The Manchester OWL Syntax Editor Guide

Matthew Horridge

May 10, 2005

Copyright

1 Introduction

This guide explains how to use the Manchester OWL Syntax Editor for Protégé-OWL . The Manchester OWL Syntax Editor provides alternative view and editor for the descriptions of OWL named classes in Protégé-OWL . Figure 1 shows an example of the editor being used to edit part of the CO-ODE tutorial pizza ontology[1].

[1] `http://www.co-ode.org/ontologies`

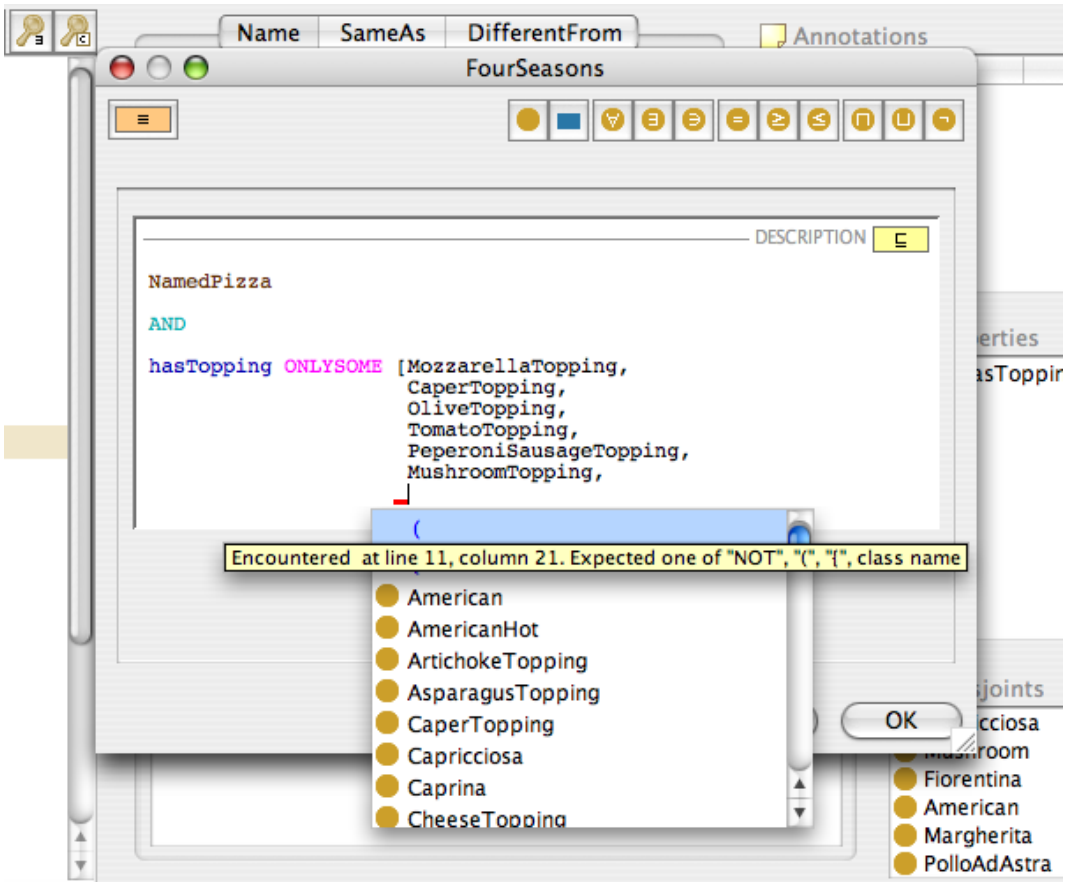

Figure 1: An example of the Manchester OWL Syntax Editor

2 The Manchester Syntax Editor

Some of the features of the editor are:

- **Pretty printing** of class expressions to improve readability, particularly of long expressions.
- **Syntax highlighting** of keywords, classes, object properties, datatype properties and individuals.
- **Auto-completion** of class, property, and individual names and also of keywords and datatypes.
- **Error highlighting** to indicate the precise location of syntax and semantic errors.
- **Auto-suggestions**, making it easy to create classes, properties and individuals whilst editing a class expression.
- **Intermediate representations** for sets of existential restrictions and closure axioms.

3 Using the Editor

3.1 Invoking the Editor

To edit a particular class, select the class in the OWL Classes Tab, and then either right click on the class in the class hierarchy and select "Edit Class Description...". Alternatively, having selected the desired class press CTRL+E on windows or linux, or COMMAND+E on a Mac. After invoking the Manchester OWL Syntax Editor, a dialog box similar to that shown in Figure 2 will be displayed.

3.2 Editor Layout

The editor displays a series of text boxes that allow the class conditions to be edited. Class conditions are displayed in one or more text boxes depending on the description/definition of the class. For example, the picture in Figure 2 shows a class being edited for which two text boxes are displayed. The upper text box contains some necessary & sufficient conditions for the class, whilst the lower text box (that could contain necessary conditions) is empty. The syntax for specifying conditions is described in detail in Section 4.

3.2.1 Class Description and Definitions

OWL distinguishes between *necessary* and *necessary & sufficient* conditions for class membership. *Necessary* conditions represent the conditions that *must* be fulfilled by individuals that are known to be members of the class in question. *Necessary* conditions alone are *not* enough to determine that any random individual that fulfils the conditions is a member of the class in question. The Manchester OWL Syntax Editor refers to a set of *necessary* conditions as a **description**.

Necessary & Sufficient conditions represent conditions that are not only necessary for class membership, but also sufficient to be able to determine that *any* random individual that meets these conditions can be inferred to be a member of the class in question. The Manchester OWL Syntax Editor refers to sets of *Necessary & Sufficient* conditions as **Definitions** – a class may have multiple definitions.[2]

In OWL, a class has *one* (possibly empty) set of *necessary* conditions, and may have zero or more sets of *Necessary & Sufficient* conditions. In the Manchester OWL Syntax Editor the manifestation of this is that there is *always* one *editable* **Description** text box, and zero or more editable **Defintion** text boxes. Even if a class does not have any *Necessary* conditions, a **Description** text box is always displayed. If a class does not have any *Necessary & Sufficient* conditions then no **Definition** text boxes will be displayed.

In the screen shot shown in Figure 2 two text boxes are shown. The upper text box contains *Necessary & Sufficient* conditions for the class being edited – this is the **Definition**. The lower text box displays the *Necessary* conditions – this is the class **Description**. Since the class depicted in Figure 2 does not have any class description (*Necessary* conditions), it is empty.

[2] Classes with *Necessary & Sufficient* conditions are known as *defined* classes. Classes that only have *Necessary* conditions are known as *primitive* classes.

