Selecione *NonVegetarianPizza* e pressione o *Add disjoint class...* (*Adicionar Classe Disjunta*) no campo *Disjoint Classes* (*Classes Disjuntas*).

Deseja-se definir uma *NonVegetarianPizza* como uma *Pizza* que não é *VegetarianPizza*.

### **Exercício 46: *NonVegetarianPizza* complemento de *VegetarianPizza***

---

1. Selecione *NonVegetarianPizza* na hierarquia de classe da etiqueta *OWLClasses*.
2. Selecione *NECESSARY & SUFFICIENT* no campo *Asserted Conditions* (*Condições Declaradas*).
3. Pressione o botão *Create new expression* (*Criar nova expressão*), abrindo o *Inline Expression Editor* (*Editor de Expressões*) no campo *Asserted Conditions*, conforme FIG. 5.1. O *Editor de Expressões* contém uma caixa de edição e um painel de construção de expressões, que pode ser usado para inserir nomes de classes e símbolos lógicos.



**Figura 5.1: *Inline Expression Editor* (*Editor de Expressões*)**

4. Digite *not VegetarianPizza* na caixa de edição. A palavra chave NOT é convertida para o símbolo ! *complement of* (*complemento de*). Como alternativa para inserir a expressão no painel para construção, use o botão *InsertComplementOf* (*InserirComplementoDe*), conforme FIG. 5.2, para inserir o símbolo

*complementOf* e use o botão *Insert class (Inserir Classe)* para exibir o diálogo no qual pode-se selecionar *VegetarianPizza*.



**Figura 5.2:** uso do *Expression Builder Panel (Painel Construtor de Expressões)* para inserir *Complement Of*

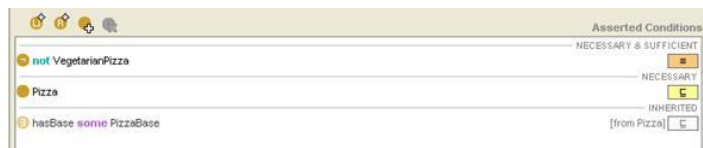
5. Pressione *enter* para criar e atribuir a expressão.



**Figura 5.2:** o recurso auto-completar

**DICA** = um recurso útil do editor de expressões é o *auto-completar*, que funciona para nomes de classes, nomes de propriedades e nomes de indivíduos. O *auto-completar* no *Inline Expression Editor (Editor de Expressões)* é ativado com a tecla *tab*. No exemplo acima, ao digitar "Vege" no editor de expressões e pressionar a tecla *tab*, as opções para completar "Vege" são apresentadas, como mostra na FIG. 5.3. As teclas de *up* e *down* podem ser usadas para selecionar *VegetarianPizza*.

A forma gráfica condições deve agora parecer com a imagem mostrada na FIG. 5.4.



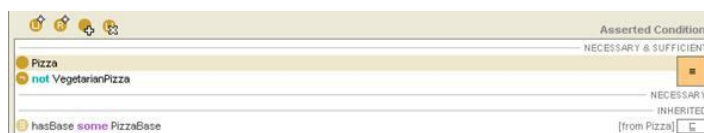
**Figura 5.4:** campo *Asserted Conditions (Condições Declaradas)* mostrando um passo intermediário na criação da definição de *NonVegetarianPizza*.

No entanto, é preciso adicionar *Pizza* as *condições necessárias e suficientes*, uma vez que no momento a definição de *NonVegetarianPizza* diz que um indivíduo que não é um membro da classe *VegetarianPizza* (qualquer outro indivíduo) é uma *NonVegetarianPizza*.

### **Exercício 47: adição de *Pizza* a condições necessárias e suficientes *NonVegetarianPizza***

1. Selecione *NonVegetarianPizza* na hierarquia de classes da etiqueta *OWLClasses*.
2. Selecione *Pizza* no campo *Asserted Conditions (Condições Declaradas)*.
3. Arraste *Pizza* de *NECESSARY*, e solte sobre a condição *! VegetarianPizza* para adicioná-la ao mesmo conjunto de *condições necessárias e suficientes* que *! VegetarianPizza*.

O campo *Asserted Conditions (Condições Declaradas)* deve parecer agora com a FIG.5.5.



**Figura 5.5: O campo *Asserted Conditions* mostrando a definição de *NonVegetarianPizza***

**SIGNIFICADO** = o complemento de uma classe inclui todos os indivíduos que não são membros da classe. Ao tornar *NonVegetarianPizza* uma subclasse de *Pizza* e complemento de *VegetarianPizza*, indica-se que os indivíduos que são *Pizzas* e não são membros de *VegetarianPizza* são membros de *NonVegetarianPizza*. Observe que *VegetarianPizza* e *NonVegetarianPizza* são disjuntas: se um indivíduo é membro de *VegetarianPizza* ele não pode ser um membro de *NonVegetarianPizza*.

### **Exercício 48: uso do MI para classificar a ontologia**

---

1. Pressione *Classify taxonomy* (*Classificar Taxonomia*) na barra de ferramentas OWL. Após classificação, os resultados são apresentados no painel de *Inferred Classes* (*Classe Inferidas*).

A hierarquia de classes inferidas deve parecer com a FIG. 5.6. As classes *MargheritaPizza* e *SohoPizza* foram classificadas como subclasses de *VegetarianPizza*; as classes *AmericanaPizza* e *AmericanHotPizza* foram classificadas como *NonVegetarianPizza*. As coisas *pareciam* estar corretas... Entretanto, vai-se adicionar uma pizza sem axioma de fechamento a propriedade *hasTopping*...

### **Exercício 49: criação de subclasse de *NamedPizza* com recheio de *Mozzarella***

---

1. Criar uma subclasse de *NamedPizza* chamada *UnclosedPizza* (*PizzaSemFechamento*).
2. Selecione *UnclosedPizza* e no campo *Asserted Conditions* (*Condições Declaradas*) selecione *NECESSARY*.
3. Pressione *Create restriction* (*Criar Restrição*) para exibir o diálogo correspondente.
4. Selecione *E someValuesFrom* para criar uma restrição existencial.
5. Selecione *hasTopping* como a propriedade a sofrer restrição.
6. Digite *MozzarellaTopping* na caixa de edição dos *fillers* para especificar que os recheios devem ser indivíduos membros da classe *MozzarellaTopping*.
7. Pressione *Ok* para fechar o diálogo e criar a restrição.

**SIGNIFICADO** = se um indivíduo é um membro de *UnclosedPizza*, é necessário que ele que seja uma *NamedPizza*, e que tenha pelo menos um relacionamento *hasTopping* com um indivíduo que é um membro da classe *MozzarellaTopping*. Por causa da *Suposição de Mundo Aberto* e do fato de que não se adicionou um *axioma de fechamento* a propriedade *hasTopping*, uma *UnclosedPizza* pode ter um recheios adicionais que não são tipos de *MozzarellaTopping*.

## Exercício 50: uso do MI para classificar a ontologia

1. Pressione *Classify taxonomy* (*Classify Taxonomy*) na barra de ferramentas OWL.

Examine a hierarquia de classes e observe que *UnclosedPizza* não é uma *VegetarianPizza* ou uma *NonVegetarianPizza*.

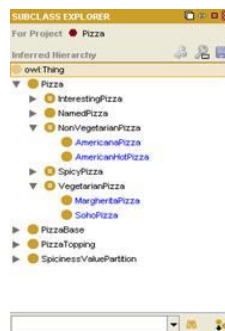


Fig. 5.6:

**SIGNIFICADO** = a *UnclosedPizza* não foi classificada como *VegetarianPizza* (em função do OWR). O MI não pode determinar se *UnclosedPizza* é uma *VegetarianPizza* porque não existe um axioma de fechamento em *hasTopping*, e a *Pizza* pode ter outros recheios. *UnclosedPizza* poderia ser classificada como *NonVegetarianPizza* desde que não fosse classificada como *VegetarianPizza*. No entanto, o OWR não demanda que *UnclosedPizza*, por não é ser um *VegetarianPizza*, seja uma *VegetarianPizza*: ela pode ser *VegetarianPizza* e pode não ser *VegetarianPizza*. Dessa forma, *UnclosedPizza* não pode ser classificada como *NonVegetarianPizza*.



