Ontology101Tutorial Documentation

Release 1.0

Nicole Vasilevsky

Jan 10, 2020

Contents:

CHAPTER 1

Initial Preparation

You will need to clone this repository to download the exercise files in the **BDK14_exercises** folder.

Associated exercise files include:

- basic-classification
- basic-disjoint
- basic-dl-query
- basic-restriction
- basic-subclass
- domain-range
- taxon-union

These exercises were tested under Protégé 5.1. *Note: some screenshots may appear different if you are using a prior version of Protégé.*

1.1 Cloning the GitHub repository

Instructions to clone a repository using the command line are available here.

You can also install a graphical user interface like Sourcetree or GitHub Desktop. Detailed instructions for downloading the repository in Sourcetree are here.

1.2 Dowloading Protégé

To download Protégé, go to: http://protege.stanford.edu/

CHAPTER 2

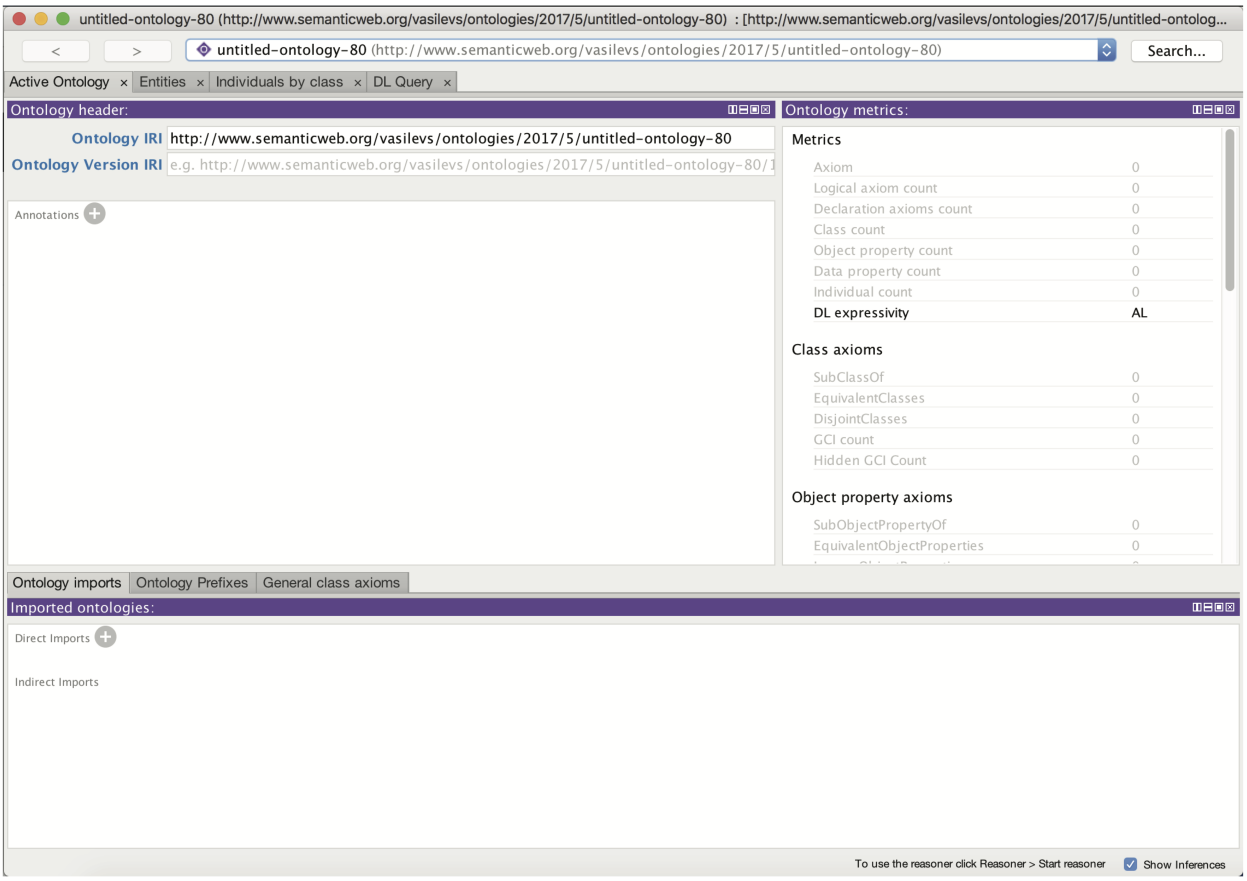
Starting Protégé

When you start Protégé you are presented with a default empty ontology.

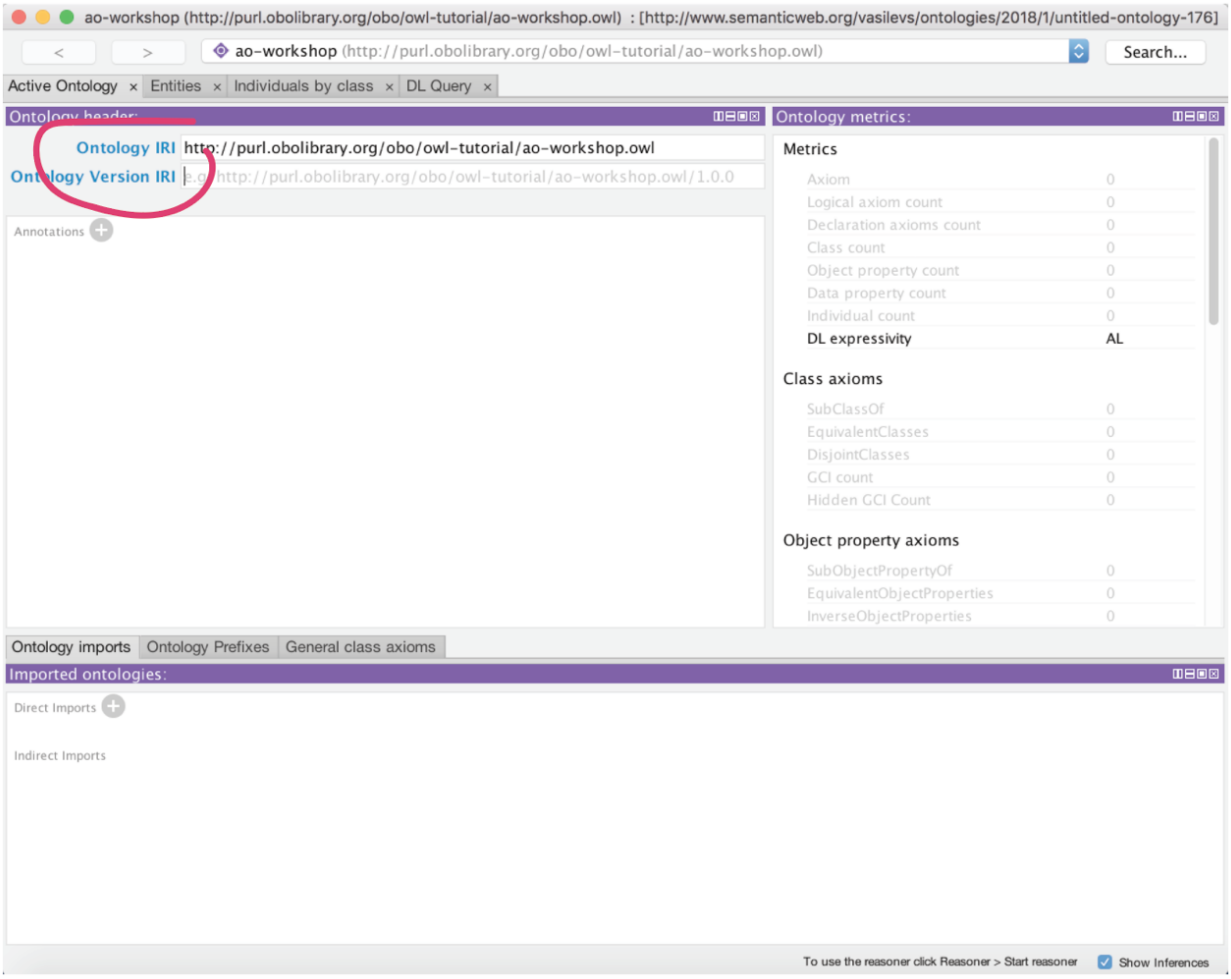
To open Protege on the command line, navigate to your directory where your application is stored. See example below:

```
cd /Applications
cd Protege-5.1.0/
```

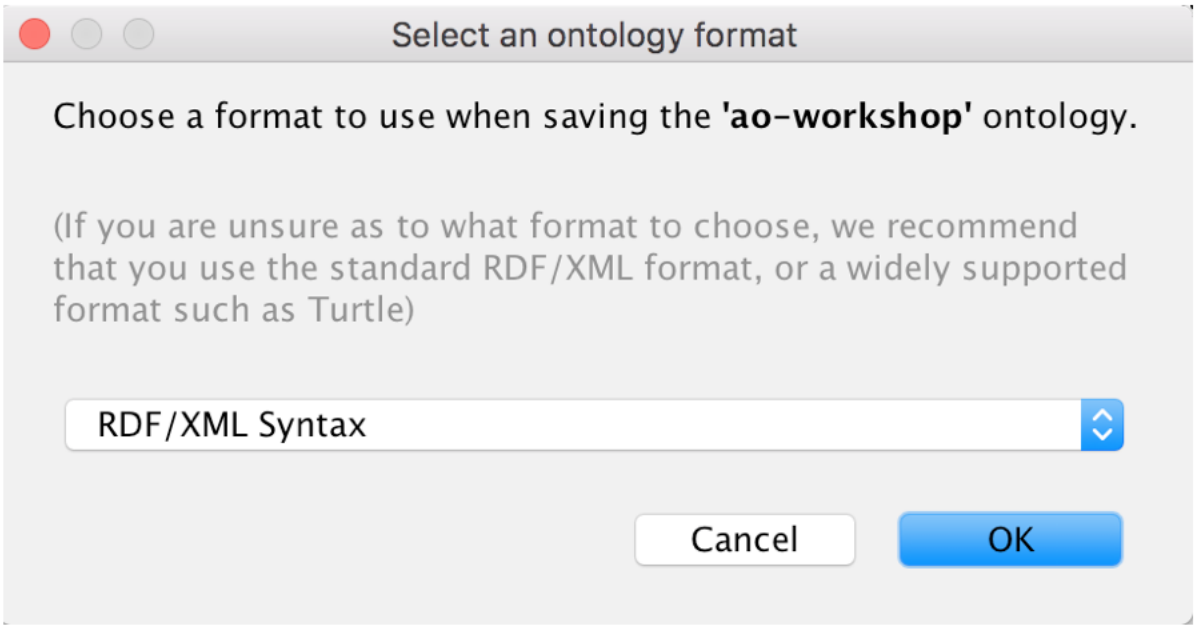
Type the command `open Protege.app` to open the Protégé application.



We will begin by clicking into the Ontology IRI field and providing an IRI. The IRI will be used to identify our ontology on the Web. You can set the IRI to anything you want at this stage, for this tutorial we will use "http://purl.obolibrary.org/obo/owl-tutorial/ao-workshop.owl"



You will also want to save this ontology file to your computer. Under the File menu select Save. Use the next dialog box to specify the format of your ontology file.

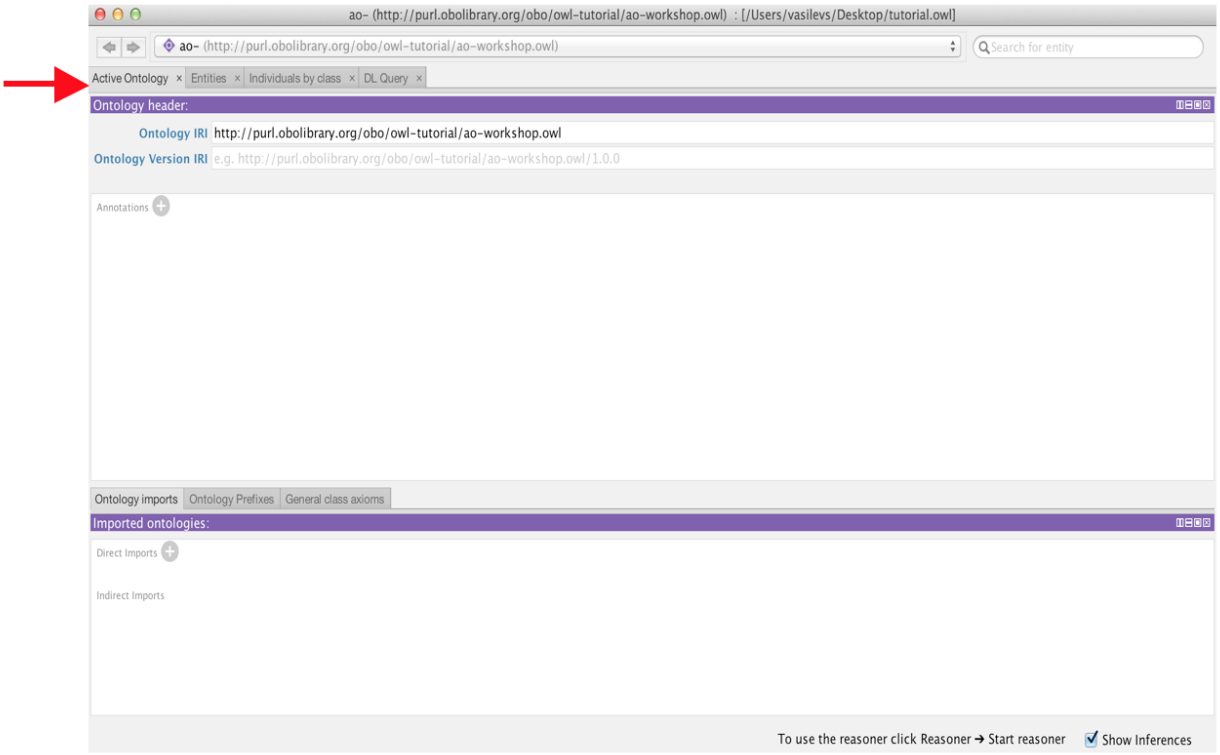


Protégé allows you to save your ontology in a variety of OWL formats, including the OBO 1.2 flat file format. We recommend that you save your ontology in RDF/XML, as this is the most stable format to work with in Protégé. You can always choose to export your file in one of the other formats later. Click OK to continue. Name your ontology, perhaps tutorial.owl. Choose a location on your computer to save your ontology.

2.1 The Protégé UI

The Protégé interface follows a basic paradigm of Tabs and Panels. By default, Protégé launches with the main tabs seen below. The layout of tabs and panels is configurable by the user. The Tab list will have slight differences from version to version, and depending on your configuration. It will also reflect your customizations.

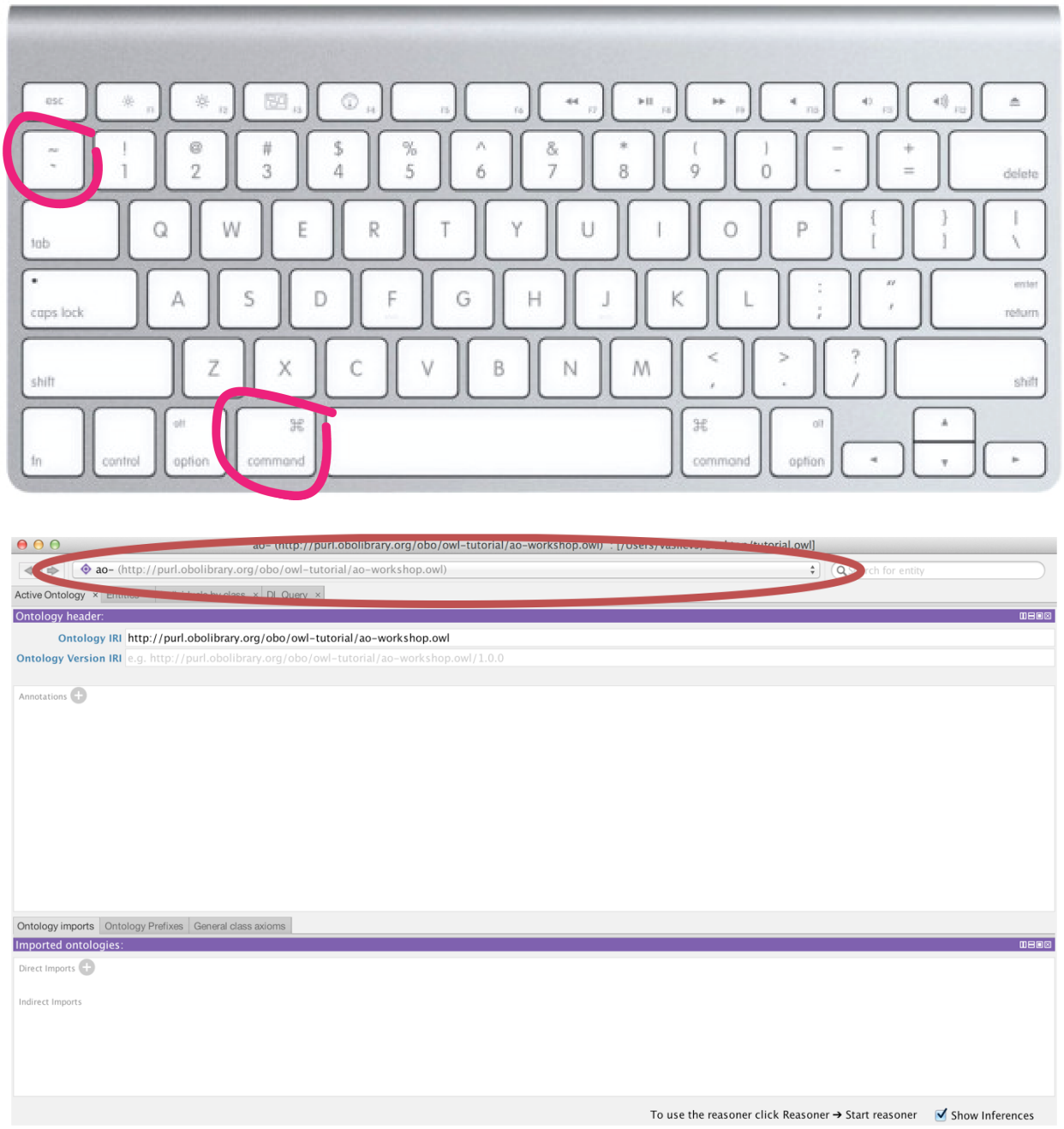
To customize your view, go to the Window tab on the toolbar and select Views. Here you can customize which panels you see in each tab. In the tabs view, you can select which tabs you will see. You will commonly want to see the Entities tab and/or Classes tab and the Object Properties tab.



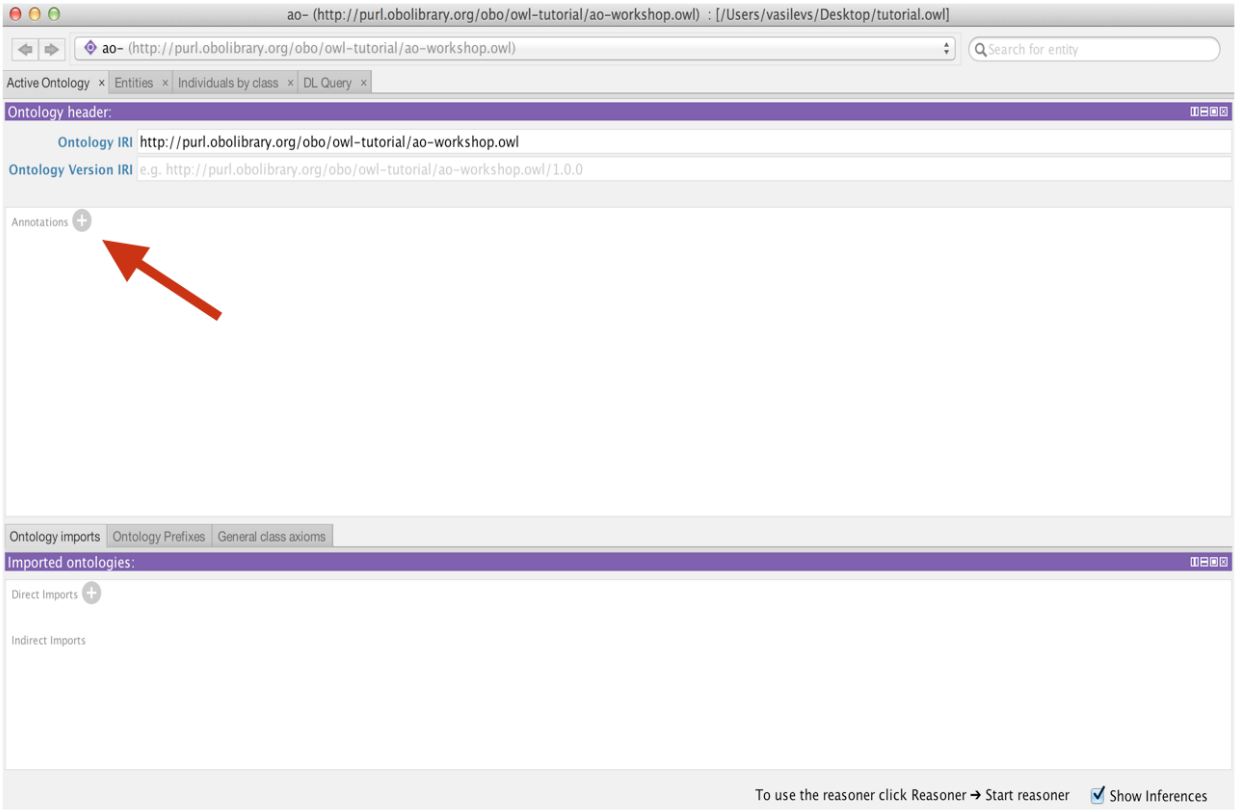
The first tab you see is the Active Ontology tab. Here you will find some basic meta-data about the ontology you are viewing. At the very top you see the IRI and file name of the active ontology you are viewing. Protégé allows you to work with multiple ontologies at once, so *this top bar is very important as it lets you know which ontology you are viewing and editing*.

Note: if you open a new ontology while viewing your current ontology, Protégé will ask you if you'd like to open it in a new window. If you select no, it will open in the current window and you can then switch back and forth to it from the Active Ontology tab.

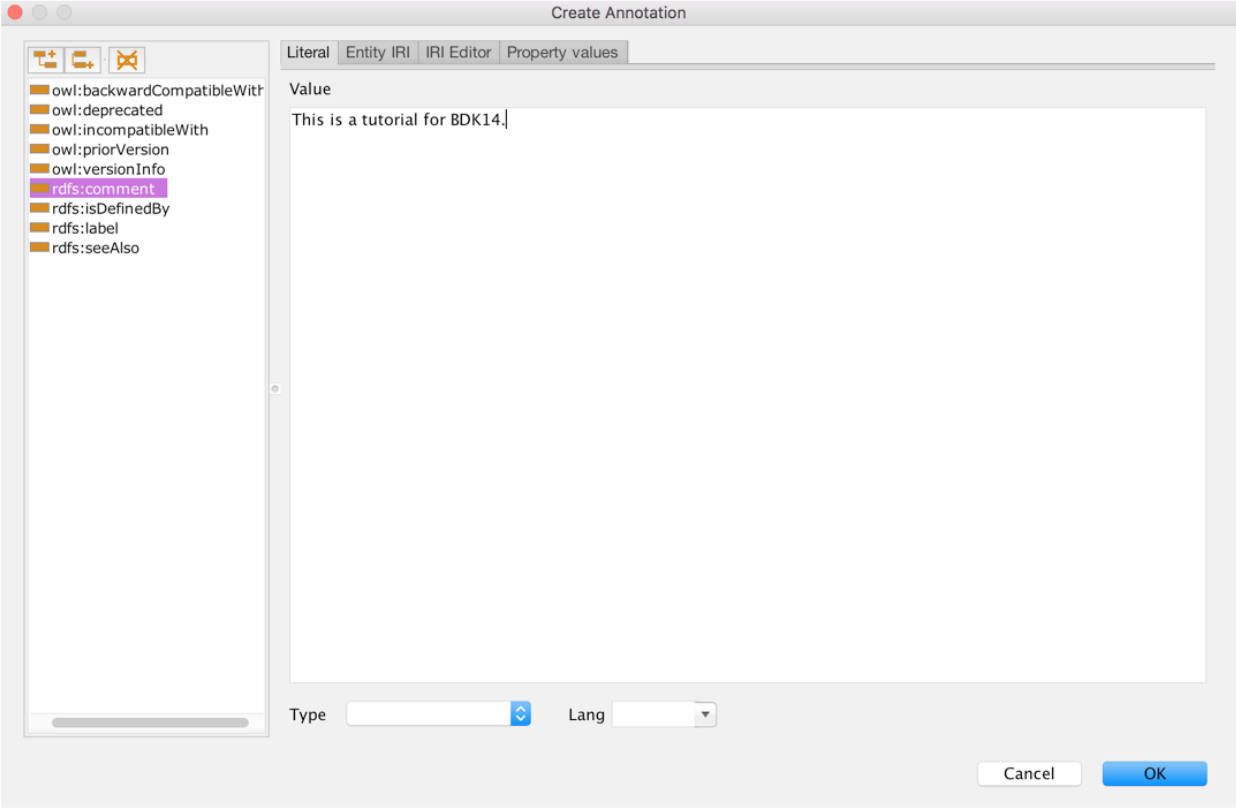
If you say yes, it will open in a new window. If you use a Mac, you can toggle back and forth between each window by using the hot keys Command ~.



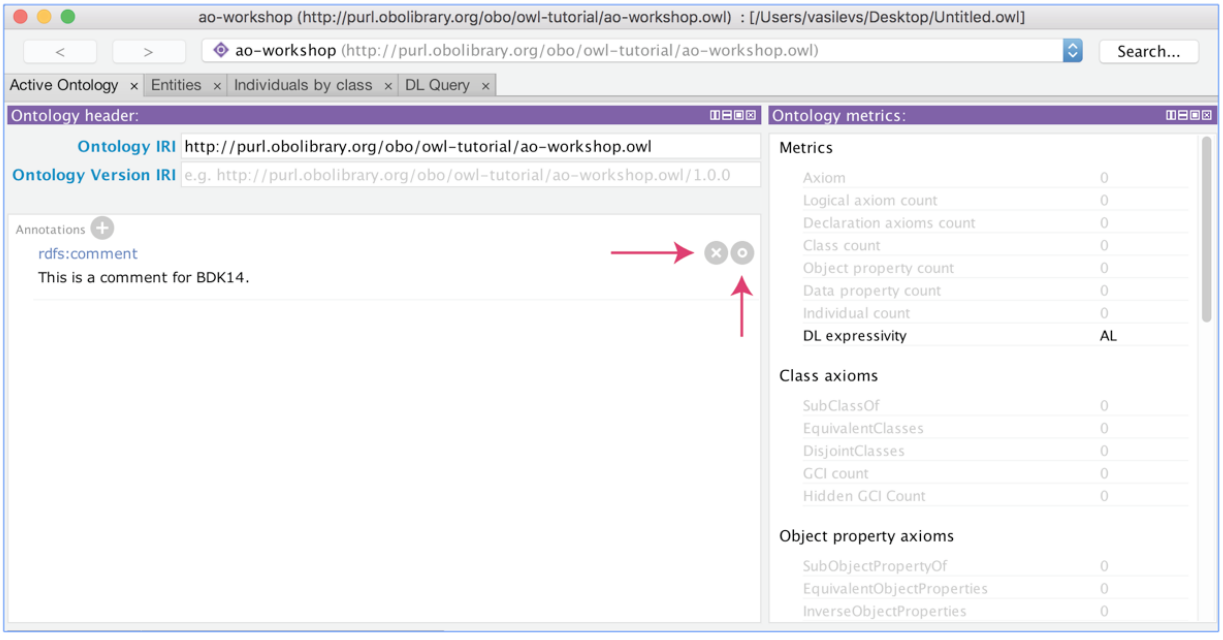
The panel in the center is the ontology annotations panel. You can use this panel to add basic meta-data to your ontology, such as the creation date, the authors and a short description. Using the annotation panel, create a simple comment on the ontology describing what it is about. First select the + button that is labelled Annotations.



A dialog will open, select the comment annotation on the left, and type your text into the text box on the right-hand side. Click OK; to add the annotation.



The comment should appear in the ontology annotations where you have the option to either edit or delete it. Throughout the application, the grey-circle icons have related functionality: a **+** is used to add, **x** to delete, and **o** to edit.

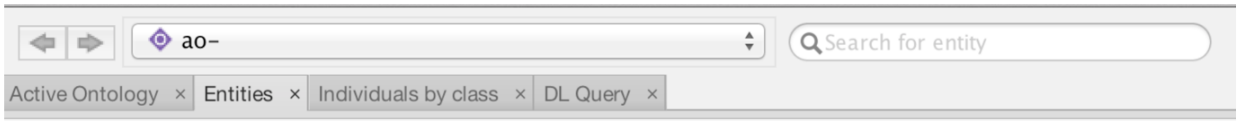


The active ontology tab contains additional information about the ontology that we will explore later. These include a panel for managing ontology imports.