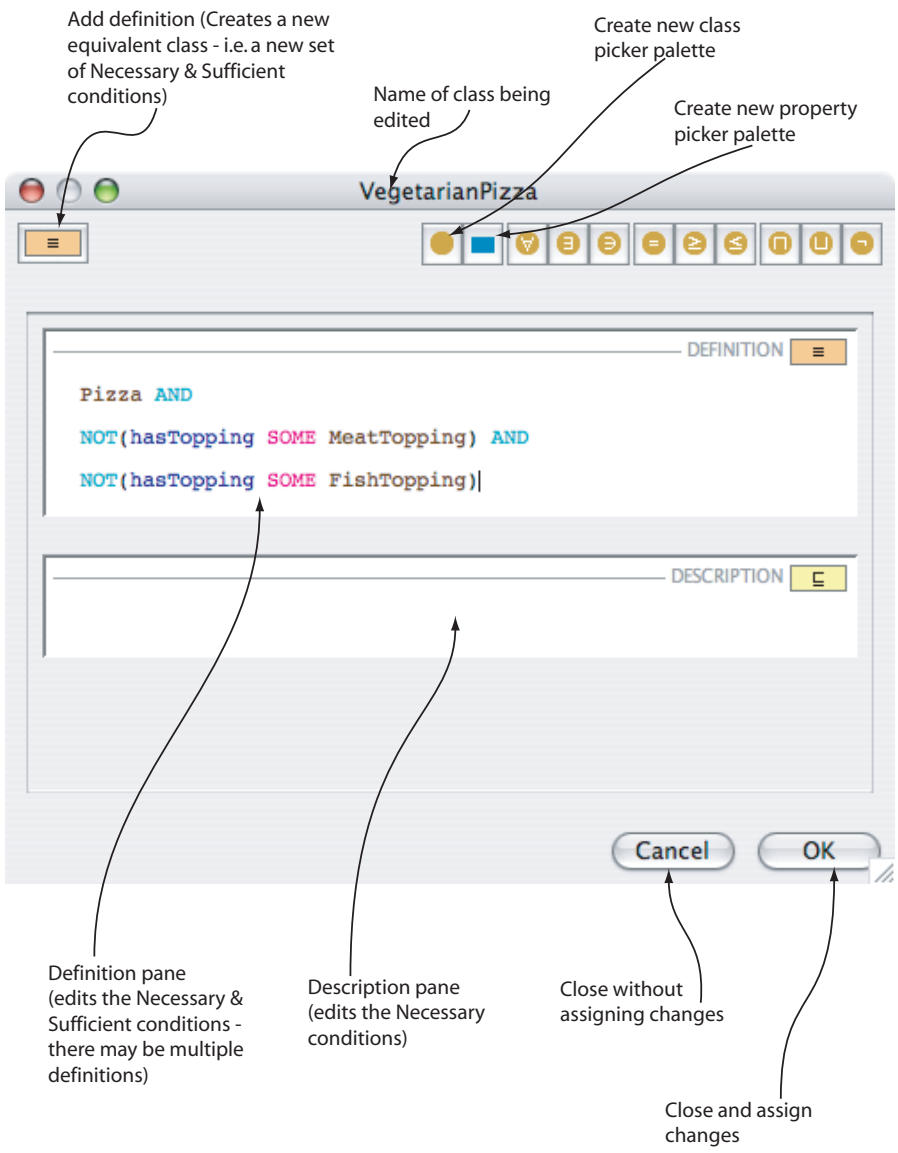

Figure 2: The Manchester OWL Syntax Editor Important Features

3.2.2 Creating Definitions

As previously stated, a class may have zero or more definitions. To add a new definition, the "Add Definition" button which is located in the top left of the editor dialog should be pressed. Figure 3 shows an example of the result of adding an extra definition to the class depicted in Figure 2.

3.2.3 Removing Definitions

The editor automatically removes any 'empty' class definitions when the editor dialog box is closed using the OK button. Therefore, to remove a definition, the text that represents the definition conditions should be deleted – the empty definition box will not be displayed the next time the class is viewed/edited.

3.2.4 Palettes

When editing a large ontology, it is often the case that frequent access is required to a number of specific subsets of classes or properties. In order to facilitate this usage pattern, the Manchester OWL Syntax Editor provides class and property palettes. These are floating windows that can be spawned from the main editor window, which display portions of the class and property hierarchies. Classes and properties in these palettes may be double clicked to insert them into the syntax editor window. Figure 4 shows and example of the class and property palettes. To create a new class or property palette, the buttons labelled in 4 should be used. There is no limit on the number of palettes that may be created. To close an existing palette the system close buttons on the title bar of the palette window should be used. When the editor is closed by pressing the OK or Cancel buttons, the open palettes are temporarily hidden until the editor is invoked once again.

To insert a class from a class palette into the editor, the position at which the class should be inserted should be first selected. This is accomplished by placing the text box caret at the required location (simply click the position in the editor window where the class should be inserted). Having done this, the class in question should be double clicked. The editor will check that it is syntactically correct to insert a class at the selected position–if all is well the class will be inserted–if it is syntactically incorrect to insert a class at the selected position then an error message will be displayed.

4 The Manchester OWL Syntax

The editor uses the Manchester OWL Syntax, which is a new syntax that has been designed for writing OWL class expressions. It has been influenced by both the OWL Abstract Syntax[3] and the DL style syntax which uses description logic symbols such as the universal quantifier ($\forall$) or the existential quantifier ($\exists$).

Although the syntax borrows ideas from the OWL Abstract Syntax, it is much less verbose, meaning that it is quicker to write and easier to read. In a similar vein, whilst following the compactness of the DL Syntax, special mathematical symbols such as $\exists$, $\forall$, $\sqcup$, $\neg$ and $\ni$, have been replaced by keywords such as "SOME", "ONLY", and "NOT". The Manchester OWL Syntax also departs from the DL Syntax by using an infix notation rather than a prefix notation for keywords – in our experience, an infix notation makes the syntax easier to read and understand for non-logicians.