## 6.Outras construções OWL no *Protege-OWL*

Nessa seção discute-se como criar outras construções OWL usando o *Protege-OWL*. Tais construções não são parte principal do tutorial e podem ser criadas em um novo projeto.

### 6.1)Criação de Indivíduos

O OWL permite definir indivíduos e declarar propriedades sobre eles. Os indivíduos podem também ser usados em descrições de classe, a saber, em restrições *hasValue* e *Classes Enumeradas* (veja explicação adiante). Para criar indivíduos no *Protege-OWL* usa-se a etiqueta *Individuals* (*Indivíduos*).

Deseja-se descrever o país de origem dos recheios usados nas *Pizzas*. Em primeiro lugar, é preciso adicionar os países a ontologia, por exemplo, *England* (*Inglaterra*), *Italy* (*Itália*), *America* (*Estados Unidos*), etc. Para criar essa situação na ontologia de *Pizza*, vai-se criar uma classe *Country* (*País*) e então popular essa classe com indivíduos.

### Exercício 51: criação da classe *Country* (*País*) e inserção de indivíduos

1. Crie *Country* como subclasse de *owl:Thing*.
2. Acesse a etiqueta *Individuals* (*Indivíduos*), conforme FIG. 6.1, e selecione *Country*.
3. Pressione o botão *Create Instance* (*Criar Instância*), conforme FIG. 6.2. O termo *Instance* (*Instância*) é outro nome para *Individual* (*Indivíduo*) na terminologia de ontologias.
4. O indivíduo membro de *Country* é criado com um nome genérico atribuído pela ferramenta. Renomeie o indivíduo para *Italy* usando o campo *For Individual*, localizada na etiqueta *Individual* (*Indivíduo*), na parte superior da *Individual Editor*.
5. Use os mesmos passos para criar mais alguns indivíduos membros da classe *Country*, como por exemplo *America*, *England*, *France*, e *Germany*.

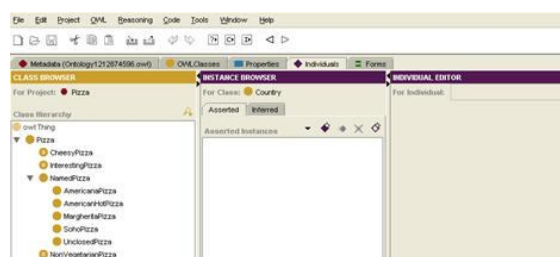


Figura 6.1: A etiqueta *Individuals* (*Indivíduos*)

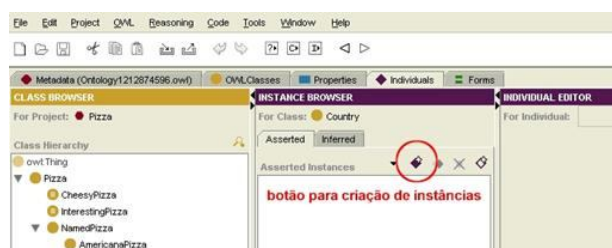
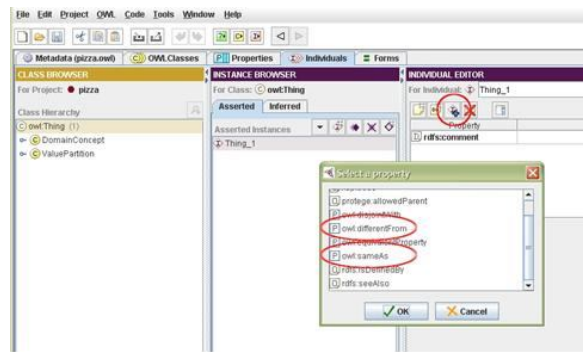


Figura 6.2: botões para manipulação de instâncias



O OWL não usa o UNA-Unique Name Assumption (vide seção 3.2.1). Por isso, os indivíduos podem ser declarados como *SameAs* (*IgualA*) ou *DifferentFrom* (*DiferenteDe*) de outros indivíduos (FIG. 6.3). Tendo criado alguns indivíduos pode-se usá-los em descrições de classes.



**Figura 6.3:** opções *SameAs* (*IgualA*) e *DifferentFrom* (*DiferenteDe*)

## 6.2) Restrições *hasValue* (*Tem Valor*)

Uma restrição *hasValue* (*Tem Valor*), representada pelo símbolo  $\sqcap$ , descreve o conjunto de indivíduos que possui *pelo menos um* relacionamento através da propriedade com *indivíduo específico*. Por exemplo, a restrição *hasValue* representada pela declaração:

*hasCountryOfOrigin*  $\sqcap$  *Italy* (onde *Italy* é um indivíduo) ...

descreve o conjunto de indivíduos (a classe anônima de indivíduos) que tem pelo menos um relacionamento através da propriedade *hasCountryOfOrigin* com o indivíduo *Italy*. Para mais informações sobre restrições *hasValue* vide Apêndice A.2.

Deseja-se especificar a origem dos ingredientes na ontologia de *Pizza*. Por exemplo, pode-se dizer que *MozzarellaTopping* vem da *Italy*. Alguns países foram inseridos na ontologia de *Pizza*, os quais que são representadas como indivíduos. Pode-se usar a restrição *hasValue* junto a esses indivíduos para especificar que o país de origem de *MozzarellaTopping* é *Italy*.

### **Exercício 52: criação da restrição *hasValue* para especificar que *MozzarellaTopping* tem *Italy* como país de origem.**

1. Acesse a etiqueta *Properties* (*Propriedades*), crie uma nova *Object Property* (*Propriedade de Objeto*) e dê-lhe o nome de *hasCountryOfOrigin*.
2. Acesse a etiqueta *OWLClasses* e selecione a classe *MozzarellaTopping*.
3. Selecione *NECESSARY* no campo *Asserted Conditions* (*Condições Declaradas*).
4. Pressione *Create restriction* (*Criar Restrição*) no campo *Asserted Conditions* para abrir o diálogo correspondente.
5. Selecione  $\sqcap$  *hasValue* como o tipo de restrição a ser criada.
6. Selecione *hasCountryOfOrigin* como a propriedade a sofrer a restrição.
7. No campo *Filler* insira o indivíduo *Italy*: digite *Italy* na caixa de edição de *fillers*, ou, pressione o botão *Insert individual* (*Inserir Indivíduos*) no painel de construção de expressões.
8. Pressione *Ok* para fechar o diálogo e criar a restrição.

O campo *Asserted Conditions* (*Condições Declaradas*) deve agora estar similar a Fig. 6.4.



**Figura 6.4:** o campo *Asserted Conditions* mostrando restrição *hasValue* para *MozzarellaTopping*

**SIGNIFICADO** = as condições definidas para *MozzarellaTopping* informam que: indivíduos membros da classe *MozzarellaTopping* são também membros da classe *CheeseTopping* e estão relacionados ao indivíduo *Italy* através da propriedade *hasCountryOfOrigin*; estão relacionadas a pelo menos um membro da classe *Mild* através da propriedade *hasSpicyiness*. Em linguagem natural, pode-se dizer que coisas que são tipos de *MozzarellaTopping* (*RecheioDeMussarela*) são também *CheeseTopping* (*RecheioDe Queijo*), vem da *Italy* e tem *Mild Spyciness* ("mediamente" apimentados).

**ATENÇÃO** = no nível de evolução atual dos MIs, a classificação automática não é completa para os indivíduos. Use indivíduos em descrições de classe com cuidado: resultados inesperados podem ser gerados pelo MI.

### 6.3) Classes enumeradas

O OWL permite que classes sejam definidas listando-se os indivíduos que são seus membros. Por exemplo, pode-se definir uma classe *DaysOfTheWeek* (*DiasDaSemana*) como a classe que contém os indivíduos (e somente os indivíduos): *Sunday* (*Domingo*), *Monday* (*Segunda*), *Tuesday* (*Terça*), *Wednesday* (*Quarta*), *Thursday* (*Quinta*), *Friday* (*Sexta*) e *Saturday* (*Sábado*). Classes definidas dessa forma são conhecidas como *Enumerated Classes* (*Classes Enumeradas*).