CHAPTER 3

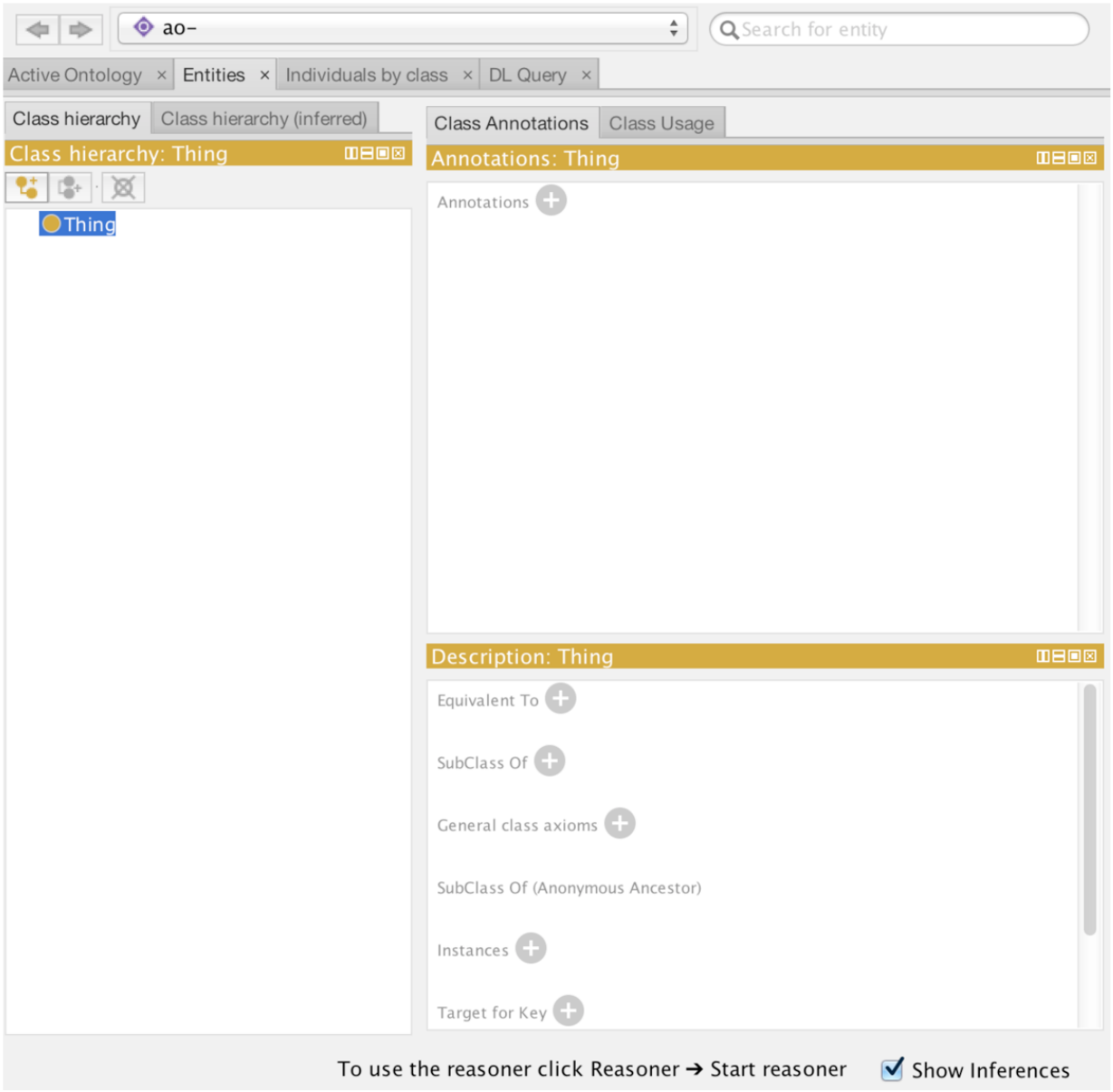
The entities tab

You will see along the top of the screen various tabs. Each tab provides a different perspective on the ontology. An entity is any class, property (object, data, or annotation), or individual. For example, the classes tab allows us to view and edit the classes in the ontology, and similarly the object properties tab focuses on the object properties in the ontology. The *primary tab* where you will spend most of your time is the Entities tab.



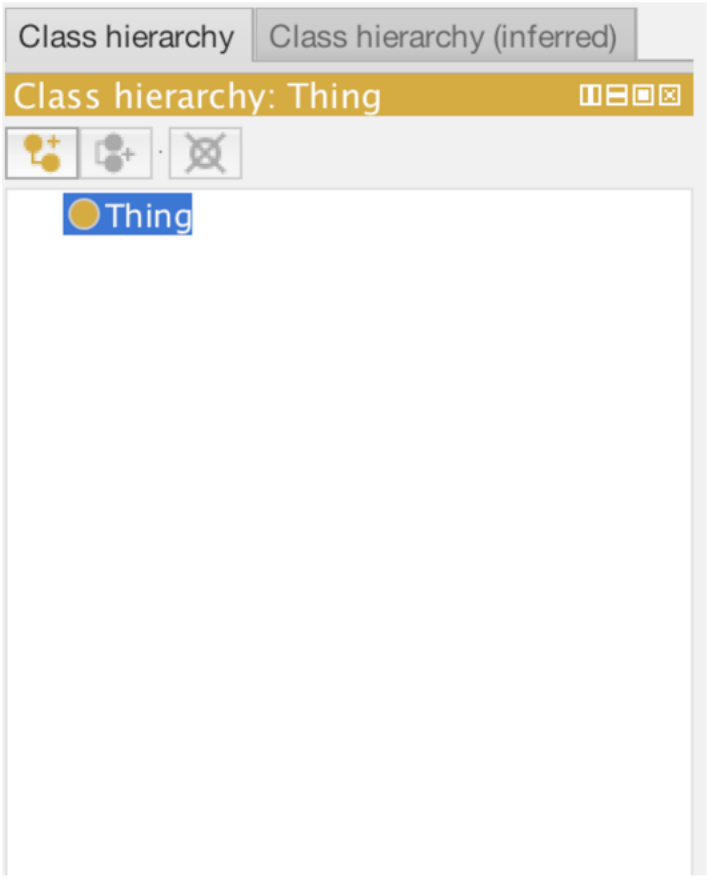
Select the Entities tab and then select the Thing class. Thing is the root class for all OWL ontologies and it cannot be deleted in Protégé.

The Entities tab is split into two halves. The left-hand side provides a suite of panels for selecting various entities in your ontology. When a particular entity is selected the panels on the right-hand side display information about that entity. The entities panel is context specific, so if you have a class selected (like Thing) then the panels on the right are aimed at editing classes. The panels on the right are customizable. Based on prior use you may see new panes or alternate arrangements.

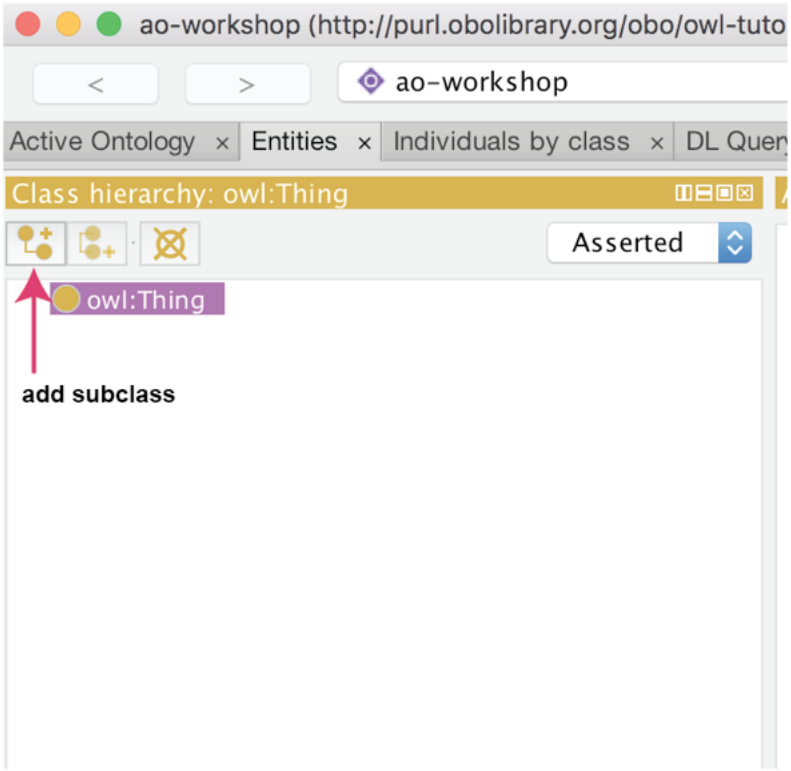


3.1 Creating your first class

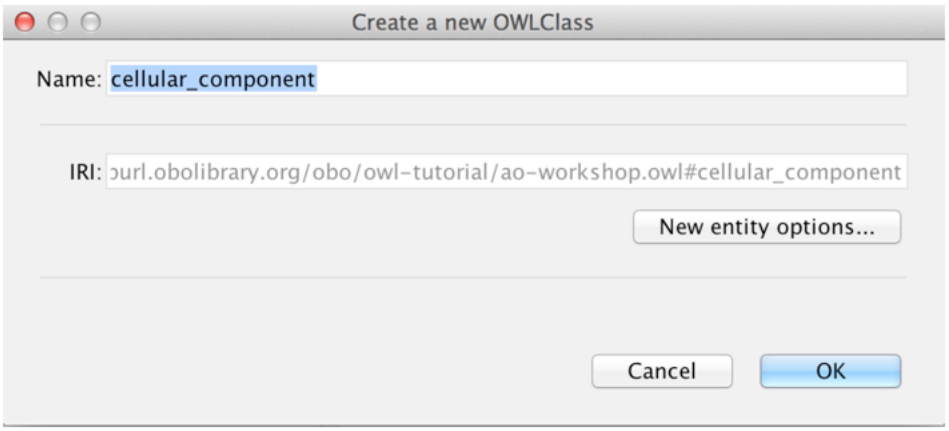
By far the most common panel for working with your ontology is the **Class hierarchy panel.**



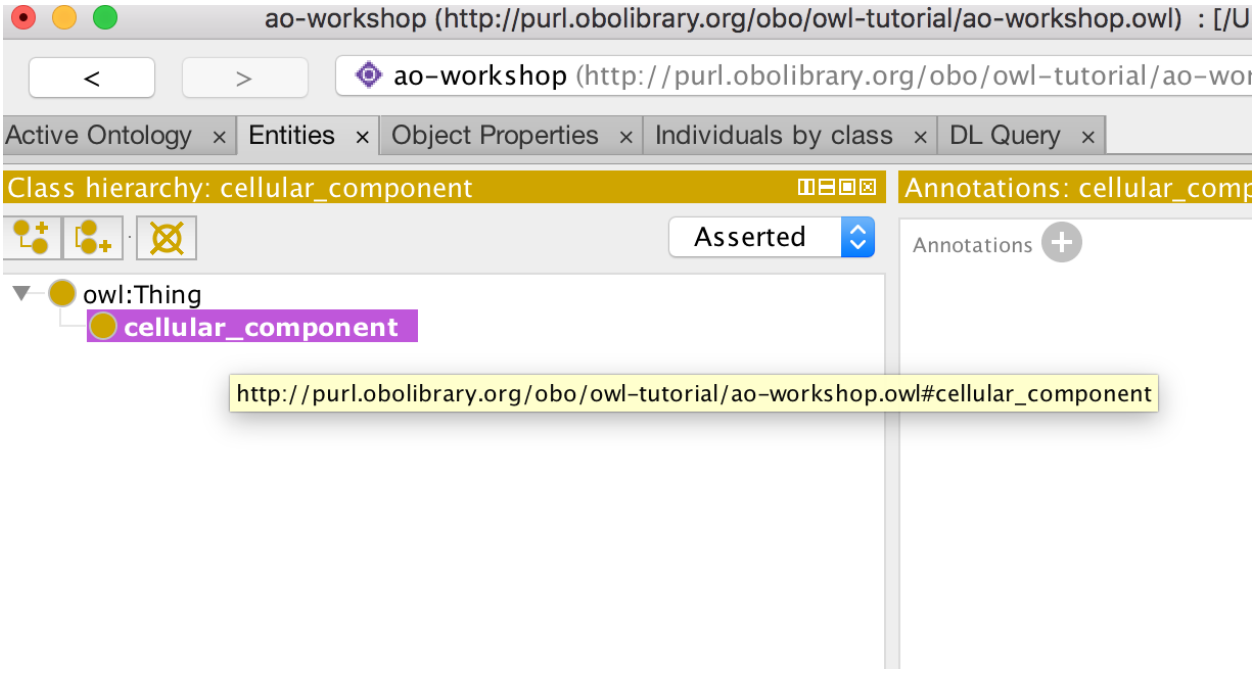
There are three button at the top of the class hierarchy view. These allow you to add a **subclass (L-shaped icon)**, **add a sibling class (c-shaped icon)**, or **delete a selected class (x'd circle)**. Click the add subclass button to add a child class to OWL thing.



A dialog will popup. For now, simply name this class **cellular_component**. Click "OK" to add the class.

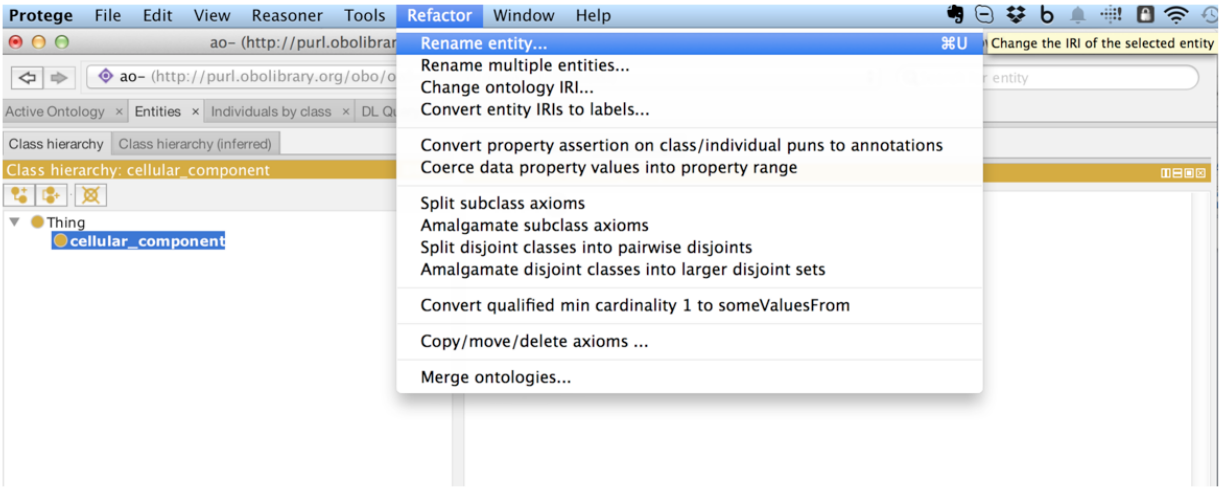


The class should have been created as follows. By default, Protégé will use the ontology IRI, followed by a #, followed by your specified name (replacing spaces with underscores) as the unique IRI for this entity. If you hover over this class with your mouse you will see the full IRI for this class. Important: If you have previously configured an IRI generation scheme you may see your IRI being generated in an alternate format (see below).

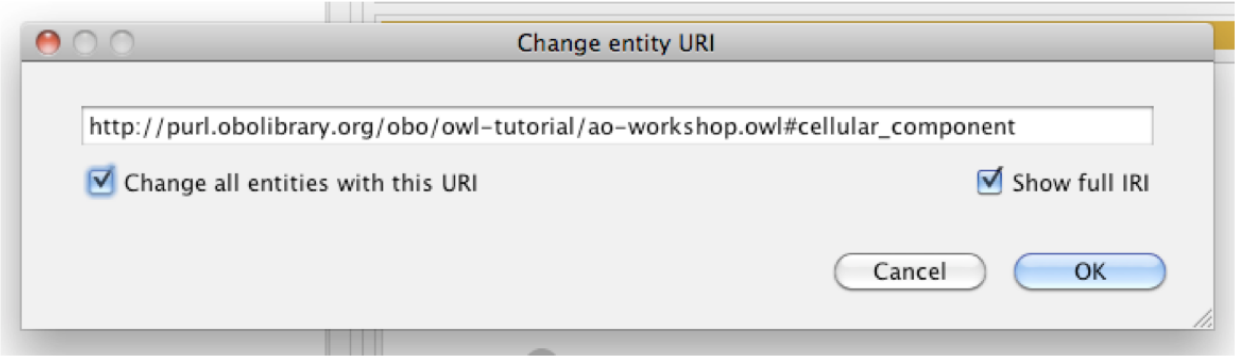


3.2 Renaming an entity

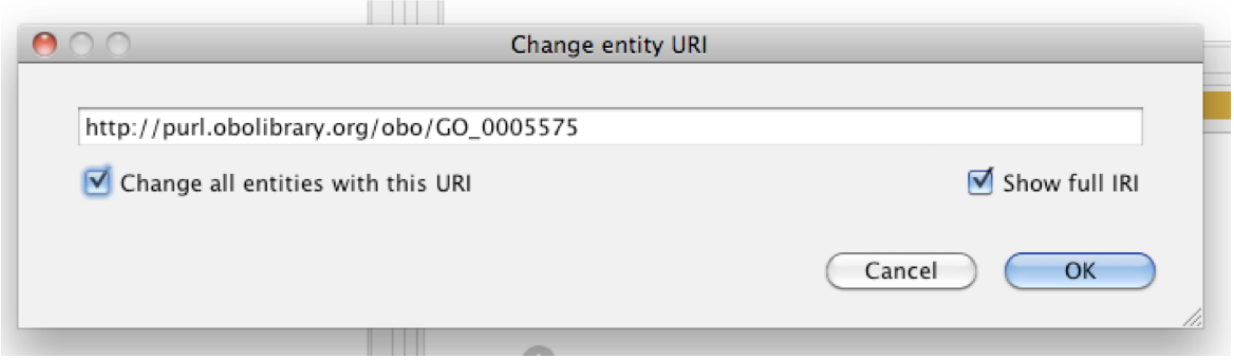
We can change the IRI for a concept using the rename function in the Refactoring menu. Note that this function can also be accessed with a command U keystroke. Rename the **cellular_component** class to use its proper IRI from the Gene Ontology http://purl.obolibrary.org/obo/GO_0005575



Make sure to check the "Show full IRI" box so you can edit the full IRI.



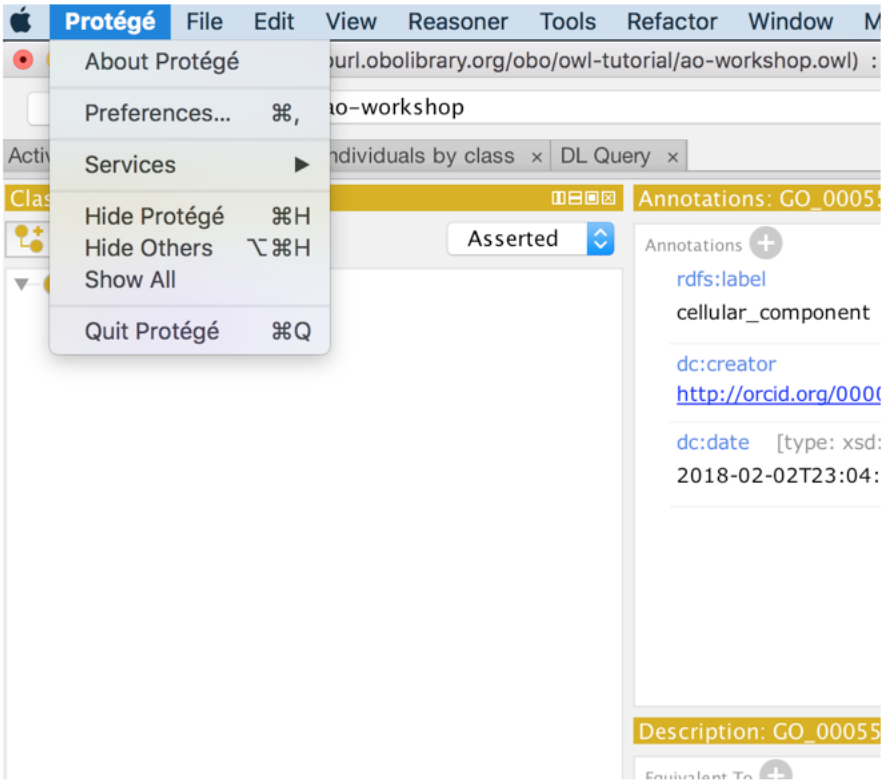
And then paste or type in the correct GO URI (http://purl.obolibrary.org/obo/GO_0005575).



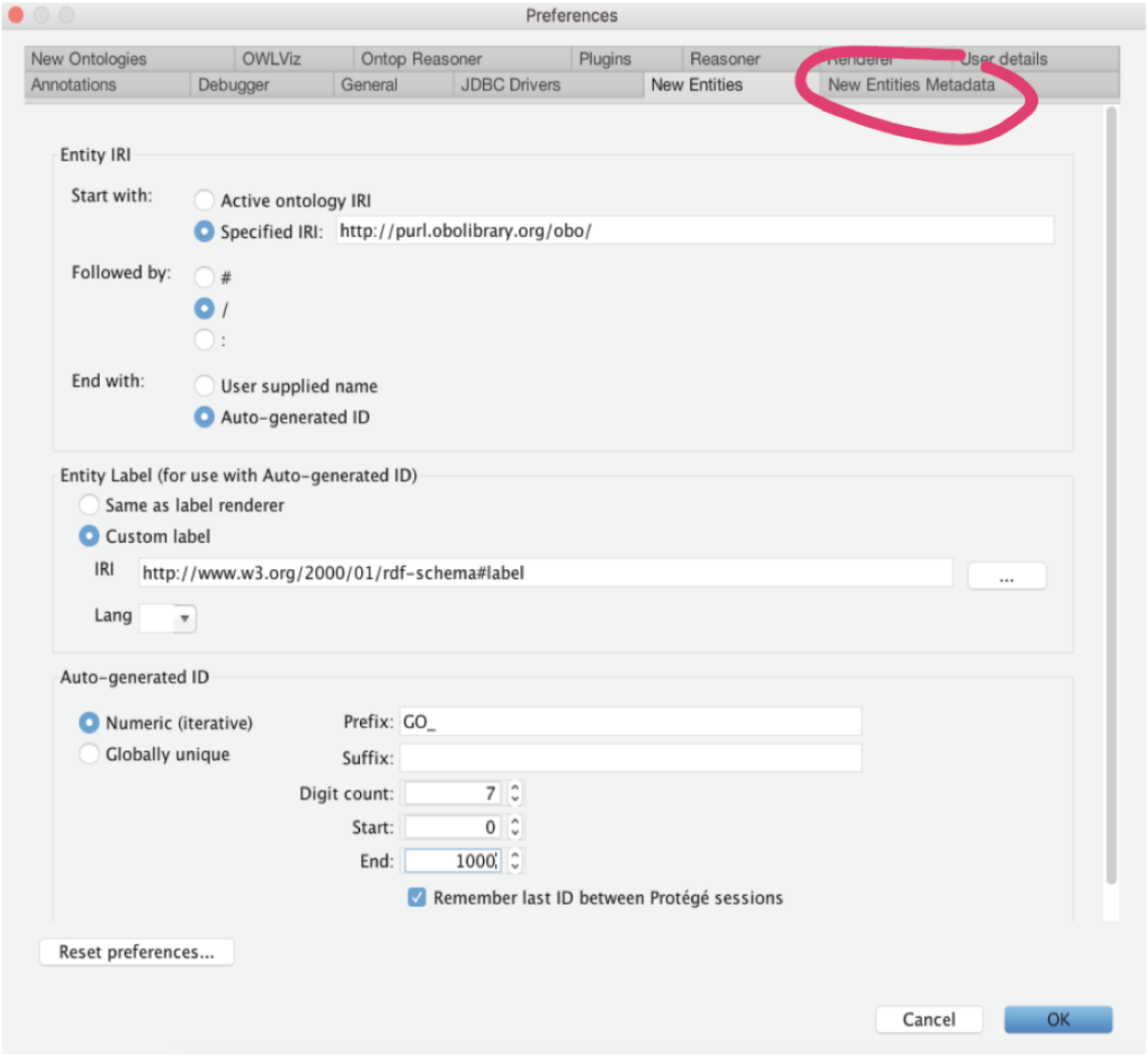
Now the correct GO URI appears in the ontology; in the Class hierarchy panel, the class will appear as "GO_0005575" (to those who have used Protégé before you might see a different label). Luckily you don't have to rename every entity you create when building your own ontology; Protégé provides a "New Entities" preferences panel where you can specify how new IRI should be created, described in the next section.

3.3 New entities

Terms in the ontologies we use have separate names and IDs. The names are annotation values (labels) and the IDs are represented using IRIs. The OBO foundry has a policy on IRI (or ID) generation (http://www.obofoundry.org/principles/fp-003-uris.html). You can set an ID strategy using the "New Entities" tab under the Protégé preferences – on the top tool bar, click the "Protégé dropdown, then click Preferences.



Set your new entity preferences as in the following screenshot, then choose the New Entities tab):

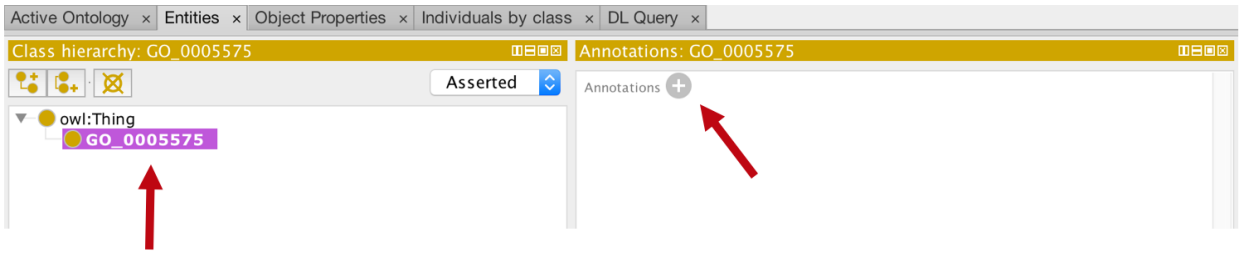


For ontologies other than GO, change the value of the prefix.

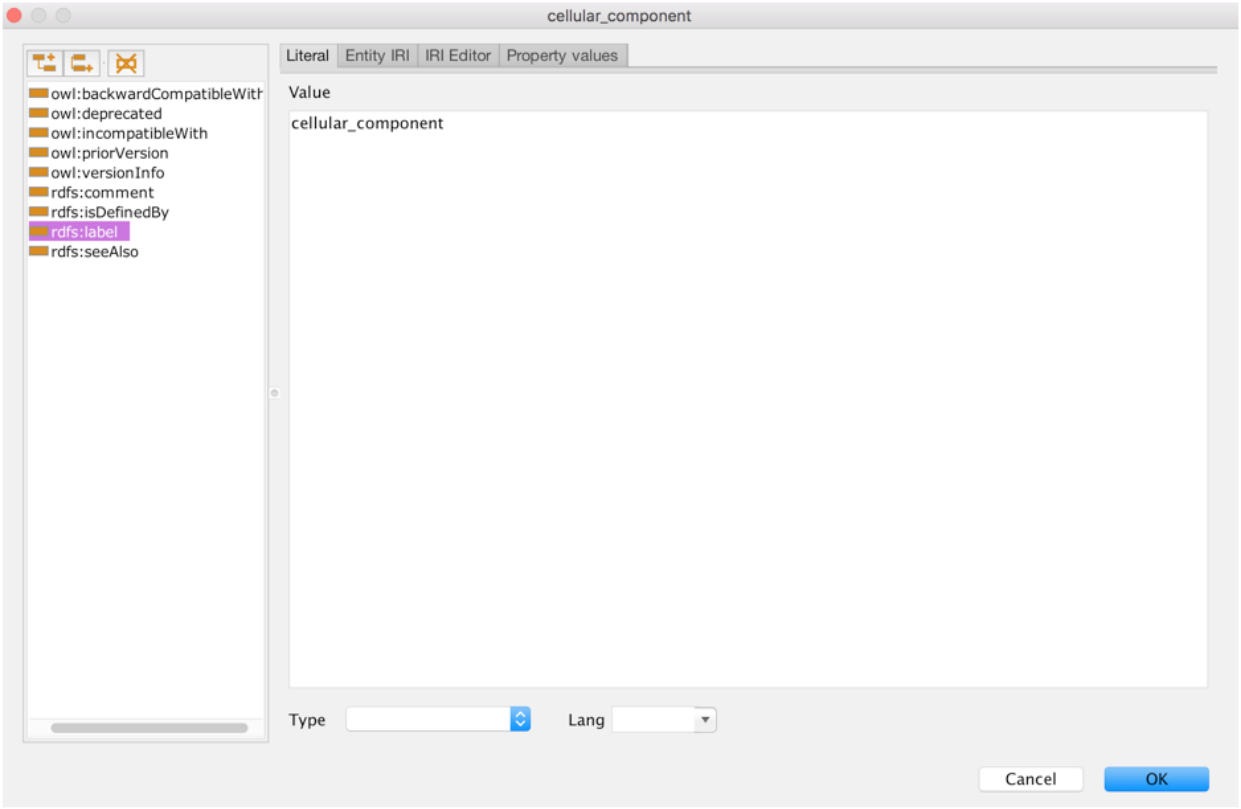
Note that all OBO library ontologies should use the "Specified URI" value: http://purl.obolibrary.org/obo

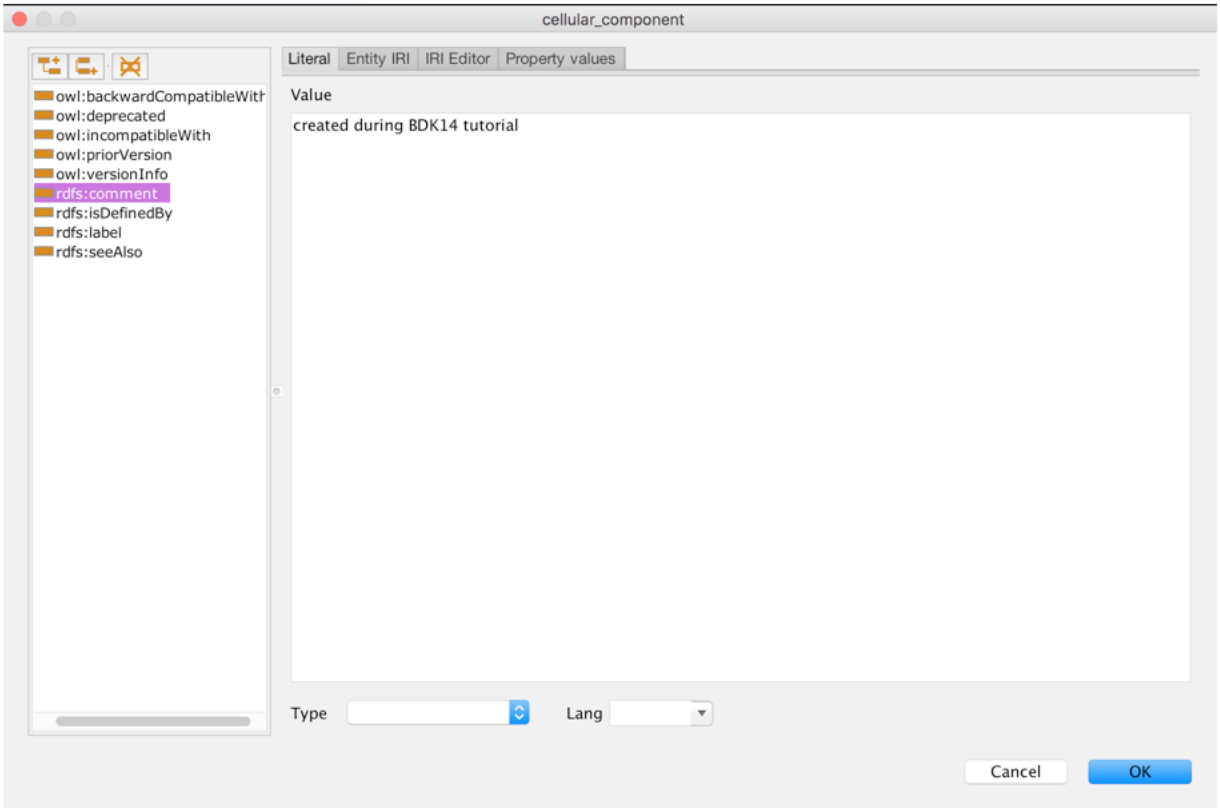
3.4 Adding annotations properties

Using Protégé you can add annotations such as labels, descriptions, cross references (xrefs) to any OWL entity. The panel on the right, named Annotations, is where these annotations are added. Use this panel to add a **cellular_component** label to the class you created previously (notice how when changed the IRI, you also lost your label. This is because the label was previously part of the IRI, and Protégé was rendering the label based on the IRI. We'll fix that in a minute.) Click on the GO labelled class to select it.