[3] `http://www.w3.org/TR/owl-semantics/syntax.html`

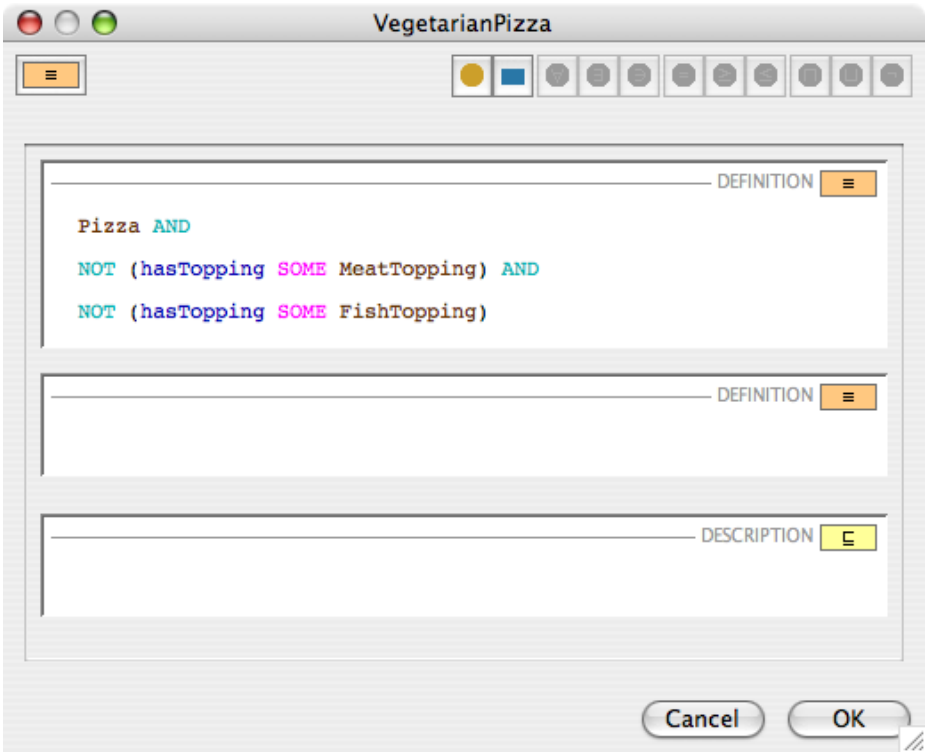

Figure 3: The Manchester OWL Syntax Editor Displaying Multiple Definitions

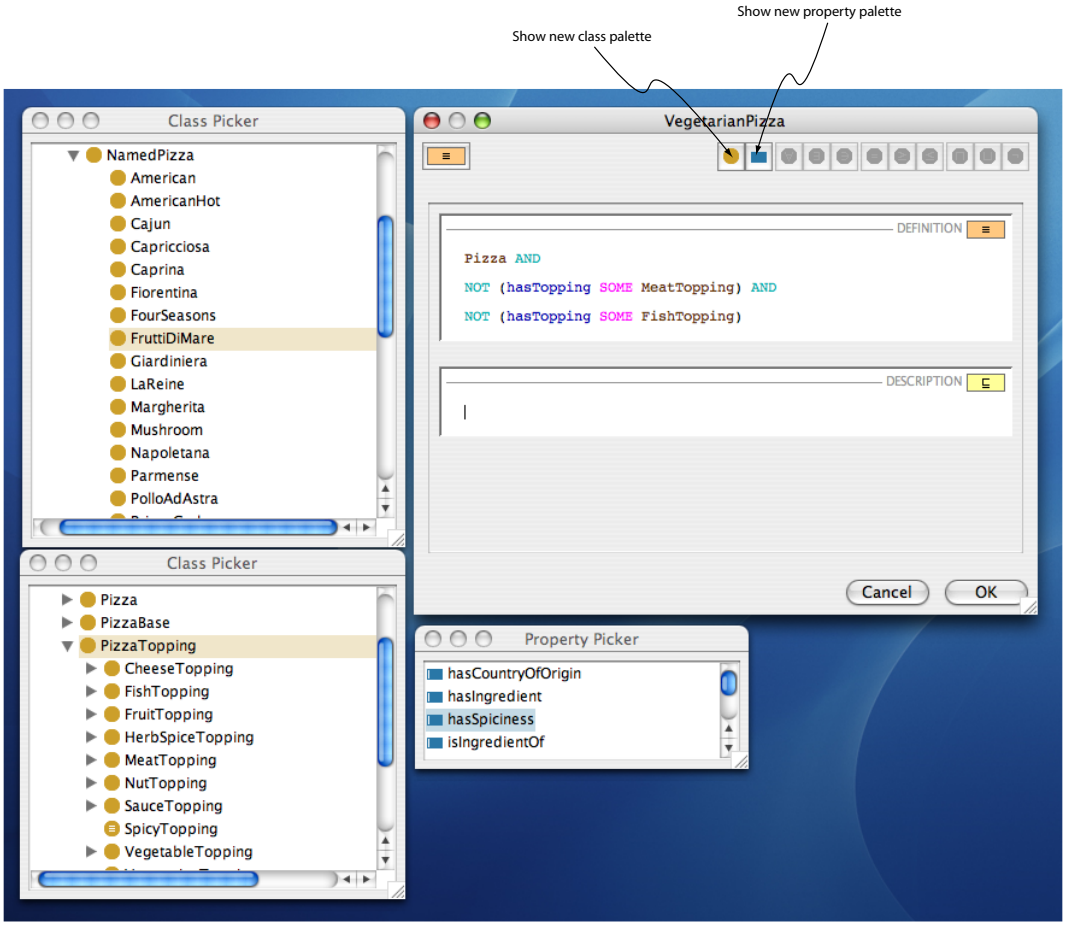

Figure 4: An Example of Class and Property Selection Palettes

4.1 Correspondence With OWL and Protégé-OWL Syntax

This section shows how the Manchester OWL Syntax maps to the the OWL class constructors and corresponds with the syntax used by Protégé-OWL .

4.1.1 Class Constructors

In general the Manchester OWL Syntax replaces all symbols with English language keywords that reflect the meaning of the symbol. Not only does this make class expressions more natural to read, it also makes it easy to paste the plain text representation of the expression into e-mails etc. without incurring the formatting problems that can arise due to the different fonts required to represent the mathematical symbols that are used in the DL syntax. Table 1 shows the OWL class constructors, their representation in the Protégé-OWL syntax, and the corresponding representation in the Manchester OWL Syntax.

Table 1: OWL Class constructors represented in Protégé-OWL and the Manchester OWL Syntax

OWL Constructor	Protégé-OWL	Example	Manchester OWL Syntax
intersectionOf	C ⊓ D	Human ⊓ Male	Human **AND** Male
unionOf	C ⊔ D	Man ⊔ Woman	Man **OR** Woman
complementOf	¬C	¬Male	**NOT** Male
oneOf	{x y z}	{England Spain Italy}	{England Spain Italy}
someValuesFrom	∃ R C	∃ hasColleague Professor	hasColleague **SOME** Professor
allValuesFrom	∀ R C	∀ hasColleague Professor	hasColleague **ONLY** Professor
minCardinality	R ≥ N	hasColleague ≥ 3	hasColleague **MIN** 3
maxCardinality	R ≤ N	hasColleague ≤ 3	hasColleague **MAX** 3
cardinality	R = N	hasColleague = 3	hasColleague **EXACTLY** 3
hasValue	R ∋ I	hasColleague ∋ Matthew	hasColleague **HAS** Matthew

In line with OWL, the Manchester OWL Syntax supports the nesting of class constructors to arbitrarily complex levels. Complex class expressions should be disambiguated by bracketing. For example, the following expression describes people that have at least one colleague that's a man or a woman who has at least one colleague that is a lecturer. The expression has been formatted using indentation to aid readability.

```
Person AND
hasColleague SOME ((Man OR Woman) AND
                   hasAcademicRole SOME LecturerRole)
```

4.2 Abbreviated Constructs

Besides the standard class constructors shown in Table 1, the Manchester OWL Syntax supports three convenient "intermediate representations". All three representations provide a more compact higher level syntax for representing sets of universal and existential restrictions. These representations are detailed below.

4.2.1 Existential Restrictions

Sets of existential restrictions that act along a common property may be bundled together using by listing the restrictions fillers separated by commas and surrounded by square brackets. For example, consider the following restrictions, where `propP` is an object property, and `A`, `B` and `C` are classes (these may be complex class descriptions).

```
propP SOME A
propP SOME B
propP SOME C
```

These three restrictions may be written in the following way.

```
propP SOME [A, B, C]
```

Figure 5 provides an example of this representation using the description of an AmericanHot pizza from the pizza ontology. It is clear to see that the intermediate representation is more compact and perhaps more readable than the full expanded representation.