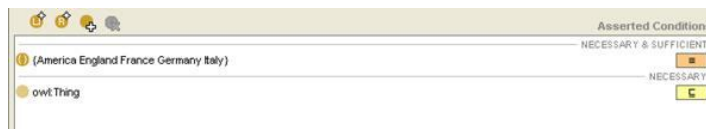
No *Protege-OWL* as classes enumeradas são definidas usando-se o editor de expressões no campo *Asserted Conditions* (*Condições Declaradas*). Os indivíduos que compõem a classe enumerada são listados, separados por espaços e dentro de colchetes, por exemplo: {*Sunday Monday Tuesday Wednesday Thursday Friday Saturday*}. Os indivíduos já devem ter sido criados na ontologia (etiqueta *Individuals*). As classes enumeradas descritas dessa forma são classes anônimas: são as classes dos indivíduos, e apenas dos indivíduos, listados na enumeração. Pode-se anexar esses indivíduos a uma *Named Class* (*Classe Nomeada*) criando a enumeração como uma condição *NECESSARY & SUFFICIENT*.

### Exercício 53: conversão da classe *Country* em uma classe enumerada

1. Selecione a classe *Country* na etiqueta *OWLClasses*.
2. Selecione *NECESSARY & SUFFICIENT* em *Asserted Conditions* (*Condições Declaradas*).
3. Pressione *Create new expression* (*Criar nova expressão*) para abrir o *Inline Expression Editor* (*Editor de Expressões*).
4. Digite {*America England France Germany Italy*} na caixa de edição de *fillers* (lembre-se de usar colchetes). A função auto-completar está disponível: para usá-la digite as primeiras letras de um indivíduo e pressione *tab* para mostrar a lista de opções.

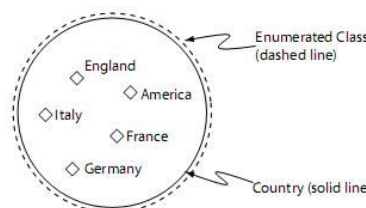
5. Pressione *enter* para aceitar a enumeração e feche o editor de expressão.

O campo *Asserted Conditions* (*Condições Declaradas*) deve parecer agora com a FIG.6.5.



**Figura 6.5:** *Asserted Conditions* mostrando a *Enumeration Class* (Classe Enumerada)

**SIGNIFICADO** = um indivíduo que é um membro da classe *Country* é na verdade um dos indivíduos listados (ou seja, um daquelas da lista *America England France Germany Italy*. Formalmente, a classe *country* é equivalente a (contém os mesmos indivíduos que) a classe anônima que é definida pela enumeração, conforme apresentado na FIG.6.6.



**Figura 6.6:** esquema da classe *Country* como equivalente a uma *Enumerated Class*

**DICA** = existem um assistente para criação de *classes enumeradas*.

#### 6.4) Propriedades de Anotação (*Annotation Properties*)

O OWL permite que classes, propriedades, indivíduos e o próprio cabeçalho ontologia sejam comentados usando metadados. Esses metadados podem assumir a forma de informações de auditoria ou de informações editoriais. Por exemplo: comentários, data de criação, autor, referências para pesquisas tais como páginas *web*, etc. O *OWL-Full* não impõe quaisquer restrições as essas propriedades, mas *OWL-DL* tem restrições ao uso da anotação, sendo que as duas mais importantes são:

- § O *filler* para as propriedades de anotação deve ter um dado literal, uma URI de referência ou um indivíduo. Um dado literal é um caractere de representação de um *datatype value* (*valor de tipo de dados*), por exemplo, "Matthew", "25", "3.11".
- § As propriedades de anotação não podem ser usadas em axiomas que atuam sobre propriedades. Por exemplo, não podem ser usados na hierarquia de propriedades, de forma que não podem ter subpropriedades, ou ser subpropriedade de outra propriedade. Também não podem ter *Domain* (*Domínio*) e *Range* (*Escopo*) específico.

O OWL tem cinco propriedades de anotação pré-definidas que podem ser usadas para fazer comentários em classes (inclusive classes anônimas, tais como restrições), propriedades e indivíduos:

1. *owl:versionInfo*: em geral, o *range* (*escopo*) dessa propriedade é um *string*.
2. *rdfs:label*: o *range* (*escopo*) é um *string*. Esta propriedade é usada para adicionar nomes significativos (para pessoas) aos elementos da ontologia, tais como classes, propriedades e indivíduos. O *rdfs:label* é também ser usado para fornecer nomes multilíngües para elementos de ontologia.

3. *rdfs:comment*: o *range* (escopo) é um *string*.
4. *rdfs:seeAlso*: o *range* (escopo) é uma URI usada para identificar recursos.
5. *rdfs:isDefinedBy*: o *range* (escopo) é uma URI usada para referenciar uma ontologia que define elementos da ontologia tais como classes, propriedades e indivíduos.

Por exemplo, a propriedade de anotação *rdfs:comment* é usada para guardar comentários sobre as classes dos *plug-ins* do *Protege-OWL*. A propriedade de anotação *rdfs:label* pode ser usada para fornecer nomes alternativos para classes, propriedades, etc.

Existem também propriedades de anotação que podem ser usadas para comentar uma ontologia. A propriedades de anotação de ontologias (listadas abaixo) tem como *range* (escopo) uma URI, usada para referência a outra ontologia. É também possível usar a propriedade de anotação *owl:VersionInfo* para comentários de ontologia.

§ *owl:priorVersion*: identifica versões anteriores da ontologia.

§ *owl:backwardsCompatibleWith*: identifica versão anterior da ontologia que é compatível com a versão atual. Isso quer dizer que todos os identificadores da versão anterior possuem o mesmo significado na versão atual. Assim, ontologias e aplicações que fazem referência a versão anterior podem alterar a referencia para a nova versão.

§ *owl:incompatibleWith*: identifica a versão anterior de uma ontologia que não é compatível com a atual.

Para criar propriedades de anotação, usam-se os botões *Create annotation datatype property* (Criar anotação de propriedade de tipo de dados) e *Create annotation object property* (Criar anotação de propriedade de objetos), localizados na etiqueta *Properties* (Propriedades). Para usar as propriedades de anotação, acessa-se a interface de anotações, conforme apresentado na FIG.6.7.

A interface de anotação está disponível em todas as etiquetas (*OWL-Classes*, *Properties*, *Individuals* e *Metadata*) possibilitando anotações em classes, propriedades, indivíduos e ontologias, respectivamente. Uma anotação pode também ser adicionada a restrições e a outras classes anônimas clicando com o botão direito no campo *Asserted Conditions* (Condições Declaradas) e selecionando *Edit annotation properties...* (Editar propriedades da anotação...).



**Figura 6.7: a interface de anotações**

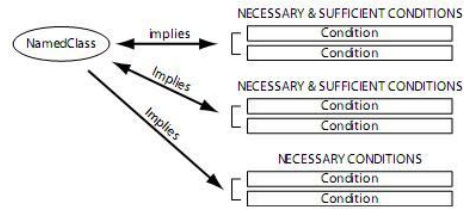
## 6.5) Conjuntos múltiplos de Condições Necessárias e Suficientes

No OWL é possível ter conjuntos múltiplos de *condições necessárias e suficientes*. No campo *Asserted Conditions* (Condições Declaradas), conjuntos múltiplos de condições necessárias e suficientes são representados usando múltiplos cabeçalhos *NECESSARY & SUFFICIENT*, com as respectivas condições listadas sob cada cabeçalho, conforme FIG.6.8.

Para criar um novo conjunto de *condições necessárias e suficientes*, seleciona-se um cabeçalho *NECESSARY & SUFFICIENT* disponível e cria-se então a condição, por exemplo, usando o diálogo *Create Restriction* (como alternativa, uma condição é arrastada sobre um *NECESSARY & SUFFICIENT*).

Para adicionar uma condição a um conjunto existente de *condições necessárias e suficientes*, uma das

condições do conjunto é selecionada e então a nova condição é criada, ou uma condição existente é arrastada sobre o conjunto existente (sobre o cabeçalho *NECESSARY & SUFFICIENT*).



