Workbook to:
An Introduction to Ontology Engineering

version 1 of the workbook

C. Maria Keet

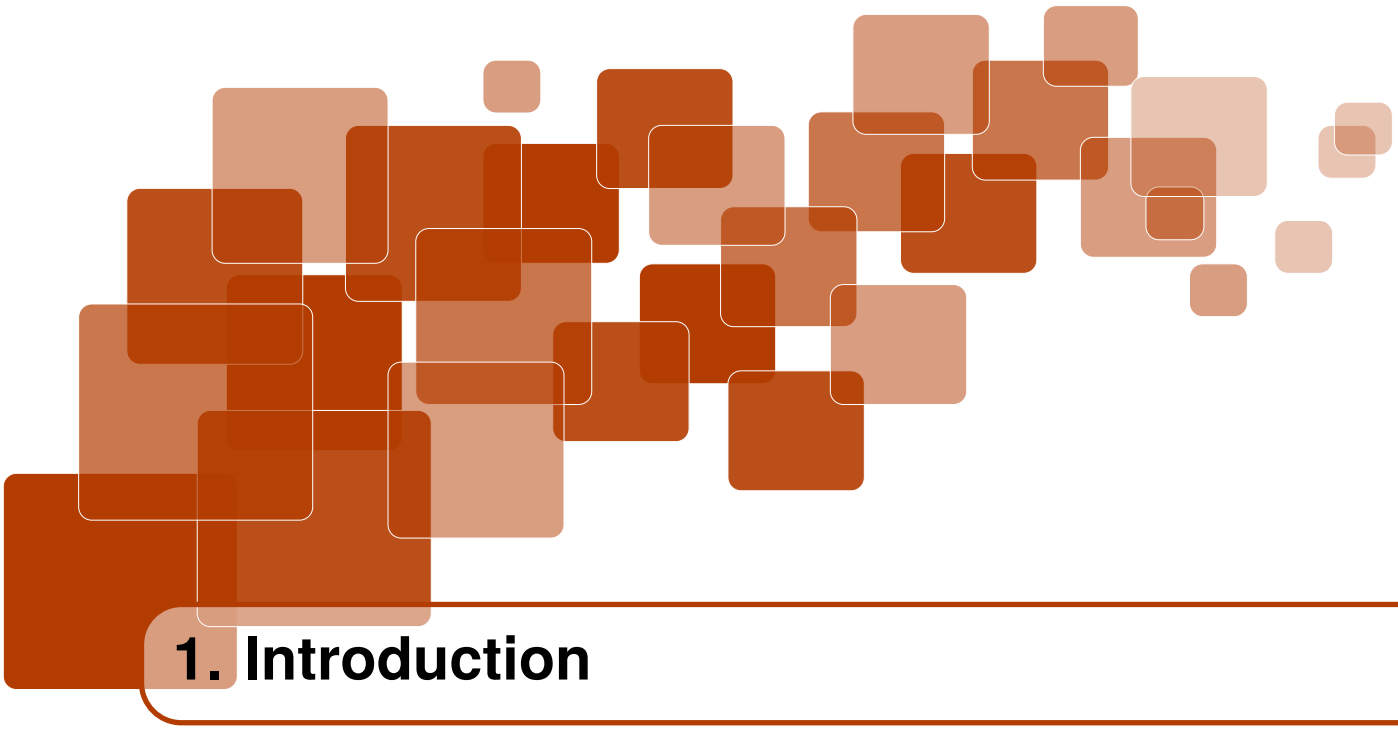
with Zola Mahlaza

Published by making it available

Location: https://people.cs.uct.ac.za/~mkeet/OEbook/
Alternate location: http://www.meteck.org/teaching/OEbook/

First public release, June 2025

Contents



1. Introduction

This workbook accompanies version 2 of the textbook *An Introduction to Ontology Engineering*. It contains tutorials, sample assignment descriptions, and answers to selected questions from the textbook.

Part 1 can be used for tutorials, with one chapter per 1.5-3 hour tutorial. All tutorials require hands-on activities, from developing an ontology in various ways to developing a tool to process ontologies. The start of each tutorial describes what is assumed to have been covered from the textbook or another source that covers the same topics and it roughly follows the order of topics in the textbook. Care has been put into the tutorials to ensure it all works. By the time you read this, it may not. Generally, that may happen with the software or be due to link rot. A lot of software is Java-based thanks to the OWL API, but there are more frequent Java updates than tool developers keep up with, and the newer Java versions are not always backward compatible. If a tool does not work, check the system requirements and downgrade the JRE if needed.

Part 2 consists of two assignments, or assignment-style exercises: developing a domain ontology from scratch and carrying out a mini-project where the topics include literature reviews, experiments, and software development to choose from. These tasks are best carried out in small groups of 2-3 people.

Part 3 contains answers to selected review questions and the exercises from the textbook, thereby reducing the number of pages to print of the textbook, and therewith reducing the book's cost.

The appendix consists of introductory notes about Protégé (Appendix A) and instructions for how to cite papers and how to manage citations (Appendix B). The former is included because not all tutorials may have teaching assistants available to guide you through the interface. The latter is included to helps writing a report for the assignments and cite scientific literature properly. My 4th-year/honours level students found the guidance helpful, and seeing the state of some of the papers I've been reviewing, there are more authors who could benefit from it.

Part One: Integrative exercises

2. Explore and build an ontology step by step

The aim of this lab is to introduce you to creating, 'debugging', and exploring improving an ontology about electric bicycles. It focuses on the following tasks:

1. Adding and editing knowledge to an ontology;
2. Familiarising oneself with the ontology language features and the automated reasoner;
3. Finding the root cause of mistakes, which may be both logical ones and undesirable deductions, and correcting them;
4. Exploring methods and tools to improve the representation of the knowledge.

No prior ontology development experience is necessary, though helpful. The difficulty level of the exercises in Sections 2.2 and 2.3 increases quickly and, depending on your knowledge and skills, may take 1.5-3h to complete. It is possible, though not optimal, to jump straight to the exercises in Section 2.4.

If you use the accompanying textbook, then the exercises in Sections 2.2 and 2.3 are suitable in conjunction with the theory in textbook Chapter 4, exercise 1 in Section 2.4 with Chapter 5, and exercise 2 in Section 2.4 with Chapter 6.

2.1 Preliminaries

Install your ontology development environment of choice, if not already done so, and acquaint yourself with the software. Look at the menu options and editing panels. If you installed Protégé, you also can add new tabs and views (Click 'Window' - 'View' or 'Window' - 'Tabs'; for views, the mouse pointer will change and you'll need to click to place that view somewhere on the open tab to be able to see it). *If you are new to Ontology Development Environments, then consult Appendix A first*; it contains a few Protégé screenshots and explanations about what to find where to get started.

2.2 Adding knowledge to an ontology

1. The following tasks aim to assist with exploring adding content to a new 'test ontology' that uses different language features and therewith different OWL species.
   (a) Create a new ontology, give it a name, and save it in RDF/XML.
   (b) Add the knowledge that bicycles are vehicles; i.e., create two classes, bicycle and vehicle, and make the former a subclass of the latter.
   (c) Add bicycle $\sqsubseteq \exists$hasComponent.Wheel, or: that each bicycle has 'at least one' ('some' in Protégé) wheel as component.
   (d) Save the file in RDF/XML format (the official exchange format), and also each time before you run the reasoner (if you're using Protégé, as newer versions might freeze at times).
   (e) Download the OWL classifier from `https://github.com/muhummadPatel/OWL_Classifier`, open it, open your ontology ('File' - 'load..') and inspect the OWL species. (note: this may or may not work fully, depending on whether you have installed Java and can run `.jar` files and if so, which version.)
   (f) Add Bicycle $\sqsubseteq \geq 2$ hasComponent.$\top$. (note: the $\top$ in Description Logics (DL) notation is the same as `owl:Thing` in OWL)
   (g) Reload your ontology in the OWL classifier and inspect the OWL species.
   (h) Update the previous axioms with the following one: Bicycle $\sqsubseteq = 2$ hasComponent.Wheel. (note: the $=$ in DL is the same as `exactly` in Protégé).
   (i) Reload the ontology in the OWL classifier and inspect the OWL species and violations. What is the main difference, if any?
   (j) Declare hasComponent to be transitive. Save your ontology, and either run the reasoner ('Reasoner' - 'start reasoner') or check the file again with the OWL Classifier and try to explain the tool's behaviour/output. How is this different from the previous outputs?
2. The following tasks are intended to experiment with adding content to the 'test ontology' and to try out automated reasoning. If you're using Protégé: it includes an 'explanation' feature that lists the axioms involved in deducing what it deduced, which shows up highlighted in yellow, and will help pinpointing errors later on as well, which can be accessed by clicking on the '?' element.
   (a) First, uncheck the 'transitive' checkbox on hasComponent so that we can use the DL reasoner again.
   (b) Add Cycle $\equiv \exists$hasComponent.Wheel. Run the reasoner and observe the deductions. Can you explain what was deduced and why? (note: if you have used the reasoner before in Protégé in this session, then click 'synchronize reasoner'.)
   (c) Declare vehicle to be the domain of hasComponent.
   (d) Add the class motorised vehicle as a subclass of vehicle and declare it disjoint from bicycle.
   (e) Create a new class electric bicycle and declare it to be a subclass of both bicycle and motorised vehicle. Run the reasoner and try to explain the deductions in your own words.
   (f) Add Wheel $\sqsubseteq \exists$hasComponent.Spoke. Run the reasoner and try to explain the deductions, i.e., describe why it deduced what it did.

2.3 Finding and correcting mistakes

1. In this exercise, we're going to fix the 'mess' we created in the previous exercise in such a way that the ontology is within OWL 2 DL expressivity, is not incoherent (i.e., has no unsatisfiable classes), and has no undesirable deductions.[1]
   (a) Let's first fix the unsatisfiable class, electric bicycle. What is the root cause of it being unsatisfiable? Consider:
      - Which axiom(s) is (are) the culprit(s)? Several valid arguments are possible.
      - We know that electric bicycles do exist (and your lab facilitator rides one, even!), so they deserve to be in the ontology somehow. Write down at least two options how to represent it differently, which need not necessarily be equally good yet not implausible either.

   Then discuss with your lab partner what the best solution would be, and why, and implement it in the ontology.[2]
   (b) We'll tackle the undesirable deduction now: Wheel $\sqsubseteq$ Vehicle is clearly wrong with respect to reality, even though logically correct given what we have represented in the ontology. This means we either made a mistake in our understanding or we understood it right but did not add at least some of the knowledge in a way that we should have done. In investigating the issue, consider:
      - Which axiom caused the deduction?
      - Does this axiom hold in all possible worlds?
        - If yes: then how should the property of Wheel be changed?
        - If no: then how should the property be changed?

   Discuss with your lab partner what the best solution would be, and why, and implement it in the ontology.
2. We'll explore peculiarities of the language and automated reasoning that beginners generally don't expect, with especially when they have a background in databases. They involve individuals in the ontology.[3]
   (a) Add the following instances: electric bicycle(eb1), motor(m1) and motor(m2).
   (b) Relate the instances as follows: hasComponent(eb1,m1) and hasComponent(eb1,m2). Save your ontology and run the reasoner.
   (c) Why is eb1 seemingly allowed to have two motors even though the class expression states that each electric bicycle has as component exactly 1 motor?
   (d) Click on the individual m1 or on m2 and examine the inference.
   (e) Add motor(m3), declare it as different individuals from motor(m1), and add hasComponent(eb1,m3). Save your ontology and run the reasoner. What happens? (and read the text; don't click it away!)
   (f) Remove hasComponent(eb1,m3).

[1] If you did not do the exercises of Section 2.2, then you may use the OWL file I created for the lab, which is available at `http://www.meteck.org/teaching/ontologies/dubiousbicycle.owl`

[2] If you did not manage to solve the unsatisfiable class issue, or simply want to see a possible (not necessarily the best) solution, then you may inspect the OWL file I created for the lab, which is available at `http://www.meteck.org/teaching/ontologies/dubiousbicycle1.owl`.

[3] If you did not do the exercises of Section 2.2 or did not manage to resolve the issues in Exercise 1, then you may use the OWL file I created for this exercise, being `http://www.meteck.org/teaching/ontologies/dubiousbicycle2.owl`.

3. This exercise zooms in on the so-called *RBox*, or things we can do with object properties, and more automated reasoning.[4]
   (a) Add the following instances: spoke(s1), Wheel(w1) and bicycle(b1).
   (b) Add partOf as object property and declare it to be transitive. (bonus question: why can't we just reuse hasComponent for that?)
   (c) Add that spoke is a part of wheel and that wheel is a part of bicycle, and do so likewise for s1, w1, and b1.
   (d) Run the reasoner and inspect the property assertions of s1. Why did it deduce that?
   (e) Add participatesIn as object property, the chain partOf ∘ participatesIn ⊑ participatesIn (in Protégé, use a lowercase 'o'), the class cycling and instance cycling(tdf1), and then participatesIn(b1,tdf1).
   (f) Run the reasoner and inspect the property assertions of s1 again. What has changed and why? And what about w1?

2.4 Additional improvements

There are multiple methods, tools, and guidelines to improve on an ontology. They may involve 'cleaning up' orphan classes that are never used, removing duplicate knowledge, adding more constraints, and more. Since this is only a 1.5h lab, these exercise are but a small sampling. They can be done in any order.[5]

1. As with software development, good housekeeping of an ontology is better than the alternative. For instance, adding an annotation to a class helps understanding it, especially if not everything could be represented. A well-known set of heuristics for that is OOPS! for which there's a web-based tool that scans the ontology on such matters. Test your ontology with it at `https://oops.linkeddata.es/`. Then examine the output:
   (a) Do you agree with all the issues OOPS! raised? If not: why not?
   (b) Which one(s) was (were) most useful to improve your ontology?
   (c) In case OOPS! is offline or you do not have internet access, then consider the following issues, which are among the ones that came up with my `dubious-bicycle2` file:
      i. It's good advice to declare domain and/or range axioms, as OOPS! also mentions, yet there were infelicities like we've seen in Exercise 1.2c and Exercise 2.1b. Enumerate all pros and cons of declaring domain and range axioms.
      ii. hasComponent has no inverse declared, yet OOPS! suggests doing so. Why should you; or should you not do so? (Hint: the answer has both a logic and ontological aspect to it.)
      iii. The file uses different naming conventions of the vocabulary and not having declared the IRI properly. What is a good naming scheme?

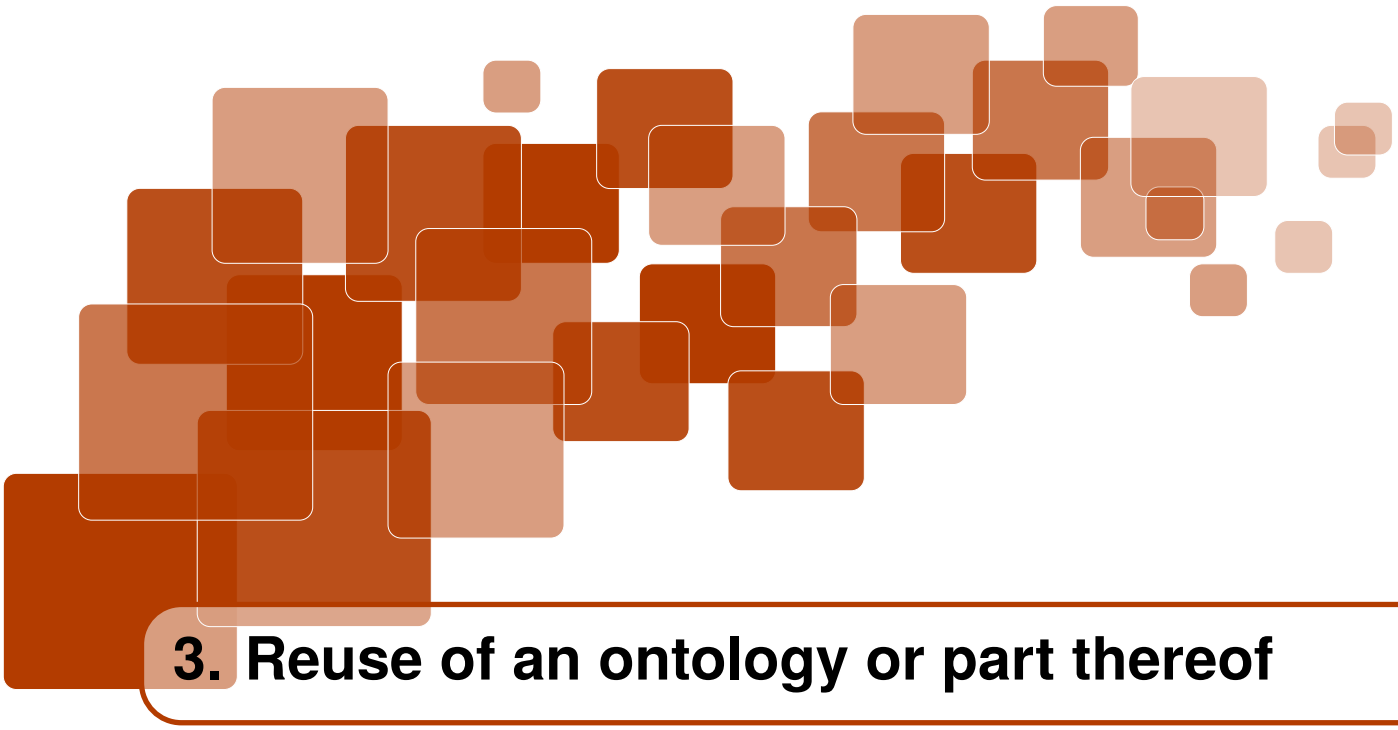
[4] If you did not do the exercises of Section 2.2, did not manage to resolve the issues in Exercise 1 or didn't do Exercise 2, then you may use the OWL file I created for this exercise, being `http://www.meteck.org/teaching/ontologies/dubiousbicycle3.owl`.

[5] If you did not do the previous exercises and just want to try explore improving an ontology, you may use the test file available at `http://www.meteck.org/teaching/ontologies/dubiousbicycle5.owl`.

2. We will now align the test ontology to a foundational ontology, being BFO 2.0. You may use either the diagram or the BFO Classifier tool, which are available from `https://github.com/mkeet/BFO2DecisionDiagram`.
   (a) Either (1) open the latest version of your test ontology on bicycles in the ontology editor and add some more entities, including at least Tour and Biker, or (2) open the facilitator's cleaned up and extended version, available from `http://www.meteck.org/teaching/ontologies/dubiousbicycle6.owl`
   (b) List the classes that are a direct subclass of `owl:Thing`. These are the entities that will have to be aligned.
   (c) If you do not use the BFO Classifier tool, then get BFO v2.0 and `import` it into your ontology yourself first. If you will use the tool, you still may wish to import it manually, but it is not a requirement (the tool can do it for you).
   (d) For each entity in the list:
      i. Start at the top and continue answering the questions until either 1) you reached the end and there is no further question, or 2) you don't know what the answer to the question should be.
      ii. Write the entity it aligns to next to the name of the class from your test ontology and any further considerations, such as a copy of the question history or why you couldn't answer the questions or where along the path you may not have been certain about the answer you gave.
      iii. Import the axiom (if using the tool) or add the subclass declaration manually in your ontology (if using the diagram only).
   (e) Assess the alignments among yourselves in the lab. If you're doing this exercise alone, you may compare to `http://www.meteck.org/teaching/ontologies/dubiousbicycle7.owl`, which definitely has one arguable alignment to probe further and discuss.

This completed the lab. If you'd like more exercises (and answers), then try the exercises in Chapters 4, 5, and 6 of the ontology engineering textbook.

This tutorial is written by C. Maria Keet and was developed for a 3-hour 'bar camp' session at ISAO'23. It was slightly adjusted for this workbook.



3. Reuse of an ontology or part thereof

The aim of this tutorial is to come to grips with the notion of *ontology reuse* concerning three aspects of the task:

1. Reusing a foundational/top-domain/core ontology for your domain ontology;
2. Reusing another domain ontology, or part thereof, for your domain ontology;
3. Avoiding practical pitfalls and dead ends doing so.

There are more aspects to ontology reuse, but this will take up a tutorial slot. If you use the accompanying textbook, then the exercises in Sections 3.1 and 3.2 are suitable in conjunction with the theory in textbook Chapter 6, and the exercises in Sections 3.2 and 3.3 align with content from Chapters 6 and 11.

For the main part of this tutorial, we're going to pretend you have been asked to develop an ontology about wastewater and stormwater utility networks, which is needed for AI-driven data management of the Digital Twins of such critical infrastructure. Better insight in the capacity of the networks and strain on them is expected to lead to better analysis and more accurate hydrological modelling, and consequent better infrastructure management to cope with draughts, floods, and rolling blackouts, and therewith lead to better services to people and businesses.

The ontology is expected to contain information about physical components such as pipes and pumps, as well as processes and tasks such as maintenance and repair, and who's going to do the repair of what.

There are several options to avoid starting with a clean slate and staring at flickering cursor on an empty page. We will look at each in turn.

3.1 Exploring reusing a top-level ontology

Instead of starting from scratch, we will explore reusing a top-level ontology to help us start structuring the prospective content, as it has been shown to be beneficial for

domain ontology development [16]. For this exercise, we will use BFO 2020 (from 2024) and Protégé v5.6.5, but it equally well can be carried out with any other OWL-ized foundational ontology and ODE.

Exercise 3.1 After opening your ODE:

1. Open `bfo-core.owl` that you already would have downloaded from an earlier tutorial, exercise, or else from `https://github.com/BFO-ontology/BFO-2020/tree/master/src`.
2. Add a few classes, including: Pipe, Manhole, Pump, Repair, Water, and Maintenance. Aligning it to BFO manually or with the aid of the BFO Classifier [3] would be good, but is not necessary for this task.
3. Save the ontology in the RDF/XML format, be it with the same file name, i.e., `bfo-core.owl`, or another name of your choice.
4. Inspect your newly saved `.owl` file in a text editor. What are the IRI's of the classes you have added? Is this correct?

■

If you carried out the steps, you should see something like shown in Listing 3.1, showing that `Pipe` was added and declared to be a subclass of BFO's Object that is represented by `BFO_0000030`.

```
<!-- http://purl.obolibrary.org/obo/bfo.owl#Pipe -->

<owl:Class rdf:about="http://purl.obolibrary.org/obo/bfo.owl#Pipe">
    <rdfs:subClassOf rdf:resource=
      "http://purl.obolibrary.org/obo/BFO_0000030"/>
</owl:Class>
```

Listing 3.1: OWL snippet resulting from Exercise 3.1.

That is: the IRI of `Pipe` is that of BFO, i.e., we revised BFO 2020, a top-level ontology! The aim was to develop a domain ontology about wastewater and stormwater networks, however, not forking BFO and bypassing the BFO team of curators and adding domain entities in a top-level ontology.

Thus, this is not what should be done when reusing a top-level ontology.

3.2 Importing an ontology

Let's try to reuse the top-level ontology in a different way.

Exercise 3.2 Start with a clean project, or close BFO if you continue from the previous task, or click File - New - No.

1. Create a new ontology, and add a few classes, include: Pipe, Manhole, Pump, Repair, Water, and Maintenance.
2. In the 'Active ontology' tab, define the IRI where you would hypothetically publish it on the web; e.g., your home page. Click 'yes' to change the IRI's of the entities already declared. (alternatively, do this step before adding entities).
3. Still in the 'Active ontology' tab, click the '+' sign next to 'Direct imports' and import BFO by 'document located on the web' and pasting `http://purl.`

obolibrary.org/obo/bfo.owl in the textbox.
4. Align some classes, e.g., Pipe $\sqsubseteq$ Object.
5. Save the file in RDF/XML with a .owl extension; e.g. sewerTut1.owl
6. Inspect the new OWL file in a text editor. What are the IRI's of the classes you have added? Is this correct? What are the IRI's of the BFO entities?

■

If all went well, it should look like Listing 3.2, where the domain entities have the IRI you chose and the BFO entity kept its own IRI.

```
1 <!-- http://www.meteck.org/teaching/ontologies/sewerTut1#Pipe -->
2
3 <owl:Class rdf:about=
4   "http://www.meteck.org/teaching/ontologies/sewerTut1#Pipe">
5     <rdfs:subClassOf rdf:resource=
6       "http://purl.obolibrary.org/obo/BFO_0000030"/>
7 </owl:Class>
```

Listing 3.2: OWL snippet resulting from Exercise 3.2.

Scrolling to the top of the file, near the end of the ontology metadata, you will see what location the purl actually pointed to when fetching the object, in the owl:imports statement, which will be along the line of http://purl.obolibrary.org/obo/bfo/2019-08-26/bfo.owl.

Now we're set for a more demanding import. Let's assume the stakeholders also want to include some aspects about time in the ontology to allow annotation about repairs, maintenance, faults, and so on, and that you all agreed on reusing the Time ontology, available at https://www.w3.org/TR/owl-time/. It is available in .ttl, so let us convert it to RDF/XML first (use 'Save as..' and select RDF/XML from the drop-down list), save it as time.owl, and use that.

