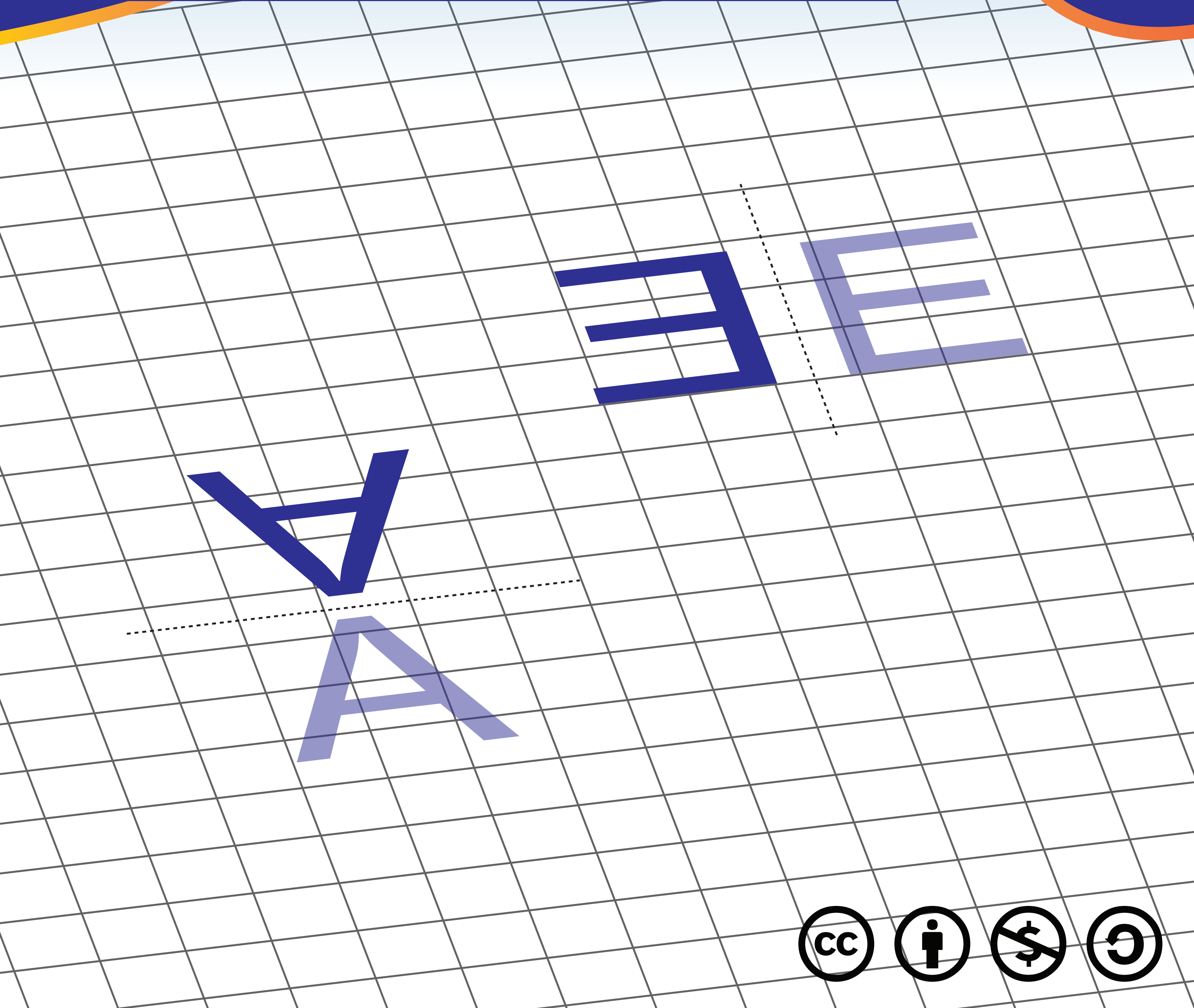
Learning Logic Backwards

Peter Susanszky



Contents

How to read this book

This textbook is intended as an introduction to logic. Granted, there are probably hundreds of introductory logic textbooks already available 'on the market.' However, this book is different from most introductory logic textbooks. It is also technically not part of 'the market.' But more on that later.

The fundamental idea that inspired the writing of this book has to do with modern logic as a discipline. Around the turn of the 20th century, logic became a highly formalized, mathematized, symbolic discipline. This is why we sometimes call it formal logic, mathematical logic, or symbolic logic! Students, especially those coming from non-mathematical disciplines like philosophy, aren't always ready to face this fact. Indeed, I often hear students say: "Professor, I know you said this course is going to be like math, but I didn't think it's going to be like actual math!"

There are essentially two ways introductory logic textbooks try to tackle this problem. Some of the textbooks attempt to focus on the 'logic' part while mostly disregarding the 'formal' part of formal logic. This is great for honing your logical and general thinking skills, and it certainly makes some ideas easier to digest. However, if you want to then move on to more advanced topics, or understand works that refer to, or make use of, logical concepts, you cannot circumvent facing the formalism, and the ideas behind it. On the other hand, there are many textbooks that do not shy away from the formal aspects of logic. However, they are also often incomprehensible to the newcomer, for they already assume familiarity with the style in which modern mathematics is written, and basic concepts that they assume have already been covered elsewhere. I have seen syllabi where this hidden prerequisite was referred to as 'basic mathmematical sophistication' – and by the end of the semester, it turned out most students were pretty unsophisticated in this regard!

This book aims to chart a third path. First, unlike textbooks in the first group, it presents various concepts from modern logic in a way similar to what you would find in more advanced textbooks, and even serious, published works on the topic. On the other hand, unlike textbooks in the second group, it does not assume that you already have the skills to read and importantly, write, in the style of modern mathematics. Accordingly, time will be spent explaining in detail how formal notation functions, what basic ideas it allows us to represent in a terse but rigorous form, and how you should go about emulating it in your own writing. All in all, though the book covers the central concepts of basic formal logic, its main aim is to provide the reader with the resources necessary to think, read, and write like a modern logician.

What this book is not

I talked briefly about what this book is – now for what this book is not. Most importantly, this book is not intended as a reference material for any topic in logic. Though many of the central concepts of basic logic are introduced, no proof of any of the major theorems of propositional logic or first-order logic (like soundness, completeness, compactness, etc.) is sketched. This is partly because almost every interesting proof in logic is essentially inductive, and inductive proofs are not covered in this book (and knowledge of them will certainly not be assumed).

Though this is not surprising for an introductory textbook, the work is also fairly limited in scope. If you are new to formal logic, you might think that there is one logical language, one logic for that language, one way to prove things in that logic, and so on. In reality, there are a lot of logical languages, to which many logics correspond, for which several proof systems may be formulated. Choices abound. However, as usual, only the classics are covered here, by which I mean classical logic up to first-order (if that means anything to you).

Read carefully

One thing you should keep in mind when reading any work of formal logic is that small things can make a big difference. For example, in the following, you may come across different typographical variants of the letter p, like $\mathbf{p}$, P, $\mathbf{P}$, $\mathcal{P}$ and even $\mathfrak{P}$. Depending on how and in what context they are introduced, these variants will have entirely different meanings attached to them. Accordingly, it is important that you notice these differences, and follow the relevant conventions when you are doing the exercises.

Coupled with this is the additional fact that the appearance of a single letter in front of another may radically change its meaning from the initial one. Here is a toy example. Suppose the letter 'C' stands for 'Curtis', and the letter '$\mathbf{f}$' stands for 'father of'. If we put these two together, we get that '$\mathbf{f}C$' stands for 'the father of Curtis'. Clearly, Curtis is not his own father, so 'C' and '$\mathbf{f}C$' mean different people. Now sometimes, these formulas might pop up as part of whole sentences. In this case, you have to be extra careful what you are writing. One recurring mistake I see often is to write things like 'the father of $\mathbf{f}C$'. Note that this does *not* mean the father of Curtis, it means the father of the father of Curtis, his grandpa (on his father's side). In these cases, you should either write 'the father of C', of simply, '$\mathbf{f}C$', either of which means, by itself, 'the father of Curtis'. And returning to our previous point, you should not write 'FC' either, since '$\mathbf{f}$' is lowercase, and in bold.

In my own experience, I might properly digest a 25 page philosophy paper in a few hours, while I have seen 5 page logic papers that took several days to get down (if at all). Given that reading philosophy is itself not a walk in the park (you should at least sit down to do it!), you may get a general sense of the type of attention and concentration required for reading logic, and how different it is from reading something like a novel.

Write!

As mentioned above, this textbook focuses not only on the substance of modern logic, but also its style, both in terms of comprehension *and* composition. Usually, students spend most of their time trying to comprehend the material presented in a textbook, and a lot less time (if any) writing about the material. This is not ideal for a variety of reasons. Most importantly, it may very well be the case that you understand most things (if not everything) presented in the book, yet when it comes to writing about them, you produce nothing more than word salad. Trust me, I have seen this happen.

This is especially problematic when it comes to writing *proofs*. It is true that at any one time, you *may* happen upon the correct answer to a question (in some way or another). However, if you cannot write a precise, concise and correct proof for a claim, you are no better off than someone who happened across the wrong answer to the question. The following considerations may help you see why.

The main reason why people write proofs in mathematics is to *persuade* other people that what they believe to be true is actually true. Naturally, in order to persuade someone, you need to present some persuasive reasons as to why you are correct in your assessment of a certain proposition. If you cannot do this, they will not be persuaded by your reasoning, and will not believe that your claim is true, even if you thought about it for a long time and you are really sure about yourself.

Take for example *Golbach's conjecture* (GB), one of the most famous unsolved problems in mathematics. It states:

(GB) Every even natural number greater than 2 is a sum of two prime numbers.

Now it is not hard to *claim* that this conjecture is true (or false, for that matter). Many people believe it is true. However, providing a satisfactory *proof* of this claim (or its opposite) is so hard that no mathematician has managed to do it since it was first formulated around 1742! In fact, it is also the case that if out of 10 people, 5 claim it is true, and 5 claim it is false, then 50% of them will be correct in their assessment! But again, this information is completely useless, since without a proof, we cannot tell which group has the correct assessment.

This book is full of exercises, interspersed with the material, to get you to write. Admittedly, these exercises are not like Goldbach's conjecture. Ideally, your instructor (if you have one) knows the answer, and can demonstrate its correctness, for each and every one of them. Nevertheless, your approach to these exercises should be no different than how a regular mathematician would go about proving Goldbach's conjecture. Namely, for each, you not only have to come up with the correct answer, but you also have to come up with some persuasive reasons as to why your answer is the correct one – in this case, persuading your instructor.

Of course, this is all very abstract. But don't worry! Many of the exercises come with detailed examples, demonstrating how you should go about formulating your answers. The most important thing is that you actually write them down. Again, claiming that you 'did it in your head' and got the correct answer is not enough, even if you did happen upon the correct answer.

Sometimes, the answer to a question may not immediately come to you. In such cases, you should not despair – you should write! In many cases, the answer will come to you once you methodically lay out the relevant available information concerning the question. And if you never arrive at an answer (correct or incorrect), you still have a demonstration of your way of thinking put in writing. For most instructors, this is a lot more valuable than the correct (or incorrect) answer without any justification, for it allows them to identify pain points that need to be addressed.

Chapter structure

Each chapter of this book relies on material presented in previous chapters (except for the first one!). This means that if you skip some sections, or do not read them carefully enough, you might not be able to understand later chapters, or worse still, you may misunderstand what is going on. If you want to make sure that this does not happen, you should try and complete all (or most) of the exercises in each chapter. The exercises are designed to be relatively straightforward, in the sense that you should be able to at least attempt to complete them, even if some mistakes are made here and there. If you find that you do not know how to even start, it is likely that you need to go back and reread the relevant parts of the chapter more closely.

It should be noted that the reason why the book is structured this way is not incidental. It is a feature of modern logic, not a bug, that every logical system is built from the ground up using some basic principles and definitions. This is to ensure that at each step, we know how some propositions follow from some previous propositions, which follow from some previous propositions, and so on, until we get back to the basic principles and definitions. In turn, this ensures that *if* you accept the basic principles and definitions, you *must* accept what follows from them, since each derived proposition can be traced back to the starting point. Unfortunately, this also means that if you want to know what's happening at any one point, you need to understand how we got there from our starting point. Hence the warning in the previous paragraph.

If we zoom in a little, the chapters follow the same blueprint. In each chapter, we start with a discussion of the motivations for the ideas, concepts, and mechanisms that are then introduced gradually, through detailed explanation and examples. This makes up the bulk of each chapter. However, in some sense, all of this is just setup for the most important part of the chapters; the definitions. These definitions are heavy with formalism and thus extremely terse yet rich in meaning. In fact, as mentioned above, everything that we say follows from these definitions.

You might be thinking we are putting the cart before the horse here, since everything we say in a chapter follows from what we say at the end of it. And you would be right! The reason why we do this is because these definitions in themselves are completely incomprehensible for a newcomer without preparation. Be that as it may, once we passed the definition of a concept, you should no longer rely on your preconceptions and intuitions regarding it, since in each case, the definition decides how the concept is to be applied. To borrow a metaphor from the philosopher and logician Ludwig Wittgenstein, these intuitive explanations are meant to function as ladders that should be thrown away after you have climbed up on them.

Don't forget to have fun

Before moving on to the next chapter, one thing you should not forget is to have fun with the material! Admittedly, this might be a tall order at first, but trust me. Though logic might seem like a dry and hopelessly abstract subject, in reality, it gives you tools and techniques to notice patterns and connections in any field of study, and even in everyday life. The examples in this book have been developed specifically to demonstrate this. In each case, you can try to come up with your own examples using the concepts introduced, and you can challenge yourself to make it funny, surprising, insightful, or all three.

License

Chapter 1

Formal languages

INTRO HERE!

1.1 The simplest writing game

Writing games are, unsurprisingly, about writing. They are about putting symbols next to each other following some specified rules, or alternatively, looking at a bunch of symbols next to each other and breaking them down into their 'constituent parts' (meaning the symbols that make them up) according to rules.

The simplest writing game is about putting symbols next to each other without any special rule. But what are these symbols? Well, they can be any symbols you want, although each one will result in a different game. Since we are talking about the simplest writing game, let's start with the simplest of symbol sets – having only one symbol. Let's say this symbol is: ●.

> ☞ What this symbol looks like is not overly important since it is the rules that specify its nature. Think of chess here. Usually, there are a bunch of different-looking chess-pieces, each with their own rules of movement. For example, the king moves one square in any direction on the board. But what the king looks like isn't important. If you were to lose the king piece from your set, you could use a pebble and nobody would bat an eye *provided you followed the rule of how it moves on the board*.

Once we have our symbol ●, we need our one and only rule about putting the symbol down. Clearly, the first step is to allow one to put down ● by itself. Let's formulate it:

RULE 1: As a first step, you can put down ● by itself.

According to RULE 1, I can now do this:

$$\bullet$$

This rule allows us to write down one, and only one, formula, namely, the symbol ● by itself. 'Formula' is the word we will be using to refer to any sequence of symbols of the right sort. The reason for this will become obvious in due time.

But first, notice what happens when we add the following rules:

RULE 2L: At any step, you can put down ● to the left of your existing formula.

RULE 2R: At any step, you can put down ● to the right of your existing formula.

We can now immediately produce a lot more formulas. Here is a sample:

1. ●●
2. ●●●●●●
3. ●●●●
4. ●●●●●●●●

Exercise 1.1.1. With RULE 1, we could only produce one formula. How many formulas can we produce with RULE 1, RULE 2L and RULE 2R together?

—STOP—

The answer is: very many! In fact, *infinitely many*! Here is another thing you can ponder:

Exercise 1.1.2. With RULE 1, RULE 2L and RULE 2R, we can produce infinitely many formulas. Does that mean that any one of these formulas will be infinitely long? *Hint: think of how many natural numbers there are, and how long any representation of a specific number may be.*

Of course, this writing game is not overly interesting, since it can only produce rows of black circles. But we can still ask some interesting questions about it. For example, suppose we do away with RULE 2L and take RULE 2R as our only rule other than RULE 1. Here are two questions you may ask yourself:

Exercise 1.1.3. Can you recreate the same formulas with RULE 1 and RULE 2R that you could with RULE 1, RULE 2L and RULE 2R?

Exercise 1.1.4. Can you tell exactly how a given formula was created using Rule 1, Rule 2L and Rule 2R? How about Rule 1 and Rule 2R?

Speaking of creating formulas, we can also be *very specific* about how we created a given formula in either rule systems. In fact, the type of extremely specific representation we will be using can be adapted for many other (and much more interesting!) rule systems.

The main idea behind an explicit representation of the creation of a formula is that anyone can see if you actually followed the appropriate rules. In other words, for every correct formula, you can (at least in principle) show how it can be constructed. Alternatively, there is no *in*correct formula for which you can show how it could be constructed (with the given rules).

Let's say you are considering the correctness of the following formula:

$$\bullet\bullet\bullet\bullet$$

If this formula is indeed a correct one, we should be able to specify how we built it using only the rules that are given to us. Let's take Rule 1, Rule 2L and Rule 2R first. Here is how we can represent in a compact way how we built the above formula:

(1) ● (by Rule 1)
(2) ●● (by Rule 2L)
(3) ●●● (by Rule 2R)
(4) ●●●● (by Rule 2R)

Let's return to Exercise 1.1.4. There, the question was whether we could tell for a formula like ●●●● how exactly it was created. Now we can show that this is not the case if we have both Rule 2L and Rule 2R since at any step other than the first, we can use either the left or the right hand rule to arrive at the next step.

For example, we could have created our formula as such:

(1) ● (by Rule 1)
(2) ●● (by Rule 2R)
(3) ●●● (by Rule 2L)
(4) ●●●● (by Rule 2L)

Again, the created formula is the exact same, but the way it was created is completely different. On the other hand, this is not the case if we only have Rule 1 and Rule 2R, for then the only way to arrive at our formula is as follows:

(1) ● (by Rule 1)
(2) ●● (by Rule 2R)
(3) ●●● (by Rule 2R)
(4) ●●●● (by Rule 2R)

In fact, we can make this fact more precise as follows:

> Any formula built with RULE 1 and RULE 2R is built using RULE 1 first, and then using RULE 2R a number of times.

Indeed, we can make this fact *even more* precise since with any construction of a formula like above, we start with a formula of a lone symbol, which is 1 character long. Then, at each step, using RULE 2R, we make it 1 symbol longer. Accordingly, if a formula is made up of n black circles (where n is any natural number you can think of), then we know it was built by one application of RULE 1 and $n-1$ applications of RULE 2R.

☞ Let's think a bit about n above. I said that n may be any natural number. The reason why this is useful is that this way, we can state extremely general facts. For example, the above regularity about our formulas holds for *any* formula, no matter its length. So if $n = 1$ it is true, and if $n = 1353463256$, it is also true. Because it is true for *any* natural number we can substitute for n.

1.2 Alphabets, formulas, languages

So far, our formulas are not very interesting. They are just black circles one after another. So let's make some generalizations that allow us to produce more interesting formulas. The first thing we can do is add some more symbols. Since these symbols are just the basic building blocks of more complicated expressions, usually, they are called the *alphabet*.

The English alphabet consists of 26 symbols (normal people call these 'letters'). Ours so far consists of one symbols, ●. There is absolutely no limit as to how many symbols you can have in your alphabet. For now, we shall add 3 more symbols, and declare our alphabet.

ALPHABET: our alphabet consists of the following symbols:

●, ▲, ★, ■

Then, we can change our rules so that the base rule, RULE 1, lets us put down any member of the alphabet by itself, and similarly, so that our productive rule, RULE 2R, lets us put down any member of the alphabet to the right of the formula we already have.

We can also be more specific about what we are doing when we are giving these rules. For in fact, what we are *really* doing is defining what a formula is! So we can write:

BASE RULE: Any member of the alphabet by itself is a formula.

PRODUCTIVE RULE: If X is a formula and Y is a member of the alphabet, then XY is a formula.

This may be confusing at first. What is X? Well, X is any formula! What is Y? Well, Y is any member of our alphabet! But how do we know what counts as a formula (here, X) if we are just now defining it? You already know the answer to this. We build formulas in steps, and at each step, we get a new formula. So anything we can construct with our rules is a formula.

Let's look at an example carefully. Suppose that you want to determine whether ■●■▲ is a formula in our new system. In order to make sure that it is, we need to show that it is, using our rules. As a first step, we know that ■ by itself is a formula, since it is a member of the alphabet (this is the Base Rule). In turn, using the Productive Rule, we can infer that there are at least four other formulas. Namely:

■●, ■▲, ■★, ■■.

This is so because we know ■ is a formula already, and the Productive Rule tells us that if X (here X = ■) is a formula and Y is a member of the alphabet, then XY, the result of putting ■ first, then putting down whatever Y we want from the alphabet, is also a formula. So in particular, ■● is a formula (when Y = ●).

Now we can use this reasoning again, but this time, the X in our Productive Rule is ■●, and Y needs to be ■. So then by the rule, XY, that is, ■●■ is a formula. Finally, using the Productive Rule again with X = ■●■ and Y = ▲, we can see that ■●■▲ is a formula too.

Of course, this is very wordy. But we already know how to make this reasoning more compact. Like this:

(1)	■	(by Base Rule)
(2)	■●	(Productive Rule: 1)
(3)	■●■	(Productive Rule: 2)
(4)	■●■▲	(Productive Rule: 3)

Remark 1.2.1. We call this a *formal derivation*, or simply, *derivation* of the formula ■●■▲ in the language. Indeed, this is a formal *proof* that the sequence of symbols ■●■▲ *is* a formula of the language.

Notice that we are now referring not only to the rule, but to the line number on which the rule is used. This is so since the Productive Rule needs two 'inputs': something we already know to be a formula, the other a member of the alphabet. And what we know to be a formula is what appears on one of the previous lines.

Finally, it is a good idea to have some general term to refer to combinations of alphabets and rules of construction (either of the base type or the productive type) to not get lost in all these different combinations. In the literature, these are called different *languages*. We will come to see why this is the case in due time (though you may already see the connections). At any rate, languages are usually referred to by the fancy letter 'l' like this: $\mathcal{L}$. And when there is more than one, we can use subscripts or superscripts (or both) to distinguish them.

Here is a nice table of the languages we considered so far:

Language	**Alphabet**	**Rules**
$\mathcal{L}_1$	●	Rule 1, Rule 2L, Rule 2R
$\mathcal{L}_2$	●	Rule 1, Rule 2R
$\mathcal{L}_3$	●, ■, ▲, ★	Base Rule, Productive Rule

☞ Note that languages as we defined them are identified by their alphabet and the rules for constructing their formulas. This means that two distinct languages may have the exact same formulas, even if they are constructed through different rules.

Exercise 1.2.1. Consider the languages $\mathcal{L}_1$, $\mathcal{L}_2$, and $\mathcal{L}_3$. Which of these three languages share all their formulas?

Exercise 1.2.2. Consider the languages $\mathcal{L}_2$ and $\mathcal{L}_3$. Is it true that all the formulas of $\mathcal{L}_1$ are formulas of $\mathcal{L}_2$? What about the converse: is it true that all the formulas of $\mathcal{L}_2$ are formulas of $\mathcal{L}_1$?

Sometimes, it is useful to abstract away from the specific symbols of a language, and just concentrate on the 'roles' they play in the writing game. We already discussed this above as the idea relates to chess. Namely, it doesn't matter what forms the chess pieces take as long as they retain the role assigned to them by the rules of the game.

We can formulate this idea related to our languages as follows. If we have a language with a certain alphabet consisting of n distinct symbols, we can replace that alphabet with an alternative one, provided it also has n distinct symbols, and we specify which symbol is exchanged for which new symbol. Then, we can call our new language an *alphabetic variant* of the old one.

What the idea of an alphabetic variant of a language captures is that we are *really* playing the same writing game, but with different looking pieces. Here is an alphabetic variant of $\mathcal{L}_3$: change ● to the letter A, change ■ to the letter E, change ▲ to the letter S, and change ★ to the letter T. The two rules, Base Rule and Productive Rule are the same as before. You can call this language $\mathcal{L}_4$:

Language	**Alphabet**	**Rules**
$\mathcal{L}_4$	A, E, S, T	Base Rule, Productive Rule

Exercise 1.2.3 (The English word game)**.** Here is a new game. Write down as many formulas of the language $\mathcal{L}_4$ as you can that coincide with English words. An example is: EATS. Make sure that you are capable of specifying how each word can be derived using the alphabet and the two rules of $\mathcal{L}_4$

Exercise 1.2.4. Make your own alphabetic variant of one of the languages above. You can use whatever symbols you'd like. Then, give 5 example formulas from your new language.

Exercise 1.2.5. Consider the language $\mathcal{L}_{EA}$, the English Alphabet language. Unsurprisingly, the alphabet of $\mathcal{L}_{EA}$ is the whole (uppercase) English alphabet. The two rules are still the Base Rule and the Productive Rule. Answer the following questions:

1. Are there more formulas of $\mathcal{L}_{EA}$ than English words, or are there more English words than formulas of $\mathcal{L}_{EA}$? In the first case, you should be able to give a formula that is not a word, in the second case, you should be able to give a word that is not a formula.

2. Are the following expressions formulas of $\mathcal{L}_{EA}$?

   (a) COMPUTER

   (b) A.I.

   (c) BLACKBOARD

   (d) BLACK BOARD

   (e) RUN!

3. Is $\mathcal{L}_{EA}$ an alphabetic variant of any of the languages $\mathcal{L}_1$, $\mathcal{L}_2$, $\mathcal{L}_3$, $\mathcal{L}_4$? Don't forget to give an explanation!

To finish this chapter, let's define a language which properly captures a class of expressions we use in everyday life. By 'properly', I mean that every such expression will be a formula of the language, and every formula of the language will be such an expressions. These expressions will be the positive natural numbers, so let's call our language $\mathcal{L}_{\mathbb{N}^+}$ (where $\mathbb{N}^+$ is the symbol for the positive natural numbers).

Every positive natural number is a finite sequence of the numbers 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 with the exception that no number starts with 0. Accordingly, our alphabet will be the following:

Alphabet of $\mathcal{L}_{\mathbb{N}^+}$: the alphabet consists of the symbols 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Then, we have the following slightly modified Base Rule:

Base Rule of $\mathcal{L}_{\mathbb{N}^+}$: every symbol of the alphabet of $\mathcal{L}_{\mathbb{N}^+}$ is a formula except 0.

On the other hand, we don't need to change the Productive Rule, since the Base Rule takes care of any exceptions. Thus:

Productive Rule of $\mathcal{L}_{\mathbb{N}^+}$: If X is a formula and Y is a member of the alphabet, then XY is a formula.

Exercise 1.2.6. Explain why the language $\mathcal{L}_{\mathbb{N}^+}$ is capable of producing all positive natural numbers as formulas, and why it is incapable of producing a formula that is not a positive natural number. In your explanation, mention why there can be no formula produced that starts with a sequence of zeroes (a sequence of the symbol 0).

Exercise 1.2.7. Think of a way to change the Base Rule and the Productive Rule for $\mathcal{L}_{\mathbb{N}^+}$ so that all the non-negative integers (the positive natural numbers plus 0) are produced as formulas, but no 'unnatural numbers' like 000323 are produced.

–STOP–

Here is one way to answer Exercise 1.2.7 and define the language $\mathcal{L}_{\mathbb{N}}$. Change the base rule so that it does allow for having each symbol as a formula by itself.

Base Rule of $\mathcal{L}_{\mathbb{N}}$: every symbol of the alphabet of $\mathcal{L}_{\mathbb{N}^+}$ is a formula.

Then, we make sure that the Productive Rule does not allow construction from 0, only from the other symbols as follows:

Productive Rule for $\mathcal{L}_{\mathbb{N}}$: If X is a formula other than 0 and Y is a member of the alphabet, then XY is a formula.

Exercise 1.2.8. It is obvious that the new rule does not allow the construction of formulas like 05. Does it allow the construction of longer 'unnatural' numbers like '00304'? Check that this is indeed impossible with the above rules.

Exercise 1.2.9. A trickier question. After you have a specification for the language $\mathcal{L}_{\mathbb{N}}$, can you give one for $\mathcal{L}_{\mathbb{Z}}$, which consists of all integers? Note: integers are the positive and negative natural numbers, plus zero.

1.3 More elaborate writing games

So far, our writing games basically put us into the role of a typewriter, where at each step, we could make a formula 1 symbol longer from our alphabet than it was before. The problem with this approach is that as it stands, it doesn't really distinguish between those sequences of symbols that may be interesting for one reason or another, and those which are just gibberish.

In fact, we *really* didn't need to go to the lengths we did, using base rules and productive rules, to specify any of the languages above. For note that in general, given their respective alphabets, in almost all of these languages, any finite sequence of symbols from the alphabet constituted a formula. With some modification, this definition can also be adapted to the languages $\mathcal{L}_{\mathbb{N}}$ and $\mathcal{L}_{\mathbb{Z}}$.

Exercise 1.3.1. As just mentioned, for the languages preceding $\mathcal{L}_{\mathbb{N}}$ and $\mathcal{L}_{\mathbb{Z}}$, we could specify immediately: *Any finite sequence of symbols of the alphabet is a formula of the language.* How would you change this definition to capture being a formula of $\mathcal{L}_{\mathbb{N}}$ and $\mathcal{L}_{\mathbb{Z}}$?

1.4 The language $\mathcal{L}_f$

To illustrate another way, most languages you are familiar with have various levels of expressions, and at each level, certain 'well-formed' expressions are distinguished from those that are not so. For example, as mentioned above, $\mathcal{L}_{EA}$ is the English Alphabet language, which has as formulas all finite sequences of letters of the English alphabet. This entails that every English word is a formula of $\mathcal{L}_{EA}$, but most formulas of $\mathcal{L}_{EA}$ are just random combinations of letters and not words of the English language. Similarly, when it comes to combinations of words in English, most of them are not well-formed. For example, while "I am running late from class" is a well-formed expression, "am class running I late from" is not. In other words, English sentences make up a small subset of all combinations of words of English.

To make these ideas more vivid, we can introduce a new language, which we will call $\mathcal{L}_F$. 'F' in the subscript stands for *file system*. You are probably familiar with file systems on your computer, that is, the *files* and *folders* on your computer that you can navigate with a click of a button. These file systems have several 'levels', based on the folders you have. Importantly, folders are stackable, so that you can have a folder inside a folder, and a folder inside a folder inside a folder, etc. The language $\mathcal{L}_F$ will be able to specify which folders and files are in which folders, and once given an expression, you will be able to read off the file structure given by a certain formula of the language.

☞ One thing that you may be less familiar with is the idea of a *root* folder. Note that on every computer, files and folders are in other folders, but only until they aren't. Specifically, there is a folder on your computer that is not itself in any folder. That is your `root` (on Windows, think of `C:\`). As we specify the language $\mathcal{L}_f$ in the following, we will be assuming that we are either in the root folder or a relative root folder (the 'working directory'). Essentially, we will be able to specify what is 'under' the folder we are working in, but not anything above it.

To start with our usual specification of the alphabet, we shall make use of certain base expressions, denoting (intuitively) the single files of our file system, and other expressions that are used merely to give structure to more complex expressions.

Let's suppose that we have 5 distinct files, which we may denote f_1, f_2, f_3, f_4 and f_5 for simplicity. Next, we shall use the comma symbol ',' to delineate individual files and folders that are 'on the same level', that is, in the same folder. For example, we may say something like f_1, f_3, f_5 to specify there are three files. We also need a way to signal that some files and folders together form a folder. For this, we can put everything that belongs to a folder into curly brackets '{' and '}'. So f_1, f_3, f_5 denotes the files at the highest level, while $\{f_1, f_3, f_5\}$ denotes these files in a single folder.

The above 'intuitive' specification can be put into our usual notation as follows.