**Figura 6.8:** conjuntos múltiplos de condições necessárias e suficientes

### **Exercício 54: criação de classe para definir um *Triangle* (triângulo) usando múltiplos conjuntos de condições necessárias e suficientes**

1. Criar uma subclasse de *owl:Thing* chamada *Polygon* (Polígono).
2. Criar uma subclasse de *Polygon* chamada *Triangle*.
3. Criar uma *object property* (propriedade de objeto) chamada *hasSide* (temLado).
4. Criar uma *object property* (propriedade de objeto) chamada *hasAngle* (temÂngulo).
5. Em *OWLClasses* selecione a classe *Triangle*. Selecione *NECESSARY & SUFFICIENT* no campo *Asserted Conditions* (Condições Declaradas). Pressione *Create restriction* (Criar Restrições) na forma gráfica *Asserted Conditions* para exibir o diálogo correspondente.
6. Selecione *= cardinality* (= cardinalidade) como tipo de restrição a ser criada. Selecione *hasSide* como a propriedade a sofrer restrição. Na caixa de edição do *filler* digite 3. Pressione Ok para fechar o diálogo e criar a restrição.
7. Selecione novamente *NECESSARY & SUFFICIENT* no campo *Asserted Conditions*. Pressione o botão *Create restriction* para exibir o diálogo correspondente.
8. Selecione *= cardinality* (= cardinalidade) como o tipo de restrição a ser criado. Selecione *hasAngle* como a propriedade a sofrer restrição. Na caixa de edição dos *fillers* digite 3. Pressione Ok para fechar o diálogo e criar a restrição.
9. Arraste *Polygon* de *NECESSARY* para sobre a restrição *hasSide = 3*.
10. Selecione a restrição *hasAngle = 3*. Clique no botão *Add named class...* para exibir um diálogo com hierarquia de classes. Selecione a classe *Polygon* e clique Ok.

N do T: Na versão 3.4 (Build 130) do Protege este exercício apresenta um erro quando se solicita seleção da classe *Polygon*. Aparece a seguinte mensagem de erro: *Could not add a polygon*.

O campo *Asserted Conditions* deve estar similar ao apresentada na FIG. 6.9



**Figura 6.9:** a definição de um triângulo usando condições necessárias e suficientes

## 7.Outros tópicos

### 7.1)Perfil da linguagem

Conforme mencionado anteriormente, existem três sub-linguagens OWL: *OWL-Lite*, *OWL-DL* e *OWL-Full*. Ao editar uma ontologia, o *Protege-OWL* tem capacidade de restringir os constructos usados, de forma que a ontologia possa ser classificada em uma das sub-linguagens, *OWL-DL* ou *OWL-Full*. A sub-linguagem desejada, ou *perfil da linguagem* a ser utilizada é definida através do comando *Preferences* (*Preferências*) do *Protege-OWL*. A escolha entre *OWL-DL* e *OWL-Full* é feita pelo comando *Supports One of the Following OWL Species* (*Suporte a Uma das Seguintes Espécies OWL*), e selecionada dentre uma das duas opções disponíveis: *OWL DL (Optimized for reasoning)* (*otimizado para inferências*), ou o *OWL Full (supports the complete range of OWL elements)* (*suporta todos os elementos OWL*).

### 7.2)Namespaces e importação de ontologias

Ontologias OWL são capazes de importar outras ontologias OWL. Esta seção descreve os *namespaces*, mecanismos para fornecer nomes genéricos, usados para facilitar a importação de ontologias. Descreve-se como importar ontologias.

#### 7.2.1)Namespaces

Cada ontologia tem seu próprio *namespace*, o que é conhecido com o *namespace padrão*. Uma ontologia pode usar outros *namespaces*. Um *namespace* é uma seqüência de caracteres que precede os identificadores de classes, de propriedades e de indivíduos em uma ontologia. É possível a uma ontologia referenciar classes, propriedades e indivíduos em outra ontologia, sem ambigüidades e sem causar problemas com nomes, através da manutenção de *namespaces* distintos para todas as ontologias. Por exemplo, as ontologias OWL (inclusive a ontologia de Pizza) fazem referência a classe *owl:Thing*, a qual reside no vocabulário da ontologia OWL (*namespace* <http://www.w3.org/2002/07/owl#>).

A garantia de que os *namespaces* são único reside em sua representação via URIs-*Unique Resource Identifiers*, terminados com / ou com #. Por exemplo, o *namespace* padrão no *Protege-OWL* (o *namespace* atribuído a ontologias recém-criadas) é <http://a.com/ontology#>. Isto significa que todos identificadores de classes, de propriedades e de indivíduos criados no *Protege-OWL* (por padrão) são prefixados com <http://a.com/ontology#>. Por ex., o nome completo da classe *PizzaTopping* é <http://a.com/ontology#PizzaTopping>. O nome completo da classe *MargheritaPizza* é <http://a.com/ontology#MargheritaPizza>. O *Protege-OWL* esconde os *namespaces* prefixados, de forma que não seja preciso digitar nomes longos a cada uso de classes, propriedades ou indivíduos.

Os *namespaces* contribuem para evitar conflitos entre nomes quando uma ontologia referencia classes, propriedades e indivíduos em uma outra ontologia. Por exemplo, uma ontologia sobre aeronaves, *AircraftOntology*, tem uma classe de nome *Wing*, que descreve a asa de um avião. Uma ontologia sobre pássaros, *BirdOntology*, também tem uma classe chamada *Wing*, que descreve a asa de um pássaro. O *namespace* para a *AircraftOntology* é <http://www.ontologies.com/aircraft#>; o *namespace* para a segunda ontologia, *BirdOntology*, é <http://www.birds.com/ontologies/BirdOntology#>. Evidentemente, a classe *Wing* na ontologia de aeronaves não é a mesma que a classe *Wing* na ontologia de pássaros. Suponha que a ontologia de aeronaves importa a ontologia de pássaros. Por causa do mecanismo *namespace*, o nome completo <http://www.ontologies.com/aircraft#Wing> representa a classe *Wing* na ontologia de aeronaves; e o



nome completo para a classe *Wing* na ontologia de pássaros é <http://www.birds.com/ontologies/BirdOntology#Wing>. Dessa forma, quando a ontologia de aeronaves se refere a classes da ontologia de pássaros não há conflito entre os nomes. Observe que nenhum dos URIs que representam os *namespaces* são URLs, ou seja, não é exigida necessariamente uma localização física na web. O uso de URIs se baseia na garantia de unicidade.

Utilizam-se prefixos de *namespaces* para tornar gerenciável o referenciamento a classes, a propriedades e a indivíduos, quando *namespaces* múltiplos são usados. Um prefixo de *namespace* é um pequeno *string*, uma seqüência de dois ou três caracteres que representa o *namespace* completo. Por exemplo, pode-se usar "ac" para representar o *namespace* da ontologia de aeronaves (<http://www.ontologies.com/aircraft#>) e o prefixo "bird" para o *namespace* da ontologia de pássaros (<http://www.birds.com/ontologies/BirdOntology#>). Ao usar identificadores tais como *nomes de classes*, usa-se o identificador com o prefixo do *namespace* seguido por "dois pontos". Por exemplo, *ac:Wing* ou *bird:Wing*.

O *namespace* padrão corresponde ao *namespace* da ontologia sob edição. Quando estão sendo usados identificadores que pertencem ao *namespace* padrão (a ontologia que está sendo editada) não se usam prefixos: classes, propriedades e indivíduos são referenciados simplesmente pelo uso de seus nomes locais. No entanto, para importar ontologias é preciso usar um prefixo *namespace* para se referir as classes, as propriedades e aos indivíduos da ontologia importada. Por exemplo, suponha a edição da ontologia de aeronaves, que tem um *namespace* <http://www.ontologies.com/aircraft#>, e deseja-se referenciar classes da ontologia de pássaros, cujo *namespace* é outro: <http://www.birds.com/ontologies/BirdOntology#>, com prefixo *namespace* "bird". Ao se referir a classes sem um prefixo *namespace*, por exemplo *Wing*, a referência é considerada para a ontologia de aeronaves. Ao se referir a classes com o prefixo *namespace* "bird", por exemplo *bird:Wing*, está se referindo a classes da ontologia de pássaros.

### 7.2.2) Criação e edição de namespaces no Protege-OWL

#### Edição do *namespace* padrão

O *namespace* padrão é definido na interface *Default Namespace*, localizada no canto superior esquerdo da etiqueta *Metadata* (*Metadados*), conforme mostra a FIG. 7.1. Para mudar o *namespace* padrão digite um novo *namespace* na caixa de edição. O *namespace* deve ser uma URI válida e deve terminar com "/" ou "#". Alguns exemplos válidos são:

- *myNameSpace#*
- *universityOfManchester:ontologies:pizzas#*
- <http://www.cs.man.ac.uk/ontologies/pizzas/>



Figura 7.1: *namespace* padrão e interface de *namespaces*

#### Criação de outros *Namespaces*

Assim como se especifica um *namespace* padrão para a ontologia é possível configurar outros prefixos *namespace* (mapeamentos de *namespaces*). Isto possibilita fazer referência a classes, a propriedades e a indivíduos em outras ontologias.