Upon inspecting the ontology, all stakeholders agree that they want to reuse only a module of Time and import that; more specifically, *sans* the data properties. We'll try to do this in the next exercise.

Exercise 3.3 Continue from the previous task. Click File - New - No, and open time.owl.

1. Inspect it to familiarise yourself with the contents.
2. Remove all data properties, either manually or use a tool such as NOMSA [25] for that, click 'Save as' and save the module as timeCore.owl.
3. Import the local file timeCore.owl into sewerTut1.owl using the 'Direct imports', observe the import statement, and verify that there are no data properties in the ontology.
4. Update the version IRI to sewerTut2, and add a few axioms that reuse Time, including Repair $\sqsubseteq = 1$ hasBeginning.Instant. Document them.
5. Save the ontology and inspect the .owl file in a text editor.
   (a) Are all IRIs as they should be, in that your IRI, BFO's IRI, and the Time's IRI each are distinct?
   (b) Is this (also) reflected in the description of the Repair class?
   (c) Can you explain why it has the Time IRI and not something mentioning

'timeCore'?

(d) Can you find a statement—anywhere in the file—that it has imported your `timeCore.owl` module specifically?

6. Close Protégé and re-open the ontology. Depending on your configuration, Protégé will ask you whether it's allowed to access the folder where your ontologies are stored.
   (a) Click 'Yes' and check the 'Data properties' tab: are there data properties?
   (b) Close and reopen, but this time, click 'No' and check the 'Data properties' tab. Are there data properties?

   What is the difference and why is it so?

■

You may delete the data properties again (select all and delete also works), save, close, and reopen. You also could email your sewerTut file to a classmate and let them open it to see what happens, or put your ontology temporarily (!) online in a staging area and load your ontology from that URL.

It does not matter what you try—the OWL file itself has no notion of your local module and therefore will keep trying to load the whole ontology located at the IRI indicated. To get around this, you could create a new IRI for the module and import that, or use DOL. The effects are left as an exercise for the reader and also will pass the revue in the bonus task in Section 3.4.

3.3 Copying over a few axioms

There is a third way of reusing an ontology. It is not the intended way of reuse as envisioned by those who standardised OWL, but some people got creative in the face of the surprises we saw in the previous tasks. Meanwhile, it has probably become the most popular way of reusing an ontology thanks to the OBO Foundry's ecosystem of ontologies—if you know where to look and what to do. The following exercise is the cookbook instruction version of it and we'll reflect on it afterward.

Exercise 3.4 Reopen `sewerTut1`, from before the Time imports; if `sewerTut1` is still open, then remove Time, save the ontology, close the application, and reopen.

1. Open `time.owl` through either File - Open - Yes and selecting the Time ontology file, or File - Open recent - and select the file.
2. Having Time in view in Protégé, select Refactor - Copy/move/delete axioms. Then in the 'Select the method of choosing axioms...' popup, choose 'Axioms by definition ...'. Click Continue.
3. You may select the classes you like, but at least also do include the Time instant class and click the >> button.
4. Navigate to the 'Object properties' tab, select has beginning and has end and click the >> button. Click Continue.
5. It then will show you that it will take 29 axioms along with it, or more if you also had selected other elements. Click Continue.
6. The next screen asks you whether to copy, move, or delete. Choose Copy and click Continue.
7. The next screen asks you whether to copy them to an existing ontology or a

new one. Choose Existing and click Continue.
8. The next screen asks you which ontology the axioms should be copied into. Choose `sewerTut` and click Finish.
9. Find Repair and add Repair $\sqsubseteq = 1$ hasBeginning.Instant.
10. Save the revised ontology, close, and reopen. Then answer the following questions.
    (a) Did your ontology import the whole ontology again?
    (b) Is the Repair axiom still there?
    (c) What does Repair look like when you open the file in a text editor?
    (d) Is the metadata about the Time ontology different from the previous exercise?

■

If all went well, it will not have imported the time ontology and there is no `owl:import` statement in the metadata. There are a number of Time ontology axioms in the file, which all involve the three elements we had selected in the exercise. They also still have the proper Time IRI rather than having being changed into the sewerTut one. The Repair axiom with the Time ontology elements appears the same as upon importing. And any closing/reopening of the application does not cause the whole ontology to be imported.

Copying over axioms like this is not how it is supposed to happen, but it clearly resolves the modular import issue. It also avoids any repercussions in case the developers of an imported ontology decide to move the file or change its contents. The flip side of that is that your ontology will not automatically update accordingly and may go woefully out of sync with the source ontology.

3.4 Inspecting Stuff

The original plan was to use the Stuff ontology for this tutorial, but it was less straightforward and so it has ended up as a 'bonus task' for those who deemed the previous tasks to be easy. Ontologies are reused, there are problems, and you are asked to propose a satisfactory solution covering the what, why, and how.

For this task, we're going to pretend you want to develop an ontology about stuff that the Ministry of Food and Agriculture asked you to develop. They want to be able to trace ingredients throughout the production and distribution chain so that when (not if) there is a food infection or poisoning, the product can be traced back to the source, i.e., from fork to farm if need be[1]. Thus, you need to develop an ontology about stuffs, like mayonnaise, mincemeat, tiramisu, treacle, and so on. They are not all of the same type: some stuffs are extracts from the source—e.g., treacle from sugar beet—others are special types of mixtures—mayonnaise is a colloid and homogeneous mixture; tiramisu is a heterogeneous mixture—which is obviously important for the traceability algorithms.

As luck would have it, there already is a core ontology of macroscopic stuff that

[1]For the morbidly fascinated: a list of the world's deadliest food-borne illnesses and deaths is regularly updated here: `https://en.wikipedia.org/wiki/List_of_foodborne_illness_outbreaks_by_death_toll`. The 2017-2018 listeriosis outbreak in the polony (a type of fermented sausage) tops the list with 216 deaths in South Africa (1,060 infected), and the first spot for most infected people goes to the Salmonella infected ice cream in the USA in 1994 where an estimated 224,000 people were infected (no-one died).

provides the high-level categorisations of types of stuff and their relations [14, 17]. On top of that, it has been aligned to both BFO and to DOLCE, so we should not have to worry about that either. Now the question becomes: how? There are several options, just like we have seen for sewers:

O1: Open Stuff in your favourite ODE and add subject domain content.
O2: Create a new file/project in your favourite ODE, import Stuff, and add subject domain content.
O3: Create a new file/project in your favourite ODE, select a section from Stuff to create a module, import the module, and add you subject domain content.
O4: Create a new file/project in in your favourite ODE, add subject domain content, select a few axioms from Stuff.owl and copy those into your ontology.

If you completed the exercises in the previous sections, you now should be able to argue for which option(s) would be preferable.

Let's see what's going on with Stuff. It used to work 10 years ago, albeit not neatly, and meanwhile it has become a bit 'messy' in 2025. You will first look into the problems in the following exercise.

Exercise 3.5 Open your ODE of choice (I will use Protégé v5.6.5) and open the Stuff Ontology after navigating in a browser to `http://www.meteck.org/stuff.html`. There are several versions and it can be done in several ways, partially depending on what happens during the process.

1. You will open and inspect the Stuff ontology four times in four different ways. Note down your observations each time.
   - Scroll down, right-click 'stuff.owl' and in the ODE, select 'open from URI', and paste `http://www.meteck.org/files/ontologies/stuff.owl`. It gives an error "could not open...". Click 'no'. Inspect the ontology.
   - Scroll down, right-click 'stuff.owl' and in the ODE, select 'open from URI', and paste `http://www.meteck.org/files/ontologies/stuff.owl`. It gives an error "could not open...". Click 'yes'. Download `OMmini.owl` from `http://www.meteck.org/stuff.html`. Go back to Protégé, select the OMmini file you just downloaded and go ahead. Inspect the ontology and take note of how DOLCE and the ontology of units of measure are reused.
   - Close Protégé and open `stuff.owl` by file, and resolve the import by loading `OMmini.owl` from you machine. Inspect the ontology and take note of how DOLCE and the ontology of units of measure are reused.
   - Scroll down, ignore v2 of the ontology, and open Stuff v1 that is aligned to DOLCE-Lite, i.e., `http://www.meteck.org/files/ontologies/stuff-dolcelite.owl`. It gives that error "could not open...". Click 'yes' and load `OMmini.owl`. Inspect the ontology and take note of how DOLCE and the ontology of units of measure are reused.
2. List the key differences you observe between the four options.
3. Describe what does not look right regarding the top level of the ontology and these processes, and what is going on with OMmini.

■

We'll look at considerations, explanations, and issues, which should inform answers

to the questions in the next exercise.

The main reason for not finding an import is link rot. The reasons why this happens vary, from Semantic Web-clueless webmasters moving files elsewhere and then the ODE cannot resolve the URL to load the file as object on the Web, to ontology developers changing the ontology and thereby rendering it incompatible with your ontology (content or IRI), to end-of-project and the bill for the domain name renewal was left unpaid. These are thus also factors to consider when you publish your ontology.

The problems with the Ontology of units of Measure (OM) that Stuff imports are intricate. First, OM used to be available at `http://www.wurvoc.org/vocabularies/om-1.8/`. Now, it automatically redirects to `http://www.foodvoc.org/page/om-1.8`, which renders the ontology as web pages. On the foodvoc page, v1.8 is available for download in `.ttl` format as `http://www.foodvoc.org/data/om-1.8.ttl` instead of the official exchange syntax RDF/XML with `.owl` file extension. Opening it in Protégé to convert it and save it in RDF/XML returns two syntax errors in long comments. If there had been no `OMmini.owl` on my website to 'resolve the import' with a local file, you could have tried this approach in the hope that the declared equivalence and subsumption alignments would still work.

An alternative option to resolve the module import for Stuff could have been creating a new IRI for the OM module, alike a `http://www.meteck.org/ontologies/OMmini`, and find-replace the OM IRIs with the new IRI. The principal downside is that an element is defined by its IRI, so then an entity like `http://www.meteck.org/ontologies/OMmini/base_quantity` is distinct from `http://www.wurvoc.org/vocabularies/om-1.8/base_quantity`, and requires an equivalence axiom alignment to state they are the same. (Another alternative would have been to import OM in full, including all the knowledge that is not relevant, which is also sub-optimal.)

In the Stuff ontology case, importing a current version or OM would have been problematic due to extensive redesign of OM that the Stuff ontology developer had no say in (and neither tried to nor knew of until creating this tutorial). In addition, OM v2 has problems in the light of reuse for Stuff; e.g., there is no `base_quantity` anymore, having been replaced by `http://www.ontology-of-units-of-measure.org/resource/om-2/hasBaseQuantity`, so the alignment would have been broken if the system had fetched the latest version. To accommodate for such cases, some ontologies include an `Obsolete` class in the OWL file, to prevent dependencies from ceasing to function immediately.

Last, a few notes on Stuff's reuse of DOLCE. Upon inspecting the ontology, you will have seen duplications in vocabulary, as shown in Listing 3.3. I had lost my alignments before due to link rot, and the equivalence assertions were a way to keep a record of the alignments in the file. The other main reason was to keep Stuff and `stuff.owl` independent of DOLCE: it can align to other foundational ontologies and these top-level entities helped structuring content independent of the foundational ontology. A minor reason was the 'consistent naming scheme' prettiness ontology quality property that the vocabulary in Stuff adheres to.

These explanations about the problems and why certain things ended up the way they did do not make Stuff right. The next exercises probes for direcrions towards a better solution.

```
<rdf:Description rdf:about=
  "http://www.meteck.org/files/ontologies/stuff.owl#hasQuality">
    <rdfs:subPropertyOf rdf:resource=
      "http://www.loa-cnr.it/ontologies/DOLCE-Lite.owl#has-quality"/>
</rdf:Description>

<rdf:Description rdf:about=
   "http://www.loa-cnr.it/ontologies/DOLCE-Lite.owl#endurant">
    <owl:equivalentClass rdf:resource=
      "http://www.meteck.org/files/ontologies/stuff.owl#Endurant"/>
</rdf:Description>
```

Listing 3.3: OWL snippets where Stuff aligns to DOLCE.

Exercise 3.6 Having gathered information, your task is to devise a solution at least on paper.

1. Consider at least the following questions first:
   - Should Stuff be quasi-agnostic about foundational ontology alignment and commit to one only? Justify your answer.
   - Should DOLCE or BFO be imported into `stuff.owl`?
   - Should the OM v1.8 be updated with OM v2.0 in part (as module) or whole?
   - Should neither be imported through a 'hard' import but rather copied in through a 'soft' import with a few axioms only?
   - Might it be useful to have a mechanism in the ontology file so that it know to fetch a module rather than the whole file from the IRI specified, for those cases where the developer has no control over that other ontology? If so: how could that be done?
2. Modify the state of affairs accordingly and test that it works.

■

This tutorial is written by C. Maria Keet and it was developed because it seemed to be needed. The wastewater and stormwater toy example was inspired by the experiences developing the SewerNet ontology, which imports DOLCE, has a few Time axioms that were copied over, and eventually did not have PW imported; that real domain ontology is available from `http://sewernet.msem.univ-montp2.fr/`.

4. OntoClean in OWL with a DL reasoner

The aim of this tutorial is to illustrate how the OntoClean methodology can be used in Protégé with OWL and its reasoner, based on the OntOWLClean approach proposed in [40]. In particular, it focuses on the following tasks:

- Punning an ontology in preparation for OntoClean.
- Assigning meta-properties to classes in OWL.
- Discovering inconsistencies in a taxonomy, and hints for fixing the hierarchy.

The rest of the document is structured such that Section 4.1 presents an overview of OntoClean and Section 4.2 presents our example ontology, the manner in which OntoClean can be followed within Protégé, and a limited number of errors that should be discovered from the provided ontology.

If you use the accompanying textbook, then this lab is suitable in conjunction with the theory in Chapter 5 and provides more practice for content presented in Chapter 4.

4.1 OntoClean

OntoClean is a philosophy-based methodology for validating the correctness and consistency of an ontology's taxonomy. It is based on general notions that are drawn from philosophy, which are Rigidity, Identity, Unity, and Dependence. The methodology is made up of two phases. The first phase involves annotating all the classes within an ontology with labels of the meta-properties referring to the four philosophical notions. The second phase deals with the checking of subsumption relationships of the ontology based on the predefined OntoClean constraints, which in this document are also referred to as rules.

In the remainder of this section, we provide a brief recap to the four philosophical concepts, the OntoClean meta-property annotation symbols, and the constraints for each of them. A larger summary is described in Section 5.2.2 of the textbook, an overview of

OntoClean is described in the handbook on ontologies [11], which build upon foundations presented in [12, 13].

Rigidity refers to an entity's property that are essential to that entity (i.e. it must be true of it in every possible situation). For instance, the property of having walls is essential to a house. Every house must have walls in every possible situation. In the event that they are demolished then you no longer have a house. Identity refers to the capability to identify individual entities in the world as being the same or different. Unity refers to the ability to describe the parts and boundaries of objects, and thus to know which parts constitute an object, which parts do not, and under what circumstances is the object a whole. Dependence is the relationship between entities whereby one will exist solely on the existence of the other.

All the entities within an ontology must be assigned with meta-properties and labelled with the letter denoting the meta-property; more precisely: (**I**) for Identity, (**U**) for unity, (**D**) for dependence and (**R**) for Rigidity. Each of these labels preceded with a +, −, or ∼ symbol, where (+) means the entity is what the letter denotes, (−) means the entity is not what the letter denotes, and (∼) means 'anti' (may or may not be) to what the letter denotes.

Recall from Exercise 5.4 of the textbook that the assignment of meta-properties is useful because there are constraints that the taxonomy must not violate, and the rules that apply. For instance, when we have two properties `x` and `z`, where `z` subsumes `x` then we know that if `z` is anti-rigid (∼R) then `x` must be anti-rigid (Rigidity constraint), if `z` carries an identity criterion (+I) then `x` must carry the same criterion (Identity constraint[1]), if `z` carries a unity criterion (+U) then `x` must carry the same criterion and if `z` has anti-unity, then `x` must also have anti-unity (Unity constraints), and if `z` is dependent (+D) on a certain property `y` then `x` is dependent on property `y` (Dependence constraint).

4.2 OntoClean in Protégé

In this section we will specify where to download the required software, describe the ontology we will use to illustrate OntoClean, introduce how to pun the ontology in Protégé in preparation for OntoClean, and present an exercise on assigning meta-properties in the tutorial ontology.

4.2.1 What to download

You will need the following material for this tutorial:

- Protégé 5.x, which can be downloaded from `https://protege.stanford.edu/`.
- OntoClean and AmountOfMatter tutorial ontologies:
  - `ontoclean-dl.owl` (OntoClean ontology)[2]
  - `OntocleanTutorialOntology.owl` (AmountOfMatter ontology, which is the domain ontology that is to be 'cleaned up')
  - `OntocleanTutorialOntologyPunned.owl` (Punned AmountOfMatter ontology)

[1] This is not true in the case of the "own" identity criteria

[2] The OWL-DL ontology developed by [40] is no longer available through the OntoClean website at `http://www.ontoclean.org/`, therefore we provide a cached version.

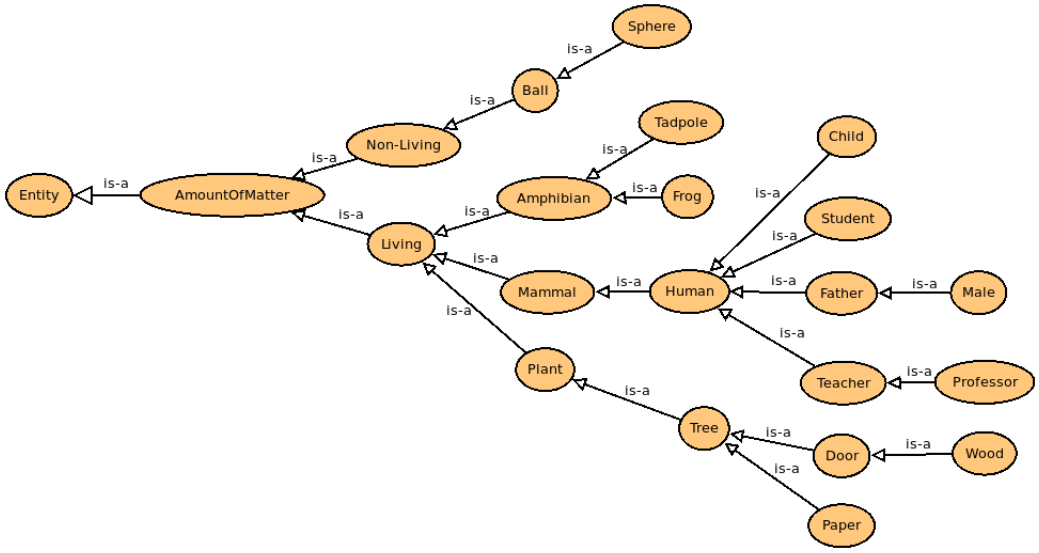


Figure 4.1: Tutorial ontology with ontological inconsistencies.

– `OntocleanTutorialOntologyPunnedMetaProperties.owl` (Punned AmountOfMatter ontology with assigned meta-properties)

which can be downloaded from the textbook's resources page at `https://people.cs.uct.ac.za/~mkeet/OEbook/ontologies/`

Please note that another version of Protégé can be used, however, the interface may use different terms and have a different layout and look-and-feel.

4.2.2 AmountOfMatter taxonomy

The AmountOfMatter tutorial ontology (see file `OntocleanTutorialOntology.owl`) that will be used to illustrate how to follow OntoClean is shown in Figure 4.1. The tutorial ontology has deliberate taxonomic issues that need to be resolved using OntoClean. In the ontology, *AmountOfMatter* can be considered as any entity and may be a *Non-Living* or *Living* object. An example of *Non-Living* object, according to the ontology, is a *Ball*. Familiarise yourself with the ontology in Protégé by doing Exercise 4.1.

> **Exercise 4.1** Open the tutorial ontology `OntocleanTutorialOntology.owl` in Protégé, import the OntoClean OWL-DL ontology `ontoclean-dl.owl`, and explore. ■

4.2.3 Punning the tutorial ontology

The first task is to push the ontology's Tbox into the ABox [40]. In particular, you must create an Individual with the same name as the class (bears the same IRI), for each of the classes on the tutorial ontology. These Individuals must be of type `ontoclean:Class`. An example is shown in Figure 4.2 where there are 21 Individuals of type `ontoclean:Class`. For each individual, you must then specify its "subclasses" through `ontoclean:hasSubClass` relation. For instance, *AmountOfMatter* has the subclasses *Non-Living* and *Living* as shown in Figure 4.2. Familiarise yourself with how to do this by carrying out Exercise 4.2. You can verify whether you've done this exercise correctly by cross-checking with the provided `OntocleanTutorialOntologyPunned.owl` file.

Exercise 4.2 Continuing from Exercise 4.1:

1. For each class in the ontology, create its corresponding Individual with the same name as the class.
2. Make each newly created individual an instance of `ontoclean:Class`.
3. Specify the "subclasses" for each Individual, using the `ontoclean:hasSubClass` object property.

■

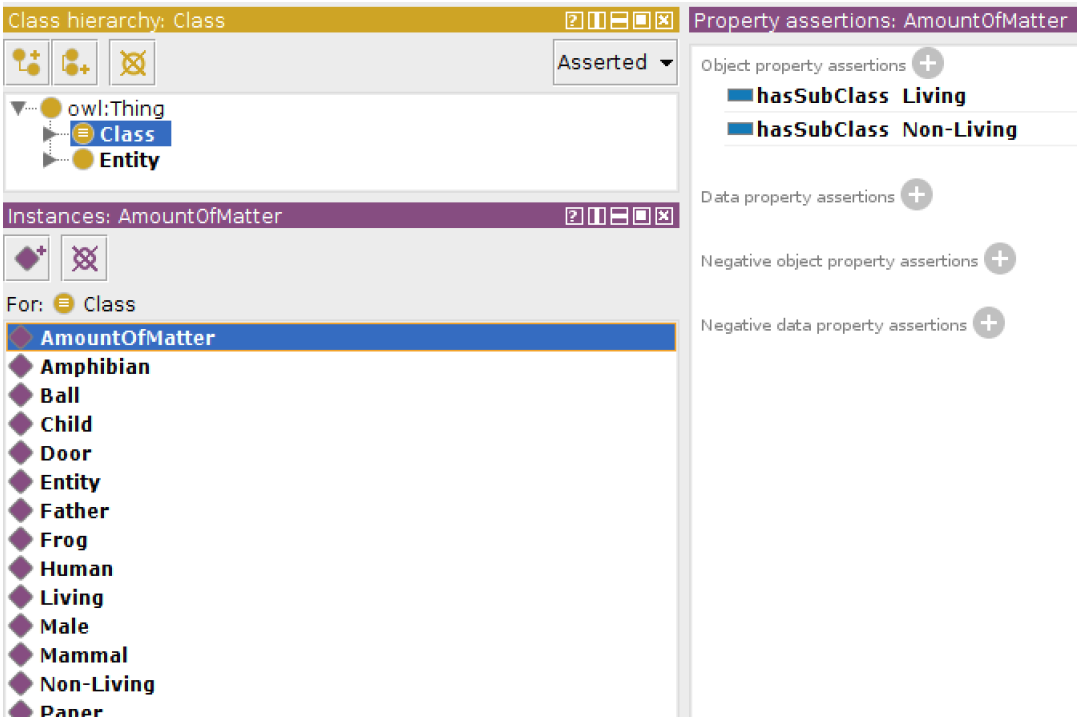


Figure 4.2: Individuals of type `ontoclean:Class` and *AmountOfMatter*'s two subclasses asserted in the ABox with `ontoclean:hasSubClass`.

4.2.4 Assigning the meta-properties

Once the ontology has been punned, you can assign meta-properties to each individual. This requires you to first decide on the meta-properties for each class/OWLfile-individual. Once that is decided, you can use Protégé to assign meta-properties to each individual by setting its type to the appropriate subclasses of `ontoclean:Class`. For instance, you could deem that *Paper* is dependent (+D) by assigning it the *Dependent* meta-property, which practically amounts to adding the assertion that *Paper* is an instance of `ontoclean:DependentClass` as shown in Figure 4.3. Carry out Exercise 4.3.

Exercise 4.3 Assign metaproperties:

1. List all the classes in the ontology on a sheet of paper and assign your own meta-properties.
2. Compare your assignments with the ones provided in Figure 4.4.

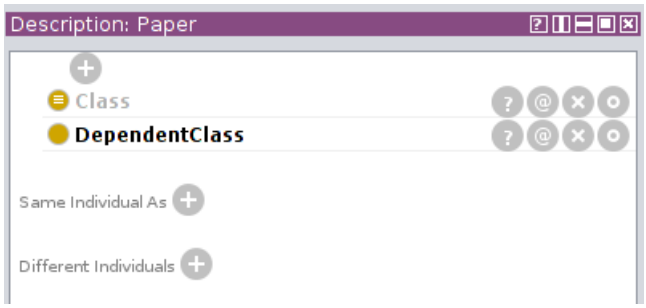


Figure 4.3: Paper individual meta-property assignment.