4.2.2 Closed Existential Restrictions

In addition to the keywords shown in Table 1 the Manchester OWL Syntax supports another keyword **ONLYSOME**. In a similar fashion to the existential restriction intermediate representation, the 'ONLYSOME' keyword is used with a list of classes surrounded by square brackets. The representation is expanded to produce a set of existential restrictions–one for each class listed within the square brackets–and a universal 'closure' restriction, which has a filler that is the union of the classes listed with the square brackets. For example, consider the following existential restrictions that act along the object property `propP`, which are 'closed' by the universal restriction that has a filler which is the union of the fillers of the separate existential restrictions. The classes denoted by `A`, `B` and `C` could be complex class descriptions.

```
propP SOME A
propP SOME B
propP SOME C
propP ONLY (A OR B OR C)
```

This may be rewritten as

```
propP ONLYSOME [A, B, C]
```

Figure 6 shows an example of the 'ONLYSOME' keyword and representation–it is clear to see how much more compact this representation can be when compared to the fully expanded version.

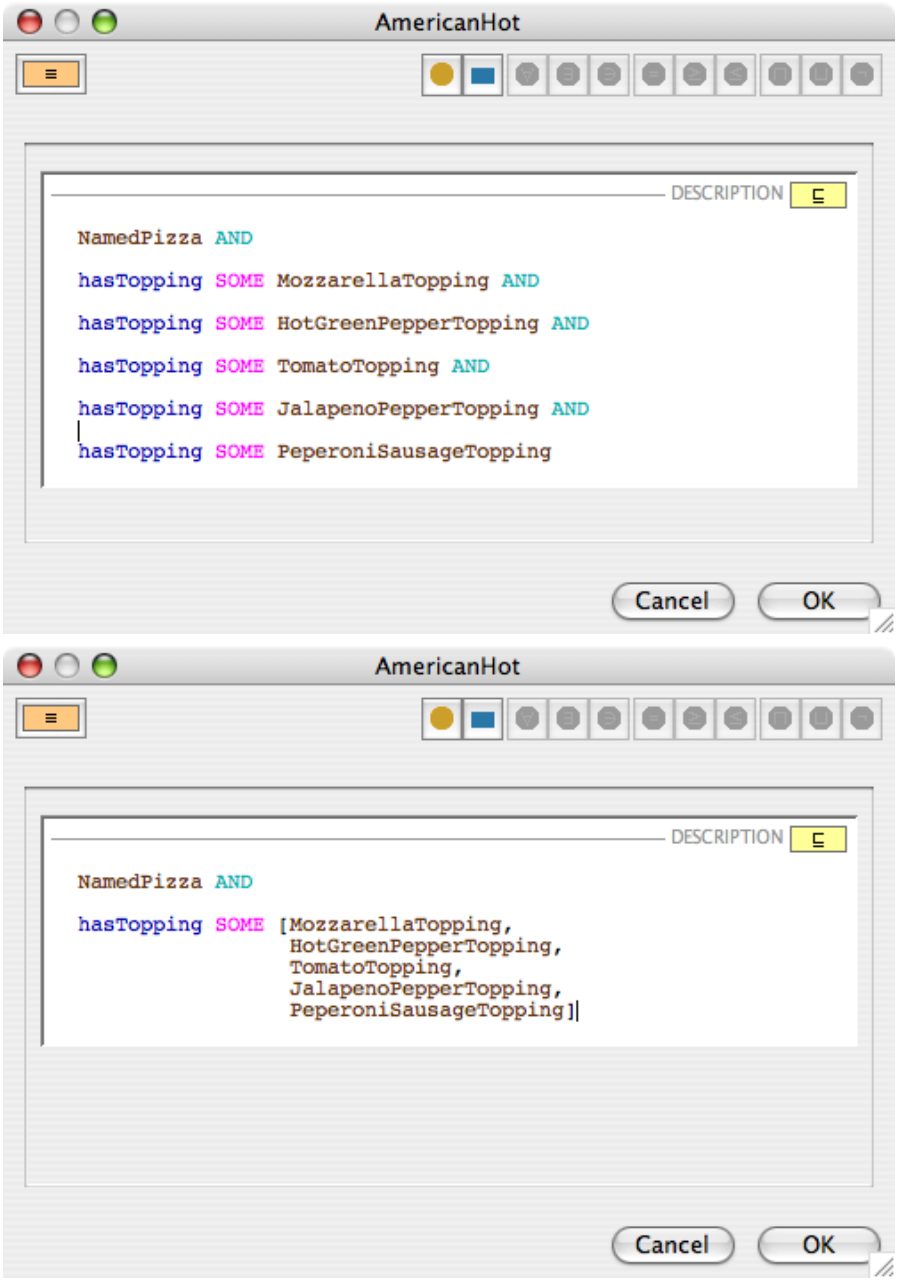

Figure 5: An Example of the Existential Restriction Intermediate Representation — The two representations are semantically equivalent. In the top representation, the five existential restrictions are written out in full. In the bottom representation, the more compact intermediate representation is used.