Alphabet: The alphabet of $\mathcal{L}_f$ consists of the symbols:

$$f_1 \mid f_2 \mid f_3 \mid f_4 \mid f_5 \mid , \mid \{ \mid \}$$

I used the symbol '|' to give the above list since the comma symbol ',' is itself a symbol of our language, so listing it with commas would look confusing. Specifying a language inside another is usually a painful experience.

Now for the base rule, we can say the following:

Base Rule of $\mathcal{L}_f$: the symbols f_1, f_2, f_3, f_4, f_5 by themselves are all formulas. Moreover, $\{\}$ is also a formula.

Then, we need two separate productive rules, one for listing the contents of a folder, and another for denoting that some list of things is *in* a folder. Accordingly:

Productive Rule 1 of $\mathcal{L}_f$: if X and Y are formulas, then X, Y is a formula.

Productive Rule 2 of $\mathcal{L}_f$: if X is a formula, then $\{X\}$ is a formula.

Now we can start writing down formulas, and then try to understand what they actually say. One thing that immediate looks out of place is $\{\}$. Intuitively, what $\{\}$ denotes is the empty folder. After all, there may be folders that do not have anything in them. In turn, there can be several empty folders in a folder, and in general, there may be a whole tower of folders that do not have any files at any level.

Exercise 1.4.1. Check whether the following sequences of symbols are formulas of the language $\mathcal{L}_f$, and if so, try to write down what file structure they represent:

1. $\{\}$
2. $\{\}, \{\}$
3. $\{\{\}, \{\}\}$
4. $\{\{\{\}\}$
5. $\{\{\{\}, \{\}\}, \{\}\}$
6. $\{\{\}, \{\}\}, \{\}$

Of course, you can have file systems with actual files in them too, and you can specify these in our language.

Exercise 1.4.2. Check whether the following sequences of symbols are formulas of the language $\mathcal{L}_f$, and if so, try to write down what file structure they represent:

1. $f_1, f_2, \{\{f_2, f_4\}, \{f_3, f_3\}\}$
2. $\{f_1, \{\}\}$
3. f_1
4. $\{\{\{\{f_4 f_2\}\}\}\}$
5. $\{\{f_2\}, \{\{f_3\}, f_5\}, \{f_1\}\}$
6. $\{f_1\}, \{\{f_2\}\}, \{\{\{f_3\}\}\}$

Exercise 1.4.3. Look at the first formula of Exercise 1.4.2 again. Can you see something peculiar in what it says? How would you make sense of it?

Let's return to the distinction between well-formed formulas and random sequences of symbols of the alphabet. Given how the language $\mathcal{L}_f$ is specified, not every sequence of symbols of the alphabet constitutes a formula. In Exercises 1.4.1 and 1.4.2, there were two sequences of symbols of the alphabet that are not formulas of the language[1] Here, it actually

[1] Specifically, number 4 in each.

becomes useful to show by a derivation as before whether something is a formula of the language.

To save some space, let's call PRODUCTIVE RULE 1 'PR1', and PRODUCTIVE RULE 2 'PR2'. Similarly, we can use 'BR' instead of BASE RULE. Here are two sample derivations of formulas from above.

(1)	$\{\}$	(BR)
(2)	$\{\},\{\}$	(PR1: 1, 1)
(3)	$\{\{\},\{\}\}$	(PR2: 2)
(4)	$\{\{\},\{\}\},\{\}$	(PR1: 1, 3)

Figure 1.1: Derivation of $\{\{\},\{\}\},\{\}$.

(1)	f_1	(BR)
(2)	$\{\}$	(BR)
(3)	$f_1,\{\}$	(PR1: 1, 2)
(4)	$\{f_1,\{\}\}$	(PR2: 3)

Figure 1.2: Derivation of $\{f_1,\{\}\}$.

Exercise 1.4.4. Derive all the other formulas of the language $\mathcal{L}_f$ from Exercises 1.4.1 and 1.4.2. Note: if something is not a formula, you cannot derive it (though you can certainly try).

Finally, let's think a bit about $\{$ and $\}$. These curly braces are really what give our language its structure and its expressive power. In fact, though the comma is useful for humans like us to parse formulas, it is not really necessary. We can change the PR1 to omit it like this:

> PRODUCTIVE RULE 1* of $\mathcal{L}_f$: if X and Y are formulas, then XY is a formula.

Then, we get the following formulas from above:

$$\{\{\},\{\}\},\{\} \mapsto \{\{\}\{\}\}\{\}$$
$$\{f_1,\{\}\} \mapsto \{f_1\{\}\}$$

Exercise 1.4.5. Rewrite all the formulas of $\mathcal{L}_f$ from Exercises 1.4.1 and 1.4.2 assuming we changed PR1 into PR1* (in other words, we did away with the previous comma convention).

On the other hand, we really cannot do away with our curly braces, since they are the ones that tell us what we should consider as being in a folder. It is very important where we put those braces, since the resulting formula may represent something entirely different from what we intended. For example, if you are interested in how many folders there are in your root folder, it matters whether the formula is $\{\}, \{\}, \{\}, \{\}$ or $\{\{\}, \{\}, \{\}, \{\}\}$. The first one says there are 4 folders, the second says there is only one.

Exercise 1.4.6. How many files or folders are there in `root` if the formula specifies the file structure $f_1, f_2, \{\{f_2, f_4\}, \{f_3, f_3\}\}$?

1.5 The language $\mathcal{L}_{AE}$

The next language we will consider is the language $\mathcal{L}_{AE}$, the language of artihmetic expressions. This language is something you are familiar with from your high school studies. Essentially, expressions in $\mathcal{L}_{AE}$ are the arithmetic expressions that you had to compute with, and which can flank the identity symbol =. For example, $(3+4)-(5\times 6)$, (300×33), $(5555-3333)$, (23×3). Note that which numbers we are working with is not immaterial, since these numbers will appear in these formulas. Accordingly, we will be using the positive and negative integers plus 0 (i.e., $\mathcal{L}_{\mathbb{Z}}$).

Let's start by explicitly defining $\mathcal{L}_{\mathbb{Z}}$. Making use of the above simplifications regarding expressions without structural delineators, we may say:

Alphabet of $\mathcal{L}_{\mathbb{Z}}$: The alphabet of the language consists of $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$ and $-$.

Formulas of $\mathcal{L}_{\mathbb{Z}}$: Any finite sequence of symbols $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$ is a *positive integer* provided the first member of the sequence is not 0. If X is a positive integer, $-X$ is a negative integer. Then, X is an integer (formula) provided it is a positive integer, a negative integer, or it is 0.

Remark 1.5.1. Note that this definition is slightly different from the previous ones. It first directly defines the positive integers. Then, it defines the negative integers from the positive ones. Then, it considers 0, and adds these three groups together.

Based on $\mathcal{L}_{\mathbb{Z}}$, one can define the language of arithmetic expressions, $\mathcal{L}_{AE}$ quite simply. Clearly, its alphabet will consist of the alphabet of $\mathcal{L}_{\mathbb{Z}}$, plus additional symbols to formulate arithmetic expressions. These symbols are the usual ones, that is: $+$, $\times$, and $-$.[2] We also need the crucial delineators, which in arithmetic expressions are just the parentheses (and).

Thus, we have:

Alphabet of $\mathcal{L}_{AE}$: the symbols $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$, $+$, $\times$, $-$, (, and).

Base Rule of $\mathcal{L}_{AE}$: every integer (formula of $\mathcal{L}_{\mathbb{Z}}$) is an arithmetic expression.

Productive Rule of $\mathcal{L}_{AE}$: if X and Y are arithmetic expressions, then $(X+Y)$, $(X-Y)$, and $(X\times Y)$ are arithmetic expressions.

[2] The division operator $\div$ is omitted since not every integer divided by another will result in an integer. This is not technically a problem for the language, but it would be confusing.

Now as before, for any arithmetic expression we can come up with, we can prove that they *are* formulas of $\mathcal{L}_{AE}$. For example, take $((414 \times -14134) \times (835 + 345))$.

(1)	414	(BR)
(2)	-14134	(BR)
(3)	(414×-14134)	(PR: 1,2)
(4)	835	(BR)
(5)	345	(BR)
(6)	$(835 + 345)$	(PR: 4,5)
(7)	$((414 \times -14134) \times (835 + 345))$	(PR: 3, 6)

Or another one of a different form: $(34 \times ((99 - -36) \times 9))$.

(1)	99	(BR)
(2)	-36	(BR)
(3)	$(99 - -36)$	(PR: 1,2)
(4)	9	(BR)
(5)	$((99 - -36) \times 9)$	(PR: 3, 4)
(6)	34	(BR)
(7)	$(34 \times ((99 - -36) \times 9))$	(PR: 6, 5)

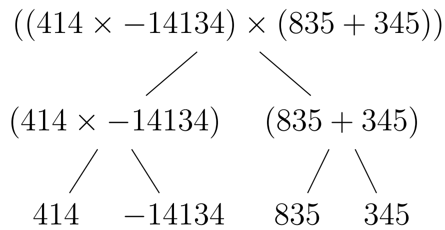
Let us continue our discussion of the structure of our newfound expressions. In arithmetic expressions, delineators are used to give an order to the computation represented by the formulas. Sometimes, the order in which these operations are carried out does not matter, but many times, it makes a crucial difference. For example, $(4+5)+3$ computes to the same number as $4+(5+3)$ (namely, 12), but $(4-5)-3$ does not compute to the same number as $4-(5-3)$. In fact, the first one is a negative number, -4, while the second is a positive number, 2. So it is clearly very important to place parentheses in the right places.

Exercise 1.5.1. Give an example of an arithmetic expressions with three (not necessarily distinct) numbers connected by two (not necessarily distinct) arithmetic operations where how the parentheses are placed does not matter. Then, give another example where how the parentheses are placed does matter. Don't forget to write down your reasoning in each case.

What is not very helpful about the linear derivations that we have been using so far is that in their line-by-line representation, they do not really show us visually what the structure of our expressions is. However, this can be remedied by using so-called *syntactic trees* to represent how formulas are formed.

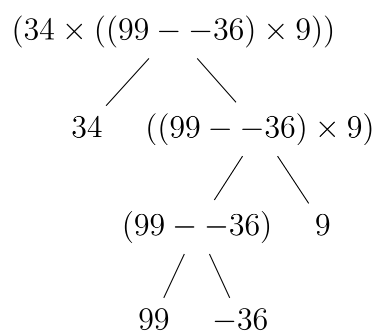
Note that in each derivation, we first take the base expressions, and then form more complex expressions, and then more complex expressions from previous expressions, until we get to the desired formula. Moreover, more complex expressions are formed by putting together two simpler expressions with an arithmetic operator, flanked by the left and right parentheses.

Let's return to $((414 \times -14134) \times (835 + 345))$ as our example. Then, we could represent this in tree-form as follows:



As you can see, this type of representation shows you the starting points, which we get by the Base Rule. These are at the bottom. Then, as we build up the more complex formulas, it shows how we put together those simpler expressions to get the more complex ones.

Our other example, $(34 \times ((99 - -36) \times 9))$, will result in a different looking tree. Namely:



You can also read these trees from top to bottom. In fact, for some reason, mathematicians call the highest formula the *root* of the tree, from which it *branches*. The lowest points (to which no other points are connected) are the tree's *leaves*. Finally, any path from the root to one of the leaves is a *branch*. So really, the tree is upside-down!

At any rate, if reading the tree from bottom-to-top tells you how a formula is formed, reading it from top-to-bottom tells you how a formula can be broken down into its constituent parts.

You might be wondering: which formulas should I take as constituent when a tree branches? For example, why is it the case that in the first example, we branched to (414×-14134) and $(835 + 345)$, while in the second example, we branched to 34 and

$((99 - -36) \times 9)$? The answer: the parentheses tell us. At each step, there is a *main arithmetic operator*, which connects together the two formulas we branch to. Other arithmetic operators occur inside parentheses, and are to be broken down at a later step.

Exercise 1.5.2. Decide which is the main operator in each of these formulas:

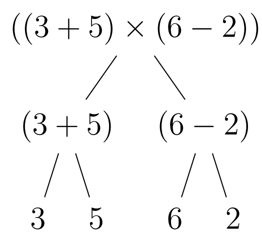
1. $((93 + 73) \times (43 - 15))$
2. $(975 \times (44 + 1))$
3. $((77 \times 3) - 33)$
4. $(((3 \times (5 + 4)) \times 9) - 7)$
5. $((((3 \times 5) + 4) \times 9) - 7)$
6. $(((3 \times 5) + 4) \times (9 - 7))$

When you are drawing a syntactic tree, it is useful to start with the formula you are aiming to construct. Then, at each step, you have to find the main operator, and put the two formulas it connects on separate branches below it. If you repeat this process enough times, each tree will have integers on its leaves. If some leaf is not an integer, you have to continue.

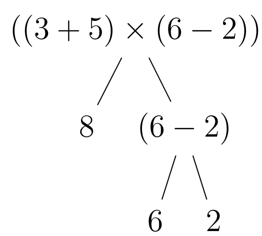
Exercise 1.5.3. Construct syntax trees for each of the six formulas in Exercise 1.5.2. Observe the difference between the last three formulas, which only differ in their structure (the way the parentheses are distributed).

As discussed above, parentheses determine the order of computation for each arithmetic expression. And in fact, when you look at syntax trees for these expressions, you can read off the order of computation from their structure. In order to compute any arithmetic expression, you need to start from the leaves, which are integers. Then, the next thing you have to compute is the expression immediately above. Once computed, you can move to the next level, substituting the result of the computation for the occurrence of the expression in the more complex formula above, then computing again. Repeating these steps until you get to the top will give you the final result.

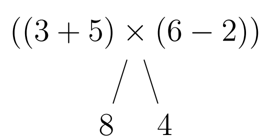
This description is probably a bit confusing on first read, so here is a simple example with $(3 + 5) \times (6 - 2)$. The starting point is the whole tree:



Then, take left side first and compute $(3 + 5)$, resulting in:



Then, we compute the right side, getting:



We can then substitute the results of our computation in the more complex formula above. This will result in:

$$8 \times 4$$

Accordingly, the answer is 32.

1.6 The language $\mathcal{L}_0$

1.6.1 Individual constants

We now have most of what we need to specify a very important logical language we will be working with, the language of *zeroth-order* logic, denoted $\mathcal{L}_0$. The alphabet of $\mathcal{L}_0$ consists, first, of infinitely many symbols called *individual constants* or *names.* These may be represented by *indexing* a single symbol with the natural numbers, as if we had a list. Like this:

$$\mathfrak{c}_1, \mathfrak{c}_2, \mathfrak{c}_3, \mathfrak{c}_4, \mathfrak{c}_5, ...$$

You can read this as: $\mathfrak{c}_1$ ('cee one') is the first constant, $\mathfrak{c}_2$ ('cee two') is the second constant, $\mathfrak{c}_3$ ('cee three') is the third constant, and so on. (The *indices*, in order, here would be 1, 2, 3, and so on.) Clearly, we cannot list all of the constants (for each n), just as we cannot list *all* the natural numbers. That would take an infinitely long paper, and let's not talk about the time involved! This is why we have the three dots ..., implying it goes on forever. More succinctly, we may say: $\mathfrak{c}_n$ is a constant for every natural number n.

Note also that the use of $\mathfrak{c}$ here is completely arbitrary. I chose it because it is the first letter in the word 'constant', but I could have chosen $\mathfrak{a}$ or $\mathfrak{b}$. On the other hand, once we specify that we use $\mathfrak{c}$, we have to stick with it. In other words, the choice of $\mathfrak{c}$ *is* arbitrary, but using $\mathfrak{c}$ afterwards is not arbitrary, since we explicitly chose it over other alternatives.

1.6.2 Predicates

Second, similar to constants, we also have infinitely many symbols called *predicates* in our alphabet. These will be denoted by $\mathfrak{P}$ (like P, but in a different font). In fact, these predicates come with an additional index called their *arity*, denoting the number of arguments they each take (see below). So really, for each natural number k, there are infinitely many predicates for that natural number (arity) k. This might be a bit confusing at first, so let's look at some examples.

First, the infinitely many predicates of arity 1 can be represented:

$$\mathfrak{P}^1_1, \mathfrak{P}^1_2, \mathfrak{P}^1_3, \mathfrak{P}^1_4, \mathfrak{P}^1_5, \ldots$$

Then, the infinitely many predicates of arity 2 can be represented:

$$\mathfrak{P}^2_1, \mathfrak{P}^2_2, \mathfrak{P}^2_3, \mathfrak{P}^2_4, \mathfrak{P}^2_5, \ldots$$

So really, the list goes on infinitely not just horizontally, but vertically too! So in full generality:

$$\begin{gathered}\mathfrak{P}^1_1, \mathfrak{P}^1_2, \mathfrak{P}^1_3, \mathfrak{P}^1_4, \mathfrak{P}^1_5, \ldots\\ \mathfrak{P}^2_1, \mathfrak{P}^2_2, \mathfrak{P}^2_3, \mathfrak{P}^2_4, \mathfrak{P}^2_5, \ldots\\ \mathfrak{P}^3_1, \mathfrak{P}^3_2, \mathfrak{P}^3_3, \mathfrak{P}^3_4, \mathfrak{P}^3_5, \ldots\\ \vdots\end{gathered}$$

Just as before, this may be represented a lot more succinctly by simply saying: $\mathfrak{P}^k_n$ is a constant for each pair of natural numbers k, n. In other words, no matter what natural number you choose for k and n, substituting it for k and n in $\mathfrak{P}_n^k$ will get you a predicate of the language. Sometimes, we may say $\mathfrak{P}_n^k$ is a k-place predicate.

> It is important to note that in a predicate like $\mathfrak{P}_9^4$, the number 4 is called its *arity*, and the number 9 is called its *index*. For example, though 4 may look like the *power* or *exponent*, this is just because of the similar notation. In reality, these two ideas are not related in any way, and should not be confused.

Exercise 1.6.1. Decide for the following symbols whether they are a constant or a predicate. If they are a predicate, identify their arity.

1. $\mathfrak{c}_5$
2. $\mathfrak{c}_{67}$
3. c_4^5
4. $\mathfrak{P}_7^4$
5. $\mathfrak{P}_{456}^{98}$
6. $\mathfrak{P}_1^{1000000}$
7. $\mathfrak{P}_{1000000}^1$

1.6.3 The connectives and the rest

Just like with the language $\mathcal{L}_{AE}$, simpler expressions will be combined together to form more complex expressions using special symbols (similar to the $+$, $-$, and $\times$ signs). We call these symbols *connectives*, for obvious reasons. Table 1.1 lists the four connectives we will be using. Note that each symbol comes with a fixed arity, like our predicates. The table also includes, in scare quotes, the closest natural language approximation for the meaning of these symbols. For now (and perhaps altogether), this is irrelevant.

Symbol	Name	Arity
$\wedge$	conjunction, ‘and’	2
$\vee$	disjunction, ‘or’	2
$\rightarrow$	conditional, ‘if-then’	2
$\neg$	negation, ‘not’	1

Table 1.1: The connectives of $\mathcal{L}_0$

☞ You may already know some or all of these connectives by their name, but perhaps not by their specific symbol (e.g., $\sim$ instead of $\neg$, & instead of $\wedge$). As noted when talking about alphabetic variants previously (see p. 6), what the exact symbols of the alphabet are do not really matter, only the role they play. On the other hand, most contemporary writings on logic use these symbols for the connectives, as opposed to some of the older variants, so our choice is not entirely arbitrary.

Finally, we will keep using the left and right parentheses '(' and ')', along with the comma symbol ','. Thus:

Alphabet of $\mathcal{L}_0$: The alphabet of $\mathcal{L}_0$ consists of the following: for each natural number n the constant $\mathfrak{c}_n$, for each pair of natural numbers n and k a predicate $\mathfrak{P}^k_n$ (of arity k), the connectives $\wedge, \vee, \neg, \rightarrow$, the left '(' and right ')' parentheses, and the comma ','.

1.6.4 The formulas of $\mathcal{L}_0$

Now that we have our alphabet, we can look at how our formulas are built up. Just as with some of the other languages we considered, not any sequence of symbols will qualify as a formula. Our base rule looks like this:

Base Rule of $\mathcal{L}_0$: if $\mathfrak{c}_{n_1}, ..., \mathfrak{c}_{n_k}$ are k-many (not necessarily distinct) individual constants and $\mathfrak{P}^k_i$ is a predicate of arity k, then $\mathfrak{P}^k_i(\mathfrak{c}_{n_1}, ..., \mathfrak{c}_{n_k})$ is a formula. We shall also call any such formula an *atomic formula*.

This is an extremely precise formulation of our base rule, and thus can be rather confusing at first. However, it really isn't very complicated, for all it says is that if you take any predicate with arity k, then you need to have k individual constants following it in order for it to be a formula. In other words, any predicate $\mathfrak{P}^k_n$ wears on its sleeves the number of individual constants it demands – namely, k many! Note however that these individual constants need not be distinct. For example, for $\mathfrak{P}^2_5$, we may write $\mathfrak{P}^2_5(\mathfrak{c}_5, \mathfrak{c}_5)$ just as well as $\mathfrak{P}^2_5(\mathfrak{c}_5, \mathfrak{c}_3)$.

Exercise 1.6.2. Determine if the following are (atomic) formulas of the language $\mathcal{L}_0$. In each case, explain your reasoning.

1. $\mathfrak{P}^3_1(\mathfrak{c}_3, \mathfrak{c}_5, \mathfrak{c}_1)$
2. $\mathfrak{P}^1_4(\mathfrak{c}_{66})$

3. $\mathfrak{P}^5_5(\mathfrak{c}_5, \mathfrak{c}_4, \mathfrak{c}_3, \mathfrak{c}_2, \mathfrak{c}_1)$

4. $\mathfrak{P}^2_6(\mathfrak{c}_2, \mathfrak{c}_2)$

5. $\mathfrak{P}^2_6(\mathfrak{c}_2)$

6. $\mathfrak{P}^2_3(\mathfrak{c}_4, \mathfrak{c}_9, \mathfrak{c}_1)$

After our atomic formulas are defined, we can give our usual productive rule, which enables us to form more complex, i.e., 'non-atomic' formulas iteratively.

PRODUCTIVE RULE OF $\mathcal{L}_0$: if X and Y are formulas of $\mathcal{L}_0$, then the following are also formulas of $\mathcal{L}_0$:

1. $\neg X$;

2. $(X \wedge Y)$;

3. $(X \vee Y)$;

4. $(X \to Y)$.

Of course, X and Y are variables, which can denote (as noted) any formula of $\mathcal{L}_0$, both atomic and non-atomic (complex).

Exercise 1.6.3. Determine whether the following expressions are formulas of the language $\mathcal{L}_0$. If not, explain why, and how they could be made into formulas of the language.

1. $((\neg\mathfrak{P}^3_4(\mathfrak{c}_1, \mathfrak{c}_2, \mathfrak{c}_1)\neg\mathfrak{P}^1_4(\mathfrak{c}_3)) \to \mathfrak{P}^2_5(\mathfrak{c}_4, \mathfrak{c}_4))$

2. $((\neg\mathfrak{P}^3_4(\mathfrak{c}_1, \mathfrak{c}_2, \mathfrak{c}_1) \wedge \neg\mathfrak{P}^1_4(\mathfrak{c}_3)) \to \mathfrak{P}^2_5(\mathfrak{c}_4, \mathfrak{c}_4))$

3. $(\neg\mathfrak{P}^3_4(\mathfrak{c}_1, \mathfrak{c}_2, \mathfrak{c}_1) \wedge \neg\mathfrak{P}^1_4(\mathfrak{c}_3)) \to \mathfrak{P}^2_5(\mathfrak{c}_4, \mathfrak{c}_4))$

4. $((\neg\mathfrak{P}^3_4(\mathfrak{c}_1, \mathfrak{c}_2, \mathfrak{c}_1) \wedge \neg\mathfrak{P}^1_4(\mathfrak{c}_3)) \to \mathfrak{P}^4_5(\mathfrak{c}_4, \mathfrak{c}_4))$

5. $((\neg(\neg\mathfrak{P}^3_4(\mathfrak{c}_1, \mathfrak{c}_2, \mathfrak{c}_1)) \wedge \neg\mathfrak{P}^1_4(\mathfrak{c}_3)) \to \mathfrak{P}^2_5(\mathfrak{c}_4, \mathfrak{c}_4))$

1.6.5 Analyzing formulas; again

As before, we can now start analyzing the possible formulas of the language $\mathcal{L}_0$, checking whether they are in fact well-formed formulas of the language, and how they can be constructed. We previously saw two ways of doing this, one *linear*, the other making use of *trees*. These two ways are both applicable to formulas of $\mathcal{L}_0$. In general, X is a formula of the language $\mathcal{L}_0$ if, and only if, it has a linear derivation and a tree derivation. Thus, if an expression cannot be derived, it is not a formula of $\mathcal{L}_0$, but if it is a formula of $\mathcal{L}_0$, you must be able to derive it somehow.

As an example, let's analyze the formula:

$$((\neg\mathfrak{P}^3_4(\mathfrak{c}_1, \mathfrak{c}_2, \mathfrak{c}_1) \wedge \neg\mathfrak{P}^1_4(\mathfrak{c}_3)) \rightarrow \mathfrak{P}^2_5(\mathfrak{c}_4, \mathfrak{c}_4))$$

From bottom to top, *linearly*, we can show that this is indeed a formula of the language as follows:

(1)	$\mathfrak{P}^3_4(\mathfrak{c}_1, \mathfrak{c}_2, \mathfrak{c}_2)$	(BR)
(2)	$\mathfrak{P}^1_4(\mathfrak{c}_3)$	(BR)
(3)	$\mathfrak{P}^2_5(\mathfrak{c}_4, \mathfrak{c}_4)$	(BR)
(4)	$\neg\mathfrak{P}^3_4(\mathfrak{c}_1, \mathfrak{c}_2, \mathfrak{c}_1)$	(PR $\neg$: 1)
(5)	$\neg\mathfrak{P}^1_4(\mathfrak{c}_3)$	(PR $\neg$: 2)
(6)	$(\neg\mathfrak{P}^3_4(\mathfrak{c}_1, \mathfrak{c}_2, \mathfrak{c}_1) \wedge \neg\mathfrak{P}^1_4(\mathfrak{c}_3))$	(PR $\wedge$: 4, 5)
(7)	$((\neg\mathfrak{P}^3_4(\mathfrak{c}_1, \mathfrak{c}_2, \mathfrak{c}_1) \wedge \neg\mathfrak{P}^1_4(\mathfrak{c}_3)) \rightarrow \mathfrak{P}^2_5(\mathfrak{c}_4, \mathfrak{c}_4))$	(PR $\rightarrow$: 3, 6)

From top to bottom, using a *tree* and visualizing its structure, we can do it as follows:

$$\begin{tikzcd}[column sep=small]
 & & ((\neg\mathfrak{P}^3_4(\mathfrak{c}_1, \mathfrak{c}_2, \mathfrak{c}_1) \wedge \neg\mathfrak{P}^1_4(\mathfrak{c}_3)) \rightarrow \mathfrak{P}^2_5(\mathfrak{c}_4, \mathfrak{c}_4)) \arrow[dl, no head] \arrow[dr, no head] & \\
 & (\neg\mathfrak{P}^3_4(\mathfrak{c}_1, \mathfrak{c}_2, \mathfrak{c}_1) \wedge \neg\mathfrak{P}^1_4(\mathfrak{c}_3)) \arrow[dl, no head] \arrow[dr, no head] & & \mathfrak{P}^2_5(\mathfrak{c}_4, \mathfrak{c}_4) \\
\neg\mathfrak{P}^3_4(\mathfrak{c}_1, \mathfrak{c}_2, \mathfrak{c}_1) \arrow[d, no head] & & \neg\mathfrak{P}^1_4(\mathfrak{c}_3) \arrow[d, no head] & \\
\mathfrak{P}^3_4(\mathfrak{c}_1, \mathfrak{c}_2, \mathfrak{c}_1) & & \mathfrak{P}^1_4(\mathfrak{c}_3) &
\end{tikzcd}$$

As before, we can also ask about the *main operator* for each formula. The main operator is always the one introduced from the previous level or levels. Here:

1. The formulas $\mathfrak{P}^3_4(\mathfrak{c}_1, \mathfrak{c}_2, \mathfrak{c}_1)$, $\mathfrak{P}^1_4(\mathfrak{c}_3)$, and $\mathfrak{P}^2_5(\mathfrak{c}_4, \mathfrak{c}_4)$ are atomic, and have no operator;

2. The main operator of $\neg\mathfrak{P}^3_4(\mathfrak{c}_1, \mathfrak{c}_2, \mathfrak{c}_1)$ and $\neg\mathfrak{P}^1_4(\mathfrak{c}_3)$ is $\neg$;

3. The main operator of $(\neg\mathfrak{P}^3_4(\mathfrak{c}_1, \mathfrak{c}_2, \mathfrak{c}_1) \wedge \neg\mathfrak{P}^1_4(\mathfrak{c}_3))$ is $\wedge$;

4. The main operator of $((\neg\mathfrak{P}^3_4(\mathfrak{c}_1, \mathfrak{c}_2, \mathfrak{c}_1) \wedge \neg\mathfrak{P}^1_4(\mathfrak{c}_3)) \rightarrow \mathfrak{P}^2_5(\mathfrak{c}_4, \mathfrak{c}_4))$ is $\rightarrow$.

Exercise 1.6.4. Derive the following formulas using either the linear method or the tree method. In each case, determine what the main operator of the formulas is.

1. $((\mathfrak{P}_2^2(\mathfrak{c}_3, \mathfrak{c}_5) \vee \neg\mathfrak{P}_2^4(\mathfrak{c}_4, \mathfrak{c}_3, \mathfrak{c}_3, \mathfrak{c}_1)) \vee \mathfrak{P}_1^1(\mathfrak{c}_1))$

2. $(\mathfrak{P}_2^2(\mathfrak{c}_3, \mathfrak{c}_5) \vee \neg(\mathfrak{P}_2^4(\mathfrak{c}_4, \mathfrak{c}_3, \mathfrak{c}_3, \mathfrak{c}_1) \vee \mathfrak{P}_1^1(\mathfrak{c}_1)))$

3. $(\mathfrak{P}_2^2(\mathfrak{c}_3, \mathfrak{c}_5) \rightarrow (\neg\mathfrak{P}_2^4(\mathfrak{c}_4, \mathfrak{c}_3, \mathfrak{c}_3, \mathfrak{c}_1) \rightarrow \mathfrak{P}_1^1(\mathfrak{c}_1)))$

4. $(\neg\mathfrak{P}_2^2(\mathfrak{c}_3, \mathfrak{c}_5) \rightarrow (\mathfrak{P}_2^4(\mathfrak{c}_4, \mathfrak{c}_3, \mathfrak{c}_3, \mathfrak{c}_1) \wedge \neg\mathfrak{P}_1^1(\mathfrak{c}_1)))$

5. $(\neg\neg\mathfrak{P}_2^2(\mathfrak{c}_3, \mathfrak{c}_5) \rightarrow \neg(\mathfrak{P}_2^4(\mathfrak{c}_4, \mathfrak{c}_3, \mathfrak{c}_3, \mathfrak{c}_1) \wedge \neg\mathfrak{P}_1^1(\mathfrak{c}_1)))$

6. $\neg(\neg(\neg\mathfrak{P}_2^2(\mathfrak{c}_3, \mathfrak{c}_5) \rightarrow \neg\mathfrak{P}_2^4(\mathfrak{c}_4, \mathfrak{c}_3, \mathfrak{c}_3, \mathfrak{c}_1)) \vee \neg\mathfrak{P}_1^1(\mathfrak{c}_1))$

1.6.6 Why the weird typeface?

You may be wondering why we are using this weird typeface where a simple P looks like this: $\mathfrak{P}$; and a simple c looks like this: $\mathfrak{c}$. This is because as we move forward, we will start using P, c, and some other letters to simplify our formulas if it does not matter at that moment which exact formula of the language we are talking about. This often happens when we want to talk about a class of formulas sharing the same structure. Thus, they will serve a separate, but very important, purpose.

