Ontology Engineering with Diagrams: A Practical User Guide

V1.1

Gem Stapleton, John Howse and Aidan Delaney

with contributions from
Jim Burton and Peter Chapman

www.ontologyengineering.org

Twitter hash tag: #ontoed

Changelog

v1.1 Aidan Delaney, 6-Nov-2013 Added CC-BY licence.

v1.0 Authors, 11-Oct-2013 Completed initial version for ISWC 2013 tutorial.

Abstract

This book provides an overview to ontology engineering using concept diagrams. These diagrams have been especially designed for ontology engineering and are richly expressive. The book is aimed at beginners: no previous knowledge of either ontology engineering or concept diagrams is required. To begin, chapter 1 briefly describes the presentational conventions adopted in the book. Chapter 2 explains fundamental concepts required to formulate ontology models. In particular, the notions of *vocabulary* is introduced and *individuals*, *classes* and *properties* are explained. The reader is introduced to the concept of an axiom, which is a constraint that the ontology must satisfy. Axioms are formulated using the vocabulary. It is these constraints that can be expressed using concept diagrams. Chapter 3 presents a set of simple diagrammatic patterns that can be used to define commonly occurring axioms. For those readers who are familiar with description logic and OWL, the book draws comparison with these other notations. Often, diagrammatic patterns can be merged, so reducing the number of diagrams needed to specify the ontology but increasing the informational content of the individual diagrams. Techniques for merging patterns are covered in chapter 4.

Contents

Acknowledgements

The authors are grateful to a number of people for their contributions to the research which resulted in this book. Firstly, Dr Ian Oliver played an instrumental in the design of concept diagrams. It was during a research visit, by Stapleton and Howse, to Dr Oliver at Nokia Research in Helsinki that saw the core ideas for the concept diagram notation developed. Dr Oliver's continued input in to the evolution of the notation, since its birth, has been invaluable, especially as he has hands-on experience of modelling ontologies with various stakeholder groups. Second, the authors acknowledge Dr Kerry Taylor's role in suggesting the development of patterns using concept diagrams and for her contributions towards refining initial ideas, resulting in the patterns seen in this book today. Thanks are also due to Dr Taylor's colleagues for the many helpful discussions with Dr Gem Stapleton during her visit to CSIRO during 2013. Lastly, we are indebted to the many anonymous reviewers of the publications that the Ontology Engineering with Diagrams project team have produced on the concept diagrams. Through their valuable feedback, the diagrammatic approach to ontology engineering presented here has been improved.

Chapter 1

Introduction

This guide introduces concept diagrams for creating ontologies visually. No previous knowledge of concept diagrams is assumed or, for the most part, of modelling ontologies using alternative notations, such as OWL and description logics (DLs). The guide has been structured so that the reader can choose to skip content that makes reference to alternative notations.
This chapter outlines presentational conventions that are adopted throughout the guide. For convenience, the conventions used in this guide are summarized here. The terminology used below will be described later in the guide, at its next point of usage.

Named classes, properties, inverse properties, and individuals are written in san serif font, following conventions used elsewhere in the literature.

Unnamed classes, properties, inverse properties, and individuals are identified (where necessary) by text written in *italicized sans serif font*.

To highlight key components of the guide, boxes will be used where appropriate. Firstly, we present examples, exercises and tips like this:

Example: contextual information

Examples on how to use concept diagrams for building ontologies will be presented like this.

Exercise: contextual Information

Tutorial exercises will be presented like this.

Tip: contextual Information

Tips for building ontologies using concept diagrams will be presented like this.

This type of layout will also be adopted for other presented items, such as patterns.

Chapter 2

Basic Ontology Building

This chapter introduces the notions of individuals, classes, and properties, which form the vocabulary over which each ontology is built. It also introduces the notion of an axiom, which is a constraint that must hold given the domain being modelled or specified. At the end of this chapter, the reader should:

1. Understand what is meant by *vocabulary*.
2. Understand the difference between *individuals*, *classes*, and *properties*, and what they represent (their semantics).
3. Know what constitutes an *axiom*.
4. Understand how to represent individuals, classes, and properties *using concept diagrams*.
5. Have a basic understanding of *how to use concept diagrams to build ontologies*.

2.1 Informal Introduction

As with ontologies specified symbolically, ontologies specified using concept diagrams are used to capture, or *model*, domains. In order to build the model, one requires a *vocabulary* that corresponds to entities which are objects, sets of objects, or two-way (binary) relationships over the domain (the set of all objects). Given the vocabulary, concept diagrams (or any alternative ontology-building notation) are used to assert that relationships hold between those entities. Formally, each of these assertions is an *axiom*, the set of which are taken to be the built ontology. The notions of vocabulary and axioms are now illustrated with some simple examples, given the following scenario.

Scenario: Degree Courses

Suppose we wish to provide an ontological model of degree courses. These courses comprise

modules, which are components that *contribute to* a *degree*. Each module has a *credit value* which is a *positive integer*. *Students* are *enrolled on* a set of modules that contribute to their degree. Each module is *taught by* at least one *lecturer*. The individual *Peter* is a lecturer who *teaches* the module called *logic* (and possibly other modules, too). The italicized words identify entities in the domain. The information provided gives rise to relationships that must hold between the entities.

Using this scenario, we are able to demonstrate how to identify a vocabulary over which axioms are to be defined.

Example: Degree Courses Vocabulary

The Degree Courses scenario has (at least) the following terms in its vocabulary: Module, contributesTo, Degree, creditValue, PositiveInteger, Student, enrolledOn, taughtBy, Lecturer, Peter, teaches, and logic. These terms split in to three types as follows. Two of the terms identify particular *objects*, namely Peter and (the module called) logic. Terms that identify objects are called *individuals*. Some of the terms identify *sets* of objects: Module, PositiveInteger, Student, Degree, and Lecturer. Terms that identify sets of objects are called *classes*. The remaining terms identify *relationships* between objects: creditValue, enrolledOn, taughtBy, and teaches. Terms that identify relationships are called *properties*. Further, we may want to include Person in the vocabulary, as both students and lecturers are people.

The above example illustrates that the vocabulary is not always known upfront (Person was not part of the original description of the Degree Courses scenario) and, moreover, it need not be fixed at any point. It is common that the vocabulary evolves as one's understanding of the domain increases.

Tip: Identifying the Vocabulary

When building an ontology, it is normal for the vocabulary to enlarge over time. It is not necessary (or usual!) to identify all of the vocabulary up front.

Notice in the above example that there were two individuals, Peter and logic. The terminology *individuals* could potentially be confusing. In particular, it should not be taken to imply that individuals must be people.

Tip: Individuals

Individuals need not be specific people, as the name could be taken to imply. Individuals are just used to represent objects in the domain of interest.

Once a vocabulary (complete or otherwise) has been identified, the next stage is to identify constraints that are required to hold. These constraints are called *axioms*.

Example: Axioms Derivable from the Degree Courses Scenario

The Degree Courses domain needs at least the following axioms:

1. Each Module contributesTo at least one Degree.
2. Each Module has a creditValue which is a PositiveInteger.
3. Each Student is enrolledOn a set of Modules.
4. Each Student is registeredFor at least one Degree; notice here that the vocabulary has been enlarged, since the notion of a student being registered for a degree was implicit in the scenario information.
5. Each Module that a Student is enrolledOn must contributeTo the Student's Degree.
6. Each Module is taughtBy a Lecturer.
7. Each Student and each Lecturer is a Person.
8. Peter is a Lecturer.
9. logic is a Module.
10. Peter teaches logic.

As with the vocabulary over which the axioms are defined, it is common that more axioms are defined as one's understanding of the domain increases. For example, the Degree Courses scenario is unlikely to need just the axioms defined above. The following axioms may also be required.

Example: Further Axioms for the Degree Courses Scenario

The axioms should ensure, for instance, that people are not modules and so forth. It would not be sensible for people to also be modules! Thus, the following are also axioms:

11. Nothing is both a Module and a Degree.
12. Nothing is both a Module and a Person.
13. Nothing is both a Module and a PositiveInteger.
14. Nothing is both a Degree and a Person.
15. Nothing is both a Degree and a PositiveInteger.
16. Nothing is both a Person and a PositiveInteger.

Notice that there is no need to assert, for instance, that nothing is both a Module and a Student because this is implied by axioms 7 and 12. Lastly, it is probably intended that the properties taughtBy and teaches are interrelated. Put simply, the property teaches represents the same as the relationship taughtBy but in the opposite direction. Formally, teaches is the *inverse* of taughtBy.

Tip: Identifying the Axioms

When building an ontology, it is normal for the axioms to be written over time. It is not necessary (or usual!) to identify all of the axioms up front. Moreover, as one's understanding of the domain increases, it is often necessary to revisit and alter or fine-tune existing axioms.

Having now informally introduced the notions of *vocabulary* and *axioms*, the guide will now focus on how to represent individuals, classes and properties.

2.2 Vocabulary

A concept diagram ontology is built using a vocabulary that comprises individuals, classes, and properties. Each of these three types can be either *named* or *unnamed*. Those which are named have specific interpretations and the examples seen so far are all named. Those which are unnamed can be used to identify arbitrary collections of things, or collections of things that obey certain rules. Examples of unnamed individuals, classes and properties will be given later.

2.2.1 *Individuals*

Individuals represent objects, sometimes called elements, in the domain. Concept diagrams do not follow a unique name assumption: two individuals with the different names could actually refer to the same object. However, the default interpretation of individuals used in *a single diagram* is that the two represented objects are distinct; this feature of concept diagrams will be demonstrated later. The requirement is on the ontology builder to assert equality if this is what was intended. We return to this point in section 2.3 and demonstrate how such an assertion is made.

Concept diagrams represent individuals using dots, which may be labelled. The label is essential if the individual is named. If the individual is not named then the use of a label – which in this context just provides a means of reference – is optional. Unnamed individuals are useful when, for example, we know that there must be a least one object in a given class, but we do not know which one.

Syntax: Individuals (Representing Objects)

Concept diagrams represent objects using dots. Dots that represent specific objects are labelled with an (named) individual written in sans serif font, capitalization is optional:

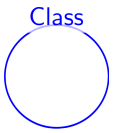
Dots that represent arbitrary objects are optionally labelled with an (unnamed) individual written in *italicized sans serif* font, capitalization is optional:

2.2.2 *Classes*

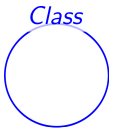
Classes (sometimes called concepts) represent sets of objects, or elements, in the domain. Concept diagrams represent classes using closed curves (e.g. circles), which may be labelled. The label is essential if the class is named. If the class is not named then the use of a label – which in this context just provides a means of reference – is optional. Unnamed classes are useful when, for example, an object is related to a set of things of a certain type, but not a named set. For example, there may be a constraint that asserts Gem teaches only modules. The set of modules on which Gem teaches has no name.

Syntax: Classes (Representing Sets of Objects)

Concept diagrams represent sets of objects using closed curves. Closed curves that represent specific sets of objects are labelled with a (named) class written in SansSerif font, in upper camel case:



Closed curves that represent arbitrary sets of objects are optionally labelled with a (unnamed) class written in *ItalicizedSansSerif* font, in upper camel case:



2.2.3 *Properties*

Properties (sometimes called roles) represent binary relationships between objects, or elements, in the domain. Concept diagrams represent properties using arrows, which may be labelled. The label is essential if the property is named. If the property is not named then the use of a label – which in this context just provides a means of reference – is optional. Unnamed properties are perhaps less useful than unnamed individuals and classes. The distracting detail of how unnamed properties can be used to specify constraints is omitted. However, for the interested reader, they are used when constraints of a more abstract nature arise, such as *these two classes represents sets that have the same number of elements* or *this class represents a set with an even number of elements*.

Syntax: Properties (Representing Binary Relationships Between Objects)

Relationships are represented using arrows. Arrows that represent specific relationships are labelled with a (named) property written in sansSerif font in lower camel case:

Arrows that represent arbitrary relationships are optionally labelled with a (unnamed) property written in *italicizedSansSerif* font in lower camel case:

2.2.4 *Summary*

The labels used in concept diagrams constitute the vocabulary over which the ontology is built.

Syntax: Vocabulary

The vocabulary comprises the labels that are used for individuals, classes and properties, represented by dots, closed curves and arrows respectively. By convention, the labels are written in sans serif font, which is *italicized* if the entity is unnamed. The labels used for classes are in upper camel case and are in lower camel case for properties.

Semantics: Vocabulary

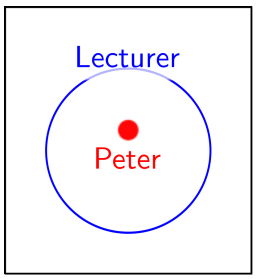
Labels that are used for individuals represent objects (sometimes called elements) of the domain. Labels that are used for classes represent sets of objects drawn from the domain. Labels that are used for properties represent two-way (directed) relationships (i.e. binary relations) between a pair of domain objects.

2.3 Axioms

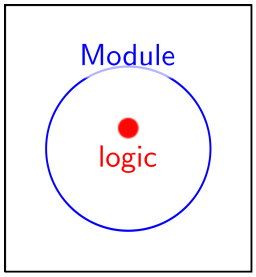
Axioms are conditions that must be satisfied by individuals, classes and properties. Their formulation is sometimes easy but can often be complex. This section gives the basic idea as to how axioms are formulated using concept diagrams. Of note is that this section introduces nearly all of the concept diagram syntax. The examples are kept relatively simple, increasing in difficulty as the section progresses. The tutorial material in the remainder of this guide will give more complex examples that illustrate various subtleties.
We begin by showing how to assert that individuals are members of particular classes.

Example: Individuals in Classes

The Degree Courses scenario asserted that Peter is a Lecturer; this is axiom 8 on page 5. Concept diagrams express this by placing a dot labelled Peter inside a circle labelled Lecturer:



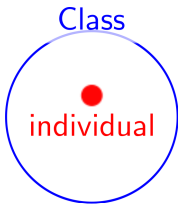
Similarly, axiom 9 asserts that logic is a Module:



These examples are sufficient to illustrate the general case: dots drawn inside closed curves assert that objects lie in particular sets.

Syntax: Individuals in Classes

Closed curves (typically circles or ellipses) labelled by classes represent sets. Dots are labelled by individuals, representing elements, and can be drawn inside those closed curves:



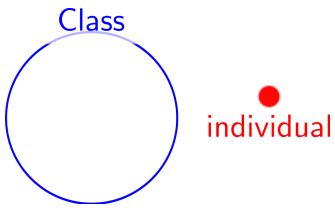
Semantics: Individuals in Classes

Placing a dot (labelled by an individual) inside a closed curve (labelled by a class) asserts that the represented object is a member of the represented set.

Whilst not necessary for the Degree Courses Scenario, sometimes ontologies need to assert that individuals are not members of sets. Similar to using containment to assert membership, exclusion is used to assert non-membership.

Syntax: Individuals not in Classes

Closed curves (typically circles or ellipses) labelled by classes represent sets. Dots are labelled by individuals, representing elements, and can be drawn outside those closed curves:



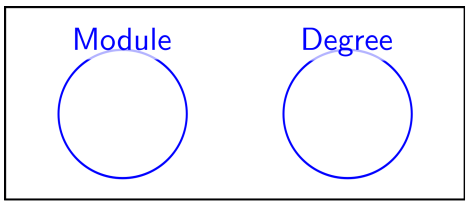
Semantics: Individuals not in Classes

Placing a dot (labelled by an individual) outside of a closed curve (labelled by a class) asserts that the represented object is *not* a member of the represented set.

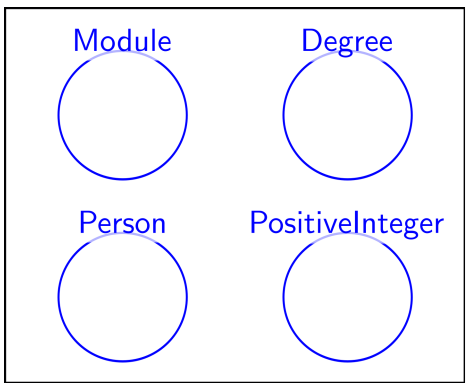
It is common to need to express that classes do not have individuals in common. Concept diagrams do so by making use of non-overlapping curves: the disjointness of the curve interiors corresponds to the disjointness of the represented sets of objects.

Example: Classes with No Individuals in Common

The Degree Courses scenario asserted that no individual could be both a Module and Degree; this is axiom 11 on page 6. A concept diagram expresses this by drawing two closed curves, one for Module and the other for Degree, which do not overlap:



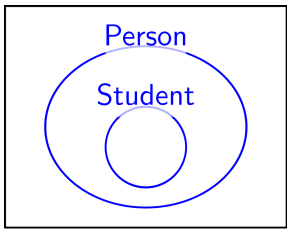
Similarly, axioms 12 to 16 assert that the respective classes have no individuals in common. All of these *disjointness* constraints can be expressed as a single concept diagram axiom:



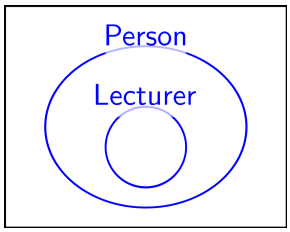
Another common need is to express that all individuals in one class are also in another class; this is known as a *subsumption* (subset) relationship. Concept diagrams express this by using curve containment: the containment of one curve by another curve corresponds to subsumption.

Example: Classes Contained by Other Classes

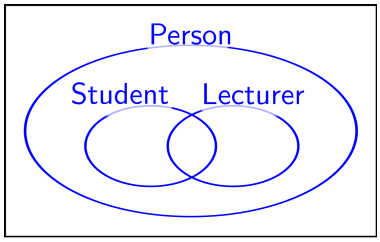
The Degree Courses scenario asserted that each Student is a Person; this is axiom 7 on page 5. Concept diagrams express this by drawing two circles, one for Student and the other for Person, where the latter contains the former:



Axiom 7 also states that each Lecturer is a Person. Similarly, this is expressed as follows:



The previous two diagrams can actually be joined into a single diagram:

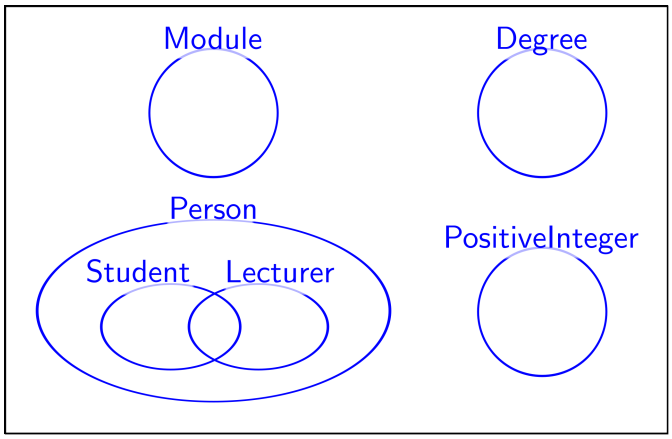


Notice that Student and Lecturer overlap in this diagram: it is unknown/unspecified as to whether they have individuals in common. It would be incorrect to draw these two circles either not overlapping or with one containing the other without having further information.

So far, the examples have shown that it is possible to express many disjointness relationships in a single concept diagram. Moreover, many subsumption relationships can be expressed in a single diagram. It is now demonstrated that these types of relationships need not be expressed in separate diagrams, but the information can be merged and presented in a single diagram. Ultimately, there is a choice about how many diagrams should be used to express disjointness and subsumption relationships which must be made by the person or team building the axioms that form the ontology.

Example: Class Disjointness and Subsumption Relationships

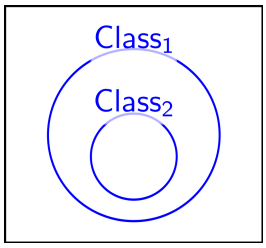
The final diagrams in the previous two examples can actually be merged into a single diagram expressing all of their combined information: only one concept diagram is needed to express axiom 7 and axioms 11-16:



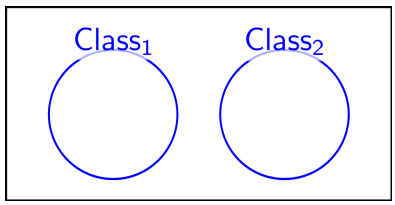
The examples presented demonstrate how concept diagrams use closed curves to express subsumption and disjointness relationships using closed curves.

Syntax: Class Disjointness and Subsumption Relationships

Concept diagrams use closed curves (typically circles or ellipses) labelled by classes to represent sets. A concept diagram representing a subsumption relationship may contain curves in this layout:



A concept diagram representing a disjointness relationship may contain curves in this layout:



In general, concept diagrams can contain many curves.

Semantics: Class Disjointness and Subsumption Relationships

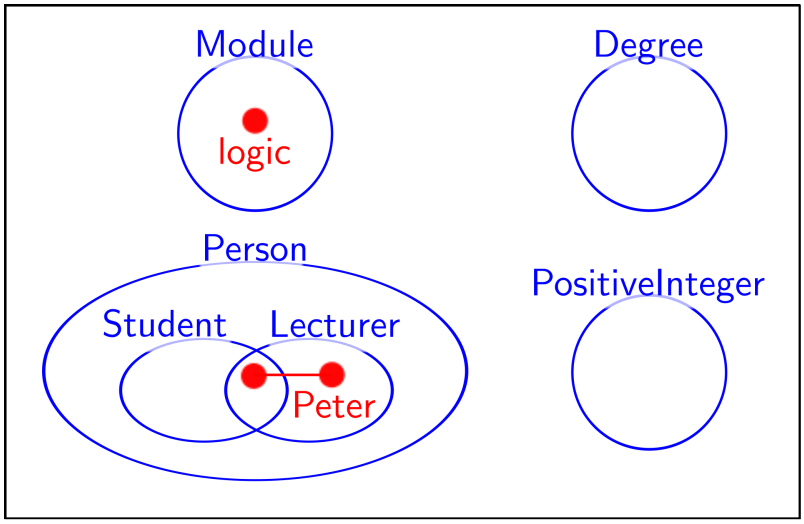
Any concept diagram that contains many curves asserts properties about the relationships between the associated classes. If one curve contains another then this asserts that all of the objects in the set represented by the contained curve are also in the set represented by the containing curve; this situation is a *subsumption* (*subset/superset*) relationship between the sets. Similarly, if two curves do not overlap then this asserts that the two represented sets have no elements in common; this situation is a *disjointness* relationship between the sets. Curves that partially overlap are used when neither a subsumption nor a disjointness relationship hold.

A single concept diagram can express a lot of information, as already seen. So far, the examples have shown how to represent whether individuals are in classes and whether classes are in a subsumption relationship or are disjoint. In addition, these two information types can also be expressed together, in a single diagram.