Usa-se a interface *Namespace Prefixes* (FIG. 7.1) para criar ou para configurar *namespaces* e prefixos associados no *Protege-OWL*. A interface contém três colunas: *Prefix*, *Namespace* e *Imported* (a coluna *Imported* será tratada mais adiante).

N do T: Na versão 3.4 (Build 130) do Protege, conforme pode-se notar na FIG. 7.1, a interface é ligeiramente diferente apresentando apenas duas colunas, a saber *Prefix* e *Namespace*.

Na criação de um novo projeto em OWL, o *Protege-OWL* automaticamente configura os três *namespaces*:

- § rdf - <http://www.w3.org/1999/02/22-rdf-syntax-ns#> (*Resource Description Framework namespace*)
- § rdfs - <http://www.w3.org/2000/01/rdf-schema#> (*RDFSchema namespace*)
- § owl - <http://www.w3.org/2002/07/owl#> (*OWL vocabulary namespace*)

Vai-se adicionar prefixo e *namespace* para uma ontologia de vinho que tem o *namespace* <http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#>. A ontologia de vinhos é um exemplo didático utilizado no guia W3C-OWL que contém dados sobre tipos de vinhos e vinícolas.

### **Exercício 55: criação de *namespace* e pre xo para se referir a classes, a propriedades e a indivíduos na ontologia de vinhos**

---

1. Pressione *Add new prefix* (*Adicionar novo prefixo*) na interface *Namespace prefix* conforme FIG. 7.1 para criar um novo *namespace*. É criado um *namespace* <http://www.domain2.com#> com o prefixo *p1*.
2. Dê um clique duplo no prefixo *p1* para editá-lo. Mude-o para *vin*, o qual é o prefixo do *namespace* usado na ontologia de vinhos.
3. Dê um clique duplo no *namespace* <http://www.domain2.com#> para editá-lo. Mude-o para <http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#>. Se o *namespace* está escrito incorretamente, ou seja, se o *namespace* não é uma URI válida e não termina em '/' ou '#', o Protege-OWL rejeita a entrada e volta ao valor anterior.
4. Agora pode-se referenciar conceitos na ontologia de vinho, e criar classes e propriedades no *namespace* da ontologia de vinho. Por exemplo, criar uma nova *propriedade objeto* e nomeá-la como *vin:myWineProperty*.

A propriedade *myWineProperty* reside no *namespace* *vin* <http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#> (o nome prefixado será *vin:myWineProperty*). O nome completo da propriedade é: <http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#myWineProperty>.

#### **7.2.3) Importação de ontologias em OWL**

Ontologias OWL podem importar uma ou várias outras ontologias. Por exemplo, suponha que *AircraftOntology* (ontologia de aeronaves) importa a *BirdOntology* (ontologia de pássaros), a qual contém descrições de vários pássaros. Todas as classes, as propriedades, os indivíduos e os axiomas contidos na *BirdOntology* vão estar disponíveis para uso na *AircraftOntology*. Isto torna possível a (re) utilização de classes, de propriedades e de indivíduos da *BirdOntology* em descrições de classe da *AircraftOntology*. É possível também *estender* as descrições de classes, de propriedades e de indivíduos na *BirdOntology* pela criação da descrições estendidas na *AircraftOntology*.

Observe a distinção entre *referenciar* as classes, as propriedades e os indivíduos em outra ontologia usando



*namespaces*, e *importar* totalmente a ontologia. Quando uma ontologia importa outra, não é feita uma simples referência às classes, às propriedades e aos indivíduos: os axiomas e fatos contidos na ontologia importada são incluídos na ontologia destino. Observe que o OWL permite importar ontologias de maneira cíclica como, por exemplo, *OntologyA* importa *OntologyB*, e *OntologyB* importa *OntologyA*.

#### 7.2.4) Importação de ontologias no Protege-OWL

A importação de ontologias é normalmente coordenada pelo uso de *namespaces*. Configuram-se o *namespace* e o *prefixo de namespace* da ontologia a importar e em seguida ocorre a importação. Para importar uma ontologia no Protege-OWL é preciso primeiro localizá-la e determinar sua URL.

Importa-se ontologia *Koala*, uma ontologia simples criada por *Holger Knublauch* (autor do plug-in Protege-OWL) para demonstrar possibilidades dos constructos OWL. A ontologia *Koala* está localizada em <http://protege.stanford.edu/plugins/owl/owl-library/koala.owl>.

Importa-se a ontologia *koala* em uma nova ontologia OWL, vazia.

### **Exercício 56: importação da ontologia *koala* para outra ontologia**

---

1. Acesse a etiqueta *Metadata*.
2. Pressione *Add new prefix* (*Adicione novo prefixo*) na interface *Namespace prefix*, criando novos *namespace* e *prefixo namespace*.
3. Edite o *prefixo namespace*, alterando-o para *koala*.
4. Especifique o *namespace*: ao importar ontologias o *namespace* deve ser a URL real onde a ontologia está localizada, seguida por ‘/’ ou ‘#’. Edite o *namespace* para o prefixo *koala*, mudando-o para: <http://protege.stanford.edu/plugins/owl/owl-library/koala.owl#>.
5. Agora clique em *Import ontology...* (*Importar ontologia...*) localizado no canto superior esquerdo do *Ontology Browser* (*Navegador da Ontologia*). Na nova janela marque a opção *Import an ontology from the web by specifying the http://...URL* (*Importar uma ontologia da Web especificando a URL http://...*) e pressione *Next*. Especifique o *namespace* digitando-o no espaço em branco (o endereço está na interface *Namespace prefixes*). Agora, clique em *Finish* para que a ontologia selecionada seja importada.

Depois da execução desses passos, o Protege-OWL importa a ontologia *koala*. Salve e recarregue o projeto. Em seguida, observe que a etiqueta *OWLClasses* contém classes da ontologia *koala*. De forma análoga, a etiqueta *Property* exibe propriedades da ontologia *koala*. Observe ainda que as classes importadas não podem ser editadas ou excluídas: pode-se ter apenas adicionar dados às descrições de classes.

### **Locais Alternativos**

Quando se pretende tornar uma ontologia disponível para importação, uma boa prática é definir a URI do *namespace* como uma URL que aponta para o lugar onde está a ontologia. Na maioria das vezes trata-se de um endereço *web*. Assinalando a caixa de seleção *Imported* (*Importada*), o Protege-OWL tenta achar a ontologia no local especificado pela URI do *namespace*. Mas e se não existe conexão à Internet, ou a ontologia não existe naquela URI? Nesse caso, é possível especificar uma URI/URL alternativa, a qual aponta para uma cópia local da ontologia. Por exemplo, uma URL que aponta para um local no disco

rígido, ou para o servidor rede local, etc. Locais alternativos são especificadas no arquivo de *Ontology Policies* (*Políticas da ontologia*), localizado na pasta do *plug-in* do *Protege-OWL*. Este arquivo não precisa ser editado manualmente, pode ser editado com o diálogo de *política da ontologia*.

Para especificar um local alternativo para uma ontologia importada, siga esses passos.

### **Exercício 57: especificação de local alternativo para uma ontologia importada**

1. Selecione ontologia considerada na interface *Namespaces Prefixes*.
2. Pressione *Import ontology...* (*Importar ontologia...*) localizado na interface *Ontology Browser* (*Navegador da Ontologia*), no canto superior esquerdo. Na nova janela marque a opção *Import an ontology contained in a specific local file* (*Importe uma ontologia de um arquivo local específico*) e pressione *Next*.
3. Na interface *Specify file path* (*Especificar o caminho do arquivo*) digite o endereço ou busque o caminho, clicando no ícone adequado.
4. Selecione a pasta onde a ontologia será salva. Selecione um arquivo da pasta e clique em *Select*. A forma gráfica *Specify file path* é preenchida. Pressione *Next*.
5. Clique em *Finish* para finalizar a importação.

N do T: Na versão 3.4 (Build 130) do Protege o exercício 57 foi realizado de forma ligeiramente diferente do original, o qual apresenta funcionalidades ainda não disponíveis na versão citada.