Select the + button to add an annotation to the selected entity. Protégé has a set of built in annotation properties, such as label and comment – add rdfs:label "cellular_component" and click OK. You can also add a comment such as "created during BDK14 tutorial", by clicking the + sign again, choosing "rdfs:comment" on the left hand side bar, and typing your comment in the "Literal" box, then click OK.



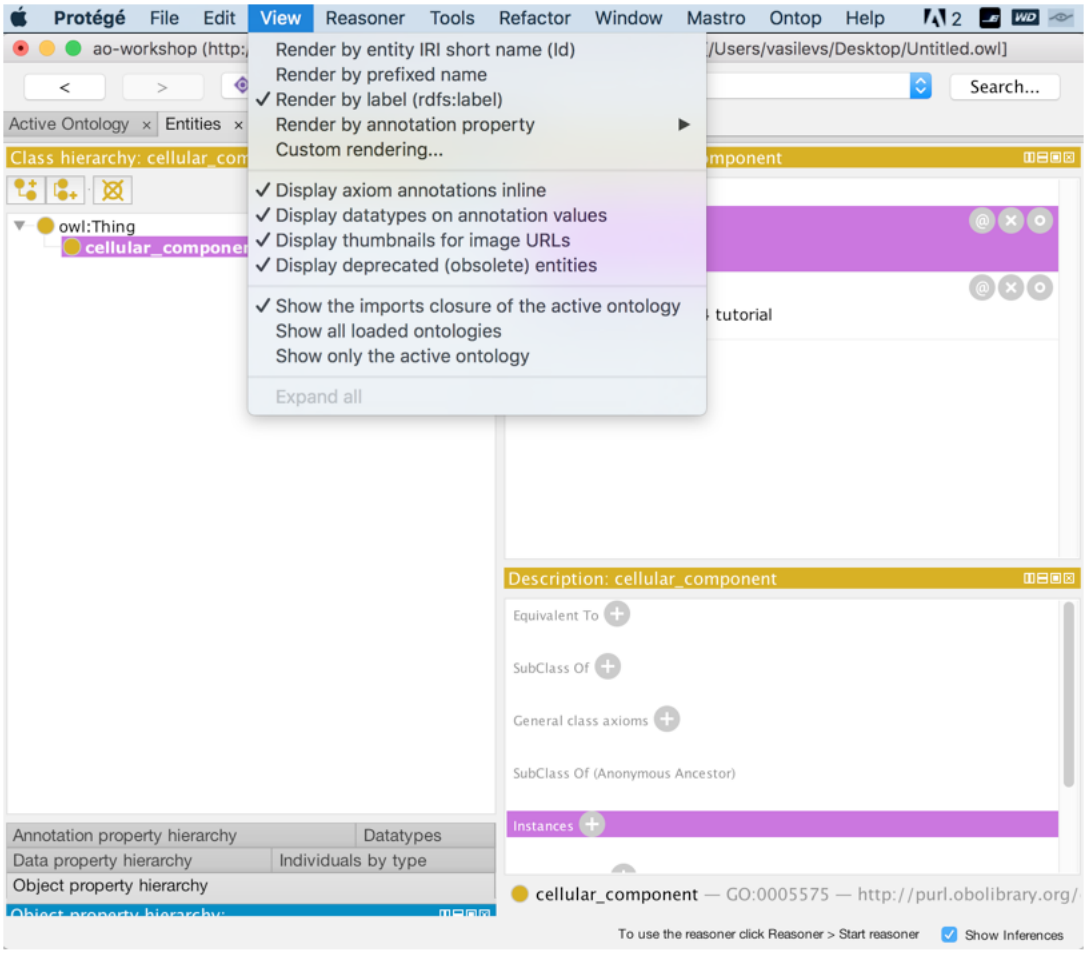


Note that often you will start from an existing OWL file, then your ontology will include a pre-declared set of annotation properties such as 'has exact synonym' and 'definition'. *You may never need to create your own annotation properties.*

3.5 Setting label rendering

You can change how Protégé renders entities. It is common to want to view entities by their **label**, rather than **identifiers.** In fact, you can tell Protégé to render on any annotation property you choose. Experiment with the different options in the menu, and to conclude, set the rendering to use the class label (rdfs:label).

In the View menu choose "Render by label":

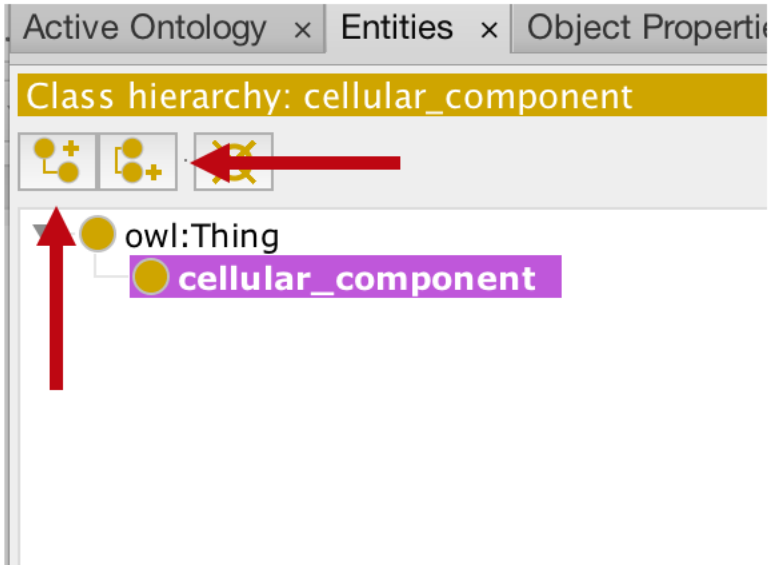


The **cellular_component** class will now render in the hierarchy view using the value of the label annotation property. Note that the ability to flip between different renderings can often be very useful in old versions of Protégé, as the Protégé search box in the upper right searches on whatever is rendered. If you are using an older version of Protégé, searching for a term by ID for example, it can be useful to render by ID and then flip back to render by label. In the 5.1 version, the search box will search for either labels or ID.

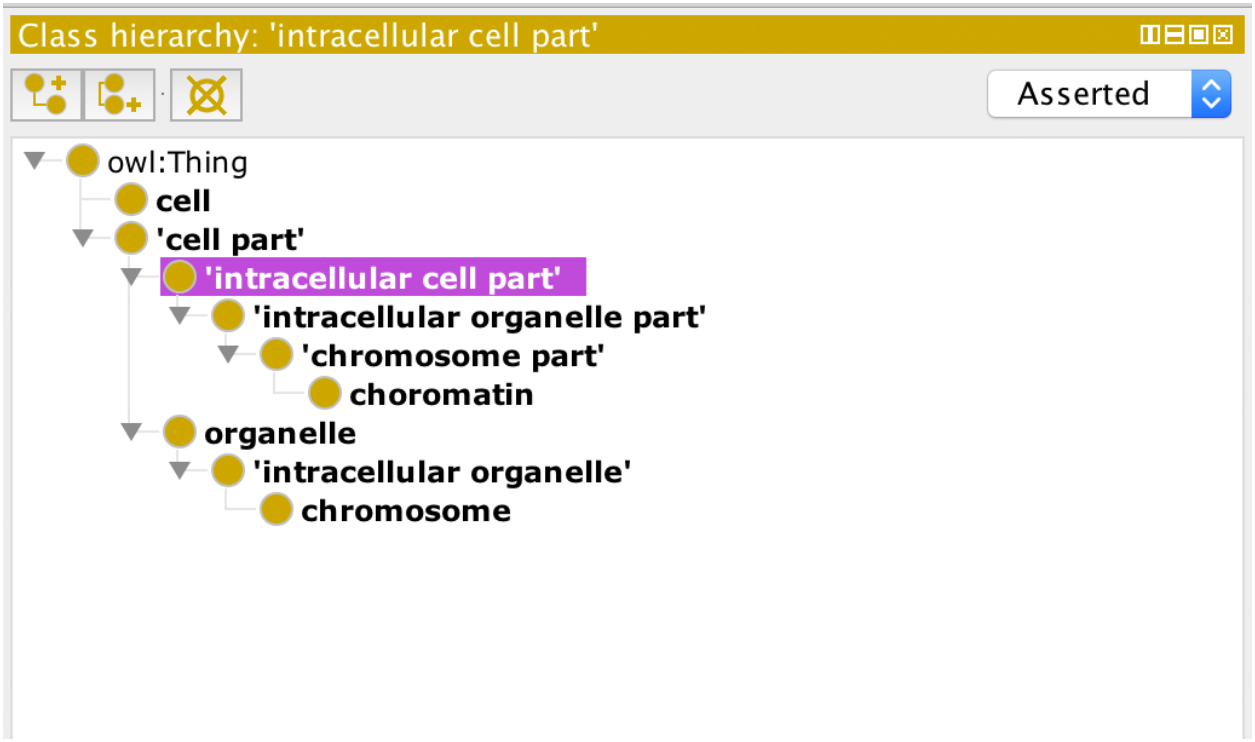
3.6 Creating the class hierarchy

We will now create a simple class hierarchy. In Protégé, 'class hierarchy' typically refers to a sub/superclass hierarchy (also known as an *'is_a'* hierarchy in OBO format). We will return to relations such as *'part_of'* later on in this tutorial. For now, we will take advantage of the fact that the GO cell component ontology allows us to bypass this for now by means of classes such as 'cell part' and 'nuclear part'.

Classes may be quickly added to an ontology with the **add subclass (vertical arrow below)** and **add sibling class (horizontal arrow below)** icons in the class hierarchy view.



Use the **add subclass (vertical arrow below)** and add **sibling class (horizontal arrow below)** buttons to create a hierarchy that looks like the following (your window will look slightly different than the view below which is from an earlier version of Protégé). Note that you can click and drag classes in the hierarchy to re-arrange them. When planning your own ontology take a good amount of time to standardize your label format, check other ontologies, and be consistent.



Don't bother to add textual definitions, synonyms, etc. at this stage, as you won't be using this ontology in latter exercises. *Note: the order of the classes in your Class hierarchy may not be the same as you see in the screenshot (e.g. 'cell part' may appear above cell). Don't worry about this. Just make sure that the subclass relationships are correct.*

CHAPTER 4

EXERCISE: Basic Subclass Hierarchy

Go to the directory basic-subclass in the BD2K14_exercises folder and open /basic-subclass/chromosome-parts.owl.

This example illustrates adding classes and class annotations into an existing Subclass hierarchy.

Note: This example does not make use of reasoning / automated classification or class expressions.

Constructs illustrated:

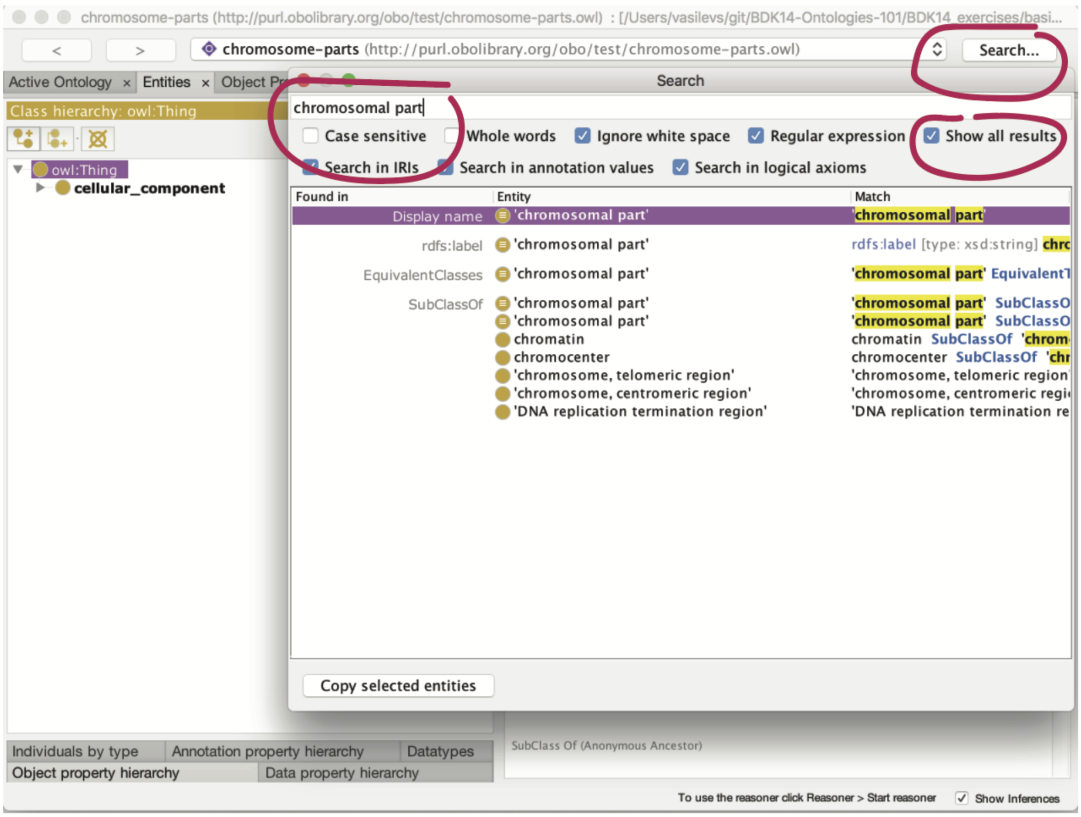
- Adding subclasses
- Adding annotations to classes
- Adding DatabaseCrossReferences (xref) classes

Instructions:

Add Subclass

1. Open chromosome-parts.owl and navigate to the Entities tab.
2. Add the class "replication fork" to the ontology as a subclass of 'chromosomal part'. (Don't worry about the ID.) *Note: Most ontologies use lowercase labels, except for proper names.*

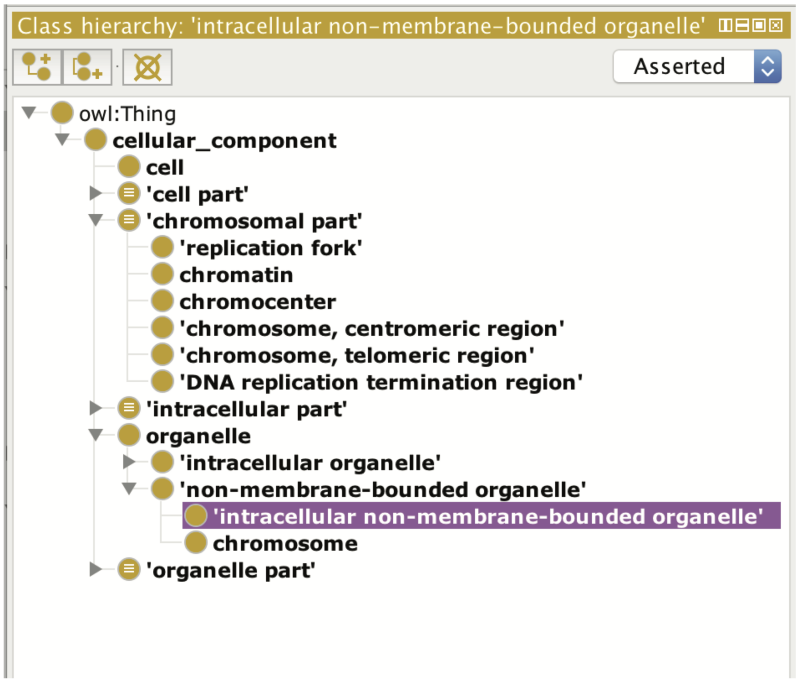
   a. To find the class 'chromosomal part', you can navigate through the hierarchy, or use the search function, see screenshot below. Make sure you select "Show all results". In the search results, double click on the highlighted class and it will open it up in the Class hierarchy pane.



Moving around classes:

1. Add the term "intracellular non-membrane-bounded organelle" as a subclass of "Thing"
2. Move it, by dragging and dropping, to place it as a subclass of 'non-membrane-bounded organelle' (if you do not see the class 'non-membrane-bounded organelle';, search for it by clicking the 'search' radio button in the upper right.)

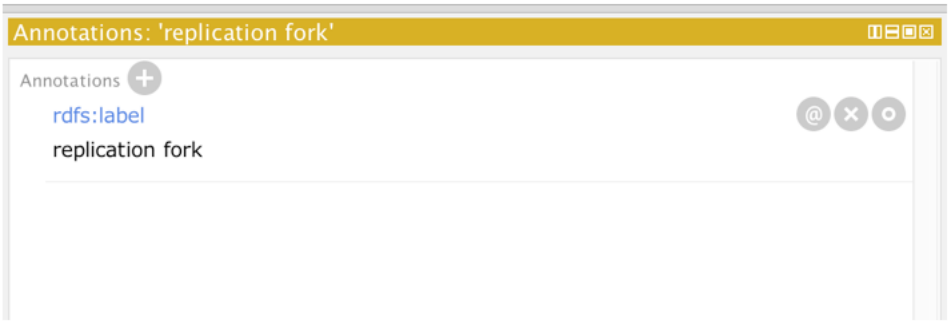
Your hierarchy should look something like this:



Annotations

The goal is to recreate the existing information from GO on the "replication fork" class.

1. Click on the 'replication fork' class you just created.
2. In the Annotations pane on the right, use the (+) next to 'Annotations' to add an annotation.



You will add the annotation values listed are below, detailed instructions follow. Make sure you click on the correct annotation on the left for each annotation.

id: GO_0005657

rdfs: label: replication fork

definition: The Y-shaped region of a replicating DNA molecule, resulting from the separation of the DNA strands and in which the synthesis of new strands takes place. Also includes associated protein complexes.

database_cross_reference: ISBN:0198547684

has_related_synonym: replication focus

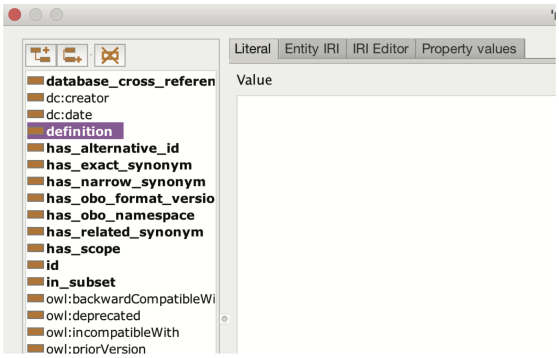
xref: Wikipedia:Replication_fork

Detailed instructions:

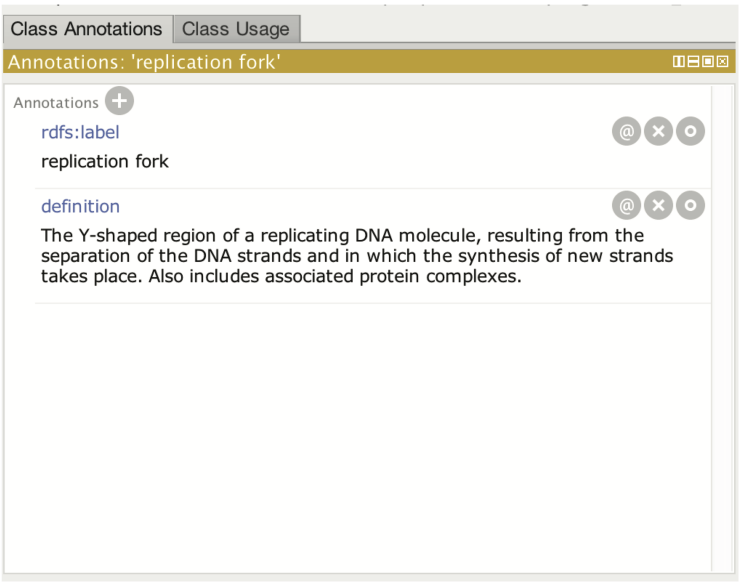
Add the following (using the values above)

1. **A text definition for the class**

- Click on the "replication fork" class, then click (+) by Annotations
- By default, the window should be on the "Literal" tab
- Click (select) "definition" on the left

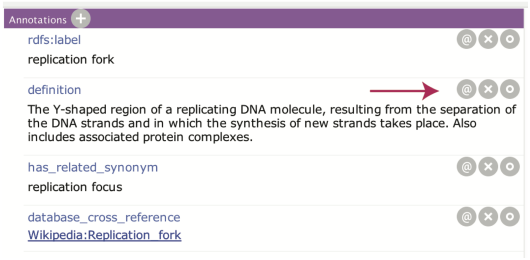


- Enter the definition in the window: The Y-shaped region of a replicating DNA molecule, resulting from the separation of the DNA strands and in which the synthesis of new strands takes place. Also includes associated protein complexes. *Note: Make sure there are not any extra spaces at the end of the sentence.*
- Click OK. The annotation should appear in the Annotations window.

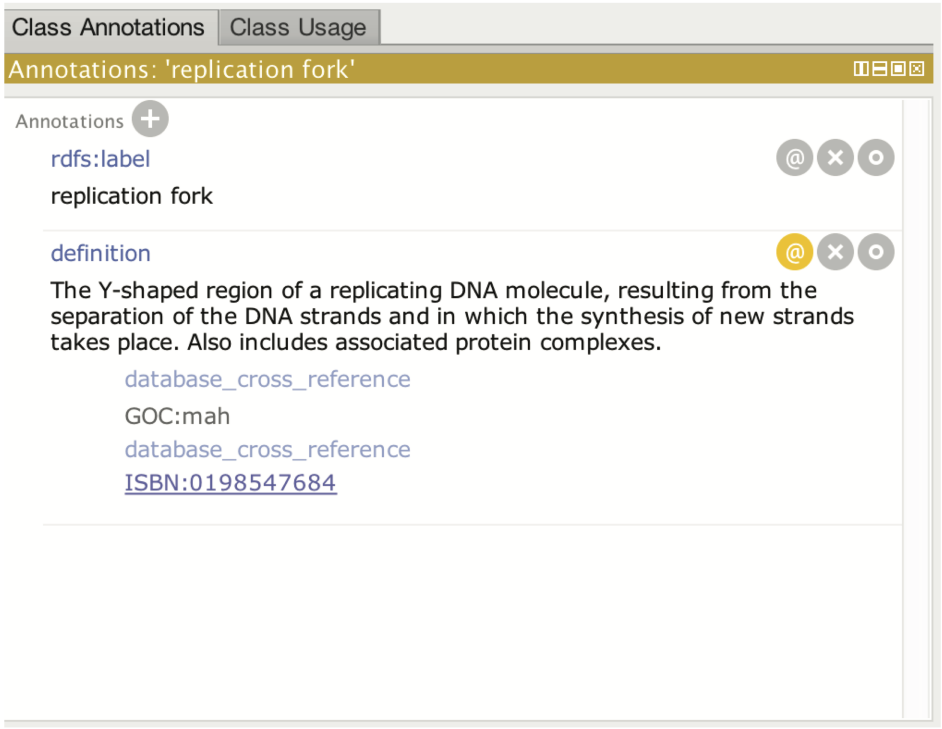


2. dbxrefs to the text definition

- Click the (@) icon beside the definition annotation



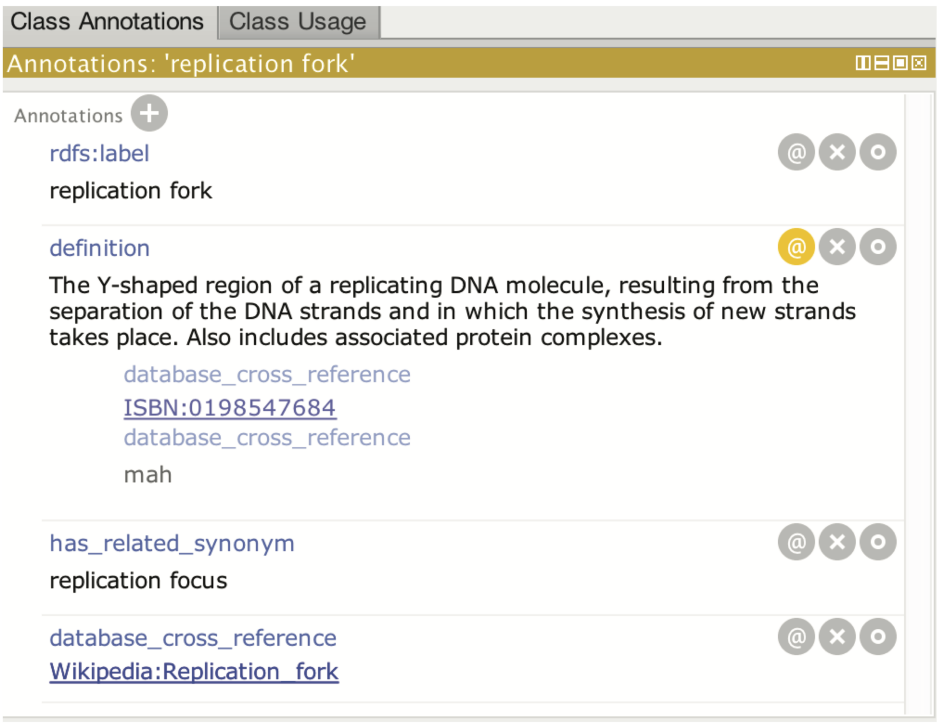
- A new box will open, and click the (+) beside Annotations
- Select “database_cross_reference” in the left pane
- Enter a value. This is often a PubMed ID (in the format PMID:xxxxxx) or your ORCID ID or initials. For this exercise, enter the initials: mah.
- Repeat for the other cross reference: ISBN:0198547684
- Your annotations should look like the screenshot below.