Example: Putting Individuals and Classes Together

It was already demonstrated that a single diagram could be used (in the Degree Courses scenario) to represent all of the subset/superset and disjointness relationships. The information about the classes in which individuals lie can be added to this diagram. The individual logic can simply

be drawn inside the curve Module. The case with Peter is not quite so straightforward. We know that Peter is a Lecturer but whether he is a Student is unknown. Concept diagrams represent this kind of uncertainty by placing Peter in multiple regions. A single diagram representing all of the information about the classes and individuals is:



Here, the individual Peter is represented by two dots joined by a line. There are two dots because there are two regions inside the curve labelled Lecturer. Informally, these two regions, between them, represent the set of lecturers in which we know Peter lies. The joining line thus represents a *disjunction*: either Peter is both a Lecturer and a Student (the left most dot) or Peter is a Lecturer but not a Student (the right most dot). Lastly, this diagram *explicitly* asserts that Peter and logic represent distinct objects. This distinctness information was implicit in the previous diagrams; it could be *inferred* from them.

The above example demonstrated that concept diagrams can be used to explicitly assert that individuals represent distinct objects. Whilst not seen in the running example, concept diagrams join dots together with $=$ to assert the individuals are the same. For example, if Peter was also known as PiratePete, then the dots labelled Peter and PiratePete would be joined by $=$, as shown here:

Syntax: Distinct Individuals

Concept diagrams use multiple dots labelled by individuals. This is illustrated here with two individuals:

To assert that individuals are distinct, concept diagrams use = between dots:

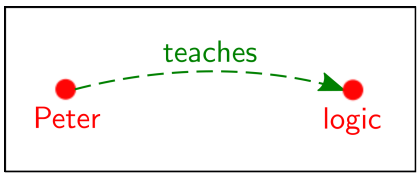
Semantics: Distinct Individuals

A concept diagram containing two dots, labelled by individuals, that are not joined by = asserts that the represented objects are different from each other. A concept diagram containing two dots, labelled by individuals, that are joined by = asserts that the represented objects are the same as each other.

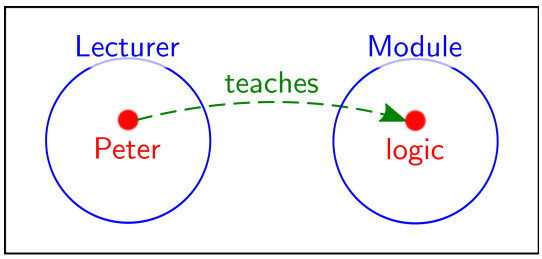
The remaining axioms given on page 5 all involve properties, which are represented by arrows. Sometimes only *partial* (incomplete) information is given about the relations represented by properties. For instance, axiom 10 asserts that Peter teaches logic. It is not stated whether there are other modules on which Peter teaches. To indicate only partial information is given, concept diagrams use *dashed arrows*. Six different ways of asserting that Peter teaches logic will be presented, two immediately.

Example: Using Dashed Arrows

A concept diagram asserting that Peter teaches logic is given here:



This diagram uses the information that Peter and logic are distinct individuals. The arrow from Peter to logic, labelled by the property teaches, asserts that Peter teaches logic. The arrow does not preclude Peter teaching other modules[1]. The diagram above can be enhanced by including curves labelled Lecturer and Module:



The enhancement is not necessary and does not convey any more information than the previous diagrams convey collectively. However, the (optional) curves serve as a visual reminder to the reader that Peter is a Lecturer and that logic is a Module.

The previous example illustrated that diagrams can be enhanced by including extra syntax to express redundant information. In this case, the redundant information appeared in another concept diagram that forms part of the Degree Courses ontology being built. Care must be taken, however, to ensure that the additional syntax – drawn with the intention of only aiding the reader – does not introduce new, unwanted information to the ontology.

Tip: Spatial Relationships Between Curves

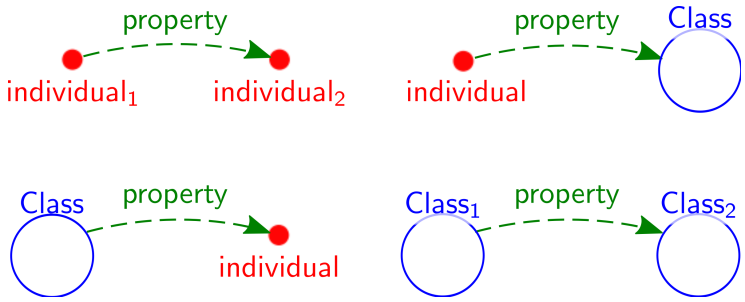
Care must be taken to ensure that the correct spatial relationships are exhibited between the curves. The second diagram in the previous example used the already given information that Lecturer and Module are disjoint.

[1] By contrast, the use of a solid arrow would assert that Peter teaches logic and only logic. Examples with solid arrows will be provided later.

The two examples given that used dashed arrows only have dots, labelled by individuals, as their sources and targets. However, in general, sources and targets can also be classes. The syntax and semantics associated with dashed arrows are now summarized.

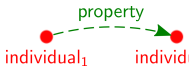
Syntax: Dashed Arrows

Concept diagrams use dashed arrows labelled by properties. Each dashed arrow is sourced on a dot or a closed curve. Each dashed arrow can be targeted on a dot or a closed curve. The four cases are exhibited here:

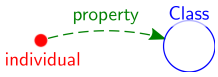


Semantics: Dashed Arrows

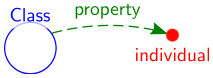
Dashed arrows are used to assert that binary relationships exists between objects as follows:

1. 

expresses that the object represented by individual$_1$ is related to (at least) the object represented by individual$_2$ under the binary relationship represented by property.

2. 

expresses that the object represented by individual is related to (at least) all of the objects in the set represented by Class under the binary relationship represented by property.

3. 

expresses that, between them, the objects in the set represented by Class are related to (at least) the object represented by individual under the binary relationship represented by property.

4. 

expresses that, between them, the objects in the set represented by Class$_1$ are related to (at least) all of the objects in the set represented by Class$_2$ under the binary

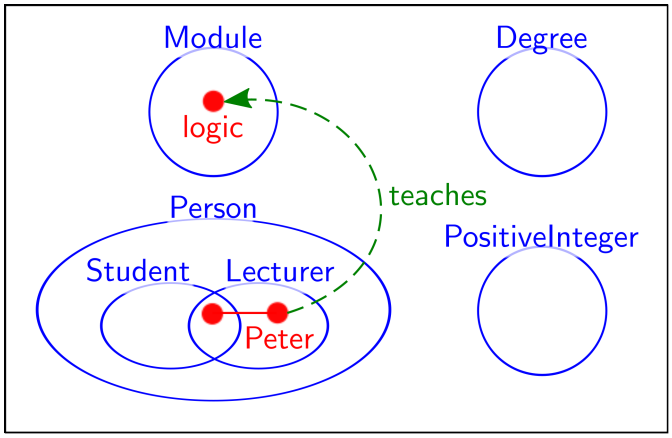
relationship represented by property.

Each of the above diagrams also expresses information about the source and target of the arrow, by using spatial relationships.

Before proceeding to give other ways of asserting Peter teaches logic, notice that the diagram on page 16 can be extended by the addition of an arrow between Peter and logic.

Example: Putting Individuals, Classes and Arrows Together

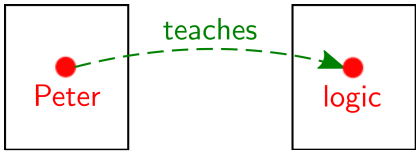
In the Degree Courses scenario, the single diagram that represents all of the subset/superset and disjointness relationships, along with the information on individuals can be extended to express that Peter teaches logic:



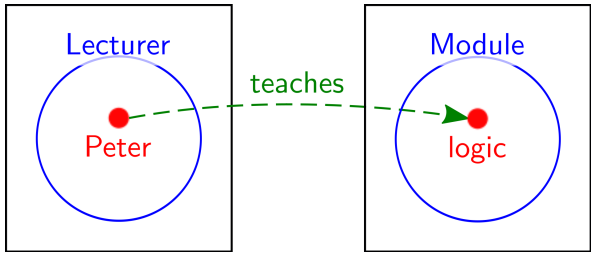
The diagrams used to describe the semantics of dashed arrows also assert other information. For example, the diagram in case 1 on page 20, given to illustrate the semantics of dashed arrows, also asserts that the two individuals are distinct. There may be situations where this is unintended, or unknown. Concept diagrams have a built in feature that avoids *overspecificity* problems: the use of *multiple boundaries*.

Example: Using Dashed Arrows and Multiple Boundaries

A third concept diagram asserting that Peter teaches logic is given here:



This diagram does not use the derivable information that Peter and logic are distinct individuals. To avoid asserting distinctness, concept diagrams make use of multiple boxes, each acting as a *boundary*. This boundary box informs the reader as to which spatial relationships have meaning: spatial relationships only convey meaning if they occur within the same boundary box[2]. This diagram can also be enhanced by adding curves for Lecturer and Module:

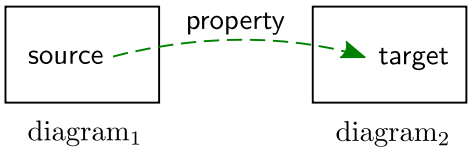


This enhanced diagram *does not* assert that Lecturer and Module are disjoint because the boundary boxes prevent non-overlapping spatial relationship between the curves from having any meaning.

[2] In general, the spatial relationships only convey meaning if they occur within the same set of boundary boxes; this will become clearer in later chapters

Syntax: Multiple Boundary Boxes

Concept diagrams use boundary boxes to contain diagrammatic elements. Diagrams contained by different boundary boxes can have arrows drawn between their syntactic components (i.e. their dots and their closed curves). As previously, these arrows are labelled by properties.



The above diagram illustrates the case when there are two boundary boxes and a single joining arrow. In general, there can be any finite number of boundary boxes and arrows[3].

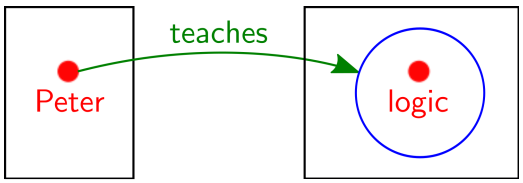
Semantics: Multiple Boundary Boxes

Boundary boxes limit the scope of the semantics conveyed by spatial relationships. The spatial relationship between items in different boundary boxes conveys no semantics.

As well as using dashed arrows to assert information about binary relationships, solid arrows are also part of the concept diagram syntax. They can assert stronger information than dashed arrows.

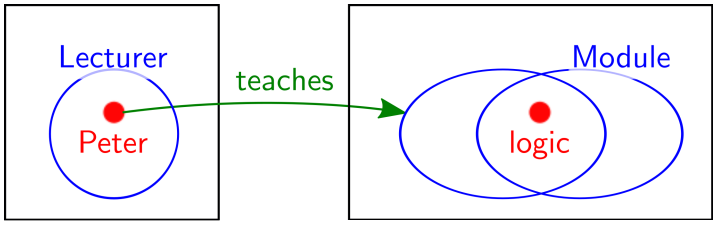
Example: Using Solid Arrows

A fifth concept diagram asserting that Peter teaches logic is given here:



[3]Essentially, boundary boxes act as brackets would in symbolic notations.

This diagram uses a solid arrow, unlike the previous examples. The target of the solid arrow is an unnamed class. This class is the set of objects that Peter teaches and it includes logic. We can also choose to enhance this diagram with the classes Lecturer and Module:

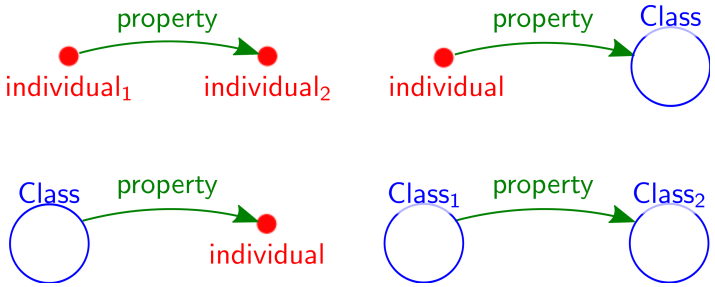


Notice that this last diagram does not assume that the property teaches always 'returns' modules: the set of things taught by Peter may not all be modules; for example, Peter also teaches students. This is visually demonstrated by the spatial relationships between the Module and the unnamed class that is the target of the arrow: they are neither disjoint nor in a subsumption relationship.

The use of solid arrows is the same as dashed arrows: they can be sourced and targeted on dots or closed curves and they are labelled by a property. However, as illustrated, their semantics is stronger. By contrast to dashed arrows, which give partial information, solid arrows give complete information about the relationship between the source and target.

Syntax: Solid Arrows

Concept diagrams use solid arrows labelled by properties. The possible sources and targets are the same as for dashed arrows: each solid arrow is sourced on a dot or a closed curve; each solid arrow is targeted on a dot or a closed curve. The four cases are exhibited here:



Semantics: Solid Arrows

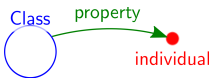
Solid arrows are used to assert that binary relationships exists between objects as follows:

1. 

expresses that the object represented by $\mathsf{individual}_1$ is related to *only* the object represented by $\mathsf{individual}_2$ under the binary relationship represented by property.

2. 

expresses that the object represented by individual is related to all of, *and only*, the objects in the set represented by Class under the binary relationship represented by property.

3. 

expresses that, between them, the objects in the set represented by Class are related to *only* the object represented by Individual under the binary relationship represented by property.

4. 

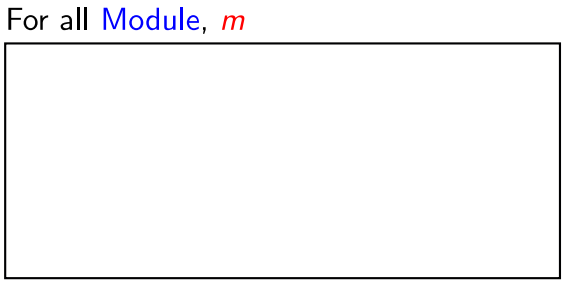
expresses that, between them, the objects in the set represented by Class_1 are related to all of, *and only*, the objects in the set represented by Class_2 under the relationship binary represented by property.

Each of the above diagrams also expresses information about the source and target of the arrow, by using spatial relationships.

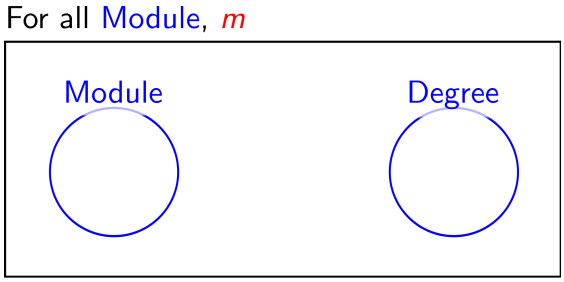
The guide now returns to the use of dashed arrows in order to express the remaining axioms. First, axioms 1, 2, and 6 on page 5, express information about modules.

Example: Relationships Between Modules and Degrees

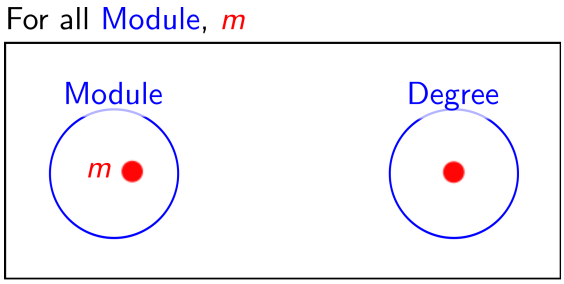
Axiom 1 expresses that each Module contributesTo at least one Degree. This diagram needs to make an assertion about *all* modules. This is done by a textual annotation outside of a boundary box:



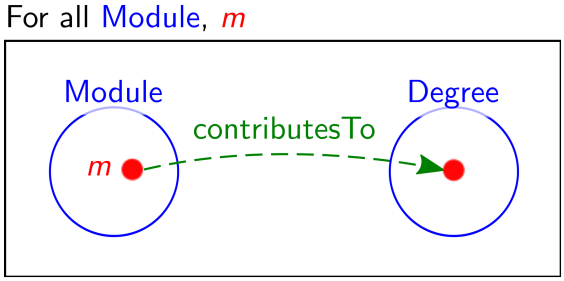
The textual annotation, For all Module, *m*, is a *quantification expression*. Moreover, *m* is an *unnamed* individual: it is given that *m* is a Module, but not a *particular* module.
A concept diagram expressing axiom 1 needs to include two curves, one for each mentioned class (namely Module and Degree). The two required curves, labelled Module and Degree can be drawn inside the boundary box. Since it is known that Module and Degree are disjoint, the curves need not overlap:



The diagram also needs to use two dots, representing individuals. As *m* is known to be a module, a dot representing this (unnamed) individual can be drawn inside (the curve labelled) Module. This dot needs to be labelled, because it must cross-refer to the use of *m* in the quantification expression. Moreover, a further unnamed individual is needed, because *m* contributes to *at least one* Degree. This unnamed individual is, thus, represented by a dot drawn inside Degree. In this case, the dot need not be labelled, as there is no need to cross-refer to that unnamed individual. Thus, the diagram above can be refined:



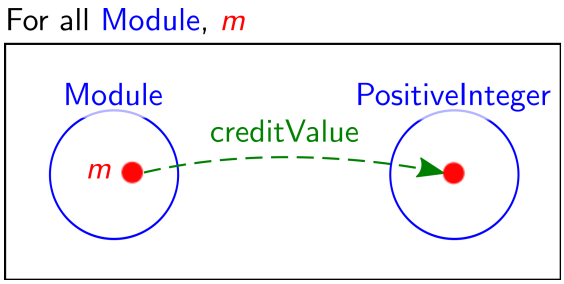
Given the quantification expression, the circle labelled Modules is actually redundant from the diagram; it has been included as a visual reminder that *m* is a Module. Lastly, an arrow needs to be added to express that *m* contributesTo the unnamed individual which is a Degree:



As with previous examples, there are many concept diagrams that could represent this information. The reader is encouraged to construct some.

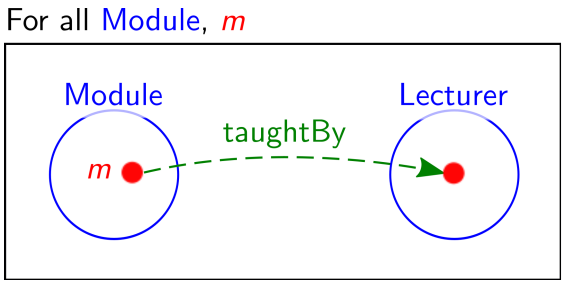
Example: Relationships Between Modules and Positive Integers

Axiom 2 expresses that each Module has a creditValue which is a PositiveInteger. This axiom is very similar to axiom 1 and can be expressed by the following concept diagram:



Example: Relationships Between Modules and Lecturers

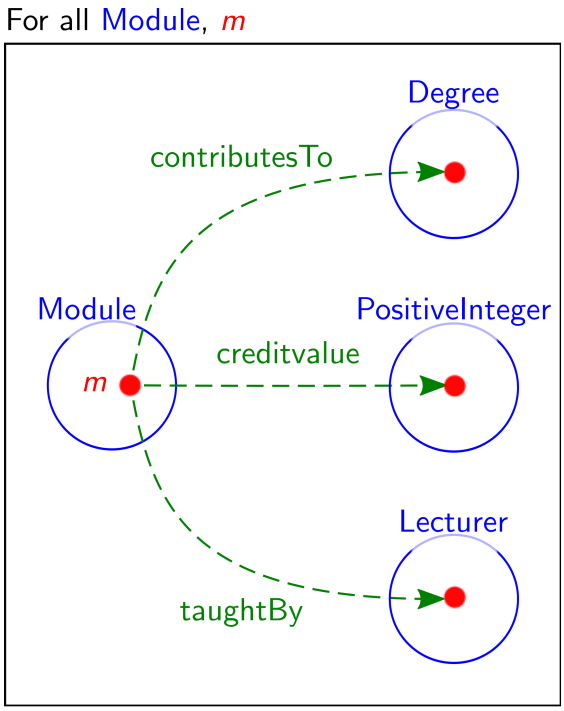
Axiom 5 expresses that each Module is taughtBy at least one Lecturer, expressed by:



The three axioms just defined using concept diagrams all have a very similar form and, moreover, all have the same quantification expression. This suggests that a single concept diagram can express the same information.

Example: Relationships Between Modules, Degrees, Positive Integers and Lecturers

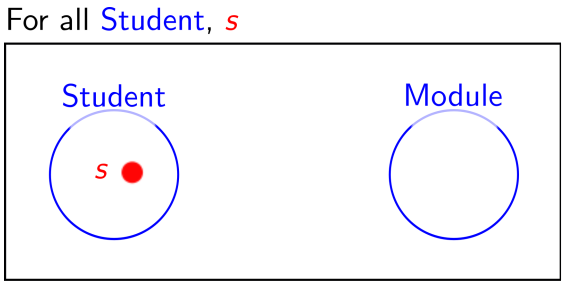
This concept diagram expresses all of the information in axioms 1, 2, and 6 and makes use of the known disjointness information:



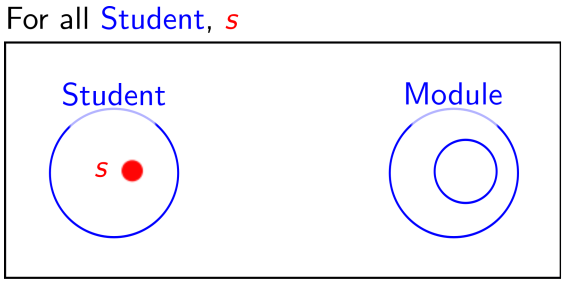
Attention now turns to the axioms concerning students and their properties, given in axioms 3, 4, and 5 on page 5. The examples now given for these axioms illustrate how the axioms may evolve as the ontology domain is better understood.

Example: Relationships Between Students and Modules

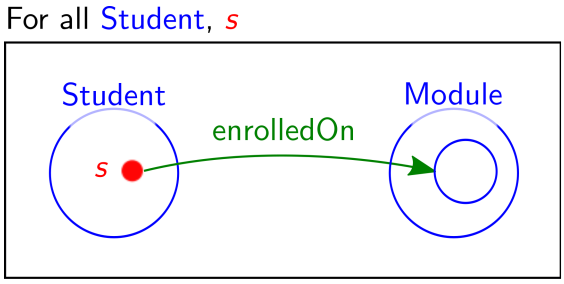
Axiom 3 expresses that each Student is enrolledOn a set of Modules. It is not stated whether students can be enrolled on things which are not modules. This example now assumes that the constraint is intended to be strong: students enrol only on modules.
A diagram expressing this information needs to make an assertion about *all* students and needs two curves, for the classes Student and Module. Since it is known that Student and Module are disjoint, the corresponding curves need not overlap. A diagram corresponding to axiom 3 also needs to use a single dot, representing individuals. As *s* is known to be a Student, a dot representing this (unnamed) individual can be drawn inside (the curve labelled) Student. Again, this dot needs to be labelled, because it must cross-refer to the use of *s* in the quantification expression. Thus, putting all of this together, the following diagram can be constructed:



As with the module axioms, the curve labelled Student is redundant. Next, a curve needs to be added for the unnamed class that is the set of modules on which *s* is enrolled:



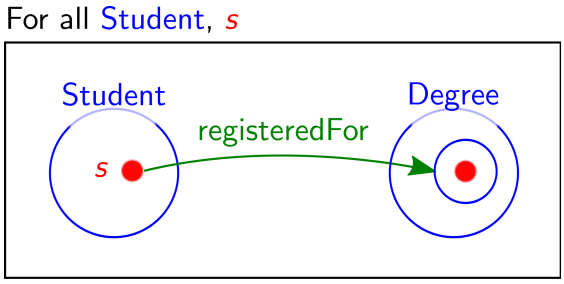
Lastly, a solid arrow needs to be added to express that *s* is enrolledOn all and only the set of objects in the set represented by the unnamed class:



Because the target of this solid arrow is contained by Module, the arrow asserts that *s* is enrolled on only modules.