> It is important that you do not mix the notation freely! In logic, different ways of writing the same thing may have a different meaning. For example, p, P, $\mathbf{P}$ and $\mathfrak{P}$ will all mean different things. You should be careful which is used at any one point (see immediately below for a distinction!).

Variables like P and c are usually called *meta*variables, emphasizing that they are *not* part of the language, but outside of it. You may think of n here and the natural numbers. When we want to talk about some natural number or other, we use n, even though n is not a (representation of a) natural number itself (it is not a formula of $\mathcal{L}_{\mathbb{N}}$). Similarly, when we want to talk about some predicate or other, we may use the uppercase $P, Q, R, ...$, and when we want to talk about some constant or another, we may use the lowercase $a, b, c, ...$.

For example, we may represent a set of atomic formulas $P(c_1, c_2, c_3)$, meaning all formulas such that they start with $\mathfrak{P}_n^3$ for some n, and continue with the required three (not necessarily distinct) constants $\mathfrak{c}_k$, $\mathfrak{c}_j$, $\mathfrak{c}_l$. That is, any formula of form: $\mathfrak{P}_n^3(\mathfrak{c}_k, \mathfrak{c}_j, \mathfrak{c}_l)$ for some n, k, j, l. We may also write something like $Q(c_1, c_1)$, which would correspond to the class of all formulas $\mathfrak{P}_n^2(\mathfrak{c}_k, \mathfrak{c}_k)$ for some n, k. Clearly, these specifications are painful, hence the simplification.

Though these variables are arbitrary, just like n, k, j, etc., you should make sure they don't clash in a given context. Thus, using something like $P(c_1, c_2) \wedge P(c_1, c_2, c_3)$ is bad, because the two P clearly denote two distinct classes of predicates, one 2-place, one 3-place.

1.7 From syntax to semantics

So far, we have been working with languages as a bunch of symbols of their alphabets put one after the other in various ways. This allowed us to specify, for a given language, which formulas belong to the language, and which formulas do not. However, something we have not done yet is specify the *meaning* of these formulas. Clearly, we use languages to convey ideas about various topics. We do this through well-formed formulas of the language at hand that have a specific meaning. This is true of formal langugages as it is true for natural ones like English.

Some of the languages above already came with some previously understood meaning. For example, for the formulas of $\mathcal{L}_{AE}$, the language of arithmetic expressions, we already know what they mean, at least intuitively, through our knowledge of mathematics. Similarly, for $\mathcal{L}_f$, we already knew what those formulas meant given our knowledge of how a computer works, and more specifically, how their file system is usually structured.

On the other hand, for langugages like $\mathcal{L}_0$, that you may not know, we specified which formulas belong to the langugage, and which formulas do not, but so far, we have not given them any meaning, which would tell us what ideas we can convey with these formulas. In the next part of the book (after a brief detour), this is precisely what we will be doing.

Technically, what we have been doing so far is specifying the *syntax* of our languages. Syntax is the way expressions are formulated in a language. In the next part, we will be dealing with the *semantics* of some languages, which is specifying the various ways in which these expressions have, or can be given, meaning.

However, before we begin talking about meaning, we have to learn a bit about *set theory*. In modern mathematics, set theory is a foundational discipline, since many different mathematical structures can be formulated in it. Indeed, our semantics will be formulated in set theory. Thus, we will go through some fundamental aspects of set theory, those we will need to continue on our journey through formal logic.

Chapter 2

Set theory

In the following, we will be going through some fundamental aspects of set theory, the mathematical study of sets. Modern set theory is a very abstract field, and to make matters more confusing, set theory is usually formulated using a formal language of logic. And in turn, formal languages of logic are specified, both syntactically and semantically, through the use of set theory. Thus, there is a chicken-or-the-egg scenario going on. To steer clear of these issues, we will be explaining set theory in plain English terms.

2.1 Sets and membership

Set theory is all about sets. For now, we can say that sets are some specific collections of things. These things can be of any sort, including sets themselves. Going back to file systems, in some important aspects, sets are like folders, which may contain other folders, which in turn may contain other folders, and so on. In fact, sets are denoted just as we denoted folders in our language $\mathcal{L}_f$; using curly braces. Thus, the following is a set:

$$\{\text{Peter}, \text{Leo}, \text{Taylor}, \text{Curtis}\}$$

Sets have *members*. The members of a set are those things that are in the set. So the members of the set $\{\text{Peter}, \text{Leo}, \text{Taylor}, \text{Curtis}\}$ are Peter, Leo, Taylor, and Curtis. Set membership is denoted by $\in$. So for example,

$$\text{Peter} \in \{\text{Peter}, \text{Leo}, \text{Taylor}, \text{Curtis}\}$$

We can denote sets by using capital letters. The first choice is usually S (you can guess why...). This makes it easier to talk about them. So for example:

$$S = \{\text{Peter}, \text{Leo}, \text{Taylor}, \text{Curtis}\}$$
$$\text{Peter} \in S$$

Similarly, we can denote something not being in the set by using $\notin$. So for example:

$$\text{Henry} \notin S$$

So far, this is rather simple. Now let's look at some complications.

Sets are *extensional* entities. This is a fancy word to say that all a set depends on is what its members are. So in a set, there is no ordering between its members, and it doesn't matter whether the set is represented as having some members twice (or however many times). Each member is only counted once, in whatever order. So for example:

$$\{\text{Peter}, \text{Leo}, \text{Taylor}, \text{Curtis}\} = \{\text{Peter}, \text{Peter}, \text{Leo}, \text{Taylor}, \text{Curtis}\}$$
$$\{\text{Peter}, \text{Leo}, \text{Taylor}, \text{Curtis}\} = \{\text{Curtis}, \text{Leo}, \text{Peter}, \text{Taylor}\}$$

Definition 2.1.1 (Set extensionality)**.** Two sets S, Q, are identical if they have exactly the same members. That is, for all x, $x \in S$ if, and only if, $x \in Q$. If S and Q are identical sets, we write $S = Q$.

Exercise 2.1.1. Determine whether the following sentences are true or false. In each case, explain your reasoning.

1. $a \in \{b, g, h, a\}$;
2. if $\{b, g, h, a\} = S$, then $b \in S$;
3. the set $\{b, b, b\}$ has exactly one member;
4. if $Q = \{3, 5, 7, 7\}$, then the sum of its members is 22;
5. if $S = \{a, h, q, r\}$, then Peter $\notin S$.

There is one set that is unlike others, denoted $\emptyset$. This is the empty set. The empty set is so-called because it is, you guessed it, empty. In other words, it has no members. So for any candidate member x, $x \notin \emptyset$.

Now sets can be members of other sets, and it is very important to be clear whether something is in a set, or it is in another set that is part of another set, and so on. Note:

since $\emptyset$ is a set, it can also be a member of other sets. Let's look at an example:

$$S = \{a, b, c, \{a, d\}, d, \emptyset\}$$
$$Q = \{\{a, b, c\}, \{d\}, d, \emptyset\}$$

Now S and Q are not the same set. They do share *some* elements, but they differ in others. Since they do not have the same elements, they are not the same set. In particular, $\emptyset \in S, Q$, and $d \in S, Q$. But note that $a, b, c \notin Q$. What *is* in Q is the *set* {a, b, c}. Conversely, $a, b, c \in S$, but not $\{a, b, c\}$. Similarly, $\{d\} \in Q$ but $\{d\} \notin S$, though again, $d \in S, Q$.

Exercise 2.1.2. Determine whether the following expressions are true or false. In each case, explain your reasoning.

1. $S = \{\emptyset\}$ has no members;
2. $\emptyset$ has exactly one member, $\emptyset$;
3. $S = \{a, b, \{b\}, \{\{a, b\}\}, b, \{\emptyset\}\}$ has exactly 5 members;
4. the sets $\emptyset$, $\{\emptyset\}$, and $\{\emptyset, \emptyset\}$ are pairwise distinct (that is, no two of them are the same set);
5. if $S = \{a, \{a\}, \{a, \{a\}\}\}$ and $Q = \{a, \{a, \{a\}\}\}$, then every member of Q is a member of S;
6. if $S = \{a, \{a\}, \{a, \{a\}\}\}$ and $Q = \{a, \{a, \{a\}\}\}$, then every member of S is a member of Q.

2.2 Sets and subsets

Now that we know what sets are, how they are represented, and which sets are the same, we can talk about another important relationship between them. That is, one set being the *subset* of another set. If we have a set, say S, and a set, say Q, then S is a subset of Q if every member of S is also a member of Q. So for example, if $S = \{a, b, c\}$ and $Q = \{a, b, c, d\}$, then S is a subset of Q.

Definition 2.2.1 (Subset)**.** If S and Q are sets, and every member of S is also a member of Q, then S is a *subset* of Q (Q is a *superset* of S). In such cases, we write $S \subseteq Q$ (or $Q \supseteq S$).

One thing that is important to note with the subset relation is that it does not exclude the possibility that two sets are the same. Indeed, there is a general fact concerning this matter. Namely, if S and Q are sets, and $S \subseteq Q$ and $Q \subseteq S$, then $S = Q$. In other words, if S is a subset of Q, and Q is a subset of S, then S and Q are identical.

Exercise 2.2.1. Explain why it is the case that if S and Q are sets, and $S \subseteq Q$ and $Q \subseteq S$, then $S = Q$.

Now if we wanted to specify explicitly that one set is a subset *and* not equal to another set, we can use the symbol $\subset$, which stands for 'proper subset'. Proper subsets are just like subsets, except they come with the additional caveat that the two sets are not the same. In other words, $S \subset Q$ is just a short way to say that $S \subseteq Q$ and $S \neq Q$ (it is not the case that $S = Q$).

One thing that usually trips up people new to set theory (and sometimes, even those who aren't) is differentiating between $\in$ and $\subseteq$, that is, differentiating between one set being a member of another set, and one set being a subset of another set. It's important to make sure you can distinguish between the two!

Exercise 2.2.2. Determine whether the following expressions are true or false. In each case, explain your reasoning.

1. if $S = \{1, 7, 3\}$ and $Q = \{1, 3\}$, then $Q \subseteq S$;
2. if $S = \{1, 7, 3\}$ and $Q = \{7, 1, 3\}$, then $S \subseteq Q$ and $Q \subseteq S$;
3. if $S = \{a, b\}$, and $Q = \{\{a, b\}\}$, then $S \in Q$;
4. if $S = \{a, b\}$, and $Q = \{\{a, b\}\}$, then $S = Q$;
5. if $S = \{a, b, \{a, b, c\}\}$, and $Q = \{a, b, c\}$, then $Q \subseteq S$;
6. if $S = \{h, f, \{g, \{f\}\}\}$ and $Q = \{g, \{f\}\}$, then $Q \subseteq S$.
7. if $S = \{a, d, h\}$, then $S \subseteq S$;
8. if $S = \{d, b, c\}$, then $S \subset S$.

Finally, we must mention the case of $\emptyset$. Though it may sound a bit strange at first, $\emptyset$ is a subset of *every* set, including itself! Why? Because all its members are members of every other set. Why? Because it has none! Thus, in general, for every set S, $\emptyset \subseteq S$. On the other hand, it is *not* the case that every set has $\emptyset$ as its member. Some sets may, some sets may not. So for example, if we take $S = \{\{\emptyset\}\}$, $\emptyset$ is not a member of S, but $\emptyset \subseteq S$. Moreover, $Q = \{\emptyset\}$ is a member of S, and $\emptyset \subseteq Q$ *and* $\emptyset \in Q$.

2.3 Sets and properties

Since our semantics will be formulated in set theory, sets will play a central role in it. One of these roles will be to represent *properties*. Some properties are 'is red', 'is a mammal', 'is 6 feet tall', 'is the friend of Honghui', and so on. These properties are either 'had' by certain things or not. The technical term is 'exemplify'. So for example, every human has or exemplifies the property 'being a mammal', since every human is a mammal, but not every human is exactly 6 feet tall, so some have or exemplify the property 'is 6 feet tall', and some do not.

The simplest way to formally represent these properties is just to have a set of all those things that exemplify that property. So for example, one may form the set H of all (and only) those things that are human (exemplify 'is a human'), and thus the set H will represent the *property* 'is a human'. Similarly, one can represent the property 'is a mammal' by a set M consisting of all (and only) the things that are mammals.

Now let's look at some examples. Taylor Swift will be in the set H and the set M, since she is a human and a mammal (exemplifies the property 'is a human' and 'is a mammal'). However, if S represents the property 'is 6 feet tall', she will not be in S, since she is not 6 feet tall (apparently, she's 5'11"). The same goes for Jay-Z, since he is also a human and a mammal, and also not 6 feet tall (apparently, he is 6'2"). On the other hand, if we take Jay-Z's Corvette C-1, it is neither in H, nor M, nor S, since it is a car that does not have any of these properties.

You can also look at which properties (understood as sets) are subsets of which other properties (understood as sets). For example, since every human is a mammal, the property 'is a human' is a subset of the property 'is a mammal', since every member of H is a member of M. Indeed, H is a proper subset of M, since there are many mammals that aren't human. In other words, there are many animals in M not in H.

☞ There is a famous philosophical problem when it comes to identifying properties with sets. Namely, there are some properties that we would say are distinct, though they apply to exactly the same things, and hence they would be identified with the same set. For example, the property 'is the the first rapper to be inducted into the Songwriters Hall of Fame' and the property 'is the first solo living rapper inducted in the Rock and Roll Hall of Fame' seem to be different properties, yet if we identify properties with sets, they are the *same* set {Jay-Z}, so they are the same property. The wrong result! However, for our purposes (which is doing logic), it is okay to identify properties with sets.

We will return to the topic of properties and semantics in the next chapter.

2.4 Ordered sets and Cartesian products

Antoher thing that is especially important in the set theoretic foundations of logic is the notion of an ordered set. As mentioned above, sets are, by nature, unordered. Again, if $S = \{a, b\}$ and $Q = \{b, a\}$, then $S = Q$. But sometimes, we *do* want to talk about ordered sets, where the ordering of the members does matter. Ordered sets to the rescue!

Unlike normal sets, which are enclosed by curly braces, ordered sets are denoted with the angled braces $\langle$ and $\rangle$. For example, if we take the ordered set of a, b, and c in just this order, we can write $\langle a, b, c\rangle$. The ordered set $\langle c, b, a\rangle$ is distinct from this, since it has its order reversed.

The fact that they are ordered is not the only difference between normal sets and ordered sets. Another important difference is that for ordered sets, duplicates do count for something. Namely, the ordered set $\langle a\rangle$ and the ordered set $\langle a, a\rangle$ are not the same ordered set (and in general, not the same set either). So in general, unlike normal sets, ordered sets can be distinguished at face value.

Sometimes, some ordered sets are called ordered n-tuples, where n is the number of members of the ordered set. For small n, we have specific words for these tuples, but after a while, they become unwieldy, and aren't generally used. So for example, $\langle a, b\rangle$ is an ordered *pair*, $\langle a, b, c\rangle$ is an ordered *triple*, $\langle a, b, c, d\rangle$ is an ordered *quadruple*, and so on.

Another important notion in set theory is the Cartesian product of two sets. If S and Q are two sets, their Cartesian product is denoted by $S \times Q$. Indeed, Cartesian products are one way in which one may 'make' ordered sets out of regular sets. In particular, if S and Q are sets, then $S \times Q$, their Cartesian product, is the set of all ordered pairs such that their first member is in S, and their second member is in Q.

For example, suppose $S = \{a, b\}$ and $Q = \{1, 2\}$. Then, $S\times Q = \{\langle a, 1\rangle, \langle a, 2\rangle, \langle b, 1\rangle, \langle b, 2\rangle\}$. Again, it is just the *set* of all ordered pairs with the first member of the pair in S, and the second in Q.

Now importantly, S and Q need not be different, so that $S \times S$ is itself a set, the set of all ordered pairs made up of members of S. So for example, if $S = \{a, b, c\}$, then $S \times S = \{\langle a, b\rangle, \langle a, c\rangle, \langle b, a\rangle, \langle b, c\rangle, \langle c, a\rangle, \langle c, b\rangle, \langle a, a\rangle, \langle b, b\rangle, \langle c, c\rangle\}$. From this, you may see why taking the Cartesian product of two sets is denoted by the $\times$ ('product') operator, since if a set S has 3 members, then $S \times S$ will have 9 members, and so on. We can also write this more succinctly as S^2, and S^3, etc.

☞ The name 'Cartesian product' comes from an interesting historical fact. 'Cartesian' refers to René Descartes, one of the most important philosophers in history, who was also a pioneering mathematician. As such, he devised the coordinate system you probably already know from mathematics. What does this have to do with Cartesian products? For example, think of the Cartesian product of $\mathbb{N} \times \mathbb{N}$. Its members are all pairs of natural number $\langle n, k \rangle$. Like $\langle 3, 4 \rangle$, $\langle 1, 1 \rangle$, $\langle 6, 0 \rangle$. But of course, these are exactly the coordinates of a two-dimensional coordinate system on the natural numbers.

Exercise 2.4.1. Answer the following questions. For each question, don't forget to explain your reasoning.

1. What is the Cartesian product of the set $S = \{Lebron, Taylor\}$ with itself?
2. Is the set $Q = \{\langle Taylor, Lebron \rangle, \langle Lebron, Leo \rangle\}$ a subset of $S \times S$?

☞ You may be wondering what the relationship is between sets and ordered sets. In set theory books, it is customary to show how ordered sets are definable from the notion of a set. In particular, one can define the ordered pair $\langle a, b \rangle$ to be the set $\{\{a\}, \{a, b\}\}$. This is a kind of encoding, which allows one to determine the order of the two elements a and b from the set of two sets that are themselves not ordered in any way. This kind of definition can be further generalized to cover any n-tuple (for any n), by iterating the ordered pairs. For example, an ordered triple would be the ordered pair of an ordered pair and a third thing. That is, $\langle a, b, c \rangle = \langle \langle a, b \rangle, c \rangle$. It is then possible to further analyze $\langle \langle a, b \rangle, c \rangle$ according to our set-based definition of ordered pairs. There are alternative ways to do essentially the same thing.

2.5 Ordered sets and relations

The reason why ordered pairs, and ordered sets in general, are useful, is because they allow us to encode more structured information than just normal, unordered sets. For example, suppose we want to represent the fact that for three people, Jamal, Kerry, and Zoltan, some are friends with one another, and some aren't. For each person, we can create the set of friends of that particular person with normal sets.

Suppose the following. Jamal is friends with Kerry, Zoltan is friends with Jamal, but Kerry is not friends with Zoltan. For each person, we may specify the set of friends they have, so that $S_J = \{\text{Kerry}, \text{Zoltan}\}$, $S_K = \{\text{Jamal}\}$ and $S_Z = \{\text{Jamal}\}$. These would be the properties 'is the friend of Jamal', 'is the friend of Kerry' and 'is the friend of Zoltan', respectively.

However, with ordered sets, we can do all of this in one set, representing the *relation* 'is friends with'. First, we can immediately specify:

$$F = \{\langle \text{Jamal}, \text{Kerry} \rangle, \langle \text{Kerry}, \text{Jamal} \rangle, \{\text{Zoltan}, \text{Jamal}\}, \langle \text{Jamal}, \text{Zoltan} \rangle\}$$

Note that we included, for both Jamal and Kerry, and Jamal and Zoltan, the two names in both configurations. This is useful, since we may desire to add lopsided relations that are not mutual. For example, suppose we want to represent that Jamal is friends with Kerry, and Kerry is friends with Jamal, Zoltan is friends with Jamal and Jamal is friends with Zoltan, and also that Kerry thinks Zoltan is her friend, but Zoltan does not view Kerry as a friend. Then, we can encode this as:

$$F' = \{\langle \text{Jamal}, \text{Kerry} \rangle, \langle \text{Kerry}, \text{Jamal} \rangle, \{\text{Zoltan}, \text{Jamal}\}, \langle \text{Jamal}, \text{Zoltan} \rangle, \langle \text{Kerry}, \text{Zoltan} \rangle\}$$

Here, F' represents the 'is friends with' relation again (in a different situation). But using this blueprint, any relation can be represented, starting from such pairs to more elaborate ones, like 'x is in between y and z', which would be a set of triples, and so on.

Related to sets of ordered pairs are three important notions that are usually specified. First, a relation may be *reflexive*. This means that for any x (that occurs in the 'field' of a relation R, see below), $\langle x, x \rangle \in R$. For example, if L encodes the 'loves' relation, and everyone loves themselves, then for every person x, $\langle x, x \rangle \in L$. This way, L would be reflexive. On the other hand, if there is a person y such that $\langle y, y \rangle \notin L$, then L would not be reflexive, because there would be a person who does not love themselves.

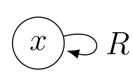


Figure 2.1: A graphical representation of an instance of reflexivity

Second, a relation may be symmetric, if whenever $\langle x, y \rangle \in R$, $\langle y, x \rangle \in R$ as well. This was already illustrated above with our three friends, Jamal, Kerry, and Zoltan. Specifically, F was a relation that was symmetric, since whenever x was a friend of y, y was a friend of x. On the other hand, F' was not symmetric, since there was a person, Kerry, who was friends with Zoltan, but Zoltan was not friends with Kerry.

Finally, a relation may be transitive. Transitive relations are everywhere in logic, though they are not always apparent. A transitive relation is more elaborate than the above two, since it is specified between three things. Namely, a transitive relation is such that if $\langle x, y \rangle \in R$, and $\langle y, z \rangle \in R$, then $\langle x, z \rangle \in R$. One relation that is usually transitive is airplane travel with connections. Suppose you can travel by plane from New York to Los Angeles, and from

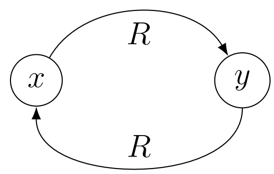


Figure 2.2: A graphical representation of an instance of symmetricity

Los Angeles to Tokyo. Then, that also means that you can travel from New York to Tokyo (with a connecting flight in Los Angeles). In other words if A represents the set of all pairs of cities reachable by air travel, if $\langle \mathrm{NYC}, \mathrm{LA} \rangle \in A$, and $\langle \mathrm{LA}, \mathrm{Tokyo} \rangle \in A$, then $\langle \mathrm{NYC}, \mathrm{LA} \rangle \in A$. Since this holds for every three cities, the relation A is transitive. Incidentally, this relation is also symmetric, since flights go back and forth (or perhaps in larger circles).

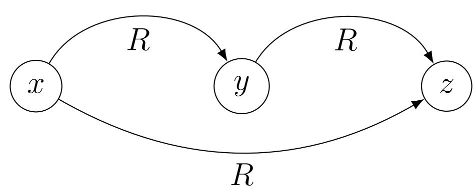


Figure 2.3: A graphical representation of an instance of transitivity

☞ If you have taken an introductory logic course before, you may remember the connective '$\rightarrow$', standing for 'if ..., then ...'. You may also remember that in your proof system, you could show that if you had as premises $X \rightarrow Y$ and $Y \rightarrow Z$, then you could derive $X \rightarrow Z$. Some systems even have a dedicated rule for this, called the 'hypothetical syllogism' or 'chain rule'. Now all this rule says is that $\rightarrow$ is transitive! In other words, if you take the set I of all pairs X, Y of propositions where $X \rightarrow Y$, and $\langle X, Y \rangle, \langle Y, Z \rangle \in I$, then $\langle X, Z \rangle \in I$ (at least in classical logic).

Note that relations can have many 'places', meaning they may be said to hold between several different things. A 2-place relation, as noted above, is a set of pairs, since it holds (or does not hold) between two things. But you may take other relations like 'x is in between, y and z', which would be a 3-place relation, and its representation would be a set of triples $\langle x, y, z \rangle$, because it holds (or does not hold) between 3 things. So in general, an n-place relation will be identified with a set of n-tuples, since it holds (or does not hold) between n different things. Notice that in each case, an n-place relation R is a subset of some set S taken n times with itself, i.e., $R \subseteq S^n$.

Definition 2.5.1. Let S be any set. An n-place *relation R defined on S* is a subset of the Cartesian product S^n, that is, the product of S taken n times with itself. In such cases, we call S the *field* of R.

If R is a two-place relation with field S, then:

1. if for all $x \in S$, $\langle x, x\rangle \in R$, then R is *reflexive*;

2. if for all $x, y \in S$, if $\langle x, y\rangle \in R$, then $\langle y, x\rangle \in R$, R is *symmetric*;

3. if for all $x, y, z \in S$, if $\langle x, y\rangle, \langle y, z\rangle \in R$, then $\langle x, z\rangle \in R$, R is *transitive.*

If the relation R is reflexive, symmetric, and transitive, we say it is an *equivalence relation.*

Exercise 2.5.1. Determine for each claim whether it is true or false. Don't forget to explain your reasoning, using the above definitions! Note: these are more philosophical questions regarding the exact meaning of some relations expressed in English. There may not be a single correct answer for each.

1. the relation 'x is related to y by blood' is transitive;

2. the relation 'x is an ancestor of y' is transitive;

3. the relation 'x is right of y' is transitive on a 1-dimensional plane;

4. the relation 'x is right of y' is transitive on a 2-dimensional plane;

5. the relation 'x is a brother of y' is symmetric;

6. the relation 'x is a sibling of y' is symmetric;

7. the identity relation '$=$' is reflexive, symmetric, and transitive.

2.6 Functions as sets of ordered pairs

Another thing that you will need to know as we move forward is a bit about *functions.* Functions may seem like mysterious entities, but set theoretically speaking, they are really quite simple. After all, what a function does is it takes an *input*, and provides a single *output.* For example, if we take the function $+2$, and apply it to the number 3, we get 5. There are various ways of specifying the $+2$ function, like $f(n) = n+2$, or $f : n \mapsto n+2$, but the only thing that is important is that there are a bunch of inputs, and for each input, there is just one output. So if f is the $+2$ function, then $f(3) = 5$.

> ☞ Perhaps somewhat confusing to the untrained eye is the fact that $f(3)$, as specified above is, by itself, the same as the number 5. So in this context, it makes sense to write things like $f(3) + 3 = 8$, or $f(3) \in \mathbb{N}$.

Functions can be represented in set theory by sets of ordered pairs. These ordered pairs essentially represent a list of all input-output pairs that a function is made up of. For example, if we take the +2 function again, defined on the natural numbers $\mathbb{N}$, set theoretically, it would be the set of all the following ordered pairs:

$$\begin{gathered}\langle 0,2\rangle \\ \langle 1,3\rangle \\ \langle 2,4\rangle \\ \langle 3,5\rangle \\ \langle 4,6\rangle \\ \vdots\end{gathered}$$

Or in one line, $\{\langle 0,2\rangle, \langle 1,3\rangle, \langle 2,4\rangle, \langle 3,5\rangle, \langle 4,6\rangle, ...\}$. Again, this is just a list of all possible input-output pairs for the function.

> ☞ Note the use of '...' in specifying the +2 function. Since the function +2 is defined on all natural numbers, naturally, its set-theoretic representation will be an infinitely large set! What the dots represent is that the list goes on forever, since for any natural number, you can always just add 2 and get another one.

One thing that you need to keep in mind is that not any set of ordered pairs constitutes a function. In particular, functions are those sets of ordered pairs where for each input, there is only one output. One way to formulate this idea is to say that if there is an input to the function, and seemingly two different outputs, then those two seemingly different outputs must be the same. Hence:

Definition 2.6.1 (Function)**.** A *function* f is a set S of ordered pairs such that if $\langle a,b\rangle \in S$ and $\langle a,c\rangle \in S$, then $b = c$. The set A of all a such that $\langle a,b\rangle \in S$ is called the function's *domain*, while the set B of all such b is called its *range* or *image*. A function's codomain C (if specified) is a superset of its range.

We may write $f : A \to C$ to specify f to be a function with domain A and codomain C. If $\langle a,b\rangle \in S$, we may write, alternatively, that $f(a) = b$, or that $f : a \mapsto b$.

Exercise 2.6.1. Is every function a relation? Is every relation a function? In your answer, explain your reasoning.

The above definition ensures that whatever conforms to that specification is an actual function. On the other hand, one can introduce more restrictions on these sets, to define certain special classes of functions.

Let's start with some useful terminology. According to the above definition, there are no functions which are one-to-many, in the sense of assigning many outputs to one input. That would immediately ensure they are not a function.

What about many-to-one functions? Indeed, these are possible. Suppose we have a function that assigns to each person their age at the moment. At any one moment, this would give us a very large set of ordered pairs:

$$\begin{gathered}\langle \text{Dwayne Johnson}, 51\rangle \\ \langle \text{Dua Lipa}, 28\rangle \\ \langle \text{Curtis Jackson}, 48\rangle \\ \langle \text{Idris Elba}, 51\rangle \\ \langle \text{Christian Bale}, 50\rangle \\ \vdots\end{gathered}$$

Since there are many people on Earth, there will be many that share their age at any one moment (at least as things currently stand). Above, you can immediately see that Dwayne Johnson and Idris Elba are the same age at the moment of writing this book. Thus, the person-to-person's-age function is many-to-one. Again, note that it *is* a function, and thus not one-to-many, and this is by nature, since every person only has one age.

On the other hand, some functions are not many-to-one, but *one-to-one*. This means that for two distinct inputs, they never have the same output. Take the $+2$ function again. The $+2$ function on the natural numbers is one-to-one, since for each two distinct natural numbers n, k, $n + 2 \neq k + 2$. This is easy to see since if $n + 2 = k + 2$ were the case, then $n = k$ would also be the case, even though we specified those two numbers must be distinct. In other words, the same output can only be had if the input is really the same. One-to-one functions are also called *injective*.

At times, we may want to explicitly specify not only the *domain* on which the function is defined (like how the function $+2$ has as domain the natural numbers), but also what its *codomain* is, which is a (not necessarily proper) superset of its range. For example, we may say that $+2$ is a function from the domain $\mathbb{N}$ of natural numbers to the codomain $\mathbb{N}$ of natural numbers. This may be written $+2 : \mathbb{N} \to \mathbb{N}$. Note that this notation is different from the above $n \mapsto n + 2$ which specifies that for each n, n is 'mapped to' $n + 2$, i.e.,

$+2(n) = n + 2$.

In connection, one may find that some functions 'cover' their codomain, in the sense that for each member of a function's codomain, there is a certain input value which outputs just that member. A function that is *not* like this is the $+2$ function specified as $\mathbb{N} \to \mathbb{N}$, since neither 0 nor 1 is an output for any input value (since the input value would have to be either -2 or -1, which are *not* natural numbers). On the other hand, we may specify a function $\times 2 : \mathbb{N} \to \mathbb{N}_{\text{Even}}$ such that $n \mapsto 2 \times n$. This function covers its codomain, the even numbers, since for every even number k, there is a natural number n such that $n \times 2 = k$. Functions like $\times 2 : \mathbb{N} \to \mathbb{N}_{\text{Even}}$ are called *surjective* or *onto*. Note that whether a function f is surjective is relative to how its codomain is specified. Clearly, $\times 2^* : \mathbb{N} \to \mathbb{N}$ would *not* be surjective, since no odd number ever gets output.

Finally, some functions are both injective and surjective, or both one-to-one and onto. These functions are called *bijective* or are said to be a *one-to-one correspondence* (sometimes, 1-1 correspondence.) For example, it is easy to see that the function $\times 2 : \mathbb{N} \to \mathbb{N}_{\text{Even}}$ is bijective or a one-to-one correspondence. We already know that it is surjective. But it is also injective, since there are no two distinct natural numbers i, j such that $i \times 2 = n$ and $j \times 2 = n$ ($n \in \mathbb{N}_{\text{Even}}$). Thus, it is bijective.