#### **7.2.5) Importação da ontologia Dublin Core**

A ontologia *Dublin Core* é baseada no *Dublin Core Meta Data Terms*, desenvolvido pela *Dublin Core Meta Data Initiative* (<http://www.dublincore.org/>). Consiste de um conjunto de elementos e termos para descrição de recursos, tais como classes, propriedades e indivíduos de uma ontologia. A definição completa de *Dublin Core Meta Data Terms* está disponível em: <http://www.dublincore.org/documents/dcmi-terms/>

A lista seguinte contém alguns exemplos:

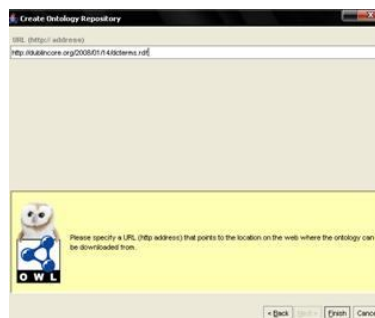
- § *title* (título): um título, um nome pelo qual o recurso é formalmente conhecido.
- § *creator* (criador, autor): uma pessoa, uma organização, ou um serviço; em geral o nome do *creator* é usado para indicar uma entidade.
- § *subject* (assunto): expresso por palavras chaves, frases ou códigos de classificação que descrevem um tópico da pesquisa; a prática recomenda selecionar um termo a partir de um vocabulário controlado ou esquema de classificação.
- § *description* (descrição): inclui resumos, tabela de assuntos, referência para uma representação gráfica de assuntos ou texto livre sobre o assunto, dentre outras;
- § *contributor* (contribuinte): uma pessoa, uma organização ou um serviço; em geral o nome do

*contributor* é usado para indicar a entidade.

Para marcar classes e outras entidades da ontologia com esse tipo de informação, por exemplo, do *Dublin Core*, é preciso importar a ontologia *Dublin Core Metadata Terms (DC Ontology)*. O *Dublin Core Metadata* é usada com frequência, pois o *Protege-OWL* tem um mecanismo automático para importar a *DC Ontology*. Veja os passos apresentados:

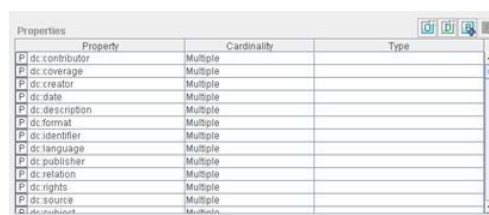
### **Exercício 58: importar a ontologia *Dublin Core Meta Data Elements***

1. A partir do menu *OWL* selecione *Ontology repositories...* (*Repositórios de Ontologias*).
2. Na nova caixa de diálogo, use *Add repository...* (*Adicionar repositório*), localizado no canto superior direito, para inserir o local do repositório. Marque a opção *HTTP Repository* e pressione *finish*.
3. Na janela *Create Ontology Repository (Criar repositório para ontologia)* preencha o campo URL com o endereço *http://dublincore.org/2008/01/14/dcterms.rdf* e pressione *Finish*.



**Figura EXTRA 10: *Create Ontology Repository (Criar repositório para ontologia)***

3. Uma mensagem solicita que a ontologia seja recarregada. Pressione *Yes*.
4. A *DC Ontology* é importada. Feche o diálogo *DC metadata* com o botão *Close*.
5. Alterne para a etiqueta *Properties*. Conforme apresentado na FIG. 7.3, verifique que várias propriedades de anotação (do *Dublin Core Metadata Terms*) foram importadas. Essas propriedades podem ser usadas normalmente.



Property	Cardinality	Type
dc:contributor	Multiple	
dc:coverage	Multiple	
dc:creator	Multiple	
dc:date	Multiple	
dc:description	Multiple	
dc:format	Multiple	
dc:identifier	Multiple	
dc:language	Multiple	
dc:publisher	Multiple	
dc:relation	Multiple	
dc:rights	Multiple	
dc:source	Multiple	
dc:subject	Multiple	

**Figura 7.3: Elementos Dublin Core disponíveis como propriedades de anotação**

#### **7.2.6) *Protege-OWL Metadata Ontology (Ontologia de metadados do Protege-OWL)***

Vários recursos usados pelo *plug-in Protege-OWL* (tal como a marcação de classes para que qualquer subclasse primitiva seja disjunta automaticamente) se baseiam em propriedades de anotação. Essas propriedades são parte da *Protege-OWL Meta Data Ontology*, localizada na pasta do *plug-in Protege-OWL*. Para usar essas propriedades de anotação é preciso importar o *Protege-OWL Meta Data Ontology*.

### **Exercício 59: importação do *Protege-OWL Meta Data Ontology***

1. Selecione *Preferences...* (*Preferências*) no menu OWL.
2. No novo diálogo, marque *Import Protege Metadata Ontology* (*Importar Ontologia de Metadados do Protege*). Pressione *Close* para fechar o diálogo.

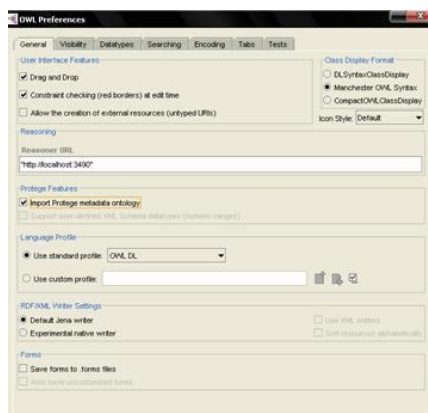


Figura Extra 11: caixa de diálogo *OWL Preferences*

### 7.3) Testes em ontologias

O *Protege-OWL* possibilita a aplicação de vários testes a ontologia em edição. Tais testes variam desde *testes de sanidade*, como por exemplo verificar se uma característica da propriedade corresponde a sua correspondente na propriedade inversa; até *testes OWL-DL*, os quais são projetados para encontrar constructos tais como metaclasses que caracterizam a ontologia como *OWL-Full*. A estrutura de testes é baseada em uma arquitetura de *plug-ins* que permite a adição de novos testes por terceiros. Verifique no site do *Protege-OWL* sobre a disponibilidade de testes *add-on*.

Os testes podem ser configurados através do diálogo *Test Settings* (*Configurações dos Testes*) conforme mostrado na FIG. 7.4, acessível através do comando *Test Settings...* no menu OWL. Para rodar os testes, selecione *Run Ontology Tests...* (*Rodar testes na ontologia*) no menu OWL, ou no botão *Run Ontology Tests...* na barra de ferramenta OWL.

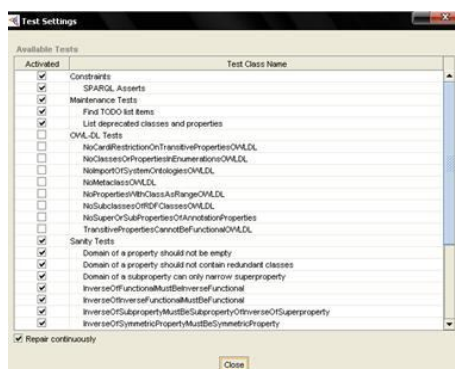


Figura 7.4: o diálogo de testes da ontologia.

Depois do teste na ontologia, os resultados são apresentados em um painel na parte inferior da tela, como mostrado na FIG. 7.5.

Type	Source	Test Result
ProteinConsistentTopping		This class has multiple asserted parents
owl:Thing		Missing disjoints on primitive subclasses: koala:Animal koala:Degree koala:Gender koala:Habitat Pizza:PizzaBase PizzaTo
koala:Marsupials		Missing disjoints on primitive subclasses: koala:Koala koala:Quokka koala:TasmanianDevil
koala:Habitat		Missing disjoints on primitive subclasses: koala:Forest koala:University
koala:Forest		Missing disjoints on primitive subclasses: koala:DryEucalyptForest koala:Rainforest
VegetableTopping		Missing disjoints on primitive subclasses: CapriTopping MushroomTopping OliveTopping OnionTopping PepperTopping Ph
NamedPizza		Missing disjoints on primitive subclasses: AmericanPizza AmericanPizza MargheritaPizza SoftPizza UnclosedPizza
CheeseTopping		Missing disjoints on primitive subclasses: MozzarellaTopping ParmesanTopping ProteinConsistentTopping

Figura 7.5: painel de resultados de testes

O painel do resultado do teste possui as seguintes colunas:

- § *Type* (tipo): corresponde ao tipo de resultado do teste (advertência, erro etc.).
- § *Source* (fonte): corresponde a fonte do resultado do teste (por exemplo, uma classe ou uma propriedade). Um clique duplo sobre a fonte leva a origem da fonte, por exemplo uma classe na etiqueta *OWLClasses*, ou uma propriedade na etiqueta *Properties*.
- § *Test Result* (*Resultados do teste*): uma mensagem que descreve o resultado obtido.