Example: Relationships Between Students and Degrees

Axiom 4 expresses that each Student is registeredFor at least one Degree. Again, it may well have been the intention that students registered only for degrees, giving a strong form of the axioms. The concept diagram given here expresses this strong form:



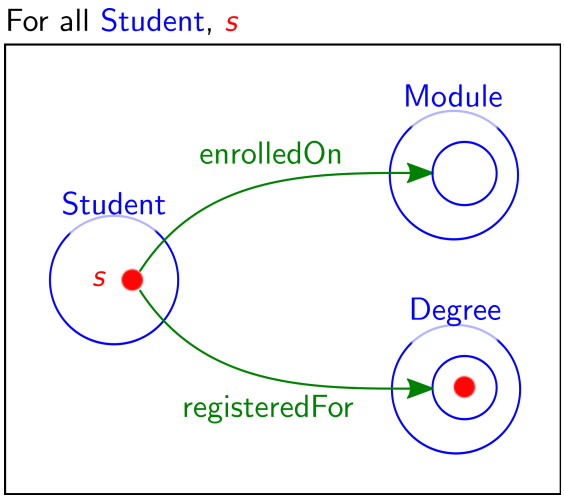
The unnamed individual represents the Degree on which *s* is registered.

As with earlier examples, it is possible to combine the diagrams for axioms 3 and 4 into a single diagram. This is readily possible because the quantification expressions are the same[4].

[4]Sometimes diagrams can be combined when the quantification expressions are different, but this needs considerably more care, as the meaning can be altered.

Example: Relationships Between Students, Modules and Degrees

Axioms 3 and 4 can be combined into a single concept diagram, as they have the same quantification expressions:



There is, of course, an onus on the ontology engineers to determine the number of diagrams used to convey information.

Tip: Combining Diagrams

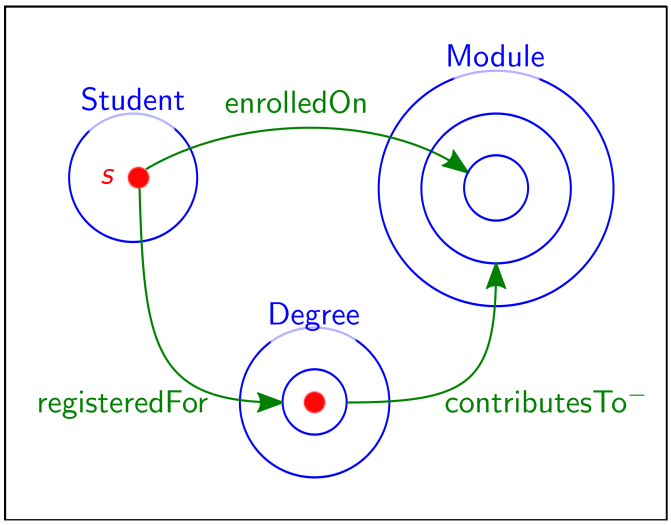
It is suggested that the ontology engineer avoids overly cluttered diagrams. Merging diagrams can be beneficial (and clutter avoiding) when the represented classes have many disjointness or subsumption relationships and the contained individuals are explicitly asserted to be, or are inferrably, distinct. Determining the balance between the number of diagrams and the amount of information conveyed in each diagram is a modelling decision. If two or more diagrams have the same quantification expression then it may be possible to combine them into one (or fewer) diagrams. Two diagrams that do not have the same quantification expression cannot necessarily be combined without changing the informational content.

There is still one axiom pertaining to students and their properties: axiom 5. This is the most complex axiom so far.

Example: Properties of Students

Axiom 5 makes reference to the modules on which a student is enrolled and the degrees on which they are registered. These unnamed classes were constructed in axioms 3 and 4. Fortunately, the diagram that combined axioms 3 and 4 can be extended to express 5:

For all Student, *s*



The newly added arrow is used to assert that the (unnamed) set of Degrees which *s* is registeredOn gives rise to an unnamed set of Modules. The elements of this unnamed set are precisely those which contributeTo this student's degrees and, moreover, include all of the modules which the student is enrolledOn. The diagram uses the *inverse* of contributedTo, denoted by contributedTo^{-}, to build this unnamed set.

The previous example made use of the in-built notation used by concept diagrams to access inverse relations.

Syntax: Property Inverses

The inverse of a property, say property, is denoted property$^-$.

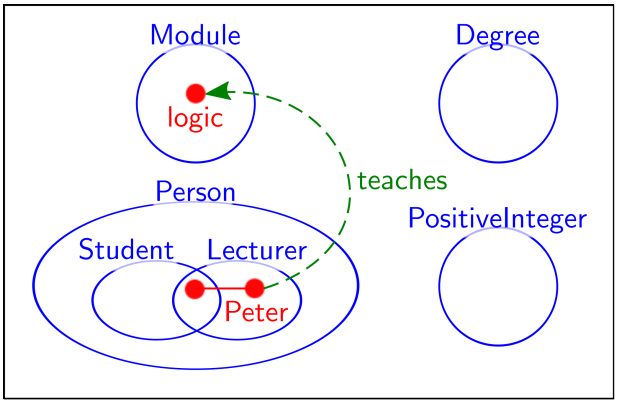
Semantics: Property Inverses

The inverse, property$^-$, of property is interpreted as the inverse binary relation. That is, one object, o_1, is related to another object, o_2 under property$^-$ if and only if o_2 is related to o_1 under property.

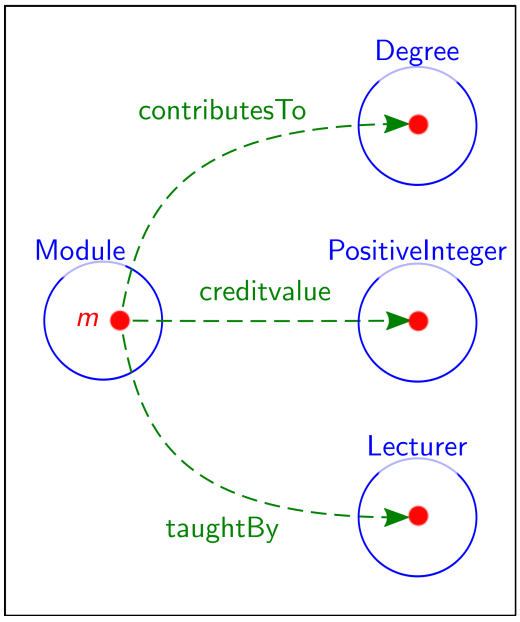
Tying together the Degree Courses scenario, many diagrams have been presented to define the axioms. However, not *all* of them are needed to capture the information given in the textually specified axioms.

Example: Necessary Axioms for the Degree Courses Scenario

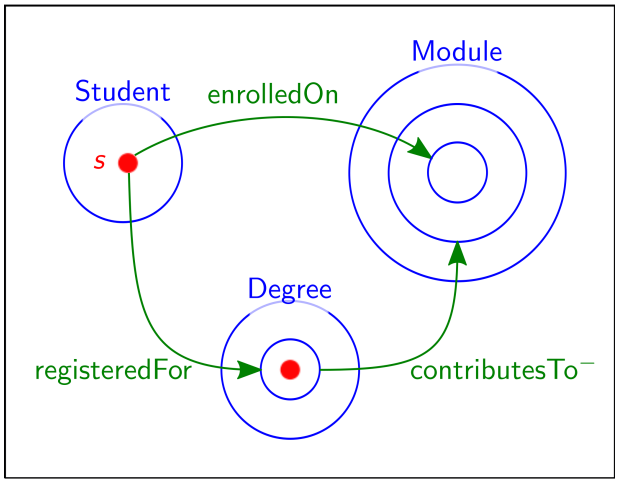
Only the following concept diagrams are needed to convey all of the information in axioms 1 to 16:



For all Module, *m*



For all Student, *s*



The diagrams in the above example were built by considering how to merge informational content in two or more diagrams into a single diagram. This activity *constitutes* reasoning about the ontology and will be revisited in later chapters.

2.4 Summary

This chapter has introduced *vocabulary* and *axioms*. The vocabulary comprises the labels used to identify individuals, classes, and properties, and these could be named or unnamed. If one of these entities is named then it represents a known (specific) entity in the domain. Unnamed properties are useful when, for example, a set of objects needs to be referenced but that set does not have a specific name. As demonstrated, the vocabulary need not be predetermined and it can evolve over time.
Axioms are conditions (sometimes called constraints) that the individuals, classes and properties must satisfy. In this sense, each concept diagram produced by the ontology engineer is an axiom. It is the axioms that form the ontology. It is common that axioms need to be revisited, extended or fine tuned as the ontology engineers gain a greater understanding of the domain for which the ontology is being built.

Chapter 3

Simple Diagrammatic Patterns for Ontology Engineering

This chapter presents a set of diagrammatic patterns for ontology engineering. Major benefits of using patterns include the simplification of the modelling process and the provision of a consistent approach to specifying commonly occurring constructions. Patterns give ontology engineers convenient access to these constructions, something which may be particularly useful to novice engineers or less technical domain experts. Similar to the design patterns of object-oriented software development, diagrammatic patterns provide a common language for analyzing and sharing reusable, composable design abstractions. The diagrammatic ontology engineering patterns we present in this chapter are lower-level than a typical design pattern from software development but fulfil the same role and serve as building blocks for more complex expressions. At the end of this chapter, the reader should:

1. Understand how to apply a range of simple patterns that relate only to classes, for *subsumption*, *disjointness*, and *equivalence*.
2. Understand how to apply patterns for *Some Values From* and *All Values From* constraints on classes and properties.
3. Understand how to apply patterns to define the *domains* and *ranges* of properties.
4. Understand how to apply patterns that relate only to properties, for *subsumption*, *disjointness*, and *equivalence*.

A key feature of these patterns is that they minimize the use of explicit quantification and use of variables. This eases the process of merging diagrams into single diagrams, as will be demonstrated later. Examples of how the patterns are applied in practice are given, continuing with (an extended version of) the Degree Courses Scenario as a case study. The examples have been chosen to illustrate the application of all of the above patterns. In chapter 4 some of these examples will be used to demonstrate how diagrammatic patterns can be merged, in order to reduce the number of diagrams needed to define the ontology.

3.1 Patterns for Classes

Common constraints that are imposed on the relationships between classes are subsumption (subset), disjointness and equivalence. This section provides six patterns for these constraints: a simple pattern for each of subsumption, disjointness and equivalence, together with a generalized pattern. As already seen, to express that one class is subsumed by another class, concept diagrams use curve containment, reflected in the first pattern. For the examples in this section, we begin by extending the Degree Courses scenario.

Scenario: Degree Courses

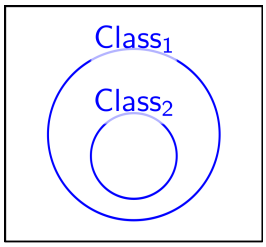
Degrees are sometimes called *courses* instead. The overall delivery of each degree is the responsibility of a *course leader*, who must be a *lecturer*; some lecturers are *course leaders*. As well as students and lecturers, there are *administrators*, who are people. This gives rise to three new terms in the vocabulary: Course, CourseLeader and Administrator. In addition, there are three more axioms:

17. Each Degree is a Course and vice versa.
18. Each Administrator is a Person.
19. Each CourseLeader is a Lecturer.

Axioms 18 and 19 are both subsumption relationships, a pattern for which is now given.

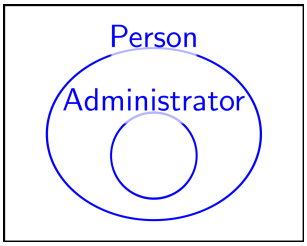
Pattern 1: Class Subsumption

Class_1 subsumes Class_2:

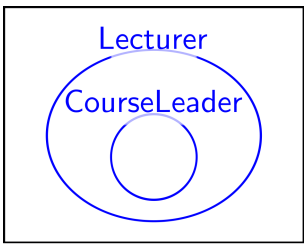


Example: Degree Courses Scenario, Class Subsumption

The Class Subsumption pattern can be applied to express Administrator is subsumed by Person, which is axiom 18:



In addition, it can be applied to express CourseLeader is subsumed by Lecturer, which is axiom 19:



Together with the axioms given previously, the above diagram implies that CourseLeader is subsumed by People.

Exercise: Degree Courses Scenario, Class Subsumption

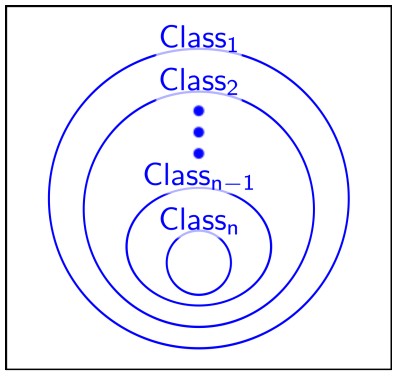
As well as the previously introduced classes, the Degree Courses Scenario also includes Exam, Test, Graduate, HonoursDegree, and PassDegree. Apply the Class Subsumption pattern to express each of the following:

1. Assessment subsumes Exam.
2. Assessment subsumes Test.
3. Person subsumes Graduate.
4. Degree subsumes HonoursDegree.
5. Degree subsumes PassDegree.

Pattern 1 generalizes to assert that a *subsumption hierarchy* exists between n classes. In the diagram below, the ellipsis indicate the presence of a further $n - 4$ circles labelled in the obvious fashion.

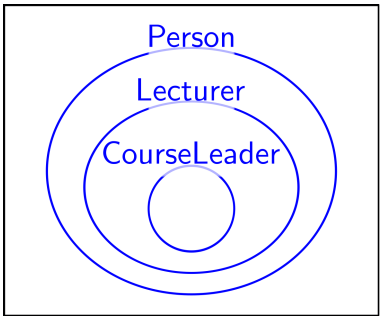
Pattern 2: General Class Subsumption

Class_1 subsumes Class_2, and ..., and Class_{n-1} subsumes Class_n:



Example: Degree Courses Scenario, General Class Subsumption

The General Class Subsumption pattern can be applied to express CourseLeader is subsumed by Lecturer which, in turn, is subsumed by Person:



Thus, the above diagram expresses axiom 19 together with the information about lecturers and people given in axiom 7.

Exercise: Degree Courses Scenario, General Class Subsumption

The Degree Courses Scenario can be extended to include information about *assessments*. Assessments can be exactly one of three types: *exams*, *tests*, and *course work*. Furthermore, course work *online*. Apply the General Class Subsumption pattern to express the following:

1. Assessment subsumes CourseWork which, in turn, is subsumed by ElectronicCourseWork.

Later examples will make use of the other vocabulary terms just introduced.

As seen above, concept diagrams exploit spatial properties to make assertions about classes in the ontology. Traditional approaches use stylized, textual syntax to expresses these, and other, assertions. Commonly used syntax includes Description Logic (DL) and OWL.

Symbolic Notation: Class Subsumption

To express class subsumption in DL, one asserts

$$\mathsf{Class}_2 \sqsubseteq \mathsf{Class}_1.$$

The OWL syntax is

$$\texttt{SubClassOf}(\mathsf{Class}_2\ \mathsf{Class}_1).$$

In the generalized case, the DL approach mirrors the diagrammatic pattern, by concatenating subsumption assertions:

$$\mathsf{Class}_n \sqsubseteq \mathsf{Class}_{n-1} \sqsubseteq \ldots \sqsubseteq \mathsf{Class}_2 \sqsubseteq \mathsf{Class}_1.$$

In OWL, however, this can be expressed using one occurrence of

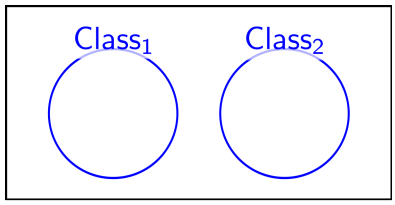
$$\texttt{SubClassOf}(\mathsf{Class}_i\ \mathsf{Class}_{i+1})$$

for each i between 1 and $n-1$.

Similarly, concept diagrams use curve disjointness (i.e. non-overlapping curves) to express class disjointness.

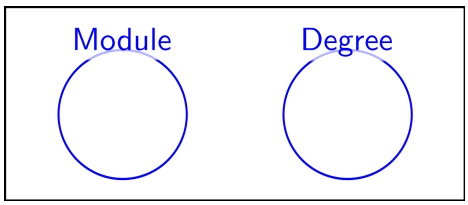
Pattern 3: Class Disjointness

Class_1 and Class_2 are disjoint:



Example: Degree Courses Scenario, Class Disjointness

The Class Disjointness pattern can be applied to express Module and Degree are disjoint, as seen in chapter 2:



Exercise: Degree Courses Scenario, Class Disjointness

Apply the Class Disjointness Pattern to define the following axioms:

1. HonoursDegree and PassDegree are disjoint.
2. Student and Graduate are disjoint[1].

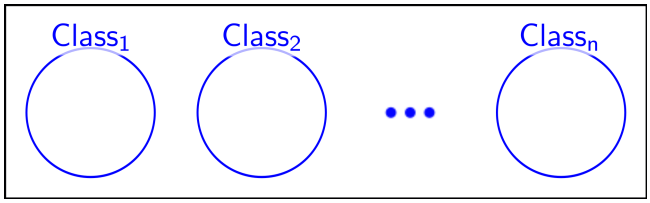
Pattern 3 has an obvious generalization to (concisely) assert that n classes are pairwise disjoint

[1] One could certainly question this modelling decision because it prevents graduates from also being students. However, given that University systems seldom work in a sensible way, one could argue that this a realistic view of how some domains may be modelled!

(that is, any pair of the n classes are disjoint). In the diagram below, the ellipsis indicate the presence of a further $n - 3$ circles labelled in the obvious fashion.

Pattern 4: General Class Disjointness

$Class_1$, $Class_2$, ..., $Class_n$ are pairwise disjoint:



Exercise: Degree Courses Scenario, General Class Disjointness

Apply the General Class Disjointness pattern to express the following:

1. Exam, Test and Coursework are pairwise disjoint.

Symbolic Notation: Class Disjointness

To express (simple) Class Disjointness in DL, one asserts

$$Class_1 \sqcap Class_2 \sqsubseteq \bot.$$

For General Class Disjointness, using DL, one axiom is required for each pair of classes:

$$Class_1 \sqcap Class_2 \sqsubseteq \bot, ..., Class_1 \sqcap Class_n \sqsubseteq \bot, ..., Class_{n-1} \sqcap Class_n \sqsubseteq \bot.$$

The OWL syntax is more succinct:

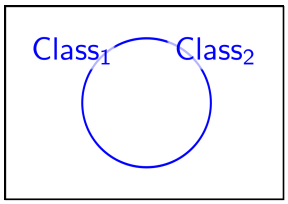
$$\texttt{DisjointClasses}(Class_1 ... Class_n).$$

Ontology engineers often want to express that two (or more) classes are equivalent. As with the

other patterns, there are many semantically equivalent, but syntactically different, concept diagrams that express class equivalence. The following pattern employs two overlaying (completely concurrent) curves.

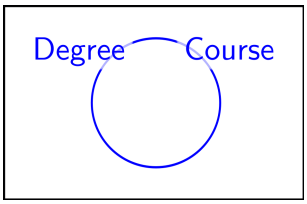
Pattern 5: Class Equivalence

Class_1 and Class_2 are equivalent:



Example: Degree Courses Scenario, Class Equivalence

The Class Equivalence pattern can be applied to express Degree and Course are equivalent (contain the same individuals), asserted in axiom 17:



Exercise: Degree Courses, Class Equivalence

Apply the Class Equivalence pattern to express each of the following:

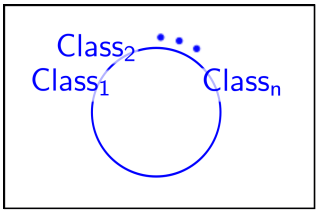
1. Lecturer and Academic are equivalent.
2. Student and Pupil are equivalent.

Again, the Class Equivalence pattern has an obvious generalization to the n-class case: to express that n classes are equivalent draw n overlaying curves. In the diagram below, the ellipsis indicate

the presence of a further $n - 3$ labels.

Pattern 6: General Class Equivalence

$Class_1$, $Class_2$,..., $Class_n$ are equivalent:



Symbolic Notation: Class Equivalence

In DL, Class Equivalence can be expressed by

$$Class_1 \equiv Class_2.$$

For General Class Equivalence, using DL, one axiom can be written for each pair of classes:

$$Class_1 \equiv Class_2, ..., Class_1 \equiv Class_n, ..., Class_{n-1} \equiv Class_n.$$

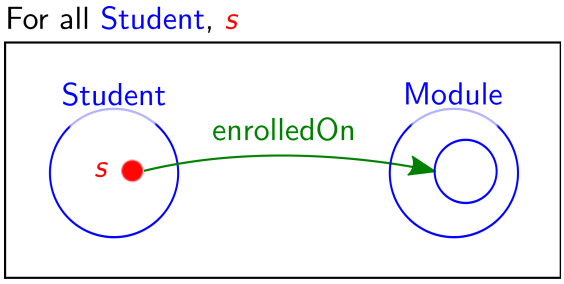
The OWL syntax is quite succinct:

$$\texttt{EquivalentClasses}(Class_1 \ ... \ Class_n).$$

In the above patterns, the classes have been represented by labelled curves. The represented class is either atomic (and, thus, has a specific interpretation) or is a constructed (unnamed) class, built using property restrictions (complex class expressions). All of the examples given above use only atomic classes. Concept diagrams use arrows to assert that curves represent complex class expressions such as $\forall P.C$; the details are omitted for space reasons. These curves can be used in the patterns to achieve required constraints.

3.2 Patterns for Property Restrictions: All Values From and Some Values From

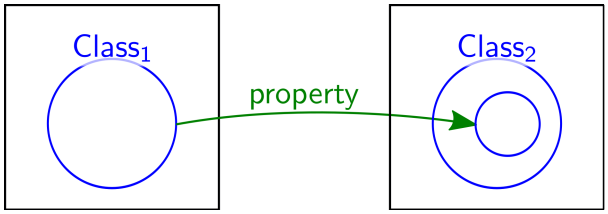
Common property restrictions are to enforce 'All Values From' and 'Some Values From' constraints, within the context of a class. Chapter 2 demonstrated how to express these types of restrictions in using quantifiers. For example, one 'All Values From' restriction told us that students can enroll only on modules:



Here, the quantification expression For all Student, s was used to ensure the concept diagram is making an assertion about all students. However, it is not usually necessary, certainly in simple cases, to use quantification expressions for 'All Values From' and 'Some Values' From property restrictions.