Bijective functions are very important, since they essentially allow one to count the members of a set by pairing them with numbers. For example, if you have the set $\{a, b, c\}$, this set can be put into 1-1 correspondence with the set $\{1, 2, 3\}$ (in various ways), so it has three members. But notice that the set $\{a, b\}$ cannot be put into 1-1 correspondence with $\{1, 2, 3\}$. And the set $\{a, b, c, d\}$ cannot be put into 1-1 correspondence with $\{1, 2, 3\}$ either. Indeed, for any set $\{a_1, a_2, ..., a_n\}$, it can only be put into 1-1 correspondence with one initial segment of the (positive) natural numbers, $\{1, 2, ..., n\}$.

Note also that if 1-1 correspondence provides a definition of 'has the same number of elements as', then $\mathbb{N}$ and $\mathbb{N}_{\text{Even}}$ have the same number of elements by the bijective function $\times 2$ (infinitely many), even though $\mathbb{N}_{\text{Even}} \subset \mathbb{N}$. Weird!

Exercise 2.6.2. Answer the following questions. In each case, explain your reasoning.

1. No human has more than 1 million hairs on their body. There are more than 8 million residents of New York City. Must there be two New York residents with exactly the same number of hairs on their body? Explain why, or why not.

2. Let A be the set of all natural numbers n such that there is a New York City resident with exactly n hairs. Let B be the set of all New York City residents. Let S be the set of all *actual* pairs $\langle x, y \rangle$ where $x \in A$ and $y \in B$, representing "hair number - person with that number of hairs" pairs. Is S a function?

3. Let Q be like S except consisting of all pairs $\langle y, x\rangle$ provided $\langle x, y\rangle \in S$. What does Q represent intuitively? Is Q a function?

4. Is Q surjective and/or injective? Is Q a one-to-one correspondence?

Definition 2.6.2 (Properties of functions)**.**

- A function $f : S \to Q$ is **one-to-one** or **injective** provided there are no $x, y \in S$ such that $x \neq y$ but $f(x) = f(y)$.

- A function $f : S \to Q$ is **surjective** or **onto** provided for every $y \in Q$ there is an $x \in S$ such that $f(x) = y$.

- A function $f : S \to Q$ is **bijective** or a **one-to-one correspondence** provided it is both injective and surjective (one-to-one and onto).

Exercise 2.6.3. A 1-1 correspondence is defined as a function between a domain and a codomain that is both injective and surjective. We already noted that there can be no 1-1 correspondence between $\{a, b\}$ and $\{1, 2, 3\}$, or between $\{a, b, c, d\}$ and $\{1, 2, 3\}$ (or indeed, between $\{a, b\}$ and $\{a, b, c, d\}$). This means that a function between any of these two is either not injective, or not surjective.

1. Take the sets $A = \{a, b\}$ and $T = \{1, 2, 3\}$. Suppose f is any function from A to T. Can it be surjective? Can it be injective?

2. Now take the sets $B = \{a, b, c, d\}$ and $T = \{1, 2, 3\}$. Suppose g is any function from B to T. Can it be surjective? Can it be injective?

3. How do the answers change if we take f' to be from T to A, and g' to be from T to B?

Finally, here is a definition that will be useful later on.

Definition 2.6.3. Let f be any function with the domain of formulas of $\mathcal{L}_0$, and let $\{\text{True}, \text{False}\}$ be its condomain. Then, f is called a truth-function.

2.7 Set-builder notation and Russell's paradox

There is way of representing sets, both small and extremely large, using what is called 'set-builder notation'. Set-builder notation specifies sets according to some specifiable rule, which decides whether something is in a given set or not. An example is:

$$S = \{x \mid x \text{ is a sheep}\}$$

What is specified above is that the set S consists of all x provided x is a sheep. In other words, S is the set of all sheep. On the right side, after the symbol '|', the rule for belonging, or not belonging, to the set is given. Namely, for any object x whatsoever, check if x is a sheep. If it is, it belongs to the set. If not, it does not belong to the set.

Here is another example:

$$Even = \{x \mid \text{there is an } n \in \mathbb{N} \text{ such that } n \times 2 = x\}$$

Here, the test is a bit more involved. It says that something belongs to the set *Even* provided you can find a natural number, n, and multiplying by 2 you get x. If you think about it, only the even numbers will pass this test. First, every even number x is such that you can find another number, n, and $n \times 2 = x$. For example, 6 is an even number, and there is another number, 3, such that $3 \times 2 = 6$. On the other hand, something like 13 will not pass the test, since there is no natural number n such that $n \times 2 = 13$. This is because $13/2 = 6.5$, and 6.5 is not a natural number.

☞ Interestingly, the set *Even* is, of course, infinite, but what the set-builder notation allows us to do is specify it precisely, in a finite manner, without relying on murky notation like the dots '...' above, which only implies that we can 'go on' indefinitely with listing our input-output pairs.

In fact, you can consider more extreme circumstances, like sheep. Take, for example, the famous cloned sheep Dolly. Is there a natural number n such that $n \times 2 = Dolly$? Clearly, there is no such number, so Dolly is also not in *Even*.

Exercise 2.7.1. According to the above definition, is $0 \in Even$? If it is, give a definition of a set in set-builder notation in which 0 is not a member. If it is not, give a definition of a set in set-builder notation in which 0 is a member.

Russell's paradox

At the inception of modern set theory, it was thought that every specifiable collection of things constitutes a set. That is, given *any* rule R, the following always gives you a set:

$$S = \{x \mid x \text{ passes } R\}$$

However, it was quickly discovered that this will not do, since not just any specifiable collection of things is a set. How could this be? The reason for this is usually attributed to Bertrand Russell, a philosopher-logician-mathematician (yet another one!), and one of the most important intellectuals of the 20$^{\text{th}}$ century. Hence the name 'Russell's paradox'.

Let's start from the beginning. As we said, some sets can be members of other sets. For example, regarding the set $S = \{\{1, 4, 2\}, 3, 6\}$, the set $\{1, 4, 2\}$ is in S. But if anything specifiable is a set, we may get some pretty weird stuff. Consider a set, S, and ask whether S is in S. Can a set have *itself* as its own member? At first glance, there is no reason why not. We can specify:

$$S = \{x \mid x = S\}$$

In this case, S would have a single member, S itself!

We can generalize on this idea, specifiying a set as follows:

$$Q = \{x \mid x \in x\}$$

The set Q is then the set of all sets that contain themselves. So the set $S = \{x \mid x = S\}$ would be in Q. But the set $\{1, 4, 2\}$ would not be, since $\{1, 4, 2\} \notin \{1, 4, 2\}$. Indeed, we seem to also be able to specify the *set of all sets that do not contain themselves*, by specifying:

$$R = \{x \mid x \notin x\}$$

Regarding R, if $S = \{x \mid x = S\}$, $S \notin R$, since $S \in S$. On the other hand, since $\{1, 4, 2\} \notin \{1, 4, 2\}$, $\{1, 4, 2\} \in R$.

In general, Q sounds like a set of rather weird sets that contain themselves, while R seems like a set of perfectly normal sets that do not have as members themselves. But in fact, R is quite problematic!

For note that R is specified by being made up of all sets that do not contain themselves. And R is itself a set! So presumably, we can apply the rule, specified for R, and see whether R is in R, or R is not in R. Now the rule specified for R is that if $x \notin x$, for any x, then $x \in R$, otherwise, $x \notin R$. But now consider R itself!

Well, if $R \in R$, then it must have passed the rule for R, so $R \notin R$. And if $R \notin R$, then it passes the rule, so $R \in R$. But surely, this is absurd! As a general rule, we want to say that a set is either in a set, or it is not in a set, but not both or neither. So we have two choices: either $R \in R$, or $R \notin R$. In the first case, we immediately get that $R \in R$ and $R \notin R$. In the second case, we immediately get that $R \notin R$ and $R \in R$. So in either case, we are in an impossible situation where R is both in R, and not in R. A paradox.

☞ There is an well-known illustration of Russell's paradox, sometimes called the barber paradox. Suppose you are in a town that has a barber with a seemingly straightforward rule. The barber shaves those, and only those people who do not shave themselves. So if you shave yourself, the barber does not shave you! But if you do not shave yourself, the barber does shave you. But what about the barber himself? If he shaves himself, then given the fact that he is the barber that only shaves those who do not shave themselves, he does not shave himself. And if he does not shave himself, then given the fact that he only shaves those who do not shave themselves, he does shave himself! So either way, we have the same problem as before, that the barber both does and does not shave himself.

Exercise 2.7.2. Let $U = \{x \mid x \text{ is a set}\}$. How would you describe this set? Is $U \in U$? If $Q = \{x \mid x \in x\}$, is $U \in Q$? Conversely, is $Q \in U$?

Because of issues like Russell's paradox, modern set theory needed to look for firmer foundations than it was originally based on. In more modern formulations, like Zermelo–Fraenkel set theory (usually denoted ZF), sets cannot be members of themselves, hence Russell's paradox is avoided. We shall not go into these matters further.

Chapter 3

The semantics of zeroth-order logic

If you have already taken an introductory logic course, you may remember talk of entailment, and in connection, validity. In particular, taking an argument with some premises and a conclusion, validity was probably specified along the following lines: if the premises are true, the conclusion must also be true. Then, it was said, the premises 'entail' the conclusion.

But so far, we have only been dealing with formal languages as being made up of various sequences of meaningless symbols, according to various rules. Where does truth enter the picture?

The answer is: semantics. In some sense, syntax and semantics are two sides of the same coin. Syntax specifies the way primitive symbols may be combined to form more complex expressions. Semantics, on the other hand, specifies how the meaning of more complex expressions can be computed from the meaning of primitive symbols. As we shall see, the rules of computation for the 'values' of expressions will match the rules of formation of expressions in our syntax. The idea underlying this is called 'compositionality'. As simpler syntactic expressions compose more complex ones, so the meanings of these simpler syntactic expressions compose the meanings of more complex ones. Then, just as being a formula is precisely definable, meaning will be too.

Once we have a grasp on the rules of meaning for $\mathcal{L}_0$, we can start designating some formulas of the language as 'true', and some as 'false', based on their respective meaning (and some other stuff, which we shall get to in due course). We will also see later on how the truth values of various sets of formulas relate to the truth values of other sets of formulas. From this, it will take just one additional step to specify how sometimes, some premises being true *ensures* that the conclusion *must* also be true.

☞ From now on, until further notice, we will only be discussing the language $\mathcal{L}_0$, the language of zeroth-order logic, and simple fragments of it with only a few predicates and connectives. In the latter cases, you can always imagine that the rest of the language is also dealt with in some way or another similar to the ones presented. We shall also drop outer parentheses in formulas where it would be annoying to have them out.

As noted above, semantics proceeds along the lines of syntax. In the syntax of $\mathcal{L}_0$, we have two types of base symbols: constants and predicates. As we have seen, every *atomic* formula is made up of an n-place predicate, followed by n constant symbols, in brackets, separated by commas. So in order to give truth-values to our atomic formulas, we first have to specify the meaning of constants and predicates. Then, we shall be able to *compute* the truth-values of our atomic formulas from this. In turn, this will allow us to *compute* the truth-values of our more complex formulas.

3.1 Constants

Let's start with the simplest case; constants. Constants in our language $\mathcal{L}_0$ function like names. In particular, their meaning is just the thing they designate, or refer to.

Note that we are talking about two distinct 'planes' here. On the side of syntax, we have symbols without meaning. On the other side, we have things assigned to them. Giving a semantics to our language is then bridging a gap, assigning things to our constants. These things are not symbols, they are the things themselves. You may have heard of the phrase 'domain of discourse'. The domain of discourse, or simply domain, is just the *set* of things we may talk about using a language.

Exercise 3.1.1. How does this notion of a domain relate to the notions of domain and codomain regarding functions?

In particular:

Constants	$\mapsto$	Domain
constant	$\mapsto$	thing

You may remember the symbol $\mapsto$ from our discussion of set theory. It designates that a certain function outputs the right-hand side value given the value on the left-hand side. And indeed, giving meaning to our language is just specifying a function that, in part, assigns to every constant a thing (sometimes called an 'object') from our domain. Like this:

Constants	$\mapsto$	Domain
$\mathfrak{c}_1$	$\mapsto$	Robert J. Oppenheimer

According to the above specification, the constant $\mathfrak{c}_1$ designates Oppenheimer in our language. Note that while $\mathfrak{c}_1$ is a symbol of the language, Oppenheimer, the thing ('person') it designates is, again, the real thing. We can continue assigning things to our constants to our own delight. For example, we can specify:

Constants	$\mapsto$	Domain
$\mathfrak{c}_1$	$\mapsto$	Robert J. Oppenheimer
$\mathfrak{c}_2$	$\mapsto$	Taylor Swift
$\mathfrak{c}_3$	$\mapsto$	the number 5
$\mathfrak{c}_4$	$\mapsto$	World War II
	$\vdots$	

As you can see, there is no limit to what a constant can designate. Thing and object is meant here in a very loose sense. It can be a person, a physical object, an event, an idea, whatever you want.

Let's recap. We have constants, which are symbols of our language. We have objects, in a loose sense, which are members of the domain of discourse. And we have a function, which assigns to each constant a member of the domain of discourse, as illustrated in the table above. This function is usually called an *interpretation function*, since it interprets the uninterpreted symbols of a language. We can make this more precise as follows:

Definition 3.1.1. A domain (of discourse) is any set $\mathbf{D}$. An interpretation function for the constants of $\mathcal{L}_0$, denoted by $\mathsf{CONS}_{\mathcal{L}_0}$, (relative to $\mathbf{D}$) is a function $\mathbf{I} : \mathsf{CONS}_{\mathcal{L}_0} \to \mathbf{D}$. If c is a constant of $\mathcal{L}_0$, then $\mathbf{I}(c) \in \mathbf{D}$, and is what c *designates*, *denotes* or *refers to*. Alternatively, we may say $\mathbf{I}(c)$ is the *value* of the symbol c.

As you can see, interpretations are functions from the set of all constants to the domain. This means that to each constant, only one member of the domain corresponds. So unlike with real names like 'Peter', which may designate many different people, a constant of $\mathcal{L}_0$ designates only one. On the other hand, the function $\mathbf{I}$ need not be one-to-one. This means that some distinct constants may designate the same thing, just like 'Miley Stewart' and 'Hannah Montana' designate the same person (in the hit TV show *Hannah Montana*). It also doesn't need to be onto, so that some members of the domain $\mathbf{D}$ may go nameless. Like the lack of injectivity, this is also natural, since many things do not have names in the real world. Just think of your left sock that fell behind the machine at the laundry.

Having constants or names that refer to things is the first step towards giving meaning to our expressions, but it is not enough to get us to truth. Names, by themselves, are neither true nor false, they just refer. The other ingredient we need is giving meaning to our *predicates.*

3.2 Predicates

Predicates in logic are used to express *properties* of objects or *relations* between them. Again, properties and relations are meant here in a very loose sense, and their representation, in set theory, is very minimal in detail.

3.2.1 ... of 1-place

Suppose you want your 1-place predicate $\mathfrak{P}_1^1$ in $\mathcal{L}_0$ to express the property 'is a physicist'. We already introduced a domain of discourse, or domain, $\mathbf{D}$ about which are language should be about. So given our domain, how do we capture that $\mathfrak{P}_1^1$ should have the meaning 'is a physicist'? Well, we can specify a subset of the domain $\mathbf{D}$, let's call it $\mathbf{P}$, which consists of just the physicists in our domain. In set-builder notation, we can say:

$$\mathbf{P} = \{x \mid x \in \mathbf{D} \text{ and } x \text{ is a physicist}\}$$

$\mathbf{P}$ here is a subset of the domain $\mathbf{D}$, since by definition, every $x \in \mathbf{P}$ is also in $\mathbf{D}$. Moreover, it only includes those members of the domain that are physicists. That is, the set of physicists in the domain. This may be called the *property* 'is a physicist', as we noted in the last chapter. Then, we can use the interpretation function to connect our 1-place *predicate* to the *property* (subset of the domain).

Again, represented in a figure:

1-PLACE PREDICATES	$\mapsto$	SUBSETS OF DOMAIN (PROPERTIES)
1-place predicate	$\mapsto$	set of things

And in particular:

1-PLACE PREDICATES	$\mapsto$	SUBSETS OF DOMAIN (PROPERTIES)
$\mathfrak{P}_1^1$	$\mapsto$	$\mathbf{P}$ ('is a physicist')

Again, the meaning of our predicates can be anything, as long as it is a property in the domain, that is, a subset of things of the domain. For example, it can be the set of things (of the domain $\mathbf{D}$) that are singers (the property of being a singer), the set of things that are numbers (the property of being a number), the set of things that are world wars (the property of being a world war), and so on. You can even have properties that only have one member, like 'is the first female artist with four Top 10 albums at once'.

1-PLACE PREDICATES	$\mapsto$	SUBSETS OF DOMAIN (PROPERTIES)
$\mathfrak{P}^1_1$	$\mapsto$	**P** ('is a physicist')
$\mathfrak{P}^1_2$	$\mapsto$	**S** ('is a singer')
$\mathfrak{P}^1_3$	$\mapsto$	**N** ('is a number')
$\mathfrak{P}^1_4$	$\mapsto$	**W** ('is a world war')
	$\vdots$	

3.2.2 ... of 2-places

The above approach takes care of our 1-place predicates. But predicates can come with more places (the superscript for $\mathfrak{P}$). Suppose you want to assign meaning to a 2-place predicate $\mathfrak{P}^2_1$, and in particular, you want it to mean 'x loves y'. Here, a set will not do, since we want to capture a *relation*, not a *property*. In particular we want to capture that a person is in the relation of loving another person (or thing in general).

If you think back to our discussion of set theory, you already know how to do this. Instead of a set of objects, you can take a *set of pairs* here, that represents two objects of the domain standing in the loving relation. Once again, we may introduce a relation **L** on the domain, and specify it as such:

$$\mathbf{L} = \{\langle x, y\rangle \mid x, y \in \mathbf{D} \text{ and } x \text{ loves } y\}$$

So **L** here is the set of pairs such that the first member of each pair loves the second member of that pair. So if our domain includes Julie and Jane, and Julie loves Jane, but Jane does not love Julie, we would have that $\langle Julie, Jane\rangle \in \mathbf{L}$, but $\langle Jane, Julie\rangle \notin \mathbf{L}$. So again:

2-PLACE PREDICATES	$\mapsto$	SETS OF PAIRS OF DOMAIN (2-PLACE RELATIONS)
2-place predicate	$\mapsto$	pairs of things

And in particular:

2-PLACE PREDICATES	$\mapsto$	SETS OF PAIRS OF DOMAIN (2-PLACE RELATIONS)
$\mathfrak{P}^2_1$	$\mapsto$	**L** ('loves')

Again, you can introduce whatever relation you want here, as long as it can be represented by a set of pairs of members of the domain. For example, 'is the favorite number of', 'is a sibling of', 'stands 2 feet to the right of', and so on. That is:

2-PLACE PREDICATES	$\mapsto$	SETS OF PAIRS OF DOMAIN (2-PLACE RELATIONS)
$\mathfrak{P}^2_1$	$\mapsto$	**L** ('loves')
$\mathfrak{P}^2_2$	$\mapsto$	**F** ('is the favorite number of')
$\mathfrak{P}^2_3$	$\mapsto$	**B** ('is a sibling of')
$\mathfrak{P}^2_4$	$\mapsto$	**R** ('stands 2 feet to the right of')

In each case, **L**, **F**, **B**, **R** are just sets of pairs representing all pairs of members of the domain that are in the specified relation.

Notice that each of these binary relations have, either on the left or the right side, a member of **D**, the domain. By the Cartesian product of **D** with itself once, i.e., $\mathbf{D} \times \mathbf{D}$ or $\mathbf{D}^2$, we can get the set of *all* pairs of members of **D**. Now relations on **D** will be subsets of $\mathbf{D}^2$, since each will be either the universal relation on **D**, the empty set, or somewhere in between. In the above example, $\langle Julie, Jane\rangle \in \mathbf{L}$, but $\langle Jane, Julie\rangle \notin \mathbf{L}$, so **L** is a non-empty proper subset of $\mathbf{D}^2$.

☞ Note that it is very important to be clear about the directionality of a relation. For example, we may have a predicate with assigned meaning 'loves'. But we may also have a predicate with assigned meaning 'is loved by'. Now, if the relation **L** is the relation 'loves', and $\mathbf{L}'$ is the relation 'is loved by', then each pair will be reversed relative to the other one. For example, if $\langle Julie, Jane\rangle \in \mathbf{L}$, but $\langle Jane, Julie\rangle \notin \mathbf{L}$, then $\langle Julie, Jane\rangle \notin \mathbf{L}'$, but $\langle Jane, Julie\rangle \in \mathbf{L}$, since x loves y if, and only if, y is loved by x. That is, if Julie loves Jane but Jane does not love Julie, then Jane is loved by Julie but Julie is not loved by Jane.

3.2.3 ... of n-places

You may see a pattern here. Predicates of 1-place (unary predicates) were interpreted as sets. Predicates of 2-places (binary predicates) were interpreted as 2-place relations. But of course, our language has predicates of every arity (every number of 'place'), and to each, we may want to attribute some meaning. Well, this is not hard to do, since for any n-place predicate, we can assign an n-place relation. The important thing is just that if a predicate is of form $\mathfrak{P}^n_k$, then its meaning must agree with n, so it has to be a set of n-tuples.

Exercise 3.2.1. Give a natural example of a 3-place, 4-place, and 5-place relation.

Following our handy figure, we have:

n-PLACE PREDICATES	$\mapsto$	SETS OF n-TUPLES OF DOMAIN (n-PLACE RELATIONS)
n-place predicate	$\mapsto$	set of n-tuples (n-place relation)

Making it a bit more concrete, but still quite abstract, we have:

n-PLACE PREDICATES	$\mapsto$	SETS OF n-TUPLES OF DOMAIN (n-PLACE RELATIONS)
$\mathfrak{P}_1^1$	$\mapsto$	$\mathbf{R}_1 \subseteq \mathbf{D}$
	$\vdots$	
$\mathfrak{P}_1^2$	$\mapsto$	$\mathbf{R}_i \subseteq \mathbf{D}^2$
	$\vdots$	
$\mathfrak{P}_1^n$	$\mapsto$	$\mathbf{R}_k \subseteq \mathbf{D}^n$
	$\vdots$	

We can then extend our interpretation function $\mathbf{I}$ to cover now not only constants, but predicates as well.

Definition 3.2.1. A domain (of discourse) is any set $\mathbf{D}$. An interpretation function for the predicates of $\mathcal{L}_0$, denoted by $\mathsf{PRED}_{\mathcal{L}_0}$, (relative to $\mathbf{D}$) is a function $\mathbf{I}$ such that for each predicate $\mathfrak{P}_k^n$, $\mathbf{I}(\mathfrak{P}_k^n) = \mathbf{R}$ for some $\mathbf{R} \subseteq \mathbf{D}^n$ (the Cartesian product of $\mathbf{D}$ taken n-times with itself).

In fact, we can put together our definition of an interpretation function for constants, and our definition of an interpretation function for predicates, into one definition. We can also introduce a new notion; *structure*. Structure is just a shorthand for what we have been saying over and over again; that when giving meaning to our expressions, we do it with an interpretation function $\mathbf{I}$ against the backdrop of a domain $\mathbf{D}$. So a structure $\mathbf{S}$ is just a pair $\langle \mathbf{D}, \mathbf{I} \rangle$ where $\mathbf{D}$ is the domain, and $\mathbf{I}$ is the interpretation function under consideration. With this in hand, we can say:

Definition 3.2.2. A structure $\mathbf{S}$ is a pair $\langle \mathbf{D}, \mathbf{I} \rangle$, where $\mathbf{D}$ is any set, and $\mathbf{I}$ is a function from the constants and predicates of $\mathcal{L}_0$ (i.e., $\mathsf{CON}_{\mathcal{L}_0} \cup \mathsf{PRED}_{\mathcal{L}_0}$) such that:

1. if c is any constant of $\mathcal{L}_0$, $\mathbf{I}(c) \in \mathbf{D}$, and;

2. if P^n is any predicate of arity n (n-place predicate) of $\mathcal{L}_0$, $\mathbf{I}(P^n) = \mathbf{R}$, where $\mathbf{R} \subseteq \mathbf{D}^n$.

As you can see, logicians can say a lot of stuff in very few words. This may seem intimidating at first. But remember that all these terse definitions hide quite intuitive ideas. We spent some time pondering these ideas so that you can read and understand the definition above, and the nuances and niceties it expresses so elegantly. This also gives you a very important skill: to go further. In more advanced logic textbooks, you won't find such long explanations as we have given. But now you won't need them either![1]

[1]Indeed, this is why they don't include them...

A brief return to our language specification

Indeed, now that we are familiar with a lot more machinery than before, we can give a definition of our language in a manner that is a lot more succint.

Definition 3.2.3. Let $\mathsf{ALPH}_{\mathcal{L}_0}$ be the alphabet of $\mathcal{L}_0$, specified as before, and thought of as forming a set. In particular, let $\mathsf{PRED}_{\mathcal{L}_0} \subseteq \mathsf{ALPH}_{\mathcal{L}_0}$ and $\mathsf{CONS}_{\mathcal{L}_0} \subseteq \mathsf{ALPH}_{\mathcal{L}_0}$, and such that:

1. $\mathsf{CONS}_{\mathcal{L}_0} = \{\mathfrak{c}_n \mid n \in \mathbb{N}\}$, and;
2. $\mathsf{PRED}_{\mathcal{L}_0} = \{\mathfrak{P}^n_k \mid n, k \in \mathbb{N}\}$.

The set of (well-formed) formulas of $\mathcal{L}_0$ is the smallest set $\mathsf{FORM}_{\mathcal{L}_0}$ such that:

1. if P is a predicate of arity n in $\mathsf{PRED}_{\mathcal{L}_0}$, and $c_1, c_2, ..., c_n$ are (not necessarily distinct) constants in $\mathsf{CONS}_{\mathcal{L}_0}$, then $P(c_1, ..., c_n) \in \mathsf{FORM}_{\mathcal{L}_0}$, and is an *atomic* formula;
2. if X and Y are in $\mathsf{FORM}_{\mathcal{L}_0}$, then:
   (a) $\neg X \in \mathsf{FORM}_{\mathcal{L}_0}$;
   (b) $(X \wedge Y) \in \mathsf{FORM}_{\mathcal{L}_0}$;
   (c) $(X \vee Y) \in \mathsf{FORM}_{\mathcal{L}_0}$; and
   (d) $(X \to Y) \in \mathsf{FORM}_{\mathcal{L}_0}$.

Again, a few weeks ago, this may have seemed extremely cryptic and impossible to comprehend, but now you are familar with all the different ideas underlying this definition, and can understand its intended meaning.

> ☞ In fact, when I was studying philosophy as an undergrad, I was reading papers from Russell, Quine, and others, and I really wanted to understand what they were saying. So I went and bought myself a book on semantics. I opened up the first page, and definitions not unlike 3.2.3 and 3.2.2 greeted me. Unlike you, I did not have a textbook like this, so I had no idea what was going on. Needless to say, my foray into semantics stopped right there, and did not continue for a few years.

3.3 Atomic formulas

Remember that we started our discussion in this chapter by setting our aim at assigning truth values to formulas. Once we have assigned meaning to our constants and predicates, we are in the position to do just that! Again, the basic idea underlying the mathematical machinery is not very difficult to grasp, but it is a very fundamental insight in several areas of thought, including philosophy, linguistics, and mathematics, and it was only precisely formulated around the middle of the 20^{th} century by the Polish logician Alfred Tarski.

We can illustrate this basic idea algorithmically, by looking at how one may go on calculating truth-values for atomic formulas, once a structure **S** is specified. Let's take some arbitrary constants from our language $\mathcal{L}_0$, using a, b, and c. Let's also take some arbitrary predicates of the language, using P, Q, and R. We can further specify that P is of arity 1, Q is of arity 2, and R is of arity 3.

Now let's take some rather arbitrary atomic formulas, let's say:

$$P(a) \tag{3.1}$$

$$P(c) \tag{3.2}$$

$$Q(a, c) \tag{3.3}$$

$$Q(c, a) \tag{3.4}$$

$$R(a, b, c) \tag{3.5}$$

$$R(a, c, b) \tag{3.6}$$

What if I ask you to decide whether these formulas are true or false? In that case, you should say: I cannot do that, since you haven't given me a domain **D** and an interpretation **I** that would tell me what these formulas mean, and against what I should evaluate them! Relative to different structures, different atomic formulas may be true or false, so there is no way to answer this question without first specifying a structure **S**. So let's do just that.

Take a domain $\mathbf{D} = \{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}\}$, and an interpretation function such that:

1. $I(a) = \mathbf{a}$, $I(b) = \mathbf{b}$, $I(c) = \mathbf{c}$;

2. (a) $I(P) = \{\mathbf{a}, \mathbf{d}\}$;
   (b) $I(Q) = \{\langle \mathbf{a}, \mathbf{c} \rangle, \langle \mathbf{d}, \mathbf{a} \rangle, \langle \mathbf{a}, \mathbf{d} \rangle\}$;
   (c) $I(R) = \{\langle \mathbf{a}, \mathbf{b}, \mathbf{c} \rangle, \langle \mathbf{b}, \mathbf{c}, \mathbf{b} \rangle, \langle \mathbf{b}, \mathbf{b}, \mathbf{b} \rangle\}$.

Call this the structure **S**. Now we can ask: which of the formulas are true *relative to* the structure **S**? To answer this question, again, we need to do some very simple calculations.

The basic idea is this. To each constant of the language, the interpretation function I assigns a member of the domain (see 1 above). And to each predicate (of arity n), the interpretation function assigns a set of n-tuples (see 2 above).

Next, all you have to do is check the *value* of a constant (or list of constants) under I, and compare it against the value of the relevant predicate under I. If the value of the constant (or sequence of constants) is in the value of the predicate under I, the atomic formula is true, otherwise, it is false.

Take the formula $P(a)$ (the formula (3.1) above). Is it true in $\mathbf{S}$? Well, we see that $I(a) = \mathbf{a}$. And we also see that $I(P) = \{\mathbf{a}, \mathbf{d}\}$. Since $I(a)$ (which is just $\mathbf{a}$) is *in* the set $I(P)$, the atomic formula $P(a)$ is true. On the other hand, if we take $P(c)$ (the formula (3.2) above), we see that $I(c) = \mathbf{c}$, and $\mathbf{c} \notin I(P)$, so $I(c) \notin I(P)$. So $P(c)$ is false in $\mathbf{S}$.

This may seem a bit elaborate, but really, all you are doing is checking if the value of the constant under I is in the set that is the value of the predicate P.

There is a special way of representing the relation 'the formula X is true in the structure $\mathbf{S}$'. It is written like this: $\mathbf{S} \models X$. You can also read it as: $\mathbf{S}$ *models* X. This is simply because $\mathbf{S}$ 'models' a world in which X would be true. So again, $\mathbf{S} \models P(a)$ ($\mathbf{S}$ models $P(a)$), but $\mathbf{S} \not\models P(c)$ ($\mathbf{S}$ does *not* model $P(c)$).

Predicates with more than 1 arity aren't more elaborate than this. The only difference is that now you have to check that the values of the constants under I are in the value of the predicate *as an n-tuple*, and of course, in the right order.