A related synonym

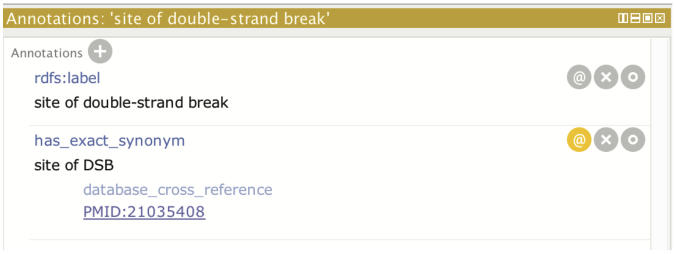
- Add an annotation to “replication fork” with the (+)
- Choose “has_related_synonym” and enter the value: replication focus
- Add an xref to the class itself:
- click the (+) beside annotations
- Choose database_cross_reference
- Add xref (Wikipedia:Replication_fork)
- Click OK

Your annotation pane should look something like this:



Synonym properties:

1. Add the subclass "site of double-strand break" to the ontology under "chromosomal part"
2. Add a synonym with a dbxref annotation. E.g. synonym: "site of DSB" has_exact_synonym [PMID:21035408]



Note that there are different synonym annotations:

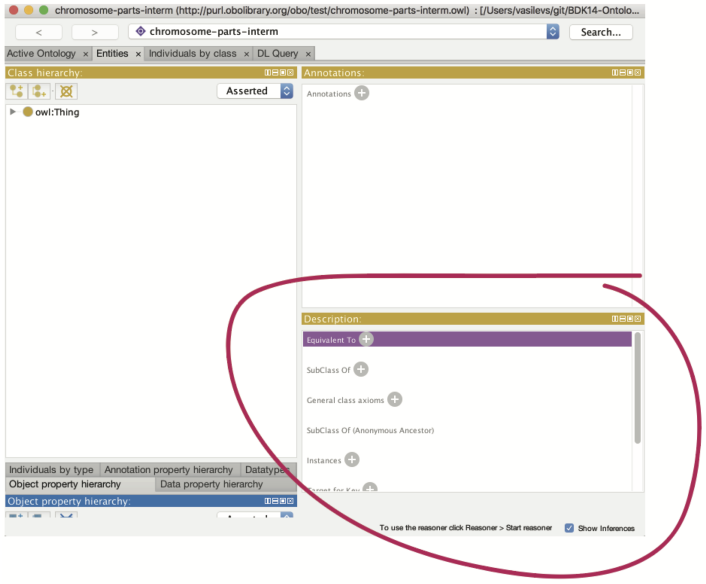
- has_exact_synonym: a synonym that has the exact same meaning as the class name
- has_narrow_synonym: a more specific synonym
- has_related_synonym: a related term

4.1 The Class description view

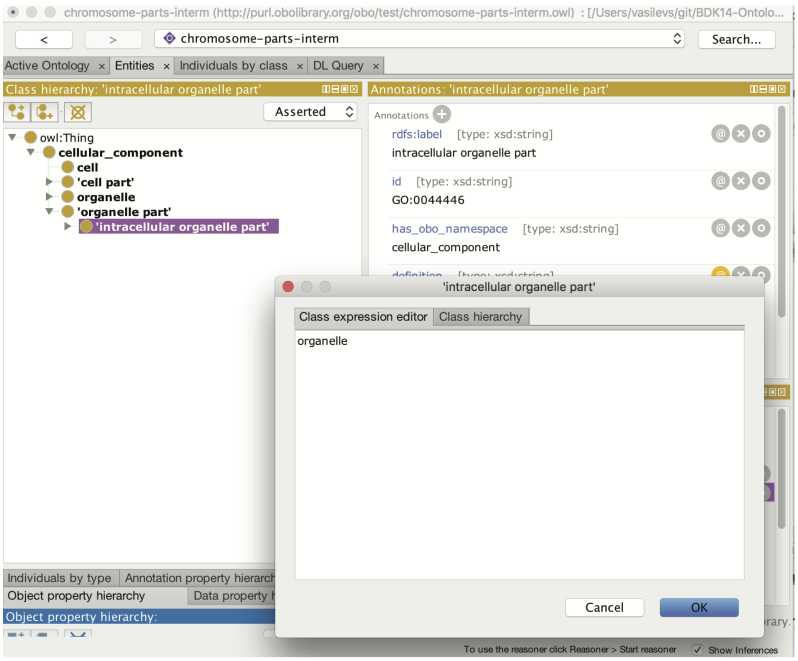
Open the ontology chromosome-parts-interim.owl, found in the "basic-subclass" exercise folder. Save it to your local computer using Save-as.

We have seen how to add sub/superclasses and annotate the class hierarchy. Another way to do the same thing is via the Class description view (circled in the figure below). When an OWL class is selected in the entities view, the right-hand side of the tab shows the class description panel. If we select the **cell** class, we see in the class description view

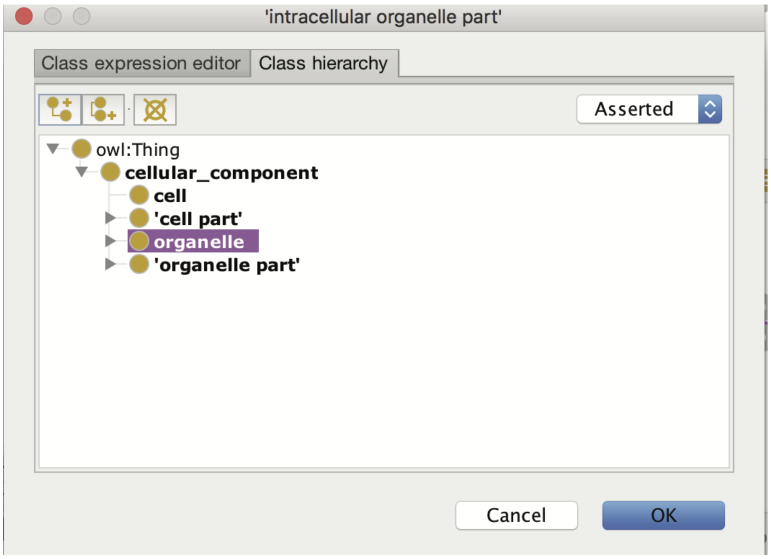
that this class is a "SubClass Of" (= has a *SuperClass*) the **cellular_component** class. Using the (+) button beside "SubClass Of" we could add another superclass to the cell class.



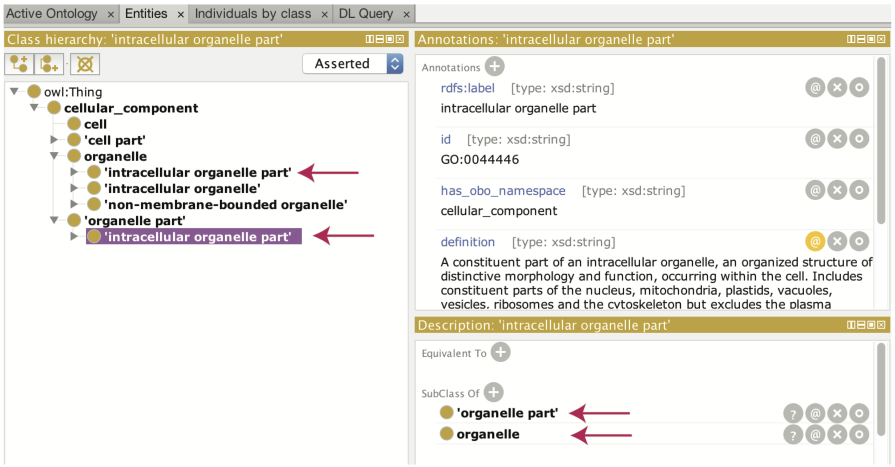
Select the **intracellular organelle part** class in your ontology. Notice it is a SubClass of **organelle part.** Using the SubClass Of (+) button, add the **organelle** class as a super class. There are various ways to assert a superclass. The simplest it to just type in the class expression editor. *Hint: Pressing Tab (or CTRL + SPACE on a Mac) allows you to autocomplete on a term.*



You can also use the class hierarchy tab here to search, browse and select the appropriate class.



The **intracellular organelle part** class will now have two parents asserted in the class hierarchy. You will also be able to see both parents in the class description view.



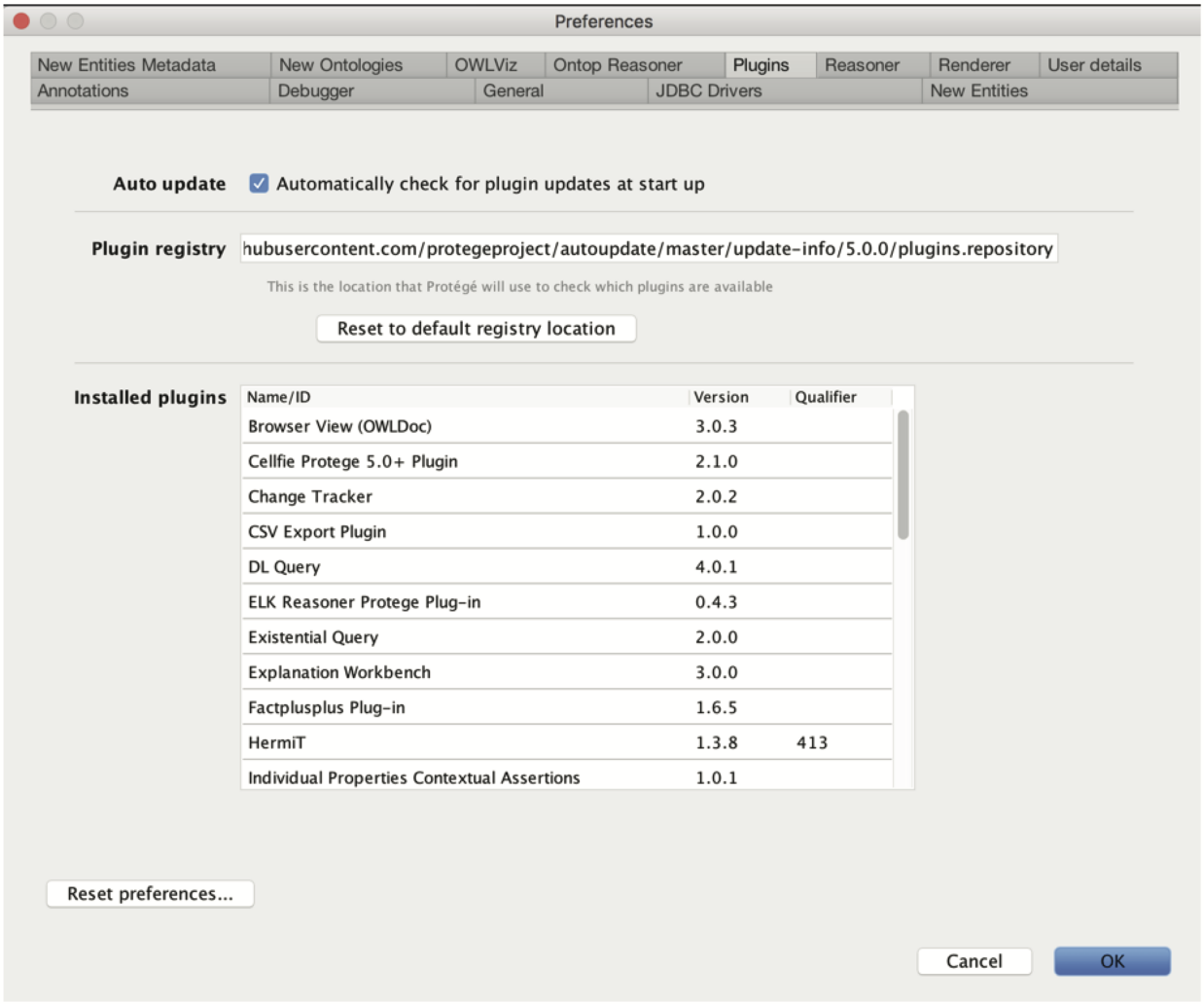
Save your interim ontology to your computer. *Note: You will use this same file in the next section: Disjointness.*

CHAPTER 5

Protégé plugins

Protégé is built on a plugin architecture. There is an active community of developers writing plugin extensions to Protégé. There is a plugin library in Protégé that allows you to pick and install plugins. You may also find plugins elsewhere on the web that must be installed manually. (*Note: Plugins are distributed as java archives (jars). To manually install a plugin you simply need to place the jar in the plugins folder inside the Protégé home/root directory.)*

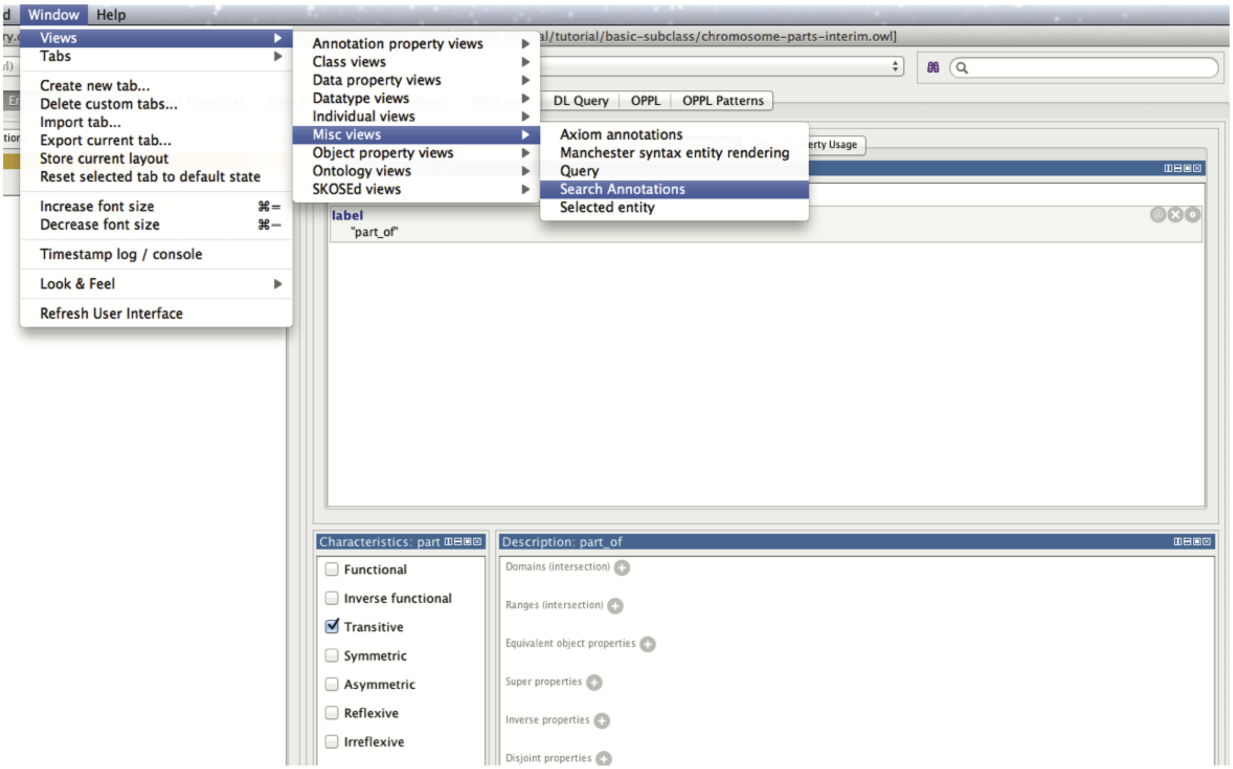
You can find the plugin library in the Protégé preferences. Go to your preferences and then the Plugins tab.

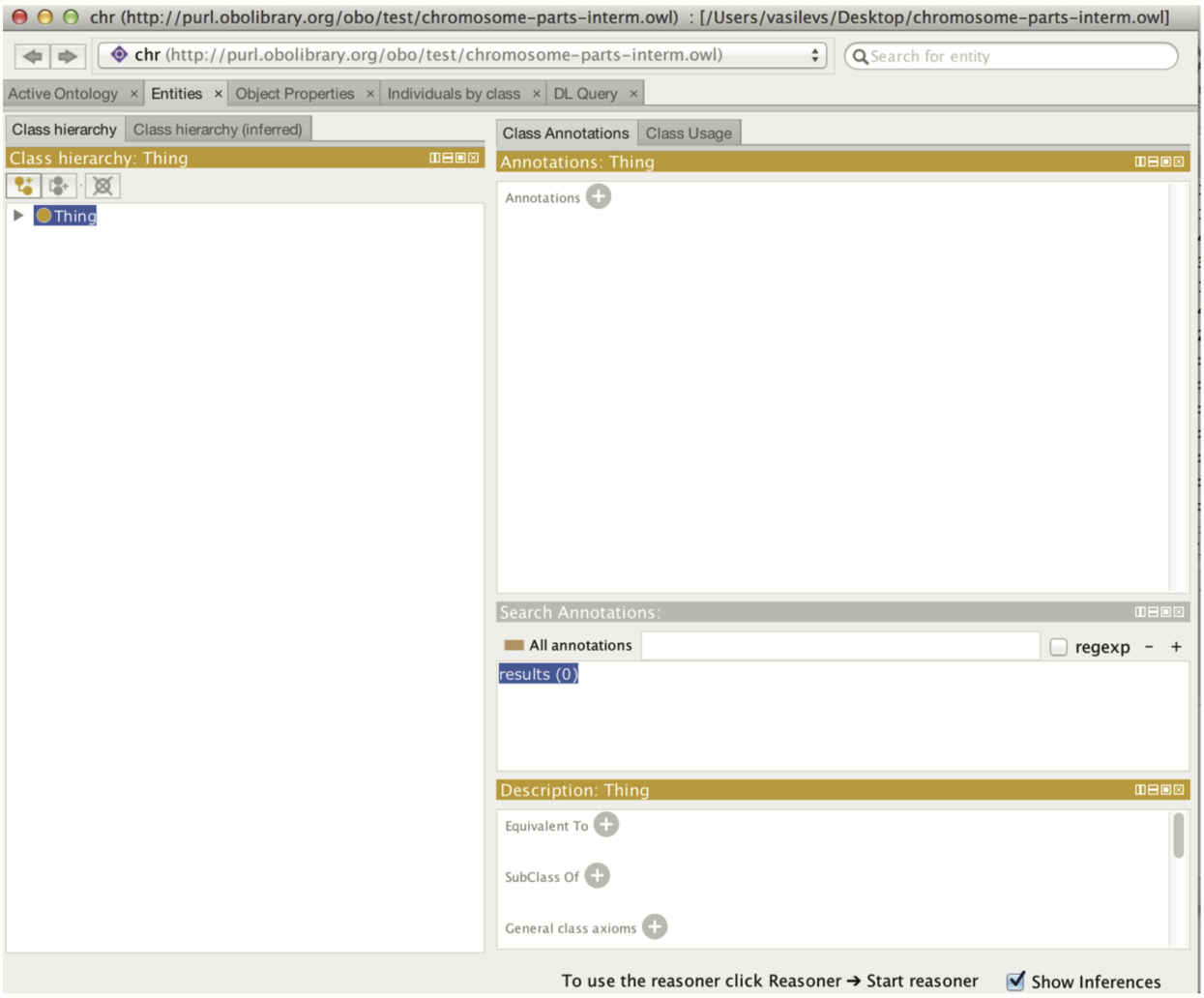


In older versions of Protégé, you may want to install the **Annotation Search Views** and the **Outline/Existential Tree** plugins. Instructions for older versions are below. Note: this is not necessary for Protégé 5.1. If you are using Protégé 5.1, skip to the next section on Disjointness.

5.1 Annotation search plugin (for older versions of Protege)

Most plugins are either tabs, panels or menu items. The annotations search plugin provides a new panel that can be used to search through OWL annotations (such as labels and definitions). Tabs and panels can be found in the Window menu. Under Window -> Views -> Misc views -> Search Annotations. Once selected your pointer will become a circular icon. You can choose to drop this panel over any existing panel in Protégé by clicking. We recommend that you drop this panel to the right of the class hierarchy view, on top of the existing annotation view panel.


Window
Help
Views
Tabs
Create new tab...
Delete custom tabs...
Import tab...
Export current tab...
Store current layout
Reset selected tab to default state
Increase font size ⌘=
Decrease font size ⌘−
Timestamp log / console
Look & Feel
Refresh User Interface
Annotation property views
Class views
Data property views
Datatype views
Individual views
Misc views
Object property views
Ontology views
SKOSEd views
Axiom annotations
Manchester syntax entity rendering
Query
Search Annotations
Selected entity
/tutorial/basic-subclass/chromosome-parts-interim.owl]
DL Query
OPPL
OPPL Patterns
label
"part_of"
Characteristics: part
Functional
Inverse functional
Transitive
Symmetric
Asymmetric
Reflexive
Irreflexive
Description: part_of
Domains (intersection)
Ranges (intersection)
Equivalent object properties
Super properties
Inverse properties
Disjoint properties



You can use the annotation search panel to search through all annotations, or restrict it to individual annotations, such as the label. The annotation view also supports *regular expression* queries.

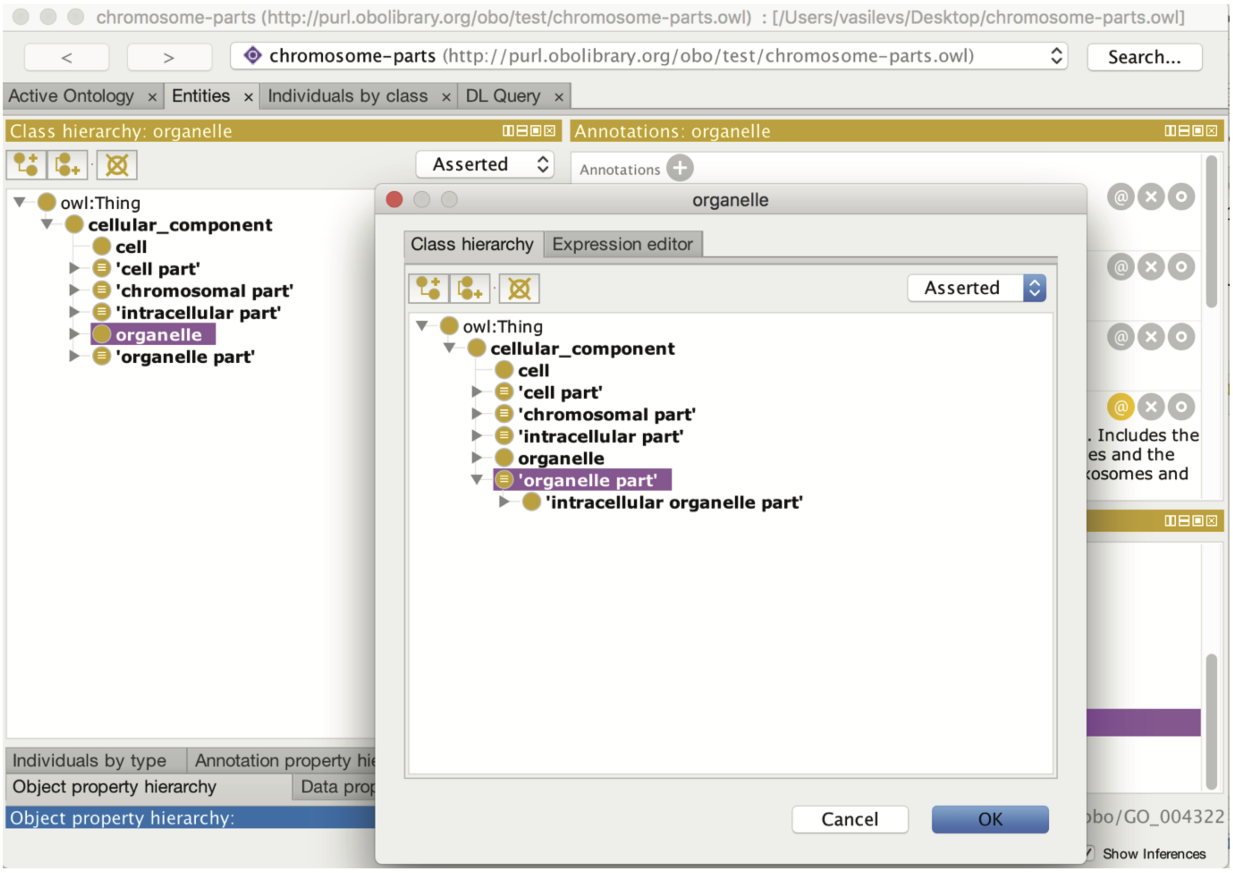
CHAPTER 6

Disjointness

In the chromosome-parts-interim.owl file (that you have now saved locally on your computer), at the top of our class hierarchy we have cell, cell part, chromosomal part, intracellular part, organelle and organelle part. By default, OWL assumes that these classes can overlap, i.e. there are individuals who can be instances of more than one of these classes. We want to create a restriction on our ontology that states these classes are different and that no individual can be a member of more than one of these classes. We can say this in OWL by creating a *disjoint classes* axiom.

If you do not already have it open, load your previous ontology that was derived from the 'interim file'. Note: you can open a recent file by going to File-> Open Recent

We want to assert that **organelle** and **organelle part** are disjoint. To do this first select the **organelle** class. In the class 'Description' view, scroll down and select the (+) button next to Disjoint With. You are presented with the now familiar window allowing you to select, or type, to choose a class. In the hierarchy panel, you can use CTRL to select multiple classes. Select 'organelle part' as disjoint with organelle.



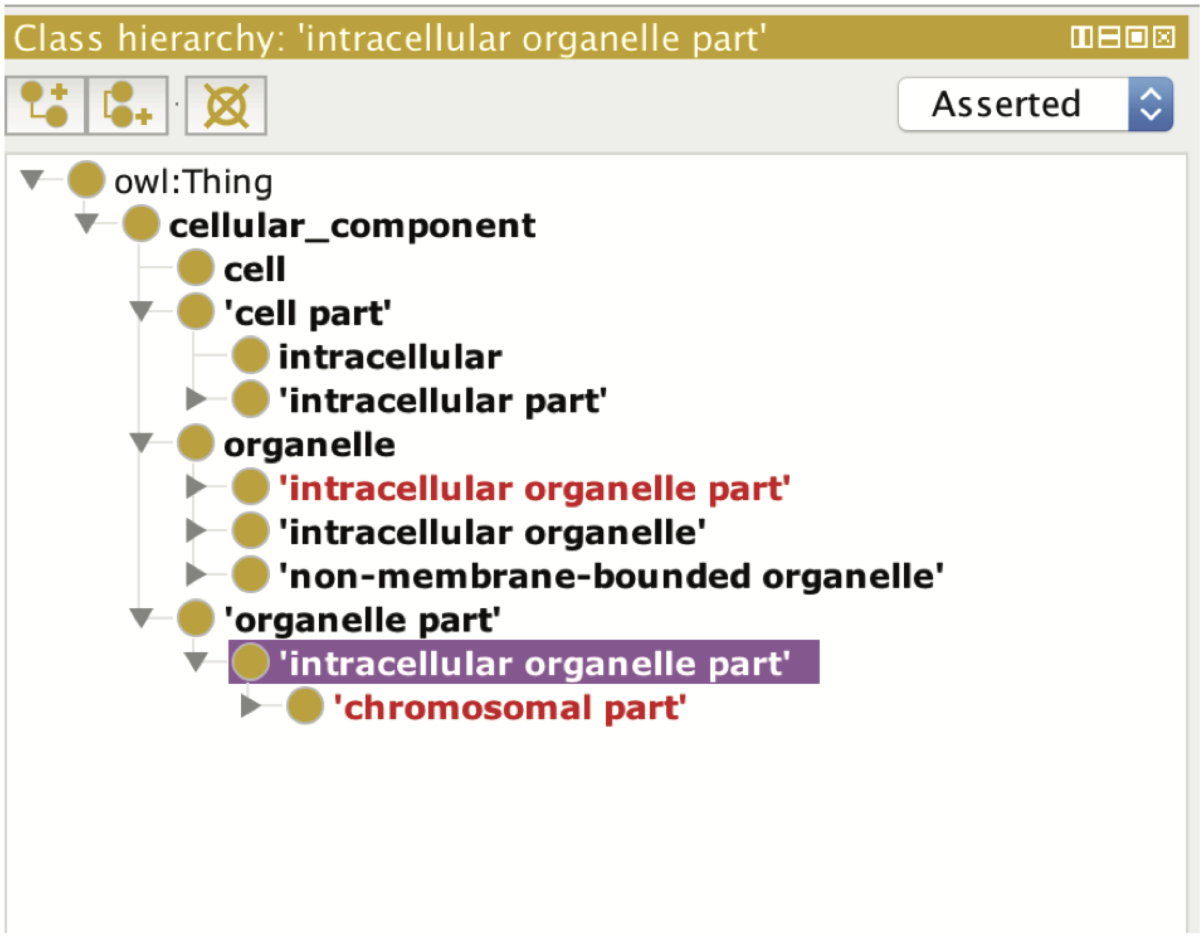
Note that the directionality is irrelevant. Prove this to yourself by deleting the disjoint axiom, and adding it back from **organelle part**.

6.1 Reasoning and inconsistency checking

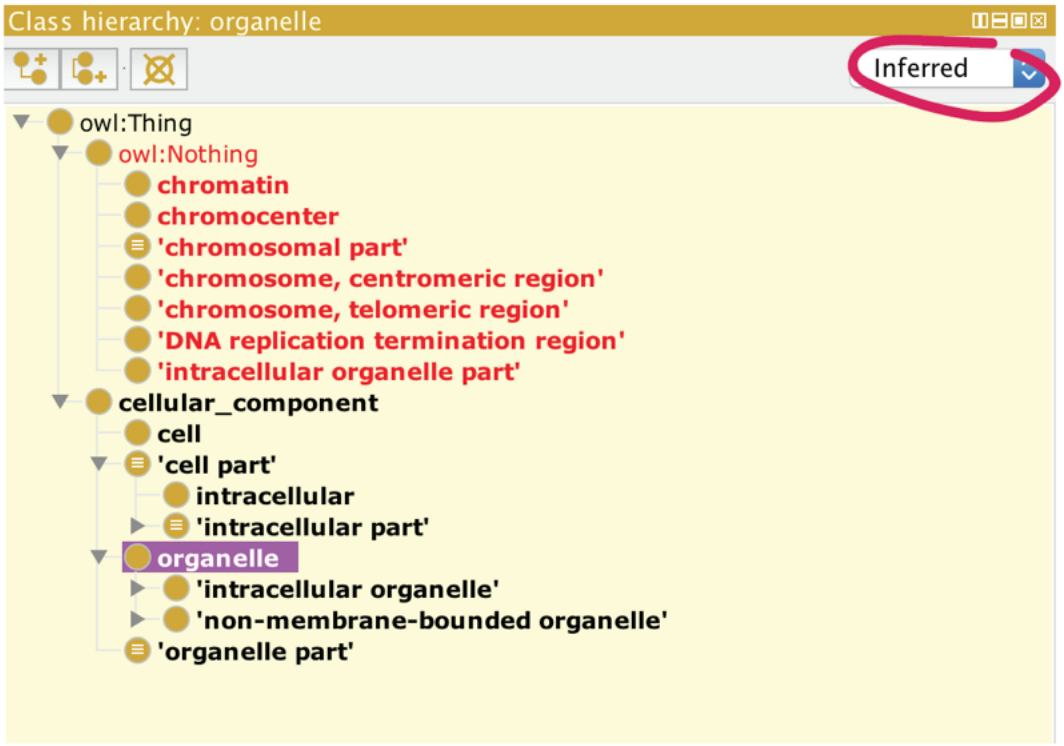
We have introduced a deliberate mistake into the ontology. We previously asserted that **intracellular organelle part** is a subclass of both **organelle part** and **organelle**. We have now added an axiom stating that **organelle** and **organelle part** are disjoint. We can use the reasoner to check the consistency of our ontology. The reasoner should detect our contradiction.