Pattern 7: All Values From

All individuals in Class_1 have all values, under property, from Class_2:



The arrow in the above diagram formally asserts that the image[2] of property (considering property as a binary relation), when its domain is restricted to Class_1, is a subset of Class_2. In other words, the only things that individuals in Class_1 are related to, under P, must be in Class_2. The use of multiple bounding boxes ensures that no unintended disjointness information between classes is asserted.

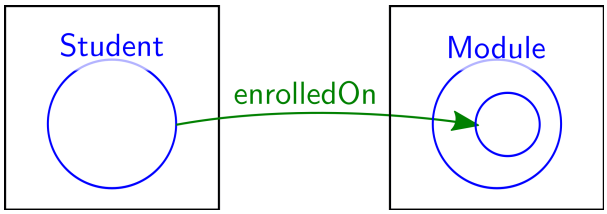
[2]The image of a property is the set of all elements related to by some element.

Tip: The Meaning of All Values From

Note that All Values From constraints provide subsumption information, not equivalence information. That is, if individuals in Class_1 have all values, under property, from Class_2 then this does not mean that each individual in Class_1 is related to *all of the individuals* in Class_2 under property. It merely asserts that each individual in Class_1 can only be related to individuals in Class_2 under property.

Example: Degree Courses Scenario, All Values From

The All Values From pattern can be applied to express all individuals in the class Student have all values, under enrolledOn, from Module:



Exercise: Degree Courses Scenario, All Values From

Apply the All Values From pattern to express each of the following:

1. Individuals in the class Student have all values, under registeredFor, from Degree.
2. Individuals in the class Module have all values, under contributesTo, from Degree.
3. Individuals in the class Administrator has all values, under administers, from Module.
4. Individuals in the class Exam have all values, under duration, from PositiveInteger.
5. Individuals in the class Test have all values, under duration, from PositiveInteger.

The last two axioms here include a property whose domain is neither Exam nor Test.

Symbolic Notation: All Values From

In DL, the All Values From pattern is captured by

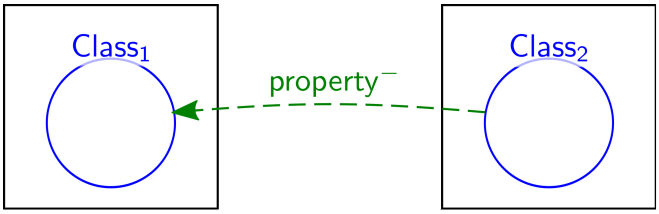
$$\mathsf{Class}_1 \sqsubseteq \forall \mathsf{P}.\mathsf{Class}_2.$$

To express All Values From property restrictions in OWL, one writes

$$\texttt{SubClassOf}\big(\mathsf{Class}_1\ \texttt{ObjectAllValuesFrom}(\mathsf{property}\ \mathsf{Class}_2)\big).$$

Pattern 8: Some Values From

Individuals in Class_1 have at least one value, under properties, from Class_2 :



The above diagram makes use of the *inverse*[3] of property. To justify the correctness of the Some Values From pattern, consider an individual, c_1, in the class Class_1. The pattern must ensure that c_1 has a value, c_2, from Class_2 under property. Well, c_1 has such a value, c_2, if and only if c_2 has the value c_1 under property$^-$. Equivalently, the image of property$^-$ when its domain is restricted to Class_2 includes at least all of the individuals in Class_1, captured by the dashed arrow.

Example: Degree Courses Scenario, Some Values From

The Some Values From pattern can be applied to express individuals in the class Module have at least one value, under contributesTo, from Degree:

Exercise: Degree Courses Scenario, Some Values From

Apply the Some Values From pattern to express each of the following:

1. Individuals in the class Module have at least one value, under contributesTo, from Degree.

[3]The inverse of property is the set of all pairs, (x, y), where y is related to x under property.

2. Individuals in the class Student have at least one value, under registeredFor, from Degree.
3. Individuals in the class Administrator have at least one value, under administers, from Module.
4. Individuals in the class Exam have at least one value, under duration, from PositiveInteger.
5. Individuals in the class Test have at least one value, under duration, from PositiveInteger.

Symbolic Notation: Some Values From

In DL Some Values From is expressed by

$$\mathsf{Class}_1 \sqsubseteq \exists \mathsf{property}.\mathsf{Class}_2.$$

In OWL, this property restriction is expressed by:

$$\texttt{SubClassOf}\big(\mathsf{Class}_1\ \texttt{ObjectSomeValuesFrom}(\mathsf{property}\ \mathsf{Class}_2)\big).$$

3.3 Patterns for Property Restrictions: Domains and Ranges

The next three patterns concern *domains* and *ranges* of properties. The domain of a property is the set of all things that *can* be related to something, under *property*. The range of a property is the set of all things that *can* be related to by something, under *property*. For example:

Scenario: Degree Courses

The property *administers* relates only administrators to modules and nothing more, so the *domain* of *administers* is *Administrator* and the *range* is *Module*. This gives rise to one new term in the vocabulary: administers. In addition, we have two more axioms:

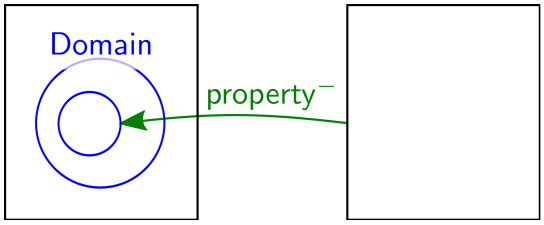
20. The domain of administers is Administrator.
21. The range of *administers* is Module.

This section demonstrates how to define these domain and range restrictions.

Firstly, consider the domain of a property. The domain of property is Domain if and only if the range of the inverse, property$^-$, of property is Domain.

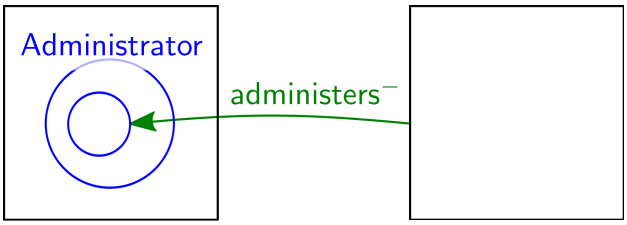
Pattern 9: Domain of a Property

The domain of property property is Domain:



Example: Degree Courses Scenario, Domain

The Domain of a Property pattern can be applied to express the domain of administers is the Administrator:



Exercise: Degree Courses Scenario, Domain

In the Degree Courses Scenario, students are enrolled on modules. Students can also *pass* or *fail* modules. In addition, students have a set of modules in which they are *referred*, and a set in which they are deferred. Apply the Domain of a Property pattern to express each of the following:

1. The domain of enrolled is Student.
2. The domain of passed is Student.

3. The domain of referred is Student.

The reader may also wish to define further domains, in the obvious fashion, for the remaining properties given above.

Later examples will make use of the vocabulary just introduced, that is, the properties passed, failed, referred, and deferred.

Symbolic Notation: Domain of a Property

The corresponding DL formalization of this pattern is

$$\forall \text{property}.\top \sqsubseteq \text{Domain};$$

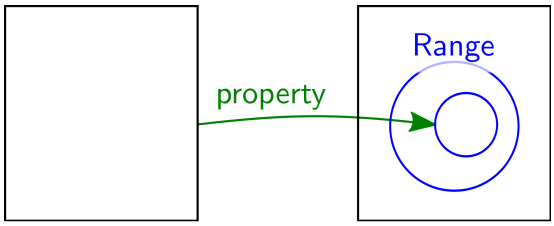
the construction $\forall \text{property}.\top$ builds the pre-image[4] of property. The Domain of a Property pattern employs the same style of construction: the arrow builds the pre-image of property. The OWL syntax more explicitly identifies that the domain is being defined:

`ObjectPropertyDomain`(property Domain).

The Range of a Property pattern constructs the image of property, using an arrow, and asserts that this image is subsumed by the range, Range.

Pattern 10: Range of a Property

The range of property is Range:

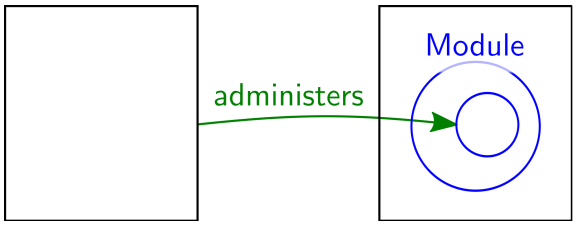


---

[4]The pre-image of property is the set of all individuals that are related to something under property.

Example: Degree Courses Scenario, Range

The Range of a Property pattern can be applied to express the range of administers is Module:



Exercise: Degree Courses Scenario, Range

Apply the Range of a Property pattern to express each of the following:

1. The range of fail is Module.
2. The range of deferred is Module.

The reader may also wish to define further ranges, in the obvious fashion, for the other properties considered in the Domain of a Property exercises.

Symbolic Notation: Range of a Property

In DL, the *range* is typically defined by

$$\top \sqsubseteq \forall P.R.$$

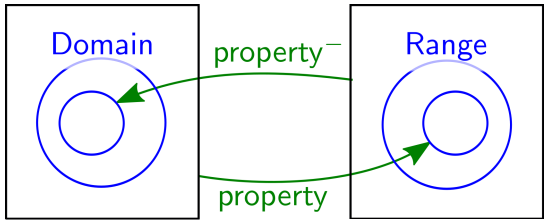
As with the concept diagram pattern, the range can also be defined in DL by constructing the pre-image of the inverse, $property^{-}$, of property and asserting that this pre-image is subsumed by Range: $\forall property^{-}.\top \sqsubseteq Range$. As with domain, in OWL the range is explicitly identified:

`ObjectPropertyRange`(property Range).

It is easy to provide a single pattern that allows ontology engineers to specify domain and range together.

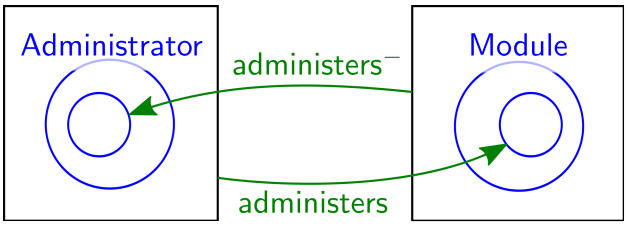
Pattern 11: Domain and Range of a Property

The domain and range of property are Domain and Range respectively:



Example: Degree Courses Scenario, Domain and Range

The Domain and Range of a Property pattern can be applied to express the domain of administers is Administrators and the range is Module:



Exercise: Degree Courses Scenario, Domain and Range

Apply the Domain and Range of a Property pattern to express each of the following:

1. The property enrolled has domain Student and range Module.
2. The property passed has domain Student and range Module.
3. The property contributesTo has domain Module and range Degree.
4. The property teaches has domain Lecturer and range Module.

3.4 Patterns for Properties

It is helpful to have standard patterns for expressing property subsumption, disjointness and equivalence, although they are perhaps not so commonly needed. This section presents six such patterns, two for each type of relationship. As with the properties for classes, each type of relationship will have a simple pattern, where just two properties are considered, and a general version for expressing, for instance, property hierarchies.

Scenario: Degree Courses

The property teaches relates lecturers to modules. Every module is *led by* a lecturer and this lecturer is said to *lead* that module. The lead lecturer must also teach on that module. In addition, students can have either *passed* or *failed* a module, but not both. If they have neither passed nor failed a module then they could have been *referred* or *deferred* in that module, but not both. This gives rise to two new terms in the vocabulary, all of which are properties: ledBy, leads, passed, failed, referred, deferred. The new axioms of interest to us in this section are:

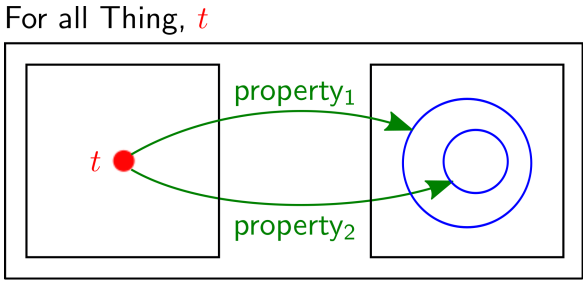
22. The property teaches subsumes leads.
23. The properties passed and failed are disjoint.
24. The properties passed and referred are disjoint.
25. The properties passed and deferred are disjoint.
26. The properties failed and referred are disjoint.
27. The properties failed and deferred are disjoint.
28. The properties referred and deferred are disjoint.
29. The property leads is (equivalent to) the inverse of the property ledBy.

This section demonstrates how to define these property relationships.

The first pattern we present, for property relationships, allows the assertion of a subsumption relationship. It uses the fact property_1 subsumes property_2 if and only if for all things, t, the set of things to which t is related under property_1 includes (subsumes) the set of things to which t is related under property_2.

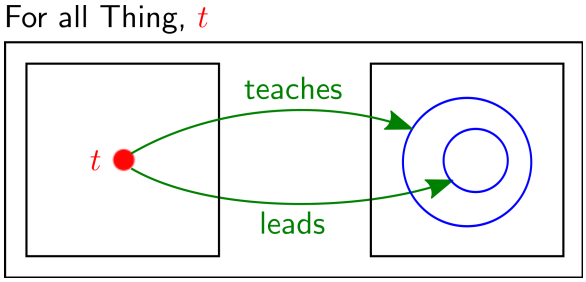
Pattern 12: Property Subsumption

property_1 subsumes property_2:



Example: Degree Courses Scenario, Property Subsumption

The Property Subsumption pattern can be applied to express property teaches subsumes leads, which is axiom 22:



Exercise: Degree Courses Scenario, Property Subsumption

Apply the Property Subsumption pattern to express each of the following:

1. enrolledOn subsumes referred.
2. enrolledOn subsumes deferred.

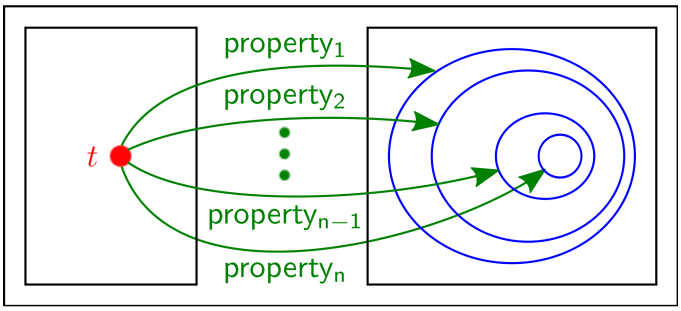
Pattern 12 generalizes to assert that a *subsumption hierarchy* exists between n properties, just as

was the case for the Class Subsumption pattern.

Pattern 13: General Property Subsumption

property_1 subsumes property_2, and ..., and property_{n-1} subsumes property_n:

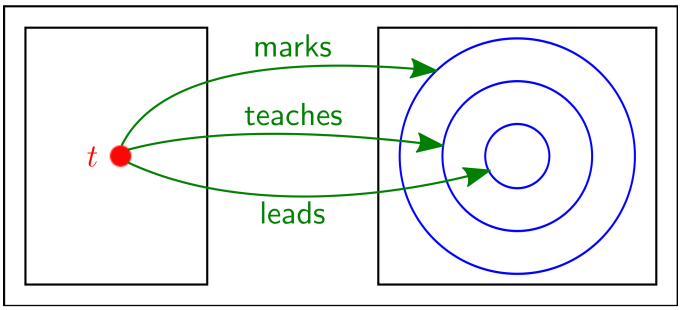
For all Thing, t



Example: Degree Courses Scenario, General Property Subsumption

In the Degree Courses Scenario, students' work has to be marked. The marking for each module is the responsibility of the person who leads the module, together with those who teach the module and other staff may be called on to contribute to the marking also. The General Property Subsumption pattern can be applied to express that marks subsumes teaches which, in turn, subsumes leads:

For all Thing, t



Symbolic Notation: Property Subsumption

In DL, property subsumption is readily expressed, using the same syntactic construction as for classes:

$$\text{property}_2 \sqsubseteq \text{property}_1$$

Again, expressions such as this can be concatenated to capture property hierarchies:

$$\text{property}_n \sqsubseteq \ldots \sqsubseteq \text{property}_{n-1} \sqsubseteq \ldots \sqsubseteq \text{property}_2 \sqsubseteq \text{property}_1.$$

In OWL, one asserts

$$\texttt{SubObjectPropertyOf}(\text{property}_2\ \text{property}_1).$$

As with class subsumption, one can then use $n-1$ of these expression, namely

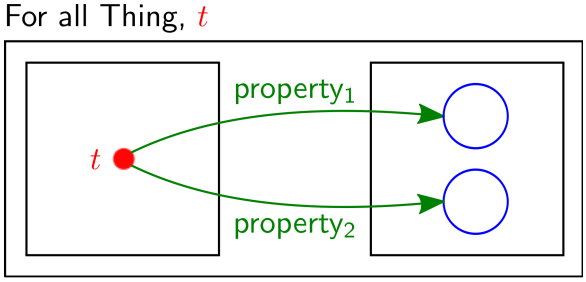
$$\texttt{SubObjectPropertyOf}(\text{property}_{i+1}\ \text{property}_i),$$

to capture a property hierarchy.

Patterns 11 and 12 both make use of curve containment to express subsumption relationships, giving some degree of *well-matchedness* to their informational content. Likewise, we can use non-overlapping curves (i.e. with disjoint interiors) to express disjointness relationships between properties.

Pattern 14: Property Disjointness

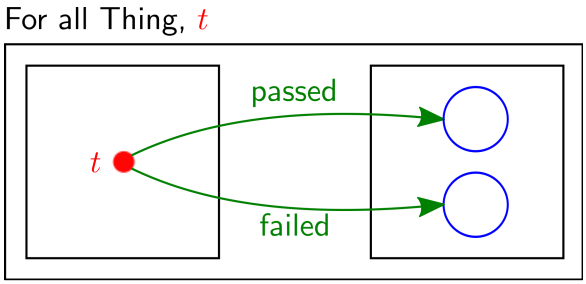
property_1 and property_2 are disjoint:



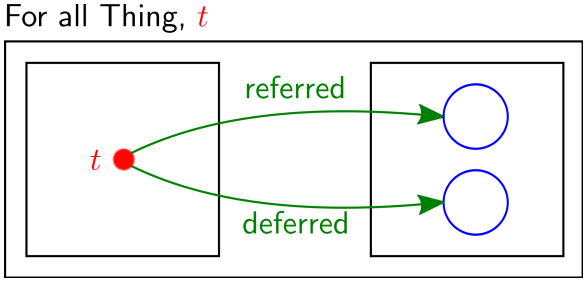
Example: Degree Courses Scenario, Property Disjointness

The Property Disjointness pattern can be applied to express passed and failed are disjoint, which

is axiom 23:



It can also be applied to express *referred* and *deferred* are disjoint, which is axiom 28:



Exercise: Degree Courses Scenario, Property Disjointness

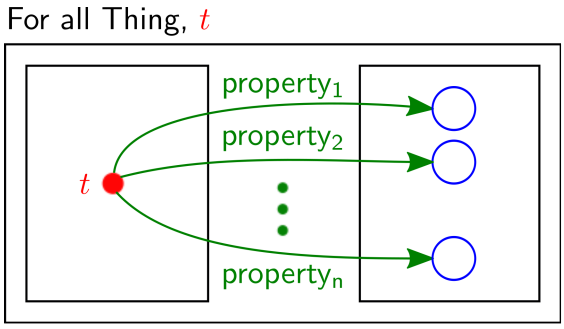
Apply the Property Disjointness pattern to express each of the following:

1. The properties passed and referred are disjoint (axiom 24).
2. The properties passed and deferred are disjoint (axiom 25).

As the exercise and examples for the Property Disjointness pattern demonstrate, it can be somewhat cumbersome to assert, separately, that a set of properties are pairwise disjoint. This motivates the following pattern, which generalizes the Property Disjointness pattern to permit succinct representation of such a set of constraints.

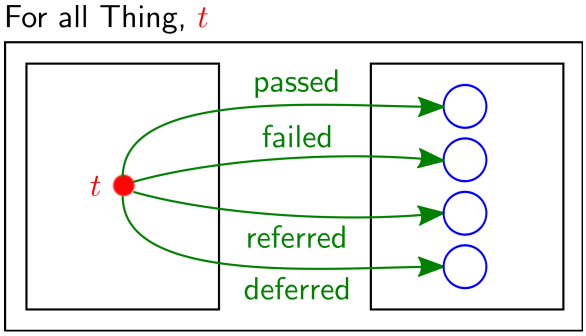
Pattern 15: General Property Disjointness

property_1, property_2, ..., and property_n are disjoint:



Example: Degree Courses Scenario, General Property Disjointness

The General Property Disjointness pattern can be applied to express axioms 23 to 28 in a single diagram:



Symbolic Notation: Property Disjointness

In DL, as with subsumption, the construction of property disjointness relationships is similar to that for classes:

$$\text{property}_1 \sqcap \text{property}_2 \sqsubseteq \bot \ .$$

For General Property Disjointness, using DL. one axiom is required for each pair of classes:

$$\text{property}_1 \sqcap \text{property}_2 \sqsubseteq \bot, \ldots, \text{property}_1 \sqcap \text{property}_n \sqsubseteq \bot, \ldots, \text{property}_{n-1} \sqcap \text{property}_n \sqsubseteq \bot \ .$$

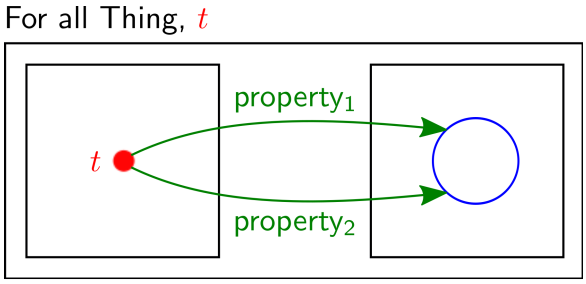
The OWL syntax is, again, more succinct:

`DisjointObjectProperties`(property$_1$ ... property$_n$).

Sometimes it is convenient to use two (or more) different names for the 'same' property. Different naming conventions can arise through different stakeholder backgrounds or, as modelling progresses, it may become apparent that two properties, originally thought to be potentially different, are actually equivalent. In such circumstances, one wishes to assert such equivalence. It is also helpful to assert equivalence in the context of inverse relations, where one wishes to give a particular name to the inverse. In the Degree Courses Scenario, the property leads is the name given to the inverse of ledBy.