Take $Q(a, c)$ (formula (3.3) above). Again, $I(a) = \mathbf{a}$, and $I(c) = \mathbf{c}$, so the question is whether $\langle \mathbf{a}, \mathbf{c} \rangle \in I(Q)$. By definition, $I(Q) = \{\langle \mathbf{a}, \mathbf{c} \rangle, \langle \mathbf{d}, \mathbf{a} \rangle, \langle \mathbf{a}, \mathbf{d} \rangle\}$. So $\langle \mathbf{a}, \mathbf{c} \rangle \in I(Q)$. So $\mathbf{S} \models Q(a, c)$, or $Q(a, c)$ is true in $\mathbf{S}$. On the other hand, $Q(c, a)$ would make us consider wether $\langle \mathbf{c}, \mathbf{a} \rangle \in I(Q)$, which as you can see it is not. So $\mathbf{S} \not\models Q(c, a)$ (formula (3.4)), or $Q(c, a)$ is false in $\mathbf{S}$.

Exercise 3.3.1. Determine whether the formulas $R(a, b, c)$ (3.5) and $R(a, c, b)$ (3.6) are true in $\mathbf{S}$. In each case, give a proof just like the ones above for (3.1), (3.2), (3.3), and (3.4).

Exercise 3.3.2. One of the members of the domain does not have a name! That means that unfortunately, we cannot talk about it, even if the object itself shows up in our properties and relations. Which member is this?

Note that we can also take the reverse of this question. Namely, instead of considering whether a given formula is true in $\mathbf{S}$, we can start with $\mathbf{S}$ and consider which formulas are true in it. This is really just the reverse reasoning. For example, you can inspect $I(R)$, and see that $I(R) = \{\langle \mathbf{a}, \mathbf{b}, \mathbf{c} \rangle, \langle \mathbf{b}, \mathbf{c}, \mathbf{b} \rangle, \langle \mathbf{b}, \mathbf{b}, \mathbf{b} \rangle\}$. You can also see that $I(b) = \mathbf{b}$. Now, since $\langle \mathbf{b}, \mathbf{b}, \mathbf{b} \rangle \in I(R)$, that means $R(b, b, b)$ is true in $\mathbf{S}$, or $\mathbf{S} \models R(b, b, b)$.

Exercise 3.3.3. Find an atomic formula X distinct from all the above such that $\mathbf{S} \not\models X$, and an atomic formula Y distinct from all the above such that $\mathbf{S} \models Y$. Don't forget to give an appropriate proof as to why that is the case.

Note that the structure above was rather trivial and intuitive. But we can give a completely different structure $\mathbf{N}$ and ask again whether the above formulas are true or false in that structure. Let $\mathbf{N} = \langle \mathbb{N}^+, I' \rangle$. In other words, the structure S is such that its domain $\mathbf{D} = \mathbb{N}^+$, the set of all positive natural numbers.

We now specify a new interpretation I', that interprets our formulas in $\mathbf{N}$.

1. $I'(a) = 1$, $I'(b) = 2$, $I'(3) = c$;

2. (a) $I'(P) = \{n \mid n \text{ is even}\}$;

   (b) $I'(Q) = \{\langle n, k \rangle \mid n < k\}$;

   (c) $I'(R) = \{\langle n, k, i \rangle \mid n + k = i\}$.

In other words, under I', $P(x)$ means 'x is even', $Q(x, y)$ means 'x is less than y', and $R(x, y, z)$ means '$x + y = z$'. And a under I' means 1, b means 2, and so on.

Then, we can consider $P(a)$ and $P(c)$ again. Well, doing the calculations, neither 1 nor 3 are even numbers, so $P(a)$ and $P(c)$ are both false.

On the other hand, one of $Q(a, c)$ and $Q(c, a)$ must be true, since a and c mean distinct numbers. In particular, $Q(a, c)$ means 1 is smaller than 3, and $Q(c, a)$ means 3 is smaller than 1 under I'. So $\mathbf{N} \models Q(a, c)$, but not $Q(c, a)$.

And incidentally, $R(a, b, c)$ is true in $\mathbf{N}$, since $1 + 2 = 3$. But $1 + 3 \neq 2$, so $R(a, c, b)$ is false in $\mathbb{N}$.

Exercise 3.3.4. The above proofs are sketchy, referring to 'doing the calculations' and 'meanings'. Write them down in a precise manner, following the style of the proofs for $\mathbf{S}$.

Putting it all together

Now we are in the position to further extend our definition to include assigning truth values to atomic formulas. It goes like this:

Definition 3.3.1. A structure $\mathbf{S}$ is a pair $\langle \mathbf{D}, I \rangle$, where $\mathbf{D}$ is any set, and I is a function from the constants and predicates of $\mathcal{L}_0$ (i.e., $\mathsf{CON}_{\mathcal{L}_0} \cup \mathsf{PRED}_{\mathcal{L}_0}$) such that:

1. if c is any constant of $\mathcal{L}_0$, $I(c) \in \mathbf{D}$, and;

2. if P^n is any predicate of arity n (n-place predicate) of $\mathcal{L}_0$, $I(P^n) = \mathbf{R}$, where $\mathbf{R} \subseteq \mathbf{D}^n$.

For each atomic formula $P^n(c_1, ..., c_n)$ of the language $\mathcal{L}_0$, we have:

$$\mathbf{S} \models P(c_1, ..., c_n) \text{ if, and only if, } \langle I(c_1), ..., I(c_n) \rangle \in I(P^n).$$

If $\mathbf{S} \models P(c_1, ..., c_n)$, we say $\mathbf{S}$ *models* $P(c_1, ..., c_n)$, or *is a model of* $P(c_1, ..., c_n)$. Alternatively, we say $P(c_1, ..., c_n)$ is *true in* $\mathbf{S}$.

3.4 Complex formulas

We are now in the position to give truth-values to our more complex formulas, based on the values of the atomic formulas. Again, everything is relativized to a certain structure, since that is how we can assign truth-values to the atomic ones.

In particular, this is done by invoking the concept of a truth-function. A truth-function is just a function that takes one or more truth-values, and outputs a single truth-value. The meaning of our connectives are truth-functions in this sense. They operate on the values of their constituent formulas, and output another truth-value.

There are several ways to formulate this idea. First, one may specify these truth-values in terms of truth (T) and falsity (F), and a function $v : \mathsf{FORM}_{\mathcal{L}_0} \to \{T, F\}$ for any formula X, Y, like this:

1. $v(\neg X) = T$ if, and only if, $v(X) = F$;
2. $v(X \wedge Y) = T$ if, and only if, $v(X) = T$ and $v(Y) = T$;
3. $v(X \vee Y) = T$ if, and only if, $v(X) = T$ or $v(Y) = T$ (or both);
4. $v(X \to Y) = T$ if, and only if, if $v(X) = T$, then $v(Y) = T$.

Now usually, these are represented in truth-tables, which are really just tables representing the above truth-function, v, by listing all its input-output pairs one by one. In particular:

X	Y	$X \wedge Y$
T	T	T
T	F	F
F	T	F
F	F	F

X	Y	$X \vee Y$
T	T	T
T	F	T
F	T	T
F	F	F

X	Y	$X \to Y$
T	T	T
T	F	F
F	T	T
F	F	T

X	$\neg X$
T	F
F	T

Figure 3.1: Truth tables

You should read these as follows: in any structure $\mathbf{S}$, if X is true in $\mathbf{S}$ and Y is true in $\mathbf{S}$, then $X \wedge Y$ is true in $\mathbf{S}$. If X is true in $\mathbf{S}$ and Y is false in $\mathbf{S}$, then $X \wedge Y$ is is false in $\mathbf{S}$, and so on, for each line of each truth-table.

There are other ways to specify these same truth-functions. One is to arithmetize the relation between the input(s) and the output using 1 for true and 0 for false. This looks nice for some connectives, but less nice for others. Thus:

1. $v'(\neg X) = 1 - v'(X)$;
2. $v'(X \wedge Y) = v'(X) \times v'(Y) = \text{MAX}(v'(X), v'(Y))$;
3. $v'(X \vee Y) = 1 - ((1 - v'(X)) \times (1 - v'(Y))) = \text{MIN}(v'(X), v'(Y))$;
4. $v'(X \to Y) = 1 - (v'(X) \times (1 - v'(Y)))$.

Remark 3.4.1. In the above, MIN means the function that outputs the minimum value of two values, and MAX is the function that outputs the maximum value of two values. In terms of this, the meaning of our connectives is simple, since the value of $\wedge$ is just the maximum function on 1 and 0, and the value of $\vee$ the minimum. However, as demonstrated, you do not need to invoke these functions to define v'.

We can check if the four tables in Figure 3.2 match our tables in Figure 3.1 by computing the right side with our equations. And indeed, they do match!

X	Y	$X \wedge Y$
1	1	$1 \times 1 = 1$
1	0	$1 \times 0 = 0$
0	1	$0 \times 1 = 0$
0	0	$0 \times 0 = 0$

X	Y	$X \vee Y$
1	1	$1 - ((1-1) \times (1-1)) = 1$
1	0	$1 - ((1-1) \times (1-0)) = 1$
0	1	$1 - ((1-0) \times (1-1)) = 1$
0	0	$1 - ((1-0) \times (1-0)) = 0$

X	Y	$X \to Y$
1	1	$1 - (1 \times (1-1)) = 1$
1	0	$1 - (1 \times (1-0)) = 0$
0	1	$1 - (0 \times (1-1)) = 1$
0	0	$1 - (0 \times (1-0)) = 1$

X	$\neg X$
1	$1 - 1 = 0$
0	$1 - 0 = 1$

Figure 3.2: Truth tables a different way

Finally, we can put the above ideas in line with our previous results, and specify it as such:

1. $\mathbf{S} \models \neg X$ if, and only if, $\mathbf{S} \not\models X$;
2. $\mathbf{S} \models (X \wedge Y)$ if, and only if, $\mathbf{S} \models X$ and $\mathbf{S} \models Y$;
3. $\mathbf{S} \models (X \vee Y)$ if, and only if, $\mathbf{S} \models X$ or $\mathbf{S} \models Y$ (or both);

4. $\mathbf{S} \models (X \to Y)$ if, and only if, if $\mathbf{S} \models X$, then $\mathbf{S} \models Y$.

Here is some helpful information as to how to read the above specifications. On the left hand side of the 'if, and only if' connective, you always find more complex formulas, while on the right hand side, you always find less complex formulas. Reading from left to right, you can determine what the truth-value of the less complex formula must be if the more complex formula is true. For example, X must be false in a structure if $\neg X$ is true in that structure. Reading from right to left, you can determine the value of the more complex formula if the conditions on the right side hold for less complex ones. So if X and Y are both true in $\mathbf{S}$, $X \wedge Y$ will also be true.

It is important that these specifications allow one to determine both truth *and* falsity for any complex formula depending on the truth *or* falsity of some simpler formulas, and vice versa, though this may not be apparent at first.

Suppose $\mathbf{S} \not\models X \vee Y$, i.e., it is false in $\mathbf{S}$. Now it is specified what simpler formulas need to be true in order for $X \vee Y$ to be true. So how do we determine what needs to happen for it to be false in $\mathbf{S}$? Well, all we have to do is negate the right hand side, and see what's left. In particular:

$$\mathbf{S} \not\models X \vee Y \text{ if, and only if, it is not the case that } [\mathbf{S} \models X \text{ or } \mathbf{S} \models Y \text{ (or both)}].$$

Now this states that $\mathbf{S} \not\models X \vee Y$ if neither X nor Y is true in $\mathbf{S}$, since all three other possibilities *would* make it true. And indeed, this is just what the truth-table says!

Similarly, but from the other way around, suppose $\mathbf{S} \not\models X$ and $\mathbf{S} \models Y$. Well, since $\mathbf{S} \models X \wedge Y$ if, and only if, $\mathbf{S} \models X$ and $\mathbf{S} \models Y$, we have:

$$\mathbf{S} \not\models X \wedge Y \text{ if, and only if, it is not the case that } [\mathbf{S} \models X \text{ and } \mathbf{S} \models Y].$$

If it is not the case that $[\mathbf{S} \models X$ and $\mathbf{S} \models Y]$, then either X or Y or both are false in $\mathbf{S}$. So in particular, if $\mathbf{S} \not\models X$ and $\mathbf{S} \models Y$, then $\mathbf{S} \not\models X \wedge Y$.

☞ One way to think about this is by simply noting that $\not\models$ is just the negation of $\models$. So if the definition specifies when something is modeled, then it will *not* be modeled precisely when that specification is *not* met. That is, once the left side of the biconditional is negated, the right side needs to be too. This is very much akin to multiplying by -1 in an equation. It is a very fundamental fact that goes for every biconditional; if you negate both sides of a true biconditional, you get a true biconditional.

From now on, we will be using this type of specification to talk about the values of complex formulas. At times, we may refer back to truth-tables as a helpful reference for those who already know how truth-tables work. Don't worry if you are not familar with truth-tables. They do not give any more information than what can already be found in our style of specification.

Computing with complex formulas

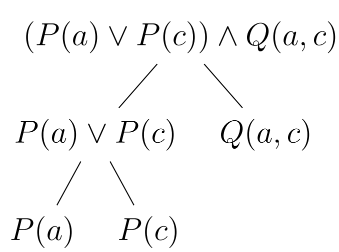
Let's turn back the wheel of time to the beginning of this book, where we specified the language of arithmetic expressions $\mathcal{L}_{AE}$. You may remember how we emphasized that the syntactic trees for those expressions give you an order of computation, based on how the parentheses are configured in the formula. If you forgot, go back to page 17 to refresh your memory. In fact, the situation is the same with the expressions of $\mathcal{L}_0$, except instead of $+$, $-$ and $\times$, we have our connectives, and instead of numbers, we have atomic formulas which themselves require computation. And formulas are either true or false. But it's the same.

Once again, take a structure $\mathbf{S} = \langle \mathbf{D}, I \rangle$ with the domain $\mathbf{D} = \{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}\}$, and an interpretation function I such that:

1. $I(a) = \mathbf{a}$, $I(b) = \mathbf{b}$, $I(c) = \mathbf{c}$;
2. (a) $I(P) = \{\mathbf{a}, \mathbf{d}\}$;
   (b) $I(Q) = \{\langle \mathbf{a}, \mathbf{c} \rangle, \langle \mathbf{d}, \mathbf{a} \rangle, \langle \mathbf{a}, \mathbf{d} \rangle\}$;
   (c) $I(R) = \{\langle \mathbf{a}, \mathbf{b}, \mathbf{c} \rangle, \langle \mathbf{b}, \mathbf{c}, \mathbf{b} \rangle, \langle \mathbf{b}, \mathbf{b}, \mathbf{b} \rangle\}$.

We already know a bunch about what truth-values various formulas get in the structure $\mathbf{S}$. Now, we can go on to talk about the values of more complex formulas, built out of our connectives.

Suppose you want to find out whether $(P(a) \vee P(c)) \wedge Q(a, c)$ is true in $\mathbf{S}$. Let's quickly draw up a syntax tree for this formula:



As with the example regarding $\mathcal{L}_{AE}$, we start from the leaves of our tree (remember that it is upside down). Now in general, you would first have to compute whether $P(a)$, $P(c)$, and $Q(a, c)$ are true in $\mathbf{S}$, but we will not revisit this issue, since this was already done above.

We had the following facts:

1. $\mathbf{S} \models P(a)$;
2. $\mathbf{S} \not\models P(c)$;
3. $\mathbf{S} \models Q(a, c)$.

We can substitute these values for the relevant formulas in our tree. Thus:

$$\begin{array}{ccc}
 & (P(a) \lor P(c)) \land Q(a,c) & \\
 & / \qquad \backslash & \\
 P(a) \lor P(c) & & T \\
 / \quad \backslash & & \\
 T \qquad F & &
\end{array}$$

Then, we need to proceed on the left side first, so we substitute the relevant values one level up:

$$\begin{array}{c}
(P(a) \lor P(c)) \land Q(a,c) \\
/ \qquad \backslash \\
T \lor F \qquad T
\end{array}$$

Now given how the connective $\lor$ was defined above, we can compute from $T \lor F$ the value T, since $\mathbf{S} \models X \lor Y$ if $\mathbf{S} \models X$, even if $\mathbf{S} \not\models Y$. So we have:

$$\begin{array}{c}
(P(a) \lor P(c)) \land Q(a,c) \\
/ \quad \backslash \\
T \quad T
\end{array}$$

Substituting again, we have:

$$T \land T$$

Invoking the rule for the connective $\land$, we get T as the final value, so

$$\mathbf{S} \models (P(a) \lor P(c)) \land Q(a, c).$$

☞ Note that this is a handy way to compute the values of formulas, but you should not confuse things like $T \lor F$ for formulas of the language, since they are not. In general, what we are doing in this computation is systematically replacing our formulas with their *values in* $\mathbf{S}$. In other words, it is just a compact representation of something more verbose (see immediately below).

There are several ways of going about computing the values of formulas. In presentation, it may look something like this:

> Take the formula $(P(a) \vee P(c)) \wedge Q(a, c)$. By our previous results, $\mathbf{S} \models P(a)$, $\mathbf{S} \not\models P(c)$, and $\mathbf{S} \models Q(a, c)$. By definition of $\vee$, $\mathbf{S} \models P(a) \vee P(c)$, since $\mathbf{S} \models P(a)$. Then, by definition of $\wedge$, $\mathbf{S} \models (P(a) \vee P(c)) \wedge Q(a, c)$, since $\mathbf{S} \models P(a) \vee P(c)$, and $\mathbf{S} \models Q(a, c)$. This is what we wanted to show.

In fact, if you really want to go crazy, you can use the equations above, and produce something like this:

$$(1 - ((1 - 1) \times (1 - 0))) \times 1 = 1$$

> Now $1 - 1 = 0$, $1 - 0 = 1$, so we get 0×1, which is 0, which we subtract from 1, which gets us 1, which we multiply by 1 to get 1. Or true.

This is not a thing we will be doing going further.

Exercise 3.4.1. Take the structure $\mathbf{S} = \langle \mathbf{D}, I \rangle$ exactly as specified above. Determine the truth-value of the following formula, giving a detailed explanation, *including* how the atomic formulas get their truth-value.

$$(Q(a, c) \vee \neg R(a, b, c)) \to \neg R(a, c, b)$$

Remark 3.4.2. In this exercise, you are essentially asked to combine two types of explanations, from the values of predicates and constants to the values of atomic formulas, and then from values of formulas to values of formulas. The latter was just demonstrated. For the former, some proofs were already provided, and some make up Exercise 3.3.1.

Computing from formulas up

We do not always have to consider how exactly the atomic formulas are specified in each case. Instead, we can talk more abstractly about assigning some truth-value to some specific formula X of the language, and some other truth-value to some specific formula Y of the language (and so on), and then see what truth-values one gets in a structure if we combine them in various ways. Indeed, doing this is equivalent to doing *propositional* or *sentential* logic, and specifically, drawing up more general truth-tables. Again, this you may already be familiar with.

For example, suppose X is any formula, and Y is any formula, and consider the formula $X \wedge \neg Y$. We do not know what these variables stand for (other than that they are formulas), so we cannot specify an exact structure for them, in which they may be true or false. But what we can do is *assume* that there is a certain structure in which, say X is true and Y is false, and then consider whether $X \wedge \neg Y$ is true or false *as a result* in a structure.

> ☞ Actually, what we are considering here is not just one structure but many; all those in which X is true and Y is false (for each possible choice of formula for the variables X and Y). But by abstracting away from the specifics, we do not need to worry about this.

Now the question is: is $X \wedge \neg Y$ true if X is true and Y is false in a structure? You might already see that in such a case, $X \wedge \neg Y$ is, in fact, true. Here is how it would go:

> Suppose there is a structure $\mathbf{S}$ such that $\mathbf{S} \models X$ but $\mathbf{S} \not\models Y$. By the above definition, $\mathbf{S} \models \neg Y$ provided $\mathbf{S} \not\models Y$, so $\mathbf{S} \models \neg Y$. Then, again, by definition, $\mathbf{S} \models X \wedge \neg Y$, provided $\mathbf{S} \models X$ and $\mathbf{S} \models \neg Y$, which we already know, so $\mathbf{S} \models X \wedge \neg Y$

This reasoning is the exact same as the following vertical fragment of a generalized truth-table:

X	Y	$\neg Y$	$X \wedge \neg Y$
T	F	T	T

Exercise 3.4.2. Determine the truth-value and construct detailed explanations (as opposed to truth-tables) for the following formulas, assuming $\mathbf{S} \models X$, $\mathbf{S} \not\models Y$, and $\mathbf{S} \models Z$:

1. $\neg(X \vee Y) \to Z$;

2. $(\neg X \wedge Y) \vee \neg Z$;

3. $Z \to (X \wedge \neg X)$;

4. $Y \to (Z \vee \neg Z)$;

5. $\neg(X \wedge Z) \to (\neg X \vee \neg Z)$;

6. $(Z \to Y) \to (\neg Z \vee Y)$.

Putting it all together

We are now in the position to formulate our whole semantics in one fell swoop. This is usually done in more advanced works at the very beginning (or not at all). You shall now be able to read those definitions with confidence.

Definition 3.4.1. A structure $\mathbf{S}$ is a pair $\langle \mathbf{D}, I \rangle$, where $\mathbf{D}$ is any set, and I is a function from the constants and predicates of $\mathcal{L}_0$ (i.e., $\mathsf{CON}_{\mathcal{L}_0} \cup \mathsf{PRED}_{\mathcal{L}_0}$) such that:

1. if c is any constant of $\mathcal{L}_0$, $I(c) \in \mathbf{D}$, and;
2. if P^n is any predicate of arity n of $\mathcal{L}_0$, $I(P^n) = \mathbf{R}$, where $\mathbf{R} \subseteq \mathbf{D}^n$.

For any formulas X, Y, Z of $\mathcal{L}_0$:

1. if X is of form $P^n(c_1, ..., c_n)$, $\mathbf{S} \models P^n(c_1, ..., c_n)$ if, and only if, $\langle c_1, ..., c_n \rangle \in I(P^n)$;
2. if X is of form $\neg Y$, $\mathbf{S} \models \neg Y$ if, and only if, $\mathbf{S} \not\models Y$;
3. if X is of form $Y \wedge Z$, $\mathbf{S} \models Y \wedge Z$ if, and only if, $\mathbf{S} \models Y$ and $\mathbf{S} \models Z$;
4. if X is of form $Y \vee Z$, $\mathbf{S} \models Y \vee Z$ if, and only if, $\mathbf{S} \models Z$ or $\mathbf{S} \models Y$ (or both);
5. if X is of form $Y \rightarrow Z$, if $\mathbf{S} \models Y$, then $\mathbf{S} \models Z$.

If $\mathbf{S} \models P(c_1, ..., c_n)$, we say $\mathbf{S}$ *models* $P(c_1, ..., c_n)$, or *is a model of* $P(c_1, ..., c_n)$. Alternatively, we say $P(c_1, ..., c_n)$ is *true in* $\mathbf{S}$.

Congratulations, this concludes the semantics of $\mathcal{L}_0$.

☞ From now on, we will omit writing out 'if, and only if' each and every time. Instead, we will opt for the abbreviation '*iff*' (in italics).

Chapter 4

Doing zeroth-order logic

We now have a cursory acquaintance with the semantics of $\mathcal{L}_0$, so we know how the meaning of formulas are specified, and in particular, how truth-values are assigned to formulas, both atomic and complex. Thus, we are in the position to start talking about *logic*. Now what logic is exactly is hard to say, since by now, the field is vast and intertwined with many other disciplines. At any rate, one of the fundamental aspects of logic is examining how the truth-values of certain sentences relate to the truth-values of certain other sentences. This sounds like a semantic question, since in this formulation, it is about truth-values. On the other hand, over the years, logicians have come up with a myriad of ways to examine these relations without talking about truth-values, or anything else in semantics. To do this, they introduced syntactic deductive systems, which rely purely on some syntactic rules of transformation to tell you whether certain sentences *entailed* other sentences, or in other words, whether those sentences were *logical consequences* of the initial set (among other things). Now since this book likes to be unorthodox, we will be using a deductive system that is technically syntactic in its nature, since it relies on purely syntactic rules for transforming the formulas of the language into other formulas of the language, but it is also highly tied to the semantics we previously examined. The approach is variously called *analytic tableau*, *semantic tableau*, or the *truth tree method*. In fact, in its modern form, it was invented by the logician-mathematician-philosopher-magician-pianist Raymond Smullyan, who was once a CUNY professor!

4.1 Some fundamental concepts of logic

If you encountered propositional logic before, you may know that there are three large classes of formulas one may distinguish. Usually, these are called *contingent*, *tautology*, and *contradiction*. In our semantics, all this means is the following:

Definition 4.1.1. A formula X of $\mathcal{L}_0$ is a *tautology iff* for *every* structure $\mathbf{S}$, $\mathbf{S} \models X$. A formula X is a *contradiction iff* there is *no* structure $\mathbf{S}$ such that $\mathbf{S} \models X$. Finally, a formula X is *contingent* if it is neither a tautology, nor a contradiction.

Remark 4.1.1. Since tautologies are true in every structure, and contradictions are true in no structure, contingent formulas are those that are true in some structures, false in others.

Now one thing to note at the beginning is that atomic formulas are all contingent. That is, for each atomic formula P, there is a structure $\mathbf{S}$ such that $\mathbf{S} \models P$, and there is a structure $\mathbf{S}'$ such that $\mathbf{S}' \not\models P$. On the other hand, some complex formulas are contingent, some are tautologies, and some are contradictions.

What makes a formula a tautology or a contradiction? It is the way the connectives work. Notice that by definition, each connective must obey some specific rules in any structure regarding the truth-values of the formulas on which it operates. For example, $X \wedge Y$ is true in any $\mathbf{S}$ *iff* X and Y are both true individually in $\mathbf{S}$. However, some formulas may say things that go against these rules. This means that they won't be true in any structure S, since again, every structure must abide by these rules, while the formula says otherwise.

Let's take the simplest example: $X \wedge \neg X$. In fact, this is not a formula of the language, but a *class* of formulas of form $X \wedge \neg X$. But as it turns out, no matter what formula you take for X, it will always be the case that for any structure $\mathbf{S}$, $\mathbf{S} \not\models X \wedge \neg X$. In other words, this formula schema cannot have a true instance.

Now again, the reason for this is the way structures are defined. In particular, in every structure $\mathbf{S}$, either $\mathbf{S} \models X$, or $\mathbf{S} \not\models X$. This is sometimes called the *Law of Excluded Middle*, and it falls out of our definition of a structure. What this means is that every formula X is either true in a structure, or false in a structure, and not a third thing (the 'excluded middle').

We also have that it is never the case that $\mathbf{S} \models X$ and $\mathbf{S} \not\models X$. This is sometimes called the *Law of Non-contradiction*. Again, this falls out of our definition of what it means for something to be a structure. Now adding these two laws together, you simply get that every formula X is either true or false, *and* not both *or* neither in a structure.

So take $X \wedge \neg X$. By the above, either $\mathbf{S} \models X$, or $\mathbf{S} \models \neg X$. Those are the only two options we have. We can consider each of them in turn:

1. Suppose $\mathbf{S} \models X$. Then, by definition of $\neg$, $\mathbf{S} \not\models X$. But $\mathbf{S} \models X \wedge \neg X$ *iff* $\mathbf{S} \models X$, *and* $\mathbf{S} \models \neg X$, by definition of $\wedge$. So since the latter half fails, $\mathbf{S} \not\models X \wedge \neg X$.

2. Suppose $\mathbf{S} \not\models X$. Then, by definition of $\neg$, $\mathbf{S} \models \neg X$ in this case. But again, $\mathbf{S} \models X \wedge \neg X$ *iff* $\mathbf{S} \models X$, *and* $\mathbf{S} \models \neg X$, by definition of $\wedge$. Moreover, in this case, we already assumed that $\mathbf{S} \not\models X$. So once again, $\mathbf{S} \not\models X \wedge \neg X$.

This means that there is no structure **S** in which $X \wedge \neg X$ is true. So every instance of $X \wedge \neg X$ is false in every structure, no matter what you take X to be.

At first, this type of talk might be confusing, but all we did was precisely represent what is usually illustrated by the following truth table:

X	$\neg X$	$X \wedge \neg X$
T	F	F
F	T	F

In particular, a formula is contradictory, as represented in a truth table, if all values in the column underneath the formula are false. Again, the truth table rows just represent two large (jointly exhaustive) classes of structures, one in which X is true, the other in which X is false, and then show that neither of these classes make $X \wedge \neg X$ true.

It will be important in what's to come that every contradictory formula hides a claim of the form $X \wedge \neg X$ for some formula X. In fact, every contradictory formula hides a claim of the form $P \wedge \neg P$, for some *atomic* formula P. In other words, the only contradictory formulas are those that try to say that something both is and is not the case (though not necessarily in exactly those words).

On the other hand, we may find formulas of the form $X \vee \neg X$, which are not contradictory, but the exact opposite, *tautological*. Again, the reason is because of how the connectives are defined in a structure. Again, in every structure, we have that either $\mathbf{S} \models X$, or $\mathbf{S} \not\models X$. So:

1. If $\mathbf{S} \models X$, then $\mathbf{S} \models X \vee \neg X$, since by definition of $\vee$, $S \models X \vee \neg X$ *iff* it models either X or $\neg X$.

2. If $\mathbf{S} \not\models X$, then by definition of $\neg$, $\mathbf{S} \models \neg X$, so by definition of $\vee$, $\mathbf{S} \models X \vee \neg X$ (because of the latter half of the disjunction now).

Thus, $X \vee \neg X$ must be true in *every* structure **S**, no matter which formula we take for the variable X. And again, this is just an explanation of the following truth-table:

X	$\neg X$	$X \vee \neg X$
T	F	T
F	T	T

As you can see, unlike with contradictions, tautologies have all values as true in the column underneath them.

Now we can formulate one of the most significant connections between tautologies and contradictions. As such, we will give it the proper form of a theorem.

Proposition 1. *A formula X of $\mathcal{L}_0$ is a tautology iff its negation $\neg X$ is a contradiction.*

Proof. From left to right, suppose X is a tautology. Then, for every $\mathbf{S}$, $\mathbf{S} \models X$. By definition, $\mathbf{S} \models X$ *iff* $\mathbf{S} \not\models \neg X$. So since *every* $\mathbf{S} \models X$, *no* $\mathbf{S} \models \neg X$. By definition, $\neg X$ is then a contradiction.

From right to left, suppose $\neg X$ is a contradiction. Then, there is no $\mathbf{S}$ such that $\mathbf{S} \models \neg X$. So *every* $\mathbf{S}$ is such that $\mathbf{S} \not\models \neg X$. But any $\mathbf{S} \not\models \neg X$ just in case $\mathbf{S} \models X$. So every $\mathbf{S} \models X$, and thus X is a tautology. □

Remark 4.1.2. Notice the form of this proof. Since we are proving a biconditional (an *iff* statement), we need to prove that the left hand side entails the right hand side, and the right hand side entails the left hand side. In order to prove that one side entails the other, we have to *assume* that one side holds, and *derive* that the other side then must also hold. Thus, we assumed the left hand side and derived the right, then assumed the right hand side, and derived the left.

As before, inverting this biconditional by negating both sides also gets us a true biconditional. Thus:

Proposition 2. *A formula X of $\mathcal{L}_0$ is* not *a tautology iff its negation $\neg X$ is* not *a contradiction.*

Here is an additional proposition that may help here:

Proposition 3. *A formula X of $\mathcal{L}_0$ is contingent iff $\neg X$ is contingent.*

Proof. From left to right, if X is contingent, it is true in some structure $\mathbf{S}$, and false in some structure $\mathbf{S}'$. Now if $\mathbf{S} \models X$, then $\mathbf{S} \not\models \neg X$, so $\neg X$ is false in $\mathbf{S}$. On the other hand, if $\mathbf{S}' \not\models X$, then $\mathbf{S}' \models \neg X$, so $\neg X$ is true in $\mathbf{S}$. So $\neg X$ is also contingent.