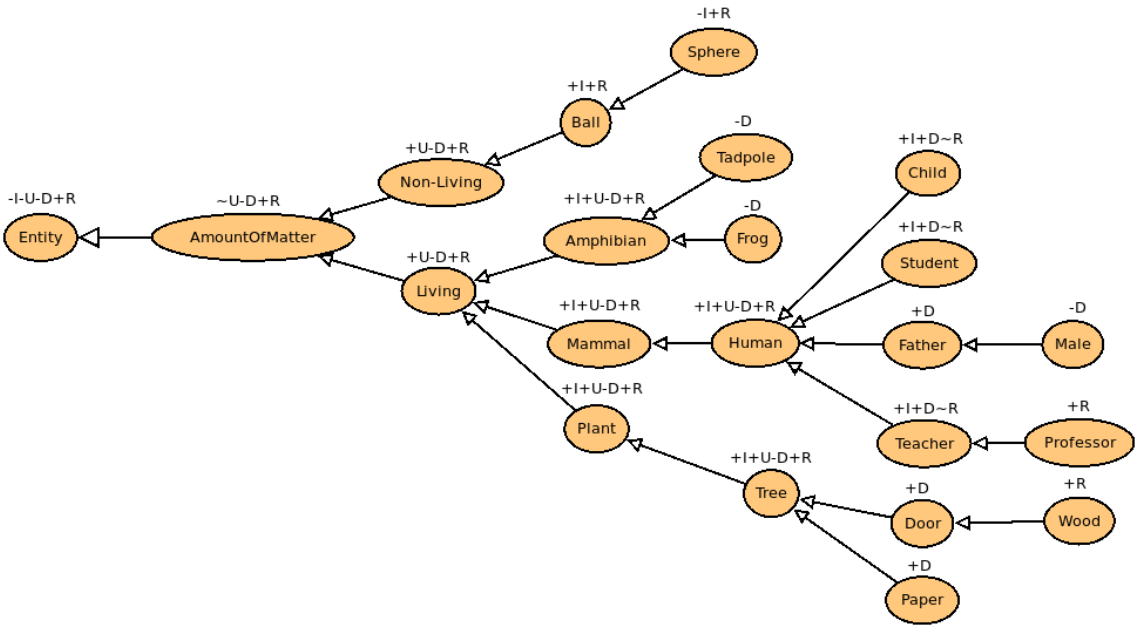


Figure 4.4: Tutorial ontology with possible meta-properties assigned.

3. Now add the metaproperty assignments to the OWL file, be it either your own assignments or the ones provided in Figure 4.4: for each Individual, declare it an instance of the respective metaproperty class. For instance, the instance version of *Paper* has to become an instance of `ontoclean:DependentClass`.

■

We provide a sample of meta-property assignments for all terms in this tutorial as shown in Figure 4.4. The file `OntocleanTutorialOntologyPunnedMetaProperties.owl` contains all the assignments per individual. You are encouraged to also make your own separate assignment to the ontology by completing Exercise 4.4. Note that your meta-property assignments may be different because the "same [terms from an ontology represent] different concepts to different people" [40]. If your meta-property assignments are different from Figure 4.4, you are encouraged to create your own version of the `OntocleanTutorialOntologyPunnedMetaProperties.owl` file. The next section will show you how to detect inconsistencies in the provided (or your own) punned file with assigned meta-properties.

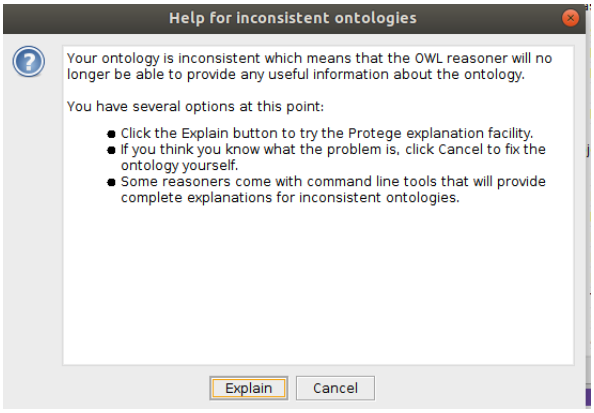


Figure 4.5: Reasoner notification for inconsistencies.

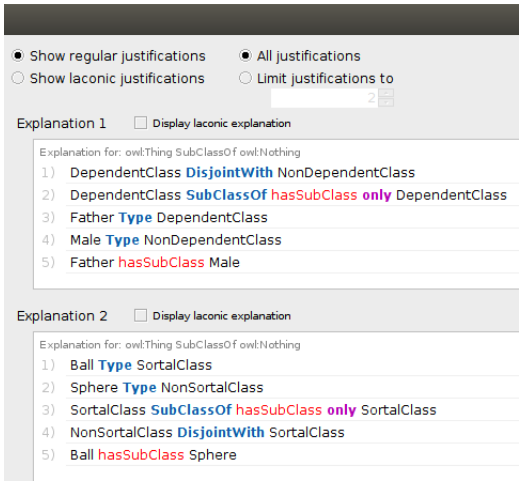


Figure 4.6: Section of detected inconsistencies.

4.2.5 Detecting inconsistencies in the hierarchy

Once meta-properties have been assigned and added to the OWL file, inconsistencies can be discovered by starting a reasoner[3]. For instance, when *HermiT 1.3.8* is started on the provided `OntocleanTutorialOntologyPunnedMetaProperties.owl` file, you should encounter something that looks like the screenshot in Figure 4.5. Contrary to intended use of reasoners for ontology development, this is supposed to happen. Upon clicking the explain button then a pop-up with a list of inconsistencies will be shown (see Figure 4.6). Carry out Exercise 4.4.

A number of errors should be detected, be it either form your own assignments or the one provided. In the remainder of this section, we describe some of those errors and their causes.

Exercise 4.4 Managing inconsistencies.

1. Ensure to have open either the provided OWL file that has the meta-properties assigned (`OntocleanTutorialOntologyPunnedMetaProperties.owl`) or your own one where the meta-properties have been assigned.
2. Start the reasoner.
3. Click the "Explain" button. You can either let it compute as many explanations as it will, or stop after a few. We assume you stop after a few.
4. Examine the first explanation for the inconsistency. What is the cause of it? That is: which OntoClean rule did it violate?
5. Once you are convinced which rules it violated, correct the taxonomy in the TBox (i.e., among the classes) and their corresponding individuals in the ABox. Resolution can be either to change the meta-property assignment or to change the position of the classes in the taxonomy.
6. Carry out steps 2-5 until there are no reported inconsistencies.

■

AmountOfMatter and Living

AmountOfMatter and Living have an inconsistency and violate the unity rule that says

[3] `https://protegewiki.stanford.edu/wiki/Using_Reasoners`

an entity with a unity tag may not be a subclass of an anti-unity entity. In this case *AmountOfMatter* has an anti-unity tag as amount of matter may be any object but may not be considered as a whole (for instance, water) hence there can be no one unifying criterion. The Living entity has a unifying criterion, that is, any instance of it should be living though it may be any living object hence it may not be a subclass of *AmountOfMatter*.

AmountOfMatter and Amphibian

Amphibian and *AmountOfMatter* are also inconsistent as they violate the unity rule that states that entities that possess positive unity tags may not be subclasses of entities with anti-unity tags. Though not a direct subclass of *AmountOfMatter*, *Amphibian* is however indirectly a subclass through the relation to the Living entity which is a subclass of *AmountOfMatter* and automatically makes it a subclass of *AmountOfMatter* too.

Sphere and Ball

Sphere and *Ball* also have an inconsistency as they violate the identity constraint that says that an entity with an identity criterion may not be a super class to a non-identity class. A ball has an identifying criterion in that a ball may easily be recognised as a ball, whereas a sphere may be any object that has a sphere shape.

Father and Male

Father and *Male* entity relationship violates the dependence constraints which states that a non-dependent entity may not be a subclass of a dependent class. Thus, for someone to be a male they do not need to be father, in other words Male does not depend on *Father*. Rather, conventionally for you to be a father then you must be a male.

Door and Wood

The *Door* and *Wood* relationship is incorrect: it violates the rigidity rule. *Door* cannot be a super class of wood since it is anti-rigid and *Wood* is rigid. A door is considered anti-rigid in the sense that a door may cease to be a door say, when it breaks and get replaced. Whereas wood will always be wood whether it is used to make some furniture or just stored for future use.

Teacher and Professor

The relationship between *Teacher* and *Professor* violates the rigidity rule. The professor is a rigid concept as a person once granted the title they will always be a professor as long as they live. A teacher however may cease to be a teacher at any moment or stage in their life, hence it is anti-rigid.

AmountOfMatter and Mammal

The relationship between *AmountOfMatter* and *Mammal* via *Living* is inconsistent as it violates the unity constraint that states that an anti-union class may not be a super class of a unity class. Mammals have unifying characteristics that may include giving birth to live babies and that they are all vertebrates but amount of water may not be unified as other matter may not be quantifiable as a whole for example water.

AmountOfMatter and Human

The relationship between *AmountOfMatter* and *Human* violates the unity rule since *AmountOfMatter* is anti-rigid and *Human* is rigid. It is not a direct violation but since

Human's superclass has an ancestor (*AmountOfMatter*) that is anti-unity. *AmountOfMatter* is not a whole for all its instances so they cannot unified but *Human* has a unifying criteria that may include being warm-blooded and giving birth to live offspring.

Human and Teacher

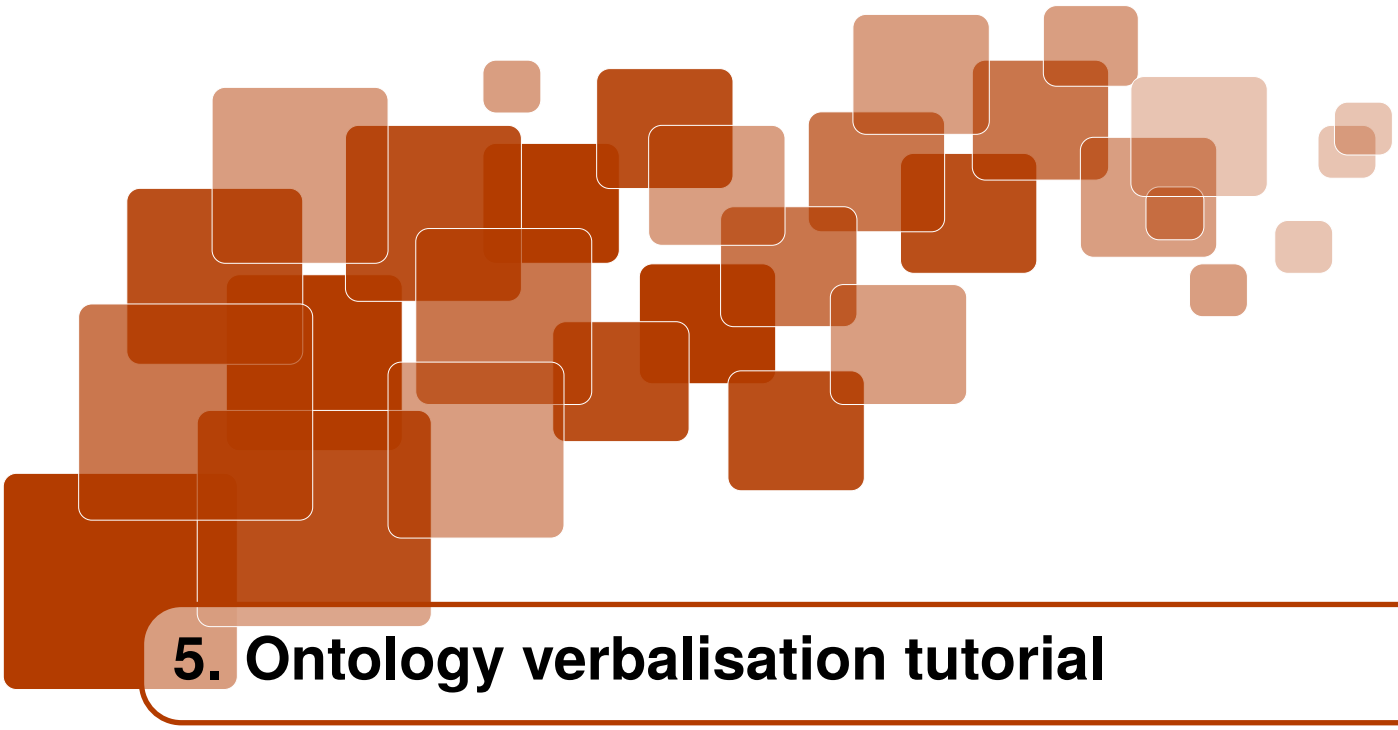
The relationship between *Human* and *Teacher* violates the unity rule; a non-unity class may not be a subclass of an anti-unity class.

Fixing the hierarchy

Following on from Exercise 4.4, you should have resolved each inconsistency so that the reasoner does not return you any more errors like in Figure 4.5. These errors can be fixed by either re-arranging the position of the classes in the taxonomy, or, upon closer inspection, one may have decided to change the meta-property assignment. An example of the former is that your revised hierarchy surely should not have *AmountOfMatter* at the top. An example of the latter is that a +R on *Professor* would generally be considered to be incorrect[4], but is ∼R instead.

This tutorial is written by Zola Mahlaza and C. Maria Keet (emanating from the DOT4D project) and is based on a mini-project by Todii Mashoko, Siseko Neti, and Banele Matsebula.

[4]being a professor is a role that a human plays, so it would be an anti-rigid property, and then not violating the rigidity rule anymore.



5. Ontology verbalisation tutorial

The main of this tutorial is to introduce participants to the task of generating natural language text from an ontology. You will learn how to generate text in English and in isiZulu, the most prevalent South African language by L1 speakers text. The two languages are chosen in order to expose participants to challenges, or lack thereof, that are due to the complexity of each output language's grammar.

If you use the accompanying textbook, then this tutorial is suitable in conjunction with the theory in Chapter 9 and it provides more practice for content introduced in Chapter 5.

5.1 Tutorial objectives and dependencies

In the tutorial, we focus on the following tasks:

- Parsing an ontology from a file using Java
- Retrieving and verbalising inferred axioms using Java
- Designing an English template to verbalise an OWL axiom
- Aggregating multiple English sentences to improve paragraph readability
- Designing an isiZulu template to verbalise an OWL axiom
- Generating isiZulu text from a linguistically-rich template and OWL axiom
- Ideation on how to use lexical resources that are separate from the ontology

The following software and dependencies are required to complete this tutorial:

- Java 11 (or later)
- OWLAPI, available at `https://github.com/owlcs/owlapi`
- Openlett reasoner, available at `https://github.com/Galigator/openllet`
- The NguniTextGeneration suite, available at `https://github.com/AdeebNqo/NguniTextGeneration`
- Task ontology for CNL-based Templates, available at `https://github.com/AdeebNqo/ToCT`

The only dependency that participants are expected to be already familiar with is Java. There is no need to have prior experience with the other dependencies, as they will be presented in a guided manner and additional documentation will be linked.

The rest of the document is structured such that Section 5.2 presents the background and tasks to be executed to verbalise an English ontology following the tutorial's constraints and Section 5.3 does the same for the verbalisation of an ontology to produce isiZulu text.

5.2 Simple Ontoverbal

The OntoVerbal(-M) verbaliser was created by Liang et al. [27, 28, 29] for generating English and Mandarin. It was intended to be used as an assistive tool by individuals who want to audit the contents of ontologies but have no expertise in neither ontologies nor ontology languages (e.g., OWL). Consider the following axioms pertaining to the Valve class:

```
1 (<AnatomicalCavity>DisjointClasses <Valve>)
2 (<TricuspidValve>SubClassOf <Valve>)
3 (<PartialValve>SubClassOf <Valve>)
4 (<Valve>SubClassOf <AnatomicalConcept>)
5 (<SemiLunarValve>SubClassOf <Valve>)
6 (<VestigialCardiacValve>SubClassOf <Valve>)
7 (<MitralValve>SubClassOf <Valve>)
8 (<AtrioVentricularValve>EquivalentTo (<Valve>and (<hasValveInput>
9 some<AtriumCavity>) and (<hasValveOutput>some<VentricularCavity>)))
```

Listing 5.1: Ontology snippet for the Valve (Source: [29]).

The verbaliser has the ability to take these axioms and transform them to produce the following text, as shown by Liang et al. [29]:

```
A valve is a kind of anatomical concept. More specialised
kinds of valve are mitral valve, partial valve, semi lunar valve,
tricuspid valve and vestigial cardiac valve. Also, a valve is
different from an anatomical cavity. Another relevant aspect
of a valve is that an atrio ventricular valve is defined as a
valve that has valve input an atrium cavity and has valve
output a ventricular cavity.
```

For this tutorial, we will reproduce the portion of the OntoVerbal English verbaliser that supports "simple axioms" [28, pg340]. We say that an axiom is simple in the context of OntoVerbal if it is one of the following axiom types $A \sqsubseteq B$, $A \equiv B$, $A \sqsubseteq \neg B$, and $a \in A$ where A and B can only be classes and not class expressions.

Exercise 5.1 In order to get started, execute the following tasks

1. Download the zip file located at `https://bit.ly/3pgnzXu`
2. Select your own ontology that includes the aforementioned simple axioms
3. Unzip the downloaded file and add all the files located under `reproOntoVerbal/lib` to your Java classpath

4. Open the Java verbaliser using your IDE and change line 17 in `src/Main.java` to include the path to your OWL file
5. Run the verbaliser

■

After running the verbaliser, it should only output the classes found in your OWL file. Before you can make changes to the provided verbaliser skeleton code, let us first design our first template. Execute the following task:

Exercise 5.2 Given an ordered list of classes $\{C_1, C_2, ..., C_n\}$, devise a possible English template that can be used to verbalise the list. The purpose of the template is to present the same information but using English text. Write down your template on a piece of paper. ■

In the task above, you designed a template on a piece of paper and presented a list of classes in (plain) English. We now turn to updating the Java verbaliser to create our first template for verbalising a list of classes.

Exercise 5.3 Execute the following tasks:

1. Consider an ordered list of classes $\{C_1, C_2, ..., C_n\}$. You must write code to retrieve each class' name from the URI. Specifically, complete the `getName` method in `OntoVerbal.java`. You can uncomment line 22 in `src/Main.java` for debugging purposes (Hint: Use the method(s) defined at `https://bit.ly/3ph5wk0` to retrieve the class name).
2. Create two declarative templates in separate files under `/src/templates/` that can be used to verbalise a list of classes $\{C_1, C_2, ..., C_n\}$. Specificaly, capture the following templates where $n \in \mathbb{Z}_{\geq 0}$:
   (a) a(n) $[C_1]$, a(n) $[C_2]$, ..., and a(n) $[C_n]$
   (b) $[C_1]$, $[C_2]$, ..., and $[C_n]$
3. Complete the `listClasses()` method in `OntoVerbal.java` by using the newly created `getName` method to retrieve each class name and using them to fill in the slots in templates you have just created. If `includeArticle`, the second argument to `listClasses()`, is `true` then fill the slots of template (a), otherwise fill in the slots of template (b).

■

We have now seen how to generate text when given a simple list of classes. We now need to use the created templates and function to verbalise basic simple axioms.

Exercise 5.4

1. The `getSubClassesOf` method in given in `Ontoverbal.java` shows you how to use a reasoner to retrieve an array of all subclasses of a given class in the context of some given ontology. Use the method as guide to assist in complete the following methods:

   (a) Complete the `getSuperClassesOf` method in OntoVerbal.java by introducing code to collect all classes that are found simple superclass axioms that involve `mainClass`.
   (b) Complete the `getClassesDisjointFrom` method in OntoVerbal.java by introducing code to collect all classes that are found simple disjoint class axioms that involve `mainClass`.
   (c) Complete the `getClassesEquivalentWith` method in OntoVerbal.java by introducing code to collect all classes that are found simple equivalent class axioms that involve `mainClass`.
   (d) Complete the `getInstancesOf` method in OntoVerbal.java by introducing code to collect all instances that are found in simple instantiations of `mainClass`.
2. The `verbaliseSimpleSubclasses` method in given in `Ontoverbal.java` shows you how to retrieve the subclasses of a given class in the context of some given ontology and call a method whose purpose is to generate the text. Use the method as guide to assist in executing the following task:
   (a) Complete the `verbaliseDisjointClases`, `verbaliseEquivalentClasses`, and `verbaliseInstances` methods in `Ontoverbal.java`

■

We have now seen how to collect the classes associated with each `AxiomClassType` and how to create high-level methods for verbalising each type. We now turn to creating the method to be used to generate text for each simple axiom type.

Exercise 5.5 Carry out the following tasks:

1. Modify the `verbaliseSimpleAxioms` method in `OntoVerbal.java` by introducing code to verbalise the argument $owlClass$ and the $exprs$ of type `AxiomClassType.Superclass` using the following template:
   (a) A(n) [$owlClass$] is a kind of [$superClassList$]
   where $superClassList$ is generated using the previously created method `listClasses(exprs, true)`
2. Modify the `verbaliseSimpleAxioms` method in `OntoVerbal.java` by introducing code to verbalise the argument $owlClass$ and the $exprs$ of type `AxiomClassType.Subclass` using the following templates:
   (a) A(n) [$owlClass$] is a kind of [$subclassList$]
   (b) More specialised kinds of [$owlClass$] are [$subclassList$]
   where $subclassList$ is generated using the `listClasses(exprs,true)` method
3. Modify the `verbaliseSimpleAxioms` method in `OntoVerbal.java` by introducing code to verbalise the $owlClass$ and $exprs$ of type `Equivalentclass` using the following templates
   (a) A(n) [$owlClass$] is defined as [$equivClassList$]
   where $equivClassList$ is generated using $listClasses(exprs, true)$ method
4. Modify the `verbaliseSimpleAxioms` method in `OntoVerbal.java` by introducing code to verbalise the $owlClass$ and $exprs$ of type `Disjointclass` using the following templates
   (a) a(n) [$owlClass$] is different from [$disjointClassList$]

where $disjointClassList$ is generated using $listClasses(exprs, true)$ method

5. Modify the `verbaliseSimpleAxioms` method in `OntoVerbal.java` by introducing code to verbalise the $owlClass$ and $exprs$ of type `Individual` using the following templates
   (a) a(n) [$owlClass$] has an instance [$individualList$]
   (b) a(n) [$owlClass$] has instances [$individualList$]

   where $individualList$ is generated using $listClasses(exprs, true)$ method

■

We have seen how to verbalise individual simple axioms. We now move on to combining them for purposes of generating a single paragraph.

Exercise 5.6 Execute the following tasks:

1. Modify the `generateAggregateSentence` method in `OntoVerbal.java` by introducing code to aggregate the sentences produced by the simple axioms using the following template:
   (a) [$superclsSent$] [$subclsSent$] [$equivalentclsSent$]
   In addition, [$disjointclsSent$] Also, [$individualSent$]

   where $^*Sent$ are the verbalised sentences associated with each type of simple axiom. In the above template, we use the abbreviations Sent = Sentence and Cls = Class. As such, $superclsSent$ refers to the sentence resulting from the verbalising of superclasses and $subclsSent$, $equivalentclsSent$, $disjointclsSent$, and $individualSent$ are the sentences pertaining to the subclasses, equivalent classes, disjoint classes, and individuals respectively.