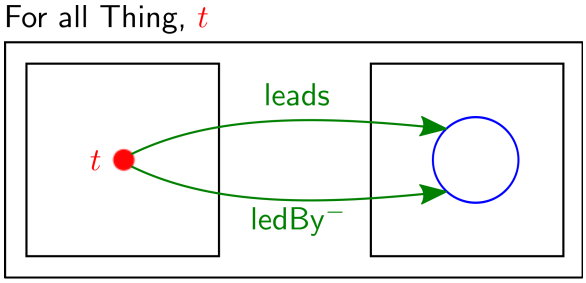
Pattern 16: Property Equivalence

$\mathsf{property}_1$ and $\mathsf{property}_2$ are equivalent:



Example: Degree Courses Scenario, Property Equivalence

The Property Equivalence pattern can be applied to express leads and ledBy^- are equivalent, which is axiom 29:



Exercise: Degree Courses Scenario, Property Equivalence

It is now given that each degree *comprises* a set of modules and that each of these modules must

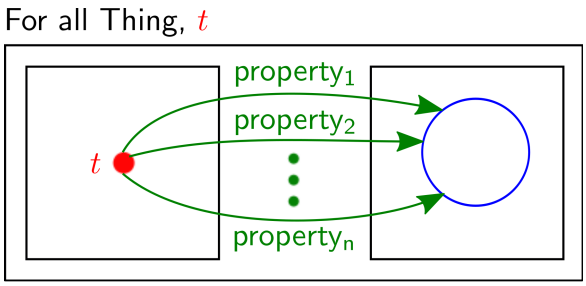
contribute to the degree. This relationships between the properties comprises and contributesTo can be ensured by expressing the following constraint:

1. comprises and contributesTo$^-$ are equivalent.

Use the Property Equivalence pattern to express this information.

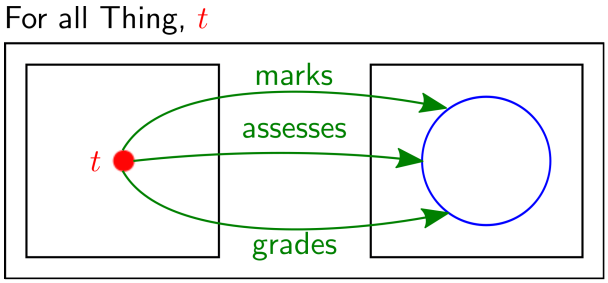
Pattern 17: General Property Equivalence

property$_1$, property$_2$, ..., property$_n$ are equivalent:



Example: Degree Courses Scenario, General Property Equivalence

Various terminology can be used to assert that a person responsible for marking a module, including assesses and grades. The General Property Equivalence pattern can be applied to express that marks, assesses and grades are equivalent:



Symbolic Notation: Property Equivalence

In DL, property equivalence can be expressed by

$$\mathsf{property}_1 \equiv \mathsf{property}_2.$$

For general class equivalence, one axiom can be written for each pair of properties:

$$\mathsf{property}_1 \equiv \mathsf{property}_2, ..., \mathsf{property}_{n-1} \equiv \mathsf{property}_n.$$

The OWL syntax is again quite succinct:

$$\texttt{EquivalentObjectProperties}(\mathsf{property}_1 \ ... \ \mathsf{property}_n).$$

3.5 Summary

This chapter has introduced *patterns* for classes and properties, including subsumption, disjointness and equivalence relationships. The generalized versions of these patterns afford more succinct expressions to be formulated than using the simple patterns multiple times. For instance, a single instance of the General Class Subsumption pattern to express that n classes are in a subsumption hierarchy corresponds to $n - 1$ instances of the Class Subsumption pattern.
In addition, patterns for property restrictions, domains and ranges have been defined. These patterns allow these commonly occurring constraints to be readily defined in a recognizable manner, which is just one of the advantages of utilizing patterns in ontology engineering. The next chapter establishes how patterns can be merged, reducing the number of diagrams required to model the domain. Merging can be advantageous in that some informational content becomes immediately visible to the reader, rather than implicit as can be the case when many axioms are utilized.

Chapter 4

Merging Simple Patterns

Each of the patterns introduced in chapter 3 has a relatively simple form, producing one diagram that represents exactly the required information. Whilst each of these diagrams exploits spatial relationships to assert information, collectively they may not deliver *all* of the information in the most succinct and accessible form. We envision an ontology editor that provides a pallet of patterns that can be selected and merged into a single diagram. The purpose of this chapter is to provide sound mechanisms by which ontology patterns can be merged.
At the end of this chapter, the reader should:

1. Understand how to merge patterns for *classes*.
2. Understand how to merge patterns for *All Values From* and *Some Values From*.
3. Understand how to merge patterns for *Domain* and *Range*.
4. Understand how to merge patterns for *properties*.

Given any two patterns (or, in general, any two diagrams arising from the merger of patterns), it is always possible to merge them into a single diagram. However, there is a balance to be struck between informational content of the diagram and its readability. Therefore, there is a need to *design*, or *engineer*, the final ontology in a manner deemed readable. This readability aspect will be discussed through the chapter.

4.1 Merging Patterns for Classes

To merge the information conveyed in one diagrammatic pattern into another, the steps needed to add a piece of syntax to a diagram in a sound manner need to be considered. In the case of patterns for classes, the only syntactic elements involved are closed curves. In order to merge two diagrams arising from the patterns for classes, one needs to understand how to add a closed curve, c, to a diagram, d. Doing this in an intuitive manner requires understanding the information c conveys and how that relates to the information already present in d. Now, all of the closed curves each represent a set, that set denoted by the curve's label.
The *regions* in the plane formed by the closed curves also represent sets. These regions are called *zones*: a zone is a region that is inside some (possibly no) curves contained by some boundary

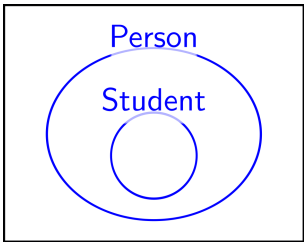
rectangle, b, and outside the rest of the curves contained by b. In general, zones represent the set that is a subset of the intersection of the sets represented by the curves it is inside, less the union of the sets represented by the curves it is outside.
When adding a curve to a diagram, the impact on the diagram's zones needs to be described. Given a diagram, d, to which a curve, c, is to be added, each zone, z, will become inside c, outside c, or split by c, as follows (blurring the distinction between syntax and semantics)[1]:

1. if z is subsumed by c then z is inside c,
2. if z is is disjoint from c then z is outside c,
3. otherwise, z is split into two zones by c.

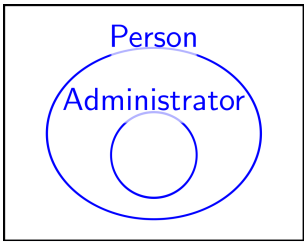
This process is now illustrated by example.

Example: Degree Courses Scenario

In the Degree Courses Scenario, the Class Subsumption pattern is used to assert that Person subsumes Student:

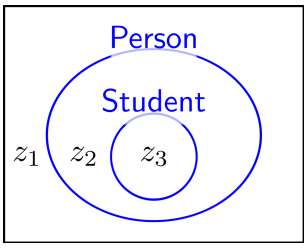


This pattern is also used to assert that Person subsumes Administrator:

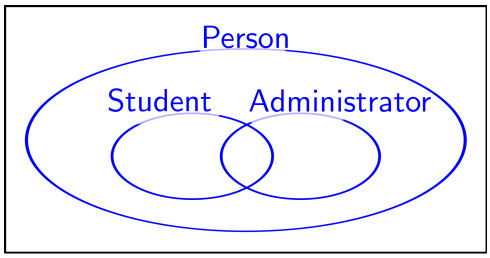


These two patterns can be *merged* into a single diagram as follows. Noting that (the curve labelled) Person occurs in both diagrams, the merging them requires, say, Administrator to be copied into the other diagram. Each of the three zones, z_1, z_2 and z_3, are considered in turn:

[1] It is possible to define the addition of c entirely syntactically, without reference to the semantics. However, the formal details are omitted.



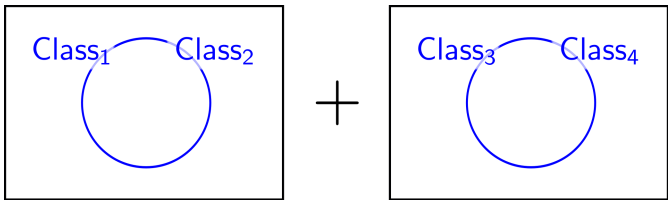
The zone z_1 is *disjoint* from Administrator, because z_1 is *disjoint* from Person and Administrator is subsumed by Person. So, z_1 is outside Administrator. The zone z_2 is neither subsumed by nor disjoint from Administrator, so z_2 is split. Likewise, z_3 is split; intuitively, no information is provided about whether Administrator and Student are disjoint, say. As the alterations to all three zones have now been described, these two instances of the Class Subsumption pattern can now be merged:



The above example demonstrates how to merge two diagrams arising from the Class Subsumption pattern. The same approach is followed for merging any two diagrams arising from either the Class Disjointness pattern or the Class Equivalence pattern. In the case of merging two instances of the Class Equivalence pattern, there are only four possible configurations of curves in the merged diagram.

Tip: Merging Instances of the Class Equivalence Pattern

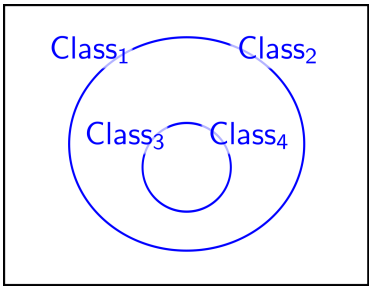
Suppose one wishes to merge two diagrams arising from the Class Equivalence pattern, given here in a generic form:



These two diagrams give that Class_1 and Class_2 are equivalent and that Class_3 and Class_4 are equivalent. To merge these two diagrams, therefore, one only needs to establish the relationship between one of the classes in the lefthand diagram and one of the classes in the righthand

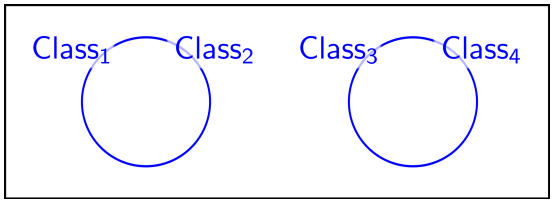
diagram. Without loss of generality, suppose the relationship between Class_1 and Class_3 is known. There are, essentially, only four cases:

1. Class_1 subsumes Class_3. The merged diagram is:

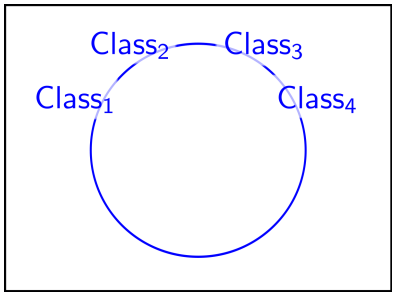


   The merged diagram obtained if Class_3 subsumes Class_1 is as above, but with the labels on the curves swapped.

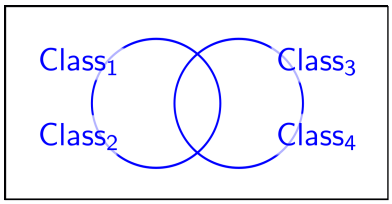
2. Class_1 and Class_3 are disjoint. The merged diagram is:



3. Class_1 and Class_3 are equivalent. The merged diagram is:



4. No known relationship exists between Class_1 and Class_3. That is, no information is provided that allows a subsumption, disjointness, or equivalence relationship to be deduced. In this case, the merged diagram is:



Indeed, these merger rules apply readily to diagrams formed from the General Class Equivalence pattern in the obvious manner.

Exercise: Degree Courses Scenario: Merging Patterns from Classes

In each case below:

(i) Draw the two diagrams arising from the application of the appropriate pattern.

(ii) Identify the zones in one of the two diagrams (the diagram chosen to have the curves from the other diagram copied in to it).

(iii) Select a curve that needs to be copied and determine whether each zone is to be inside, outside or split by the curve.

(iv) Copy the selected curve in the appropriate manner.

(v) If two curves need to be copied then determine how to copy the second curve: follow the same algorithmic process, which involves identifying the zones in the diagram after copying the first curve and so forth.

In all cases, however, the reader may prefer to adopt a more intuitive approach to merging the diagrams than following the more prescribed approach above.

1. Assessment subsumes Exam. Assessment subsumes Test.
2. Person subsumes Graduate. Student and Graduate are disjoint.
3. Person subsumes Student. Student and Pupil are equivalent.
4. HonoursDegree and PassDegree are disjoint. Degree and Course are equivalent.
5. Lecturer and Academic are equivalent. Student and Pupil are equivalent.
6. Person subsumes Graduate. Degree subsumes PassDegree.

Apply the same principles to merge three instances of patterns: Person subsumes Graduate; Person subsumes Student; and Degree subsumes PassDegree.

Tip: Merging Patterns in Complex Ontologies

When merging patterns, it may not be immediately obvious whether a subsumption or disjointness relationship exists between (the set denoted by) a zone and (the set denoted by) a class. If

unsure, split the zone when adding the curve for the class. This procedure ensures that no new information is introduced.

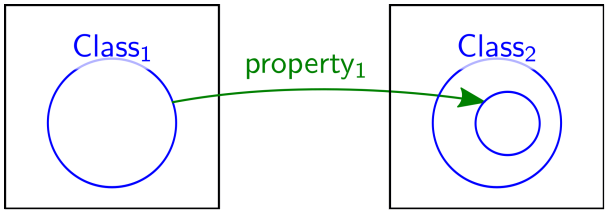
4.2 Merging Patterns for All Values From and Some Values From

A key advantage of merging instances of All Values From and Some Values From patterns is when the two to-be-merged diagrams have a class in common. This section demonstrates how to merge such diagrams. Such mergers have two key stages, reflecting that *curves* and *arrows* are now present: identify common classes and then 'merge the rest' using techniques described below.

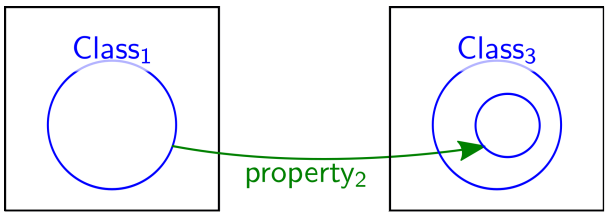
4.2.1 Merging All Values From Patterns

This section is subdivided to cover the following cases:

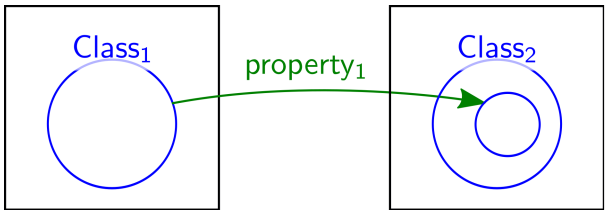
1. *Common Source Classes* Merging two instances of All Values From patterns of the form



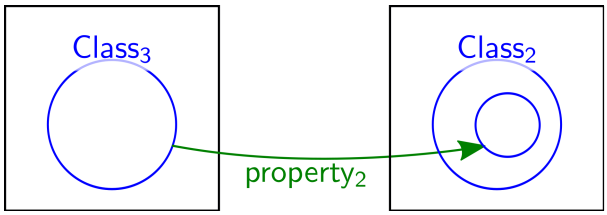
and



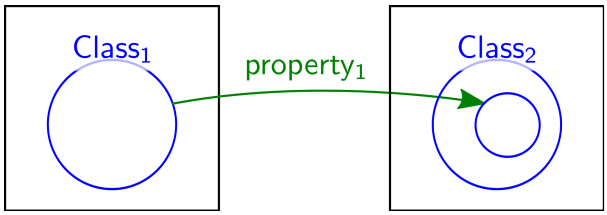
2. *Common Target Classes* Merging two instances of All Values From patterns of the form



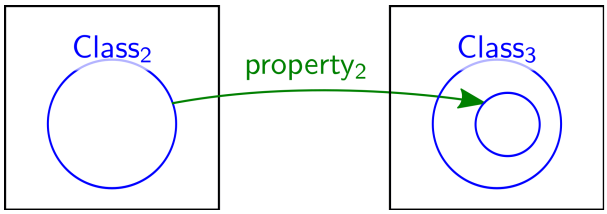
and



3. *Common Source Class and Target Class* Merging two instances of All Values From patterns of the form



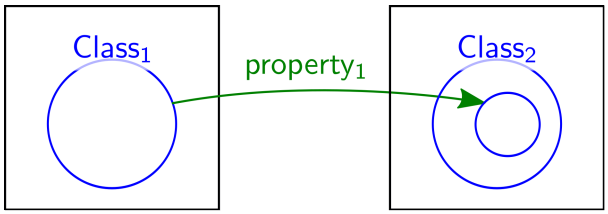
and



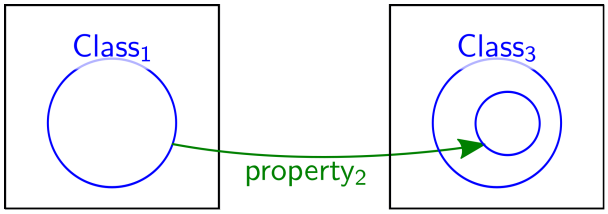
The remaining cases are left to the reader, using the principles of curve addition previously described.

All Values From: Merging with Common Source Classes

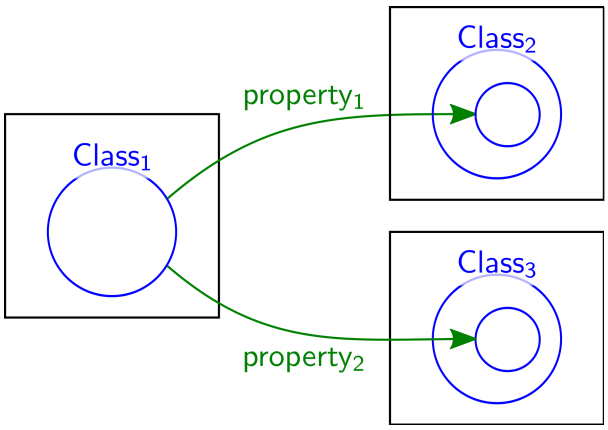
This subsection demonstrates how to merge two instances of the All Values From pattern where the two *source classes* are the same:



and

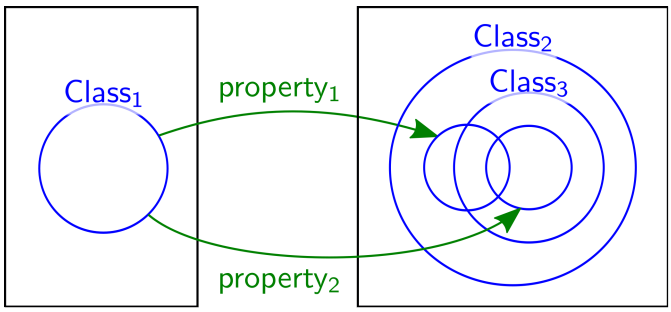


The first step in the merger is trivial:



The diagram above is obtained by merging the syntax in the two lefthand boundary boxes[2]. If no information is available about the relationship between Class_2 and Class_3 then the above diagram is the result of the merger[3]. However, if a subsumption, disjointness of equivalence relationship is known to hold between Class_2 and Class_3 then the result is as follows:

1. Class_2 subsumes Class_3:

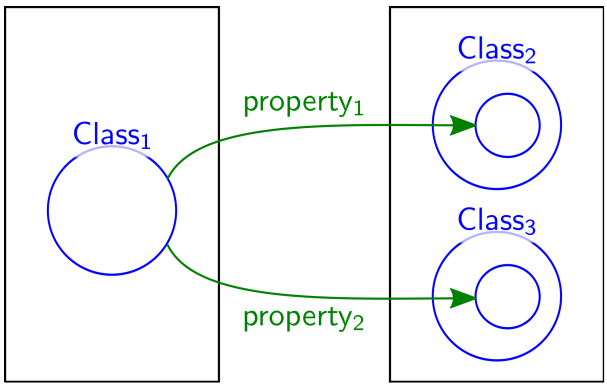


If Class_3 subsumes Class_2 then the above diagram is altered in the obvious way.

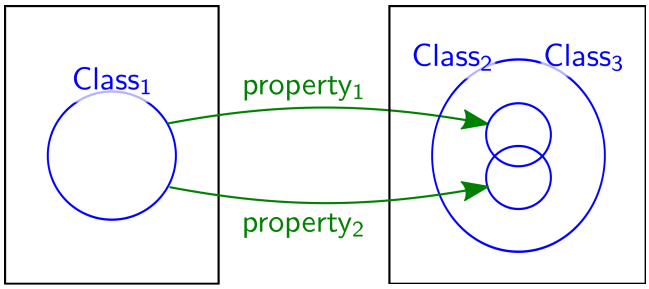
[2] All mergers are, essentially, mergers of information in two or more boundary boxes, possibly using information derivable from other diagrams in the ontology.

[3] Note that one can, however, proceed to merge the two diagrams containing the arrow targets, but this *can* lead to a cluttered (less readable) diagram. It is at the discretion of the ontology engineer to decide whether to continue merging aspects of these pattern instances.

2. Class_2 and Class_3 are disjoint:

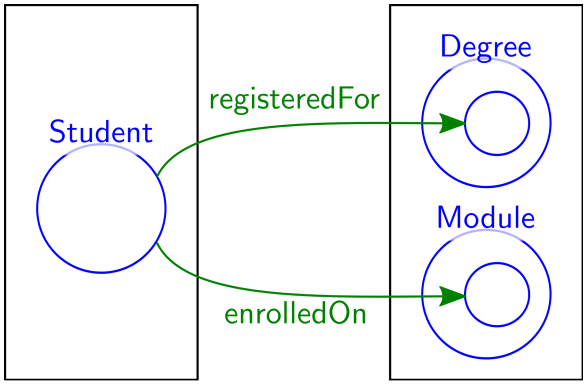


3. Class_2 and Class_3 are equivalent:

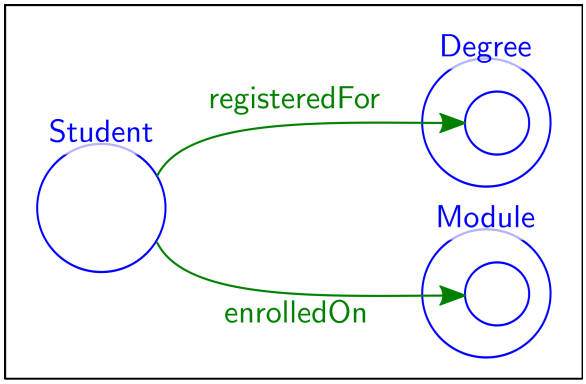


Example: Degree Courses Scenario

As seen in chapter 3, individuals in the class Student have all values, under registeredFor, from Degree. Likewise, individuals in the class Student have all values, under enrolledOn, from Module. Merging the two instances of the All Values From pattern arising from these two property restrictions yields:



It is left to the reader to derive the original pattern instances. In fact, as it is known that Student, Degree and Module are pairwise disjoint, further merging can be conducted:



As with all of the other merging that has been demonstrated, the informational content in two boundary boxes has been combined into a single box.

Exercise: Degree Courses Scenario

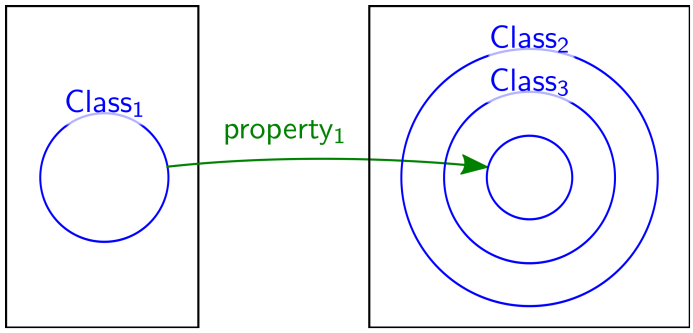
The following three All Values From constraints hold:

1. Individuals in the class Administrator have all values, under administers, from Module.
2. Individuals in the class Administrator have all values, under keepsRecordsAbout, from People.
3. Individuals in the class Administrator have all values, under recordsMarksFor, from Student.