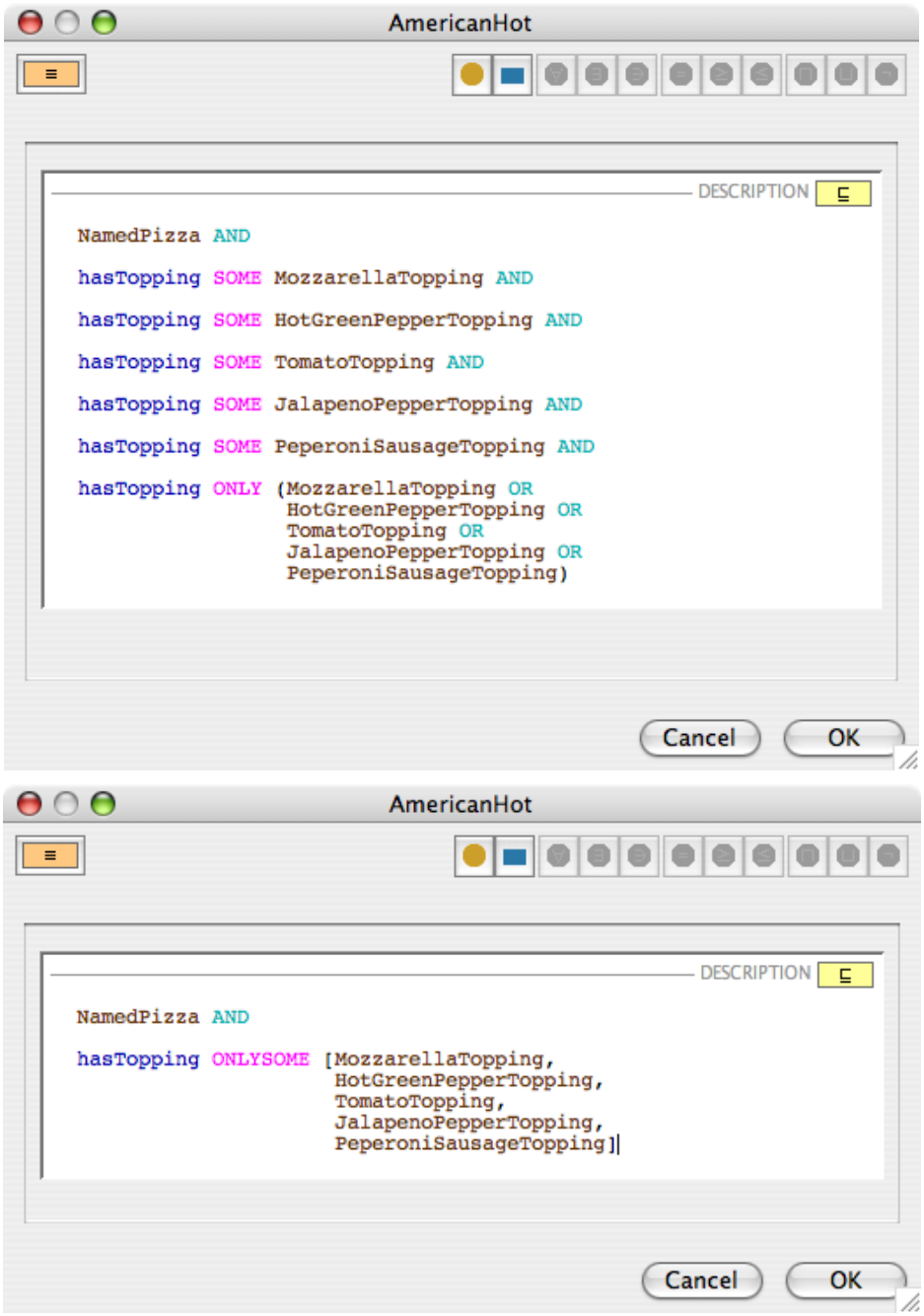

Figure 6: An Example of the usage of the ONLYSOME keyword — The two representations are semantically equivalent. In the top representation, the five existential restrictions are written out in full with a corresponding closure restriction. In the bottom representation, the more compact intermediate representation is used.

5 Editor Features

5.1 Error Highlighting and Reporting

The editor constantly checks for and highlights errors whilst class expressions are being edited. Figure 7 shows and example of how errors are flagged within the editor–the class name PepperoniSausage was mistyped as `Peperonusage`. The first indication of an error is that the location of the error gets underlined in red. Secondly, if the mouse cursor is hovered over the error, a popup tooltip will appear describing the error and suggestions of what was expected. It should be noted that it is impossible to accept the changes close the editor if errors are present–in this situation a dialog will display the error message and the editor will not close.

5.2 Syntax Highlighting

The Manchester OWL Syntax Editor provides syntax highlighting to aide readability of class descriptions both from a syntactic and semantic point of view. For example cyan is used as the colour for the boolean class constructors AND, OR and NOT, whilst magenta is used for restriction keywords such as SOME, ONLY and HAS. Furthermore, class names are coloured brown, object property names are blue, datatype property names are green and individual names are pink. Both typed and untyped literals are shown in light orange.

5.3 Auto Completion

In line with modern IDEs and the Protégé-OWL expression editor, the Manchester OWL Syntax Editor provides auto-completion for class, property and individual names, datatypes and also keywords. To use auto-completion either press the TAB key or the combination of CTRL+SPACE. If several completion options are available when the auto-complete keys are pressed, a popup window containing a list of these options will be displayed. By design, auto-completion will only work if there are no errors prior to the location of auto-completion–if auto-completion isn't working as expected the preceding text should therefore be checked for syntactic errors.

5.4 Auto Suggestion

The Manchester OWL Syntax Editor supports the creation of classes, properties and individuals whilst editing a class expression through its 'auto-suggestion' mechanism. If a potential identifier is typed in a class expression but the identifier cannot be matched to a class, property or individual name in the ontology, the auto-suggestion click area will appear in the left-hand side margin of the editor as shown in the top picture of Figure 8[4]. This area may be clicked to display the suggestions similar to the those depicted in the bottom picture of Figure 8—the editor determines the possible types of resources that could be used where the identifier name is, and compiles the auto-suggestion list based on this information.

In the example shown in Figure 8 the word `hasPrice` has been typed into the editor. This is initially flagged as an error because a class or property name was expected but the ontology does not contain any classes or properties called `hasPrice`. The auto-suggestion mechanism therefore determines that this

[4]Note that the name will also be flagged as an error

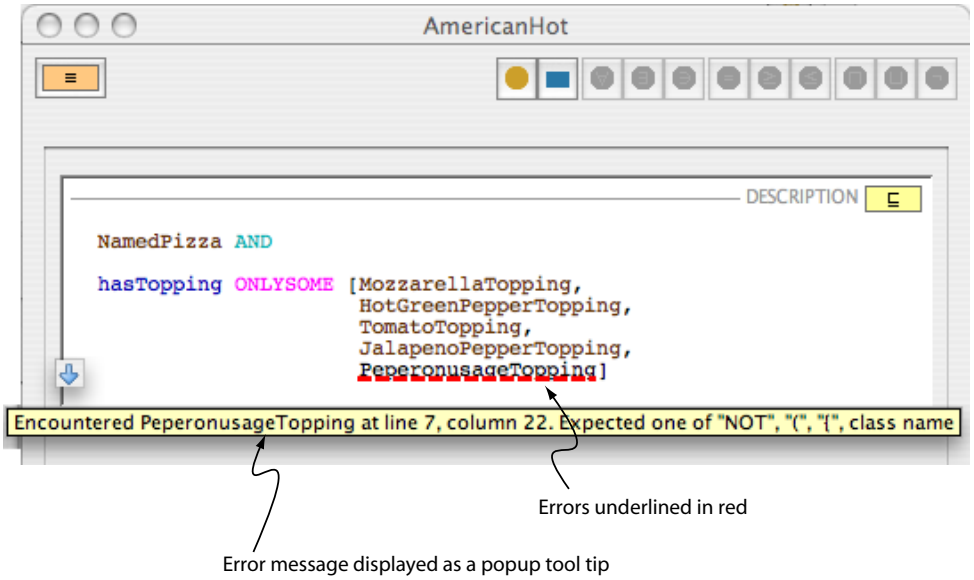

Figure 7: An Example of Error Highlighting

Figure 8: An Example of Auto-Suggestion

problem could be rectified by creating a class or property[5] called `hasPrice`–when the auto-suggestion area is clicked, the list of suggestions shown in the lower picture in Figure 8 is displayed. A suggestion is accepted and executed by double clicking it.

5.5 Useful Shortcut Keys

Acction	Windows/Linux Keys	Mac Keys
Invoke editor	CTRL+E	CMD+*E*
Drop changes and close editor	ALT+ESC	CMD+ESC
Accept changes and close editor	ALT+ENTER	CMD+ENTER
Auto-complete	CTRL+SPACE or TAB	CTRL+SPACE or TAB
Insert AND+newline	CTRL+ENTER	CTRL+ENTER
Duplicate current line	CTRL+D	CTRL+D

6 Example Exercises

This section contains some short examples of using the Manchester OWL Syntax Editor. The examples are based on a skeleton version of the CO-ODE pizza ontology, which can be downloaded from `http://www.co-ode.org/downloads/manchesterowlsyntaxeditor/example/EditorExample.owl`. After building new OWL project from the downloaded owl file and switching the the OWL Classes Tab, the class hierarchy should appear similar to the picture shown in Figure 9.

Exercise 1: Open the editor to edit MargheritaPizza

1. Select MargheritaPizza in the class hierarchy.
2. Invoke the editor by pressing CTRL+E on windows or linux, or CMD+E on the Mac. The editor will consist of one text box containing the *description* of MargheritaPizza as shown in Figure 10 Step 1.