■

When all the methods/tasks are implemented and the code is fed the `American` class from the pizza ontology (i.e., `http://www.co-ode.org/ontologies/pizza/pizza.owl#American`) from the pizza ontology, then it should produce the following text:

```
A(n) American is a kind of DomainConcept, Thing, NonVegetarianPizza, InterestingPizza,
MeatyPizza, CheeseyPizza, NamedPizza, Food, and Pizza.
More specialised kinds of American are IceCream, and  CheeseyVegetableTopping.
A(n) American is defined as American.
In addition, a(n) American is different from a(n) VegetableTopping, a(n) CheeseyVegetableTopping,
a(n) Spiciness, a(n)PizzaBase, a(n) ParmaHamTopping, a(n) RocketTopping, a(n) Rosa,
a(n) QuattroFormaggi, a(n) RosemaryTopping, a(n) PrawnsTopping, a(n) PrinceCarlo,
a(n) GorgonzolaTopping, a(n) TobascoPepperSauce, a(n) MeatTopping, a(n) FruitTopping,
a(n) CheeseTopping, a(n) AnchoviesTopping, a(n) MushroomTopping, a(n) MixedSeafoodTopping,
a(n) Napoletana, a(n) SweetPepperTopping, a(n) GarlicTopping, a(n)  SultanaTopping,
a(n) HotSpicedBeefTopping, a(n) Mushroom, a(n) SloppyGiuseppe, a(n) PetitPoisTopping,
a(n) SauceTopping, a(n) HerbSpiceTopping, a(n) AmericanHot, a(n) PepperTopping, a(n)
FourCheesesTopping, a(n) GreenPepperTopping, a(n) HamTopping, a(n) Fiorentina, a(n) Caprina,
a(n) DeepPanBase, a(n) CaperTopping, a(n) UnclosedPizza, a(n) TomatoTopping, a(n)
SlicedTomatoTopping, a(n) ParmesanTopping, a(n) ArtichokeTopping, a(n) SpicyPizza,
a(n) ThinAndCrispyBase, a(n) LeekTopping, a(n)Mild, a(n) Parmense, a(n) CajunSpiceTopping,
a(n) FishTopping, a(n) NutTopping, a(n) SpicyTopping, a(n) Capricciosa, a(n) Cajun,
a(n) PeperonataTopping, a(n) PeperoniSausageTopping, a(n) Margherita, a(n) Giardiniera,
a(n) FruttiDiMare, a(n) RedOnionTopping, a(n) MozzarellaTopping, a(n) LaReine, a(n) Soho,
a(n) IceCream, a(n) OliveTopping, a(n) VegetarianTopping, a(n) OnionTopping, a(n)
AsparagusTopping, a(n) Hot, a(n) ChickenTopping, a(n) SpicyPizzaEquivalent, a(n) PizzaTopping,
a(n) FourSeasons, a(n) VegetarianPizza, a(n) Medium, a(n) PolloAdAstra, a(n)
SundriedTomatoTopping, a(n) JalapenoPepperTopping, a(n) GoatsCheeseTopping,
a(n) SpinachTopping, a(n) HotGreenPepperTopping, a(n) PineKernels, a(n)
VegetarianPizzaEquivalent2, a(n) VegetarianPizzaEquivalent1, a(n) Siciliana, and Veneziana.
```

The above text is readable, but it does not look completely natural. Let us now consider how to improve the quality of the verbaliser's output text.

R In the verbalised text, we rely on the name of the class found in the URI. Specify one alternate location resource/location you can use to retrieve the name of each class.

5.3 Basic isiZulu Verbaliser

The isiZulu verbaliser that was created by Keet et al. [23] was designed to be the first natural language interface for ontologies that supports an African language. IsiZulu is member of the Nguni language group and it is primarily spoken in Southern Africa. Like other Niger-Congo B languages, it has complex morphology and its agreement system makes verbalisation challenging. We demonstrate the challenge of verbalising axioms in the language by using output that is produced by the OWLSIZ verbaliser [31]. When the verbaliser is fed the axiom subClassOf(herb plant), it generates the question Ingabe **lo**nke ihebhu **ling**umuthi? “Is every herb a plant?" where underlined sections are the isiZulu class names. The sections that are highlighted in bold are automatically generated and may change based on the value of the classes. Technically, the values change as a result of the noun class of the OWL class’ lexical item (see [38], for instance, for additional details on the isiZulu noun class system).

The isiZulu verbaliser is multi-platform as its written in Python and has the ability to take supported axioms and transform them to produce text as shown in Figure 5.1.

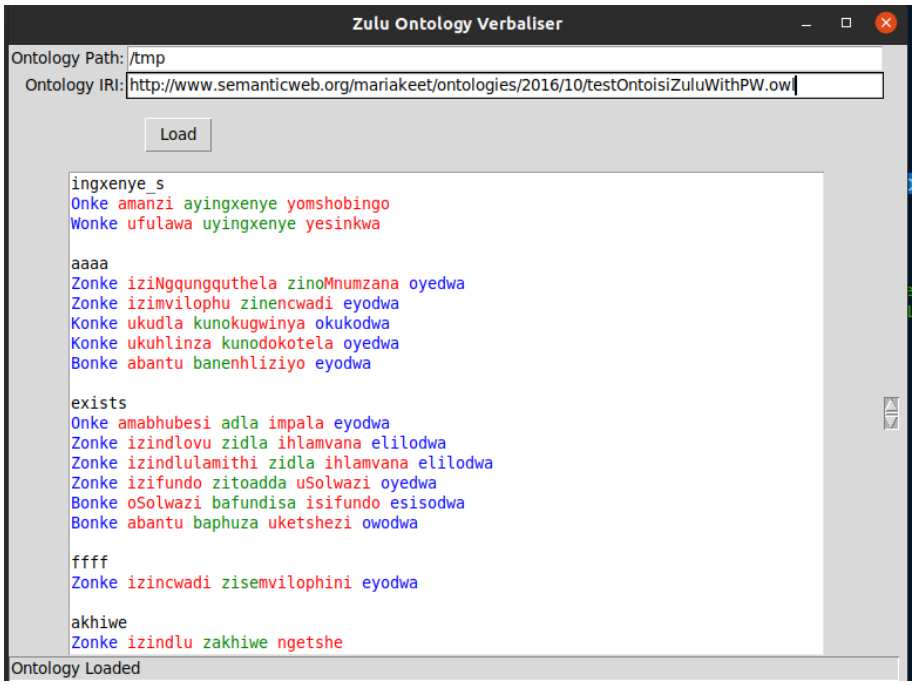


Figure 5.1: Sample of texts generated by the isiZulu verbaliser

For this tutorial, we will reproduce the portion of the IsiZulu English verbaliser that supports simple axioms. Unlike the case of OntoVerbal, we say that an axiom is simple in the context of the isiZulu verbaliser if it is one of the following $A \sqsubseteq B$, $A \sqsubseteq \neg B$, $A \sqsubseteq \exists R.B$, $A \sqsubseteq \neg\exists R.B$, and $A \sqcap B$ where A and B can only be classes and not class expressions. We do not include other supported types, as detailed in [21].

In order to get started, execute the following tasks.

Exercise 5.7 Carry out the following tasks:

1. Download the zip file located at `https://bit.ly/3zKiEms`
2. Download the OWL file located at `https://bit.ly/3Qn1GSi`
3. Unzip the downloaded file and add all files located under `reproZuluVerb/lib` to your Java classpath
4. Open the Java verbaliser in your IDE and change line 18 in `src/main/Main.java` to include the path to the downloaded OWL file
5. Run the verbaliser

■

After executing the above steps, you should not see any errors. Specifically, you should get the output from verbalising some `SubClassOf` axioms since the template specified in `src/templates/template-subclass-1.ttl` supports some of them. We now turn to creating our first template using ToCT [30].

Exercise 5.8 1. Consider the following templates (labelled (a) and (b)) for verbalising axioms of the kind $A \sqsubseteq B$. Each of the two templates has two elements, the first element is a slot that takes the noun associated with the subclass, and the second element is a polymorphic word whose first item is a fixed affix that either takes the values `y-` or `ng-` and the second element is a slot that takes the noun associated with the super class. If the file located at `src/templates/template-subclass-1.ttl` captures template (a), complete `src/templates/template-subclass-2.ttl` to capture template (b). Uncomment lines 83-84 after creating the template in `src/main/verbaliser/ZuluVerbaliser.java`

   (a) [**subclassNoun**] y[**superclassNoun**]

   (b) [**subclassNoun**] ng[**superclassNoun**]

■

Since we have seen how to create a basic template using ToCT, with an existing template as a guide, we now turn to create other complex templates.

Exercise 5.9 Carry out the following tasks.

1. Complete the `template-disjoint.ttl` file by using ToCT to create the following template where $C1$ and $C2$ are classes, `Conc` refers to a concord, and `affix` refers to a $PolymorphicWord$ as defined in ToCT [30]:

$$\underbrace{\underbrace{[SubjectConc]}_{concord}\underbrace{o}_{affix}}_{polymorphicword}\ \underbrace{[pl(C1)]}_{slot}\ \underbrace{[NegativeConc]}_{concord}\ \underbrace{\underbrace{SubjectConc}_{concord}\underbrace{o}_{affix}\underbrace{na}_{affix}}_{polymorphic\ word}\ \underbrace{[C2]}_{slot}$$

2. Complete the `template-existential.ttl` file by using ToCT to create the following template where the forth item `reliesOn` the third item:

$$\underbrace{\underbrace{[SC]}_{concord}\underbrace{o}_{affix}}_{polymorphic\ word}\ \underbrace{[pluralise(C1)]}_{slot}\ \underbrace{\underbrace{[conjugate(op)]}_{slot}\underbrace{a}_{affix}}_{polymorphic\ word}\ \underbrace{[C2]}_{slot}\ \underbrace{\underbrace{[RC]}_{concord}\underbrace{SC}_{concord}\underbrace{odwa}_{affix}}_{polymorphic\ word}$$

   where SC is the SubjectConc, and RC the RelativeConc.

3. Complete the `template-negatedexistential.ttl` file by using ToCT to

create the following template where the forth item `reliesOn` the third item.:

$$\underbrace{\underbrace{[SC]}_{concord}\ \underbrace{o}_{affix}}_{polymorphic\ word}\ \underbrace{[pluralise(C1)]}_{slot}\ \underbrace{\underbrace{[conjugate(op)]}_{slot}\ \underbrace{i}_{affix}}_{polymorphicword}\ \underbrace{[C2]}_{slot}\ \underbrace{\underbrace{[RC]}_{concord}\ \underbrace{SC}_{concord}\ \underbrace{odwa}_{affix}}_{polymorphic\ word}$$

where SC is the SubjectConc, and RC the RelativeConc.

4. Complete the `template-conjuct.ttl` file by using ToCT to create the following template:

$$\underbrace{[firstClass]}_{slot}\ \underbrace{\underbrace{na}_{affix}\ \underbrace{[secondClass]}_{slot}}_{polymorphic\ word}$$

5. Once all each of the templates have been created, uncomment lines 113-114, 144-145, 178-179, and 204-205. Run the verbaliser.

■

Now that we have been able to verbalise all the supported axioms type, we now turn to focus on some improvements that can be made. Execute the following tasks:

Exercise 5.10 Carry out the following tasks.

1. All previous tasks' axioms are in `testOntoisiZuluWithPW.owl`. In order to extend verbaliser to handle other ontologies, modify the `verbalise` method found `src/mai/ZuluVerbaliser.java` to use an external lexicon instead of using each class' URI.
2. Create a test ontology with simple axioms whose URIs that are not in isiZulu and associate isiZulu lexical items for each class and relation.
3. Change line 14 in `src/main/Main.java` to use the full path to your newly created OWL file
4. Run the verbaliser

■

This tutorial is written by Zola Mahlaza and was developed for the NLG tutorial at JOWO'22. The work was funded in part by the MoreNL project funded by the NRF under grant number 120852.

II Part Two: Group Assignments

Besides the exercises to engage with the material, there are two assignments. They have a strong integrative flavour to it and will take up more time than the exercises and tutorials. You are expected to apply as much as you have learned, and look up more information online, be they tools, methods, other ontologies, or scientific papers. Reuse of ontologies is permitted, but make sure to cite them properly (at least a URL, but also try to find the corresponding reference paper) and state where, how, and why it is reused.

The assignments in this part are described in such as way that they may serve as a draft description of the actual assignment. For instance, there are hard deadlines for hand-in, notwithstanding that I know the material will be better with more work put into it and ontologies are never really 'finished'. I consider both assignments to be group assignments, even though they could be done individually, and I assign people to groups and topics if they did not make groups themselves by a given date. If the practical assignment is scheduled for the end of Part I rather than for hand in at the end of Part II (of the accompanying textbook), then remove the requirements on Part II topics form the description below.

Regarding the project assignment: most topics can be reused across years, and some of them can be done by more than one group as it invariable ends up in different results. Some of the previously listed projects that were carried out in earlier instalments have some material available online, which gives an indication of success (i.e., examples of projects that received top marks). One such mini-project from the OE course at the University of Havana (2010) resulted in OntoPartS—a tool and creative extension to the part-whole taxonomy of [18]—that was subsequently formalised, written up, and published [19]. Another one is the OWL Classifier[1], which was developed as part of one of the mini-projects of the 2016 OE course at the University of Cape Town, and subsequently used toward conflict resolution in [22], and the BFO Classifier [3, 5]. These topics are not listed in Chapter 7 anymore, for the obvious reasons.

[1] https://github.com/muhummadPatel/OWL_Classifier

6. Develop a domain ontology

The aim of this practical assignment is for you to demonstrate what you have learned about the ontology languages, top-down and bottom-up ontology development, and methods and methodologies, and experiment with how these pieces fit together.

You can do this assignment in groups of two or three students. It should be mentioned in the material you will hand in who did what.

Tasks

1. Choose a subject domain of interest for which you will develop a domain ontology. For instance, computers, tourism, furniture, some hobby you may be familiar with (e.g., diving, dancing), or some other subject domain you happen to be knowledgable about (or know someone who is).
2. Develop the domain ontology in the best possible way. You are allowed to use any resource you think is useful, be it other ontologies, non-ontological resources, tools, domain experts, etc.. If you do so (and you are encouraged to do so), then make sure to reference them in the write-up.
3. Write about 2-3 pages (excluding figures or screenshots) summarising your work. This can include—but is not limited to—topics such as an outline of the ontology, why (or why not) you have used a foundational ontology (if so, which, why), if you could reuse a top-domain or other subject domain ontology, which non-ontological resources you have used (if any, and if so, how), if you encountered subject domain knowledge that should have been in the ontology but could not be represented due to the limitations of OWL, or perhaps a (real or imagined) purpose of the ontology and therefore a motivation for some OWL fragment, any particular reasoning services that was useful (and how and why, which deductions did you have or experimented with), any additional tools used.

Material to hand in

Send in/upload to the course's CMS the following items:

1. The OWL file of your ontology;
2. Imported OWL ontologies, if any;
3. The write up in pdf.

Assessment

1. Concerning the ontology: quality is more important than quantity. An ontology with more advanced constraints and appropriate reuse of foundational or general ontologies is a better illustration of what you have learned than a large bare taxonomy.
2. Concerning the ontology: it will be checked on modelling errors in the general sense (errors such as is-a vs. part-of, class vs. instance, unsatisfiable classes). Regarding the subject domain itself, it will be checked only insofar as it indicates (mis)understanding of the ontology language or reasoning services.
3. Concerning the write up: a *synthesis* is expected, not a diary. For instance, "We explored a, b, and c, and b was deemed to be most effective because blabla" would be fine, but not "We tried a, but that didn't work out, then we had a go at b, which went well, then we came across c, tried it out of curiosity, but that was a dead end, so we went back to b.". In short: try to go beyond the 'knowledge telling' and work towards the so-called *knowledge transformation*.
4. Concerning the write up: while a brief description of the contents is useful, it is more important to include something about the *process* and *motivations* how you got there, covering topics such as, but not limited to, those mentioned under Tasks, item 3 (and recollect the aim of the assignment—the more you demonstrate it, the better).

Notes

In random order:

1. The assignment looks easy. It isn't. If you start with the development of the ontology only the day or so before the deadline, there is an extremely high probability that you will fail this assignment. Your assignment will be of a higher quality if you start thinking about it some 2 weeks before the deadline, and the actual development at most one week before the deadline, and spread out the time you are working on it.
2. If you use non-English terms for the classes and properties, you should either add the English in the annotations (preferred), else lend me a dictionary if it is in a language I do not speak.
3. Use proper referencing when you use something from someone else, be it an ontology, other reused online resources (including uncommon software), textbooks, articles etc. Not doing so amounts to plagiarism. See Appendix B for guidelines.
4. Spell checkers tend to be rather useful tools.
5. Some of the mini-project topics can benefit from an experimental ontology that you know in detail, which you may want to take into consideration when choosing a subject domain or purpose so that your ontology might be reused later on.

7. Mini-project assignment

The project assignment aims to let you explore in more detail a specific subtopic within ontology engineering. There is enough variation in the list of topics below to choose either a software development project, literature review, experimentation, and occasionally a bit of research. The two subsections below could be used as is or function as a template for one's own specification of constraints and more or less project topics.

7.1 Suggested set-up of the assignment

The aim of this assignment is to work in a small group (2-4 students) and to investigate a specific theme of ontology engineering. The topics are such that either you can demonstrate the *integration of various aspects* of ontologies and knowledge bases or *going into quite some detail on a single topic*, and it can be either theory-based, implementation-focussed, or a bit of both.

Possible topics to choose from will be communicated in week x, and has to be chosen and communicated to me (topic + group members) no later than in week x+1, else you will be assigned a topic and a group.

Tasks

1. Form a group of 2-4 people and choose a topic, or vv.: choose a topic and find other people to work with. It should be mentioned in the material you will hand in who did what.
2. Carry out the project.
3. Write about 4-6 pages (excluding figures or screenshots) summarising your work. The page limit is flexible, but it surely has to be < 15 pages in total.
4. Give a presentation of your work during the last lecture (10 minutes presentation, ± 5 minutes discussion). *Everyone must attend this lecture.*

Material to hand in

You have to upload/email the following items:

1. The write up.
2. Additional material: this depends on the chosen topic. If it is not purely paper-based, then the additional files have to be uploaded on the system (e.g., software, test data).
3. Slides of the presentation, if any.

Note that the deadline is after the last lecture so that you have the option to update your material for the mini-project with any feedback received during the presentation and discussion in class.

Assessment

1. Concerning the write up: a *synthesis* is expected, not a diary. For instance, "We explored a, b, and c, and b was deemed to be most effective because blabla" would be fine, but not "We tried a, but that didn't work out, then we had a go at b, which went well, then we came across c, tried it out of curiosity, but that was a dead end, so we went back to b.". In short: try to go beyond the 'knowledge telling' and work towards the so-called *knowledge transformation*.
2. Concerning the write up: use proper referencing when you use something from someone else, be it an ontology, other reused online resources (including uncommon software), textbooks, articles etc. Not doing so amounts to plagiarism, which has a minimum penalty of obtaining a grade of 0 (zero) for the assignment (for all group members, or, if thanks to the declaration of contribution the individual can be identified, then only that individual), and you will be recorded on the departmental plagiarism list, if you are not already on it, and further steps may be taken.
3. The presentation: respect the time limit, coherence of the presentation, capability to answer questions.
4. Concerning any additional material (if applicable): if the software works as intended with the given input, presentability of the code.
5. Marks will be deducted if the presentation or the write-up is too long.

Notes

Things you may want to take into consideration (listed in random order):

1. LaTeX is a useful typesetting system (including beamer for presentation slides), has a range of standard layouts as well as bibliography style files to save you the trouble of wasting time on making the write up presentable, and generates a pdf file that is portable[1]. This is much less so with MS Word; if you use MS Word nevertheless, please also include a pdf version of the document.
2. Regarding the bibliography: have complete entries. Examples of referencing material (for conference proceedings, books and book chapters, and journal articles) can be found in the scientific papers included in the lecture notes' bibliography, scientific literature you consult, LaTeX documentation, and Appendix B of this workbook.
3. Spell checkers tend to be rather useful tools.

[1] in case you are not convinced: check, e.g., `http://openwetware.org/wiki/Word_vs._LaTeX`

4. One or more of the domain ontologies developed in the previous assignment may be suitable for reuse in your chosen topic.
5. Some of the mini-projects lend themselves well for extension into an Honours/-Masters project, hence, could give you a head-start.

7.2 Topics

You can select one of the following topics, or propose one of your own. If the latter, you first will have to obtain approval from you lecturer (if you are doing this assignment as part of a course).

The topics are listed in random order, have different flavours or emphases; e.g., more of a literature review project, or programming, or theory, or experimentation. The notes contain pointers to some more information, as does the corresponding section in the book.

Some descriptions may seem vague in that it still offers several possibilities with more or less work and/or more or less challenging activities; this is done on purpose. You should narrow it down as you see fit—bearing in mind the aims of the mini-project and the number of people in your group—and be able to justify why if asked to do so. If you choose a topic that involves processing OWL files, then try to use either the OWL API (for Java-based applications) or Owlready (Python), the Jena Toolkit (or similar), or OWLink, rather than spending time reinventing the wheel.

You probably will find out it's not as easy as it looked like initially, which may make it tempting wishing to change, but that also holds for the other topics, and by changing topic you very likely have lost too much time to bring the other one to passable completion.

Some topics can be done by more than one group simultaneously, after which it can be interesting to compare what came out of, such as topic 8.

1. There are some 15 working automated reasoners for various Description Logics (e.g., HermiT, FaCT++, TrOWL, Pellet, Racer, ELK, MoRE, Quonto, Quest). Conduct a comparison with a sensible subset, including their performance on a set of ontologies you select or create artificially, and possibly also along the direction of examining the effect of the use of different features in the ontology. Some ideas may be gleaned from the "short report" in the proceedings of the Ontology Reasoner Evaluation workshop[2].
2. Write a literature review about ontology-driven NLP or ontology-enhanced digital libraries (minimum amount of references to consider depends on group size and whether this is an undergraduate or postgraduate course).
3. Set up an OBDA system (that is not one of the samples of the Ontop website). You are advised to either reuse your ontology from the practical assignment or take an existing one. Regarding data, you may want to have a look at the GitHub list of publicly available data sets[3], or you can create a mock database.
4. Set up and carry out an experiment converting a set of ontologies to conceptual data models, executing substitutions from Humusa [4] in different order and assessing the effects on the generated output.

[2] `http://ceur-ws.org/Vol-1015/`, and there have been subsequent editions of the ORE workshops.
[3] `https://github.com/caesar0301/awesome-public-datasets`

5. Create a decision diagram for GFO or YAMATO *classes*, alike the D3 of FORZA [20] (it also has the link to the code and a 'lessons learnt' paragraph on designing that diagram) or the BFO classifier [3], and validate it (e.g., by checking whether extant ontologies linked to GFO or YAMATO were linked correctly), or revise D3 and evaluate the effect on domain ontology alignment.
6. Create a decision diagram for BFO/DOLCE/GFO/YAMATO *object properties* (alike the D3 of FORZA [20] or the BFO classifier [3] for classes), and validate it.
7. Write a literature review on where and how LLMs could assist in the ontology development processes. There are a few early papers about it from 2023 and 2024, but the field moves quickly, and you are expected to also include more recent references.
8. There are many works on criteria to evaluate the quality of an ontology or an ontology module (e.g., [32, 35, 39]). Provide an overview and apply it to several of your classmates' ontologies developed in the practical assignment and/or ontologies taken from online repositories. In a larger group, this may also include evaluating the evaluation strategies themselves.
9. Take a thesaurus represented in SKOS[4] and convert that into a real ontology (i.e., not simply only converting into OWL). There are older manual and semi-automated approaches (see e.g., [26, 37]) and likely newer ones as well.
10. Compare and contrast tools for ontology visualisation (i.e., their graphical renderings in, e.g., Ontograf, OWLGrEd, SOVA, and so on).
11. Redesign the OWL Classifier[5] to work both with the latest JDK and provide better explanations.
12. Extend ONSET [24] with UFO, be it the paper-based one or the OWL version.
13. Create a Protégé plugin or stand-alone tool for 'learning' competency questions from queries posed in the DL query tab (i.e., from DL query to natural language).
14. Create an ontology editor (standalone or as a plugin), where a modeller can write DL axioms (in LaTeX format or with buttons) and it converts it into OWL (RDF/XML exchange syntax).
15. Design your own ontology language. A process and additional information may be found in [8].
16. Conduct a literature review on, and, where possible, test, several so-called 'non-standard reasoning' reasoners. Among others, you could consult one of the least-common subsumer papers [33], and/or reasoner-mediated ontology authoring tools [6, 7], among others.
17. Consider bottom-up ontology development starting from a conceptual data model (e.g., UML class diagrams, EER). Find a way to generate an OWL file from it such that it provides candidate classes, object properties, and constraints for an actual ontology. You may want to take one of the freely available conceptual modelling tools (e.g., ArgoUML) and use that serialisation for the transformations. There are a few tools around that try this already, but they have incomplete coverage (at best).
18. Implement the RBox reasoning service and demonstrate correctness of implementation in the software.

[4] see, e.g., the list at `http://code.google.com/p/lucene-skos/wiki/SKOSThesauri`, but there are others you may wish to consider

[5] `https://people.cs.uct.ac.za/~mkeet/OEbook/OEsoftware.html#OWLclass`

19. OWL does not consider time and temporal aspects. To address this shortcoming, several 'workarounds' as well as temporal description logics have been proposed. Compare and contrast them on language features and modelling problems and reasoning.
20. Provide a state of the art on research into competency questions.
21. Represent DOLCE—or another ontology that has indicated to be needing more than OWL expressiveness—in DOL on OntoHub; e.g., by adding those axioms in a module and link them to the OWL file into a network of ontologies.

Part Three: Answers to exercises

8. Answers to textbook exercises

8.1 Answers Chapter 2

Answer Exercise 2.1. Indicative descriptions:

(a) All lions are mammals.
(b) Each PC has as part at least one CPU and at least one Monitor connected
(c) Proper part is asymmetric.

Answer Exercise 2.2.

(a) $\forall x(Car(x) \rightarrow Vehicle(x))$
(b) $\forall x(HumanParent(x) \rightarrow \exists y(haschild(x,y) \wedge Human(y)))$
(c) $\forall x,y(Person(x) \wedge Course(y) \rightarrow \neg(lecturerOf(x,y) \wedge studentOf(x,y)))$

Answer Exercise 2.3.