From right to left, the reasoning is almost exactly the same. □

Exercise 4.1.1. Finish the proof for Proposition 3.

Now given the above two propositions, we can see that if a formula X of $\mathcal{L}_0$ is *not* a tautology, then $\neg X$ is not a contradiction, because if X is not a tautology, it is either a contradiction or contingent, so $\neg X$ will either be a tautology (hence not a contradiction), or contingent (hence not a contradiction). And similarly the other way around.

In fact, the following also follows:

Proposition 4. *$\neg X$ is a tautology iff X is a contradiction.*

Proof. By Proposition 1, Y is a tautology *iff* $\neg Y$ is a contradiction. Here is the trick. Since Y can be *any formula*, we substitute $\neg X$ for Y. Thus, $\neg X$ is a tautology *iff* $\neg\neg X$ is a contradiction. But $\neg\neg X$ is a contradiction *iff* X is also a contradiction. This is because for any $\mathbf{S}$, if $\mathbf{S} \models \neg\neg X$, then $\mathbf{S} \not\models \neg X$, then $\mathbf{S} \models X$ (and similarly the other way around). So $\neg X$ is a tautology *iff* X is a contradiction. Which is what we wanted to show. □

Some of what the above propositions tell us is summarized in Table 4.1.

	if, and only if	
X is tautology	$\Leftrightarrow$	$\neg X$ is contradiction
X is contradiction	$\Leftrightarrow$	$\neg X$ is tautology
X is contingent	$\Leftrightarrow$	$\neg X$ is contingent

Table 4.1: Relations between tautologies, contradictions and contingencies

Here is another thing that will be relevant presently. Suppose you have a formula X, and it is *not* a tautology (so it is either a contradiction or contingent). Then, its negation $\neg X$ will *not* be a contradiction (it will either be a tautology or contingent). Thus, if X is not a tautology, there is at least one structure $\mathbf{S}$ such that $\mathbf{S} \models \neg X$, and *vice versa*. This is usually what is called a *counterexample*. In other words:

Proposition 5. *X is not a tautology iff there is a structure $\mathbf{S}$ such that $\mathbf{S} \models \neg X$.*

Exercise 4.1.2. Using the reasoning with *counterexamples* sketched above, show for each of the following formulas that they are *not* tautologies.

1. $(\neg Y \wedge \neg X) \wedge \neg X$
2. $(X \rightarrow Y) \rightarrow X$
3. $(X \rightarrow Y) \rightarrow Y$
4. $(X \vee Y) \rightarrow X$

4.2 The basic idea behind tableau systems

Here is the basic idea behind tableau systems. Imagine that your goal is to show that certain formulas of the language $\mathcal{L}_0$ are tautologies, or that in every structure $\mathbf{S}$, they are true. Now clearly, given a formula X, you cannot just consider every structure $\mathbf{S}$ individually to see whether that particular structure models X or not. That would take an infinite amount of time. What you *can* do is employ some of the proofs we have given above to show that, e.g.,

$X \vee \neg X$ is a tautology. The problem with this is that as the formulas get more complex, the proofs get longer and longer, to the point where it is just not feasible to carry them out in natural language. Moreover, these solutions are not very systematic as they stand. If you are ingenious, you may be able to do them (up to a point), but we don't want to rely on ingenuity to solve these problems. We need a method.

In the tableau method, the whole approach is based on the following facts, discussed above:

1. if X is a tautology, $\neg X$ is a contradiction;

2. if something is a contradiction, then it hides an explicit contradictory claim of form $Y \wedge \neg Y$ (and indeed, for some atomic formula P, $P \wedge \neg P$).

If we put together these two facts, we get a strategy. We take the candidate tautology X. If it *really* is a tautology, then its negation $\neg X$ is a contradiction. If it is a contradiction, then with the help of some appropriate rules, we can hopefully show in what way it is a contradictory statement. In other words, we show which is the formula Y it claims is both true and false.

On the other hand, if X is not a tautology, then it is either a contradiction or contingent, so its negation $\neg X$ will either be a tautology or contingent (as described above). In such cases, we will find no contradiction.

To continue our example, take $X \vee \neg X$ again. We already know this is a tautology, but what if we want to show in our tableau system that it is? Well, since we want to show $X \vee \neg X$, what we put at the beginning of our proof is *its negation* $\neg(X \vee \neg X)$. Then come the rules, which will tease out where exactly is the contradictory statement hidden in the formula.

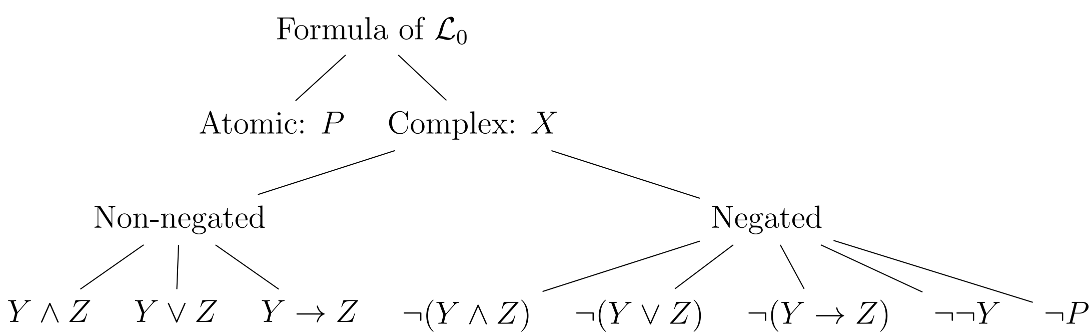
4.3 The rules of the system

Each rule in our tableau system codifies some simple facts about our semantics. This is a general feature of deductive systems in logic. What is less so is the fact that tableau systems are not linear, but take a tree form. In fact, our tableau deductions will look something like the syntax trees we covered before. However, despite the similarities, do not confuse the two. They are entirely different systems with a different purpose. Tableau tree deductions are used to show whether X is a *tautologous* formula, syntax tree derivations are used to show whether X is a formula at all.

Just as syntax trees, tableau trees are also *binary*. This means that at any one point, the tree may branch to at most two separate points. But sometimes, there is no branching, and only one additional point is connected. Because of this, we may classify our rules as those which branch, and those which do not.

The basic idea behind the rules is that given a complex formula X of the language assumed to be true, we can infer what other, less complex formulas must also be true as a consequence. For this, we need to know what formulas we may encounter in our language, and what we can do with them.

First, we have the basic case, where everything is either an atomic formula P, a conjunction $Y \wedge Z$, a disjunction $Y \vee Z$, a conditional $Y \to Z$, or a negation $\neg Y$. Now as it turns out, it is a good idea to take the negation of each of these formulas separately. So we have the negation of an atomic formula $\neg P$, the negation of a conjunction $\neg(Y \wedge Z)$, the negation of a disjunction $\neg(Y \vee Z)$, the negation of a conditional $\neg(Y \to Z)$, the negation of a negation $\neg\neg Y$, or the negation of an atomic formula $\neg P$. This covers every possible negated formula form, and the two sets cover every possible formula form. Here is a diagram of this:



Remark 4.3.1. It might seem like there is a problem here, since $\neg Y$ does not appear in the diagram, only $\neg\neg Y$. But this is not the case, since if we consider $\neg Y$, and what Y could be, it could either be a negated formula or a non-negated one. If it is negated, then it is a double negation, which is covered by $\neg\neg Y$. If it is a simple negation, then Y is either a complex or an atomic formula, which is also covered by the diagram. So there is nothing missing.

Now, as far as our rules are concerned, we will not have rules for P and $\neg P$, but we will have rules for every other form. The principles behind these rules will be familiar if you have ever taken a logic course before.

4.3.1 Non-branching rules

First, we will consider non-branching rules, since they are simpler. And we shall start with the simplest of them all.

Double negation

The double negation rule is extremely simple. If you have a formula of form $\neg\neg X$ occurring on a branch of the tree, you can extend that branch with X. In practice, this simply looks like this:

1.	$\neg\neg X$	
2.	X	$\neg\neg$ 1

Figure 4.1: Double negation rule

Why does this rule work? Simply because of the way structures are defined. In particular, if you have a structure $\mathbf{S}$, and $\mathbf{S} \models \neg\neg X$, then by definition of $\neg$, $\mathbf{S} \not\models \neg X$, and so by the same definition, $\mathbf{S} \models X$. This works the other way around too, but we are always moving from more complex to simpler formulas in tableau systems, so we do not need to consider the reverse.

Here is a representation of the above in a truth table:

X	$\neg X$	$\neg\neg X$
T	F	**T**
F	T	F

Conjunction

The conjunction rule is another one of these really simple rules. It says that if you have a formula of form $X \wedge Y$ on a branch, then you can extend that branch with either X or Y, i.e., with either the left or the right side. Like this:

1.	$X \wedge Y$			1.	$X \wedge Y$	
2.	X	$\wedge$ 1		2.	Y	$\wedge$ 1

Figure 4.2: Conjunction rule

Clearly, by two successive applications of the rule on $X \wedge Y$, you can also get the following:

1.	$X \wedge Y$	
2.	X	$\wedge$ 1
3.	Y	$\wedge$ 1

Now why does this rule work? Again, it is a simple matter of checking that whenever $\mathbf{S} \models X \wedge Y$, $\mathbf{S} \models X$, and $\mathbf{S} \models Y$. In fact, this is literally just the definition of how $\wedge$ behaves in any structure, so we don't have to show anything. Of course, we can represent this in a truth table, the exact one that was given above for $\wedge$:

X	Y	$X \wedge Y$
T	**T**	**T**
T	F	F
F	T	F
F	F	F

Negated disjunction

The next type of non-branching rule that we cover is a bit more elaborate than the previous ones. It is of the following form, similar to the simple $\wedge$ rule above.

1.	$\neg(X \vee Y)$		1.	$\neg(X \vee Y)$	
2.	$\neg X$	$\neg\vee$ 1	2.	$\neg Y$	$\neg\vee$ 1

Figure 4.3: Negated disjunction rule

Now why does *this* work? The above is an instance of what may be called DeMorgan's laws. The relevant instance is the following (usually formulated in the second part as $\neg X \wedge \neg Y$):

Proposition 6. *If* $\mathbf{S} \models \neg(X \vee Y)$, *then* $\mathbf{S} \models \neg X$ *and* $\mathbf{S} \models \neg Y$.

Proof. Suppose $\mathbf{S} \models \neg(X \vee Y)$. Then, $\mathbf{S} \not\models X \vee Y$. Now $\mathbf{S} \models X \vee Y$ just in case either $\mathbf{S} \models X$, $\mathbf{S} \models Y$, or both. Thus if $\mathbf{S} \not\models X \vee Y$, that means none of these cases hold. So that leaves us with $\mathbf{S} \not\models X$ and $\mathbf{S} \not\models Y$. Thus, $\mathbf{S} \models \neg X$ and $\mathbf{S} \models \neg Y$. $\square$

As you can see, our rule just codifies the above instance of DeMorgan's law. And once again, with two successive application of the rule, you can get the negation of either formula X or formula Y, if needed. Compare the following table:

X	Y	$\neg X$	$\neg Y$	$X \wedge Y$	$\neg(X \wedge Y)$
T	T	F	F	T	F
T	F	F	T	F	F
F	T	T	F	F	F
F	F	**T**	**T**	F	**T**

We can stop now for a second and look at our initial question, that of proving that $X \vee \neg X$ is a tautology. As mentioned, in order to do this, we need to put its *negation* at the top of our tree. Then, we have to apply our rules until we get to an explicit contradiction of form $Y \wedge \neg Y$ (for some Y). Thus:

1.	$\neg(X \vee \neg X)$	Start
2.	$\neg X$	$\neg\vee$ 1
3.	$\neg\neg X$	$\neg\vee$ 1
4.	X	$\neg\neg$ 3
	$\underset{2,4}{\otimes}$	

The above is a complete proof of $X \vee \neg X$ being a tautology. You might be asking what the symbol $\otimes$ stands for at the end. It represents that the branch (the only one of the tree) is 'closed', because both a formula and its negation occurs on it. Where? Well, that is given by the numbers below $\otimes$, namely line 2 and 4. Line 2 has the formula $\neg X$, while line 4 has the formula X on it, so we have both X and $\neg X$. Thus, we have our desired explicit contradiction derived from the negation of the tautology candidate. This means that $X \vee \neg X$ *is* a tautology according to our system. Nice!

Let's get back to our other rules before we go further into examples.

Negated conditional

The negated conditional is another one of those non-branching rules that are not immediately obvious. It is specified as follows:

1.	$\neg(X \to Y)$	
2.	X	$\neg \to 1$

1.	$\neg(X \to Y)$	
2.	$\neg Y$	$\neg \to 1$

Figure 4.4: Negated conditional rule

Why does this work? Again, it just codifies a fact, the following fact, from our semantics:

Proposition 7. *If* $\mathbf{S} \models \neg(X \to Y)$, *then* $\mathbf{S} \models X$ *and* $\mathbf{S} \models \neg Y$.

Proof. Suppose $\mathbf{S} \models \neg(X \to Y)$. Then, $\mathbf{S} \not\models X \to Y$. But $\mathbf{S} \models X \to Y$ *iff* whenever $\mathbf{S} \models X$, $\mathbf{S} \models Y$. So again, by negating both sides of the biconditional, $\mathbf{S} \not\models X \to Y$ *iff* it is not the case that if $\mathbf{S} \models X$, then $\mathbf{S} \models Y$. So $\mathbf{S} \not\models X \to Y$ *iff* $\mathbf{S} \models X$ but $\mathbf{S} \not\models Y$. Or in other words, *iff* $\mathbf{S} \models X$ and $\mathbf{S} \models \neg Y$. Which is what we wanted to show. $\square$

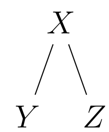
Here is the truth-table fragment to compare:

X	Y	$\neg Y$	$X \to Y$	$\neg(X \to Y)$
T	T	F	T	F
T	F	**T**	F	**T**
F	T	F	T	F
F	F	T	T	F

This concludes our non-branching rules. Now we turn to our branching ones.

4.3.2 Branching rules

As mentioned above, our tree only ever branches to two separate branches from any one point. Thus, our branching rules are all of the form:



Why do we need branching rules? Note that in all of the above examples, the assumption that a certain complex formula is true determined exactly the truth values of some simpler formulas. In the truth table illustrations, this just means that for each complex formula, they were only true in one, and only one, row (the one in gray), so we just had to check what truth values the simpler formulas received in that row (put in bold). However, many formulas are true in several rows in a truth-table. In that case, we need to consider the possibilities that afford the complex formula to be true separately. And this gets compounded if we go further and encounter another such formula, and another one, and so on.

Disjunction

The disjunction rule is the simplest rule of the branching ones. In many ways, it is like the conjunction rule. It looks like this:

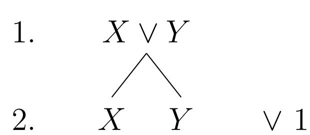


Figure 4.5: Disjunction rule

Here is the reasoning behind this rule. As we know, $\mathbf{S} \models X \vee Y$ *iff* $\mathbf{S} \models X$ or $\mathbf{S} \models Y$ (or both). The problem is that just the fact that $\mathbf{S} \models X \vee Y$ does not tell us whether $\mathbf{S} \models X$ or $\mathbf{S} \models Y$ or both. It only tells us that *it is at least one of these*. Given this lack of information, we need to consider the consequences separately. Interestingly, we need not consider the case where *both* X and Y is true separately. But we also don't make any explicit assumptions about what the truth-value of the *other* formula is on either branch. So all we do is consider the case where X is true, whatever truth value Y may have (T or F), and consider the case where Y is true, whatever truth value X may have (T or F). Here is the table representation:

X	Y	$X \vee Y$
$\mathbf{T}$	$\mathbf{T}$	$\mathbf{T}$
$\mathbf{T}$	F	$\mathbf{T}$
F	$\mathbf{T}$	$\mathbf{T}$
F	F	F

As described above, when we branch to X, we are considering *either* line 1 or line 2 of the truth table. When we branch to Y, we are considering *either* line 1 or line 3 of the truth table.

Negated conjunction

Negated conjunction is also an instance of DeMorgan's laws. It is as follows:

1.	$\neg(X \wedge Y)$		
2.	$\neg X$	$\neg Y$	$\neg\wedge$ 1

Figure 4.6: Negated conjunction rule

Again, we can show the following:

Proposition 8. *If* $\mathbf{S} \models \neg(X \wedge Y)$, *then* $\mathbf{S} \models \neg X$ *or* $\mathbf{S} \models \neg Y$ *(or both).*

Proof. Suppose $\mathbf{S} \models \neg(X \wedge Y)$. Then, $\mathbf{S} \not\models X \wedge Y$. But $\mathbf{S} \models X \wedge Y$ *iff* $\mathbf{S} \models X$ and $\mathbf{S} \models Y$, so $\mathbf{S} \not\models X \wedge Y$ *iff* either $\mathbf{S} \not\models X$ or $\mathbf{S} \not\models Y$, or both. So $\mathbf{S} \models \neg X$, or $\mathbf{S} \models \neg Y$, or both. $\square$

Again, on either branch, we do not make assumptions about the truth value of the formula on the other branch. Thus, on the left branch, we assume $\neg X$ is true, but nothing about $\neg Y$ being true or false, and conversely for the right branch. In table representation:

X	Y	$\neg X$	$\neg Y$	$X \wedge Y$	$\neg(X \wedge Y)$
T	T	F	F	T	F
T	F	F	$\mathbf{T}$	F	$\mathbf{T}$
F	T	$\mathbf{T}$	F	F	$\mathbf{T}$
F	F	$\mathbf{T}$	$\mathbf{T}$	F	$\mathbf{T}$

Conditional

The rule for conditionals is our last rule for the system. It also has a form that may make you stop and think why it is formulated as such.

$$\begin{array}{lccc} 1. & \multicolumn{2}{c}{X \to Y} & \\ 2. & \neg X & Y & \to 1 \end{array}$$

Figure 4.7: Conditional rule

The reasoning behind it is as follows:

Proposition 9. *If* $\mathbf{S} \models X \to Y$*, then* $\mathbf{S} \models \neg X$ *or* $\mathbf{S} \models Y$ *(or both).*

Proof. The proof follows the usual lines. Suppose $\mathbf{S} \models X \to Y$. Then, whenever $\mathbf{S} \models X$, $\mathbf{S} \models Y$. We have already seen that this is *false* when $\mathbf{S} \models X$ but $\mathbf{S} \not\models Y$. This leaves 3 other options left for it to be *true*.

1. First, if $\mathbf{S} \models X$ and $\mathbf{S} \models Y$, clearly, $\mathbf{S} \models X \to Y$. Note that $\mathbf{S} \models X$ only if $\mathbf{S} \not\models \neg X$, but $\mathbf{S} \models Y$.

2. Second, if $\mathbf{S} \not\models X$ but $\mathbf{S} \models Y$, then $\mathbf{S} \models X \to Y$ again. This is the case where both $\mathbf{S} \models \neg X$ (because $\mathbf{S} \not\models X$) *and* $\mathbf{S} \models Y$.

3. Third, if $\mathbf{S} \not\models X$ and $\mathbf{S} \not\models Y$, then still, $\mathbf{S} \models X \to Y$. Again, this means that $\mathbf{S} \models \neg X$, but $\mathbf{S} \not\models Y$ in this case.

As you can see, in all three cases where $\mathbf{S} \models X \to Y$, it is true that either $\mathbf{S} \models \neg X$ or $\mathbf{S} \models Y$ or both. Indeed, those are exactly the three options we have for $X \to Y$ to be true in $\mathbf{S}$. □

Or in truth table form:

X	Y	$\neg X$	$X \to Y$
T	**T**	**F**	**T**
T	F	F	F
F	**T**	**T**	**T**
F	**F**	**T**	**T**

As usual, the rule form only explicitly represents the possibility where $\neg X$ is true, and the possibility where Y is true, leaving open whether Y is true or false on the left branch (and similarly for $\neg X$ on the right one).

The strategy

Here is the strategy again, now with all our rules in hand. Suppose you have a formula X that is a candidate for being a tautology. In order for us to show this in our system, we first have to take $\neg X$, and put that at the *root* of our tree. Then, we have to apply our rules until we have an explicit contradiction of form Y and $\neg Y$ on *each* branch of the tree.

Suppose we want to show something quite elaborate, like the following:

$$(X \to Y) \to (\neg X \vee Y)$$

Remark 4.3.2. You may note the similarity between this *formula schema* and the *rule* for conditionals. Of course, this is no accident. In fact, every one of our rules can be turned into a tautology in our system, with the use of $\to$.

Since the formula is $(X \to Y) \to (\neg X \vee Y)$, again, we put its negation at the root, thus:

1.	$\neg((X \to Y) \to (\neg X \vee Y))$	Start

The next step is to apply one of the rules. For this, you need to identify whether the formula is negated or not, and what the main connective of the formula is, or in the negated case, what the main connective *after* the negation sign is. Here, the answer is: it is a negated conditional. Obviously, you then apply the negated conditional rule. In fact, we don't quite see where we are going yet, so we can apply it twice to get both possibilities. Thus:

1.	$\neg((X \to Y) \to (\neg X \vee Y))$	Start
2.	$X \to Y$	$\neg \to 1$
3.	$\neg(\neg X \vee Y)$	$\neg \to 1$

Now that we have two formulas, we have a choice how we want to proceed. On the other hand, unlike with other systems, we do not have a choice about which rules to apply. Each formula admits of one, and only one, rule, depending on its form.

In general, it is usually better to apply first all the rules that do not branch, and *then* apply any branching rules. If you think about it, this makes sense, since each branching rule doubles the work we have to carry out. At any rate, the non-branching rule is the one for $\neg\vee$, so we can start with that, and apply it twice:

1.	$\neg((X \to Y) \to (\neg X \vee Y))$	Start
2.	$X \to Y$	$\neg \to 1$
3.	$\neg(\neg X \vee Y)$	$\neg \to 1$
4.	$\neg\neg X$	$\neg\vee$ 3
5.	$\neg Y$	$\neg\vee$ 3

So far, we only have a single branch, and that branch is open. On the other hand, we still have formulas on which to apply rules. So we are certainly not done. One trivial step is to get rid of the double negation in line 4. This will not help us much, but it will simplify things a bit. Thus:

1.	$\neg((X \to Y) \to (\neg X \vee Y))$	Start
2.	$X \to Y$	$\neg \to 1$
3.	$\neg(\neg X \vee Y)$	$\neg \to 1$
4.	$\neg\neg X$	$\neg\vee\ 3$
5.	$\neg Y$	$\neg\vee\ 3$
6.	X	$\neg\neg\ 4$

Now most of the formulas in the proof are 'exhausted'. There is no rule for $\neg Y$ or X to apply, and for all the other formulas (except one), we have applied the appropriate rules as many times as possible. So we are left with $X \to Y$. This is a branching rule, so now we branch.

1.	$\neg((X \to Y) \to (\neg X \vee Y))$	Start
2.	$X \to Y$	$\neg \to 1$
3.	$\neg(\neg X \vee Y)$	$\neg \to 1$
4.	$\neg\neg X$	$\neg\vee\ 3$
5.	$\neg Y$	$\neg\vee\ 3$
6.	X	$\neg\neg\ 4$
7.	$\neg X \quad Y$	$\to 2$

Our tree is now complete, there are no more rules to apply. Now comes the crucial part. You have to see if each of the branches are *closed*. Again, this means that you have a formula of form Z and a formula of form $\neg Z$ on each branch. Note that each branch is the sequence of formulas from the root to one of the leaves. Thus, the first branch here is from $\neg((X \to Y) \to (\neg X \vee Y))$ to $\neg X$, the second branch is from $\neg((X \to Y) \to (\neg X \vee Y))$ to Y, through all the other formulas.

Let's take the first branch. Can you see a contradiction? I can, since line 6 has X on it, and line 7 (left) has $\neg X$ on it. So the first branch is closed. Thus:

$$\begin{array}{lcl}
1. & \neg((X \to Y) \to (\neg X \vee Y)) & \text{Start} \\
2. & X \to Y & \neg \to 1 \\
3. & \neg(\neg X \vee Y) & \neg \to 1 \\
4. & \neg\neg X & \neg\vee\ 3 \\
5. & \neg Y & \neg\vee\ 3 \\
6. & X & \neg\neg\ 4 \\
7. & \neg X \qquad Y & \to 2 \\
 & \underset{6,7}{\otimes} \qquad\quad &
\end{array}$$

What about the second branch? Again, there is a contradiction, since line 5 has $\neg Y$, while line 7 (right) has Y. Thus:

$$\begin{array}{lcl}
1. & \neg((X \to Y) \to (\neg X \vee Y)) & \text{Start} \\
2. & X \to Y & \neg \to 1 \\
3. & \neg(\neg X \vee Y) & \neg \to 1 \\
4. & \neg\neg X & \neg\vee\ 3 \\
5. & \neg Y & \neg\vee\ 3 \\
6. & X & \neg\neg\ 4 \\
7. & \neg X \qquad Y & \to 2 \\
 & \underset{6,7}{\otimes} \qquad \underset{5,7}{\otimes} &
\end{array}$$

This completes the proof. Once you have a complete proof of a formula in the system, you can say that that formula is a *theorem* of zeroth-order logic. Thus, $(X \to Y) \to (\neg X \vee Y)$ is a theorem of zeroth order logic. We represent this as follows:

$$\vdash (X \to Y) \to (\neg X \vee Y)$$

The symbol $\vdash$ means something like '*syntactically provable*'. It is the syntactic equivalent to being a tautology in the semantics. In fact, being a tautology (in the semantics) can be represented as follows:

$$\models (X \to Y) \to (\neg X \vee Y)$$

This makes sense, as it is just like writing $\mathbf{S} \models (X \to Y) \to (\neg X \vee Y)$, except we omit a specific structure designation $\mathbf{S}$, since it is modeled by *all* structures, not just a specific one.

Definition 4.3.1 (Theorem)**.** Let X, Y be any formulas. We call a branch of a tree *closed* provided there are some formulas of form Y, $\neg Y$ occurring on it. We call a tree *closed* if all its branches are closed. Then, X is a *theorem of zeroth-order logic* provided there is a closed tree with $\neg X$ at its root. In such cases, we also write $\vdash X$, and call the closed tree for $\neg X$ the tableau (tree) proof of X.

Here is another proof to inspect, with more branching, before it is your turn:

1.	$\neg(((Y \vee \neg Y) \to X) \to (X \wedge (Y \vee \neg Y)))$			Start
2.	$(Y \vee \neg Y) \to X$			$\neg \to 1$
3.	$\neg(X \wedge (Y \vee \neg Y))$			$\neg \to 1$
4.	$\neg(Y \vee \neg Y)$	X		$\to 2$
5.		$\neg X$	$\neg(Y \vee \neg Y)$	$\neg\wedge\ 3$
6.		$\otimes$ $4, 5$	$\neg Y$	$\neg\vee\ 5$
7.			$\neg\neg Y$	$\neg\vee\ 5$
8.	$\neg Y$		$\otimes$ $6, 7$	$\neg\vee\ 4$
9.	$\neg\neg Y$			$\neg\vee\ 4$
	$\otimes$ $8, 9$			

Note that we could have made this derivation even longer, but we opted to apply $\neg\wedge$ to line 3 to extend only the right-side branch of the tree (to line 5). This is because it would not have made any difference on the left-hand side, given that it closed by $\neg(Y \vee \neg Y)$ alone (line 4, then 8 and 9). In any derivation, you are looking to close every branch and nothing more. Thus, you can make derivations significantly shorter if you cut out the steps that are not necessary to close a branch.

> ☞ Here is an intuitive explanation of why $((Y \vee \neg Y) \to X) \to (X \wedge (Y \vee \neg Y))$ is a tautology. If you look at the first half of the formula, it says that $(Y \vee \neg Y) \to X$. I.e., if $Y \vee \neg Y$, then X. But since $Y \vee \neg Y$ is known to be a tautology that holds in all structures, *if* it entails X, then X would also be a tautology (since again, $Y \vee \neg Y$ holds in every structure, and so if in every structure, it entails X, X holds in every structure). But *if* that is the case, then both $Y \vee \neg Y$ and X must hold (both being tautologies).

Exercise 4.3.1. Prove that the following formulas are all theorems of the tableau system:

1. $Y \to (X \to Y)$;

2. $(\neg X \wedge \neg Y) \to \neg(X \vee Y)$;

3. $(\neg X \vee \neg Y) \rightarrow \neg(X \wedge Y)$;

4. $\neg(X \wedge Y) \rightarrow (\neg X \vee \neg Y)$;

5. $\neg(X \vee Y) \rightarrow (\neg X \wedge \neg Y)$;

6. $(\neg X \vee Y) \rightarrow (X \rightarrow Y)$;

7. $(X \rightarrow Y) \rightarrow (\neg Y \rightarrow \neg X)$;

8. $((X \vee Y) \wedge ((X \rightarrow Z) \wedge (Y \rightarrow Z))) \rightarrow Z$.

4.4 Some facts about the tableau system

4.4.1 On soundness and completeness

A fundamental goal for every logical system is to show that the semantics and the syntactic system agree on every formula. That is, if X is a tautology according to the semantics, it can be proved in the system that X is a theorem, and if X is a theorem in the system, it really is a tautology in the semantics. This can be represented as follows:

$$\models X \textit{ iff } \vdash X$$

Or again, X is a theorem *iff* it is a tautology. I have been using these two terms somewhat interchangeably above, since as it stands, in the system I just introduced, they really do come down to the same thing. Usually, this is characterized two ways:

1. if whenever $\mathbf{S} \vdash X$, then $\mathbf{S} \models X$, the system relative to the semantics is *sound*;

2. if whenever $\mathbf{S} \models X$, then $\mathbf{S} \vdash X$, the system relative to the semantics is *complete*.

Of course, if you put together these two conditionals, you get the biconditional above. In general, it is important to keep in mind that our semantics and our deductive system are two distinct things, and that they coincide on their judgments using two distinct methods is not a trivial feat (in fact, large classes of logics *cannot* have a non-trivial sound and complete deductive system). On the other hand, *proving* soundness and completeness is rather involved and takes more machinery than we currently possess (or will possess by the end of this book). Thus, for now, you will have to trust the process.

The process, our tableau method, is in many ways better than other approaches you may have seen before, like natural deduction. The reason is because the tableau method does not require you to be creative in your proofs at all. In fact, it is provable that given a formula X, if X is a tautology, then applying the relevant rule as many times as possible to $\neg X$ and each resulting formula completely mindlessly will result in a tree all of whose branches *will* close, and if X is not a tautology, there will be at least one open branch left in the end. So in general, all you have to do is recognize for each formula which rules can be applied, and apply them. Of course, if you are an expert in using the system, you will be able to do shorter proofs by using your brain (e.g., by using non-branching rules only once, deriving the relevant side only).

4.4.2 Open tableaus and their uses

In fact, here is another way in which tableaus are better than natural deduction. Consider again the above crucial fact about our syntax and deductive system:

$$\models X \textit{ iff } \vdash X$$

Now using the usual technique on the biconditional, negate both sides. Thus:

$$\not\models X \textit{ iff } \nvdash X$$

Now we can think about what *this* means. First, $\not\models X$ means that X is not a tautology. Second, $\nvdash X$ means that X is not a theorem of the system. The latter in turn just means that $\neg X$ (!) does not (cannot) have a closed tableau. So it can only have open tableaus. So X is not a tautology provided $\neg X$ only has open tableaus. So far so good. Let's continue.

What does it mean for X to not be a tautology? Well, by definition, it means that X is either a contradiction or contingent. And in turn, if X is either a contradiction or contingent, then $\neg X$ is either a tautology or contingent. So injecting this back to our previous line of reasoning, we get:

Proposition 10. *$\neg X$ is either a tautology or contingent iff $\neg X$ does not have a closed tableau.*

We can chase these facts a bit further. In particular, if $\neg X$ is either a tautology or contingent, that means that in either case, there is at least one structure $\mathbf{S}$ such that $\mathbf{S} \models \neg X$. In other words, if $\neg X$ does not have a closed tableau, we can immediately answer that it has a model! (And of course, the same goes for X.)