The current description of MargheritaPizza is simply NamedPizza. This is because MargheritaPizza is

[5] Either an object property or datatype property

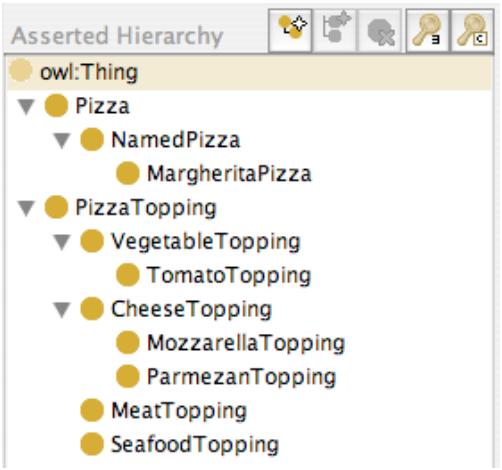

Figure 9: The Skeleton Class Hierarchy

a subclass of NamedPizza–in other words, everything that is a MargheritaPizza is also a NamedPizza.

Exercise 2: Add a restriction along the hasTopping **property to specify that** MargheritaPizza**s have some** MozzarellaTopping

1. Place the text caret/cursor after NamedPizza.
2. Press CTRL+ENTER – this will insert the intersection keyword (AND) followed by a new line. The editor window should not appear similar to the picture shown in Figure 10 Step 2.
3. Type `hasTopping` into the editor, as we want the restriction to be along the hasTopping property.
4. At this point the editor will have underlined `hasTopping` in red and will indicate an error. If the mouse cursor is hovered over `hasTopping` an error message will pop up. The reason for the error is that the ontology does not currently contain a hasTopping property. In order to create a property without leaving the editor, the auto-suggestion feature can be used. Click on the auto-suggestion click area that should have appeared at the left hand side of the editor window (see Figure 10 Step 3). The popup shown in Figure 10 Step 4 should be displayed.
5. Double click on the "Create hasTopping object property" suggestion. The popup will close, `hasTopping` will now appear in blue[a], and the error message for hasTopping will have disappeared.
6. Press CTRL+SPACE to auto-complete the `SOME` keyword – a list of possible keyword will pop up, from which "SOME" should be selected. (Alternatively, type `some` followed by a space – the `some` word will be capitalised and highlighted in magenta). The editor window should now appear like the picture in Figure **??** Step 4.
7. To specify a filler of MozzarellaTopping, type `M` and press CTRL+SPACE to auto-complete the word – a list of possible completions will appear as shown in Figure 10 Step 5 – select MozzarellaTopping

[a]The blue syntax highlighting indicates that hasTopping is an object property

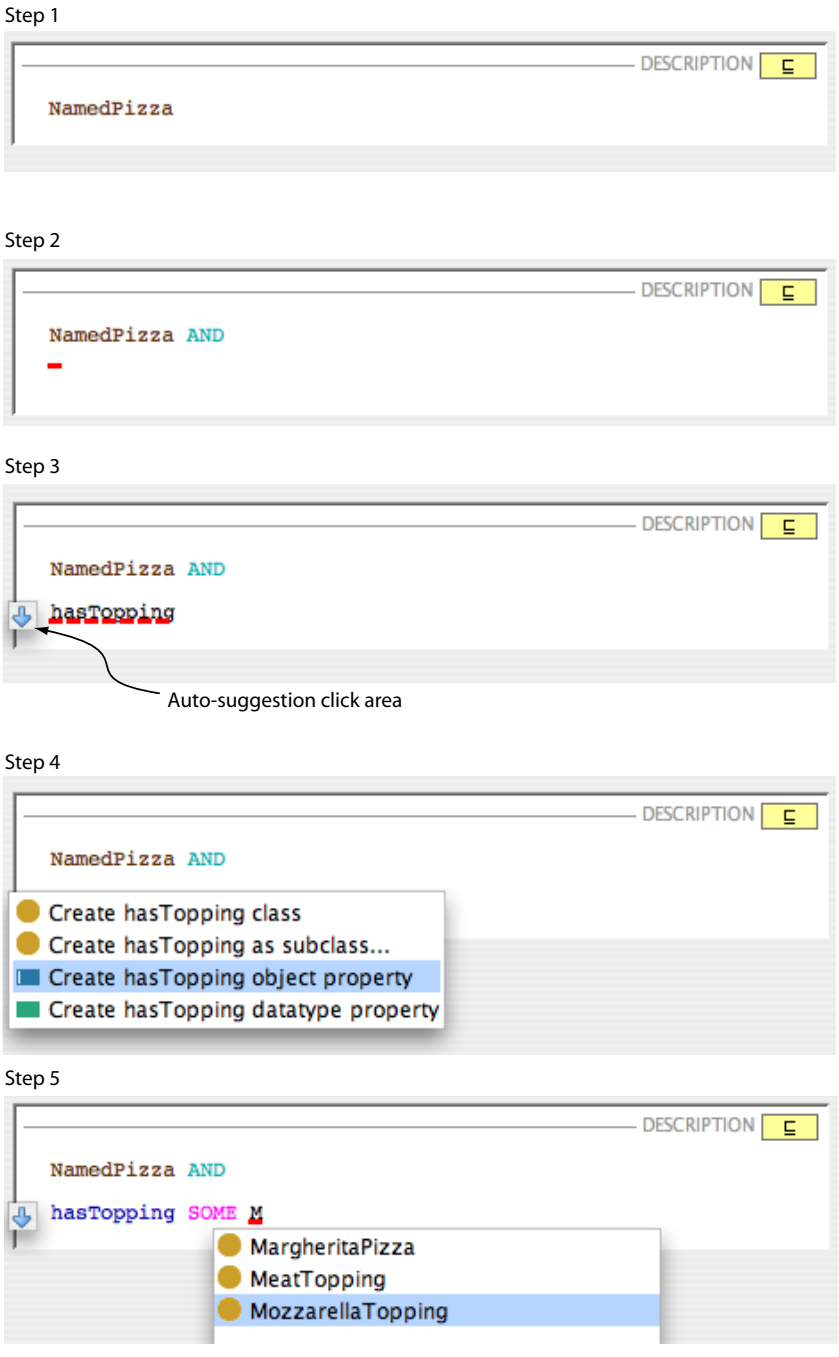

Figure 10: Editing the description of MargeritaPizza

Exercise 3: Add a futher restriction along the hasTopping **property to specify that** MargheritaPizza**s have some** TomatoTopping

1. Ensure the text caret/cursor is placed after MozzarellaTopping.
2. Press CTRL+ENTER to insert the intersection keyword (AND) followed by a new line.
3. Type `hasTopping` into the editor in order to specify that the restricted property should be hasTopping.
4. Type `some` followed by a space – the `some` word will be capitalised and highlighted in magenta.
5. Type another space and press CTRL+SPACE to show possible completions – type the beginning of TomatoTopping until the auto-completion is performed.

The description of MargheritaPizza should now appear as shown in Figure 11. This says that all MargheritaPizzas are also NamedPizzas, *and* have a topping that is some kind of MozzarellaTopping *and* has a topping that is some kind of TomatoTopping.