(b) There exists a node that does not participate in an instance of R, or: it does not relate to anything else: $\exists x \forall y.\neg R(x,y)$.
(c) $\mathcal{L} = \langle R \rangle$ as the binary relation between the vertices. Optionally, on can add the vertices as well. Properties:
R is symmetric: $\forall xy.R(x,y) \rightarrow R(y,x)$.
R is irreflexive: $\forall x.\neg R(x,x)$.
If you take into account the vertices explicitly, one could say that each note participates in at least two instances of R to different nodes.

Answer Exercise 2.4.

(a) R is reflexive (a thing relates to itself): $\forall x.R(x,x)$.
R is asymmetric (if a relates to b through relation R, then b does not relate back to a through R): $\forall xy.R(x,y) \rightarrow \neg R(y,x)$.
(b) See the example on p21 of the lecture notes.

Answer Exercise 2.6. Note: there may be more than one solution; only one is given. Also note

that a problem of natural language is that it can be imprecise.

1. $\forall x(Lion(x) \rightarrow Animal(x))$
2. $\forall x(Professor(x) \rightarrow \exists y(teaches(x,y) \wedge Course(y)))$
3. $\forall x(Human(x) \rightarrow \exists y,z(eat(x,y) \wedge Fruit(y) \wedge eat(x,z) \wedge Cheese(z)))$ (note: twice the 'eat', with y and z, not "$fruit(y) \wedge cheese(y)$", for that refers to the objects that are *both*, which don't exist)
4. $\forall x(Animal(x) \rightarrow Herbivore(x) \vee Carnivore(x))$
5. $\forall x,y,z(hasmother(x,y) \wedge hassister(y,z) \rightarrow hasaunt(x,z))$ (or with $\leftrightarrow$ and/or with the composition operator $\circ$)
6. $\forall x(Pap(x) \rightarrow \neg Pizza(x))$
7. $\forall x \exists y((Person(x) \wedge worksfor(x,y) \wedge Company(y) \rightarrow Employee(x)))$
8. $\forall x,y(manages(x,y) \leftrightarrow Manager(x))$
9. $\forall x(Fly(x) \rightarrow \exists^{=2} y(haspart(x,y) \wedge Eye(y)))$ (note: this is shorthand notation...)
10. $\forall x,y,z(life(x,y) \wedge life(x,z) \rightarrow y = z)$
11. $\forall x,y(participation(x,y) \rightarrow PhysicalObject(x) \wedge Process(y))$
12. $\forall x,y(hasPart(x,y) \rightarrow partOf^{-}(x,y))$
13. $\forall x,y(connection(x,y) \rightarrow connection(y,x))$
14. Vehicles: combine the pattern for the 'or' from 4 with the disjoints of 6, for the vehicles.
15. $\exists x(Snail(x) \wedge slow(x))$ but not this is suboptimal (recall the apple & green; similar story here)
16. $\forall x(Patient(x) \rightarrow \exists y,z(registration(x,y,z) \wedge Hospital(y) \wedge Weekday(z)))$
17. Note: this requires either a temporal 'extension' or necessity (beyond the current scope). Let's take temporal, for which we introduce a notions of time, t, that quantifies over time points only (for simplicity, and linear time): $\forall x,t(Student(x,t) \rightarrow \exists t' \neq t(\neg Student(x,t')))$

8.2 Answers Chapter 3

Answer Exercise 3.1. Note: there may be more than one solution; only one is given. Also note that these axioms are agnostic about particular fragments, and we don't consider datatypes.

1. Lion $\sqsubseteq$ Animal
2. Professor $\sqsubseteq \exists$teaches.Course
3. Human $\sqsubseteq \exists$eat.Fruit $\sqcap \exists$eat.Cheese
4. Animal $\sqsubseteq$ Herbivore $\sqcup$ Carnivore
5. hasMother $\circ$ hasSister $\sqsubseteq$ aunt (or with $\equiv$, i.e., that the notion of 'aunt' is defined by it)
6. Pap $\sqsubseteq \neg$Pizza (or with 'bottom': Pap $\sqcap$ Pizza $\sqsubseteq \bot$)
7. Person $\sqcap \exists$worksFor.Company $\sqsubseteq$ Employee
8. $\forall$manages.$\top \equiv$ Manager
9. Fly $\sqsubseteq = 2$ hasPart.Eye
10. lazy option: Func(life), less lazy, as part of another axiom, ≤ 1 life or ≤ 1 life.$\top$
11. lazy option: Participation $\sqsubseteq$ PhysicalObject $\times$ Process,
    and in full: $\exists$participation $\sqsubseteq$ PhysicalObject and $\exists$participation$^{-} \sqsubseteq$ Process
12. hasPart $\sqsubseteq$ partOf^{-}
13. lazy option (in $\mathcal{SROIQ}$): Sym(connection)
14. Vehicles: combine the 'or' from 4 with the disjoints of 6.
15. Not easily represented in DLs (rework it with some subtype of snails for which it always holds)
16. This can be represented in the $\mathcal{DLR}$ family of Description Logics, but not in most DLs and not in OWL either (which has only binaries—we'll return to this in the second part of the module)

17. This can be represented in several temporal description logics, using temporal operators, alike Student $\sqsubseteq \diamond^* \neg$Student with the diamond-shape the temporal counterpart of $\exists$ and with $*$, this reads as 'sometime'. More about this can be found in the 'advanced topics'.

Answer Exercise 3.2.

(a) Rewrite (Eq. 3.27) into negation normal form:
$Person \sqcap \forall eats.Plant \sqcap (\neg Person \sqcup \neg \forall eats.(Plant \sqcup Dairy))$
$Person \sqcap \forall eats.Plant \sqcap (\neg Person \sqcup \exists \neg eats.(Plant \sqcup Dairy))$
$Person \sqcap \forall eats.Plant \sqcap (\neg Person \sqcup \exists eats.(\neg Plant \sqcap \neg Dairy))$
So our initial ABox is:
$S = \{(Person \sqcap \forall eats.Plant \sqcap (\neg Person \sqcup \exists eats.(\neg Plant \sqcap \neg Dairy)))(a)\}$

(b) Enter the tableau by applying the rules until either you find a completion or only clashes.
($\sqcap$-rule): $\{Person(a), \forall eats.Plant(a), (\neg Person \sqcup \exists eats.(\neg Plant \sqcap \neg Dairy))(a)\}$

($\sqcup$-rule): (i.e., it generates two branches)

(1) $\{Person(a), \forall eats.Plant(a), (\neg Person \sqcup \exists eats.(\neg Plant \sqcap \neg Dairy))(a), \neg Person(a)\}$ <clash!

(2) $\{Person(a), \forall eats.Plant(a), (\neg Person \sqcup \exists eats.(\neg Plant \sqcap \neg Dairy))(a), \exists eats.(\neg Plant \sqcap \neg Dairy)(a)\}$
($\exists$-rule): $\{Person(a), \forall eats.Plant(a), (\neg Person \sqcup \exists eats.(\neg Plant \sqcap \neg Dairy))(a), \exists eats.(\neg Plant \sqcap \neg Dairy)(a), eats(a,b), (\neg Plant \sqcap \neg Dairy)(b)\}$
($\sqcap$-rule): $\{Person(a), \forall eats.Plant(a), (\neg Person \sqcup \exists eats.(\neg Plant \sqcap \neg Dairy))(a), \exists eats.(\neg Plant \sqcap \neg Dairy)(a), eats(a,b), (\neg Plant \sqcap \neg Dairy)(b), \neg Plant(b), \neg Dairy(b)\}$
($\forall$-rule): $\{Person(a), \forall eats.Plant(a), (\neg Person \sqcup \exists eats.(\neg Plant \sqcap \neg Dairy))(a), \exists eats.(\neg Plant \sqcap \neg Dairy)(a), eats(a,b), (\neg Plant \sqcap \neg Dairy)(b), \neg Plant(b), \neg Dairy(b), Plant(b)\}$ <clash!

(c) $\mathcal{T} \vdash Vegan \sqsubseteq Vegetarian$? yes

8.3 Answers Chapter 4

Answer Exercise 4.2. Use a property chain.

Answer Exercise 2.

(c) Expressivity: $\mathcal{ALN}$, in OWL DL. Not in OWL Lite and OWL 2 EL, QL, RL because of the minCardinality.

(e) Expressivity: $\mathcal{ALQ}$, in OWL 2 DL. Not in OWL DL anymore because of the qualified number restriction.

Answer Exercise 4.8. As historical note: the original exercise came from SSSW 2005[1], which was at the time of OWL 1. The answers here and the next ones have been updated taking into account OWL 2.

(a) The description is sufficiently vague that it may be either of
JointHonsMathsComp $\equiv \exists$takes.MathsModule $\sqcap \exists$takes.ComputerScienceModule
JointHonsMathsComp $\equiv$ Student $\sqcap \exists$takes.MathsModule $\sqcap$

[1] http://owl.man.ac.uk/2005/07/sssw/university.html

$\exists$takes.ComputerScienceModule

In any case, observe it is not "takes.(MathsModule $\sqcap$ ComputerScienceModule)"; see Example 5.3 for further explanation.

(b) SingleHonsMaths $\equiv$ Student $\sqcap \exists$takes.MathsModule $\sqcap \forall$takes.ComputerScienceModule. This is also called 'closing' the axiom.

So: yes, this is possible. A possible solution is shown in `university1.owl`

Answer Exercise 4.9. Deductions: they are undergrad students, and students 2, 4, and 7 are JointHonsMathsComp. Student7 is one, because it is an instance of $\exists$takes.MathsModule and has a property assertion that s/he takes CS101.

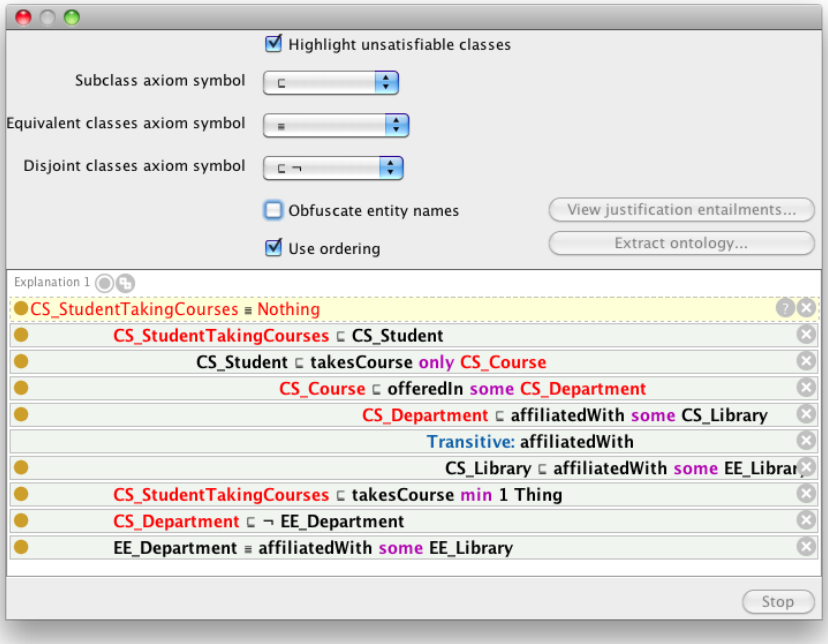
No student is a SingleHonsMaths, despite that, e.g., Student3 has declared taking two math modules. This is due to the Open World Assumption: student3 may well take other courses that we don't know of as of yet, so it is not guaranteed in all possible worlds that student3 takes *only those two* math courses.

Answer Exercise 4.10. This can now be done with OWL 2, for it permits qualified number restrictions; see `university2.owl`

(a) This poses no problem, because of the no unique name assumption: it will infer that CS101 and CS102 must be the same object, so then student 9 still takes 2 courses and all is well; see `university2.owl`.

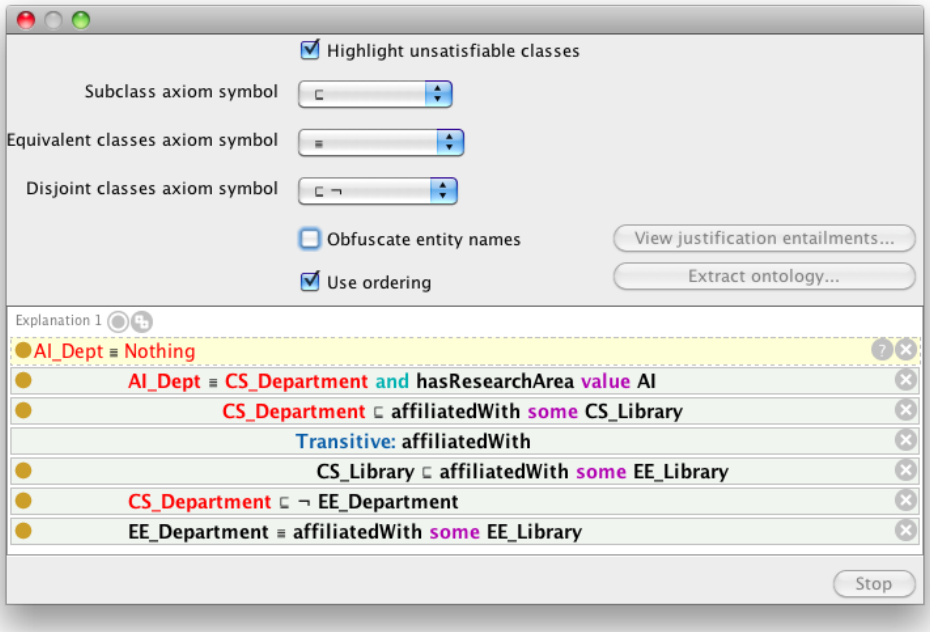
(b) This does pose a problem, because each of the three courses mentioned are member of their respective classes that are disjoint, so they must be distinct individuals, and thus we obtain a violation in cardinality restrictions (=2 vs =3), and therewith the ontology has become inconsistent.

Answer Exercise 4.11. Let us first have a look randomly at a deduction and its explanation (click on the "?" right from the deduction in Protégé) as a first step toward figuring out why so many classes are unsatisfiable (i.e., equivalent to `Nothing`, or $\bot$). Take the explanation for `CS_StudentTakingCourses`:

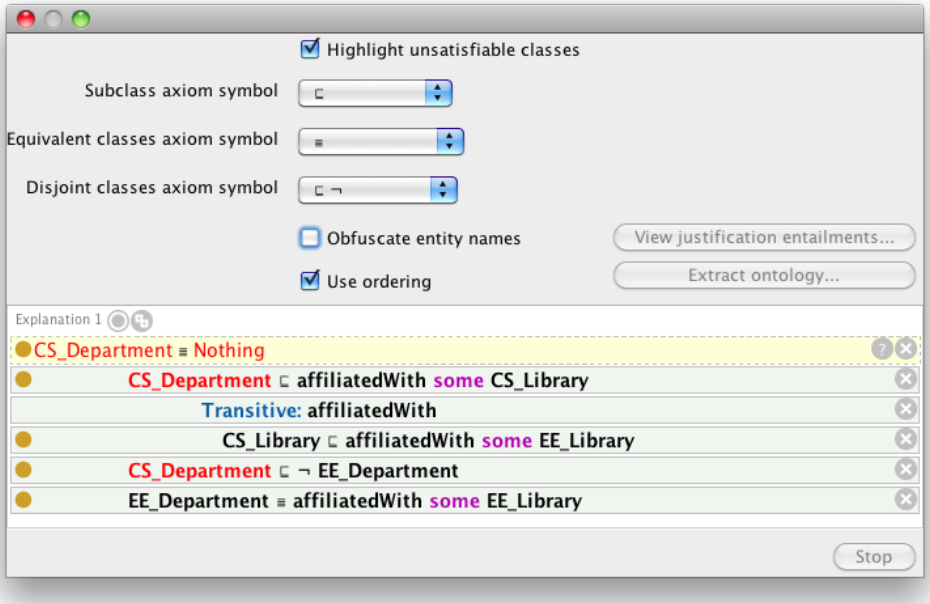


This `CS_StudentTakingCourses` has a long explanation of why it is unsatisfiable, and we see that some of the axioms that it uses to explain the unsatisfiability also have unsatisfiable classes. Hence, it is a good idea to set this aside for a while, as it is a knock-on effect of the others that are unsatisfiable.

Let us have a look at the unsatisfiability regarding departments.



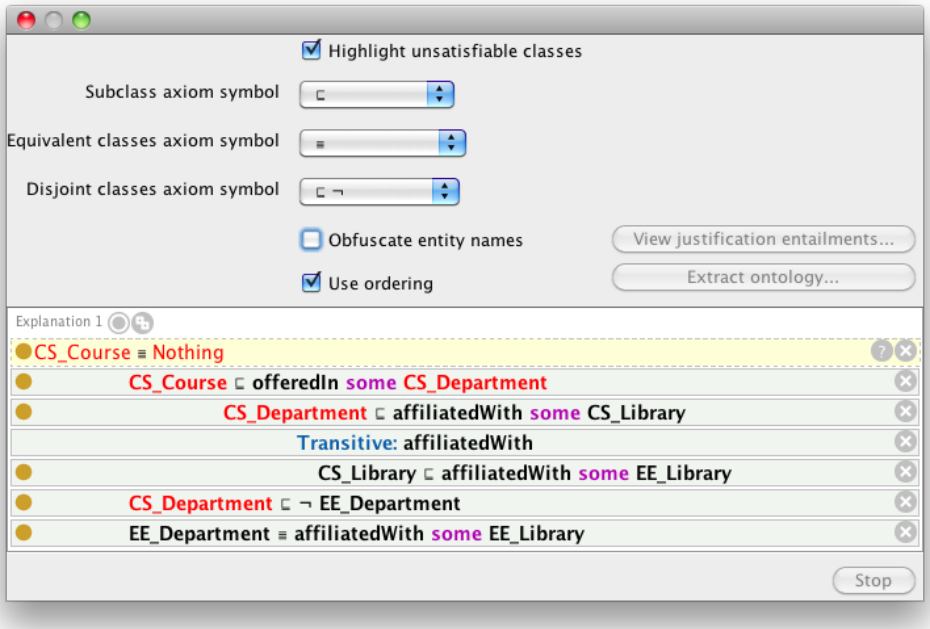
So, the `AI_Dept` is unsatisfiable because its superclass `CS_Department` is, i.e., it is a knock-on effect from `CS_Department`. Does this give sufficient information as to say why `CS_Department` is inconsistent? In fact, it does. See the next screenshot, which is the same as lines 3-7, above.



`CS_Department` is unsatisfiable, because it is `affiliatedWith` some `CS_Library` that, in turn (by transitivity), is `affiliatedWith some EE_Library` that belongs to the

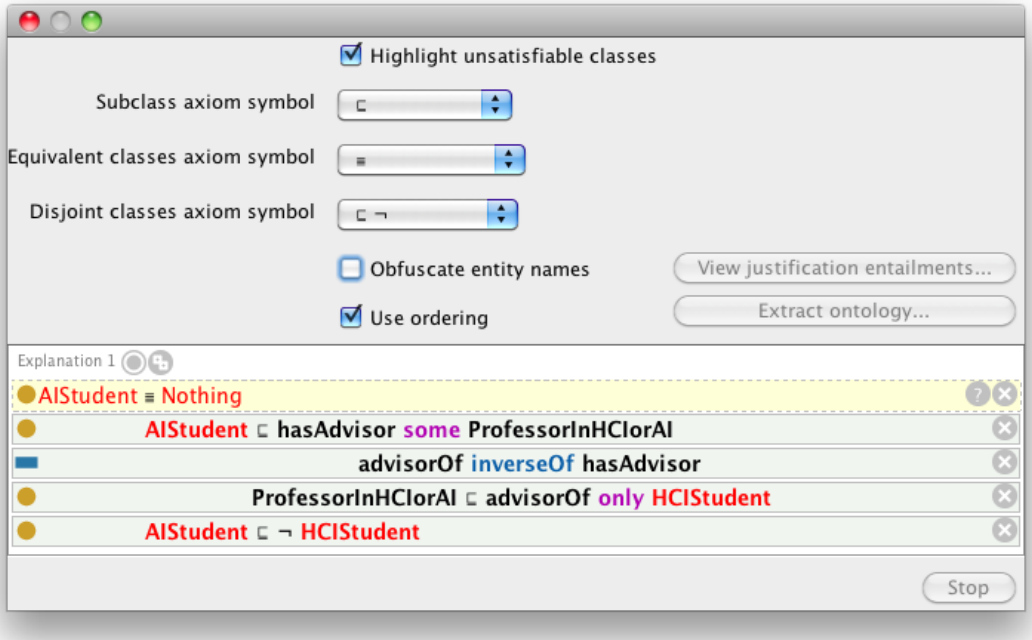
`EE_Department`, which is disjoint from `CS_Department`. Two 'easy' options to get rid of this problem are to remove the transitivity or to remove the disjointness. Alternatively, we could revisit the domain knowledge; e.g., CS library may not be `affiliatedWith` EE library, but is, `adjacentTo` or disjoint with the EE library.

Let us now consider why `CS_course` is unsatisfiable:

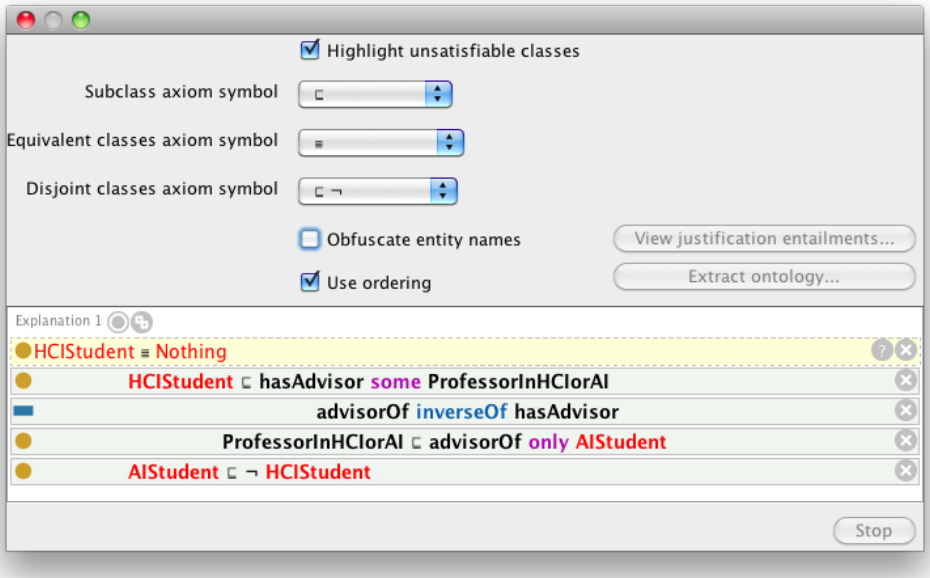


We have again that the real problem is `CS_Department`; fix that one, and `CS_course` is satisfiable, too.

There is a different issue with `AIStudent`. From the explanation in the next screenshot, we can see immediately it has something to do with the inconsistency of `HCIStudent`.

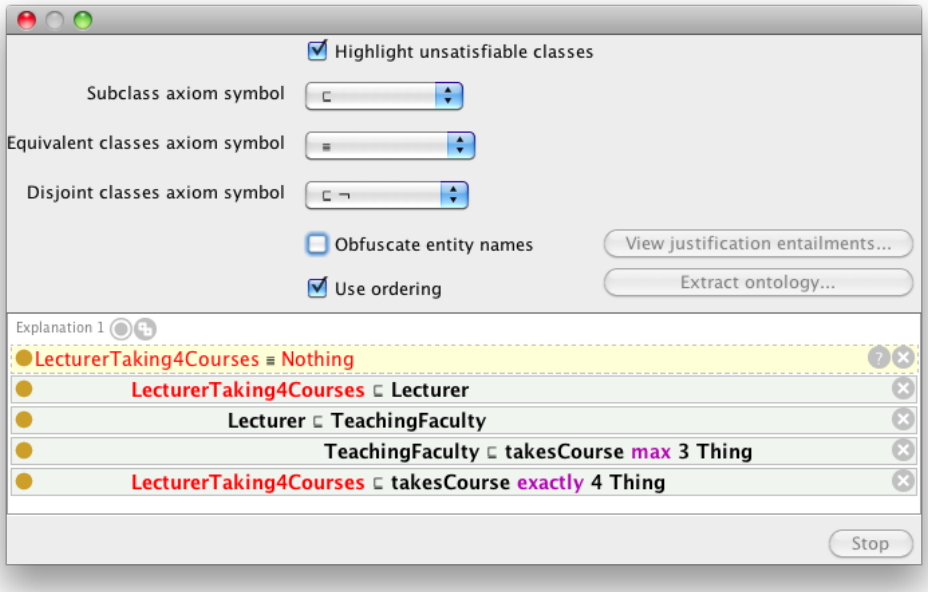


But looking at `HCIStudent` for a clue does not help us further in isolating the problem:



Considering the axioms in the explanation only, one can argue that the root of the problem is the disjointness between `AIStudent` and `HCIStudent`, and remove that axiom to fix it. However, does it really make sense to have the union `ProfessorInHCIorAI`? Not really, and therefore it would be a better fix to change that one into two separate classes, `ProfessorInHCI` and `ProfessorInAI` and have them participating in
$\texttt{ProfessorInHCI} \sqsubseteq \forall \texttt{advisorOf}.\texttt{HCIStudent}$ and
$\texttt{ProfessorInAI} \sqsubseteq \forall \texttt{advisorOf}.\texttt{AIStudent}$,
respectively.