Em alguns casos, o *Protege-OWL* é capaz de modificar ou corrigir aspectos da ontologia em que um teste detectou defeitos. Quando o teste é selecionado, fica disponível um pequeno botão de ferramentas denominado *spanner* (símbolo de uma chave inglesa), do lado esquerdo do painel do resultados. Clicando neste botão, é possível reparar o defeito na ontologia detectado pelo teste.

#### **7.4) *TODO list* (Lista de tarefas a fazer)**

Uma característica e útil simples do *Protege-OWL* é o *TODO List* (*Lista de tarefas a realizar*). Classes, propriedades e mesmo ontologias podem ser marcadas com itens *TODO*. Tais itens são associados as entidades usando o botão *Add TODO List Item* (*Adicionar um item na lista de tarefa*) localizado na interface *Annotation*. Pressionando *Add TODO List Item* cria-se uma nova propriedade de anotação que pode ser preenchida com a descrição textual da tarefa *TODO*. Para localizar os itens *TODO*, selecione o comando *Show TODO List...* (*Mostrar lista de tarefas...*) no menu *OWL*, ou use o botão *Show TODO List...* da barra de ferramentas *OWL*. Abre-se assim uma lista de itens no painel inferior da tela. Com um clique duplo em cada *TODO* da lista, é possível navegar para o item correspondente na ontologia.

# Apêndice A

## Tipos de restrições

Esse apêndice contém informações adicionais sobre os tipos de restrições para propriedades OWL. É indicado para leitores não familiarizados com noções de lógica.

Todos os tipos de restrições descrevem um conjunto sem nome que pode conter indivíduos. Esse conjunto corresponde a uma classe anônima. Quaisquer indivíduos membros dessa classe anônima satisfazem a restrição que descreve a classe (FIG. A.1). O termo "*restrições*" descreve restrições sobre relações através de propriedades, das quais participam indivíduos.

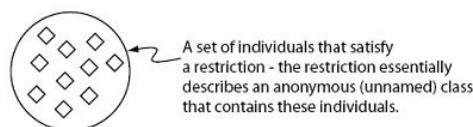
Quando se descreve uma classe nomeada usando restrições, o que se faz realmente na prática é descrever uma superclasse anônima da classe nomeada.

### A.1 Restrições de quantificação

Restrições de quantificação possuem três partes:

1. O *quantificador*, que pode ser um *quantificador existencial* (E) ou um *universal* (A);
2. Uma *propriedade*, sobre a qual a restrição atua;
3. Um *filler*, que corresponde a *classe nomeada* cujos indivíduos atendem a restrição.

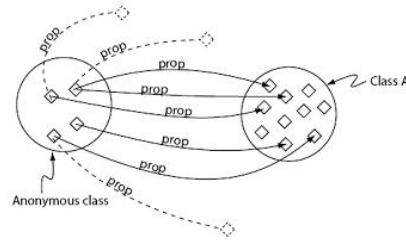
O quantificador impõe restrições sobre um relacionamento do qual um indivíduo participa. Isso ocorre porque o quantificador especifica que pelo menos um tipo de relacionamento deve existir, ou especifica os tipos de relacionamentos que podem existir.



**Figura A.1: restrições que descrevem uma classe de indivíduos anônima**

#### A.1.1 *someValuesFrom* – Restrições Existenciais

*Restrições existenciais*, conhecidas como *someValuesFrom*, ou *Some*, são representadas pelo símbolo E ("E" ao contrário). As restrições existenciais descrevem o conjunto de indivíduos que tem *pelo menos um* tipo específico de relacionamento com indivíduos membros de uma classe. A FIG. A2 apresenta uma visão esquemática de uma restrição existencial (E *prop ClassA*), ou seja, uma restrição sobre a propriedade *prop* com um descritor *ClassA*. Observe que todos os indivíduos da classe anônima definida pela restrição têm pelo menos uma relação através da propriedade *prop* com um indivíduo que é membro da classe *ClassA*. As linhas pontilhadas (FIG. A2) representam o fato de que os indivíduos podem ter relacionamentos *prop* adicionais com outros indivíduos que não são membros da classe *ClassA*, mesmo que isso não tenha sido explicitamente estabelecido. A *restrição existencial* não restringe a relação *prop* a membros da classe *ClassA*, apenas estabelece que cada indivíduo deve ter pelo menos um relacionamento *prop* com um membro de *ClassA*. Essa característica recebe o nome de OWA - *Open World Assumption* (*Pressuposição de mundo aberto*).



**Figura A2: esquema de uma restrição existencial ( $E \text{ prop ClassA}$ )**

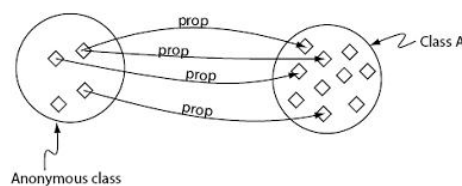
Por exemplo, a restrição existencial  $E \text{ HasTopping MozzarellaTopping}$  descreve o conjunto de indivíduos participante de *pelo menos uma* relação *hasTopping* com outro indivíduo, que é membro da classe *MozzarellaTopping*. Em linguagem natural pode-se dizer que a restrição existencial pode é aquela que descreve as coisas que tem um recheio de mussarela.

O fato de a restrição existencial descrever um grupo de indivíduos que tem pelo menos um relacionamento através da propriedade *hasTopping* com um indivíduo membro da classe *MozzarellaTopping*, não significa que os relacionamentos apenas podem ocorrer através da propriedade *hasTopping*, ou apenas com indivíduo membro de *MozzarellaTopping*. Podem existir outras relações *hasTopping* não especificadas explicitamente.

#### A.1.2 *allValuesFrom* – Restrições Universais

Restrições Universais são também conhecidas como *allValuesFrom*, ou *All*, uma vez que restringem o *filler* de certa propriedade para uma classe específica. Restrições Universais são representadas pelo símbolo A ("A" de cabeça para baixo). Descrevem o conjunto de indivíduos os quais, para uma dada propriedade, tem relacionamento apenas com outros indivíduos, membros de uma classe específica. Para uma dada propriedade, o conjunto de indivíduos descritos pela restrição universal vai também conter os indivíduos que não tem qualquer relacionamento com essa propriedade, para qualquer outro indivíduo.

A restrição universal, apresentada na FIG. A.3. possui a propriedade *prop* e o *filler* *ClassA*. Cabe destacar que uma restrição universal não garante a existência de um relacionamento através da propriedade. Ela simplesmente estabelece que se existe tal relacionamento através da propriedade, então ele deve ocorrer com um indivíduo que é um membro da classe especificada.



**Figura A.3: Uma visão esquemática da restrição universal ( $A \text{ prop ClassA}$ )**

Por exemplo, a restrição  $A \text{ hasTopping TomatoTopping}$  descreve a classe anônima de indivíduos que tem apenas relacionamentos *hasTopping* com indivíduos membros da classe *TomatoTopping*, ou, indivíduos que definitivamente não participam em qualquer outro relacionamento *hasTopping*.

#### A.1.3 Combinação de restrições Existenciais e Universais na descrição de classes

Um procedimento comum é combinar restrições universais e existenciais em definições de classes para certa propriedade. Por exemplo, as seguintes restrições podem ser usadas em conjunto:  $E \text{ hasTopping MozzarellaTopping}$  e  $A \text{ hasTopping MozzarellaTopping}$ .



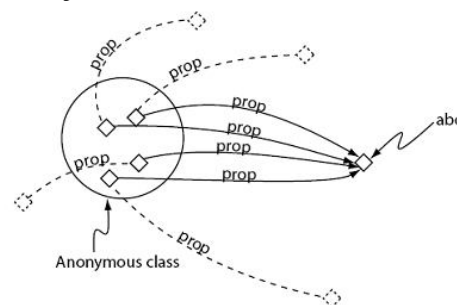
Essa combinação descreve o conjunto de indivíduos que tem pelo menos um relacionamento *hasTopping* com um indivíduo da classe *MozzarellaTopping*, e apenas tem relacionamentos *hasTopping* com indivíduos da classe *MozzarellaTopping*.