Draw an appropriate instance of the All Values From pattern for each of the above property restrictions to produce a single diagram. Merge each pair of pattern restrictions. As a more advanced (optional) exercise, merge all three of these instances of the All Values From pattern into a single diagram.

Tip: Merging when the Properties are the Same

If $\text{property}_1 = \text{property}_2$ then the results of the above mergers can be simplified further, noting that the unnamed classes that form the arrow targets are equivalent (they represent the same classes). For instance, when Class_2 subsumes Class_3, one instead obtains:



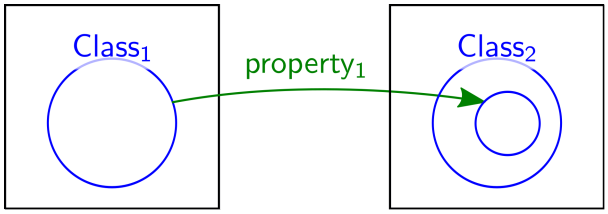
In general, if two solid arrows share a common source and have the same label (i.e. they are making an assertion about the same property) then their targets *must* represent the same classes: the targets are equivalent. This allows the diagram to be simplified by removing one of the arrows and both targets, placing in a new curve as the target of the remaining arrow. The new curve is drawn so that it is the 'intersection' of the two original targets.

Tip: Merging for Readability

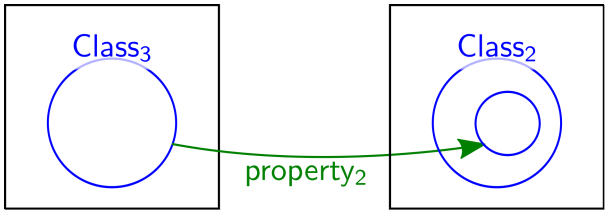
It can be helpful to merge many instances of the All Values From pattern when there is a common source class. This observation allows a lot of information about that class to be readily visualized. For readability, the ontology engineer must decide a suitable way of merging the bounding boxes (or not) to ensure overall readability.

All Values From: Merging with Common Target Classes

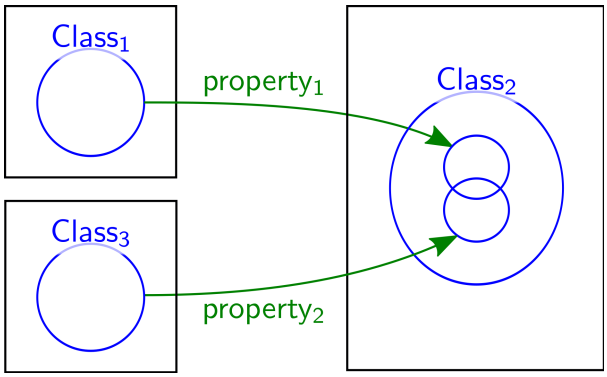
This subsection demonstrates how to merge two instances of the All Values From pattern where the two *target classes* are the same:



and

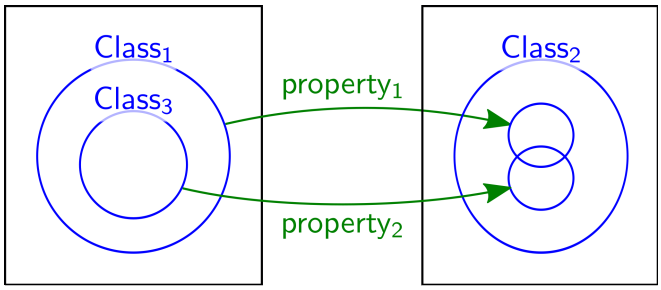


The first step in the merger, as with the case when the two source classes are the same, is trivial:



If no information is available as to whether Class_1 and Class_3 are in a subsumption relationship, are disjoint or equivalent, then the above diagram is the result of the merger[4]. However, if one of these three relationships is known to hold then the result is as follows:

1. Class_1 subsumes Class_3:



[4]As previously, one can proceed to merge the two diagrams containing the arrow sources but this *can* lead to a cluttered diagram. It is at the discretion of the ontology engineer to decide whether to continue merging aspects of these pattern instances.

If Class_3 subsumes Class_1 then the above diagram is altered in the obvious way.

2. Class_1 and Class_3 are disjoint: given as exercise.
3. Class_1 and Class_3 are equivalent: given as exercise.

Exercise: Merging All Values From Patterns

In each case below, draw a diagram representing the merger of two instances of the All Values From pattern when there are common target classes, with curve labelling as given above:

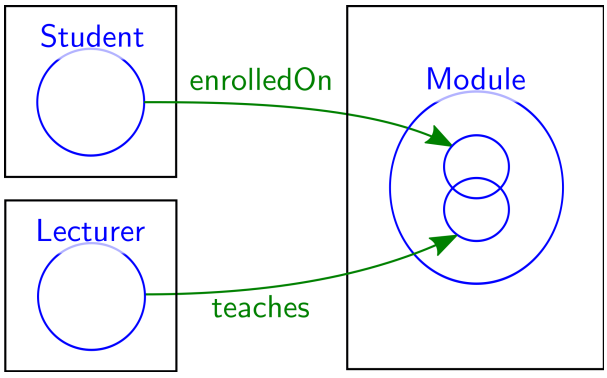
1. Class_1 and Class_3 are disjoint.
2. Class_1 and Class_3 are equivalent.

Example: Degree Courses Scenario

The following two All Values From property restrictions have the same target classes:

1. Individuals in the class Student have all values, under enrolledOn, from Module.
2. Individuals in the class Lecturer have all values, under teaches, from Module.

Merging the two instances of the All Values From pattern arising from these two property restrictions yields:



It is left to the reader to derive the original pattern instances. Moreover, the reader may also wish to perform further mergers on the above diagram, noting the disjointness of Student and Module and the disjointness of Lecturer and Module.

Exercise: Degree Courses Scenario

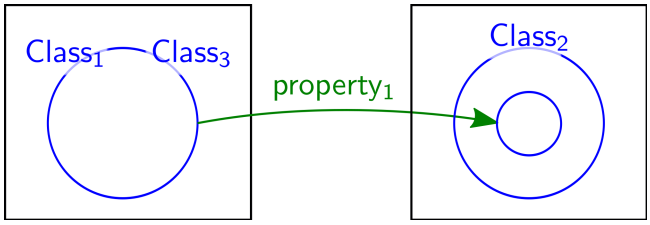
Draw an instance of the All Values From pattern for each of the property restrictions below:

1. Individuals in the class Exam have all values, under duration, from PositiveInteger.
2. Individuals in the class Test have all values, under duration, from PositiveInteger.

Merge the two resulting pattern instances.

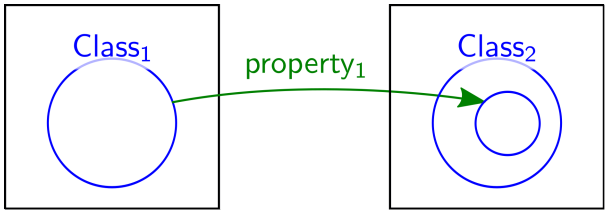
Tip: Merging when the Properties are the Same

If $\text{property}_1 = \text{property}_2$ then the results of the above mergers for the subsumption case and equivalence case can be simplified further. For instance, when Class_1 subsumes Class_3, the (unnamed class which is the) target of the arrow from Class_3 is subsumed by the target of the other arrow. When Class_1 and Class_3 are equivalent one instead obtains:

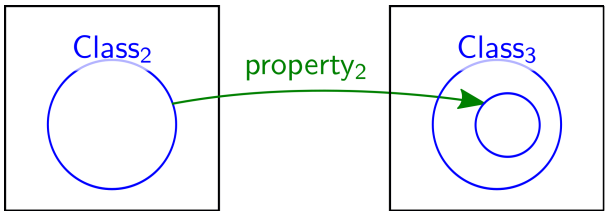


All Values From: Merging with Common Source Class and Target Class

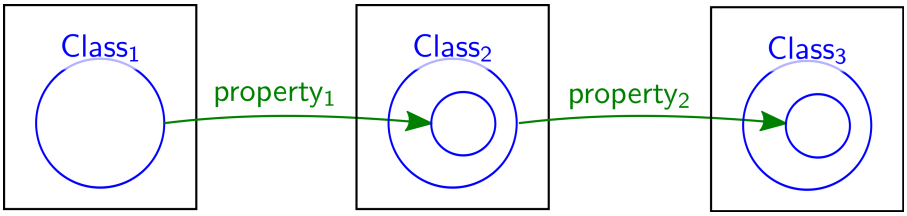
This subsection demonstrates how to merge two instances of the All Values From pattern where one *source class* is the same as one of the *target classes*:



and



The first step in the merger, as with the previous cases, is trivial:



If no information is available as to whether Class_1 and Class_3 are in a subsumption relationship, are disjoint or equivalent, then the above diagram is the result of the merger. As with the previous mergers, one can further merge the boundary boxes concerning Class_1 and Class_3 by using subsumption, disjointness, and equivalence.

Exercise: Merging All Values From Patterns

In each case below, draw a diagram representing the merger of two instances of the All Values From pattern when there is a common source class and target class, with curve labelling as given above:

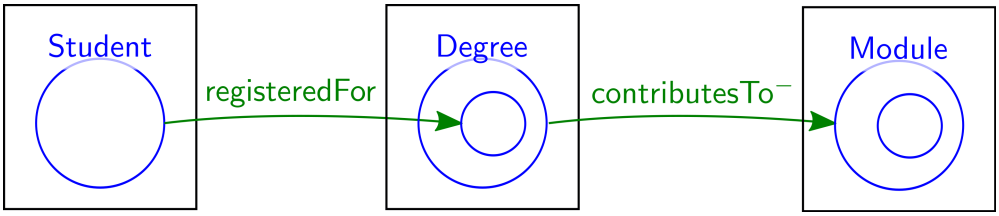
1. Class_1 subsumes Class_3.
2. Class_1 and Class_3 are disjoint.
3. Class_1 and Class_3 are equivalent.

Example: Degree Courses Scenario

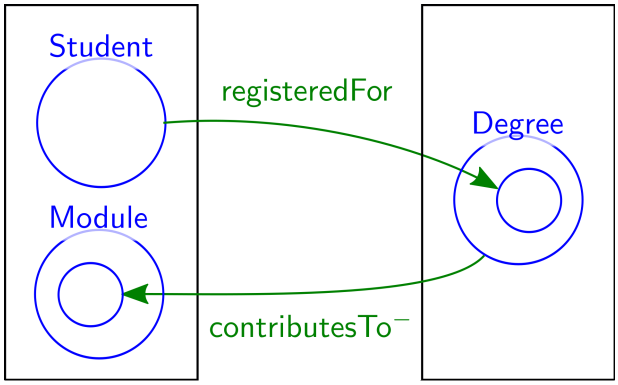
Consider the following two All Values From property restrictions:

1. Individuals in the class Student have all values, under registeredFor, from Degree.
2. Individuals in the class Degree have all values, under $\mathsf{contributesTo}^-$, from Module.

Merging the two instances of the All Values From pattern arising from these two property restrictions yields:



Further merging, of the lefthand and righthand boundary boxes yields:



It is left to the reader to derive the original pattern instances.

Exercise: Degree Courses Scenario

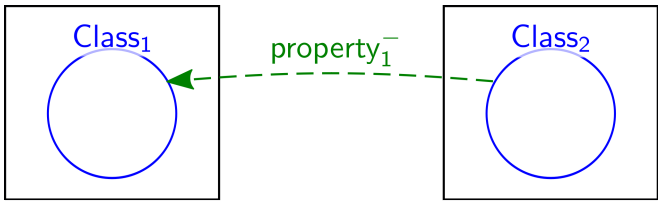
For each of the following two property restrictions, draw an appropriate instance of the All Values From pattern. Merge the resulting pattern instances.

1. Individuals in the class Lecturer have all values from, under teaches, Module.
2. Individuals in the class Module have all values from, under contributesTo, Degree.

4.2.2 *Merging Some Values From Patterns*

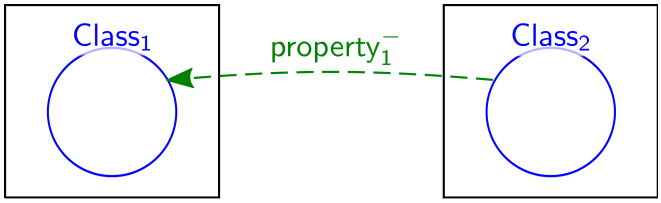
This section is subdivided to cover the following cases:

1. *Common Source Classes* Merging two instances of Some Values From patterns of the form



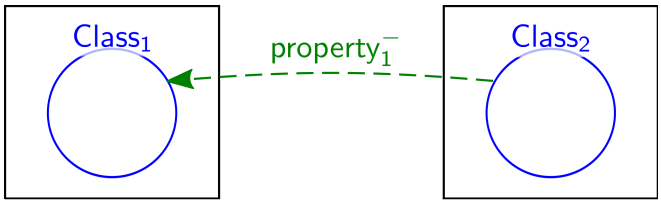
and

2. *Common Target Classes* Merging two instances of Some Values From patterns of the form

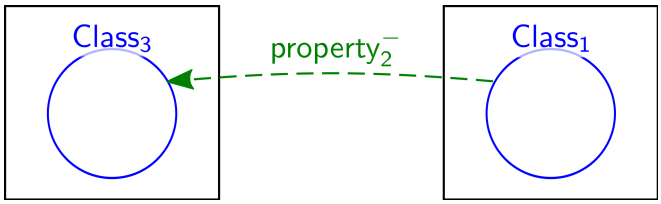


and

3. *Common Source Class and Target Class* Merging two instances of Some Values From patterns of the form



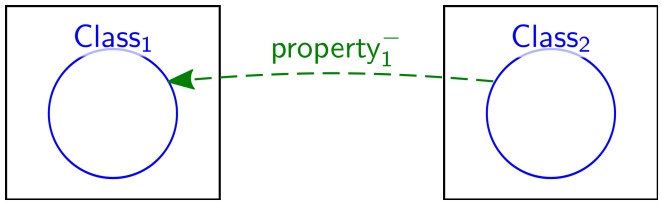
and



The remaining cases are left to the reader, using the principles of curve addition previously described.

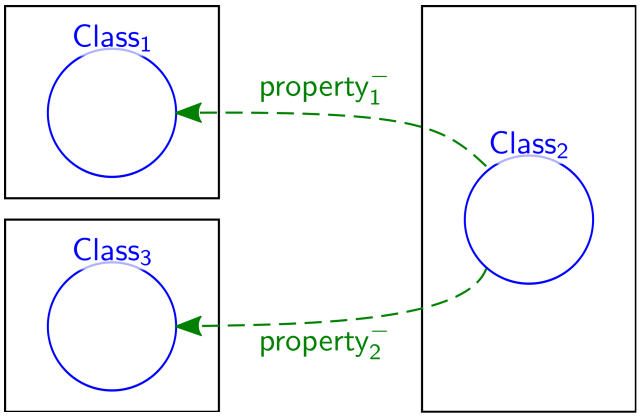
Some Values From: Merging with Common Source Classes

This subsection demonstrates how to merge two instances of the Some Values From pattern where the two *source classes* are the same:



and

The first step in the merger is trivial:



The diagram above is obtained by merging the syntax in the two lefthand boundary boxes[5]. If no information is available as to whether Class_2 and Class_3 are in a subsumption relationship, are disjoint or equivalent then the above diagram is the result of the merger.

[5]All mergers are, essentially, mergers of information in two or more boundary boxes, possibly using information derivable from other diagrams in the ontology.

Exercise: Merging Some Values From Patterns

In each case below, draw a diagram representing the merger of two instances of the Some Values From pattern when there is a common source classes, with curve labelling as given above:

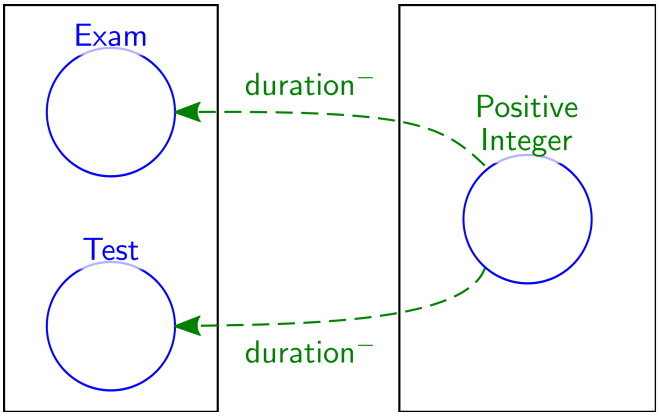
1. Class_1 subsumes Class_3.
2. Class_1 and Class_3 are disjoint.
3. Class_1 and Class_3 are equivalent.

Example: Degree Courses Scenario

The following are two examples of Some Values From property restrictions:

1. Individuals in the class Exam have at least one value, under duration, from PositiveInteger.
2. Individuals in the class Test have at least one value, under duration, from PositiveInteger.

Merging the two instances of the Some Values From pattern yields the following diagram:



It is left to the reader to further simplify this diagram, by noting that Exam, Test and PositiveInteger are pairwise disjoint.

Tip: Merging when the Properties are the Same

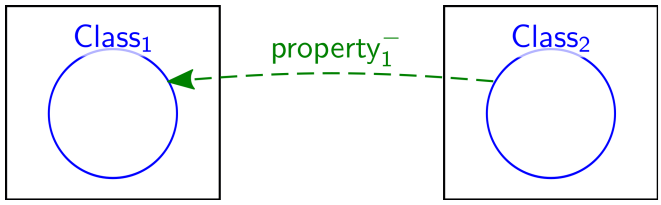
By contrast to the All Values From case, there is not a clear readability benefit from aiming to remove arrows when $\text{property}_1 = \text{property}_2$. This is because, whilst one fewer arrow is needed, an additional curve is likely to be introduced (representing the union of the target classes). In the All Values From case, the number of arrows and the number of curves required are reduced.

Tip: Merging for Readability

It can be helpful to merge many instances of the Some Values From pattern when there is a common source class. As with the All Values From patterns, this allows a lot of information about that class to be readily visualized.

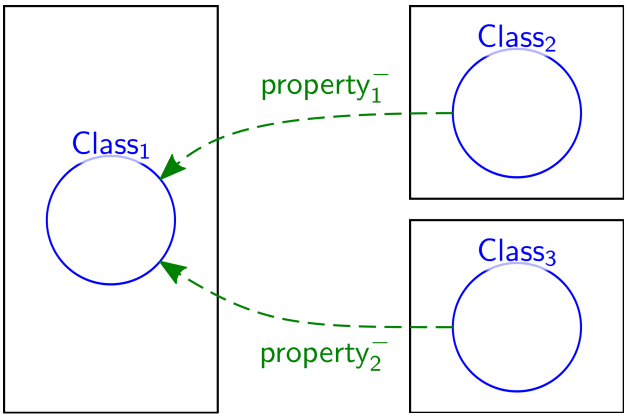
Some Values From: Merging with Common Target Classes

This subsection demonstrates how to merge two instances of the Some Values From pattern where the two *target classes* are the same:



and

The first step in the merger, as with the case when the two source classes are the same, is trivial:



If no information is available as to whether Class_1 and Class_3 are in a subsumption relationship, are disjoint or equivalent, then the above diagram is the result of the merger.

Exercise: Merging Some Values From Patterns

In each case below, draw a diagram representing the merger of two instances of the Some Values From pattern when there is a common target classes, with curve labelling as given above:

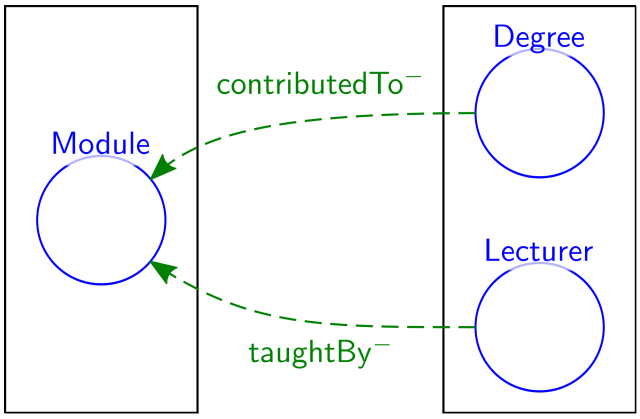
1. Class_2 subsumes Class_3.
2. Class_2 and Class_3 are disjoint.
3. Class_2 and Class_3 are equivalent.

Example: Degree Courses Scenario

The following are two examples of Some Values From property restrictions:

1. Individuals in the class Module have at least one value, under *contributesTo*, from Degree.
2. Individuals in the class Module have at least one value, under *taughtBy*, from Lecturer.

Merging the two instances of the Some Values From pattern yields the following diagram:



It is left to the reader to further simplify this diagram, by noting that Module, Degree and Lecturer are pairwise disjoint.

Exercise: Degree Courses Scenario

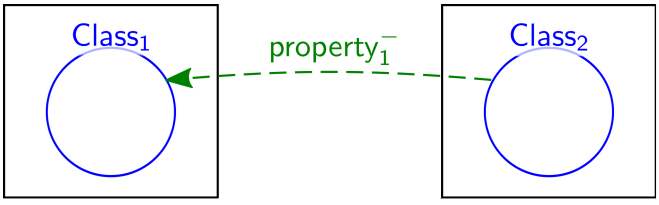
The following Some Values From constraints hold:

1. Individuals in the class Student have at least one value, under registeredFor, from Degree.
2. Individuals in the class Student have at least one value, under enrolledOn, from Module.

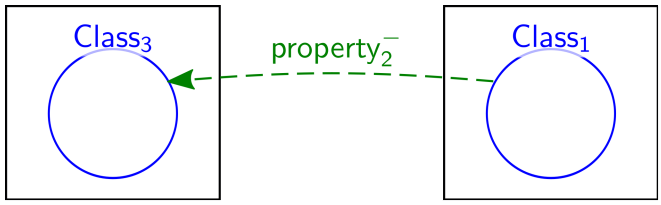
Draw an appropriate instance of the Some Values From pattern for each of the above property restrictions to produce a single diagram. Merge the resulting pair of pattern restrictions.