Last, we have a problem of conflicting cardinalities with `LecturerTaking4Courses`: it is a subclass of `TeachingFaculty`, which is restricted to taking at most 3 courses, which is in conflict with the "exactly 4" of `LecturerTaking4Courses`. This can be fixed by changing the cardinality of either one, or perhaps a lecturer taking 4 courses is not a sub- but a sister-class of `TeachingFaculty`.



Answer Exercise 4.12. The general issue is that so-called 'ideal' solutions are not possible within OWL 2 DL.

(a) The crux is that Penguin is an exception to the general case (that birds fly). There are two principal options: either use some non-monotonic logic, which has an operator alike 'generally' and 'deviant case', or use some approximation with classes like a NonFlyingBird and a FlyingBird. An example of the latter approach is described in the GoodOD guide [36] (its section 6.2.1 in v1).

(b) The key issue here is how to deal with the phases and also here there are several options. One is the OntoClean way (which we will see in Chapter 5). Another option may be a temporal extension to be able to assert logically that there are successive stages in that way and no other (see Chapter 10), or a workaround approximating the knowledge in some way within OWL expressiveness, for which there are manifold options.

8.4 Answers Chapter 5

Answer Exercise 5.1. No, in that case, it is definitely not a good ontology. First, there are several CQs for which the ontology does not have an answer, such as not having any knowledge of monkeys represented. Second, it does contain knowledge that is not mentioned in any of the CQs, such as the carnivorous plant.

Answer Exercise 5.2. *Note:* Double-checking this answer, it turns out there are various versions of the pizza ontology. The answer given here is based on the output from the version I submitted in 2015; the output is available as "OOPS! - OntOlogy Pitfall Scanner! - Results pizza.pdf" in the PizzaOOPS.zip file from the book's website. The other two are: "OOPS! - OntOlogy Pitfall Scanner! - ResultsPizzaLocal.pdf" based on pizza.owl from Oct 2005 that was distributed with an old Protégé version and "OOPS! - OntOlogy Pitfall Scanner! - ResultsPizzaProtegeSite.pdf", who's output is based on having given OOPS! the URI `https://protege.stanford.edu/ontologies/pizza/pizza.owl` in July 2018.

There are 39 pitfalls detected and categorised as 'minor', and 4 as 'important'. (explore the other pitfalls to see which ones are minor, important, and critical).

The three "unconnected ontology elements" are used as a way to group things, so are not really unconnected, so that can stay.

ThinAndCripsyBase is detected as a "Merging different concepts in the same class" pitfall. Aside from the typo, one has to inspect the ontology to determine whether it can do with an improvement: what are its sibling, parent and child classes, what is its annotation? It is disjoint with DeepPanBase, but there is no other knowledge. It could just as well have been named ThinBase, but the original class was likely not intended as a real merging of classes, at least not like a class called, say, UndergradsAndPostgrads.

Then there are 31 missing annotations. Descriptions can be added to say what a DeepPanBase is, but for the toppings this seems less obvious to add.

The four object properties missing domain and range axioms was a choice by the modellers (see the tutorial) to not 'overcomplicate' the tutorial for novice modellers as they can have 'surprising' deductions, but it would be better to add them where possible.

Last, OOPS detected that the same four properties are missing inverses. This certainly can be added for isIngredientOf and hasIngredient. That said, in OWL 2, one also can use $\mathsf{hasIngredient}^-$ to stand in for the notion of "isIngredientOf", so missing inverses is not necessarily a problem. (Ontologically, one easily could argue for 'non-directionality', but that is a separate line of debate; see e.g., [9, 15]).

Answer Exercise 5.3. No, A is unsatisfiable. Reason: $A \sqsubseteq ED$ (EnDurant), it has a property R (to B), which has declared as domain PD (PerDurant), but $ED \sqsubseteq \neg PD$ (endurant and perdurant are disjoint), hence, A cannot have any instances.

Answer Exercise 5.4. There are several slides with the same 'cleaning procedure' and one of them is uploaded on the book's webpage, which was from the Doctorate course on Formal Ontology for Knowledge Representation and Natural Language Processing 2004-2005, slide deck "Lesson3-OntoClean". Meanwhile, there is also a related paper that describes the steps in more detail, which appeared in the Handbook on Ontologies [11].

8.5 Answers Chapter 6

Answer Review question 6.3. Some of the differences are: descriptive, possibilism, and multiplicative for DOLCE versus prescriptive and realist, actualism, and reductionist for BFO. You can find more differences in Table 1 of [24] and online in the "comparison tables" tab at `http://www.thezfiles.co.za/ROMULUS/`.

Answer Review question 6.4. The most often recurring relationships are parthood, participation, constitution, and inherence or dependence.

Answer Exercise 6.1. Informal alignments:

(a) dolce:Endurant maps roughly to bfo:Continuant (though actually, more precisely to bfo:IndependentContinuant), dolce:Process as a sub-class of bfo:Process, and dolce:quality to bfo:quality.

(b) Amount of Matter, Accomplishment, Agentive Physical Object, and Set do not have a mapping. An example of the possible reasons: Set is abstract, but not existing in nature (hence, by philosophical choice, not in BFO).

A more detailed comparison—or: the results of trying to align DOLCE, BFO, and GFO—is available at `http://www.thezfiles.co.za/ROMULUS/`.

Answer Exercise 6.2. Options may vary:

(a) DOLCE or GFO

(b) BFO or GFO

(c) Depends on you chosen topic

Answer Exercise 6.4. The main 'trick' with such questions is to be able to detect key words and phrases, such as the description stating that there will be "concrete entities ... and ... abstract entities": this data provide answers to one of the questions in ONSET, and will affect the choice of the foundational ontology (BFO does have abstract entities, but GFO and DOLCE do), and likewise the sentence on mereology and the text mentioning OWL 2 DL. Three use case with sample answers can be found at `http://www.meteck.org/files/onset/UseCasesExperiment.pdf`.

Answer Exercise 6.7. I use the version with DOLCE in the following answers

(a) To have RockDassie classified as a subclass of Herbivore (still both animals, and physical objects, and physical endurants, and endurants), it needs to have more, or more constrained properties than Herbivore. In Protégé notation, each `Herbivore` is equivalent to:
`(eats only plant) or (eats only (is-part-of some plant))`.
Rockdassies eat grasses and broad-leafed plants. The easiest way to modify the ontology is

to add that grasses are plants (already present), that broad-leafed plants are kinds of plants, and that rockdassies eat only grass or broad-leafed plant. This is not to say this is the best thing to do: there are probably also other animals that eat grasses and broad-leafed plants, which now unintentionally will be classified as rockdassies. This does not really need any of the foundational ontology content. One could align the parthood relations.

(b) The ontology does not contain any knowledge on 'residing in' and 'nature reserves', let alone sloppy word use of 'found on' (or, more precisely: in an area where a university campus is located). Nature reserves are administrative entities, but also can be considered only by their region-of-space aspect; for the sake of example, let's add NatureReserve $\sqsubseteq$ space-region. Trickier is the living, or living in: one could add it as an OWL object property livesIn or as a subclass of Process and add participation relations between that, the nature reserve, and the lions, impalas, and monkeys. The former is less cumbersome, the latter more precise and interoperable.

(c) Ranger is a role that a human plays for some time, with Human being a physical object, Ranger an agentive social object, and that the latter inheres in the former.

Answer Exercise 6.8. From the BFO viewpoint, the problem lies with the axioms that have eats existentially quantified: e.g., Warthog $\sqsubseteq \exists$eats.FruitingBody: some individual warthog `abc123` may never get around to *actually* eating any fruit, although the axiom asserts that each warthog will have at least one instance of eating a fruit. Thus, that axiom may not hold in all possible worlds after all and therefore ought not be in the ontology, for it would not represent reality fully correctly, according to the realist dogma. To resolve that, we first need to state that warthogs have a disposition to eat:

$$\text{Eating} \sqsubseteq \text{Process}$$
$$\text{EatingDisposition} \equiv \text{Disposition} \sqcap \forall \text{hasRealization.Eating}$$
$$\text{Warthog} \sqsubseteq \exists \text{bearerOf.EatingDisposition}$$

Then we need to find a way to relate it to FruitingBody, like berries and apples. The Eating process may have as participant the fruiting body, but one cannot simply state either of

$$\text{Eating} \sqsubseteq \exists \text{hasParticipant.FruitingBody}$$
$$\text{Eating} \sqsubseteq \forall \text{hasParticipant.FruitingBody}$$

because not all eating instances involve fruiting bodies and other food can be eaten as well. So, we need another approach. The first step is to constrain the participants in the eating process to only those eating processes where warthogs are involved and where fruiting bodies are involved:

$$\text{Warthog} \sqsubseteq \exists \text{bearerOf.}(\text{EatingDisposition} \sqcap \forall \text{hasRealization.}(\text{Eating} \sqcap \forall \text{hasParticipant.FruitingBody})$$

or, more precisely: in place of the FruitingBody, to put all the things they're eating according to AWO v1: fruiting body, grass, root, or animal. They may not be all part of the same eating event, but we will gloss over that detail for the moment. Obviously, we'll need to change the definitions for Omnivore and the other animals that were eating something as well, for else we'll lose the inference that Warthog $\sqsubseteq$ Omnivore. That covers it with respect to refining the semantics for eats. There is also an eaten-by object property that tasty-plant participates in:

$$\text{tasty-plant} \sqsubseteq \exists \text{eaten-by.carnivore} \sqcap \exists \text{eaten-by.herbivore}$$

It is left as an exercise to the reader to update this axiom[2].

Now, what to do if you want to assert that Warthog(j123) is eating Apple(apple1)? In AWO v1, it is simply added with eats(j123,apple1), but we can't do that anymore now. First, we need to add 'intermediate' individuals: EatingDisposition(ed1) and Eating(e1) and then add assertions bearerOf(j123,ed1), hasRealization(ed1,e1), and hasParticipant(e1,apple1). Yet, upon running the

[2] also: intuitively, this axiom ought to result in an inconsistent ontology when one adds tasty-plant(p1). Why does that not happen? Hint: examine the definition of carnivore.

reasoner, we still won't observe that `j123` is eating `apple1`. In order to get some of the 'cognitive simplicity'—our eats shortcut—back, one could add a role chain:

bearerOf ∘ hasRealization ∘ hasParticipant ⊑ eats

It will then derive eats(j123,apple1) and show that in the inferences.

This resultant ontology is available as `AfricanWildlifeOntology3b.owl`.

8.6 Answers Chapter 7

Answer Exercise 7.1. Bottom-up development and harmonisation considerations from conceptual data models:

(a) An OWL file as example may be made available online.

(b) Multiple answers may be 'not wrong' and acceptable due to various design decisions. You were expected to have considered, among others,
- EER's Class and UML's Course refer to the same concept, and the ontology should have been used to to harmonise those into one representation.
- EER's Role is the same in intent as UML's Position, and the ontology should have been used to to harmonise those into one representation.
- UML's Employed Dancer is a subclass of Employee, as the dance school may have employees who do not dance or teach, such as administrators, cleaners, marketers, event organisers and so on.
- Identifiers and names can safely be ignored in the bottom-up ontology development.
- A tricky difference is that in the EER diagram, members must pay per semester but they may not be taking a class then (they may be, e.g., competing in dance contest), whereas in the UML diagram, members must be attending at least one course. Without further context, your ontology likely will include only the weaker constraint.
- Technically, in the EER diagram, the relationships are named whereas in UML the association ends are, adhering more visible to the positionalism (see Sidebar 9). You likely will have 'bumped up' those association ends to object properties. In so doing, you probably have resolved the similarity of take and attends.

Answer Exercise 7.2. Thesauri:

(a) Language: SKOS or OWL 2 EL. Why:
- SKOS: was the purpose of it, to have a simple, but formal, language for 'smooth transition' and tagging along with the SW
- OWL 2 EL: intended for large, simple, type-level ontologies, and then still some reasoning possible

(b) Regarding mass media, films and news media: not necessarily, but to be certain, check yourself what the definition of `Mass Media` is, when something can be called `News Media`, and then assess the differences in their properties.
`Propaganda` has as broader term `Information Dissemination`, but a characteristic of propaganda is dissemination of *mis*information.

Answer Exercise 7.5. The description of the n-ary ODP can be found in the NeON deliverable D2.5.1 on pp67-68. Also, you may wish to inspect the draft ODPs that have been submitted to the ODP portal (at `http://www.ontologydesignpatterns.org`).

Answer Exercise 7.6. One could make a Content ODP out of it: for each AssistiveDevice that is added to the ontology, one also has to record the Disability it ameliorates, it requires some Ability to use/operate the device, and performs a certain Function. With that combination, one even can

create some sort of an 'input form' for domain experts and administrators, which can then hide all the logic entirely, yet as long as they follow the pattern, the information gets represented as intended.

Another one that may be useful is the Architectural OP: `adolena.owl` now contains some bits and pieces of both DOLCE (endurant, perdurant, and some of their subclasses) and some terms from BFO (realizable), neither of the two ontologies were imported. The architectural ODP can help cleaning this us and structuring it.

Answer Exercise 7.7. One could check the design against a foundational ontology and check whether the instantiations makes sense. There may be more options to evaluate it, as there has not been done much work on ODP quality.

Answer Exercise 7.9. Principally:

- expressive foundational ontology, such as DOLCE or GFO for improved ontology quality and interoperability
- bottom-up onto development from the thesaurus to OWL
- integration/import of existing bio-ontologies
- Domain ontology in OWL taking the classification of the chemicals, define domain & range and, ideally, defined concepts
- Add the instance data (the representation of the chemicals in stock) in the OWL ABox (there are only 100, so no real performance issues), and add a dummy class disjoint from DruTopiate destined for the 'wrong' chemicals
- Take some suitable reasoner for OWL 2 DL (either the 'standard' reasoners or names like Fact++)
- Then classify the instances availing of the available reasoning services (run Fact++ etc.): those chemical classified as instances of the 'ideal chemical' are the candidate for the lab experiments for the drug to treat blood infections.
- Alternatively: add DruTopiate as class, add the other chemicals as classes, and any classes subsumed by DruTopiate are the more likely chemicals, it's parents the less likely chemicals.
- Methods and methodologies that can be used: single, controlled ontology development, so something like METHONTOLOGY will do, and for the micro-level development something like OD101, ontospec, or DiDON.

8.7 Answers Chapter 8

Answer Exercise 8.1. Two of the reasons are:

- The ontology is specific to the application, rather than being application-independent. If it wouldn't be, then there would be mismatches, such as either too much/irrelevant content in the ontology, or data that should be queried but can't if there's no corresponding knowledge in the ontology.
- The ontology contains implementation decisions, such as data properties, which will hamper any reuse, be that for another application or as a module of another ontology.

8.8 Answers Chapter 9

Answer Exercise 9.1. There are no clear rules on IRI usage, but...

(a) The IRIs in `AfricanWildlifeOntologyAF.owl` are a mess. It was translated from an old AWO version that was once worked on in an old Protégé version (v4.1) that automatically put the path to where the local copy was stored (`file:/Applications/Protege_4.`

`1_beta/AfricanWildlifeOntology1.owl`), and this IRI was not updated in the file. The ontology IRI was updated, to `http://www.meteck.org/teaching/ontologies/AfricanWildlifeOntology1.owl`, but this was not propagated through to the IRIs of the classes and properties already declared in the ontology. In addition, that IRI is the location of the AWO v1 with element names in English, whereas this file has the element names in Afrikaans. New entities were added afterward, such as Witwortel (parsnip), which somehow got an `xml:base`, which is also incorrect.

(b) Recall that a full IRI is the official address of the element (class or property). While all 1:1 translated elements are semantically the same thing, the strings of the full IRIs are not, yet no equivalence is asserted either. There are several possible solutions to this, which relate to the architecture for maintenance (Exercise 9.4). Arguably, neither option is a good option.

(c) There is no explicit link to the original AWO other than same IRI for the Spanish one, and an annotation at best. What would be a way to make it clear(er) that they are all related to AWO v1?

Answer Exercise 9.3. Possible templates are as follows, noting that one can choose other words as well, and choose between being as close to the structure of the axiom, or decide on a more 'colloquial' rendering

(a) "$< C >$ and $< D >$ are disjoint"

(b) "If there exists an outgoing arc from $< R >$ to $< C >$, then it originates in $< D >$", or, easier to read: "$< D >$ is the domain of $< R >$ (when $< R >$ relates to $< C >$)"

(c) "Each $< C > < R >$ only $< D >$"

Answer Exercise 9.4. There are clearly many possibilities. Some essential ingredients, however, are: some place where language annotations can be stored, where some rules for the sentence can be stored and used, templates or patterns to generate a sentence for the axioms and/or parts thereof, and possibly algorithms to finalise the sentence, and the ontology.

8.9 Answers Chapter 10

Answer Exercise 10.1. They do not fare well in case 2 (*for* ISs), at least not in theory. First, because of the reliance on concrete domains. Second, the numbers you and your classmate had chosen for 'old' was likely not the same—it certainly wasn't for the students in one of my classes and the cut-off point I had in mind!—which then raises the general question as to what to do with something like such a number difference when faced with choosing to reuse an ontology and when aligning ontologies or integrating system. Conversely, it thus may work well for a particular application scenario (case 1, *in* ISs), assuming all its users agree on the fuzzy membership functions.

Answer Exercise 10.2. Chronos and PROTON are relatively easy to find. Chronos uses constraint satisfaction for the temporal component, PROTON is based on Prolog. Chronos uses the 4d-fluents approach (i.e.: perdudantist [recall Chapter 6]) and implements a reasoner for the Allen relations. PROTON uses intervals and extends the situation calculus.

Answer Exercise 10.3. No. (this is basically the same as the previous review question). Consider also the scope, as stated in the W3C standard, with emphasis added: "OWL-Time is an OWL-2 DL ontology of temporal concepts, for *describing* the temporal properties of resources in the world or described in Web pages. The ontology provides a *vocabulary for expressing facts* about topological

(ordering) relations among instants and intervals, together with information about durations, and about temporal position including date-time information. Time positions and durations may be expressed using either the conventional (Gregorian) *calendar and clock, or using another temporal reference system such as Unix-time, geologic time, or different calendars*.". That is: for annotations, but not for reasoning with it.

8.10 Answers Chapter 11

Answer Exercise 11.4. The use case for this request is likely to have been reuse. The type of module to be created is functional, because only the taxonomy should remain, and so it is an 'isolation branch' sort of module. Techniques are largely manual selection for deletion. at least in part. For instance, once could first extract a locality module with all herbivore knowledge using a locality-based algorithm, and then pass that module on to another algorithm to prune the object properties (or select all - delete in the ODE).

Answer Exercise 11.5. Assuming you used a downloaded SewerNet, click 'Load', select the file, select the radio button for 'Entity type abstraction', where 'Object Property ' is already selected by default, and then click 'modularise'.

Inspecting the ontology, you will notice that it only removed the object properties with the SewerNet IRI, not DOLCE's object properties. The tutorial on reusing ontologies, if not completed yet, makes clear why. In short: the DOLCE used in SewerNet is located at another location and therefore it is not modifiable. If you want DOLCE modularised as well, then do so separately, import that module, and align the SewerNet classes to those entities.

Appendix

A. A brief introduction to Protégé Desktop

Since you may be studying this from home without a teaching assistant or lecturer hovering across the lab, we have included a few annotated screenshots highlighting the rudimentary components of Protégé to get you started, especially in relation to the first tutorial. The screenshots and descriptions are for Protégé v5.6.5, and apply also to a number of older versions.

Let us begin by opening Protégé, loading an ontology and having a basic look-around. Explanation of the numbers in Figure A.1:

0: The Menu. Here you can open, save, select to run the reasoner, select or add additional plugins via 'Tool', and change the tabs and views you access directly, and more. In this case, I had clicked File - Open from URL... and loaded the African Wildlife Ontology v1 from the textbook's website.

1: The tool opens by default on the Active Ontology tab, which has a number of different areas. You may not see all the boxes (purple headings) that are shown here; if not, then add them via the Window menu option. When you look to the right in the screenshot, there are other tabs and we'll get to those next.

2: The ontology IRI: where the ontology is to be found on the web. You will have to update that at some point to something that is relevant to your ontology.

3: Ontology metadata. The rdfs:comment is an example annotation to provide some context. Typical other fields that are used, and added using the encircled "+" next to Annotations, are version information, contributors, a licence, contact information, and how to reference it.

4: Imported ontologies will show here, and you can add them here by clicking on the encircled "+".

5: Ontology metrics are computed over your ontology together with the imported ontologies. For instance, there are 31 classes in this ontology.

6: I then ran the reasoner by going to Reasoner - Start reasoner. If no reasoner was selected and it only shows ELK and HermiT in the list, then choose HermiT (at least

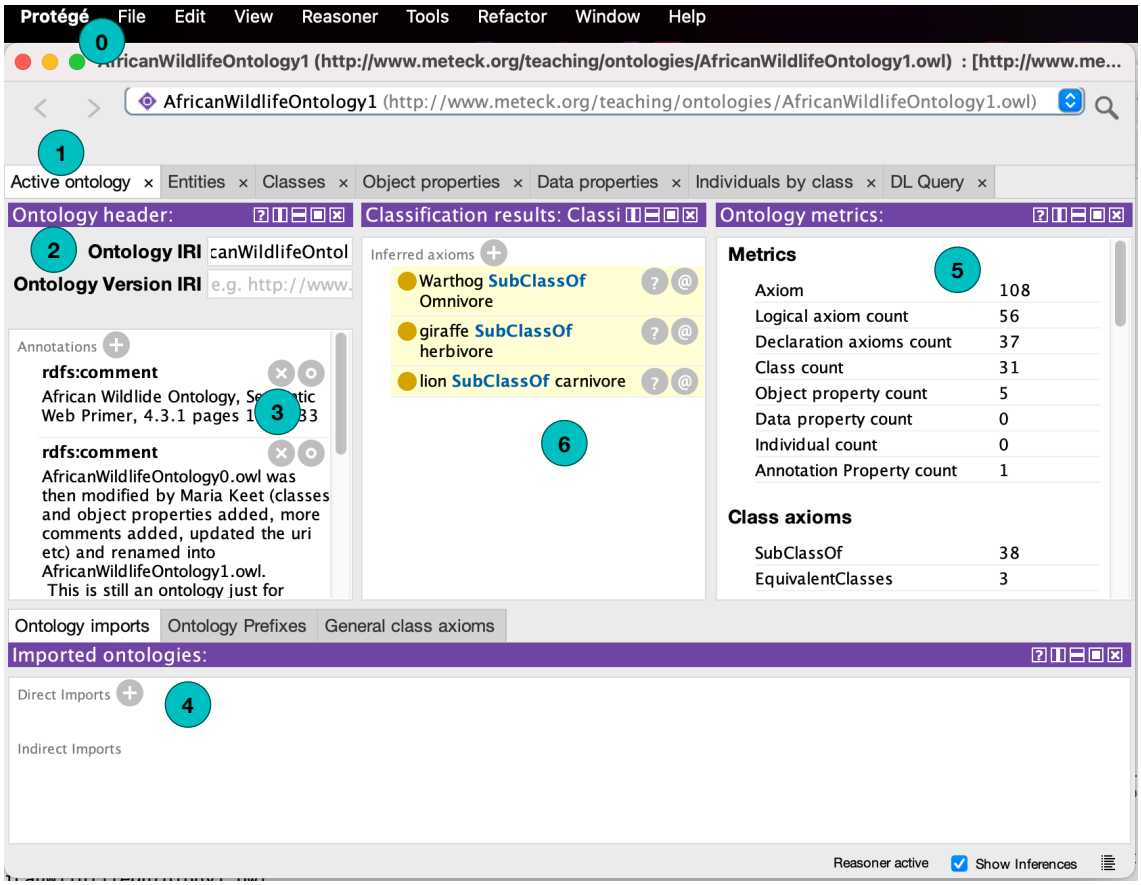


Figure A.1: Welcome screen after loading the AWO v1 ontology.

for Tutorial 1) by ensuring it has a check next to the name (click on the string). The results of the reasoner is shown in yellow highlight throughout the tool and they are collated in one list in the Classification Results view. It inferred that Warthog is an Omnivore, among others.

This completes a typical fist basic exploration of what we have in the ontology.