Protégé comes with several reasoners, and more can be installed via the plugins mechanism (see plugins chapter). Select a reasoner from the Reasoner menu (Elk, HermiT, Pellet, or Fact++ will work). Once a reasoner is highlighted, select 'Start reasoner' from the menu. *Note: you may get several pop-boxes/warnings, ignore those.*

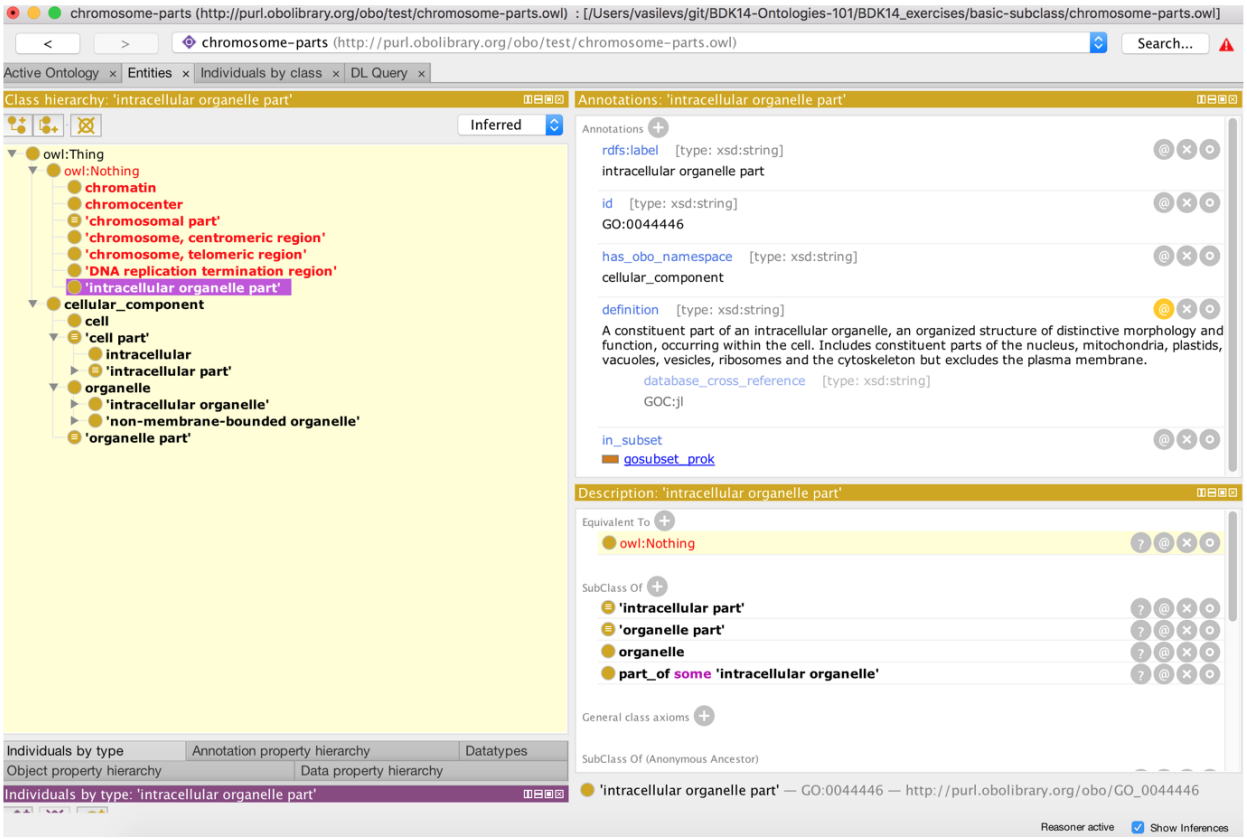
The **intracellular organelle part** class will have changed to red indicating that the class is now *unsatisfiable*.



You can also see unsatisfiable classes by switching to the inferred view.

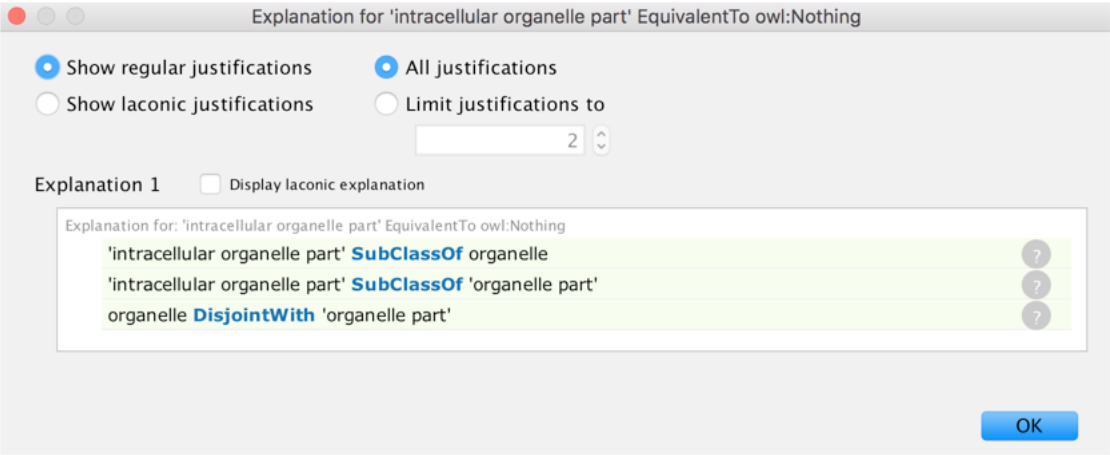


Here you will a special class called **Nothing**. When we previously said that all OWL classes are subclasses of OWL Thing. OWL **Nothing** is a leaf class or bottom class of your ontology. Any classes that are deemed unsatisfiable by the reasoner are shown as subclasses or equivalent to OWL Nothing. The inferred view will show you all subclasses of Nothing.



Once the ontology is classified, inferred statements or axioms are shown in the various panels with a light-yellow shading. The class description for **intracellular organelle part** should look something like the screen shot below. You will see that the class has been asserted equivalent to the **Nothing** class. Inside this statement, a small question mark icon appears, clicking this will get an explanation from the reasoner for this inconsistency.

Select the (?) icon to get an explanation for this inconsistency. The explanation shows the axioms involved. We see the disjoint class axiom alongside the two subclass axioms are causing the inconsistency. We can simply repair this ontology by removing the **intracellular organelle part** subClassOf **organelle** axiom.



Remove the Disjoint with axiom (click the (x) beside **organelle** in the Description pane for **intracellular organelle part**), and resynchronise the reasoner from the reasoner menu. Save your ontology, you'll return to it after this following exercise.

CHAPTER 7

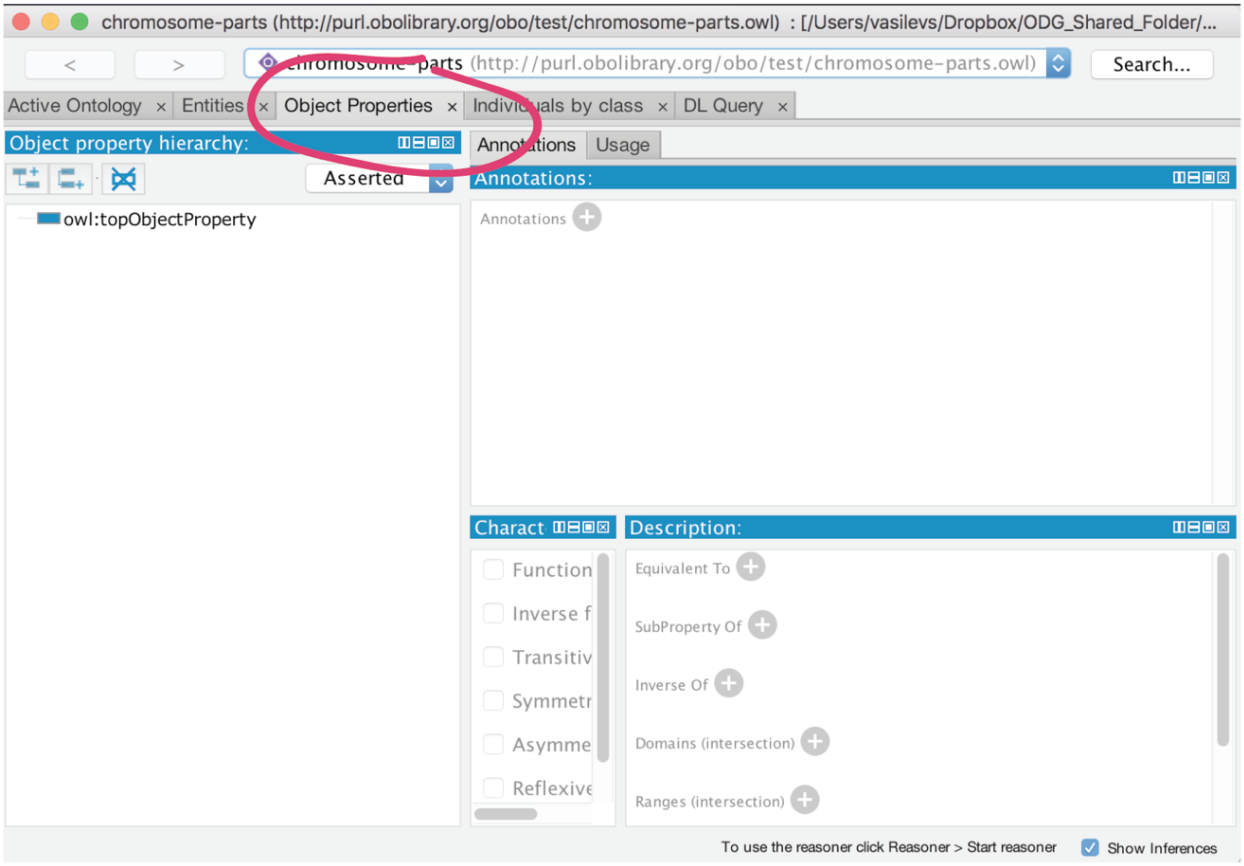
Object properties

Re-open your previously saved ontology, which you may have named something like "chromosome-parts-interim.owl".

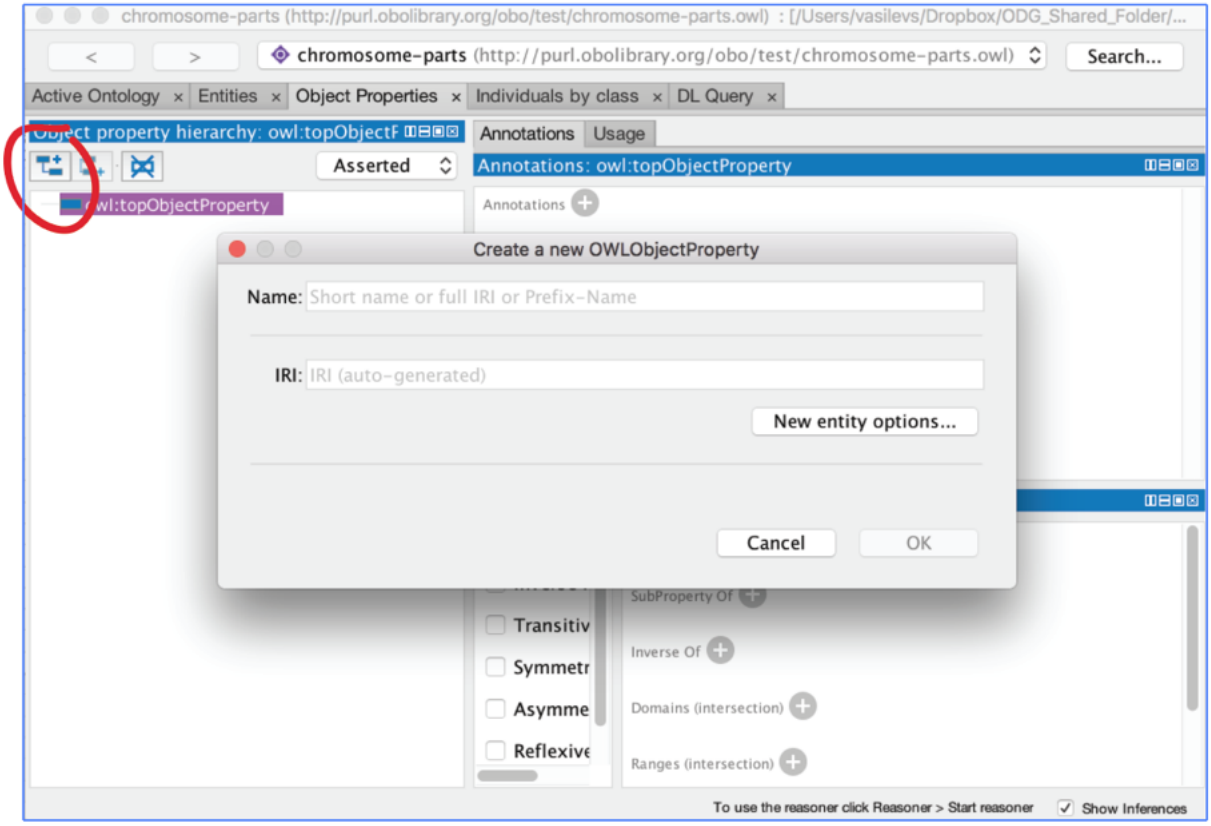
We will now create an object property and use this to add some restrictions onto classes. In OWL, object properties are used to assert relationships between individuals (or instances). Object properties in OWL can have characteristics such as being *transitive* or *symmetric*. We can assert additional information about properties such their *domain* and *range*, along with defining *inverse* properties.

7.1 Create an object property

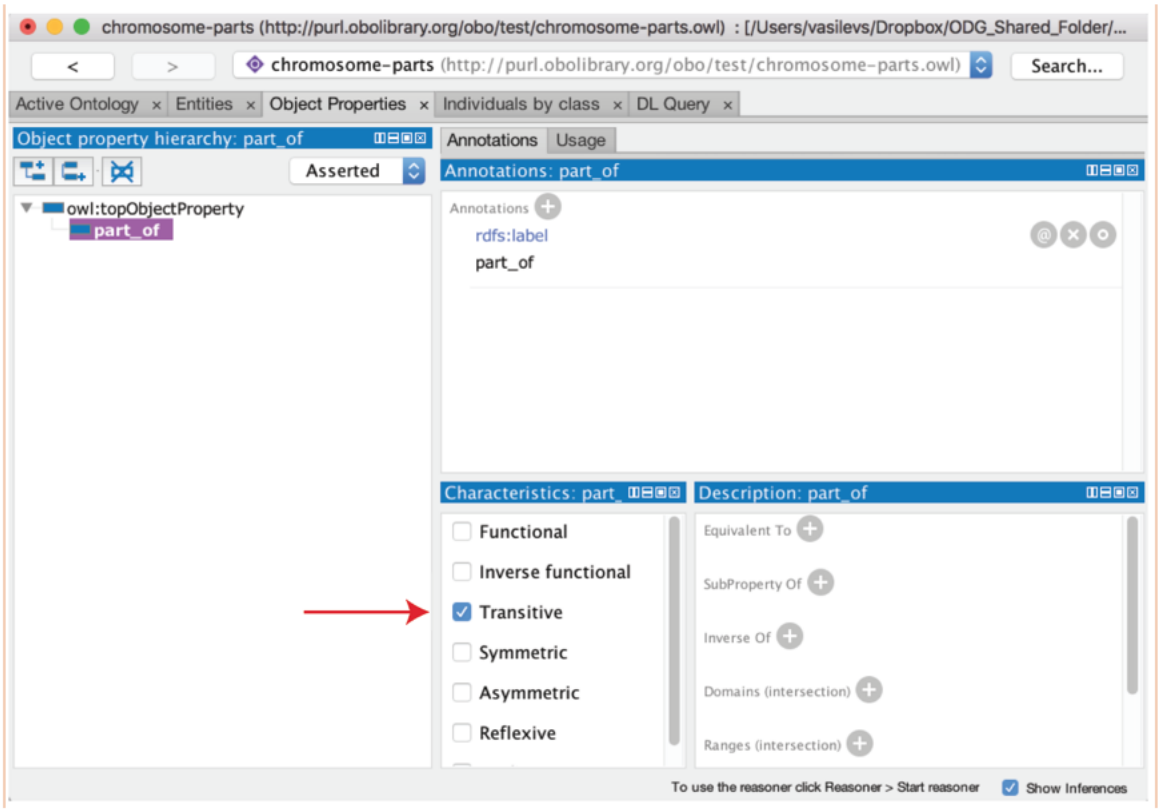
We will use the object property view in the Object Properties tab to create a part_of property. In OWL, all properties are a sub-property of topObjectProperty.



Select owl:topObjectProperty, then the “Add sub property button” circled below and name the property part_of.



We can use the property description view shown below to make assertions about this property. We want to state that the **part_of** property has the characteristic of being transitive. If a property is transitive, and the property relates individual a to individual b, and also individual b to individual c, then we can infer that individual a is related to individual c via property P. A good example of a transitive property is the geneological 'ancestor of' relationship. We can make a property transitive in Protégé by simply selecting the transitive check box.


chromosome-parts (http://purl.obolibrary.org/obo/test/chromosome-parts.owl) : [/Users/vasilevs/Dropbox/ODG_Shared_Folder/...
chromosome-parts (http://purl.obolibrary.org/obo/test/chromosome-parts.owl)
Search...
Active Ontology
Entities
Object Properties
Individuals by class
DL Query
Object property hierarchy: part_of
Asserted
owl:topObjectProperty
part_of
Annotations
Usage
Annotations: part_of
Annotations
rdfs:label
part_of
Characteristics: part_
Functional
Inverse functional
Transitive
Symmetric
Asymmetric
Reflexive
Description: part_of
Equivalent To
SubProperty Of
Inverse Of
Domains (intersection)
Ranges (intersection)
To use the reasoner click Reasoner > Start reasoner
Show Inferences

CHAPTER 8

OWL class restrictions

Keep the previously opened interim ontology open.

As previously stated, in OWL we use object property to describe binary relationships between two individuals (or instances). We can also use the properties to describe new classes (or sets of individuals) using *restrictions*. A restriction describes a class of individuals based on the relationships that members of the class participate in. In other words, a restriction is a kind of class, in the same way that a named class is a kind of class.

For example, we can use a named class to capture all the individuals that are chromosome parts. But we could also describe the class of chromosome parts as all the instances that are '*part of'* a chromosome.

In OWL, there are three main types of restrictions that can be placed on classes. These are **quantifier restriction** , **cardinality restrictions** and **hasValue** restriction. In this tutorial will initially focus on quantifier restrictions.

Quantifier restrictions are further categorized into two types, the **existential** and the **universal** restriction.

- **Existential** restrictions describe classes of individuals that participate in at least one relationship along a specified property to individuals that are members of a specified class. For example, the class of individuals that have at least one (**some**) 'part of' relationship to members of the 'Chromosome clas. In Protégé, the keyword ' **some'** is used to denote existential restrictions.
- **Universal** restrictions describe classes of individuals that for a given property only have relationships along this property to individuals that are members of a specified class. For example, we can say a cellular component is capable of many functions using the existential quantifier, however, OWL semantics assume that there could be more. We can use the universal quantifier to add closureto the existential. That is, we can assert that a cellular component is capable of these function, and is only capable of those functions and no other. Another example is that the process of hair growth is found **only** in instances of the class Mammalia. In Protégé the keyword '**only**' is used.

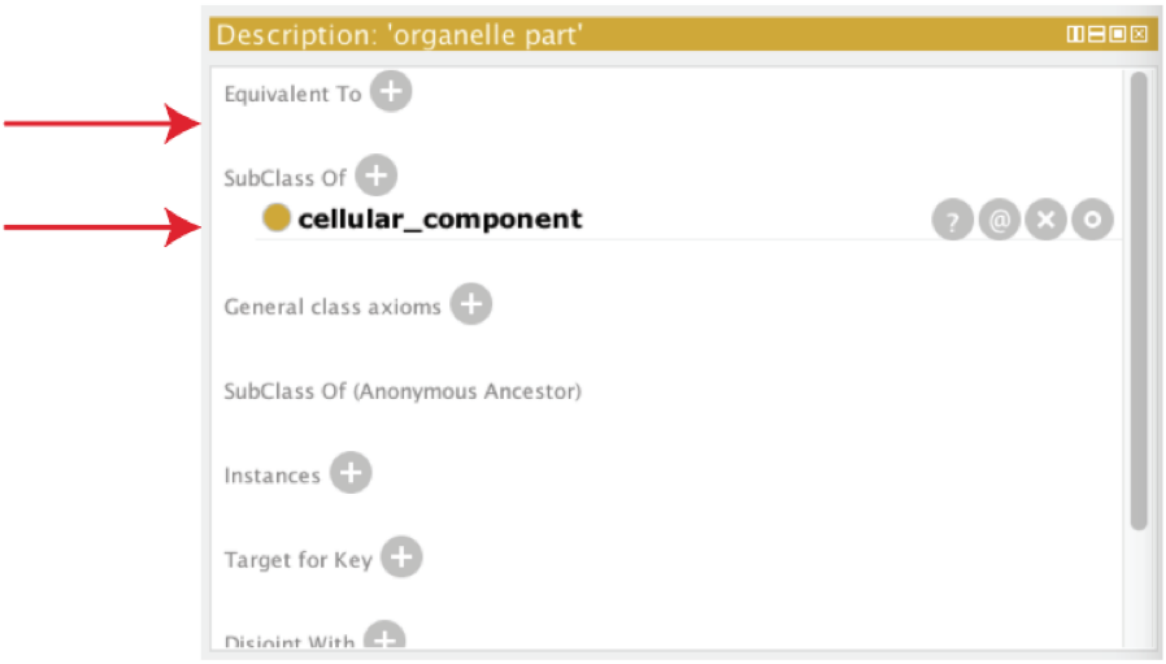
In this tutorial, we will deal exclusively with the existential (some) quantifier.

8.1 Superclass restrictions

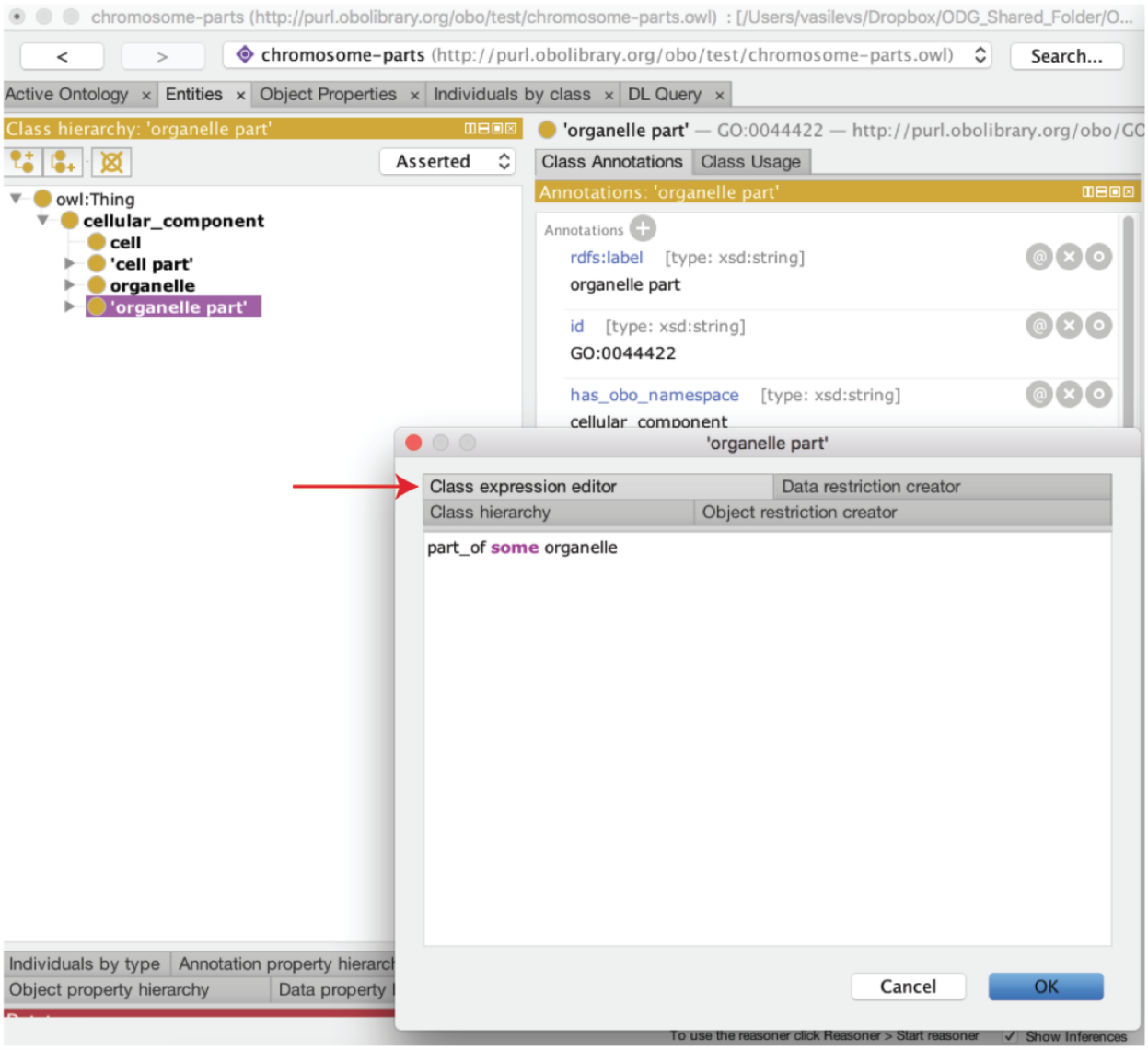
Strictly speaking in OWL, you don't make relationships between classes, however, using OWL restrictions we essentially achieve the same thing.

We want to capture the knowledge that the named class ' **organelle part**' is part of an **organelle**. In OWL speak, we want to say that every instance of an ' **organelle part**' is also an instance of the class of things that have at least one 'part of' relationship to an ' **organelle'**. In OWL, we do this by creating an existential restriction on the ' **organelle part**' class.

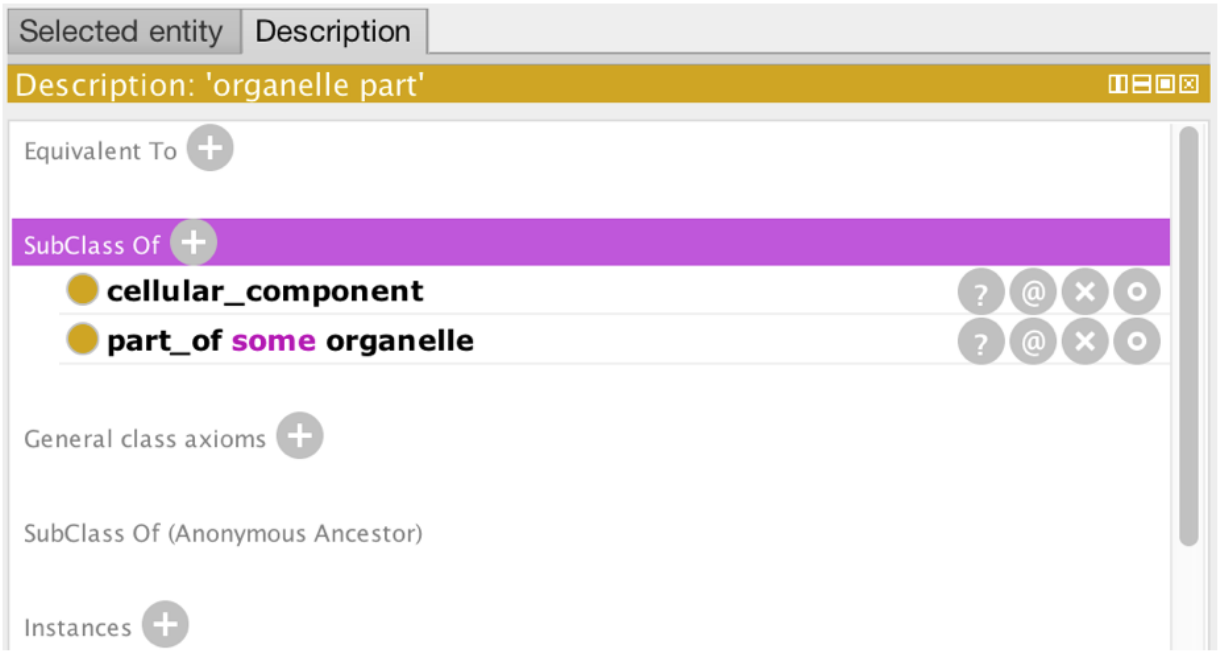
In the Entities tab, select '**organelle part**' in the class hierarchy and look at its current class description in the bottom right box. At the top of this view there are two slots for defining **equivalent classes** and superclasses (as denoted by the SubClass Of list). '**organelle part**' already has one superclass named **cellular_component**.



We will create a restriction on '**organelle part**' stating '**organelle part**' has a *'part of'* relationship to some '**organelle**'. Select the (+) icon next to the SubClass Of slot. Select the Class expression editor pane. We will define this *anonymous superclass* in Manchester OWL syntax as 'part_of some organelle'.



The class restriction will be shown in the SubClass of slot as follows.



Using Protégé create your own part_of restrictions for the '**cell part**', '**intracellular part**' and '**chromosomal part**' classes. *Note: you must use single quotes around text strings that are separated by a space, e.g. 'intracellular organelle part'.*

NOTE: After each edit to the ontology you might want to synchronize the reasoner to make sure you didn't introduce any inconsistencies into your ontology. The edit, reason, edit, reason iteration becomes particularly important as your ontologies grow more complex.

CHAPTER 9

EXERCISE: Basic Restictions

This example illustrates how to use object properties to make existential restrictions.