Some Values From: Merging with Common Source Class and Target Class

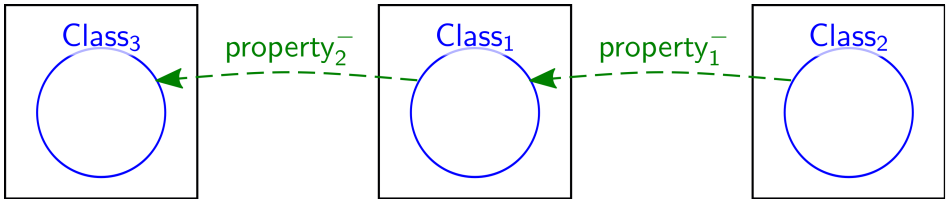
This subsection demonstrates how to merge two instances of the Some Values From pattern where one *source class* is the same as one of the *target classes*:



and



The first step in the merger, as with the previous cases, is trivial:



If no information is available as to whether Class_1 and Class_3 are in a subsumption relationship, are disjoint or equivalent, then the above diagram is the result of the merger. As with the previous mergers, one can further merge the boundary boxes concerning Class_1 and Class_3 by using subsumption, disjointness, and equivalence.

Exercise: Merging Some Values From Patterns

In each case below, draw a diagram representing the merger of two instances of the Some Values From pattern when there is a common source class and target class, with curve labelling as given above:

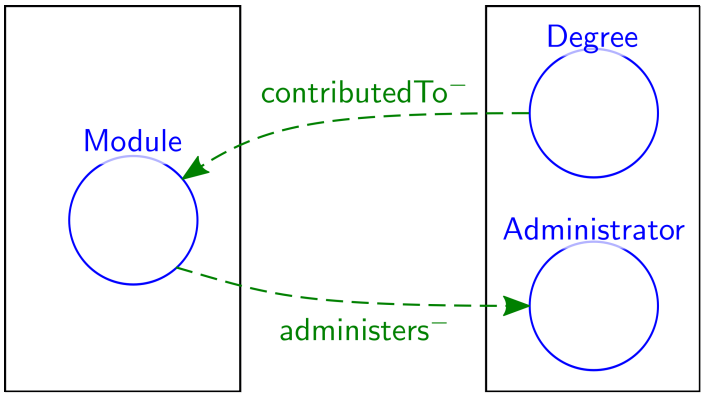
1. Class_2 subsumes Class_3.
2. Class_2 and Class_3 are disjoint.
3. Class_2 and Class_3 are equivalent.

Example: Degree Courses Scenario

The following Some Values From constraints hold:

1. Individuals in the class Module have at least one value, under contributesTo, from Degree.
2. Individuals in the class Administrator have at least one value, under administers, from Module.

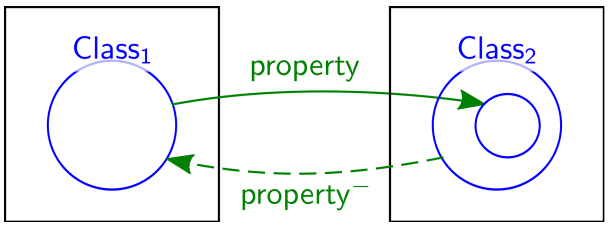
Merging the two instances of the Some Values From pattern yields the following diagram:



It is left to the reader to further simplify this diagram, by noting that Module, Degree and Administrator are pairwise disjoint.

4.2.3 *Merging All Values From and Some Values From Patterns*

The same principles can be applied to merge instances of All Values From and Some Values From patterns. A commonly occurring pair of property restrictions is to assert that individuals from Class_1 have some values from and all values from Class_2 under property. Diagrammatically, the merger of the two pattern instances arising from this pair of property restrictions is:



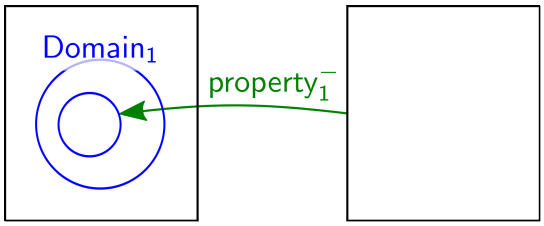
Other ways of merging All Values From and Some Values from pattern instances are left to the reader.

4.3 Merging Patterns for Domain and Range

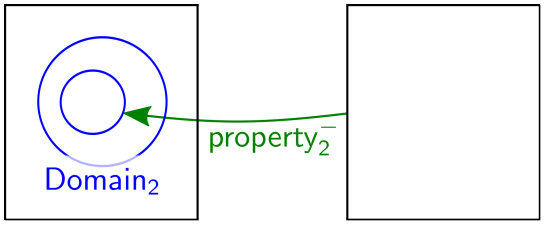
Merging patterns that specify domains and ranges allows the visualization of many such constraints in a single diagram. Unlike the All Values From and Some Values From patterns, mergers are particularly beneficial, perhaps, when the patterns do not have classes in common (at least, when merging two domain pattern instances or two range pattern instances). Again, the mergers we present have two key stages: an initial merger, which is then followed by some further simplifications if relationships are known to exist between the represented classes.

4.3.1 *Merging Domain of a Property Patterns*

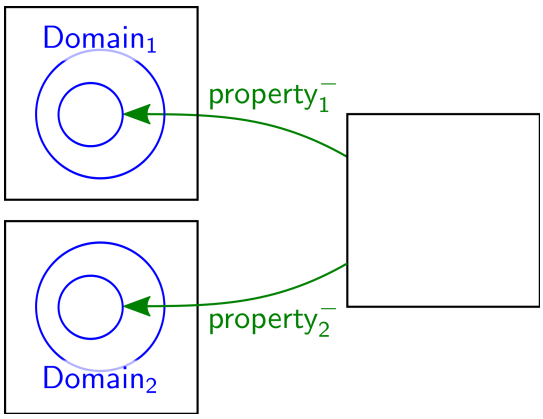
Suppose two instances of the Domain of a Property pattern assert, respectively, that the domain of $\mathsf{property}_1$ is Domain_1 and that the domain of $\mathsf{property}_2$ is Domain_2:



and:

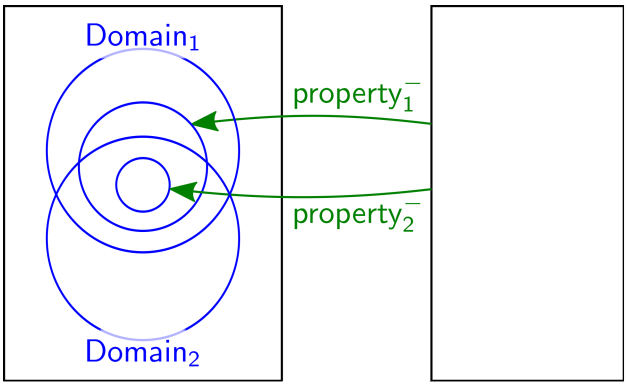


Then the two arrows involved have essentially the same source: the boundary rectangle, which represents the class Thing. This observation means that the merging process is simpler than for diagrams with arrows sourced on classes which are subsumed by Thing. The result of the initial merging is simply:



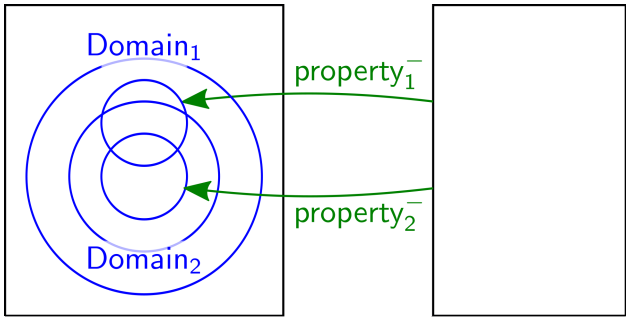
Just as with previous cases, the diagrams involving Domain_1 and Domain_2 can be further merged, if a subsumption, disjointness, or equivalence relationship is known to hold. Otherwise, the diagram above is the result of the merger, unless the ontology engineer wishes to express the domain information for $\mathsf{property}_1$ and $\mathsf{property}_2$ in a single boundary box, bearing in mind that the resulting diagram may appear cluttered. This could be advantageous if the ontology engineer wishes to *additionally* assert that the two properties have pre-images[6] in a subsumption relationships, even if their domains are not:

[6]Recall, the pre-image of a property is the set of all individuals that are related to something under that property.



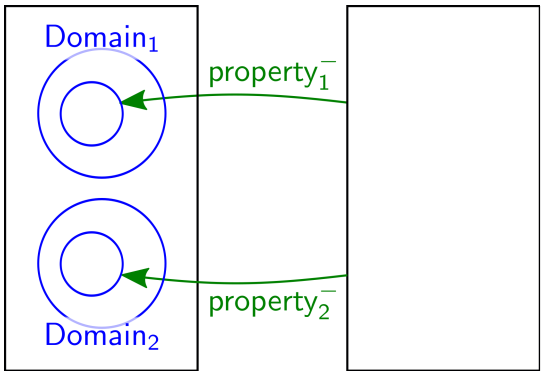
Likewise, merging the two boundary boxes containing the domain information would also make it readily possible to express that a disjointness or equivalence relationship holds between the property pre-images, even if no information is known about the relationship between the domains.
If a subsumption, disjointness or equivalence relationship is known to hold between the domains then further merging can make these relationships explicit.

1. Domain_1 subsumes Domain_2:



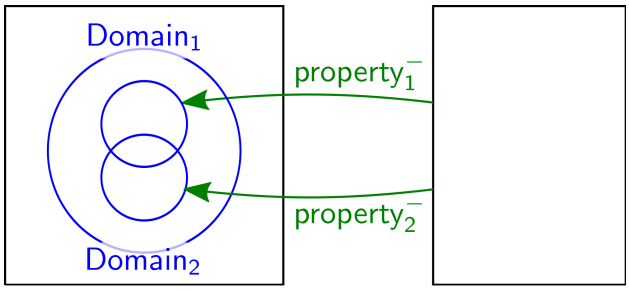
   If Domain_2 subsumes Domain_1 then the above diagram is altered in the obvious way.

2. Domain_1 and Domain_2 are disjoint:



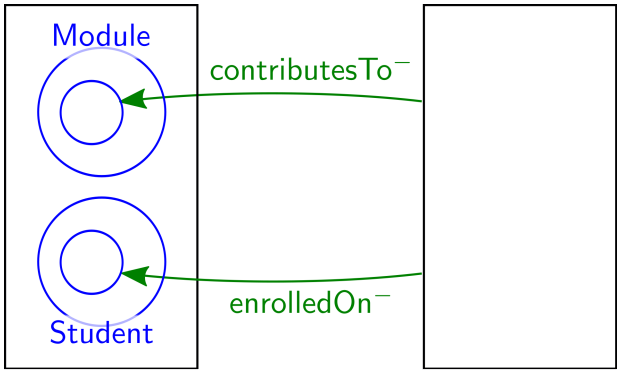
   This merger allows the previously derivable information that the pre-images of $\mathsf{property}_1$ and $\mathsf{property}_2$ are disjoint to be explicitly represented diagrammatically.

3. Domain_1 and Domain_2 are equivalent:



Example: Degree Courses Scenario

The property contributesTo has domain Module whereas enrolledOn has domain Student:



The above diagram makes it visually explicit that these two properties have disjoint domains. It is left to the reader to determine the instances of the Domain of a Property patterns that gave rise to this merged diagram.

Exercise: Degree Courses Scenario

In each of the following cases, draw an appropriate instance of the Domain of a Property pattern:

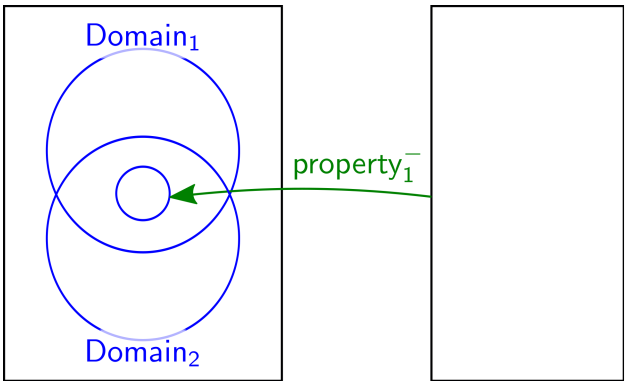
1. The domain of creditValue is Module.
2. The domain of taughtBy is Module.
3. The domain of teaches is Lecturer.
4. The domain of teaches is Academic.

5. The domain of leads is CourseLeader.

For each pair of the resulting pattern instances, produce the merged diagram. The interested reader may also wish to consider merging all, or more than two, of these pattern instances into a single diagram.

Tip: Merging when the Properties are the Same

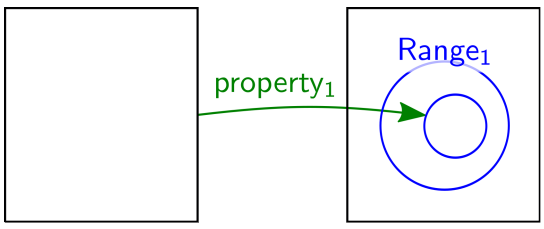
If the two properties are the same then the above mergers can be further simplified, noting that the two unnamed classes that form the arrow targets are equivalent (they represent the same classes). For instance, when no information is known about the relationship between Domain_1 and Domain_2, the merged diagram would be:



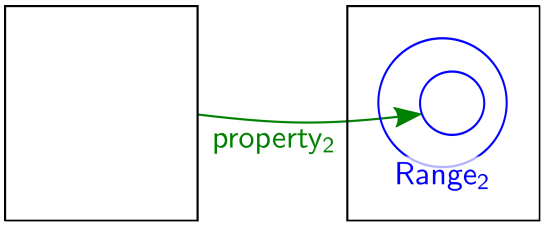
Such simplifications were previously demonstrated, on page 75, for merging instances of All Values From pattern instances, where the two arrow source classes were the same.

4.3.2 *Merging Range of a Property Patterns*

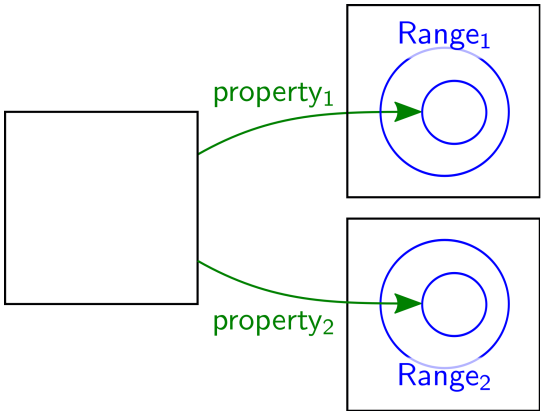
Now suppose two instances of the Range of a Property pattern assert, respectively, that the range of property_1 is Range_1 and that the range of property_2 is Range_2:



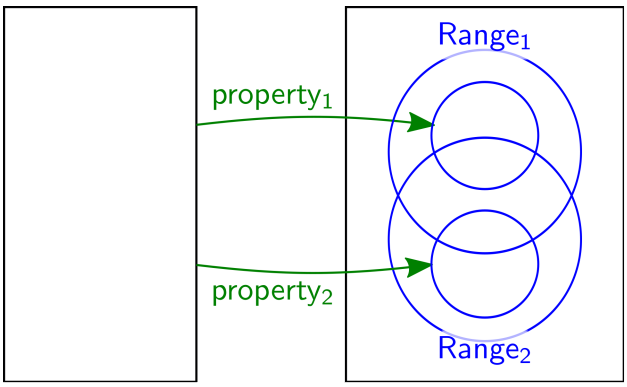
and



Then, as with the Domain of a Property pattern, the two arrows involved both have essentially the same source which represents the class Thing. The result of the initial merging is simply:



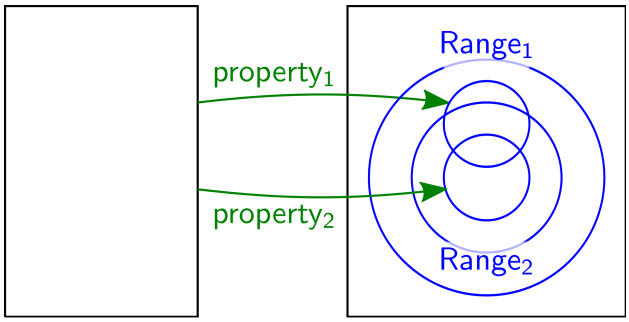
Again, the diagrams involving Range_1 and Range_2 can be further merged, if a subsumption, disjointness, or equivalence relationship is known to hold. However, further merging could still be beneficial if the ontology engineer wishes to additionally assert that the two properties have images[7] in, say, a disjointness relationship, even if their ranges are not:



Likewise, merging the two boundary boxes containing the domain information would also make it readily possible to express that a subsumption or equivalence relationship holds between the property images, even if no information is known about the relationship between the domains. If a subsumption, disjointness or equivalence relationship is known to hold between the ranges then, as with domains, further merging can make these relationships explicit.

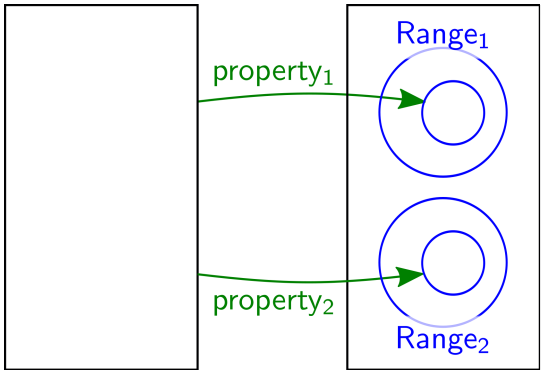
1. Range_1 subsumes Range_2:

[7] Recall, the image of a property is the set of all individuals that are related to, by something, under that property.



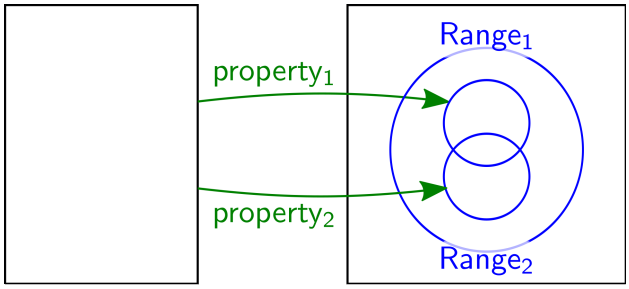
If Domain_2 subsumes Domain_1 then the above diagram is altered in the obvious way.

2. Range_1 and Range_2 are disjoint:



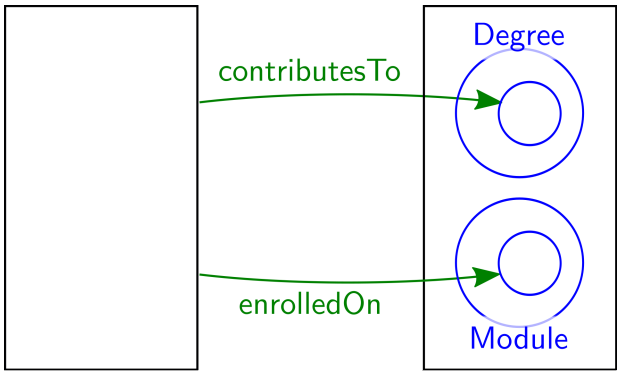
Similar to the Domain of a Property case, this merger allows the previously derivable information that the images of property_1 and property_2 are disjoint to be explicitly represented diagrammatically.

3. Range_1 and Range_2 are equivalent:



Example: Degree Courses Scenario

The property contributesTo has range Degree whereas enrolledOn has range Module:



It is now visually explicit that these two properties have disjoint domains. It is left to the reader to determine the instances of the Range of a Property patterns that gave rise to this merged diagram.

Exercise: Degree Courses Scenario

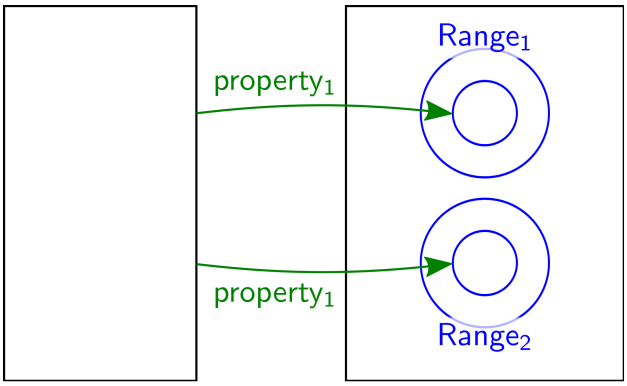
In each of the following cases, draw an appropriate instance of the Range of a Property pattern:

1. The range of creditValue is PositiveInteger.
2. The domain of taughtBy is Lecturer.
3. The domain of teaches is Degree.
4. The domain of teaches is Course.
5. The domain of leads is Degree.

For each pair of the resulting pattern instances, produce the merged diagram. The interested reader may also wish to consider further merging all or more than two of these pattern instances into a single diagram.

Tip: Merging when the Properties are the Same

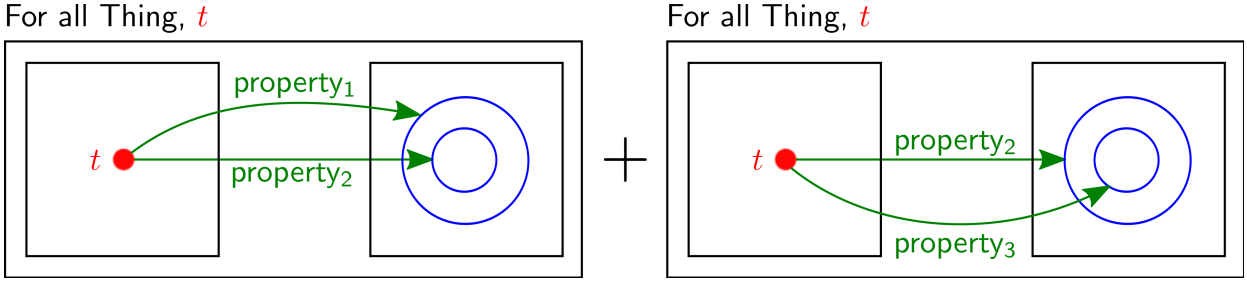
If the two properties are the same, i.e. $\text{property}_1 = \text{property}_2$, then the above mergers can be further simplified, because the two unnamed classes that form the arrow targets are equivalent (they represent the same classes). For instance, when Range_1 and Range_2 are disjoint, the merged diagram would be:



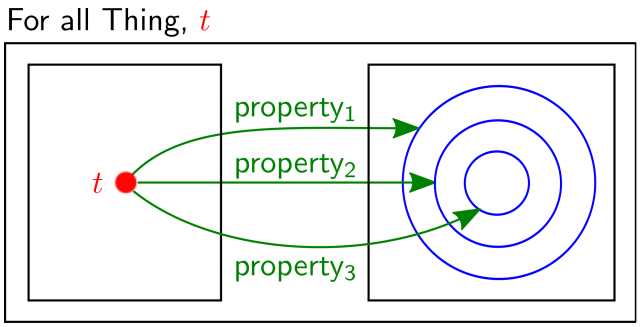
Here, the ontology engineer can readily deduce that the image of property_1 is empty, as it is subsumed by two disjoint classes. Thus, this may alert the ontology engineer to a potential problem with the model being produced and, therefore, aid with debugging the model.