4.5 Extending our approach to sets of formulas

Students of logic are usually taught at first about tautologies, contradictions, and contingent propositions. However, these are not the notions you will encounter in more advanced, and more modern, works. This is at least partly because these notions are usually used for individual formulas, while a lot of the times, we want to talk about sets of formulas instead.

One thing that we can immediately extend to sets of formulas is contingency. Sometimes, it is not just one formula that has a model, but a whole set of them. Indeed, perhaps an infinitely large set of them! In these cases, we sometimes say that the set S of formulas is (zeroth-order) *satisfiable* or *semantically consistent*. This can be captured in a precise manner as follows:

Definition 4.5.1 (Satisfiability)**.** A set S of formulas of $\mathcal{L}_0$ is (zeroth-order) *satisfiable* or *semantically consistent iff* there is a structure $\mathbf{S}$ such that $\mathbf{S} \models X$ for each $X \in S$. We say S is (zeroth-order) *unsatisfiable* or *semantically inconsistent* provided it is not semantically consistent.

Remark 4.5.1. Note that the above definition is *not* the same as claiming that for each $X \in S$, there is a structure $\mathbf{S}$ such that $\mathbf{S} \models X$. This would not make *the set* consistent, it would make the formulas contingent individually. For example, if X is contingent, then $\neg X$ is also contingent, so both X and $\neg X$ have a model. But clearly, $\{X, \neg X\}$ is not consistent, since there is no structure in which both of these could be true.

Here is a useful way to think about consistency and contingency:

Proposition 11. *The formula X is contingent iff the singleton set $\{X\}$ is semantically consistent.*

Exercise 4.5.1. Prove Proposition 11. Hint: show from left to right that if X is contingent, $\{X\}$ is consistent, and from right to left that if $\{X\}$ is consistent, X is contingent. To do this, use the definitions for these notions.

Here is an interesting fact about consistency and contradictions:

Proposition 12. *If $S = \{X_1, ..., X_n\}$, and some $X \in S$ is a contradiction, then S cannot be semantically consistent.*

Proof. In other words, a single contradiction can make a set of formulas inconsistent. This is easy to see since semantic consistency demands that there be at least one structure $\mathbf{S}$ in which each member of S is true. Now if X is a contradiction, there is no structure in which it is true, and $X \in S$ by assumption. $\square$

On the other hand, we have:

Proposition 13. *If S is a set of tautologies, S is semantically consistent.*

Exercise 4.5.2. The proof is left as an exercise.

4.5.1 Validity

Here comes something magical. So far, we talked about certain sets of formulas that are satisfiable, and certain sets of formulas that are unsatisfiable. And this boiled down to whether the set of formulas have a common model, or they do not. Now as it turns out, satisfiability and unsatisfiability are sufficient to define the central notion of logic: validity.

This may not be apparent at first. Take the usual definition of validity:

> (A1) An argument is valid *iff* whenever the premises are true, the conclusion must also be.

It seems like this has absolutely nothing to do with whether sets of formulas are satisfiable. Not so fast!

What is an argument? We can define it as such:

Definition 4.5.2. An argument is a set of formulas, of which one is designated the conclusion, and the rest are designated the premises.

This could be formalized with more structure, using ordered sets and whatnot, but what I want to emphasize here is that arguments are just sets of formulas with some designated members.

Let's return to validity. Validity says that if an argument is valid, then whenever its premises are true, its conclusion must also be true. You may ask yourself: is it possible for a *valid* argument's premises to be *true*, but its conclusion to be *false*? It seems not, since that is exactly what validity excludes. It says that *if* a valid argument's premises happen to be true, then whatever else is the case, the conclusion *must* also be true. Thus, we can also characterize validity as follows:

> (A2) An argument is valid *iff* it is *impossible* for its premises to be true, but its conclusion to be false.

Note the word 'impossible' in italics. What does it mean that it is impossible for the premises to be true and the conclusion to be false? Well, it just means: there cannot be a structure $\mathbf{S}$ such that $\mathbf{S}$ models the premises, but $\mathbf{S}$ does not model the conclusion. Let's use some specific variables to make this more precise.

Suppose you have an argument with (possibly infinite) premises $P = \{X_1, X_2, ...\}$, and conclusion Y. Then, by the above, if this argument is *valid*, that means there is no structure $\mathbf{S}$ such that $\mathbf{S} \models X$ for all $X \in P$, but $\mathbf{S} \not\models Y$ (the conclusion). We also know that $\mathbf{S} \not\models Y$ *iff* $\mathbf{S} \models \neg Y$, so we can say: if an argument is *valid*, that means there is no structure $\mathbf{S}$ such that $\mathbf{S} \models X$ for all $X \in P$, and $\mathbf{S} \models \neg Y$. Again, this is just a formal way of saying it is impossible for the premises to be true but the conclusion to be false (or equivalently, it is impossible for the premises to be true, and the negation of the conclusion to also be true).

But note that now we really are talking about unsatisfiability! For what we just described may simply be put: the set $\{X_1, X_2, ..., \neg Y\}$ is unsatisfiable. And this is, indeed, what validity boils down to. In particular, an argument is valid *if, and only if* its premises, together with the negation of its conclusion, are unsatisfiable. And again, all this just means that it is impossible for a valid argument to have true premises and a false conclusion. Thus:

Definition 4.5.3 (Validity)**.** Let $A = \{X_1, X_2, ...\}$ be any set of formulas and let Y be any formula of $\mathcal{L}_0$. Then, the argument from *premises* A to the *conclusion* Y is *valid iff* the set $\{X_1, X_2, ..., \neg Y\}$ is not satisfiable. If an argument from A to Y is *valid*, we write $A \models Y$, or say that Y is a *semantic consequence* of, or *semantically entailed* by A.

Remark 4.5.2. Indeed, from this, we get an alternative definition of being a tautology for free. For note that the premise set A was not required to be non-empty. So A may be $\emptyset$. So by definition, $\emptyset \models Y$ *iff* $\{\neg Y\}$ is unsatisfiable. And $\{\neg Y\}$ is unsatisfiable *iff* Y is a tautology. In such cases, we can just write $\models Y$, as we have been doing.

Exercise 4.5.3. Show that the following hold in zeroth-order logic:

1. $\neg\neg X \models X$ and $X \models \neg\neg X$
2. $\neg X \wedge \neg Y \models \neg(X \vee Y)$ and $\neg(X \vee Y) \models \neg X \wedge \neg Y$;
3. $\neg X \vee \neg Y \models \neg(X \wedge Y)$ and $\neg(X \wedge Y) \models \neg X \vee \neg Y$;
4. $\neg X \vee Y \models X \to Y$ and $X \to Y \models \neg X \vee Y$;
5. $X \to Y \models \neg Y \to \neg X$ and $\neg Y \to \neg X \models X \to Y$;

4.5.2 Validity and tableau

From the above, you may already see how reasoning with sets of formulas is implemented in our system. With single formulas, we had that if *every* possible tree for $\neg X$ remained open, that must mean $\neg X$ is satisfiable (i.e., it is a tautology or contingent), which in turn ensures that X is *not* a tautology. On the other hand, if there is a possible tree for $\neg X$ that closes (closes on all its branches), then X is indeed a tautology, since its negation is not satisfiable. Similarly, but with sets of formulas, if a tree for $\{X_1, X_2, ..., \neg Y\}$ closes at least once, then $\{X_1, X_2, ..., \neg Y\}$ is *not* satisfiable, which in turn means that $\{X_1, X_2, ..\}$ entails Y. On the other hand, if *no* tree for $\{X_1, X_2, ..., \neg Y\}$ closes, that means the set is satisfiable, and $\{X_1, X_2, ...\}$ does not entail Y.

The only question is: how do we work with multiple formulas, when in the simpler case, we put the sole formula $\neg X$ at the root of the tree (when trying to prove X is a tautology)? First, with arguments, we put the *negation* of the *conclusion* at the root of the tree. So if we are considering whether the set $S = \{X_1, X_2, ...\}$ of premises is such that $S \vdash Y$, we put $\neg Y$ at the root of the tree.

The other change is more substantial. We introduce a very simple new rule into the system that enables us to inject any one of the premises at any part of our derivation. Thus:

1.	X	
2.	Y	Pr.

Provided Y is one of the premises.

Figure 4.8: Premise rule

Note that X and Y here are completely independent of one another. X may be any formula that you happen to consider at any one point of your proof, and Y can be any formula as long as it is one of the premises under consideration. Accordingly, the premise rule is radically different from all the previous rules, since it does not 'depend' on any formula at any one point of the tree. In other words, you can put any premise down at any one point of your derivation, and in fact, as many times as you want.

☞ The relationship between the premise rule and all the other rules of our tableau system is similar to the relationship between the base rule and the productive rule(s) in our syntactic derivations.

Suppose you want to show that $\{X, X \to Y\} \vdash Y$. This is usually called *Modus Ponens*, and it is a valid argument form. In other words, for any choice of X, Y, it is true that $\{X, X \to Y\} \models Y$. So let's show the corresponding fact syntactically.

As noted above, the first step here is to put the *negation* of the conclusion at the root of the tree derivation. Since Y is the conclusion, the tree will start out as such:

1.	$\neg Y$	Start

Now unlike with tautologies, just having the negation of the conclusion gets us nowhere (if the conclusion is not itself a tautology). In fact, in this case, there is no rule you can use on $\neg Y$ to get further. On the other hand, what you *can* do is use the premise rule. There are two premises, so you have a choice. It is not *always* the case that all the premises are used, and in fact, the order in which they are used is also not predetermined, so there is a degree of freedom here.

Let's start with the more complex premise:

1.	$\neg Y$	Start
2.	$X \to Y$	Pr.

Great! You should now see that you can use one of your other rules on the new formula. Thus:

1.	$\neg Y$		Start
2.	$X \to Y$		Pr.
3.	$\neg X$	Y	$\to 2$

You may have immediately noticed that the right side branch closes by the occurrence of Y and $\neg Y$ on it. Thus, we have:

1.	$\neg Y$		Start
2.	$X \to Y$		Pr.
3.	$\neg X$	Y	$\to 2$
		$\underset{1,3}{\otimes}$	

On the other hand, you might also see that the left hand side branch of the tree remains open, and there are no formulas left on which we might apply a rule. Thus, we have to look for another premise. (Not) incidentally, our other premise, X is just what we need. In particular:

1.	$\neg Y$		Start
2.	$X \to Y$		Pr.
3.	$\neg X$	Y	$\to 2$
4.	X	$\underset{1,3}{\otimes}$	Pr.
	$\underset{3,4}{\otimes}$		

Accordingly, we can conclude that $\{X, X \to Y\} \vdash Y$.

Definition 4.5.4. Let $A = \{X_1, X_2, ...\}$ be any set of formulas and let Y be any formula of $\mathcal{L}_0$. Then, the set of premises A *syntactically entails* Y, or Y is a *syntactic consequence* of A, *iff* there is a closed tree with $\neg Y$ at its origin, and with possible uses of the rule Pr. with any member of A. In such cases, we write $A \vdash Y$, and call the closed tree a *proof* of Y from the premise set (or simply, premises) A.

Let's look at a more complicated proof before it is your turn. Suppose you want to evaluate the following argument:

(P1)	$X \vee \neg Y$
(P2)	$\neg\neg Y$
(C)	$(Q \vee Z) \to X$

As a first step, like before, we put the negation of the conclusion at the root of our tree. Thus:

1.	$\neg((Q \vee Z) \to X)$	Start

Note that we are negating the whole formula, so you have to provide the outer parentheses if they were omitted previously.

It is a good idea to break this down a bit more immediately, since it won't branch.

1.	$\neg((Q \vee Z) \to X)$	Start
2.	$Q \vee Z$	$\neg \to 1$
3.	$\neg X$	$\neg \to 1$

As before, there are various ways to proceed, some shorter than others. In particular, we could start breaking down $Q \vee Z$. However, this would make the proof longer than necessary. Once you are more comfortable with how the system works, you will be able to see this.

For now, let's proceed by adding $\neg\neg Y$ with our premise rule. We could immediately simplify it further, but for now, we will leave it as is. So we have:

1.	$\neg((Q \vee Z) \to X)$	Start
2.	$Q \vee Z$	$\neg \to 1$
3.	$\neg X$	$\neg \to 1$
4.	$\neg\neg Y$	Pr.

Then, we can add the other premise to our tree, again, non-branching:

1.	$\neg((Q \vee Z) \to X)$	Start
2.	$Q \vee Z$	$\neg \to 1$
3.	$\neg X$	$\neg \to 1$
4.	$\neg\neg Y$	Pr.
5.	$X \vee \neg Y$	Pr.

We now have a choice to branch. But doing the proof briefly in our head, we immediately see that if we branch to X and $\neg Y$, our tree immediately closes by $\neg X$ and $\neg\neg Y$. On the other hand, if we branch to Q and Z, nothing happens, and we still need to branch further to X and $\neg Y$ to close the tree. And in fact, in the latter case, we have to do it twice, resulting in four branches in the end.

So the simplest way is to go immediately to:

1.	$\neg((Q \vee Z) \to X)$		Start
2.	$Q \vee Z$		$\neg \to 1$
3.	$\neg X$		$\neg \to 1$
4.	$\neg\neg Y$		Pr.
5.	$X \vee \neg Y$		Pr.
6.	X	$\neg Y$	$\vee$ 5
	$\otimes$ 3, 6	$\otimes$ 4, 6	

Again, you can compare this to:

1.	$\neg((Q \vee Z) \to X)$				Start
2.	$Q \vee Z$				$\neg \to 1$
3.	$\neg X$				$\neg \to 1$
4.	$\neg\neg Y$				Pr.
5.	$X \vee \neg Y$				Pr.
6.	Q		Z		$\vee$ 2
7.	X	$\neg Y$	X	$\neg Y$	$\vee$ 5
	$\otimes$ 3, 7	$\otimes$ 4, 7	$\otimes$ 3, 7	$\otimes$ 4, 7	

And in fact, if you branched immediately at the beginning as many times as possible, and applied every rule as many times as possible, you would have had this:

1.	$\neg((Q \vee Z) \to X)$				Start
2.	$Q \vee Z$				$\neg \to 1$
3.	Q		Z		$\vee$ 2
4.	$X \vee \neg Y$		$X \vee \neg Y$		Pr.
5.	X	$\neg Y$	X	$\neg Y$	$\vee$ 4
6.	$\neg X$	$\neg X$	$\neg X$	$\neg X$	$\neg \to 1$
7.	$\neg\neg Y$	$\neg\neg Y$	$\neg\neg Y$	$\neg\neg Y$	Pr.
8.	Y	Y	Y	Y	$\neg\neg$ 7
	$\otimes$ 5, 6	$\otimes$ 5, 8	$\otimes$ 5, 6	$\otimes$ 5, 8	

Now this may not look *that* bad on paper, but notice that the first proof had 7 occurrences of formulas, the second had 11, while the last one had 22! On the other hand, none of these proofs are incorrect. It is just that they become less and less elegant.

4.5.3 On soundness and completeness, again

Note that we can now reformulate our soundness and completeness claims from above in a stronger form. Previously, we had:

$$\models X \textit{ iff } \vdash X$$

Again, from left to right, this is completeness, and from right to left, this is soundness. But we also have a stronger version of this, with multiple formulas, as opposed to singular ones. In particular:

$$A \models X \textit{ iff } A \vdash X$$

Or in words, A *semantically* entails X *iff* A *syntactically* entails X. And once again, from left to right, this is (strong) completeness, and from right to left, this is (strong) soundness (for zeroth-order logic).

We shall not prove this biconditional in this book, as it involves more advanced techniques than we are ready for. However, we can note the following.

Let's negate both sides of the biconditional again, just as before. Then, we get:

$$A \not\models X \textit{ iff } A \nvdash X$$

Again, we can try and unpack the two sides, and see what we get. First, if $A \nvdash X$, that means that $\neg X$ cannot have a closed tableau, no matter which members of A are used in any of the attempted trees. In other words, all the trees starting with $\neg X$ will remain open, no matter how long they might be.

On the other side, we have that $A \not\models X$, i.e., that A does not semantically entail X. Now previously, we defined semantic entailment in terms of unsatisfiability. In particular, A semantically entails X provided the set A together with $\neg X$ form a set (precisely A plus $\neg X$) that is unsatisfiable. Accordingly, A does *not* semantically entail X provided A together with $\neg X$ is not unsatisfiable. Or in other words, A together with $\neg X$ *is* satisfiable.

Putting the above two results together, we then get that A together with $\neg X$ is satisfiable *iff* $\neg X$ does not have a closed tableau from premises A.

Exercise 4.5.4. For each set of formulas A and formula B of $\mathcal{L}_0$ below, show that $A \vdash B$ using the tableau method.

1. $A = \{X \to Y, Y \to Z, \neg Z\}$ and $B = \neg X$;
2. $A = \{X \vee (Z \to Q), R \wedge Z, \neg Q\}$ and $B = X$;
3. $A = \{\neg((X \vee \neg Y) \wedge (Z \wedge \neg Q))\}$ and $B = (\neg X \wedge Y) \vee (\neg Z \vee Q)$
4. $A = \{Z \to (X \wedge \neg X), \neg Q \vee Z, X \to Q\}$, $B = \neg X$;
5. $A = \{(\neg X \wedge Y) \vee (Z \wedge \neg Q), \neg X \to \neg Y, Z \to Q\}$, $B = R$;
6. $A = \{X \mid X \text{ is a formula of } \mathcal{L}_0\}$, $B = \neg Y$.

Chapter 5

The syntax of first-order logic

Now that we are comfortable with the syntax, semantics, and tableau system of zeroth-order logic, we are in a position to consider a more expressive language, the language of *first-order* logic. Unsurprisingly, we will denote the language of first-order logic by $\mathcal{L}_1$. Just as before, the language has a syntactic aspect, the way its expressions are formed, and a semantic aspect, the way its expressions are given meaning. Clearly, the second depends on the first, so we shall start, again, with the syntax of the language under consideration.

In fact, the language $\mathcal{L}_1$ is very similar to $\mathcal{L}_0$, save for one crucial device; quantification. Accordingly, we still have the same predicates and constants as before, and the same connectives to connect atomic formulas together. However, what we *also* have are *variables*, and *quantifier symbols* that bind these variables.

5.1 Variables and atomic formulas

Let's start with the variables. Variables in $\mathcal{L}_1$ function similar to constants. In particular, just as there is a set of constants $\mathfrak{c}_n$ for each natural number n, in $\mathcal{L}_1$, there is also a set of *variables* $\mathbf{x}_n$, for each natural number n. In list format, we have:

$$\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_5, \mathbf{x}_6, \ldots.$$

Moreover, these variables can take exactly the same places as the constants. For example, if you have a predicate $\mathfrak{P}^2_2$, then both $\mathfrak{P}^2_2(\mathfrak{c}_4, \mathfrak{c}_2)$ and $\mathfrak{P}^2_2(\mathbf{x}_4, \mathbf{x}_9)$ are atomic formulas. Because of this, constants and variables *together* are usually called *terms*, to simplify the definition of what it means for something to be an atomic formula. Thus first, we have:

Definition 5.1.1 (Constants, variables, terms)**.** The set $\mathsf{CONS}_{\mathcal{L}_1} = \{\mathfrak{c}_n \mid n \in \mathbb{N}\}$, and we call it the *constants* of $\mathcal{L}_1$. The set $\mathsf{VAR}_{\mathcal{L}_1} = \{\mathbf{x}_n \mid n \in \mathbb{N}\}$ and we call it the *variables* of $\mathcal{L}_1$.

The set $\mathsf{TERM}_{\mathcal{L}_1} = \{t \mid t \in \mathsf{VAR}_{\mathcal{L}_1} \text{ or } t \in \mathsf{CONS}_{\mathcal{L}_1}\}$, and we call it the *terms* of $\mathcal{L}_1$.

Remark 5.1.1. Notice that we do not use our special font for denoting terms, because technically, every term t stands for either a variable or a constant of $\mathcal{L}_1$

Once we have the notion of a *term*, which is, again, either a variable or a constant of the language $\mathcal{L}_1$, we can easily define again what it means for something to be an atomic formula – this time for $\mathcal{L}_1$.

As just mentioned, there is nothing more to this than specifying that if you have an n-place predicate $\mathfrak{P}^n_k$, and it is followed by n *terms* (constants or variables) of $\mathcal{L}_1$ in parentheses, separated by commas, then the resulting expression $\mathfrak{P}^n_k(t_1, ..., t_n)$ is an atomic formula of $\mathcal{L}_1$.

Here are a few examples of atomic formulas of $\mathcal{L}_1$

$$\mathfrak{P}^3_3(\mathfrak{c}_3, \mathbf{x}_5, \mathbf{x}_9)$$
$$\mathfrak{P}^3_3(\mathbf{x}_2, \mathfrak{c}_6, \mathfrak{c}_7)$$
$$\mathfrak{P}^2_4(\mathfrak{c}_3, \mathfrak{c}_5)$$
$$\mathfrak{P}^2_4(\mathbf{x}_7, \mathbf{x}_8)$$

There are a few things you should notice here. First, there are atomic formulas in which both variables and constants occur. Moreover, there are other atomic formulas which are just like the ones before, but in place of variables, there are now constants, and in place of constants, there are now variables. So in general, constants and variables can always take each others' places. Second, there are atomic formulas that *only* have constants, and atomic formulas that *only* have variables, and in each of these cases, for any predicate $\mathfrak{P}$, the situation could have been the converse.

So again, we have:

Definition 5.1.2 (Atomic formula)**.** If P^n is an n-place predicate of $\mathcal{L}_1$ (i.e., $P^n \in \mathsf{PRED}_{\mathcal{L}_1}$), and $t_1, ..., t_n$ are terms of $\mathcal{L}_1$ (i.e., $t_1, ..., t_n \in \mathsf{TERM}_{\mathcal{L}_1}$), then:

$$P^n(t_1, ..., t_n)$$

is an atomic formula of $\mathcal{L}_1$. The set of atomic formulas of $\mathcal{L}_1$ is denoted $\mathsf{ATOM}_{\mathcal{L}_1}$.

5.2 Complex formulas and quantifiers

So far so good. Now comes the more confusing part. When we are forming more complex formulas in $\mathcal{L}_1$, we can not only use connectives, but also *quantifiers*. And quantifiers work nothing like the connectives! Moreover, quantifiers and variables have an important relation to each other.

Let's start with the *formulas* of $\mathcal{L}_1$. It is not hard to define what it means for something to be a formula of our new language. We already know what the atomic formulas are, we still have our connectives as usual, and we add two other symbols, $\forall$ ('for all', 'for every') for the *universal* quantifier, and $\exists$ ('for some', 'there exists') for the *existential quantifier*. Thus:

$\forall$	Universal quantifier	'For all', 'For every'
$\exists$	Existential quantifier	'For some', 'There exists'

How the quantifiers show up in our formulas is as follows. In each case, for a formula X, you can form a quantified formula by adding the quantifier *with a specific variable* in front of the formula. So again, if X is any formula of $\mathcal{L}_1$, then for example, $\forall \mathbf{x}_1 X$ is also a formula of $\mathcal{L}_1$. And so is $\exists \mathbf{x}_1 X$. And so is $\exists \mathbf{x}_{45967} X$. But of course, since X can be *any* formula of $\mathcal{L}_1$, X might already be a *quantified* formula (that is, it might already have quantifiers), and it might even be a complex formula with quantifiers and connectives (or just connectives). So this new rule of ours gets us infinitely many new formulas in a very strong sense. For example, it gets us infinitely many new formulas even if we fix a quantifier and a variable.

Thus, we have the definition:

Definition 5.2.1 (Formulas)**.** The set of formulas $\mathsf{FORM}_{\mathcal{L}_1}$ of $\mathcal{L}_1$ is the smallest set such that:

1. every $X \in \mathsf{ATOM}_{\mathcal{L}_1}$ is in $\mathsf{FORM}_{\mathcal{L}_1}$;

2. if X, Y are in $\mathsf{FORM}_{\mathcal{L}_1}$, then $\neg X$, $(X \wedge Y)$, $(X \vee Y)$, and $(X \to Y)$ are in $\mathsf{FORM}_{\mathcal{L}_1}$;

3. if Y is in $\mathsf{FORM}_{\mathcal{L}_1}$ and x is in $\mathsf{VAR}_{\mathcal{L}_1}$, then $\forall x Y$ and $\exists x Y$ are in $\mathsf{FORM}_{\mathcal{L}_1}$.

Exercise 5.2.1. For each of the following expressions, determine if it is a formula of $\mathcal{L}_1$ or not. If it is, provide a linear or tree derivation of it. If it is not, explain why it is not.

1. $\forall \mathbf{x}_3 \mathfrak{P}^1_3(\mathfrak{c}_9)$

2. $(\mathfrak{P}^2_5(\mathbf{x}_9, \mathbf{x}_3) \vee \neg \exists \mathbf{x}_4 (\mathfrak{P}^1_1(\mathbf{x}_9) \vee \neg \mathfrak{P}^1_1(\mathbf{x}_7)))$

3. $\neg \forall \mathbf{x}_5 (\mathfrak{P}^3_2(\mathfrak{c}_4, \mathfrak{c}_1, \mathbf{x}_3) \vee \exists \mathbf{x}_1 \mathfrak{P}^1_2(\mathfrak{c}_{13}))$

4. $\exists \mathbf{x}_3 (\mathfrak{P}^1_5(\mathfrak{c}_9))$

5. $(\exists \mathbf{x}_6 \mathfrak{P}^1_4(\mathbf{x}_3))$

6. $\forall x \neg(\mathfrak{P}_1^2(\mathfrak{c}_2, \mathbf{x}_4) \vee \exists \mathbf{x}_3 \mathfrak{P}_5^1(\mathfrak{c}_9))$

7. $\neg\neg\neg\exists \mathbf{x}_6 \mathfrak{P}_4^1(\mathbf{x}_3)$

8. $\neg(\exists \mathbf{x}_3 \mathfrak{P}_5^1(\mathfrak{c}_9))$

So far, not very complicated. But here's the rub. In first-order logic, we are not (at least currently) interested in *all* formulas. Instead, we are interested in a special subset of them; those that are called 'closed formulas'. The formulas that are not closed are called 'open'. So there are open and closed formulas, and we want the closed ones. These closed formulas are also sometimes called 'sentences'. Again, this way, sentences are just a special subset of all formulas – the closed ones.

Here is the basic idea behind what makes a formula closed or its opposite, open. Note that the quantifiers $\forall$ and $\exists$ always come with some specific variable after them. They also have a *scope*, which is just the formula which they flank from the left. In other words, the scope of $\exists xY$ is Y, and the scope of $\forall xY$ is, again, Y, and in either case, the variable is x. This is important for the following reason.

If a certain variable that occurs after the quantifier also occurs in the *scope* of the quantifier (and is not already bound), then we say that that variable's occurrence is bound by that quantifier. If an occurrence of a variable is bound, it is not free, and it is free if it is not bound.

Let's look at some examples:

$$\mathfrak{P}_1^1(\mathbf{x}_1) \wedge \mathfrak{P}_1^1(\mathbf{x}_2) \tag{5.1}$$
$$\forall \mathbf{x}_1(\mathfrak{P}_1^1(\overline{\mathbf{x}_1}) \wedge \mathfrak{P}_1^1(\mathbf{x}_2)) \tag{5.2}$$
$$\forall \mathbf{x}_1(\mathfrak{P}_1^1(\overline{\mathbf{x}_1}) \wedge \mathfrak{P}_1^1(\mathbf{x}_2)) \wedge \mathfrak{P}_1^1(\mathbf{x}_1) \tag{5.3}$$
$$\exists \mathbf{x}_2 \forall \mathbf{x}_1(\mathfrak{P}_1^1(\overline{\mathbf{x}_1}) \wedge \mathfrak{P}_1^1(\overline{\mathbf{x}_2})) \tag{5.4}$$
$$\forall \mathbf{x}_1(\mathfrak{P}_1^1(\overline{\mathbf{x}_1}) \wedge \exists \mathbf{x}_2 \mathfrak{P}_1^1(\overline{\mathbf{x}_2})) \tag{5.5}$$
$$\forall \mathbf{x}_1 \exists \mathbf{x}_1 \mathfrak{P}_1^1(\overline{\mathbf{x}_1}) \tag{5.6}$$

To have a visual representation, I put a horizontal line over each of the *bound* occurrences of the variables. Every other occurrence of a variable occurs *free*.

First, with formula 5.1, there are no quantifiers, so none of the occurrences of the variables are bound. In other words, all the variables occur free. In formula 5.2, $\mathbf{x}_1$ is in the scope of the quantifier $\forall$ with $\mathbf{x}_1$, so it gets bound by it. But there is no quantifier for $\mathbf{x}_2$, so that remains open.

The next formula, formula 5.3 shows something important. Sometimes, some *occurrences* of variables are bound, but some other *occurrences* remain free. As you can see, the *first* occurrence of $\mathbf{x}_1$ gets bound by the universal quantifier with $\mathbf{x}_1$, but the *second* occurrence does *not*. Why? Because that occurrence of $\mathbf{x}_1$ is not in the *scope* of the quantifier. If you made a syntax tree for the formula, that specific occurrence of the variable would not be *underneath* the quantifier, it would be on a different branch altogether, while the first occurrence would be. This is why we talk of occurrences of variables, and not simply variables.

Formulas 5.4 and 5.5 show two ways in which both variables can be bound. Note that though $\exists\mathbf{x}_2$ takes a different place in the second, it still manages to have the sole occurrence of $\mathbf{x}_2$ be in its scope, so it binds it either way.

Finally, 5.6 shows something unfortunate. As our syntax is defined, there can be *overlapping* quantifiers with identical variables. In such cases, we need a rule to determine which quantifier binds the variable. In this case, that quantifier is $\exists$. Why? Because in $\mathfrak{P}^1_1(\mathbf{x}_1)$, $\mathbf{x}_1$ occurs free. In $\exists\mathbf{x}_1\mathfrak{P}^1_1(\mathbf{x}_1)$, $\mathbf{x}_1$ no longer occurs free, it gets bound by $\exists\mathbf{x}_1$. So when $\forall\mathbf{x}_1$ arrives to the party late, it has no *free* occurrences of variables to bind anymore. Thus $\forall\mathbf{x}_1$ does nothing – it binds no variables because there are no *free* variables to bind.

As mentioned above, formulas in which all the occurrences of the variables are bound are *closed*, and they are *sentences*. Formulas in which there is at least one unbound (free) occurrence of a variable are called *open*, and they are *not* sentences.

So above, 5.4, 5.5, and 5.6 are closed formulas, and are therefore sentences. On the other hand, 5.1, 5.2, 5.3 are open formulas, and are therefore *not* sentences.

Now there is a clever way to define what it means to be a sentence of $\mathcal{L}_1$, by keeping count at each turn of forming a formula which variables have at least one free occurrence. It is as follows:

Definition 5.2.2 (Sentences)**.** Let Y, Z be any formulas and x be any variable of $\mathcal{L}_1$. We define the variables with at least one occurrence free in a formula, the free variables of a formula, as follows:

1. if Y is an *atomic* formula, all its variables are free;
2. the free variables of $\neg Y$ are the same as the free variables of Y;
3. the free variables of $(Y \wedge Z)$, $(Y \vee Z)$, $(Y \rightarrow Z)$ are the free variables of Y and of Z taken together;
4. the free variables of $\forall xY$ and $\exists xY$ are the free variables of Y *minus* the variable x.