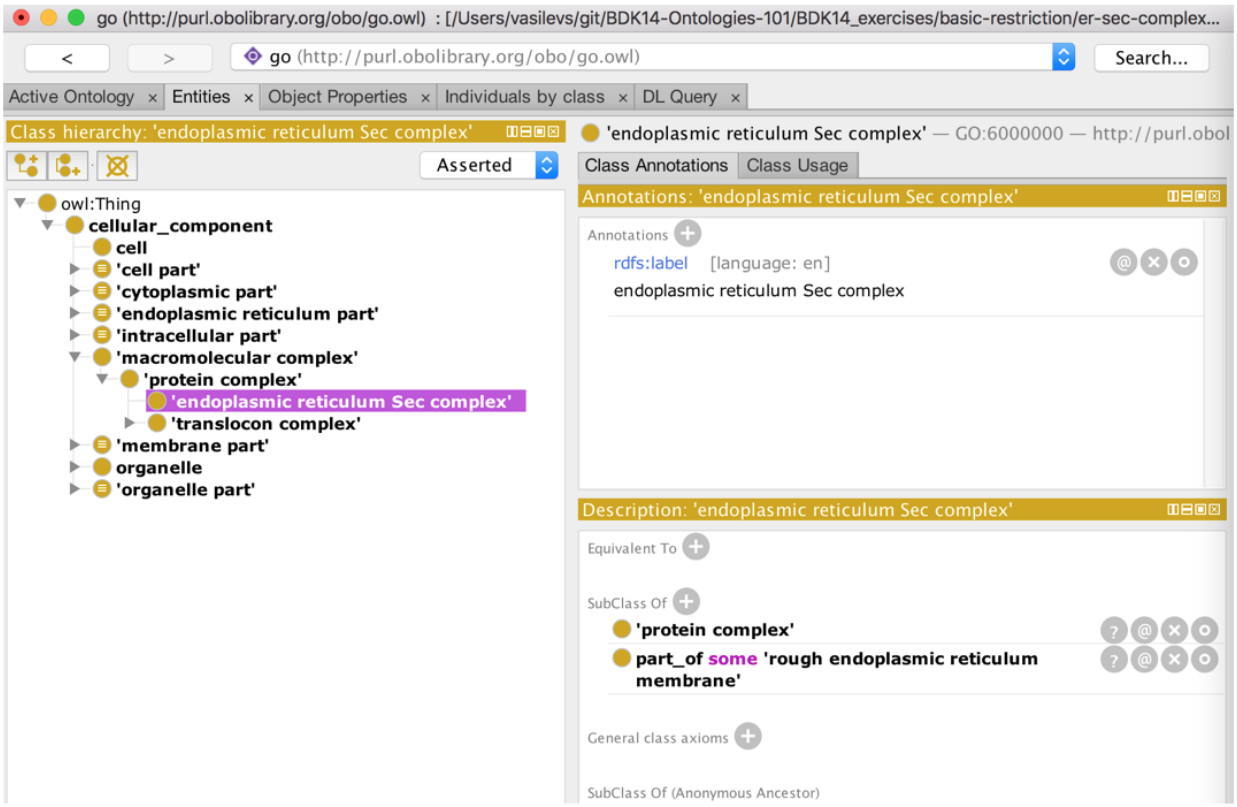
In OWL, it helps to think in terms of the set of entities represented by each class. To say: 'every finger is part of a hand' we say:

finger SubClassOf part_of some hand

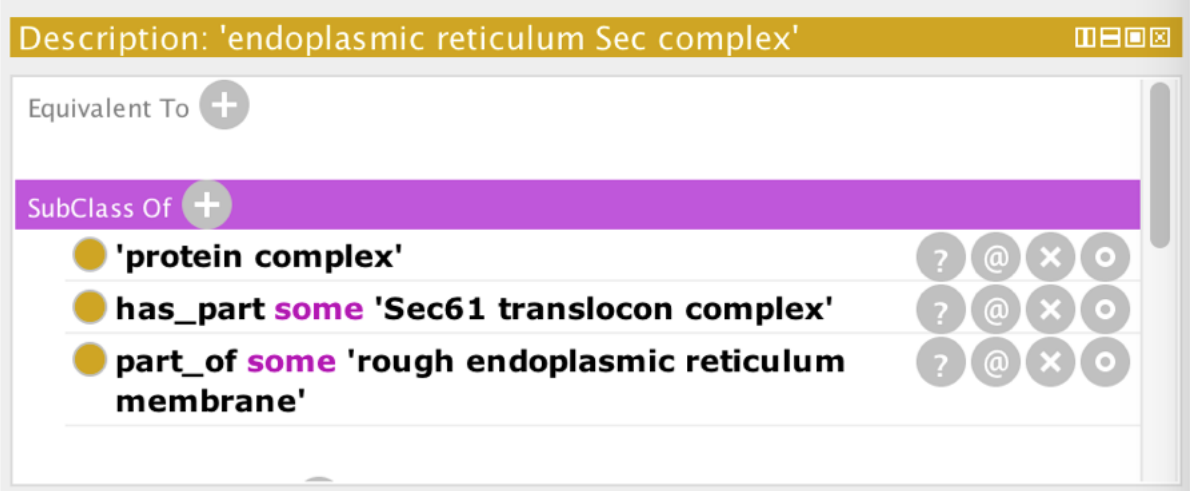
The anonymous class expression 'part_of some hand'; represents the set of all instances that have a part_of relationship to a hand. Every member of the set of all fingers is a member of the set of all things that are part of a hand.

Instructions:

1. Open er-sec-complex.owl from the basic-restiction folder
2. Navigate to the class 'protein complex' using the search box
3. Add a class 'endoplasmic reticulum Sec complex' as a subclass of 'protein complex'
4. Say that every 'endoplasmic reticulum Sec complex' is part of a 'rough endoplasmic reticulum membrane'

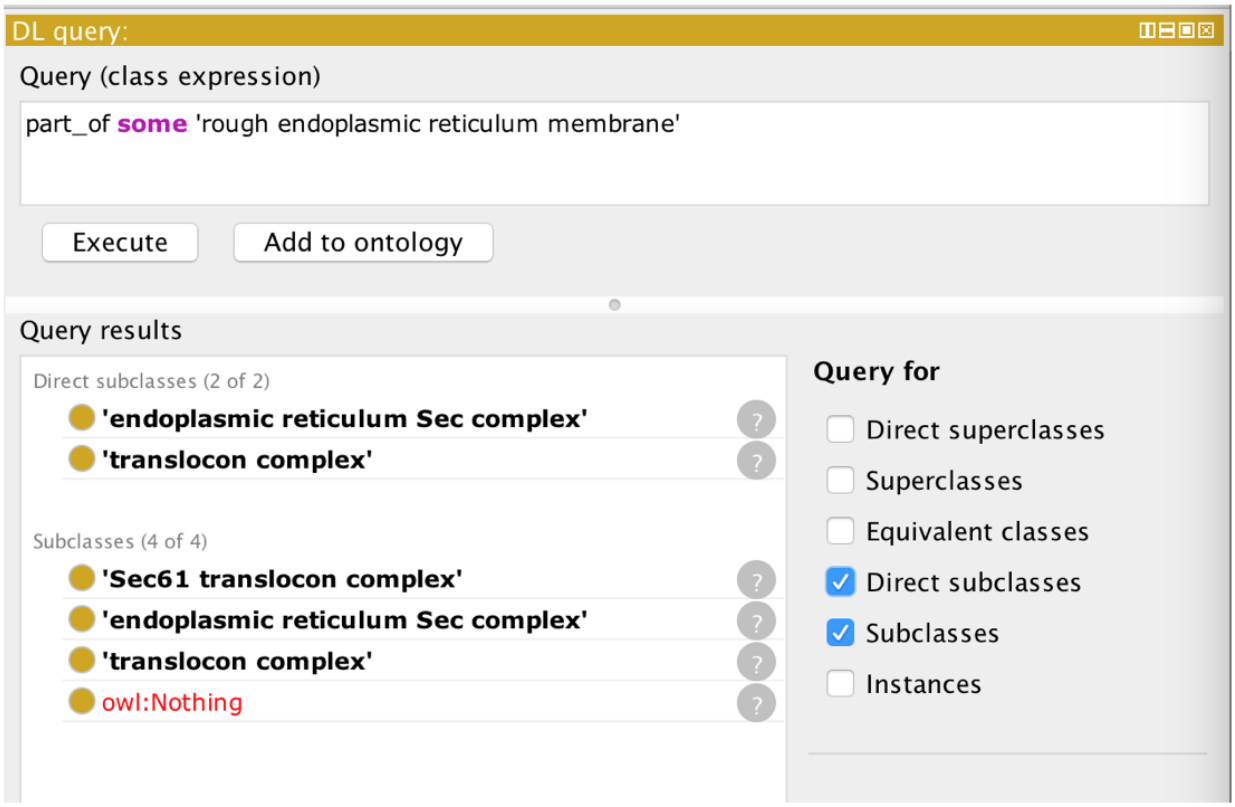


1. Say that a 'endoplasmic reticulum Sec complex' has a 'Sec61 translocon complex' as part



Navigating over the resulting ontology:

1. Synchronize the reasoner
2. Navigate to 'rough endoplasmic reticulum membrane'.
3. Find the parts of the rough ER membrane. To do this, go to the DL query tab and write the query as depicted below. Your results should look something like the screenshot below.



Aside

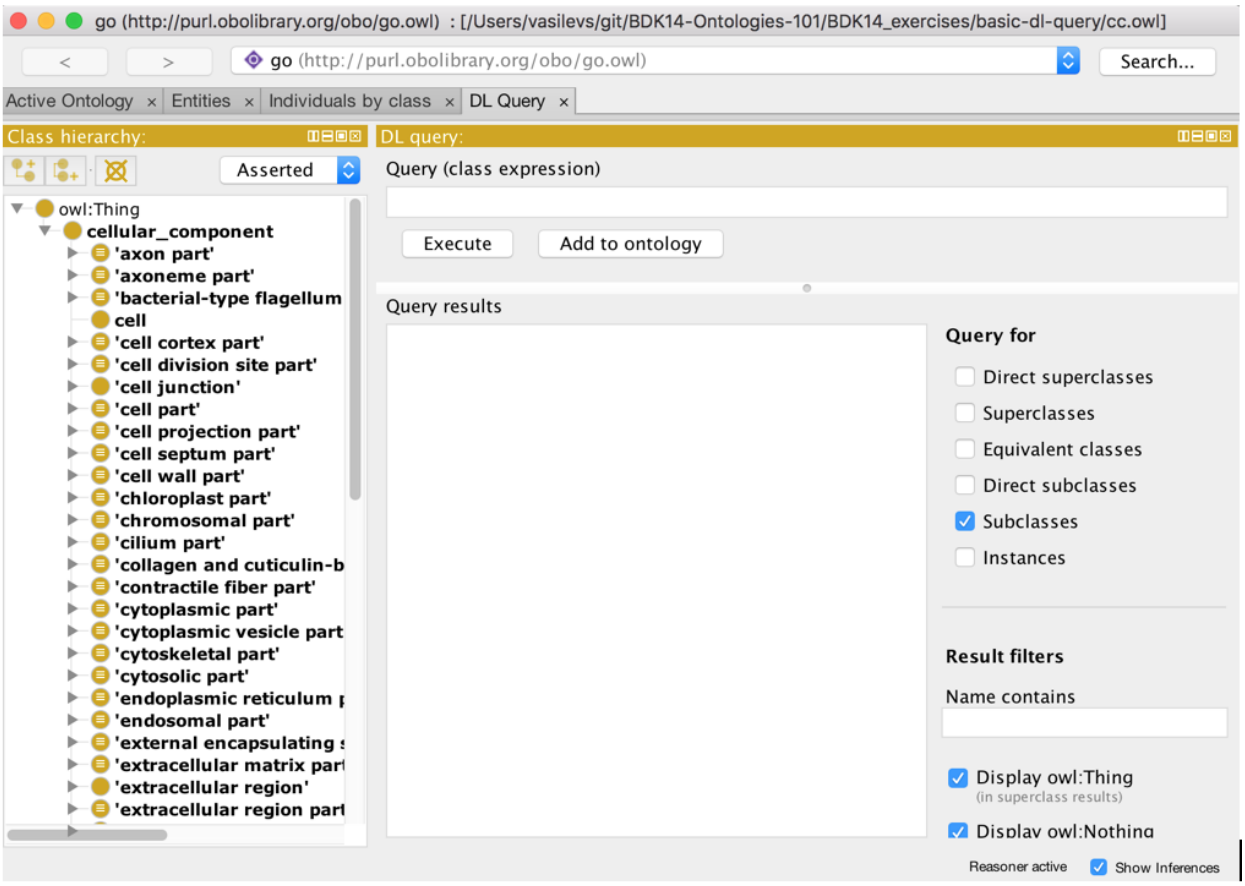
If you like, you can look up the current GO file (open go.owl from http://purl.obolibrary.org/obo/go.owl directly in Protégé -> File-> 'open from URL' to examine how the part of restrictions in the actual ontology were created.

CHAPTER 10

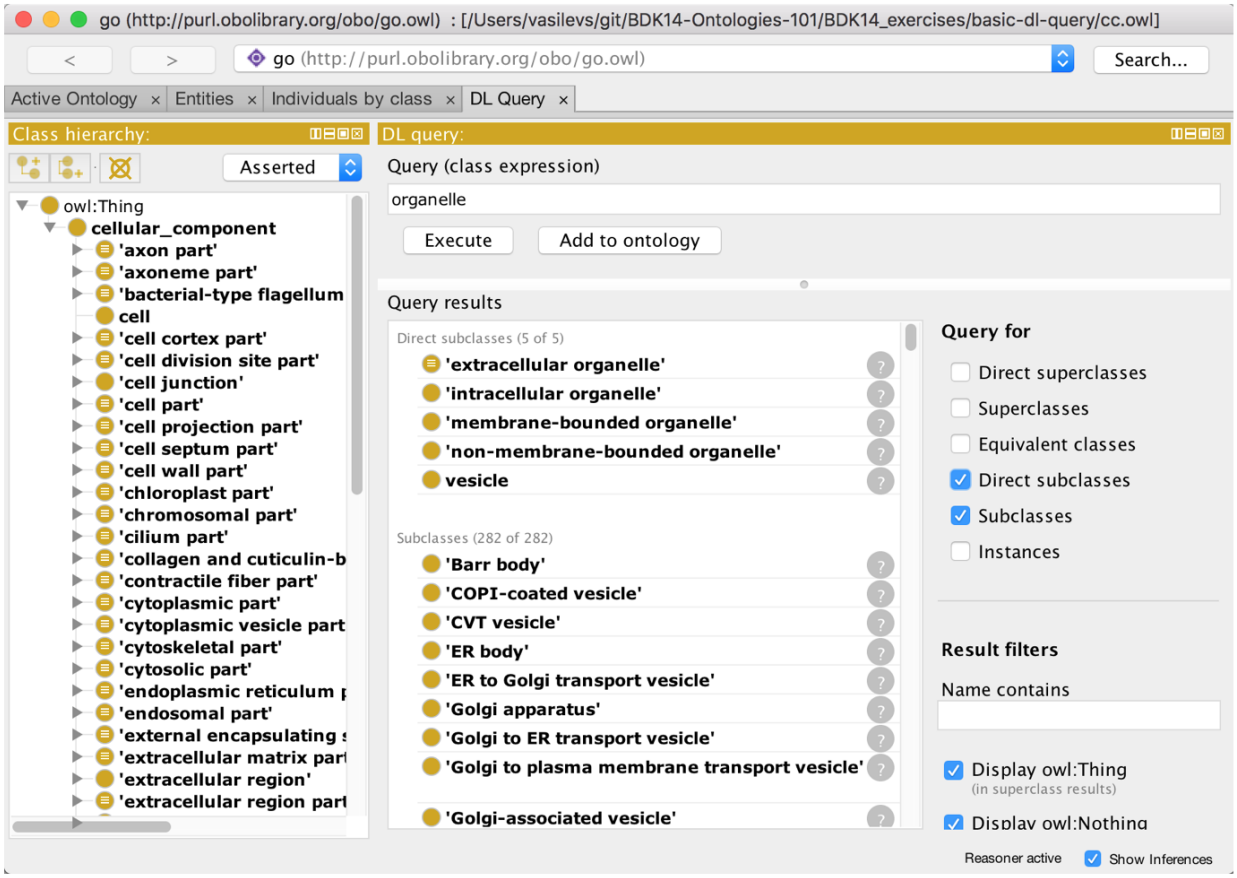
DL query tab

The DL query tab shown below provides an interface for querying and searching an ontology. The ontology must be classified by a reasoner before it can be queried in the DL query tab.

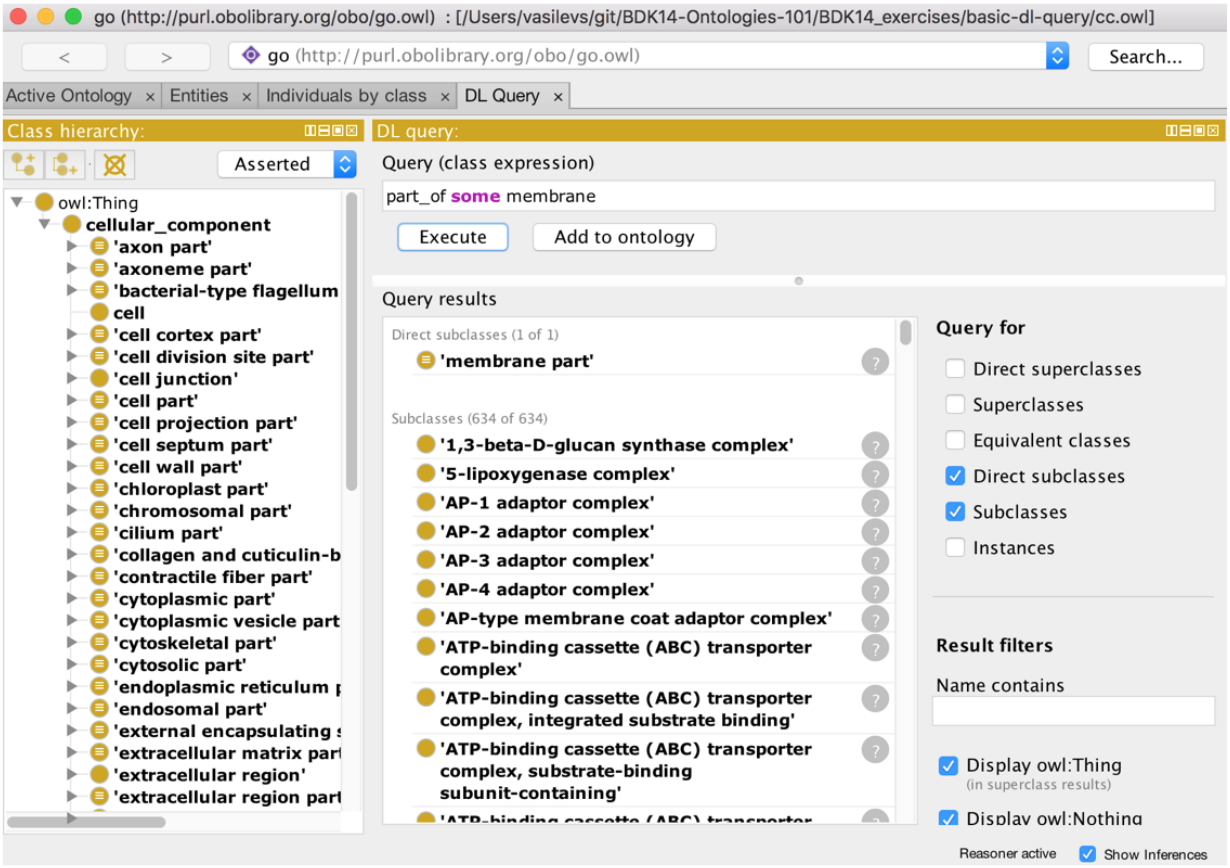
Go to the “basic-dl-query” folder and open “cc.owl”. Run the reasoner. Navigate to the DL Query tab.



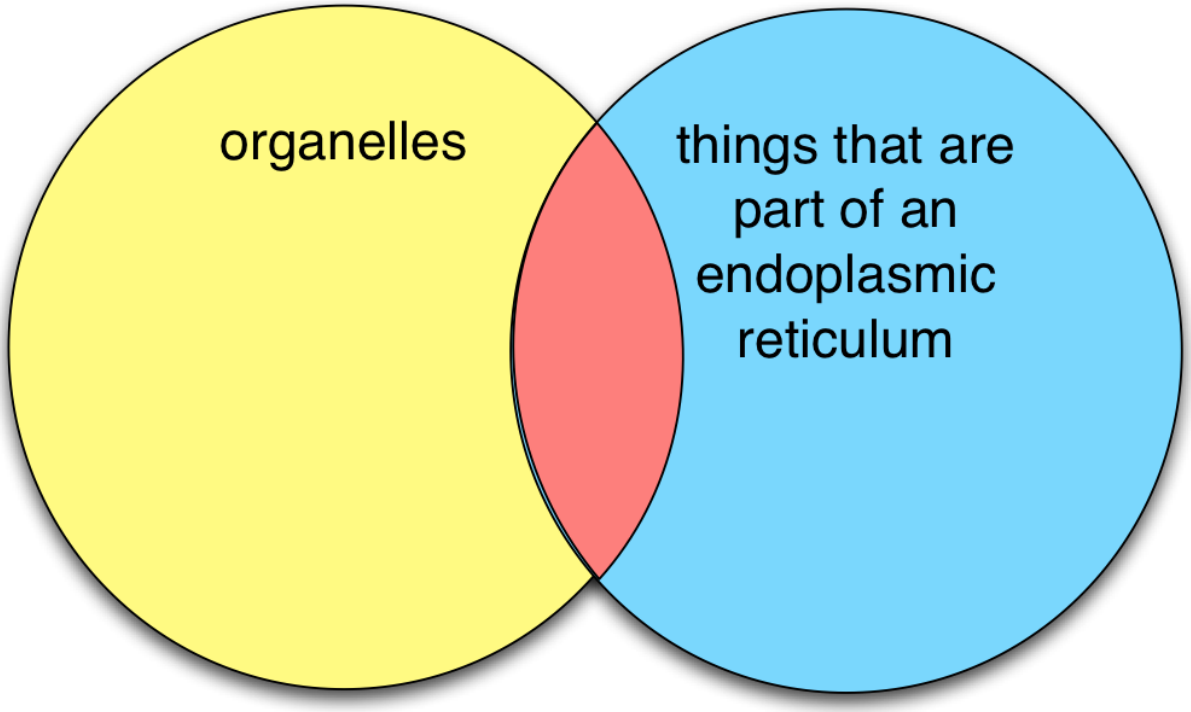
Type “organelle” into the box, and make sure “subclasses” and “direct subclasses” are ticked.

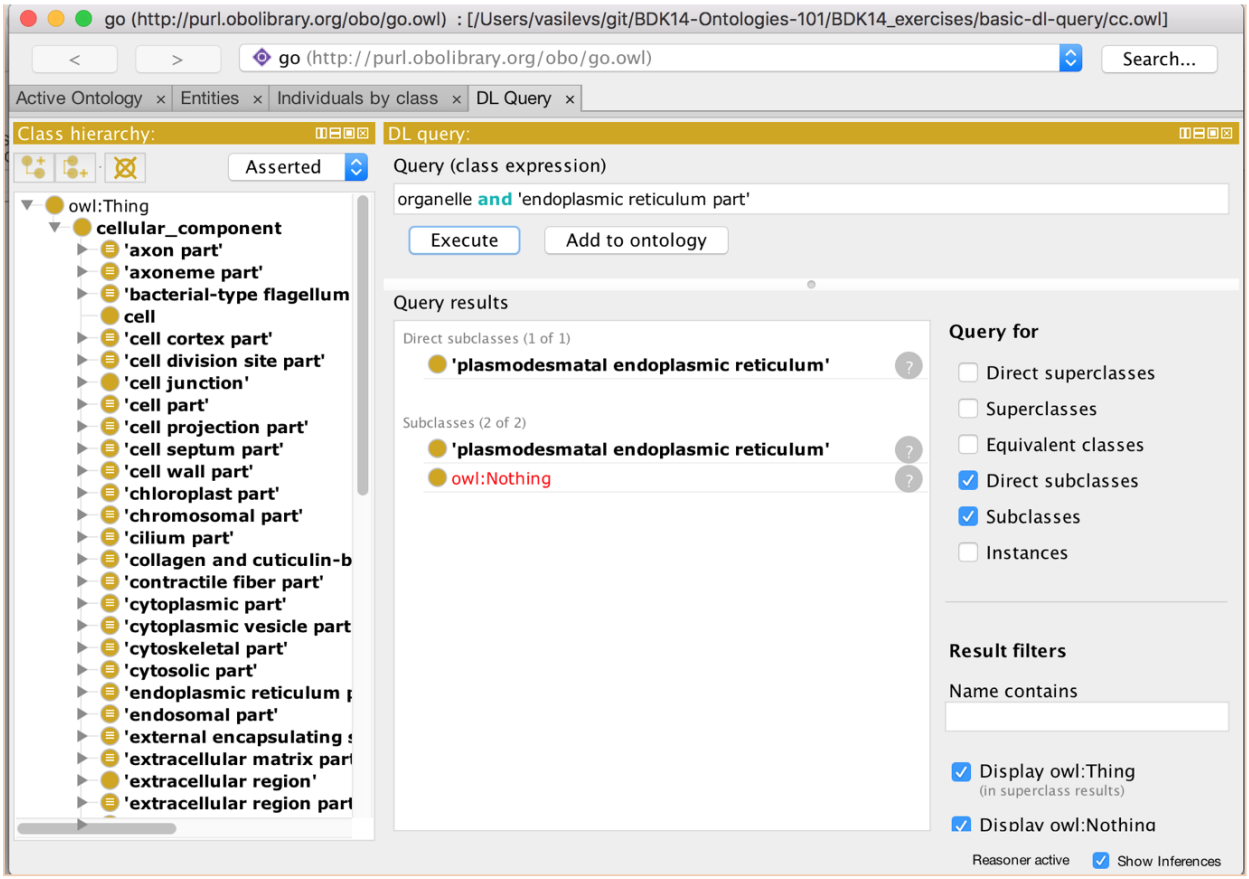


You can type any valid OWL class expression into the DL query tab. For example, to find all classes whose members are part_of a membrane, type “part_of some membrane” and click ‘execute’. Note the linking underscore for this relation in this ontology. Some ontologies do not use underscores for relations, whereby you’d need single quotes (i.e. ‘part of’).