4.4 Merging Patterns for Properties

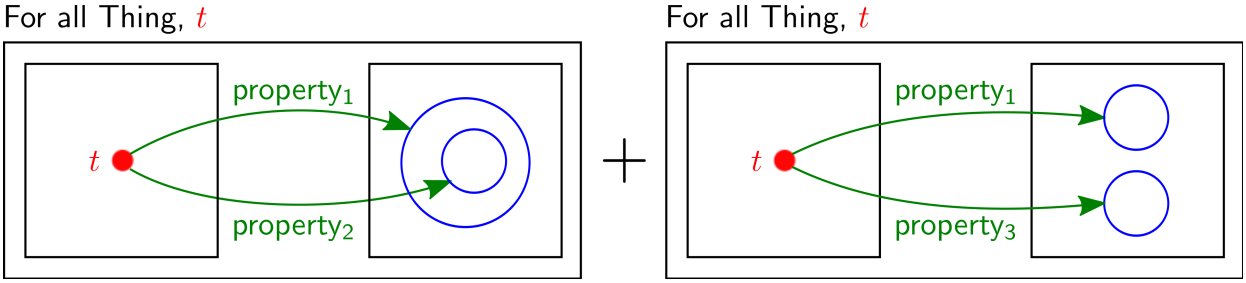
Three types of patterns for properties were presented in chapter 3, for property subsumption, disjointness and equivalence. First, three simple patterns were presented, namely: Property Subsumption, Property Disjointness and Property Equivalence. The General Property Subsumption pattern, which asserts a property hierarchy, is essentially a merger of instances of Property Subsumption patterns, where the hierarchy information is given:



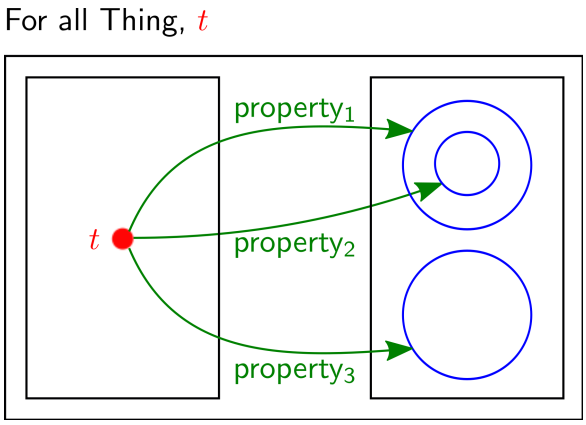
gives rise to:



Similarly, the General Disjointness pattern and the General Equivalence pattern can be viewed as mergers of the respective 'simple' versions of these patterns. It is also possible to merger property patterns of different types. For example, given:



by merging, one obtains:



Here, the way in which the unnamed classes are merged is akin to the merger techniques given for classes: one simply uses the known subsumption and disjointness relationship to derive the new relative positioning of the curves in the merged diagram. This observation applies to merging other combinations of property patterns. It is left to the reader to determine how to merge such cases.

4.5 Summary

This chapter has introduced *techniques for merging patterns*. There can be benefits to merging patterns, not least because a single, merged, diagram can be used to convey the same informa-

tional content as several simpler diagrams. The resulting single diagrams can readily convey subsumption, disjointness, and equivalence information that would have previously needed to be inferred. This is an aid to understanding the model built by the ontology engineers and can reveal unintended consequences of modelling decisions. Therefore, the act of merging patterns can allow models to be debugged and, hence, improved. In addition, merging can provide all stakeholders with a more rounded understanding of the ontology.

Chapter 5

Case Study: The Semantic Sensor Networks Ontology

This chapter is designed to allow the reader to draw on all of the material presented in chapters 2 to 4, tying together the key concepts that have been introduced. This will be done via a case study, centered on the Semantic Sensor Networks (SSN) ontology, developed by an incubator group of the W3C, available from `http://www.w3.org/2005/Incubator/ssn/ssnx/ssn`. This chapter does not attempt to present the entire ontology here – it is far too large – but selects just some of the classes and properties. Moreover, in some instances, the actual axioms have been informationally weakened, for simplicity. Readers interested in the full ontology are referred to the URL given above, where there are plentiful examples of axioms that can be converted to diagrammatic form using the methods described in this book.
At the end of this chapter the reader should:

1. Be able to *identify the axioms* for a non-trivial example of an ontology, described textually.
2. Be able to *define those axioms* using concept diagrams, identifying the use of suitable patterns where appropriate.
3. Be able to *determine which patterns can be merged*, in order to reduce the number of diagrams required to define the ontology, and to highlight information that would otherwise need to be derived.

This SSN ontology is now introduced, in a textual form, covering the core vocabulary. After this presentation, the reader will be presented with a series of exercises. The first set of exercises will ask the reader to identify the axioms relating to relationships between classes, arising from the information contained in the textual description. The reader is advised to read the exercises in parallel with the scenario described, rather than reading the scenario in full first. Here it is noted that the selected fragment of the ontology *does not* include any named individuals.

Scenario: SSN Ontology

The SSN ontology provides an ontological model centered around sensors. In this ontology, there are many classes, including Entity, Abstract, Quality, Event and Object. The latter four of these classes only contain individuals that are entities and do not have any individuals in common with each other.The class object has two sub-classes, called PhysicalObject and SocialObject. Nothing can be both a PhysicalObject and SocialObject. The information just provided gives some of the relationships between various classes in the SSN ontology.

Perhaps the most important class in the SSN ontology is Sensor. Sensors are all physical objects and they have certain functions. In particular, they are able to detect stimuli. In more formal terms, there is a property called detects, that relates sensors to individuals in the class Stimulus and no other classes. Any individual in the class Stimulus is also called a SensorInput and vice versa. Informally, the input to a sensor is what it detects. As well as detecting sensor inputs, sensors are able to make observations. These observations are all individuals in a class called Property. Note here that Property is the name of a class and, thus, part of the vocabulary. It should not be confused with properties, which represent binary relations between individuals. The individuals from the Property class that sensors observe are identified by the property observes. Sensors also have only MeasurementCapabilities under the the property hasMeasurementCapability. Sensors also implement some Sensing and, possibly, implement other things too. The information just given describes the core of functions of sensors, as indicated in the SSN ontology.

Individuals in the class Sensor can be further classified as being sensing devices. Individuals in the class SensingDevice have to be sensors and they also have to be in the class Device. Devices are all in the class System and systems are all physical objects. Moreover, the class Quality includes all individuals in the class Property. Each MeasurementCapability is a Property (i.e. an individual in the class Property. It is also given that each stimulus is an event. This gives us further hierarchical information about some classes in the SSN ontology.

Like Sensor, the class Observation is important. Observations are all in the class called Situation. Moreover, situations are all in the class SocialObject. It is also known that observations are made when something stimulates them. More precisely, there is a property called includesEvent which links observations to individuals in the class Stimulus. For each observation, there is at least one stimulus. Observations are observedBy a sensor, and only sensors. There is also a property (in the binary relation sense) called observedProperty which relates observations to individuals in the class Property. Observations use a sensing method to relate them to individuals that are in the sensing class. More precisely, the property sensingMethodUsed, 'returns' some sensing individual, given any observation. In addition, sensingMethodUsed, can only return individuals from the class Sensing. Each observation has an ObservationResult which can only be in the class SensorOutput. Thus, observations are involved with a number of properties in the SSN ontology.

Some more hierarchical information concerning classes is now given. Sensor outputs are all in the class informationObject which, in turn, are social objects. No information object can be a situation. The class Sensing contains only individuals from the class Process which, in turn, contains only

individuals in the class Method. Moreover, methods are also in the class called Description. All descriptions are social objects. Descriptions cannot be situations or information objects.

In terms of scale, a recent survey by Paul Warren affiliated with the Open University, indicates that – amongst respondents – over 50% of ontologies had fewer than 30 classes, so the fragment described above can be considered as a realistic size in its own right, at least in terms of the number of classes. That survey can be accessed from:

`http://kmi.open.ac.uk/people/member/paul-warren`.

Again, considering scale, Paul Warren's survey indicates that about $\frac{2}{3}$ of ontologies had fewer than 20 properties (about $\frac{1}{3}$ had fewer than 10 properties). Again, this demonstrates that the fragment of the SSN ontology considered in this chapter is not dissimilar in size to ontologies arising in every day situations.

Exercise: SSN Ontology: Informal Axioms for Class Relationships

The textual description of the SSN ontology captures a number of axioms about relationships between classes. For example:

1. Entity subsumes Abstract.
2. Abstract and Quality are disjoint.

Identify the remaining axioms for class subsumption, disjointness and equivalence.

Exercise: SSN Ontology: Formal Axioms for Class Relationships

For each of the axioms identified in the previous exercises, draw an appropriate diagrammatic representation of it, using patterns, or merged patterns, where appropriate.

Exercise: SSN Ontology: Informal Axioms for Property Restrictions

The textual description of the SSN ontology captures a number of axioms that give rise to property restrictions. For example:

1. Individuals in the class Sensor have all values, under detects, from Stimulus.
2. Individuals in the class Sensor have at least one value, under implements, from Sensing.

Identify the remaining axioms for property restrictions.

Exercise: SSN Ontology: Formal Axioms for Property Restrictions

For each of the axioms identified in the previous exercises, draw an appropriate diagrammatic representation of it, using patterns, or merged patterns, where appropriate.

A partial solution to the exercises so far is given on page 105. In addition to the information already given for the SSN ontology, there are additional restrictions placed on the properties.

Exercise: SSN Ontology: Domain and Range Axioms for Properties

Translate the following constraints on properties into diagrammatic axioms.

1. includesObject has domain Situation and range Object.
2. isDescribedBy has domain Entity and range Description.
3. isSettingFor has domain Situation and range Entity.
4. hasQuality has domain Entity and range Quality.

Consider ways in which the resulting pattern instances can be merged.

Tip: Considerations when Merging

Merging can be useful if one wishes to see the properties that have domain which is a particular class. For example, merging the diagrams arising from the domain information in questions 2 and 4 immediately above would convey the information that isDescribedBy and hasQuality both have domain Entity. Similarly, combining patterns that involve a given property can convey all of that information about that property in a single diagram. In addition, merging diagrams that give information about properties that are, say, intended to be in a subsumption relationship could highlight unintended model properties, or places where individual constraints could be made stronger without changing the overall ontology. Such situations arise, for instance, where $\mathsf{property}_1$ subsumes $\mathsf{property}_2$ which implies the domain of $\mathsf{property}_1$ subsumes the domain of $\mathsf{property}_2$.

Exercise: SSN Ontology: Subsumption, Disjointness and Equivalence Axioms for Properties

Translate the following constraints on properties into diagrammatic axioms.

1. isDescribedBy subsumes implements.
2. isSettingFor subsumes observedProperty.
3. isSettingFor subsumes observationResult.
4. isSettingFor subsumes includesObject.
5. hasProperty subsumes hasMeasurementCapability.
6. hasQuality subsumes hasProperty.
7. includesObject and hasQuality are disjoint.
8. observedBy and $\mathsf{madeObservation}^-$ are equivalent.

9. implementedBy and implements$^-$ are equivalent.

Select some of the pattern instances arising from the above questions for merging. Consider the benefits of merging two or more diagrams when determining which ones to merge.

Exercise: SSN Ontology: Making Deductions

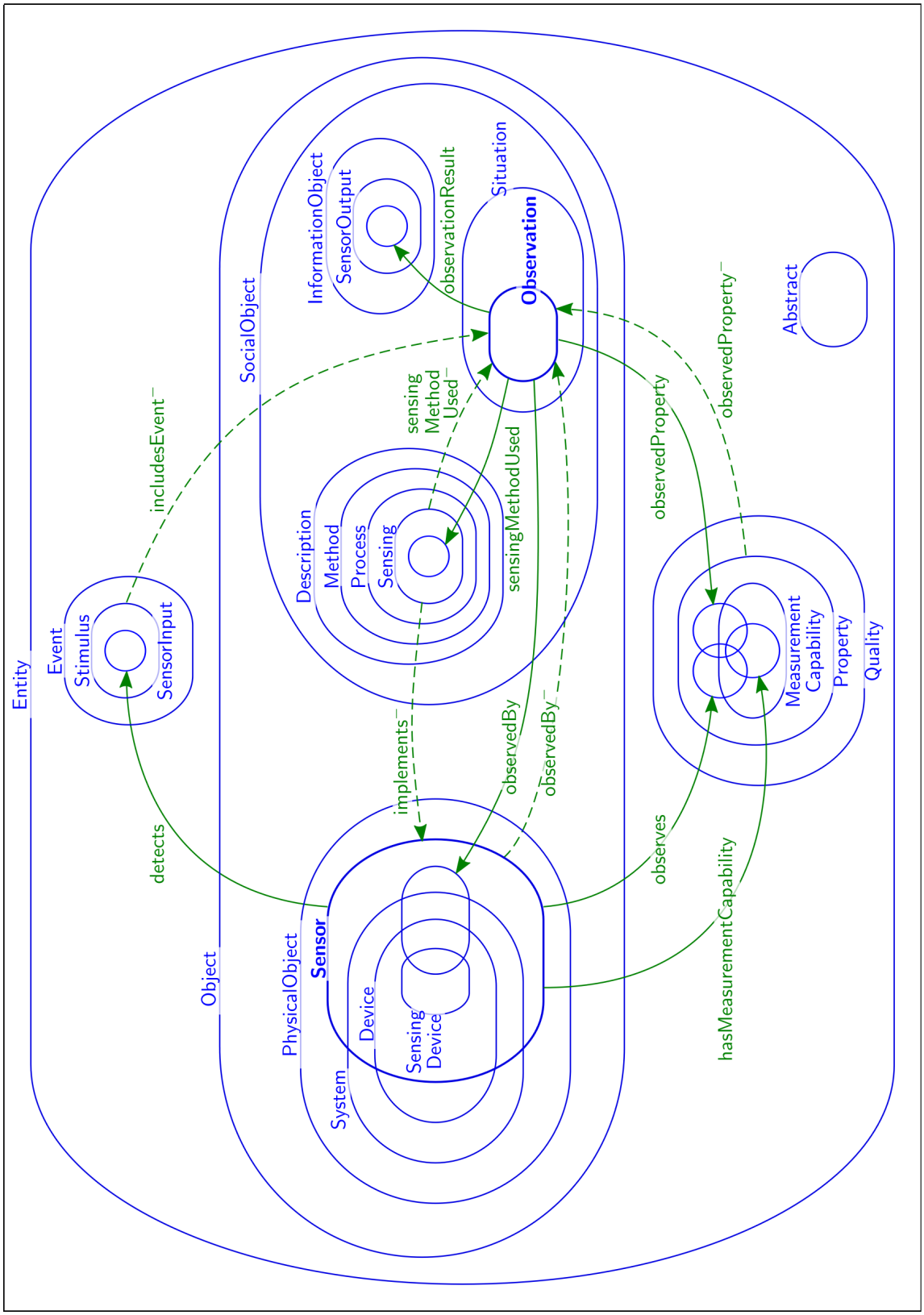
What, if anything, can be deduced about the domains and/or ranges given the axioms above:

1. implements [Hint: consider the property isDescribedBy]
2. observedProperty
3. hasMeasurementCapability

5.1 Summary

This chapter has allowed the reader to convert a textually specified ontology into diagrammatic form. This involved *identifying the axioms* for a non-trivial example of an ontology. The next step was to *define those axioms* using concept diagrams, identifying the use of suitable patterns where appropriate. Lastly, the exercises included *determining which patterns to merge*, in order to reduce the number of diagrams required, and to highlight information that would otherwise need to be derived.

Partial Solution to SSN Exercises

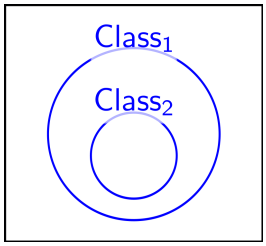


Appendix A

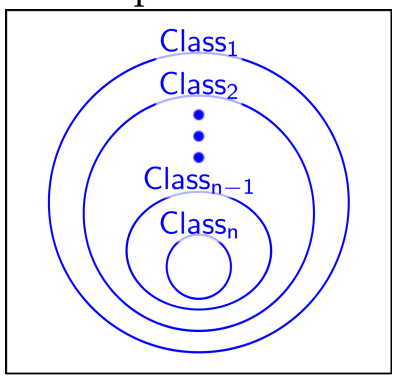
Patterns

The diagrammatic patterns introduced in chapter 3 are recreated for convenience.

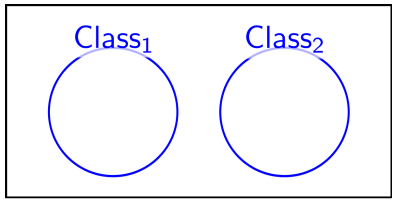
Pattern 1: Class Subsumption



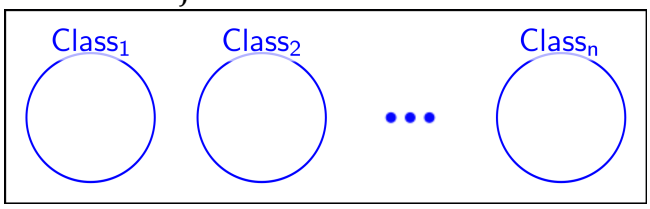
Pattern 2: General Class Subsumption



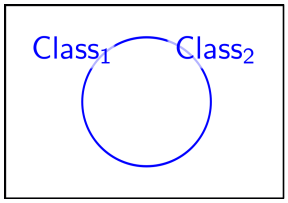
Pattern 3: Class Disjointness



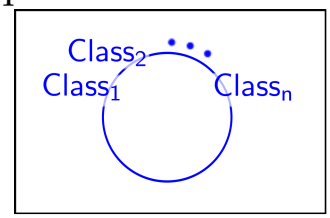
Pattern 4: General Class Disjointness



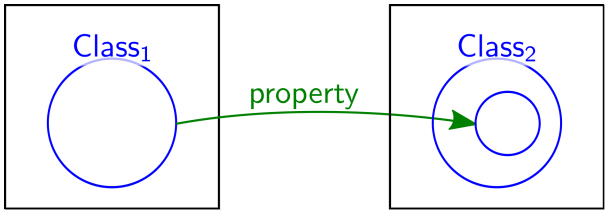
Pattern 5: Class Equivalence



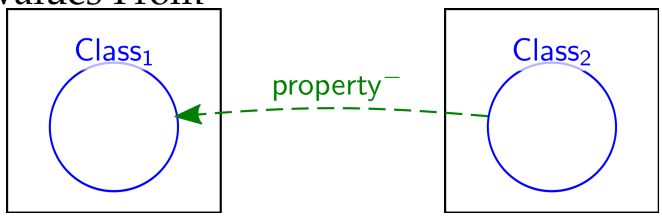
Pattern 6: General Class Equivalence



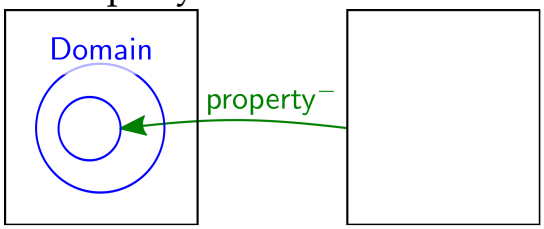
Pattern 7: All Values From



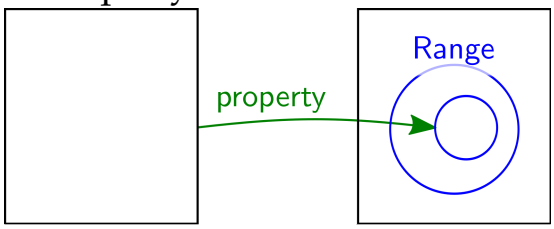
Pattern 8: Some Values From



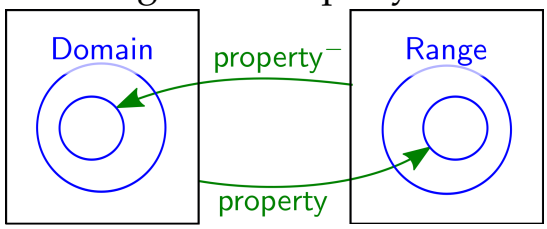
Pattern 9: Domain of a Property



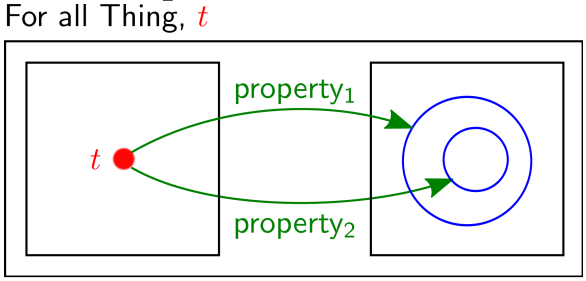
Pattern 10: Range of a Property



Pattern 11: Domain and Range of a Property

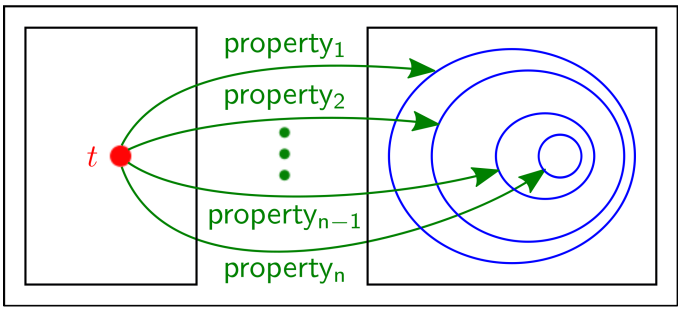


Pattern 12: Property Subsumption



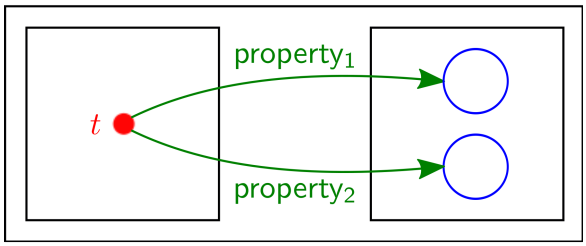
Pattern 13: General Property Subsumption

For all Thing, t



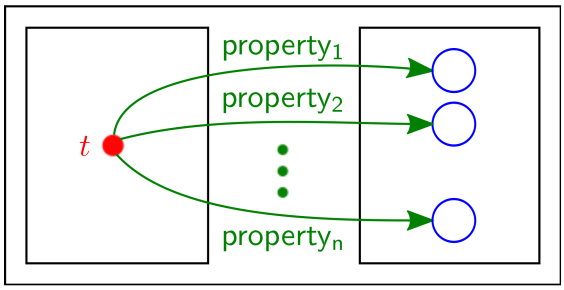
Pattern 14: Property Disjointness

For all Thing, t



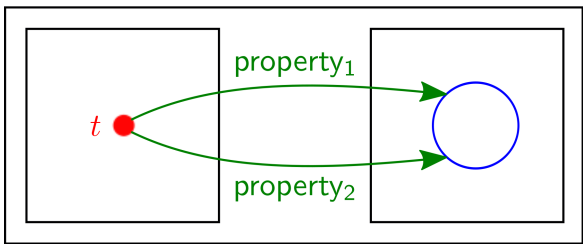
Pattern 15: General Property Disjointness

For all Thing, t



Pattern 16: Property Equivalence

For all Thing, t



Pattern 17: General Property Equivalence