Não é comum, e em geral significa um erro, ao descrever uma classe, que se use a restrição universal junto a propriedade, sem uso da restrição universal correspondente junto à mesma propriedade. No exemplo acima, caso se use apenas a restrição universal *A hasTopping Mozzarella*, então se pode ter descrito o conjunto de indivíduos que apenas participa na relação *hasTopping* com membros da classe *Mozzarella*, e também aqueles indivíduos que não participam em qualquer relacionamento *hasTopping* (provavelmente, um erro).

## A.2 Restrições *hasValue*

Uma restrição *hasValue*, representada pelo símbolo  $\sqcap$ , descreve uma classe anônima de indivíduos que estão relacionados a outros indivíduos específicos por uma propriedade. Trata-se de uma situação diferente daquela apresentada na restrição de quantificador, em que os indivíduos descritos pela restrição estão relacionados a qualquer indivíduo de uma classe específica através da propriedade específica. A FIG. 4 mostra uma visão esquemática da restrição *hasValue prop*  $\sqcap$  *abc*.

Essa restrição descreve a classe anônima de indivíduos que tem pelo menos um relacionamento através da propriedade *prop* com o indivíduo *abc*. As linhas pontilhadas (FIG. A.4) representam o fato de que, para um dado indivíduo, a restrição *hasValue* não restringe a propriedade usada na restrição a um relacionamento com o indivíduo participante da restrição, isto é, podem existir outros relacionamentos com a propriedade *prop*. Cabe notar que as restrições *hasValue* são semanticamente equivalentes a uma restrição existencial junto a uma propriedade *hasValue*, a qual tem um *filler* que é uma classe enumerada que contém o indivíduo (e apenas ele) usado na restrição *hasValue*.



**Figura A.4:** restrição *hasValue prop*  $\sqcap$  *abc*, onde as linhas pontilhadas indicam que esse tipo de restrição não restringe a propriedade usada na restrição, a indivíduos usados exclusivamente na restrição *hasValue*.

## A.3 Restrições de cardinalidade

As restrições de cardinalidade são usadas para explicitar o número de relacionamentos em que um indivíduo pode participar para uma propriedade. São conceitualmente simples e apresentadas em três formas: *restrições de cardinalidade mínima*, *restrições de cardinalidade máxima* e *restrições de cardinalidade (exata)*.

### A.3.1 Restrições de cardinalidade mínima

As *restrições de cardinalidade mínima* especificam o número mínimo de relacionamentos que um indivíduo pode participar para uma dada propriedade. O símbolo para a restrição de cardinalidade mínima é o *maior*

ou igual a ( $\geq$ ). Por exemplo, a cardinalidade mínima  $\geq hasTopping\ 3$  descreve os indivíduos (uma classe de anônimos que contém indivíduos) que participam em pelo menos três relacionamentos *hasTopping*. As restrições de cardinalidade mínima não estabelecem limites máximos relativos ao número de relacionamentos que um indivíduo pode participar, em uma propriedade.

### A.3.2 Restrições de cardinalidade máxima

As *restrições de cardinalidade máxima* especificam o número máximo de relacionamentos que um indivíduo pode participar para uma dada propriedade. O símbolo para restrições de cardinalidade máxima é o *menor ou igual a* ( $\leq$ ). Por exemplo, a cardinalidade máxima  $\leq hasTopping\ 2$  descreve a classe de indivíduos que participam no máximo de dois relacionamentos *hasTopping*. Cabe notar que as restrições de cardinalidade máxima não estabelecem limites mínimos sobre o número de relacionamentos que um indivíduo deve participar para uma propriedade.

### A.3.3 Restrições de cardinalidade exata

As *restrições de cardinalidade exata* especificam o número exato de relacionamentos dos quais um indivíduo deve participar, para uma propriedade. O símbolo para as restrições de cardinalidade é o *igual* ( $=$ ). Por exemplo, a cardinalidade  $= hasTopping\ 5$  descreve o conjunto dos indivíduos (a classe anônima de indivíduos) que participam em exatamente 5 relacionamentos *hasTopping*. Cabe notar que a restrição de cardinalidade é, na verdade, uma forma simplificada de combinação entre as cardinalidades mínimas e máximas. Por exemplo, a restrição de cardinalidade acima pode ser representada como uma interseção de duas restrições (mínima e máxima):  $\leq hasTopping\ 5$ , e,  $\geq hasTopping\ 5$ .

### A.3.4 UNA-Unique Name Assumption (Presunção de nome único) e cardinalidades

A OWL adota o UNA, o que significa que diferentes nomes pode se referir ao mesmo indivíduo. Por exemplo, os nomes *Matt* e *Matthew* podem se referir ao mesmo indivíduo ou não. Restrições de cardinalidade consideram indivíduos distintos, e dessa forma, é importante especificar que *Matt* e *Matthew* são o mesmo indivíduo, ou que são diferentes. Seja um indivíduo *Nick*, relacionado a indivíduos *Matt*, *Matthew* e *Matthew Horridge* via propriedade *worksWith*. Também existe um indivíduo *Nick*, membro da classe de indivíduos que trabalham com no máximo dois outros. Uma vez que o OWL não adota o UNAs, ao invés da situação resultar em um erro, a inferência automática indica que dois dos nomes se referem ao mesmo indivíduo. Ao afirmar que *Matt*, *Matthew* e *Matthew Horridge* são indivíduos diferentes, a base de conhecimento pode ser considerada inconsistente.

## Apêndice B

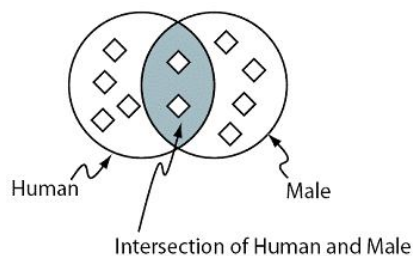
Uma classe OWL é especificada em termos de suas superclasses. Tais superclasses são em geral classes nomeadas e restrições, sendo que as restrições são de fato classes anônimas. As superclasses também podem tomar a forma de "descrições complexas" construídas a partir de descrições simples, conectadas por operadores lógicos. Os operadores mais conhecidos são:

- § AND (é): a classe formada com o operador AND e conhecida como classe interseção; a classe resultante é a interseção das classes individuais;
- § OR (è): a classe formada com o operador OR e conhecida como classe união; a classe resultante é a união das classes individuais.

### B.1 Classes interseção (é)

Uma classe interseção é descrita pela combinação de duas ou mais classes, as quais utilizam o operador AND é. Por exemplo, seja a interseção de *human* (*humano*) e *male* (*masculino*), conforme FIG. B.1: descreve-se a classe anônima que contém indivíduos membros da classe *human* e da classe *male*.

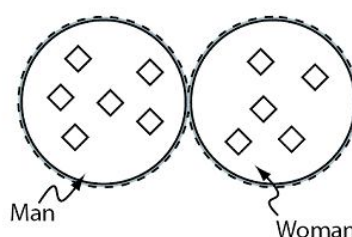
A semântica de uma classe interseção define que a classe anônima descrita é uma subclasse de *human* e também uma subclasse de *male*. Essa classe interseção anônima pode também ser usada em outra descrição de classes. Por exemplo, ao se construir uma descrição da classe *man* especifica-se que *man* é uma subclasse da classe anônima descrita pela interseção de *human* e de *male*. Em outras palavras, *man* é subclasse de *human* e de *male*.



**Figura B.1: a interseção (*Human é Male*); a área escura representa a interseção**

### B.2 Classes união (è)

A classe união é criada pela combinação de duas ou mais classes usando o operador OR è. Por exemplo, seja a união de *man* e *woman*, conforme a FIG. B.2. Ela descreve a classe anônima que contém os indivíduos pertencentes a classe *man* ou a classe *woman* (ou a ambas).



**Figura B.2: união (*Man è Woman*); a área escura representa a união**

A classe anônima descrita pode ser usada em outra descrição de classe. Por exemplo, a classe *person* pode ser equivalente a união de *man* e *woman*.

\* \* \*