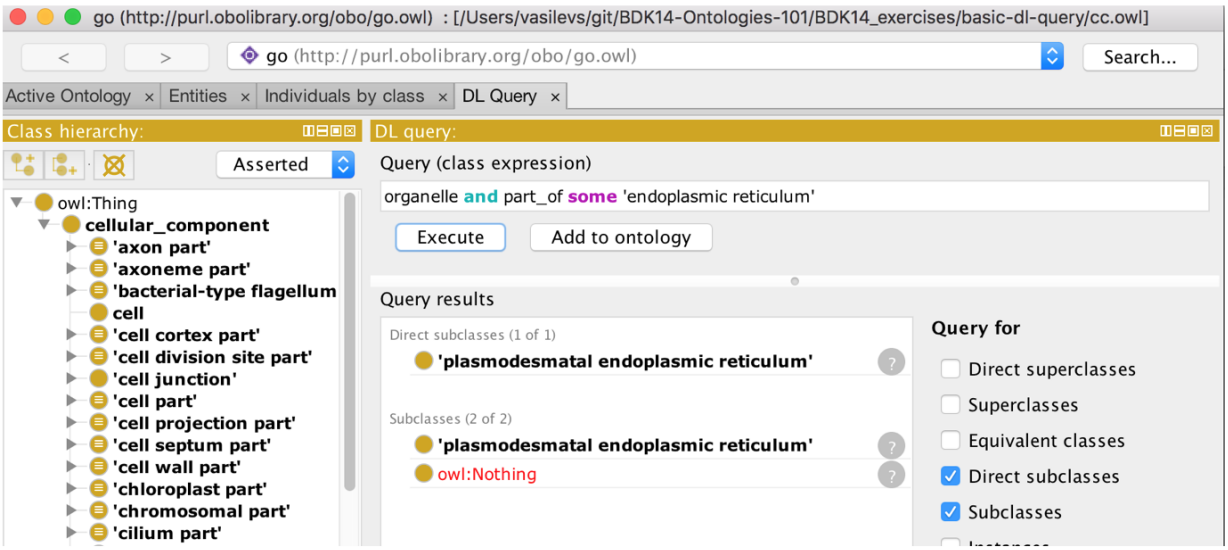
The OWL keyword “and” can be used to make a class expression that is the intersection of two class expressions. For example, to find the classes in the red area below, we want to find subclasses of the intersection of the class ‘organelle’ and the class ‘endoplasmic reticulum part’



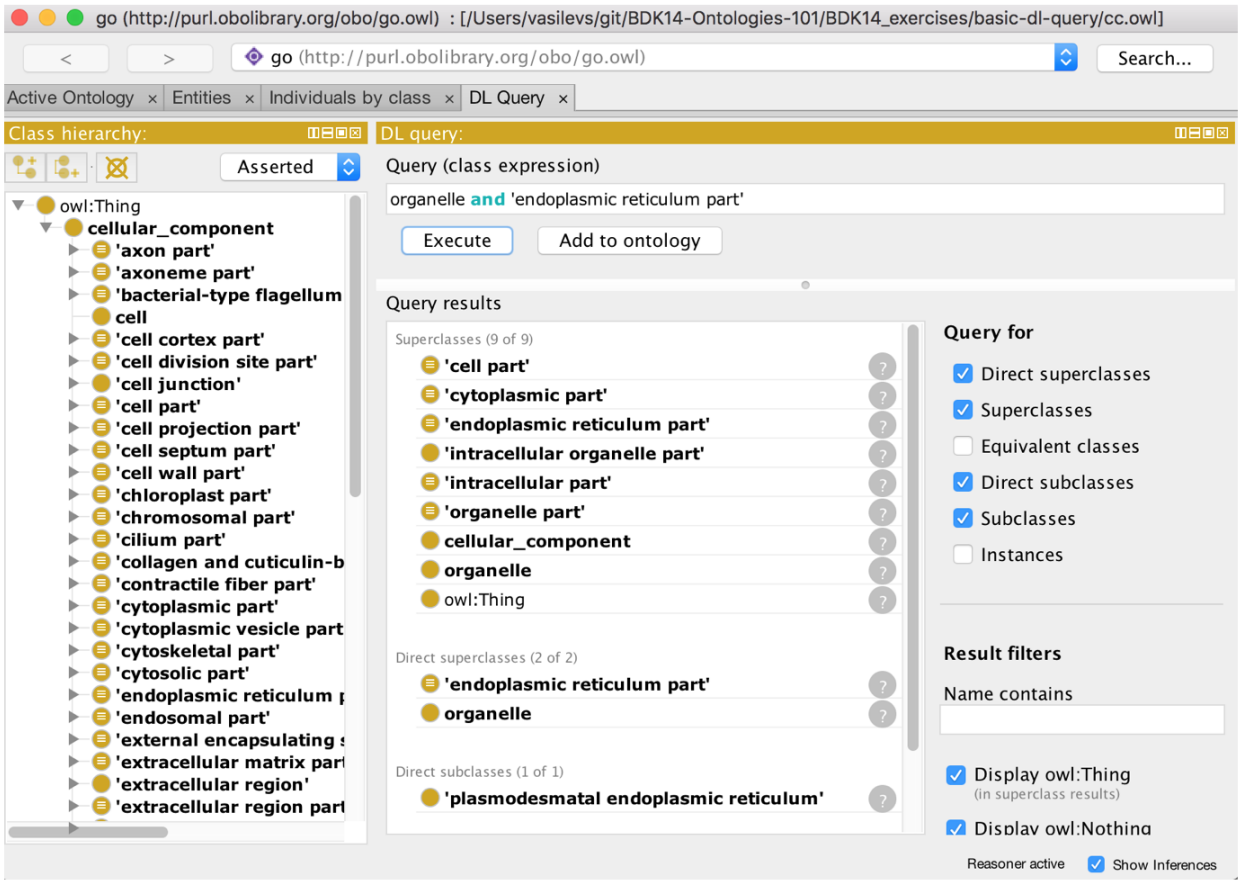


Note that we do not need to use the "part" grouping classes in the gene ontology (GO). The same results can be obtained by querying for the intersection of the class "organelle" and the restriction "part_of some ER" – try this and

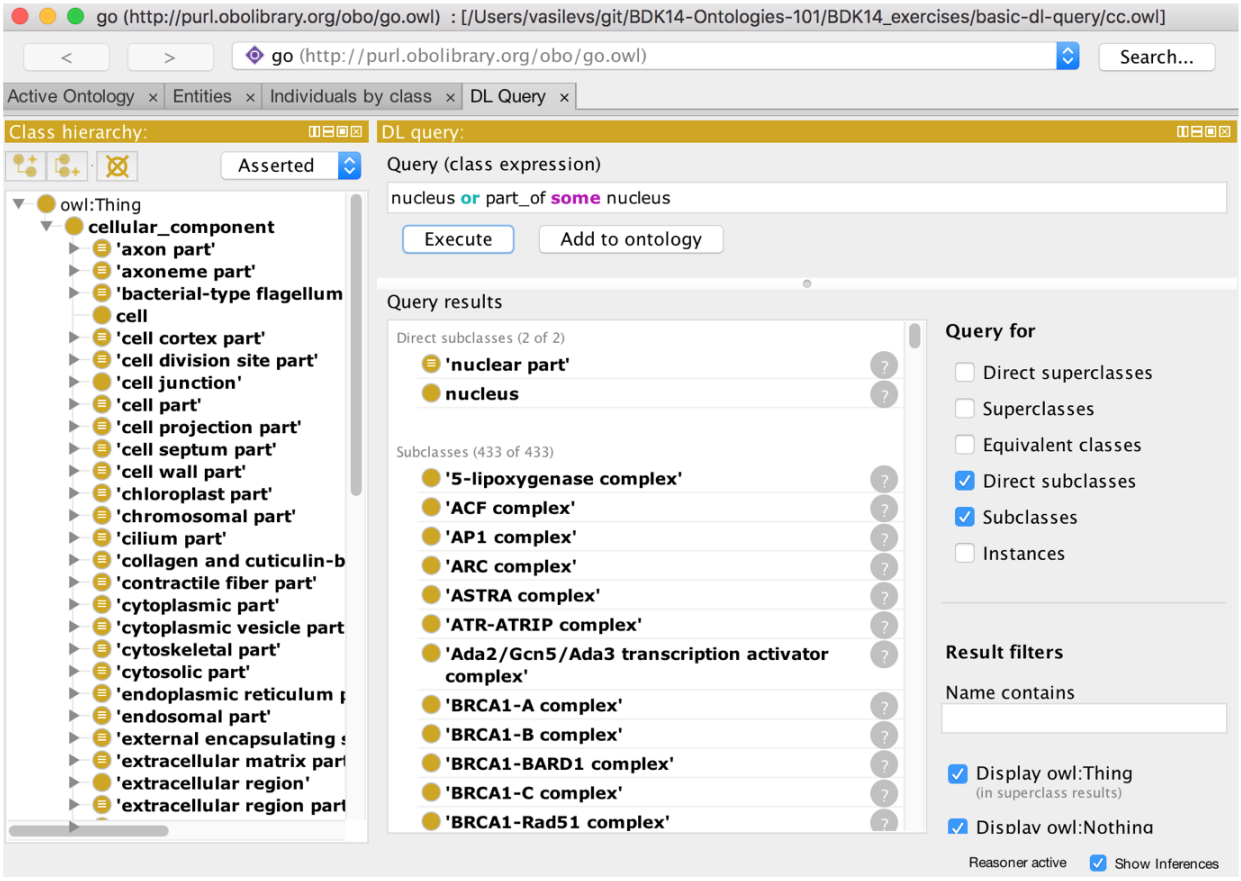
see.



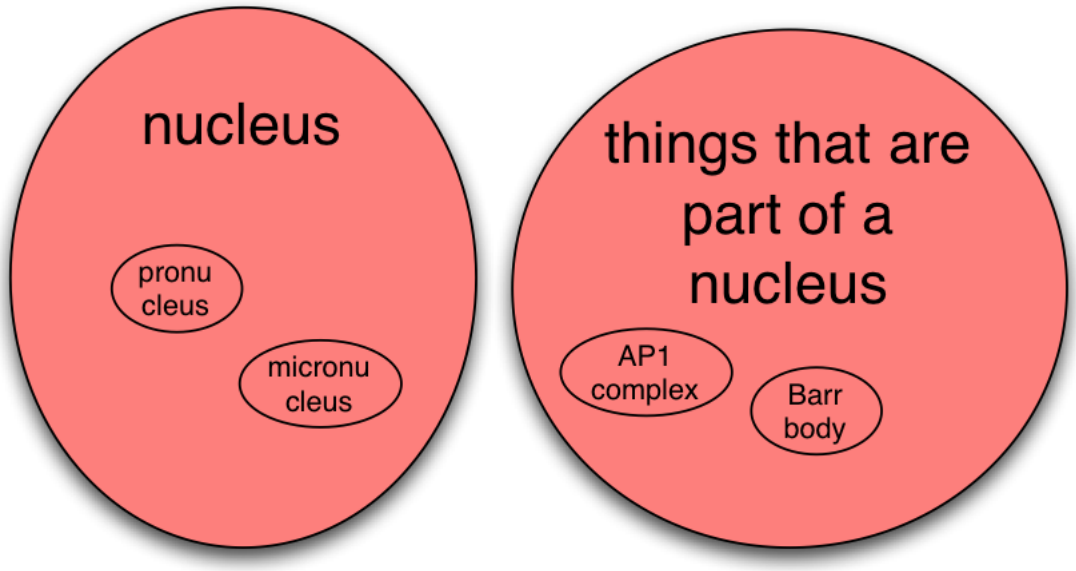
We can also ask for superclasses by ticking the boxes as below:



The ‘or’ keyword is to used to create a class expression that is the union of two class expressions. For example:



This is illustrated by the red area in the following Venn diagram:



CHAPTER 11

EXERCISE: Basic DL Queries

Go to the ‘basic-dl-query’ folder in the Exercises directory and open the file cc.owl.

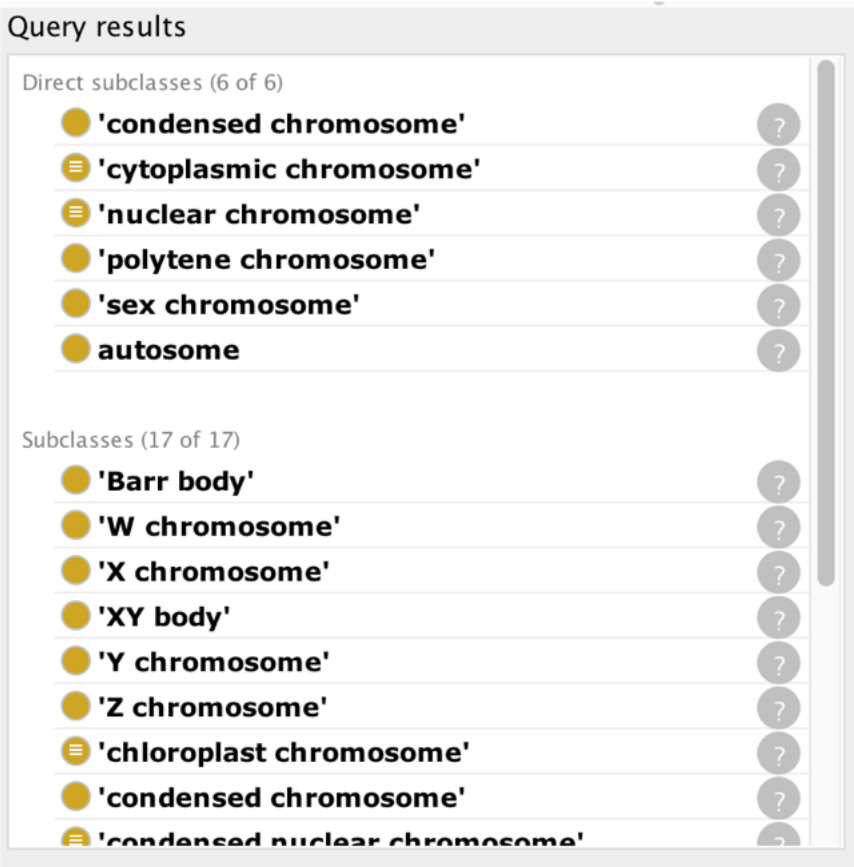
This illustrates basic DL (description logic) queries.

A DL query is a class expression that is constructed using constructs such as ‘and’ (corresponding to set intersection) and ‘some’ (see previous example).

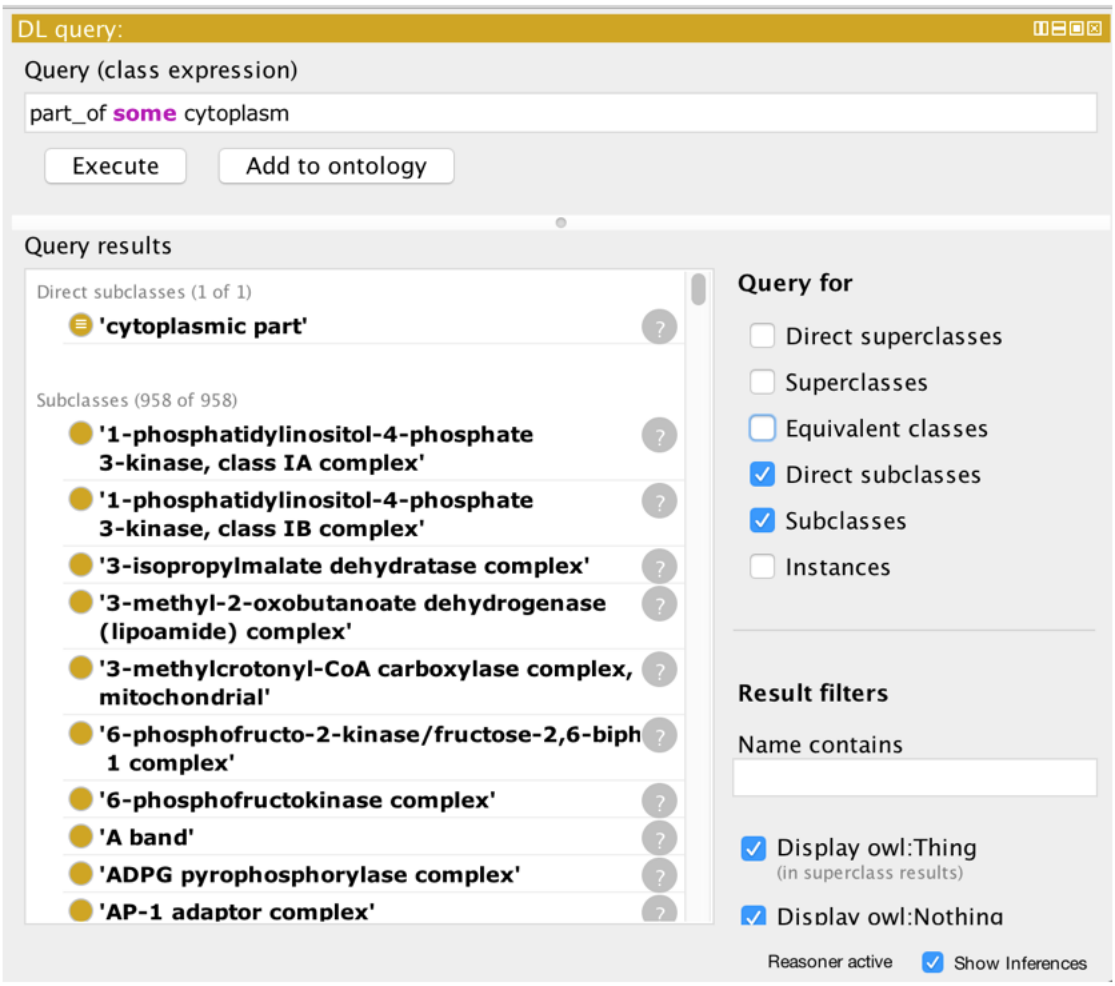
Note that the example that follows this one also revisits DL queries.

Constructs used: ‘and’, ‘some’

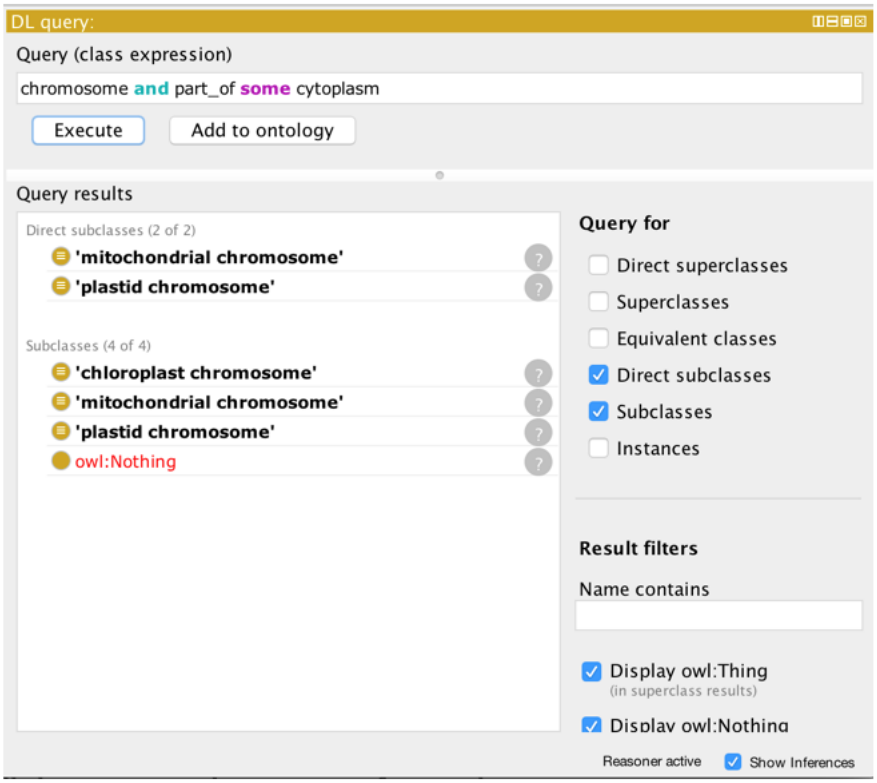
1. open cc.owl
2. Turn on the ELK reasoner
3. Go to DL query tab
4. Find all subtypes of chromosome



1. Experiment with the tab - what do the different checkboxes give you?
2. Find all parts of a cytoplasm

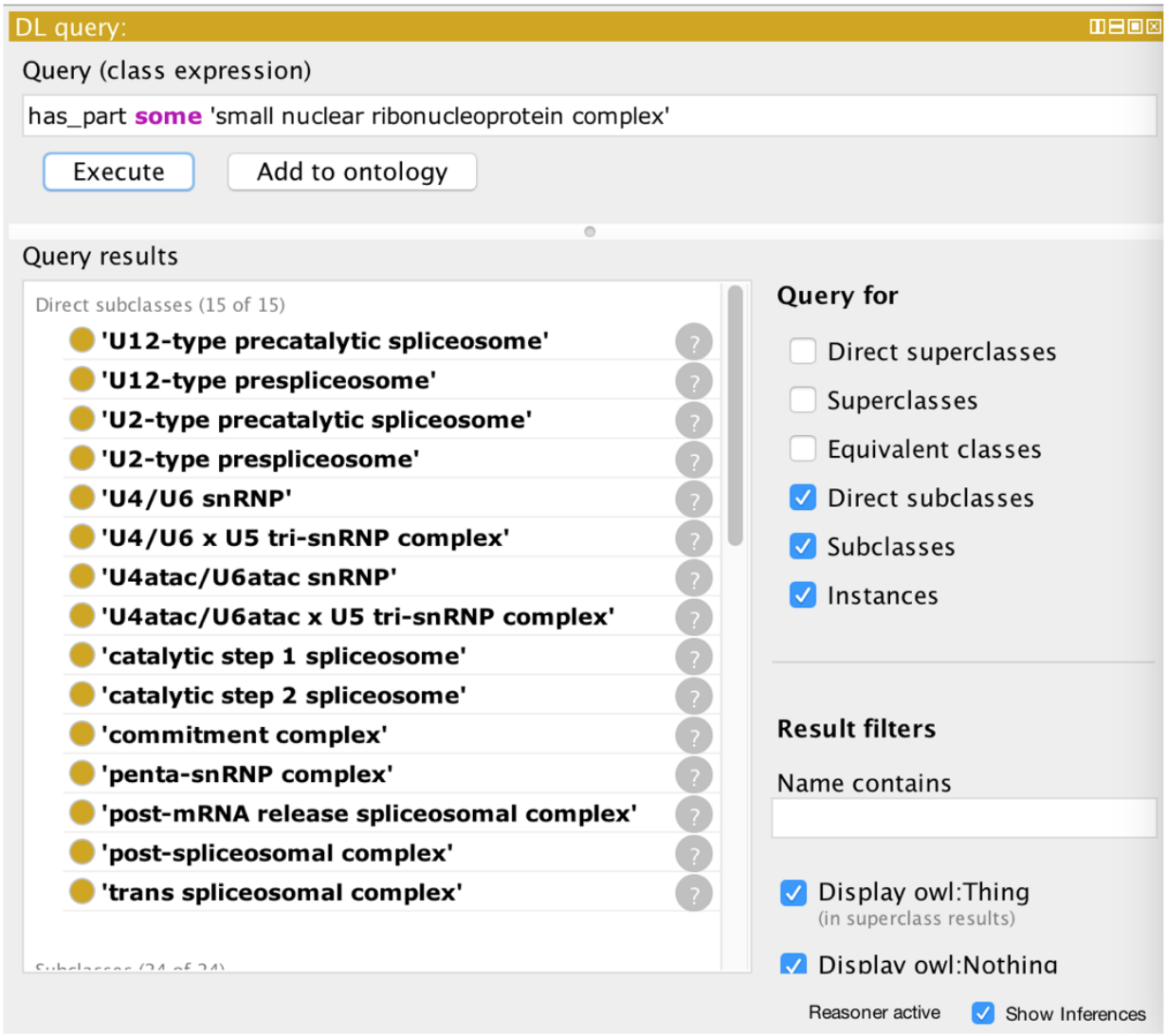


1. Find all chromosomes that are part of a cytoplasm

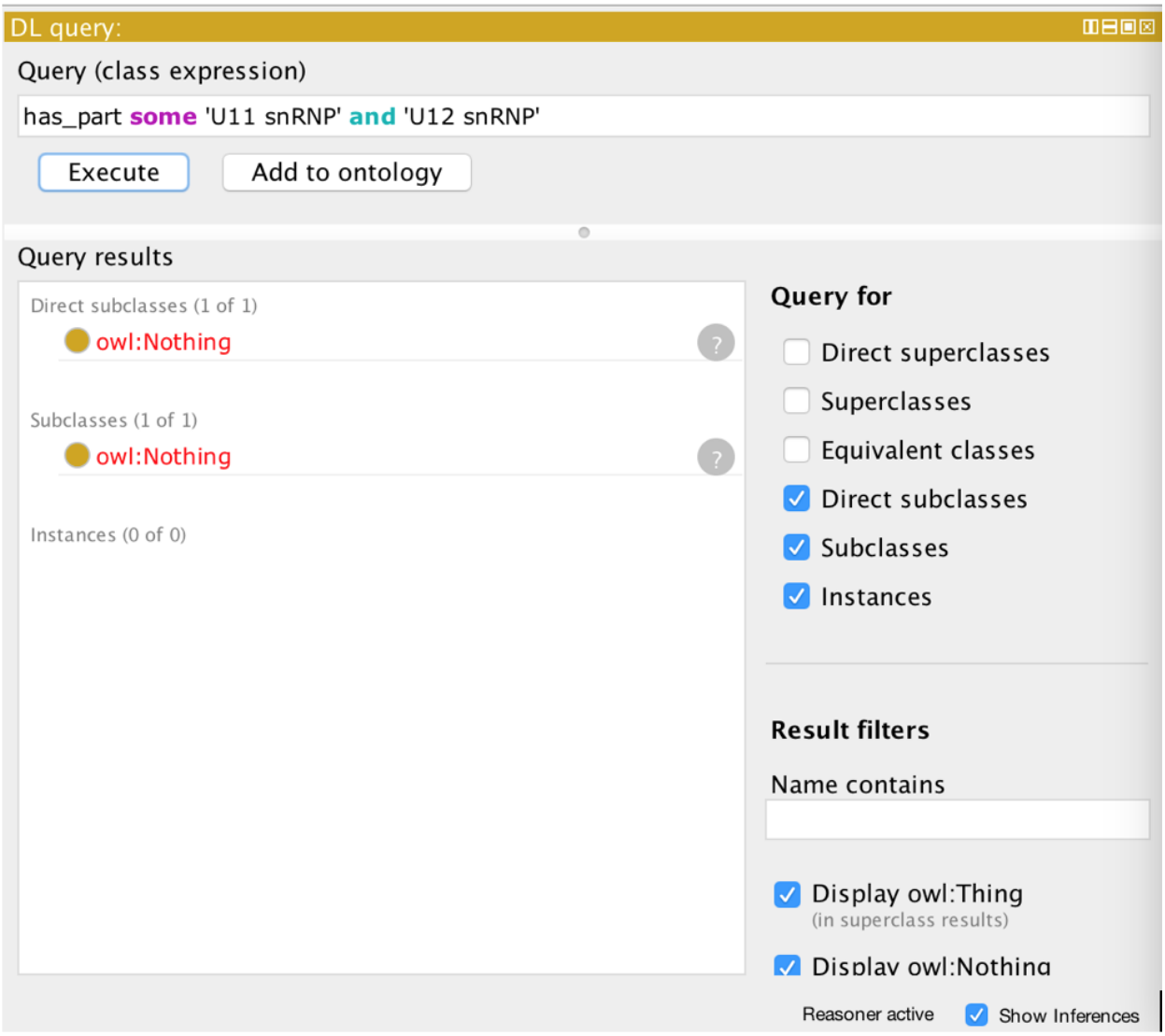


NEXT:

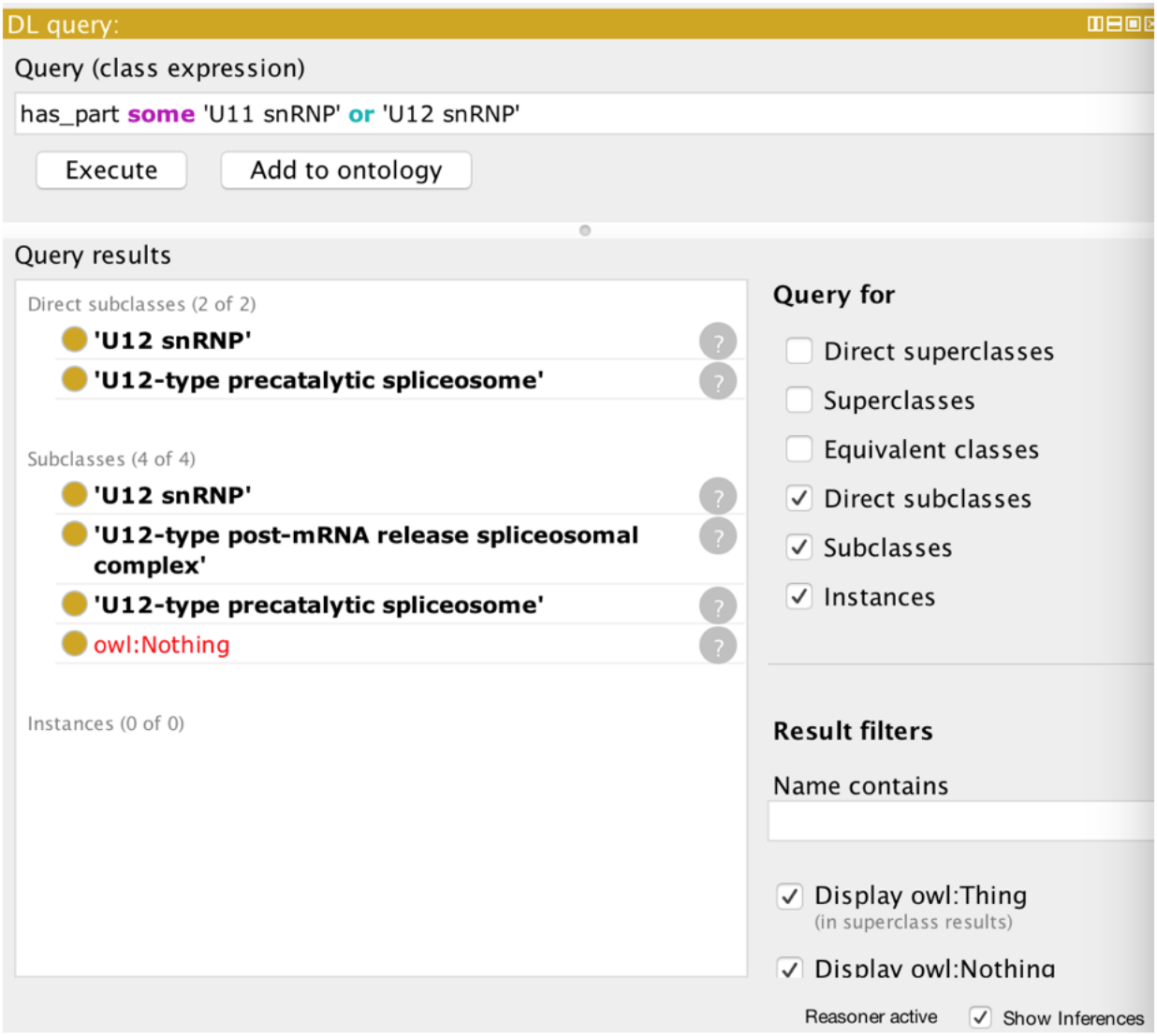
1. Find all classes whose instances have a snRNP ('small nuclear ribonucleoprotein complex') as part



1. Find all classes whose instances have both a U11 snRNP AND a U12 snRNP as parts



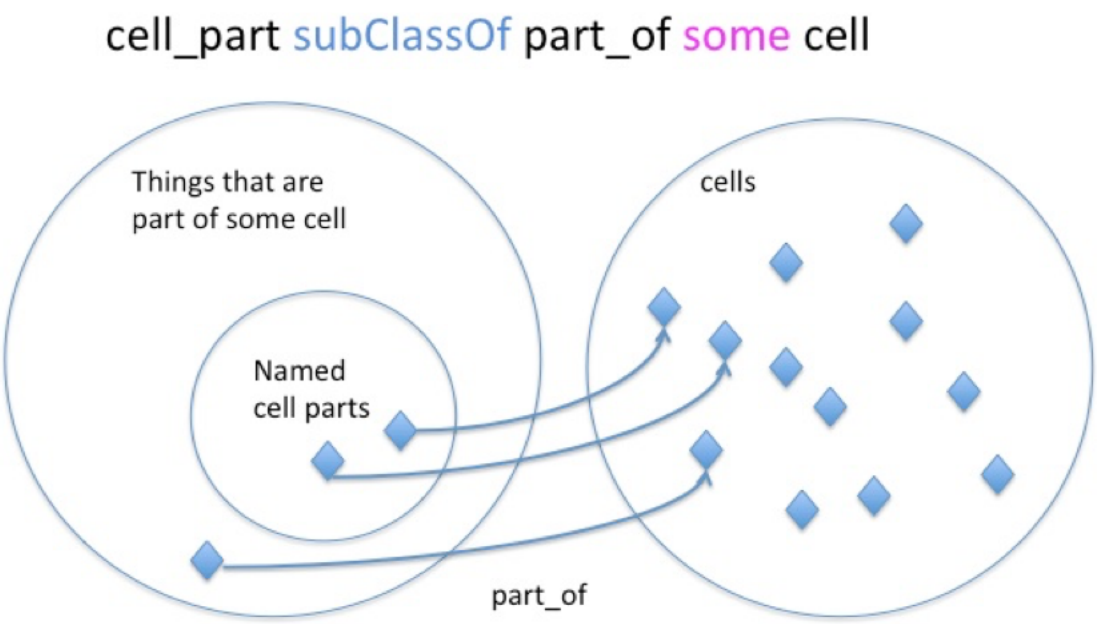
1. Find all classes whose instances have both a U11 snRNP OR U12 snRNP as parts



1. Create a class from this DL query by clicking the 'Add to ontology' button.

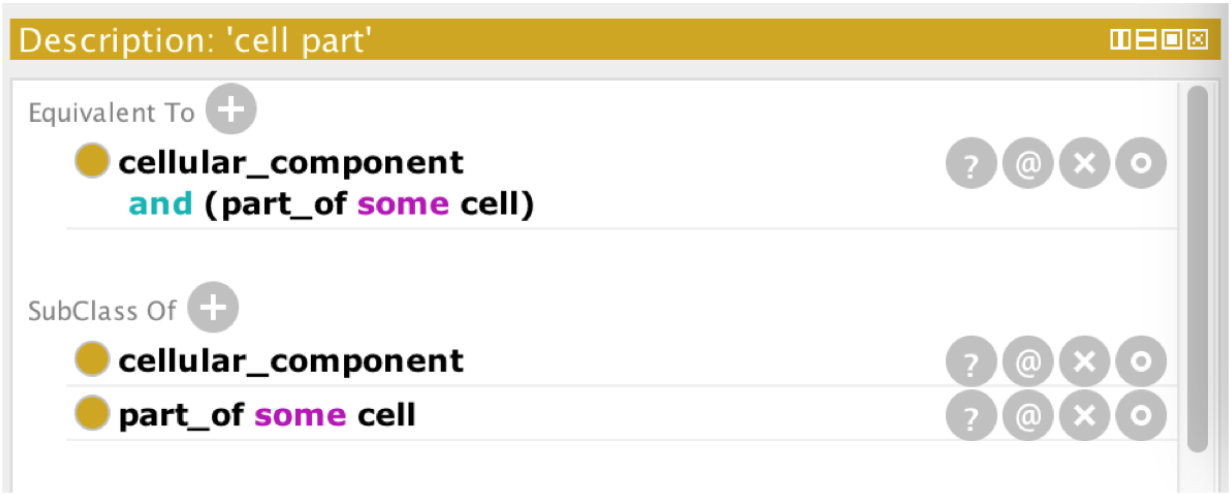
11.1 Equivalent classes

The previous example showed the creation of a class restriction. These restrictions were asserted as superclass restrictions, and are sometimes known as *necessary conditions*. That is, if an individual is a member of the 'cell part' then it is necessary for it to also be related to a 'cell' along the 'part of' property.