Now we move on to inspecting the knowledge represented in the ontology and adding new content. Explanation of the numbers in Figure A.2:

1: We navigated from the Active Ontology tab to the Classes tab.
2: The Class Hierarchy with the taxonomy of classes. Clicking the “>” opens the subclasses. Clicking the icon with the two blobs connected with a line allows you to add a class; the circle with the cross lets you delete the selected class. A selected class is highlighted in blue, like Warthog is.
3: Description view/box. It contains all that has been asserted about the selected class. The “(eats some FruitingBody)... ” under the SubClassOf are the properties of Warthog, i.e., what it is related to. Click on the encircled “+” next to SubClass Of to add more knowledge (we will do this further below). There are other things you can assert here, such as disjointness with other classes (scroll down in the Protégé view to check this).
4: All deductions are shown in yellow and also here we can see that it had deduced

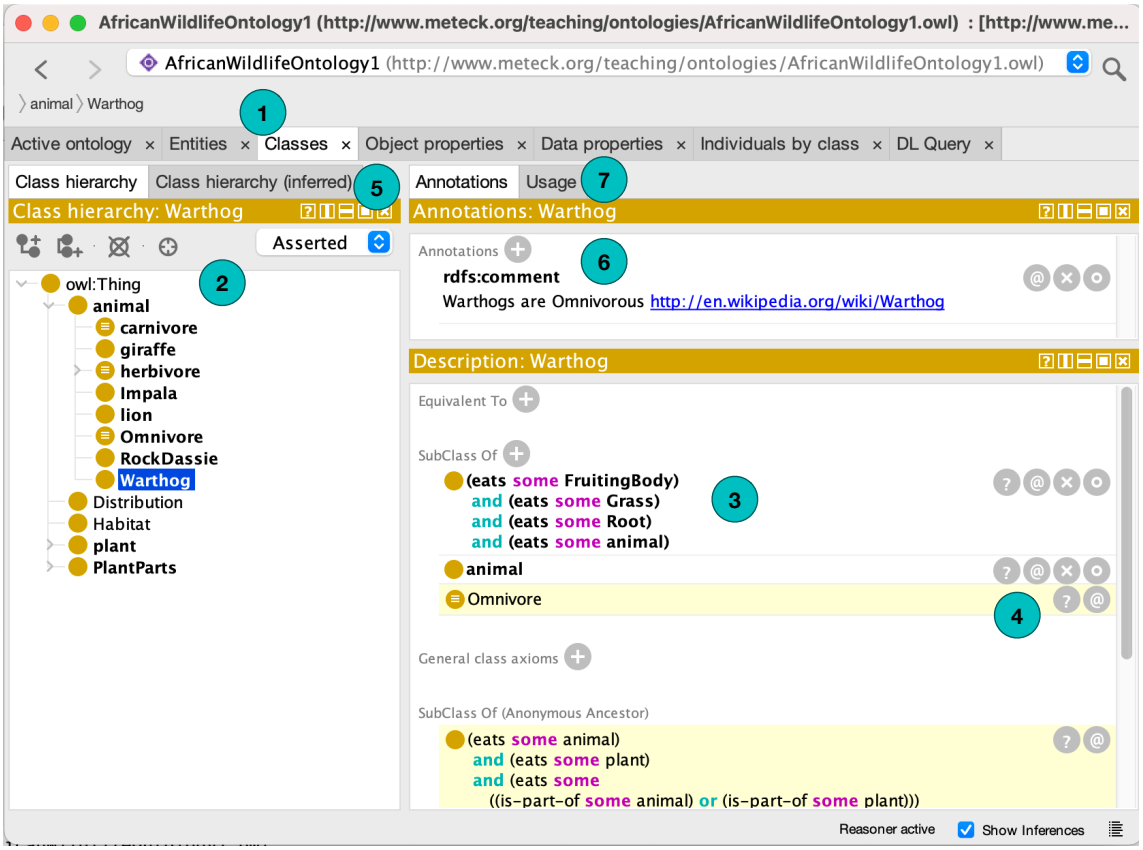


Figure A.2: Classes tab, with inferences shown.

that Warthog is an Omnivore. If you want to know why: click on the "?" in that yellow line. There is also a yellow highlight on Subclass Of (Anonymous Ancestor), which are the properties that Warthog inherited from Omnivore now that it is a subclass of Omnivore.

5: Another way of inspecting the deductions, is to open the Class hierarchy (inferred) view. There, you will find the rearranged class hierarchy, where Warthog is now positioned as a subclass of Omnivore, not as a sibling class as it is in the Class hierarchy view (second element above Warthog).

6: Annotations. Each element can be annotated with a variety of things, such as where the information was obtained from, a definition or description, examples, or notes about the modelling choices.

7: Usage. At times, notably when 'debugging' the ontology when there are errors, it can be useful to switch from Annotations to the Usage view, which collates all axioms that the entity participates in.

Before we start adding content, let us first look at the Object properties tab. Explanation of the numbers in Figure A.3:

1: Object property hierarchy. It works with the same approach as in the Classes tab.

2: Description. It works with the same approach as in the Classes tab, with the difference that different things can be said about object properties. Notably, inverses are declared here, as well as the property's domain and range. In this case, no domain is specified, which means it is owl:Thing, i.e., $\top$, i.e., any thing is permitted

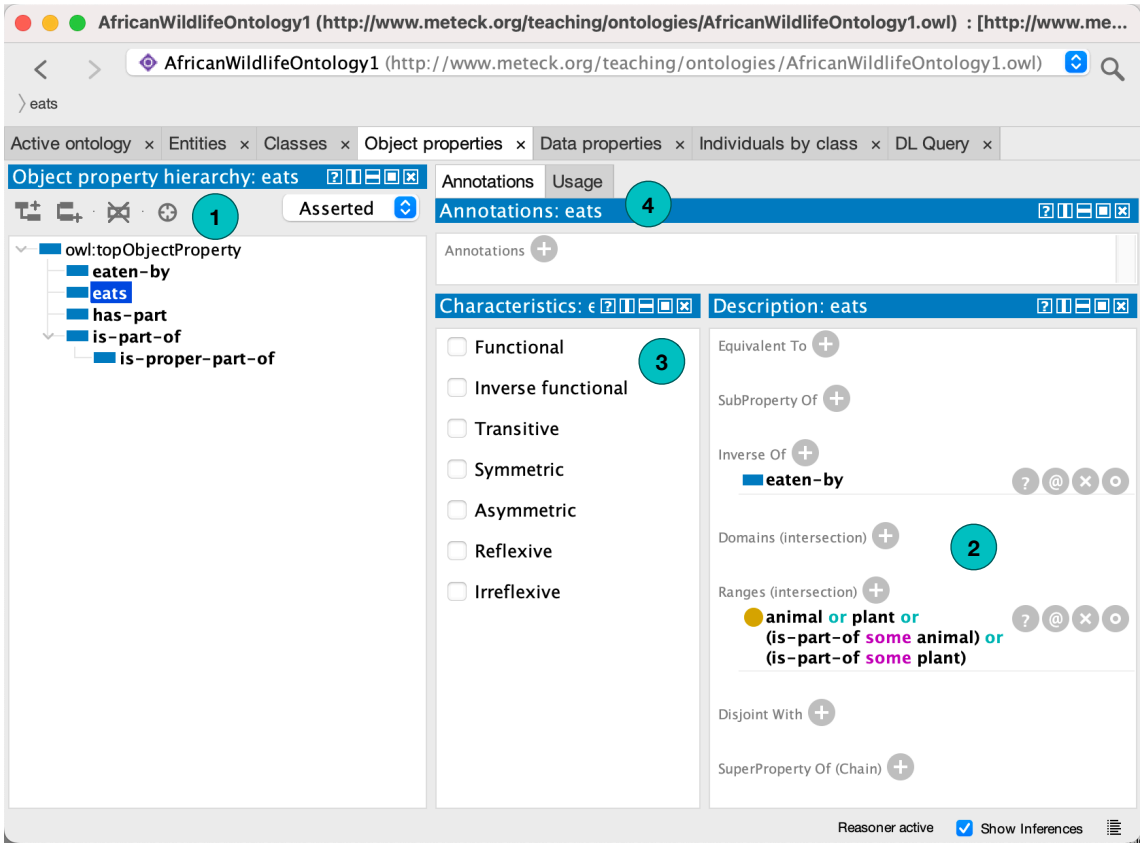


Figure A.3: Object properties tab, with the properties of the AWO v1.

to eat. Not everything can be eaten, however: animals, plans, or parts of animals and plants can be eaten. Note the or in the range declaration: here, and in most cases, that is probably what you want, not the intersection. If you add each of the four classes separately by clicking the encircled "+" four times, it will take the intersection of those classes rather than the union of that list.

3: Characteristics. The characteristics asserted there (by clicking the checkboxes) hold 'globally', i.e., for the whole ontology and thus wherever that property is used.

4: Annotations, Usage. See Classes tab, above.

Let's add content via the Classes tab, upon clicking the encircled "+" next to its SubClass Of. It generates a pop-up with four tabs. Explanation of the numbers in Figure A.4:

1: Object restriction creator. Here you can add basic relations to other classes for the class that was selected in the class hierarchy (branch in this case).

2: Restricted property. Select the applicable object property, which is is-proper-part-of in the screenshot.

3: Restriction type. Select the multiplicity/cardinality constraints by using the drop-down menu. For "at east one"/min 1 use Some (existential); for "zero or more", use Only (universal); and then there are Min (min cardinality), Exactly (exact cardinality), and Max (max cardinality) and only upon selecting either of those will the cardinality field become available.

4: Restriction filler. Select one or more classes that branch relate to; here, only tree is

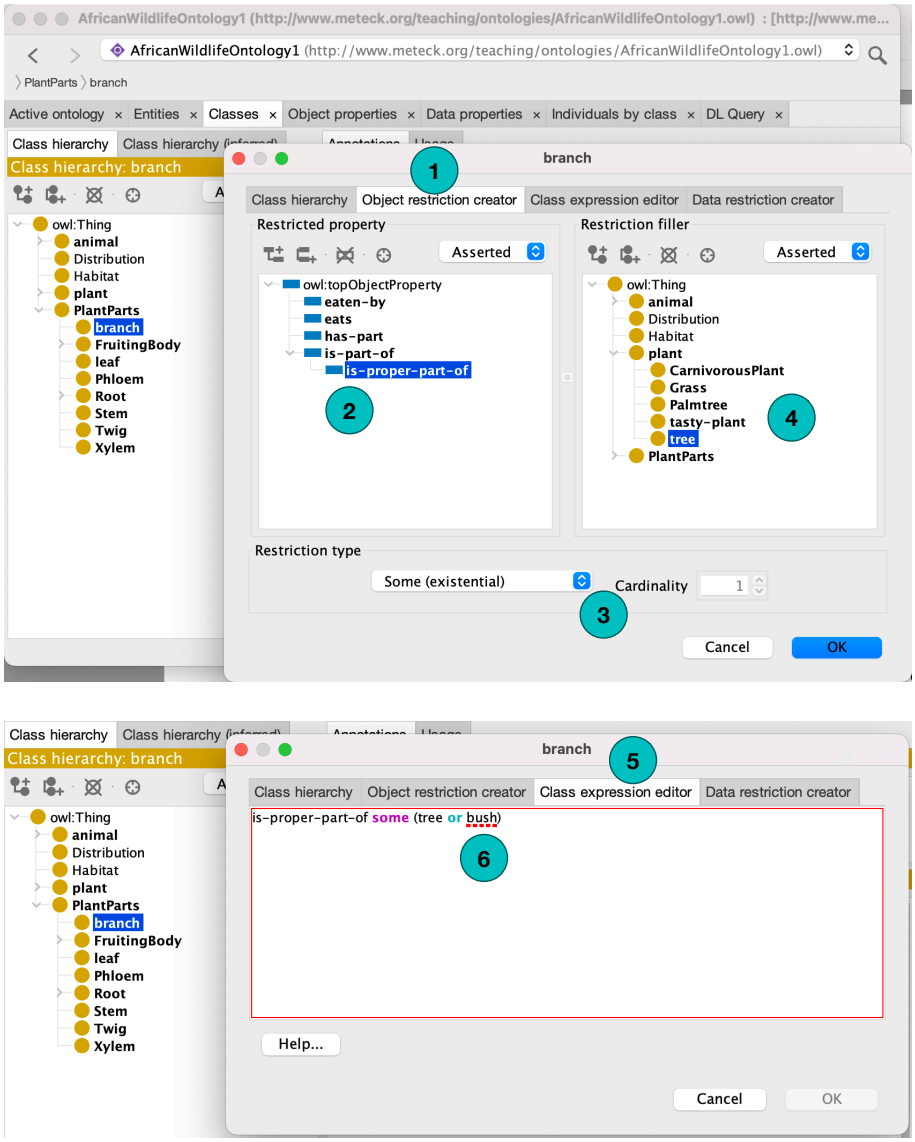


Figure A.4: Adding content via the Classes tab, two options.

selected. Then click OK to add it.

5: Class expression editor. Instead of clicking around in the Object restriction creator, it is also possible to add the relation by typing it, which becomes necessary especially if the restriction filler is not simply an atomic class.

6: The Class expression editor has an input checking functionality. The red-lined box means something is amiss and the tool rejects the class expression. Hints are indicated with the red dotted line. In this case, bush is underlined, because that element does not exist in the ontology. If we want to make the branch a proper part also of bushes, then we first need to go back to the Classes tab, add bush there, and then return here to re-type the expression and click OK.

Lastly, we take a quick look at the Individuals tab. Explanation of the numbers in Figure A.5:

1: Direct instances. I added Pumba as an individual warthog by clicking on the grey diamond. (Make use you have selected the right class in the Class hierarchy.)

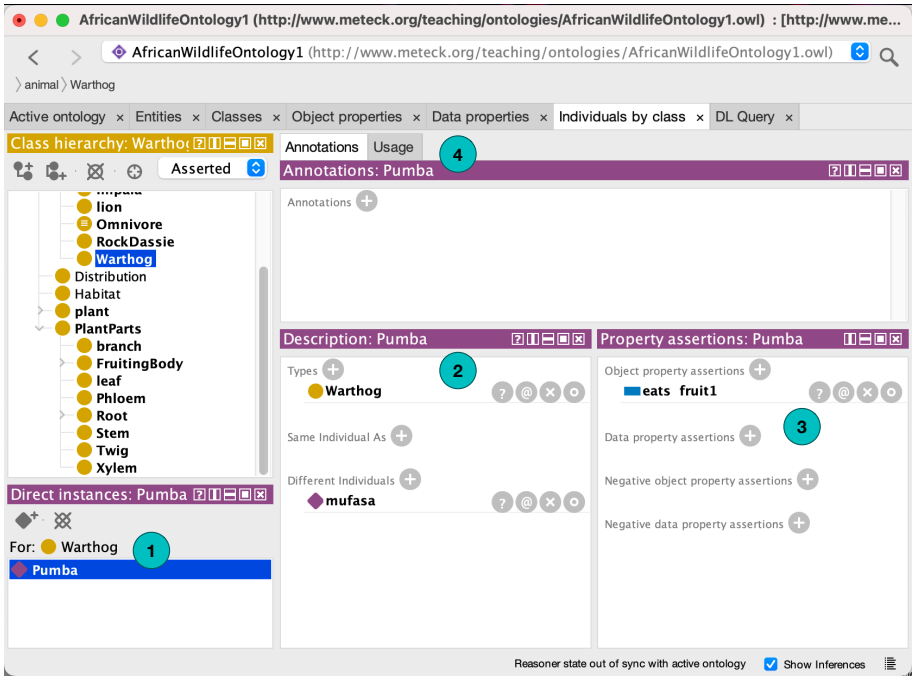


Figure A.5: The individuals tab, with three individuals added to AWO v1 for illustration.

2: Description. Basic information about the selected individual is displayed here. Pumba is an instance of Warthog and it is asserted to be different from the individual named mufasa.

3: Property assertions. Various things can be declared about Pumba. In this case, we have eats(Pumba, fruit1). More ABox assertions can be added by clicking the encircled "+" next to the respective headings.

4: Annotations, Usage. See Classes tab, above.

B. Dealing with References

Introduction

When working on a project or a thesis, one may, at times, feel as if one is working alone in isolation. This may be true for various reasons, but ought not to be the case, unless you are working on a PhD thesis when you have to demonstrate you have what it takes to go at least one whole turn through the knowledge creation spiral. What certainly holds, however, is that

1. the *topic* of your project or thesis does not exist in isolation, and
2. you are supposed to reinvent the wheel.

Put differently: *you are building upon other people's contributions.* That is how science and engineering move forward.

Building upon other people's results entails that you are aware what they did and understand the pros and cons of those earlier proposals. This *must* be reflected in the article or thesis you are or will be writing. In this document, we focus on how to go about giving credit where credit is due; not doing so amounts to **plagiarism**, which is the sin of sins in academia, on a par with inventing data and modifying them (that also constitutes scientific misconduct). If you plagiarise during your thesis writing, you will be reprimanded officially, if you have done so in your thesis and it is found out after graduation, your degree will be revoked, if you do it as a scientist, you will be fired. If you do so in a write up for an assignment of this module and I find out, you will receive a 0, a warning, and you are required to give a statement you wont do it again; if I catch you a second time, you will fail the module and the case will be recorded in the university system (note: given the weighting of the components that make up the grade, with one 0 you are very likely to fail the module already anyway).

Plagiarism and borderlines

Plagiarism is to copy whole sentence(s), figures, tables—any material—from another document without giving a reference to the source you took it from, be this someone else's or your own. 'Copy' here is taken liberally. So, if you were to take the original sentence from [16]

> On average, those who commenced with a foundational ontology added more new classes and class axioms, and significantly less object properties than those who started from scratch.

and copy it without putting it in indentation as quote or between quotation marks (“”), it is plagiarism. Or when you make only very minor changes, e.g.,

> On average, those who started with a foundational ontology added more new classes and class axioms, and significantly less object properties than those who started with a *tabula rasa*.

without adding a reference to [16], then it is still plagiarism, even though the two strings are not exactly the same. Note that when you find useful information online that you use in your writings, you have to reference that. It may not always be clear who provided that information and what the exact publication date was, or it may take some effort to find out, but it wasn't yours, so you are not allowed to claim as if it were.

This is difficult for mathematically-oriented papers when one has to introduce the syntax and semantics of a language that was already presented in another paper, so there are exceptions for those paragraphs (but try to fiddle a bit, where possible), provided the reference to the paper that introduced the language is given (e.g., [2] reuses the $\mathcal{DLR}_{\mathcal{US}}$ language introduced in [1], duly referenced).

This raises a new question: how to cite your source? This appendix is not a long list about the different referencing styles for the different type of documents—there is software who can handle that for you—but instead it contains a few guidelines on how you can refer to the references, the references section itself, managing references, and what is (not) referencable. (To reiterate: a section with references is *essential* to the report/article/thesis.)

Referencing in text

I will introduce first a difference between quoting and citing material, which also gives a first example of a reference in the text. There are two principle *ways of referencing* in the text: footnote/endnote or names/numbers, and, within each option, there are minor style variations; either way you choose: **be consistent** throughout the text.

Quoting versus citing

Longer quotes, like the one from [16] in Section B, should be *indented*, whereas shorter pieces can be put in-line in quotation marks, e.g.:

> We are interested only in the “significantly less object properties” of [16] instead of the whole gamut.

You also can emphasise something in a quotation, but this has to be indicated as such. Here is an example of a longer, indented quotation and emphasis:

> Against expectations, Keet [16] concluded the following from her experiments:
>
> > On average, those who commenced with a foundational ontology added more new classes and class axioms, and *significantly less object properties* than those who started from scratch. (emphasis added)
>
> Now, let us take a look at that peculiar aspect of it.

Where possible, add also the page number to the reference.

Quotations have to be truthful to the original. For instance, if the original sentence were:

> Ontologists tend to be outspoken about the usefulness of foundational (top-level) ontologies, such as BFO, DOLCE [1], GFO [2], and SUMO: either they are perceived to be essential or an impractical burden.

and you quote it as:

> Despite our own experiences in developing domain ontologies, Keet [16] dares to claim that "... foundational (top-level) ontologies ... are ... essential."

then this is clearly *not* truthful to the original sentence. It is permitted to skip *irrelevant* parts of a larger piece of text as long as they do not modify the overall message of the original.

Fiddling a bit with the layout to correct style (captialisation) or substituting a relative pronoun with the original noun is permitted, which is always put between square brackets; for instance:

> The dispute was succinctly summarised as "either [foundational ontologies] are perceived to be essential or an impractical burden" [16].

Instead of a quotation, which is fairly common in the humanities but not in computer science (because it takes up a lot of space), once can paraphrase it. For instance, you can put in the text

> Keet [16] demonstrated that the 52 test subjects added, on average, more new OWL classes and class axioms, and significantly less object properties when they started with the OWLized DOLCE.

or

> It has been shown that more new classes and class axioms were added when ontology developers started with a foundational ontology compared to those who did not [16].

In these two cases you *cite* material, but do not *quote* it. You put the same message in your own words with such citations: the fact or idea is not new, and hence the reference where it originates, but it is not taken verbatim from that source.

Referencing modes

The first way of referencing, footnote or endnote, is very common in the humanities when it comes to 'real' references. For computer science, they may be used for Web references *only*, and even then it depends on the outlet which one is preferred (and due to page limitations, one may take up less space than the other, hence make the difference). An example of footnote referencing is demonstrated the following text passage:

> What constitutes a nonviolent personality? Pontara describes 10 characteristics that a person with a nonviolent personality should posess to a great extent[1]. One of these is mildness, where he asserts in §3.2.7 that "[a] person equipped with a nonviolent personality is not outside the political struggle"[2], which I will discuss in detail in the remainder of this essay.

That is, the first reference to the document written by Pontara [34] is referenced in full, the subsequent one refers to the previous note and contains only the page number. The full citation is normally also included in a separate "References" section at the end of the text.

Referencing with footnotes occurs also in computer science literature, but this is limited to online sources, and is not nice behaviour toward the originators when a proper article of the online source exist (references count in citation indices—the more your paper is cited, the higher the impact of your contribution to the field—but footnotes/endnotes do not count). For instance, it is possible to do the following

> We developed our ontology in Protégé 4.1beta[3] by first importing the taxonomy of part-whole relations[4].

Very nice would be, instead:

> We developed our ontology in Protégé 4.1beta [10] by first importing the taxonomy of part-whole relations [18].

However, citing software is, to a large extent, a judgement call and depends on how widely known the software is. Protégé is so widely known within the ontology development community so that, if you write for *that* community, one would not cite it anymore unless there is something special about it. Surely one would not cite—article or URL—say, Linux OS in your materials & methods section (unless there's a very recent flashing new earth-shattering feature to be highlighted). New, or for the target audience unfamiliar, software applications should be cited so that a reader can follow up on what you have done.

You have already come across the other referencing mode, because it has been used throughout this file. More specifically, I used the numbering scheme where a number (that referred to a reference) was inserted in the main text, like the "... Protégé 4.1beta

[1] Pontara, G. (2011). The nonviolent personality. In: Keet, C.M. (Ed.), translated from Italian *La personalità nonviolenta*. 54p. Online: `http://www.meteck.org/files/NonviolentPersonality.pdf`. Last accessed: April 15, 2011.

[2] *op. cit.* p27.

[3] `http://stanford.protege.edu`

[4] `http://www.meteck.org/files/ontologies/pwrelations.owl`

[10] by..." in the previous example. There is just one section of references, which are numbered alphabetically in this case, and that's it. The previous example with a naming scheme instead of numbers could look like this:

> We developed our ontology in Protégé 4.1beta [GMF+03] by first importing the taxonomy of part-whole relations [KA08].

or one can use the full names instead of an abbreviation:

> We developed our ontology in Protégé 4.1beta (Gennari *et al.*, 2003) by first importing the taxonomy of part-whole relations (Keet & Artale, 2008).

The final display of the in-text references depends on which *referencing style* you use, which will receive attention in Section B.

What to put where

The examples in the previous section already introduced several options for how to include a reference in the text. Here I shall list that more systematically.

What has most effect on the formatting is if you choose a numbering or a naming scheme. For instance, this is good:

> Keet and Artale (2008) developed a taxonomy of part-whole relations.

but if the reference were a number, rendering the sentence like this:

> [18] developed a taxonomy of part-whole relations.

then that is not acceptable. In the second case, you would have to write

> Keet and Artale [18] developed a taxonomy of part-whole relations.

or, if you refer to more details, then the reference can be put at the end of the sentence:

> Keet and Artale developed a taxonomy of part-whole relations, which has as major division between 'real' parthood relations versus those relations that are motivated by linguistics and might seem a parthood relation, but are not [18].

or you can skip the author(s)' names, and write it in a more detached way:

> The most recently proposed taxonomy of part-whole relations has as major division between 'real' parthood relations versus those relations that are motivated by linguistics and might seem a parthood relation, but are not [18].

When using the naming scheme and there are more than three authors for a single reference, you do not include the whole list of authors in the text, but abbreviate it with *et al.*—in italics, and with the dot, as it abbreviates *et alia* 'and others'. We have seen this in an earlier example already, e.g., with the reference to Protégé:

> ... ontology in Protégé 4.1beta (Gennari *et al.*, 2003) by first...