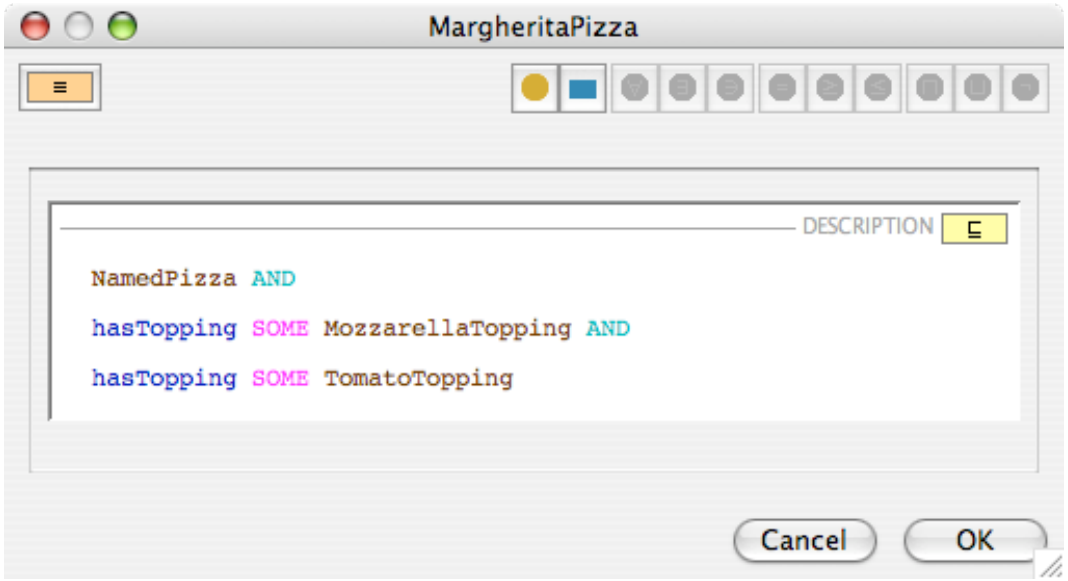

Figure 11: Intitial Description of MargeritaPizza

Exercise 4: Apply the changes and close the editor

1. Either press the 'OK' button to apply the changes and close the editor, or press ALT+ENTER to apply the changes and automatically close the editor.

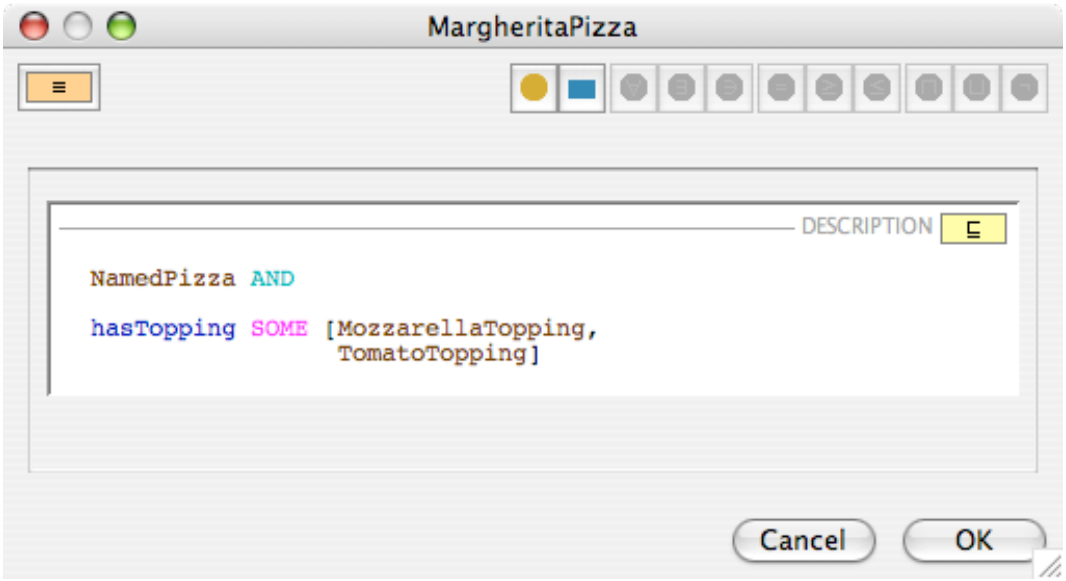

Figure 12: Description of MargeritaPizza

Exercise 5: Open the editor in order to modify the description of MargheritaPizza **by adding a closure axiom**

1. Ensure that MargheritaPizza is selected in the class hierarchy.
2. Press CTRL+E on Windows or Linux, or CMD+E on the Mac to open the editor. The description of MarghertiaPizza should look like the picture shown in Figure 12. Note that the description is shown in a compact form as described in Section 4.2.1.

We want to 'close off' the relationships along the hasTopping property so that they are limited to MozzarellaTopping or TomatoTopping. Recall that in OWL this manifests itself as a universal restriction with a filler that is the union of MozzarellaTopping and TomatoTopping (i.e. MozzarellaTopping ⊔ Tomato-

Topping). To do this in the Manchester OWL Syntax the `SOME` keyword is simply swapped to `ONLYSOME`.

Exercise 6: Change the `SOME` keyword in the description of MargheritaPizza **to `ONLYSOME`**

1. Delete the `SOME` keyword and replace it with `ONLYSOME`
2. Click the 'OK' button to apply the changes and close the editor.

View the description of MargheritaPizza in the Protégé-OWL conditions widget – they should look like the picture shown in Figure 13.

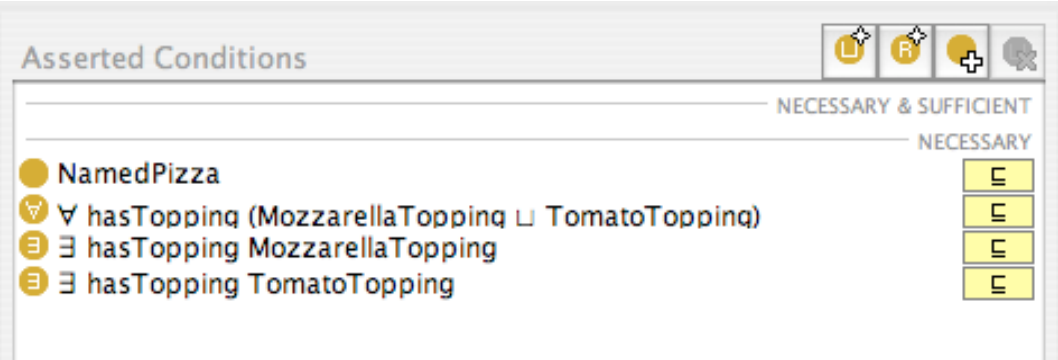

Figure 13: Conditions Widget View of MargeritaPizza

Exercise 7: Create a description for AmericanPizza

1. Create a subclass of NamedPizza called AmericanPizza
2. Ensure that AmericanPizza is selected and press CTRL+E on Windows or Linux, or CMD+E on the Mac to open the editor.
3. Press CTRL+ENTER to automatically enter the AND keyword followed by a new line.
4. Type `hasTopping ONLYSOME [MozzarellaTopping,` (note the comma after MozzarellaTopping) and then hit the ENTER key.
5. Type `TomatoTopping,` and hit the ENTER key
6. Type `PepperoniTopping]`

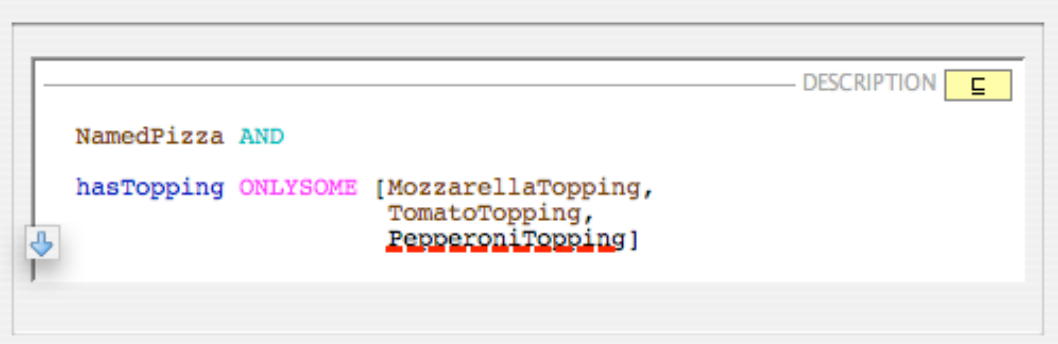

Figure 14: Description of AmericanPizza

The editor should now look like the picture shown in Figure 14. Note that `PepperoniTopping` is underlined in red. This is due to the fact that there is no class called PepperoniTopping in the ontology. Fortunately, the auto-suggestion feature can be used to create this class.