Necessary conditions alone mean that individuals can exist that are part of a cell, but are not a type of 'cell part'. In OWL, we can make an even stronger statement and define the 'cell part' class as being equivalent to 'part of' some cell. This is known as a necessary and sufficient condition.

In Protégé we can create an equivalent class restriction inside the 'Equivalent To' slot of the class description view.



CHAPTER 12

Automatic classification

CHAPTER 13

EXERCISE: Basic classification

The file you need is in the basic-classification folder. Follow the instructions below.

This example introduces 'defined classes' and automatic classification. The example involves classification of different ubiquitin ligase complexes. It is based on a subset of the gene ontology (GO) http://purl.obolibrary.org/obo/go.owl with some classes removed for teaching purposes.

Constructs:

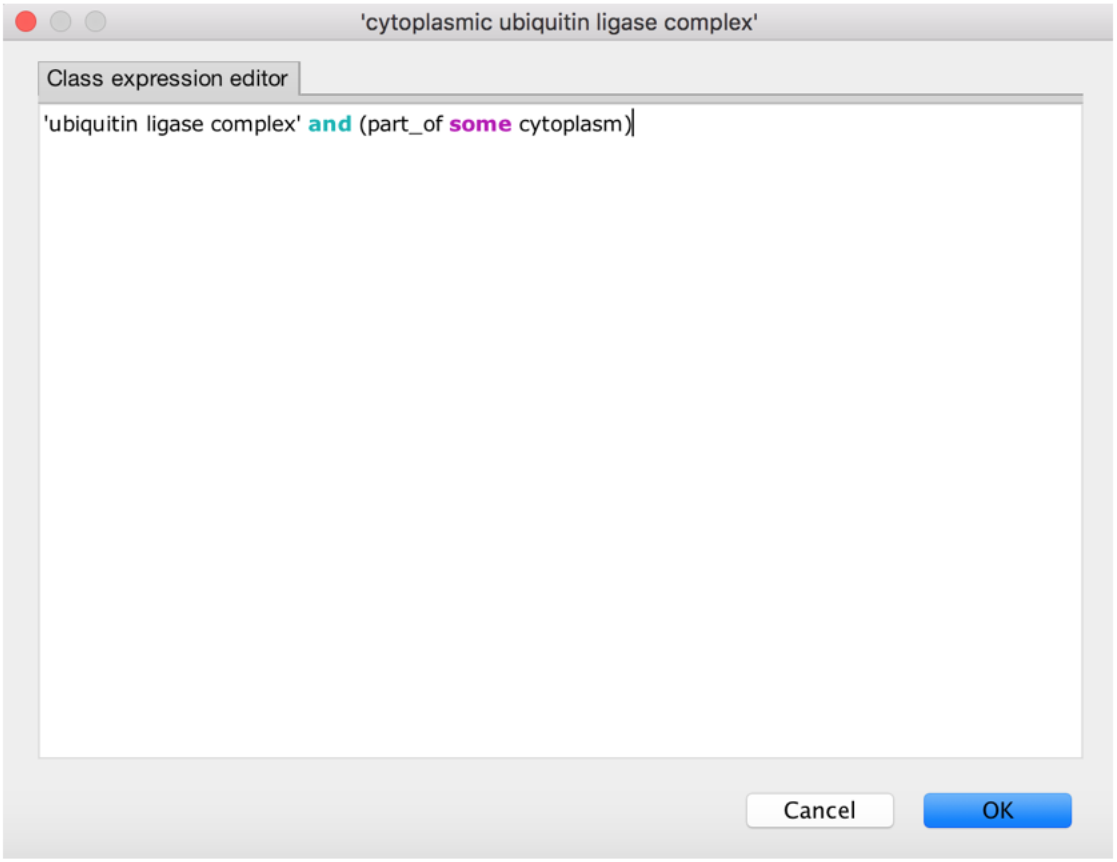
- and (intersection)
- equivalence (logical definitions)
- existential restrictions (e.g. part_of some)

Background knowledge for non-GO people:

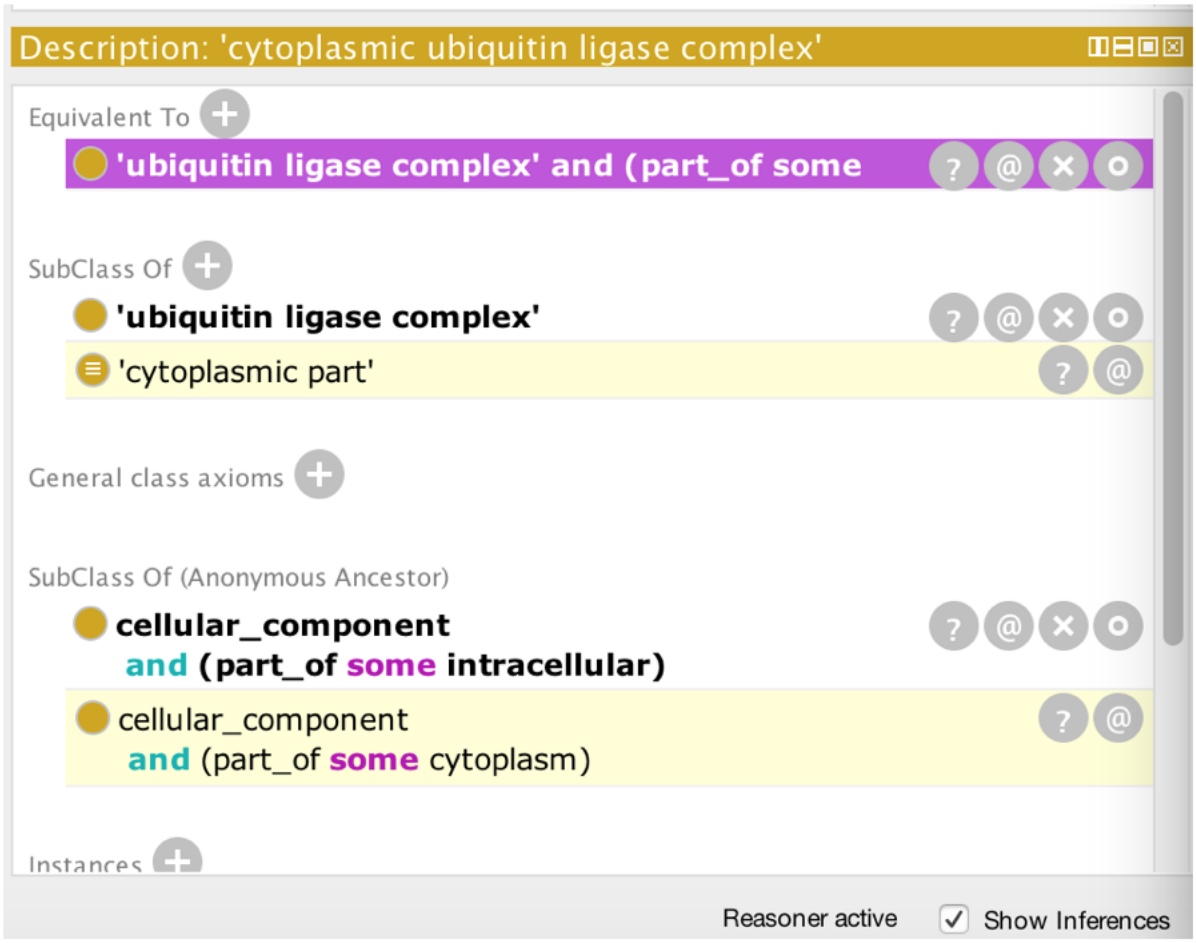
GO includes pre-composed grouping classes such as 'chromosomal part' and 'nuclear part'.

PART 1: Adding classes and automatically classifying them

1. Open basic-classification/ubiq-ligase-complex.owl
2. Navigate to 'ubiquitin ligase complex'
3. Add a subclass of 'ubiquitin ligase complex' called 'cytoplasmic ubiquitin ligase complex'
4. NOTE: do this *directly* under the 'ubiquitin ligase complex' class, don't move things around!
5. NOTE: this class already exists in the main GO, but it has been removed for this tutorial example
6. Give it a logical definition (equivalence axiom)



1. Synchronize the reasoner
2. Find the class you made under 'Class hierarchy (inferred)'
3. You should see 'cytoplasmic part' is now inferred to be a parent class of 'cytoplasmic ubiquitin ligase complex'.

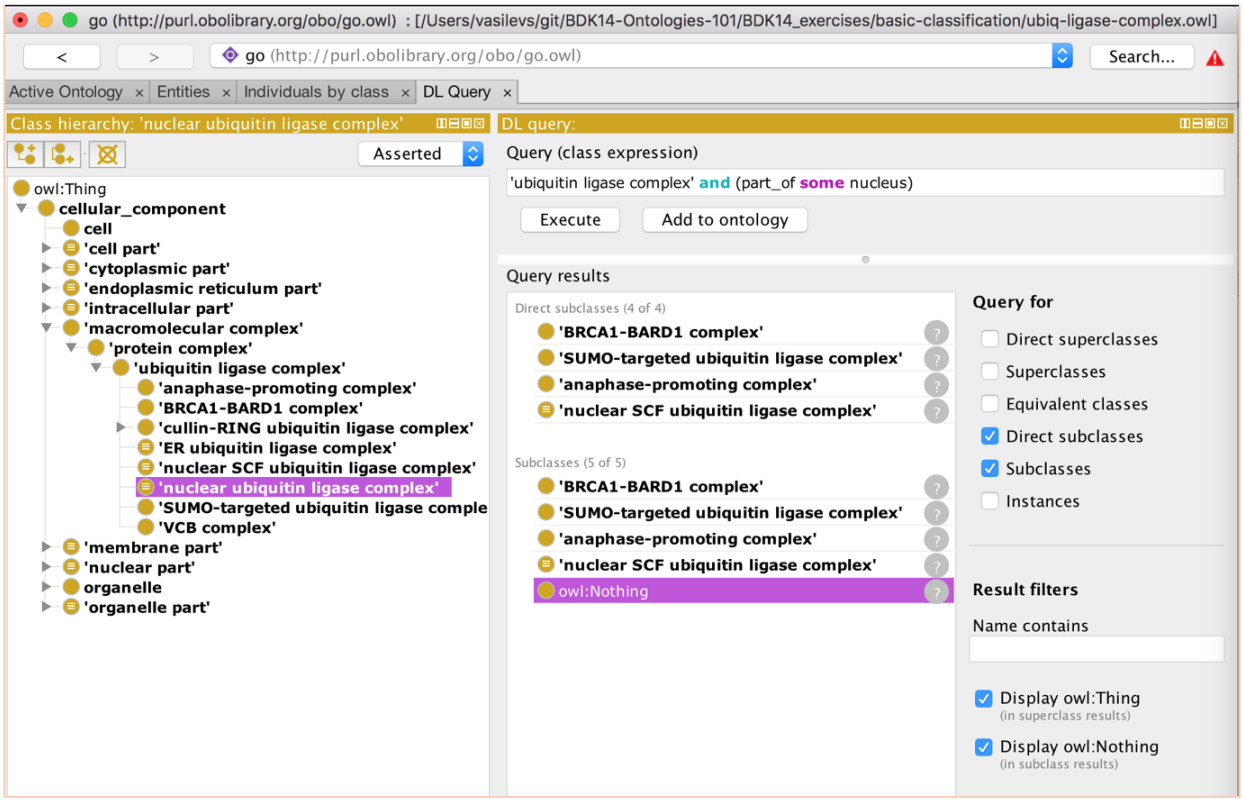


PART 2: Another example

1. Do the same for 'nuclear ubiquitin ligase complex' – create the class and add the equivalence axiom.
2. Synchronize the reasoner.
3. You should see 'nuclear ubiquitin ligase complex' is inferred to be a child of 'nuclear part'.

BONUS:

1. Remove the classes you have created.
2. Find all 'ubiquitin ligase complex' classes whose instances are in a nucleus in the DL query tab
3. Make the class directly from here


go (http://purl.obolibrary.org/obo/go.owl) : [/Users/vasilevs/git/BDK14-Ontologies-101/BDK14_exercises/basic-classification/ubiq-ligase-complex.owl]
go (http://purl.obolibrary.org/obo/go.owl)
Search...
Active Ontology
Entities
Individuals by class
DL Query
Class hierarchy: 'nuclear ubiquitin ligase complex'
Asserted
owl:Thing
cellular_component
cell
'cell part'
'cytoplasmic part'
'endoplasmic reticulum part'
'intracellular part'
'macromolecular complex'
'protein complex'
'ubiquitin ligase complex'
'anaphase-promoting complex'
'BRCA1-BARD1 complex'
'cullin-RING ubiquitin ligase complex'
'ER ubiquitin ligase complex'
'nuclear SCF ubiquitin ligase complex'
'nuclear ubiquitin ligase complex'
'SUMO-targeted ubiquitin ligase comple
'VCB complex'
'membrane part'
'nuclear part'
organelle
'organelle part'
DL query:
Query (class expression)
'ubiquitin ligase complex' and (part_of some nucleus)
Execute
Add to ontology
Query results
Direct subclasses (4 of 4)
'BRCA1-BARD1 complex'
'SUMO-targeted ubiquitin ligase complex'
'anaphase-promoting complex'
'nuclear SCF ubiquitin ligase complex'
Subclasses (5 of 5)
'BRCA1-BARD1 complex'
'SUMO-targeted ubiquitin ligase complex'
'anaphase-promoting complex'
'nuclear SCF ubiquitin ligase complex'
owl:Nothing
Query for
Direct superclasses
Superclasses
Equivalent classes
Direct subclasses
Subclasses
Instances
Result filters
Name contains
Display owl:Thing
(in superclass results)
Display owl:Nothing
(in subclass results)

CHAPTER 14

EXERCISE: More basic classification

Go to the taxon-union folder and follow the instructions below. This introduces classification using ‘or’ and ‘not’.

This example extends the previous one introducing ‘or’ (UnionOf) and ‘not’ (complementOf)

Instructions:

1. Open taxslim-with-union.owl
2. Briefly check the asserted hierarchy - this is a subset of the NCBI taxonomy
3. Examine the ‘union’ classes at the top of the class hierarchy. In particular:
4. ‘Nematoda or Protostomia’ (note that NCBI classifies nematodes as pseudocoelomata)
5. ‘Viridiplantae or Bacteria’
6. Select ELK reasoner and start reasoner (ignore the pop-up windows).
7. Navigate to the inferred hierarchy. How have the union terms we examined before been placed? * Note that you’ll need to know a little taxonomy here to translate the common names into scientific names (Wikipedia is your friend).
8. Create your own grouping classes and classify them. Some examples:
9. Mouse or Human
10. Mouse or Primate
11. Pescetarian dietary component (plant or fish)
12. Pescetarian dietary component, more relaxed variant (plant or fish or fungi or mollusc or arthropod)
13. Note: don’t manually place these in the hierarchy, let the reasoner do this. How can you quickly tell where this class will be placed using the DL query pane?
14. Tip: use the DL query tab to test your class expression first
15. Create a non-sensical ‘transgenic hybrid’ class, such as a fly-human - which is both a Drosophila and a human. What happens to this when you classify?

16. Try using the explanations feature (the '?'). Hint: the explanation is more compact if you choose two sibling taxa - e.g. Deuterostome and Protostome
17. Remove the hybrid class before moving on.
18. Use the DL query tab to find all mammals that are not humans
19. Try creating one or more of the following paraphyletic classes. This will involve the 'not' construct
20. Nonhuman primate (How can you quickly find the label used for 'Primates' if you know that human = 'Homo sapiens'?)
21. Invertebrate
22. Invertebrate chordate
23. Reptilia, as traditionally defined: (amniote minus aves and mammals)
24. A Land mammal
25. Classify your classes. What does superclass of the classified class represent? Discuss with your instructors- does this reasoned classification reflect an evolutionary history?!
26. See HINTS.txt or talk to an instructor if you get stuck on how to classify things.

CHAPTER 15

Object Properties

CHAPTER 16

EXERCISE: Domains and Ranges

Got to the domain-range folder and follow the instructions below. This introduces the concepts of ‘domains’ and ‘ranges’ on object properties.

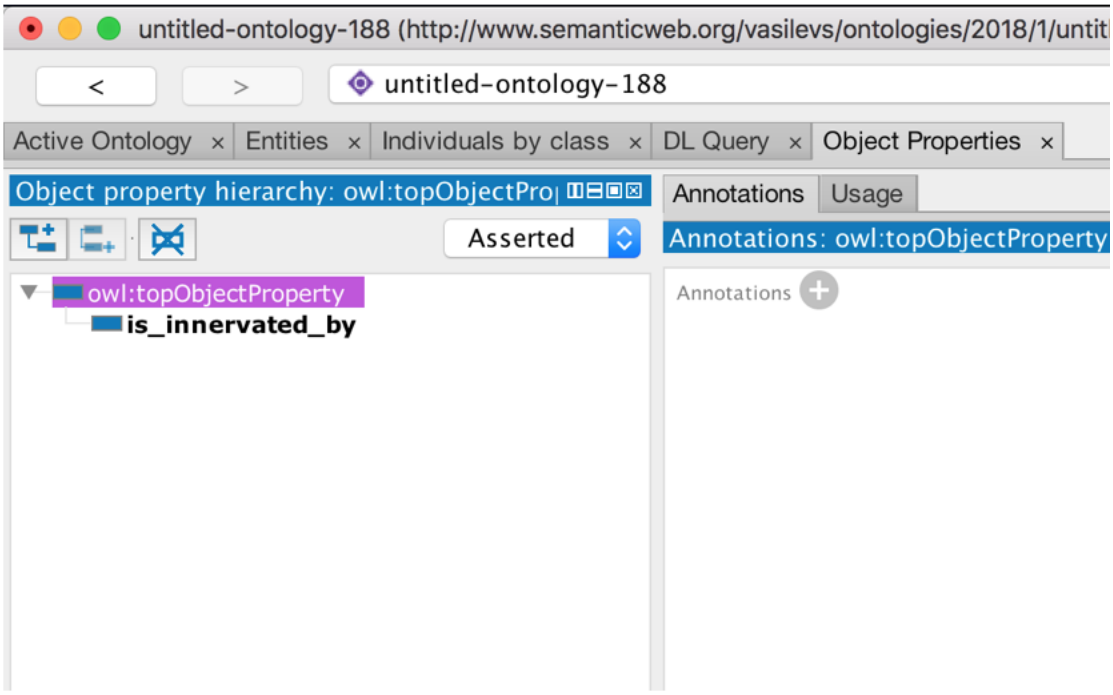
In this exercise, we will illustrate how object properties can be used to subclassify classes that are restricted by them.

1. Create a new ontology.
2. Add the following class hierarchy:

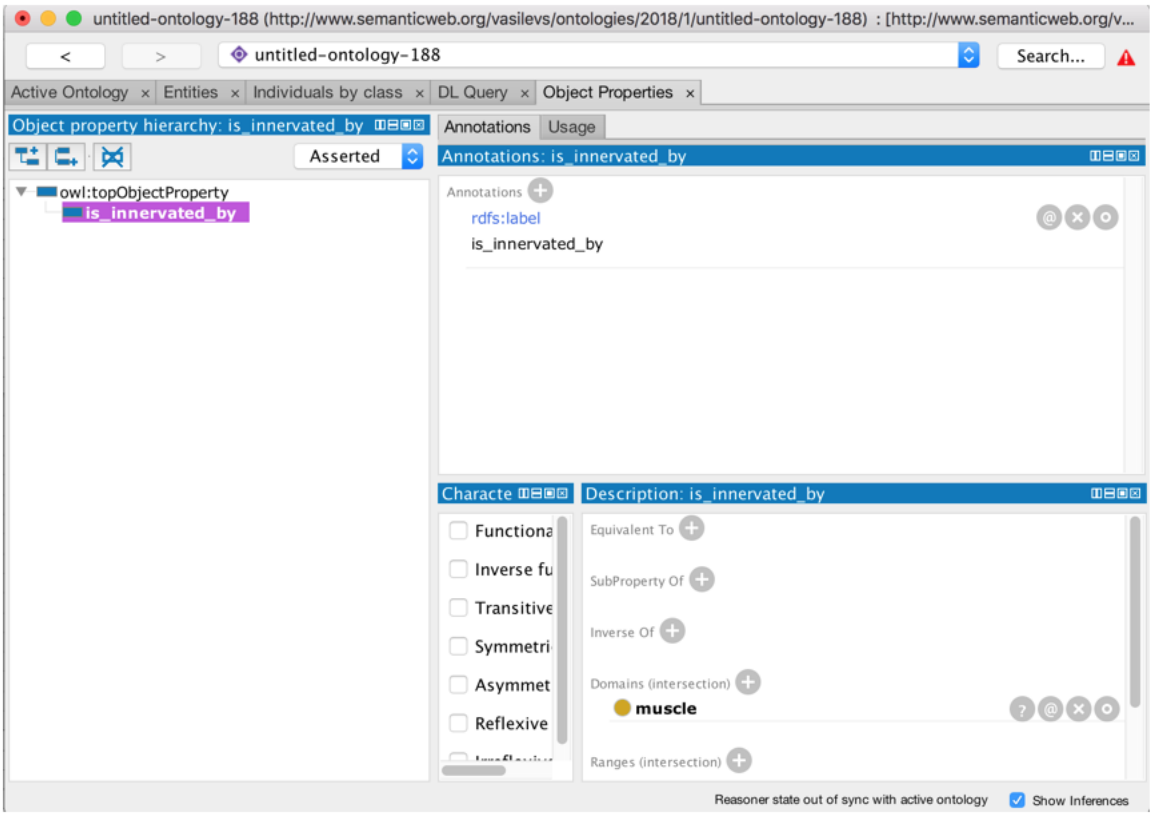
depressor mandibulae muscle leveator quadrati muscle nerve facial nerve trigeminal nerve

1. Add the following:

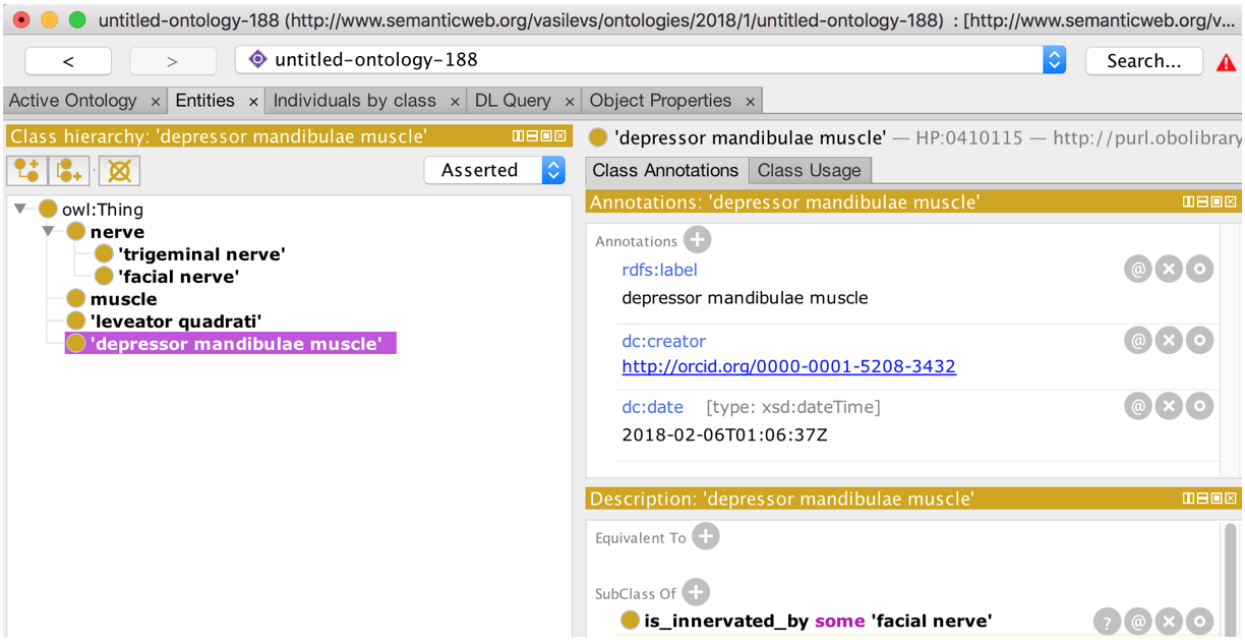
- An object property named ‘is_innervated_by’



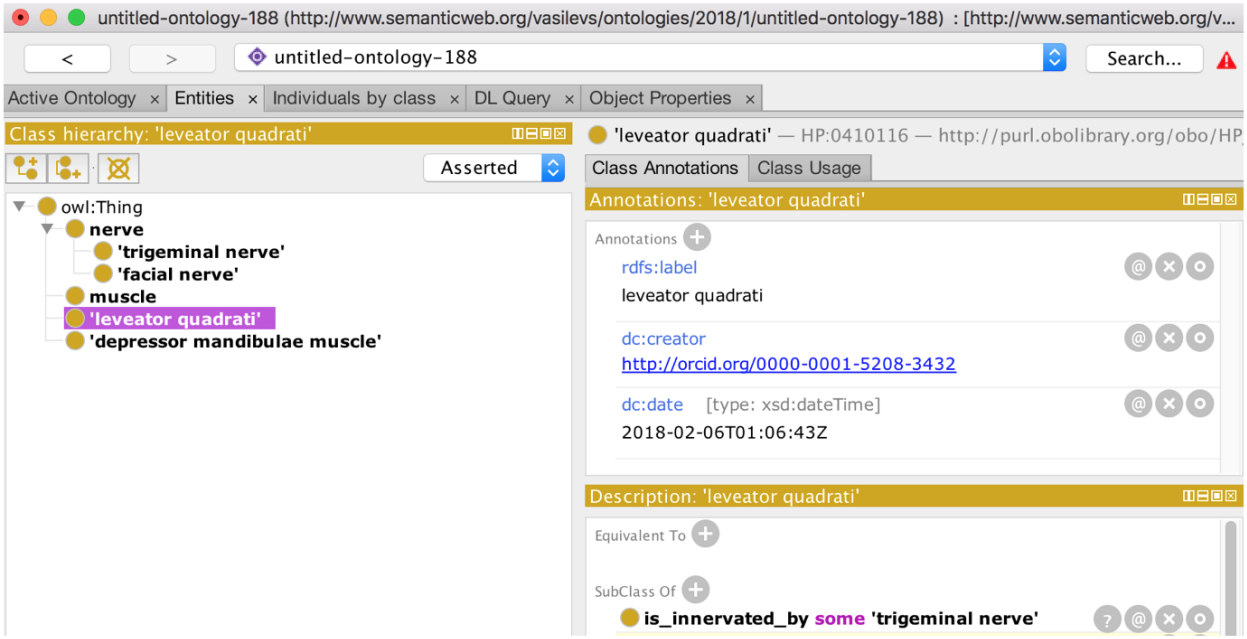
- Add a domain on ‘is_innervated_by’ - ‘muscle’



- To ‘depressor mandibulae muscle’ a subclass restriction ‘is innervated_by some ‘facial nerve”



- To ‘leveator quadrati’ a subclass restriction ‘is innervated_by some ‘trigeminal nerve’



1. Now run the reasoner and inspect the inferred class hierarchy. You should see a new classification under the 'muscle' class.