Some software can do this automatically for you.

You also can combine references into one list, where appropriate. For instance,

> ... with some recent results on part-whole relations that also build upon temporal logics in general [1, 2, 18].

instead of the not incorrect but unpleasantly looking

> ... with some recent results on part-whole relations that also build upon temporal logics in general [1], [18], [2].

Observe two things in the example: the neater version has all the references within the same square brackets and in numerical order. The latter is not required, but neater.

The "References" section

The list of references goes at the end of the text. There is no consistency to do that always after the main text but before the appendix, or also after the appendix. Check other theses/papers for that, or, if you have an official template, adhere to the template. Regardless where it ends up after the text, the references in the "References" section have to be **consistent** and **complete**.

Whichever referencing style you use, *do the layout the same for the same type of entries*: if you do book titles in italics, then do so for all of them, if you include the total amount of pages of a book, do so for all of them, and so forth.

They also have to be complete to the extent that a reader should be able to find that paper. For instance, [10, 18] are complete entries. Conference and journal abbreviations and cutting down a long list of authors to one or three authors with an "*et al.*" may be acceptable, too (and in some outlets even required), e.g.:

> Gennari, J. H. *et al.*. The evolution of Protégé: an environment for knowledge-based systems development. *Int. J. of Hum.-Comp. Studies 58*, 1 (2003), 89-123.

may be fine if you run out of space, but *not* something like

> Gennari, J. H. *et al.*. The evolution of Protégé: an environment for knowledge-based systems development. 2003.

because it misses the publication venue details, nor

> Gennari, J. H., Musen, M. A., Fergerson, R. W., Grosso, W. E., Crubézy, M., Eriksson, H.. *Int. J. of Hum.-Comp. Studies*, 1 (2003).

because it misses the title of the paper, volume of the journal and page numbers, and has only several authors instead of all (or: misses the "*et al.*" after the last-mentioned author). A comprehensive version of this and other references used in this appendix can be found at the end of this appendix in the references section.

References to conference papers have their own acceptable or tolerable 'shorthand notations'. For instance, if the authors were really short on space, then a complete reference like [1] can be seen to be abbreviated as:

> Artale, A., Franconi, E., Wolter, F., and Zakharyaschev, M. A temporal description logic for reasoning about conceptual schemas and queries. *Proc. of JELIA'02*, Springer LNAI 2424, 98-110.

But again, this is only by exception and is done only when the readership is familiar with those venues. You, as a budding scientist reading those papers, may not be—and that is the only reason why the two examples of 'condensed' references are described here. You are not short on space, and have to give full details of the reference.

Finding the details of those 'random' online documents

There are many documents on the web and when you have search for information on, e.g. Google Scholar, clicked a pdf, then it might seem that is all there is. This is not the case, especially since computer scientists generally archive their papers on their home page—just that that document does not have all the citation data does not mean it has not been published. So for instance, the following practice is wrong:

> ... One of the semantic wikis is SweetWiki [1] that uses RDF [2]. ...
>
> *References*
> 1. http://smtp.websemanticsjournal.org/index.php/ps/article/viewFile/138/136
> 2. http://www.semantic-web-book.org/w/images/7/73/Informatik09-Semantic-Web-1-RDF.pdf

The *real* references for [1] and [2] that should appear in your references section are as follows:

> 1. Buffa, M., Gandonb, F., Ereteo, G. Sander, P., Faron. C. SweetWiki: A semantic wiki. *Journal of Web Semantics*, 2008, **6**(1): 84-97.
> 2. Hitzler, P., Kroetzsch, M., Rudolph, S. *Foundations of Semantic Web Technologies*. Chapman & Hall/CRC, 2009, 455p.

The trick is to find out that the first URL is actually pointing to a a preprint version of a published scientific paper and the second URL to tutorial slides of a textbook book (which can be referenced provided you did read the book).

The quick and dirty way to find out is to put the title of the document in Google Scholar. If it finds it, then look at the bottom row of the hit, which has hyperlinks like 'cited by', and also a 'cite'. Click on 'cite' and it gives you the reference in three different styles. Pick one and put it in the references; considering the gradations of (in)completeness, that is definitely better than just the URL. However, those automatically generated references are somewhat incomplete, especially when it comes to published conference papers. Another option is to look up the publications page of one of the paper's authors, and take the data from there. If you have figured out the publisher, then you also can go to the publisher's site to obtain the details. Finally, a lot of computer science papers have been indexed in the computer science bibliography server at `http://www.informatik.uni-trier.de/~ley/db/index.html`, which can generate the reference for you. If that does not work out well, like with the second URL: try to understand the URL. A "semantic-web-book" in it is a pretty obvious clue.

Managing your references

Nevertheless, this raises the question: What are the 'minimal' components of a reference? This depends on the reference *style* you use. For instance, there is a so-called "Harvard

referencing" and "Chicago style" that have elaborate guides with the dos and don'ts, there are common styles in computer science, such as of the IEEE, Springer LNCS, and ACM, and journals tend to have their own requirements how to reference each type of resource. You'd be crazy to read through all those guidelines and adapt your references each time you have to follow another style. Scientists and software developers got together, and developed reference management software to do this for you, so, unlike other resources on referencing (and there are a lot of them), I will not waste time re-writing a guide on that, because we can get that sorted out automatically.

When you conduct your literature research to read up on the topic you have chosen for your thesis, article, or technical report, you will come across many resources, some more useful than others. Of those deemed relevant, you should note the publication details immediately to save yourself from wasting time searching for that resource again in a few months' time. What information about the publication venue of that resource should you record? We have seen in the previous section that the notion of 'complete' information might, in fact, vary slightly. In addition, you may have noticed that the references you have come across have different layouts. Although you are going to focus on writing your honours project now, you may wish to continue with a masters and/or, if the results obtained are really good and novel, you may be involved in writing a scientific paper afterwards, which may have its own requirements for formatting references. Manually reformatting the references is a painstaking task and prone to introduce inconsistencies in the layouts. The latter is considered *sloppiness* and not appreciated: if you obviously do not care about your text, why should the reader spend precious time to read it?

Fortunately, many have gone before you who wanted to have a nice software-based reference management system that also does automatically the reformatting of the references adjusted to the venue and in the right order. There are now several such tools around that come in three different flavours. First, there are 'general science' reference management tools for people who write texts in OpenOffice or MS Word, such as Endnote[5], which, however, are stand-alone applications that are not for free. Second, there are free tools that take a social networking approach to reference management, such as Mendeley[6]. The third one is for LaTeX aficionados, such as Jabref[7], BibDesk[8], and BibTool[9] (more information at: `http://www.bibtex.org/`).

Basically, you store your references in a fancy database and each time you use a reference, you insert its key in the text. Once ready, the selected keys, your text editor, and your 'selected export format' get together and produce the right amount of references in the right format in the right order—automatically.

Let's have a quick look at an example for LaTeX, which is the preferred editing and typesetting program of computer scientists. Your text goes in a `.tex` file, your references go in a `.bib` file, and your choice for reference format is a selected `.bst` file. At the end of your main file, say, `myproject.tex` you add two lines, one to specify the reference *style*, say, `model3-num-names.bst`, and one to refer to your bibliography, say `mybib.bib`, which then looks like this:

[5] `http://www.endnote.com/`
[6] `http://www.mendeley.com/`
[7] `https://www.jabref.org/`
[8] `http://bibdesk.sourceforge.net/`
[9] `http://www.gerd-neugebauer.de/software/TeX/BibTool/index.en.html`

[Artale et al.(2002)Artale, Franconi, Wolter, and Zakharyaschev] A. Artale, E. Franconi, F. Wolter, and M. Zakharyaschev. A temporal description logic for reasoning about conceptual schemas and queries. In S. Flesca, S. Greco, N. Leone, and G. Ianni, editors, *Proceedings of the 8th Joint European Conference on Logics in Artificial Intelligence (JELIA-02)*, volume 2424 of *LNAI*, pages 98–110. Springer Verlag, 2002.

[Artale et al.(2008)Artale, Guarino, and Keet] A. Artale, N. Guarino, and C. M. Keet. Formalising temporal constraints on part-whole relations. In G. Brewka and J. Lang, editors, *11th International Conference on Principles of Knowledge Representation and Reasoning (KR'08)*, pages 673–683. AAAI Press, 2008. Sydney, Australia, September 16-19, 2008.

[Gennari et al.(2003)Gennari, Musen, Fergerson, Grosso, Crubézy, Eriksson, Noy, and Tu] J. H. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubézy, H. Eriksson, N. F. Noy, and S. W. Tu. The evolution of Protégé: an environment for knowledge-based systems development. *International Journal of Human-Computer Studies*, 58(1):89–123, 2003. ISSN 1071-5819.

[Keet(2011)] C. M. Keet. The use of foundational ontologies in ontology development: an empirical assessment. In G. Antoniou et al., editors, *8th Extended Semantic Web Conference (ESWC'11)*, volume 6643 of *LNCS*, pages 321–335. Springer, 2011. Heraklion, Crete, Greece, 29 May-2 June, 2011.

[Keet and Artale(2008)] C. M. Keet and A. Artale. Representing and reasoning over a taxonomy of part-whole relations. *Applied Ontology*, 3(1-2):91–110, 2008.

[Pontara(2011)] G. Pontara. *The nonviolent personality.* page 54. March 2011. URL `http://www.meteck.org/files/NonviolentPersonality.pdf`. Translated from the Italian original *La personalità nonviolenta.* Online: `http://www.meteck.org/files/NonviolentPersonality.pdf`. Last Accessed: April 15, 2011.

Figure B.1: The same references, but formatted with `abbrvnat`.

```
\bibliographystyle{model3-num-names}
\bibliography{mybib}
```

You then have to run latex-bibtex-latex-latex to get all references fully resolved. To produce the references for this document, I configured `biblatex` specifically, but there are very many more options available that come both with a LaTeX editor distribution and online. Have a look at, e.g., this resource[10]. For instance, when I use the very same bib file but with

```
\bibliographystyle{abbrvnat}
\bibliography{mybib}
```

the references look like depicted in Figure B.1. Not just that, but you would not have seen numbers in square brackets in this document, but the authors' names, like:

> ...importing the taxonomy of part-whole relations [Keet and Artale(2008)].

You may not like the square brackets, or want to have the in-text format alike "(Keet & Artale, 2008)" instead of "[Keet and Artale(2008)]". The `.bst` files are customizable, but before you have a go at that, you may want to check out the very powerful `natbib` package that already does most of this for you.

[10] `http://www.cs.stir.ac.uk/~kjt/software/latex/showbst.html`

What you can (not) cite

Your thesis/article/tech report is a serious academic document. This means that you use, and build upon—hence, also reference—proper academic and reliable, material of good quality. As a rule of thumb:

- What you can reference: journal articles, conference papers, books, chapters in books (including edited conference proceedings), technical reports, other theses, standards. In general: they are *primary sources*.
- What you should *not* reference (under normal circumstances, in CS): blogs and blog posts, posts on forums, Joe Soap's tips 'n tricks. These are, at best, *secondary sources*.

Web pages and company product manuals are trickier, but, in general, they should be avoided whenever possible. For instance, a 'white paper' by Company of Ruby And Perl on the Fabulously Implemented Algorithm to SCam Others may claim that their tool is the coolest around, but any company will say that about their applications. Let "[7]" be the reference to that online white paper, then you might be tempted to write a sentence in your thesis alike

> CRAP's FIASCO has the best performance [7].

A press release may say so, but your thesis is not a mouthpiece for the company. If, on the other hand, there is a paper *independently* from the organisation that compared it to similar tools and has shown it experimentally to hold, then that resource should be referenced.

Another tricky resource is Wikipedia, because it provides nice introductory overviews of many topics. However, Wikipedia is a secondary source and you should have a (reliable) textbook or handbook on those topics, in particular when it comes to your thesis. In this case, it is appropriate to reference the textbook.

Regarding blogs, while they should be avoided as reference, if you find, say, a piece of *good* code that someone put on their blog and it cannot be found elsewhere, it is better to reference that than claiming you invented it when you did not.

Bibliography

[1] Artale, A., Franconi, E., Wolter, F., and Zakharyaschev, M. “A temporal description logic for reasoning about conceptual schemas and queries”. In: *Proceedings of the 8th Joint European Conference on Logics in Artificial Intelligence (JELIA-02)*. Ed. by S. Flesca, S. Greco, N. Leone, and G. Ianni. Vol. 2424. LNAI. Springer Verlag, 2002, pp. 98–110.

[2] Artale, A., Guarino, N., and Keet, C. M. “Formalising temporal constraints on part-whole relations”. In: *11th International Conference on Principles of Knowledge Representation and Reasoning (KR’08)*. Ed. by G. Brewka and J. Lang. Sydney, Australia, September 16-19, 2008. AAAI Press, 2008, pp. 673–683.

[3] Bernabé, C. H., Keet, C. M., Khan, Z. C., and Mahlaza, Z. “A method to improve alignments between domain and foundational ontologies”. In: *14th International Conference on Formal Ontology in Information Systems 2023 (FOIS’23)*. Vol. 377. FAIA. IOS Press, 2023, pp. 125–139.

[4] Dawson, W. L. and Keet, C. M. “Ontology Pattern Substitution: Toward their use for domain ontologies”. In: *15th International Conference on Formal Ontology in Information Systems 2024 (FOIS’24)*. Vol. in print. 15-19 July, 2024, Enschede, The Netherlands. CEUR-WS, 2024, in print.

[5] Emeruem, C., Keet, C. M., Khan, Z. C., and Wang, S. “BFO Classifier: Aligning domain ontologies to BFO”. In: *FOUST-VI: 6th Workshop on Foundational Ontology, part of JOWO’22*. Ed. by T. Prince Sales, M. Hedblom, and H. Tan. Vol. 3249. CEUR-WS. 15-19 August 2022, Jönköping, Sweden. 2022, 13p.

[6] Ferré, S. “Semantic Authoring of Ontologies by Exploration and Elimination of Possible Worlds”. In: *Proceedings of the 20th International Conference on Knowledge Engineering and Knowledge Management (EKAW’16)*. Ed. by E. Blomqvist, P. Ciancarini, F. Poggi, and F. Vitali. Vol. 10024. LNAI. 19-23 November 2016, Bologna, Italy. Springer, 2016, pp. 180–195.

[7] Ferré, S. and Rudolph, S. “Advocatus Diaboli – Exploratory enrichment of ontologies with negative constraints”. In: *18th International Conference on Knowledge Engineering and Knowledge Management (EKAW’12)*. Ed. by A. ten Teije et al. Vol. 7603. LNAI. Oct 8-12, Galway, Ireland. Springer, 2012, pp. 42–56.

[8] Fillottrani, P. R. and Keet, C. M. “KnowID: An architecture for efficient Knowledge-driven Information and Data access”. In: *Data Intelligence* 2.4 (2020), pp. 487–512.

[9] Fine, K. “Neutral Relations”. In: *The Philosophical Review* 109.1 (2000), pp. 1–33.

[10] Gennari, J. H., Musen, M. A., Fergerson, R. W., Grosso, W. E., Crubézy, M., Eriksson, H., Noy, N. F., and Tu, S. W. "The evolution of Protégé: an environment for knowledge-based systems development". In: *International Journal of Human-Computer Studies* 58.1 (2003), pp. 89–123. ISSN: 1071-5819. DOI: 10.1016/S1071-5819(02)00127-1.

[11] Guarino, N. and Welty, C. "An Overview of OntoClean". In: *Handbook on Ontologies*. Ed. by S. Staab and R. Studer. Springer Verlag, 2009, pp. 201–220.

[12] Guarino, N. and Welty, C. "A Formal Ontology of Properties". In: *Proceedings of 12th International Conference on Knowledge Engineering and Knowledge Management (EKAW'00)*. Ed. by R. Dieng and O. Corby. Vol. 1937. LNCS. Springer Verlag, 2000, pp. 97–112.

[13] Guarino, N. and Welty, C. "Identity, Unity, and individuality: towards a formal toolkit for ontological analysis". In: *Proceedings of the European Conference on Artificial Intelligence (ECAI'00)*. Ed. by W. Horn. IOS Press, Amsterdam, 2000, pp. 219–223.

[14] Keet, C. M. "Relating some stuff to other stuff". In: *Proceedings of the 20th International Conference on Knowledge Engineering and Knowledge Management (EKAW'16)*. Ed. by E. Blomqvist, P. Ciancarini, F. Poggi, and F. Vitali. Vol. 10024. LNAI. 19-23 November 2016, Bologna, Italy. Springer, 2016, pp. 368–383.

[15] Keet, C. M. and Chirema, T. "A model for verbalising relations with roles in multiple languages". In: *Proceedings of the 20th International Conference on Knowledge Engineering and Knowledge Management (EKAW'16)*. Ed. by E. Blomqvist, P. Ciancarini, F. Poggi, and F. Vitali. Vol. 10024. LNAI. 19-23 November 2016, Bologna, Italy. Springer, 2016, pp. 384–399.

[16] Keet, C. M. "The use of foundational ontologies in ontology development: an empirical assessment". In: *8th Extended Semantic Web Conference (ESWC'11)*. Ed. by G. Antoniou et al. Vol. 6643. LNCS. Heraklion, Crete, Greece, 29 May-2 June, 2011. Springer, 2011, pp. 321–335.

[17] Keet, C. M. "A core ontology of macroscopic stuff". In: *19th International Conference on Knowledge Engineering and Knowledge Management (EKAW'14)*. Ed. by K. Janowicz and S. Schlobach. Vol. 8876. LNAI. 24-28 Nov, 2014, Linkoping, Sweden. Springer, 2014, pp. 209–224.

[18] Keet, C. M. and Artale, A. "Representing and Reasoning over a Taxonomy of Part-Whole Relations". In: *Applied Ontology* 3.1-2 (2008), pp. 91–110.

[19] Keet, C. M., Fernández-Reyes, F. C., and Morales-González, A. "Representing mereotopological relations in OWL ontologies with OntoPartS". In: *Proceedings of the 9th Extended Semantic Web Conference (ESWC'12)*. Ed. by E. Simperl et al. Vol. 7295. LNCS. 29-31 May 2012, Heraklion, Crete, Greece. Springer, 2012, pp. 240–254.

[20] Keet, C. M., Khan, M. T., and Ghidini, C. "Ontology Authoring with FORZA". In: *Proceedings of the 22nd ACM international conference on Conference on Information & Knowledge Management (CIKM'13)*. Oct. 27 - Nov. 1, 2013, San Francisco, USA. ACM proceedings, 2013, pp. 569–578.

[21] Keet, C. M. and Khumalo, L. "On the verbalization patterns of part-whole relations in isiZulu". In: *9th International Natural Language Generation conference (INLG'16)*. 5-8 September, 2016, Edinburgh, UK. ACL, 2016, pp. 174–183.

[22] Keet, C. M. and Kutz, O. "Orchestrating a Network of Mereo(Topo)Logical Theories". In: *Proceedings of the Knowledge Capture Conference (K-CAP'17)*. K-CAP 2017. Austin, TX, USA: ACM, 2017, 11:1–11:8. DOI: 10.1145/3148011.3148013.

[23] Keet, C. M., Xakaza, M., and Khumalo, L. "Verbalising OWL ontologies in isiZulu with Python". In: *The Semantic Web: ESWC 2017 Satellite Events*. Ed. by E. Blomqvist, K. Hose, H. Paulheim, A. Lawrynowicz, F. Ciravegna, and O. Hartig. Vol. 10577. LNCS. 30 May - 1 June 2017, Portoroz, Slovenia. Springer, 2017, pp. 59–64.

[24] Khan, Z. and Keet, C. M. "ONSET: Automated Foundational Ontology Selection and Explanation". In: *18th International Conference on Knowledge Engineering and Knowledge Management (EKAW'12)*. Ed. by A. ten Teije et al. Vol. 7603. LNAI. Oct 8-12, Galway, Ireland. Springer, 2012, pp. 237–251.

[25] Khan, Z. C. and Keet, C. M. "NOMSA: Automated modularisation for abstraction modules". In: *Proceedings of the EKAW 2018 Posters and Demonstrations Session (EKAW'18)*. Vol. 2262. CEUR-WS. 12-16 Nov. 2018, Nancy, France. 2018, pp. 13–16.

[26] Kless, D., Jansen, L., Lindenthal, J., and Wiebensohn, J. "A method for re-engineering a thesaurus into an ontology". In: *Proceedings of the Seventh International Conference on Formal Ontology in Information Systems (FOIS'12)*. Ed. by M. Donnelly and G. Guizzardi. Vol. 239. Frontiers in Artificial Intelligence and Applications. Graz, Austria, July 24-27, 2012. IOS Press, 2012, pp. 133–146. DOI: 10.3233/978-1-61499-084-0-133.

[27] Liang, S. F., Scott, D., Stevens, R., and Rector, A. L. "OntoVerbal: a Generic Tool and Practical Application to SNOMED CT". In: *CoRR* abs/1312.2798 (2013). arXiv: 1312.2798. URL: http://arxiv.org/abs/1312.2798.

[28] Liang, S. F., Stevens, R., Scott, D., and Rector, A. L. "Automatic Verbalisation of SNOMED Classes Using OntoVerbal". In: *Artificial Intelligence in Medicine - 13th Conference on Artificial Intelligence in Medicine, AIME 2011, Bled, Slovenia, July 2-6, 2011. Proceedings*. Ed. by M. Peleg, N. Lavrac, and C. Combi. Vol. 6747. Lecture Notes in Computer Science. Springer, 2011, pp. 338–342. DOI: 10.1007/978-3-642-22218-4_43.

[29] Liang, S. F., Stevens, R., Scott, D., and Rector, A. L. "OntoVerbal: a Protégé plugin for verbalising ontology classes". In: *Proceedings of the 3rd International Conference on Biomedical Ontology (ICBO 2012), KR-MED Series, Graz, Austria, July 21-25, 2012*. Ed. by R. Cornet and R. Stevens. Vol. 897. CEUR Workshop Proceedings. CEUR-WS, 2012.

[30] Mahlaza, Z. and Keet, C. "ToCT: A task ontology to manage complex templates". In: *FOIS 2021 Ontology Showcase, The Joint Ontology Workshops (JOWO'21)*. Ed. by E. M. Sanfilippo et al. Vol. 2969. CEUR-WS. 2021.

[31] Mahlaza, Z. and Keet, C. M. "OWLSIZ: An isiZulu CNL for structured knowledge validation". In: *Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+)*. Dublin, Ireland (Virtual): Association for Computational Linguistics, Dec. 2020, pp. 15–25. URL: https://aclanthology.org/2020.webnlg-1.2.

[32] McDaniel, M. H. "An Automated System for the Assessment and Ranking of Domain Ontologies". PhD thesis. Department of Computer Science, 2017.

[33] Peñaloza, R. and Turhan, A.-Y. "A Practical Approach for Computing Generalization Inferences in EL". In: *8th Extended Semantic Web Conference (ESWC'11)*. Ed. by G. Antoniou et al. Vol. 6643. LNCS. Heraklion, Crete, Greece, 29 May-2 June, 2011. Springer, 2011, pp. 410–423.

[34] Pontara, G. "*The nonviolent personality*". In: ed. by C. M. Keet. Translated from the Italian original *La personalità nonviolenta*. Last Accessed: April 15, 2011. Mar. 2011, p. 54. URL: http://www.meteck.org/files/NonviolentPersonality.pdf.

[35] Poveda-Villalón, M., Suárez-Figueroa, M. C., and Gómez-Pérez, A. "Validating ontologies with OOPS!" In: *18th International Conference on Knowledge Engineering and Knowledge Management (EKAW'12)*. Ed. by A. ten Teije et al. Vol. 7603. LNAI. Oct 8-12, Galway, Ireland. Germany: Springer, 2012, pp. 267–281.

[36] Schulz, S., Seddig-Raufie, D., Grewe, N., Röhl, J., Schober, D., Boeker, M., and Jansen, L. *Guideline on Developing Good Ontologies in the Biomedical Domain with Description Logics*. Technocal Report. v1.0. Dec. 2012. URL: http://www.purl.org/goodod/guideline.

[37] Soergel, D., Lauser, B., Liang, A., Fisseha, F., Keizer, J., and Katz, S. "Reengineering Thesauri for New Applications: the AGROVOC Example". In: *Journal of Digital Information* 4.4 (2004). URL: http://journals.tdl.org/jodi/article/view/jodi-126/111.

[38] Twala, E. K. "The noun class system of IsiZulu". MA thesis. University of Johannesburg, 1992.

[39] Vrandečić, D. "Ontology Evaluation". In: *Handbook on Ontologies*. Ed. by S. Staab and R. Studer. 2nd. Springer, 2009, pp. 293–313.

[40] Welty, C. "OntOWLClean: cleaning OWL ontologies with OWL". In: *Proceedings of Formal Ontologies in Information Systems (FOIS'06)*. Ed. by B. Bennet and C. Fellbaum. IOS Press, 2006, pp. 347–359.