Exercise 8: Use the Auto-suggestion feature to create PepperoniTopping

1. Click the auto-suggestion click area to pop up a list of suggestions – note that the list only contains options to create classes, as it only makes sense to create a class for the filler of a restriction.
2. Double click on the "Create PepperoniTopping as subclass..." option. A class selection dialog will appear.
3. Expand the tree in the class selection dialog and select MeatTopping so that PepperoniTopping will be created as a subclass of MeatTopping. Click OK to close the selection dialog.
4. Click OK to apply the changes and close the editor dialog.

Our description of AmericanPizza now says that all AmericanPizzas are NamedPizzas and have toppings that are only some kinds of MozzarellaTopping, TomatoTopping and PepperoniTopping.

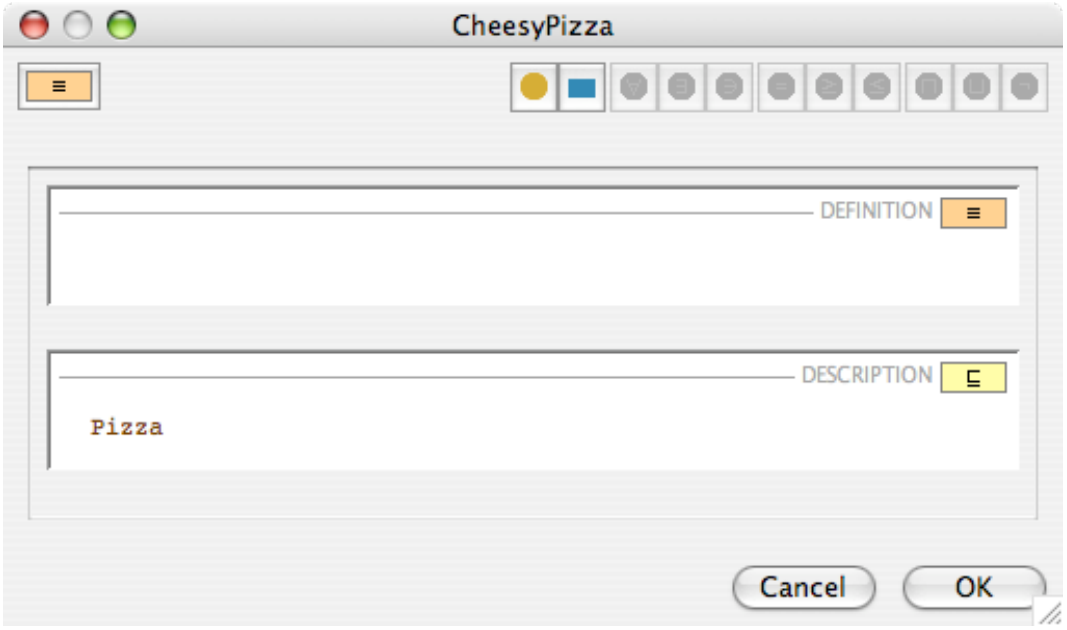

Figure 15: Initial Definition of CheesyPizza

Exercise 9: Create the definition of a CheesyPizza **as a** Pizza **with some** CheeseTopping

1. Create a subclass of Pizza called CheesyPizza.
2. Ensure that CheesyPizza is selected and then open the editor to create a definition for it.
3. Press the 'Add definition' button, which is located in the top left corner of the editor (see Figure 2) to create a new definition – the editor should now look like the picture in Figure 15.
4. Delete `Pizza` from the editor *Description* text box, and type `Pizza AND hasTopping SOME CheeseTopping` into the *Description* text box, so that the editor appears similar to the picture shown in Figure 16.
5. Click the 'OK' button to apply the changes and close the editor.

CheesyPizza is now defined to be any Pizza that has a topping that is some kind of CheeseTopping.

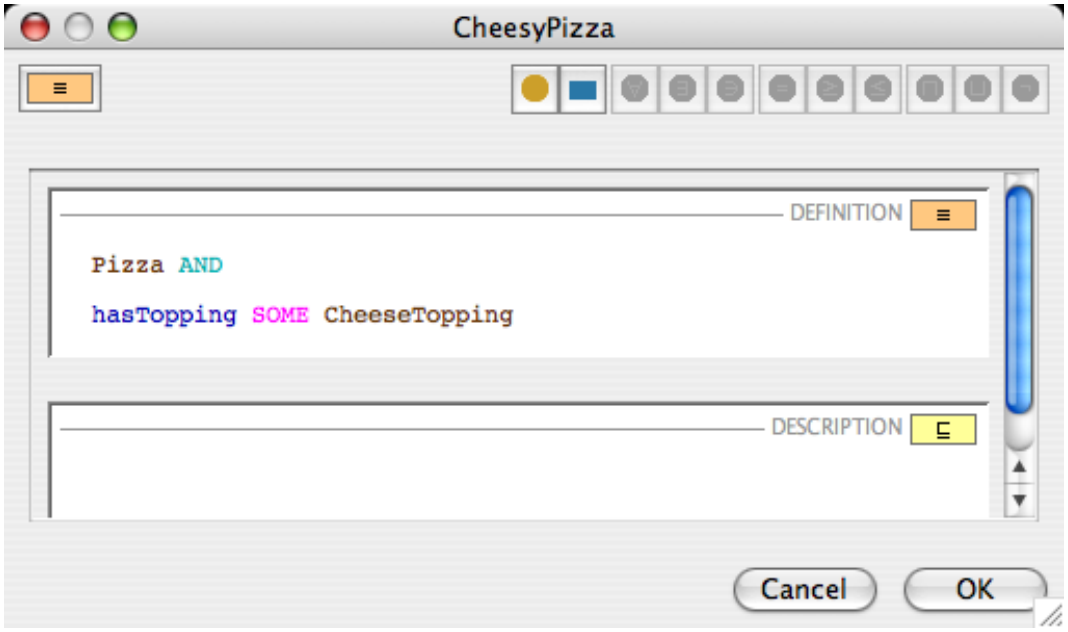

Figure 16: Full Definition of CheesyPizza

Acknowledgements

This work was supported in part by the CO-ODE project funded by the UK Joint Information Services Committee and the HyOntUse Project (GR/S44686) funded by the UK Engineering and Physical Science Research Council and by 21XS067A from the National Cancer Institute.