A formula Y of $\mathcal{L}_1$ is *open iff* it does not have a free variable, and *closed* otherwise. The formula Y is a *sentence iff* it is closed.

Exercise 5.2.2. Determine for each of the following formulas whether they are a sentence of $\mathcal{L}_1$ or not. If any formula is *not* a sentence, underline the *unbound* occurrences of its variables.

1. $\mathfrak{P}^1_4(\mathbf{x}_4)$
2. $\mathfrak{P}^1_4(\mathfrak{c}_4)$
3. $\forall \mathbf{x}_5 \mathfrak{P}^2_5(\mathbf{x}_3, \mathfrak{c}_1)$
4. $\mathfrak{P}^1_7(\mathbf{x}_9) \to \exists \mathbf{x}_4 \mathfrak{P}^2_6(\mathbf{x}_4, \mathfrak{c}_9)$
5. $\mathfrak{P}^1_7(\mathfrak{c}_9) \to \exists \mathbf{x}_4 \mathfrak{P}^2_6(\mathbf{x}_4, \mathbf{x}_4)$
6. $\forall \mathbf{x}_1(\mathfrak{P}^1_4(\mathbf{x}_1) \to \neg\forall \mathbf{x}_1 \mathfrak{P}^1_5(\mathbf{x}_1))$

5.3 Some further conventions

As before, we shall make some simplifications to our notation to make our formulas more readable, and our claims more general. We already know that we can refer to any predicate $\mathfrak{P}^n_k$ (for some $n, k \in \mathbb{N}$) by the metavariables P, Q, R, ..., and we can refer to any constant $\mathfrak{c}_n$ ($n \in \mathbb{N}$) by the metavariables a, b, c, ... We also had that formulas can be referred to by the metavariables X, Y, Z, ... We can now further add that for each *variable* $\mathbf{x}_n$ ($n \in \mathbb{N}$), we can use the metavariables x, y, z, ...

It's important here to understand the distinction between *meta*variables and variables. Previously, we had less of a problem distinguishing between the two because we were using a language, $\mathcal{L}_0$, without variables. But now that we are using $\mathcal{L}_1$, *with* variables, we need to distinguish between the two types. We shall see how the *variables* of $\mathcal{L}_1$ function once we get to its semantics. But we can already say how its *metavariables* function. In particular, metavariables 'range over' the *symbols* of the language. In other words, if we write something like $P(x)$, it means some formula of the language *of form* $\mathfrak{P}^1_n(\mathbf{x}_k)$ for some $n, k \in \mathbb{N}$. Given this, $P(x)$ could be any one of *infinitely many* formulas of the language, *of that form*. In context, we may use this to mean *one particular* formula *of that form*, without specifying which one *exactly* we mean (e.g., in an exercise). Or we can talk about all of them at once, like when we specify that $(X \to Y) \to (\neg X \vee Y)$ is a tautology, meaning *all* formulas of this form are tautologies.

5.4 The definition of $\mathcal{L}_1$

Let's put together everything into some neater definitions.

Definition 5.4.1 (Formulas of $\mathcal{L}_1$)**.** Let:

1. $\mathsf{CON}_{\mathcal{L}_1} = \{\neg, \wedge, \vee, \rightarrow\}$, the *connectives* of $\mathcal{L}_1$;
2. $\mathsf{QUAN}_{\mathcal{L}_1} = \{\exists, \forall\}$, the *quantifiers* of $\mathcal{L}_1$;
3. $\mathsf{CONS}_{\mathcal{L}_1} = \{\mathfrak{c}_n \mid n \in \mathbb{N}\}$, the *constants* of $\mathcal{L}_1$;
4. $\mathsf{VAR}_{\mathcal{L}_1} = \{\mathbf{x}_n \mid n \in \mathbb{N}\}$, the *variables* of $\mathcal{L}_1$;
5. $\mathsf{PRED}_{\mathcal{L}_1} = \{\mathfrak{P}^n_k \mid n, k \in \mathbb{N}\}$, the *predicates* of $\mathcal{L}_1$, and;
6. $\mathsf{S}_{\mathcal{L}_1} = \{,, (,)\}$.

Let $\mathsf{ALPH}_{\mathcal{L}_1}$, the *alphabet* of $\mathcal{L}_1$, be the smallest set such that $\mathsf{S}_{\mathcal{L}_1}$, $\mathsf{QUAN}_{\mathcal{L}_1}$, $\mathsf{CON}_{\mathcal{L}_1}$, $\mathsf{CONS}_{\mathcal{L}_1}$, $\mathsf{VAR}_{\mathcal{L}_1}$, $\mathsf{PRED}_{\mathcal{L}_1} \subseteq \mathsf{ALPH}_{\mathcal{L}_1}$. Let $\mathsf{TERM}_{\mathcal{L}_1} = \{t \mid t \in \mathsf{CONS}_{\mathcal{L}_1} \text{ or } t \in \mathsf{VAR}_{\mathcal{L}_1}\}$, the terms of $\mathcal{L}_1$.

Let $\mathsf{ATOM}_{\mathcal{L}_1}$, the *atomic formulas* of $\mathcal{L}_1$, be the smallest set such that if P is a predicate of arity n in $\mathsf{PRED}_{\mathcal{L}_1}$, and $t_1, ..., t_n$ are (not necessarily distinct) terms in $\mathsf{TERM}_{\mathcal{L}_1}$, then $P(t_1, ..., t_n) \in \mathsf{ATOM}_{\mathcal{L}_1}$.

The set of (well-formed) *formulas* of $\mathcal{L}_1$ is the smallest set $\mathsf{FORM}_{\mathcal{L}_1}$ such that:

1. $\mathsf{ATOM}_{\mathcal{L}_1} \subseteq \mathsf{FORM}_{\mathcal{L}_1}$;
2. if X and Y are in $\mathsf{FORM}_{\mathcal{L}_1}$, and $x \in \mathsf{VAR}_{\mathcal{L}_1}$, then:
   (a) $\neg X \in \mathsf{FORM}_{\mathcal{L}_1}$;
   (b) $(X \wedge Y) \in \mathsf{FORM}_{\mathcal{L}_1}$;
   (c) $(X \vee Y) \in \mathsf{FORM}_{\mathcal{L}_1}$;
   (d) $(X \rightarrow Y) \in \mathsf{FORM}_{\mathcal{L}_1}$;
   (e) $\exists x Y \in \mathsf{FORM}_{\mathcal{L}_1}$, and;
   (f) $\forall x Y \in \mathsf{FORM}_{\mathcal{L}_1}$.

Definition 5.4.2 (Sentences of $\mathcal{L}_1$)**.** Let $\mathcal{F} : \mathsf{FORM}_{\mathcal{L}_1} \rightarrow \mathsf{VAR}_{\mathcal{L}_1}$ be the function defined for each $X \in \mathsf{FORM}_{\mathcal{L}_1}$ such that:

1. if X is of form $P(t_1, ..., t_n)$, then $\mathcal{F}(X) = \{x \mid x \in \mathsf{VAR}_{\mathcal{L}_1} \text{ and } x = t_k, 1 \leq k \leq n\}$;
2. if X is of form $\neg Y$, then $\mathcal{F}(X) = \{x \mid x \in \mathcal{F}(Y)\}$;

3. if X is of form $(Y \wedge Z)$, $(Y \vee Z)$, $(Y \rightarrow Z)$, then $\mathcal{F}(X) = \{x \mid x \in \mathcal{F}(Y) \text{ or } x \in \mathcal{F}(Z)\}$;

4. if $y \in \mathsf{VAR}_{\mathcal{L}_1}$ and X is of form $\forall y Z$, $\mathcal{F}(X) = \{x \mid x \in \mathcal{F}(Z) \text{ and } x \neq y\}$.

Let $\mathsf{SEN}_{\mathcal{L}_1} = \{X \mid \mathcal{F}(X) = \emptyset\}$, the set of *closed formulas* or *sentences* of $\mathcal{L}_1$. If $\mathcal{F}(X) \neq \emptyset$, we say X is an open formula, and $\mathcal{F}(X)$ is the set of variables which have at least one free occurrence in X.

Exercise 5.4.1. Read the above two definitions carefully, and try to understand every part. Then, explain in your own terms what the main aim of $\mathcal{F}$ is, and how it achieves it with this specific definition.

Definition 5.4.3. Provide a proof for the following claim:

Every atomic formula with at least one occurrence of a variable is open.

5.5 Yet another convention

Finally, here is yet another convention we shall make use of. Sometimes, we want to state exactly which variables of a *complex* formula Y have at least one free occurrence in Y. In such cases, we may write $Y(x_1, ..., x_n)$, with the list of free variables of Y occurring between the parentheses. This is useful, for example, if we write $Y(x)$, meaning that Y only has x occurring free somewhere, and then writing $\forall x Y(x)$, which immediately shows that the latter formula is now closed.

Now $Y(x)$ looks suspiciously like an open atomic formula of $\mathcal{L}_1$, and there are some clear parallels between the two, but it is still important to note that X, Y, ..., here are *not* predicates but entire complex formulas. For example, $Y(x)$ may just be something like the open formula $\forall y(P(y,x) \vee Q(y))$, and thus, $\forall x Y(x)$ would be the formula $\forall x \forall y(P(y,x) \vee Q(y))$. As expected, this latter formula is closed.

Exercise 5.5.1. Determine for each of the following formulas whether they are open or closed. In each case, explain your reasoning.

1. $\forall x_1, \forall x_2 ... \forall x_{10} X(x_1, x_2, ..., x_{10})$

2. $\forall x \exists y X(y)$

3. $\exists y X(z)$

4. $\exists x \exists y \exists z X(x, y, z)$

In connection, and this will be crucial when we introduce our first-order tableau system, we will also work with *substitutions*. Most importantly, sometimes we may want to consider formulas where the variables have been substituted for names. In fact, this is one way to turn an open formula into a closed one.

For example, take the closed formula $\forall x(P(x) \wedge Q(x))$. In the notation above, this may be represented as $\forall x Y(x)$, while the open formula $P(x) \wedge Q(x)$ may be represented as $Y(x)$. Now substituting the name a for the variable x in Y, we get $P(a) \wedge Q(a)$. You simply take each unbound occurrence of x, and replace it with the name a. We use the notation $Y(a/x)$ to denote substituting a for x. In full generality, we can say that for any formula $Y(x)$, we denote by $Y(a/x)$ the result of substituting a for each unbound occurrence of the variable x in Y.

Exercise 5.5.2. For each of the following, write the down the formula that is more concisely represented by our conventions.

1. $(P(x) \vee Q(y))(a/x)$
2. $Y(b/z)$ where $Y = \exists y(R(y, z) \wedge R(b, z))$
3. $(Q(x) \wedge \forall x R(x, a))(a/x)$
4. $((R(x, y) \vee R(y, x)) \rightarrow P(x))(a/x)(b/y)$

Chapter 6

The semantics of first-order logic

We are now ready to tackle the semantics of first-order logic. In some sense, the semantics of first-order logic is very much like the semantics of zeroth-order logic, with the exception of the newly introduced variables and the quantifiers that bind them. On the other hand, the *expressive power* of the language $\mathcal{L}_1$ is vastly more powerful than that of $\mathcal{L}_0$. Indeed, for some, first-order classical logic is *the* logical system, or at any rate, the *strongest but still well-behaved* logical system. At any rate, this added expressive power comes with added complexity, and everything starts with dealing with the semantic values of our variables.

6.1 On the values of variables

For a moment, let's forget about our logics and consider everyday mathematical practice. Suppose you have a simple equation of one unknown variable as follows:

$$(2 \times x) + 7 = 57$$

If you 'solve' this equation, you get that $x = 25$. Now really, all this means is that if you take the *value of the variable* x to be 25, then $(2 \times x) + 7 = 57$ will come out correct, while if you take *the value of the variable* x to be anything else, it will come out incorrect. Thus, one way we can talk about variables is to assign them a value and see whether the resulting formula is correct, computing with the value assigned. In such cases, the variables function like *temporary constants*. Essentially, what we are saying above is that for a moment, you should understand $(2 \times x) + 7 = 57$ the same way as you would understand $(2 \times 25) + 7 = 57$. Of course, unlike with constants, what the value of x is is temporarily assigned. For example, in $(5 \times x) + 7 = 107$, the formula will now come out true provided x is assigned the value 20, not 25. Clearly, the value of 20 and 25 did not change from one equation to the other, but the assigned value of x did, provided the aim is to make the formula correct. And of course, we can assign a value to x that makes neither of these equations come out correct. For example, if x is assigned the value 97, both equations will come out incorrect. As we shall see presently, our semantics for first-order logic will be founded on these temporary assignments of values to the variables of the language $\mathcal{L}_1$.

Ultimately, our goal is to assign meaning to quantified formulas with no free variables, i.e., quantified closed formulas, in a systematic way. This will be based on a generalization of the idea of temporary assignments of values. For example, take the claim:

$$\text{The equation } (2 \times x) + 7 = 57 \text{ has a solution.}$$

What this means is that *there is* a temporary assignment of a value to the variable x such that $(2 \times x) + 7 = 57$ comes out correct. This is clearly a use of the existential quantifier. And clearly, this is true if there really is such an assignment. Moreover, as demonstrated, there is such an assignment (assigning 25 to x), so it is true that the equation has a solution.

On the other hand, take the claim:

$$\text{The equation } (0 \times x) + 7 = 57 \text{ has a solution.}$$

This is false, and the reason why it is false is because *there is no* assignment of a value to the variable x such that $(0 \times x) + 7 = 57$.

The same type of reasoning holds for the universal quantifier. For example, for the equation $2 \times (x + 2) = (2 \times x) + 4$, *every* temporary assignment of a value to the variable x makes the equation come out correct. Incidentally, this is equivalent to the fact that *there is no* assignment of a value to the variable x which would make the equation come out incorrect.

6.2 The meaning of terms

As specified above, the terms of $\mathcal{L}_1$ are the constants and variables of $\mathcal{L}_1$. For $\mathcal{L}_0$, assigning a semantic value to any constant was done through the interpretation function of the structure $\mathbf{S} = \langle \mathbf{D}, I \rangle$. This will be retained, so that the value of a constant $\mathfrak{c}_n$ ($n \in \mathbb{N}$) relative to the structure $\mathbf{S}$, in other words (symbols), $I(\mathfrak{c}_n)$, is just some member of $\mathbf{D}$. But again, though variables have a function similar to constants, their semantic value is only temporarily assigned, and is otherwise *variable*. Accordingly, we do not use the interpretation function I to assign values to our variables, but a separate function $\mathbf{a}$ called a *variable assignment*.

The variable assignment function $\mathbf{a}$ assigns to each *variable* $\mathbf{x}_n$ ($n \in \mathbb{N}$) a member of $\mathbf{D}$, like I does for constants. But unlike with I, we will introduce a device that will allow us to change the variable assignment *inside* a structure $\mathbf{S}$. The basis of this is what is called an x-variant assignment $\mathbf{a}'$ for an existing assignment $\mathbf{a}$. Unsurprisingly, $\mathbf{a}'$ is called an x-variant assignment of $\mathbf{a}$ because it differs from it in at most what it assigns to the variable x. (It should be noted that in usual mathematical fashion, $\mathbf{a}$ may be its own (trivial) x-variant assignment, in which case nothing is changed from the initial assignment.)

Let's look at an example for this. Suppose you have a mathematical statement as follows:

$$x \times 4 < 2 \times y$$

Here, we have multiple variables, so each assignment of values to the variables will assign a value to x, and to y. Suppose we assign the value 5 to x and 10 to y. This will result in an incorrect statement, since $20 < 20$ is incorrect. So now take the x-variant assignment that assigns 4 to x, but otherwise leaves the assignment as it was. This time, we get a correct statement, since $16 < 20$. Of course, we can also take a y-variant assignment. And in general, not all x- or y-variant assignment will make the statement correct. For example, the y-variant assignment (to the initial one) where $y = 5$ will clearly make it incorrect again.

We can make the above ideas more precise as follows:

Definition 6.2.1 (Variable assignment)**.** Given a structure $\mathbf{S} = \langle \mathbf{D}, I \rangle$, the function $\mathbf{a} : \mathsf{VAR}_{\mathcal{L}_1} \rightarrow \mathbf{D}$ is a *variable assignment* in $\mathbf{D}$. If $\mathbf{a}'$ is a variable assignment in $\mathbf{D}$ just like $\mathbf{a}$, except possibly for some x, $\mathbf{a}'(x) \neq \mathbf{a}(x)$, we call $\mathbf{a}'$ an x-variant variable assignment of $\mathbf{a}$. If $\mathbf{a}'(x) = d$ ($d \in D$), i.e., $\mathbf{a}'$ is the x-variant variable assignment of $\mathbf{a}$ such that x is sent to d by $\mathbf{a}'$, we may also write $\mathbf{a}^x_d$.

Let's return to the general picture. Since some terms of the language $\mathcal{L}_1$ are variables, it won't be enough to work solely with the interpretation function I to assign semantic values to our terms in $\mathcal{L}_1$ – we need a variable assignment too. Clearly, since these terms then go into forming atomic and complex formulas, we will need to relativize the semantic values of expressions of $\mathcal{L}_1$ both to a particular structure $\mathbf{S} = \langle \mathbf{D}, I \rangle$ and to a corresponding variable assignment $\mathbf{a}$ in $\mathbf{D}$ at any one time. We will introduce some new notation for this. In particular, if X is any expression of $\mathcal{L}_1$, then its semantic value relative to a structure $\mathbf{S} = \langle \mathbf{D}, I \rangle$ *and* variable assignment $\mathbf{a}$ in $\mathbf{D}$ will be denoted:

$$I(X)[\mathbf{a}]$$

Using this notation, we can easily specify what it means to assign a semantic value to a *term* of $\mathcal{L}_1$ as follows:

Definition 6.2.2 (Term values)**.** Let $\mathbf{S} = \langle \mathbf{D}, I \rangle$ be any ordered pair such that $\mathbf{D}$ is a non-empty set and I is defined for each $c \in \mathsf{CONS}_{\mathcal{L}_1}$ so that $I(c) \in \mathbf{D}$. Let $\mathbf{a}$ be a variable assignment in $\mathbf{D}$ as above. Then, for each term $t \in \mathsf{TERM}_{\mathcal{L}_1}$, we define the value of t in the structure $\mathbf{S}$ relative to the variable assignment $\mathbf{a}$ as follows:

1. if $t = x$ for some $x \in \mathsf{VAR}_{\mathcal{L}_1}$, $I(t)[\mathbf{a}] = \mathbf{a}(t)$;

2. if $t = c$ for some $c \in \mathsf{CONS}_{\mathcal{L}_1}$, $I(t)[\mathbf{a}] = I(c)$.

Remark 6.2.1. Take some time to try and thoroughly understand the above definition. What it says is that if a term is a variable, then its value is taken care of by the variable assignment, while if it's a constant, then the interpretation function decides.

Exercise 6.2.1. Let $\mathbf{S} = \langle \mathbf{D}, I \rangle$ be such that $\mathbf{D} = \mathbb{N}$, $I(\mathfrak{c}_n) = n \times 2$, and let $\mathbf{a}(\mathbf{x}_n) = n + 7$ ($n \in \mathbb{N}$). Then, determine for each of the expressions below which natural number they stand for. In each case, specify how you reduced the calculation to one of two options, in accordance with the definition.

1. $I(\mathfrak{c}_4)[\mathbf{a}]$

2. $I(\mathbf{x}_4)[\mathbf{a}]$

3. $I(\mathbf{x}_2)[\mathbf{a}]$

4. $I(\mathbf{x}_5)[\mathbf{a}_9^{\mathbf{x}_5}]$

5. $I(\mathbf{x}_5)[\mathbf{a}_4^{\mathbf{x}_8}]$

Remark 6.2.2. Your answers should look something like this:

$$I(\mathbf{x}_1)[\mathbf{a}] = \mathbf{a}(\mathbf{x}_1) = 1 + 7 = 8.$$

> One thing you may notice here is that variables may 'see' more of the domain than the constants of a language. For example, since $I(\mathfrak{c}_n) = n \times 2$, no odd number in the domain will have a constant that refers to it. On the other hand, a variable may be assigned the value of an odd number nevertheless. In the above case, since $\mathbf{a}(\mathbf{x}_n) = n + 7$, if n is even, then the variable assignment $\mathbf{a}$ will assign it an odd number as its value. As we will see soon when we introduce quantifiers, at times, they will be able to see more of the domain than our constants, and this way, they can express more facts about a structure. Indeed, the consequences of this are at the core of many foundational theorems in logic.

6.3 On truth and satisfaction

Now that we know how to assign values to our terms relative to a structure $\mathbf{S}$ and a variable assignment $\mathbf{a}$, it is easy to see how each *quantifier-free* formula gets its value in the language $\mathcal{L}_1$. In particular, the calculations are essentially identical to those of $\mathcal{L}_0$, except if you encounter a variable (all of which will be free by assumption of the formulas being quantifier-free), you need to calculate with the assignment $\mathbf{a}$ instead of the interpretation function I.

In particular, for *atomic* formulas, of form $P^n(t_1, ..., t_n)$, their semantic value, as determined relative to $\mathbf{S} = \langle \mathbf{D}, I \rangle$ and the variable assignment $\mathbf{a}$ in $\mathbf{D}$, will just be:

$$\mathbf{S} \models P^n(t_1, ..., t_n)[\mathbf{a}] \textit{ iff } \langle I(t_1)[\mathbf{a}], ..., I(t_n)[\mathbf{a}] \rangle \in I(P)$$

In other words, for each term in our atomic formula, we check if their value under $\mathbf{S}$ and $\mathbf{a}$ is *in* the interpretation of P or not. Notice that this means most of the time, the value of an atomic formula of $\mathcal{L}_1$ with free variables relative to a structure $\mathbf{S}$ will depend on the particular variable assignment we are calculating with.

One crucial thing regarding our terminology is that technically, if the atomic formula $P^n(t_1, ..., t_n)$ is *open* because it has some (trivially, free) variables occurring in it, then what $\mathbf{S} \models P^n(t_1, ..., t_n)[\mathbf{a}]$ says is *not*, in general, that the formula is *true*. Rather, what it says is that the formula $P^n(t_1, ..., t_n)$ is *satisfiable* in $\mathbf{S}$, and specifically, satisfied in $\mathbf{S}$ under $\mathbf{a}$. The formula cannot be called *true* because relative to some other assignment $\mathbf{b}$, it may *not* be the case that $\mathbf{S} \models P^n(t_1, ..., t_n)[\mathbf{b}]$.

Indeed, *truth* in a structure $\mathbf{S}$ will be defined just as satisfaction in $\mathbf{S}$ under *every* variable assignment $\mathbf{a}$ in $\mathbf{D}$. And incidentally, sentences will get their value independent of any particular assignment $\mathbf{a}$ so they will all be *truth-apt*; either true or false in a structure. This is partly why sentences are so crucial. Because we are interested (as of now) in *truth*, and not *satisfaction under an assignment*.

Accordingly, to say that a formula X is *true* in $\mathbf{S}$, we will suppress the notation $[\mathbf{a}]$ (as by definition, it is irrelevant), and write:

$$\mathbf{S} \models X$$

If we want to say that a formula X is *satisfied* in $\mathbf{S}$ under $\mathbf{a}$, we write:

$$\mathbf{S} \models X[\mathbf{a}]$$

You can think of the distinction between satisfaction under an assignment and truth as follows. If you say something like "x is tall", you cannot really say that this is either a true or false statement as it is. Clearly, *if* we understand 'x' as former professional basketball player Yao Ming (height: 7' 6"), it would be 'true', and *if* we understand 'x' as movie star Danny DeVito (height: 4'10"), it would be 'false'. But in itself, it is neither true nor false, because x may stand for anything! On the other hand, the sentence "Yao Ming is tall" *is* true, because he *is* tall independent of how we understand 'x' (because it is not even part of the sentence).

For example, let $\mathbf{S} = \langle \mathbf{D}, I \rangle$ be such that $\mathbf{D} = \mathbb{N}$, $I(\mathfrak{c}_n) = n \times 2$, and let $\mathbf{a}(\mathbf{x}_n) = n + 7$ ($n \in \mathbb{N}$) as before. Let $I(\mathfrak{P}^3_1) = \{\langle n, j, k \rangle \mid n + j = k\}$, and $I(\mathfrak{P}^3_2) = \{\langle n, j, k \rangle \mid n \times j = k\}$. Then, consider an atomic formula, such as:

$$\mathfrak{P}^3_1(\mathfrak{c}_1, \mathfrak{c}_3, \mathbf{x}_5)$$

Is the above formula satisfied in $\mathbf{S}$ under to the assignment $\mathbf{a}$? Well, $I(\mathfrak{c}_1)[\mathbf{a}] = 2$, $I(\mathfrak{c}_3)[\mathbf{a}] = 6$, and $I(\mathbf{x}_5)[\mathbf{a}] = 12$. On the other hand, $2 + 6 \neq 12$, so this triple is not in $I(\mathfrak{P}^3_1)$. Thus:

$$\mathbf{S} \not\models \mathfrak{P}^3_1(\mathfrak{c}_1, \mathfrak{c}_3, \mathbf{x}_5)[\mathbf{a}]$$

Of course, as we discussed above, a well-chosen $\mathbf{x}_5$-variant of $\mathbf{a}$ may very well satisfy this formula in $\mathbf{S}$ relative to it. Of course, this is easy to specify with our explicit notation:

$$\mathbf{S} \models \mathfrak{P}^3_1(\mathfrak{c}_1, \mathfrak{c}_3, \mathbf{x}_5)[\mathbf{a}^{\mathbf{x}_5}_8]$$

Clearly, we can also consider:

$$\mathbf{S} \models \mathfrak{P}^3_1(\mathfrak{c}_1, \mathfrak{c}_3, \mathbf{x}_1)[\mathbf{a}].$$

And in fact, with a different change in our formula, we get:

$$\mathbf{S} \models \mathfrak{P}^3_2(\mathfrak{c}_1, \mathfrak{c}_3, \mathbf{x}_5)[\mathbf{a}]$$

Exercise 6.3.1. Explain in detail, demonstrating the calculations at each turn, why it is the case that:

$$\mathbf{S} \models \mathfrak{P}^3_1(\mathfrak{c}_1, \mathfrak{c}_3, \mathbf{x}_1)[\mathbf{a}] \tag{6.1}$$

$$\mathbf{S} \models \mathfrak{P}^3_2(\mathfrak{c}_1, \mathfrak{c}_3, \mathbf{x}_5)[\mathbf{a}] \tag{6.2}$$

$$\mathbf{S} \not\models \mathfrak{P}^3_2(\mathfrak{c}_1, \mathfrak{c}_3, \mathbf{x}_5)[\mathbf{a}^{\mathbf{x}_5}_8] \tag{6.3}$$

$$\mathbf{S} \not\models \mathfrak{P}_1^3(\mathfrak{c}_1, \mathfrak{c}_3, \mathbf{x}_1)[\mathbf{a}_{12}^{\mathbf{x}_1}] \tag{6.4}$$

Remark 6.3.1. Here is what your answer should look like for each of the above. Let's take $\mathbf{S} \models \mathfrak{P}_1^3(c_3, c_2, x_3)[\mathbf{a}]$.

First, by definition, $\mathbf{S} \models \mathfrak{P}_1^3(c_3, c_2, x_3)[\mathbf{a}]$ *iff* $\langle I(c_3)[\mathbf{a}], I(c_2)[\mathbf{a}], I(x_3)[\mathbf{a}]\rangle \in I(\mathfrak{P}_1^3)$. Then, you need to calculate the value of each term, and check it against the interpretation of the predicate. In succession:

1. $I(c_3)[\mathbf{a}] = I(c_3) = 3 \times 2 = 6$;

2. $I(c_2)[\mathbf{a}] = I(c_2) = 2 \times 2 = 4$;

3. $I(x_3)[\mathbf{a}] = \mathbf{a}(x_3) = 3 + 7 = 10$.

Thus, the question is whether $\langle 6, 4, 10\rangle \in I(\mathfrak{P}_1^3)$. By definition, $\langle 6, 4, 10\rangle \in I(\mathfrak{P}_1^3)$ *iff* $6 + 4 = 10$, which is the case, so $\langle 6, 4, 10\rangle \in I(\mathfrak{P}_1^3)$. Thus, $\mathbf{S} \models \mathfrak{P}_1^3(c_3, c_2, x_3)[\mathbf{a}]$.

As far as the connectives go, there is no change in how their meaning is specified relative to their less complex constituents, except again, we are calculating with both I and $\mathbf{a}$ at each turn.

Thus, we get that:

1. $\mathbf{S} \models \neg X[\mathbf{a}]$ if, and only if, $\mathbf{S} \not\models X[\mathbf{a}]$;

2. $\mathbf{S} \models (X \wedge Y)[\mathbf{a}]$ if, and only if, $\mathbf{S} \models X[\mathbf{a}]$ and $\mathbf{S} \models Y[\mathbf{a}]$;

3. $\mathbf{S} \models (X \vee Y)[\mathbf{a}]$ if, and only if, $\mathbf{S} \models X[\mathbf{a}]$ or $\mathbf{S} \models Y[\mathbf{a}]$ (or both);

4. $\mathbf{S} \models (X \to Y)[\mathbf{a}]$ if, and only if, if $\mathbf{S} \models X[\mathbf{a}]$, then $\mathbf{S} \models Y[\mathbf{a}]$.

Exercise 6.3.2. Provide two complex formulas distinct from those given in Exercise 6.3.1 that are satisfied under $\mathbf{a}$, and similarly, provide two complex formulas distinct from them that are *unsatisfied* under $\mathbf{a}$ in the structure specified. In each case, provide a detailed derivation to show why that is the case.

Remark 6.3.2. Your answers should look similar here to your answers in Exercise 6.3.1, except now you have to include the calculations for the connectives too. Take for example $\neg\mathfrak{P}_2^3(c_1, x_4, c_7) \vee \mathfrak{P}_1^3(x_9, x_1, x_4)$. Is it the case that $\mathbf{S} \models \neg\mathfrak{P}_2^3(c_1, x_4, c_7) \vee \mathfrak{P}_1^3(x_9, x_1, x_4)[\mathbf{a}]$?

Well, $\mathbf{S} \models \neg\mathfrak{P}_2^3(c_1, x_4, c_7) \vee \mathfrak{P}_1^3(x_9, x_1, x_4)[\mathbf{a}]$ *iff* $\mathbf{S} \models \neg\mathfrak{P}_2^3(c_1, x_4, c_7)[\mathbf{a}]$ or $\mathbf{S} \models \mathfrak{P}_1^3(x_9, x_1, x_4)[\mathbf{a}]$. And $\mathbf{S} \models \neg\mathfrak{P}_2^3(c_1, x_4, c_7)[\mathbf{a}]$ *iff* $\mathbf{S} \not\models \mathfrak{P}_2^3(c_1, x_4, c_7)[\mathbf{a}]$.

Let us consider the simpler case first, that of $\mathfrak{P}_1^3(x_9, x_1, x_4)$. Now $\mathbf{S} \models \mathfrak{P}_1^3(x_9, x_1, x_4)[\mathbf{a}]$ *iff* $\langle I(x_9)[\mathbf{a}], I(x_1)[\mathbf{a}], I(x_4)[\mathbf{a}]\rangle \in I(\mathfrak{P}_1^3)$. Taking each term in succession:

1. $I(x_9)[\mathbf{a}] = \mathbf{a}(x_9) = 9 + 7 = 16$;

2. $I(x_1)[\mathbf{a}] = \mathbf{a}(x_1) = 1 + 7 = 8$;

3. $I(x_4)[\mathbf{a}] = \mathbf{a}(x_4) = 4 + 7 = 11$.

Substituting back, $\langle 16, 8, 11\rangle \notin I(\mathfrak{P}^3_1)$, because of course, 16+8 $\neq$ 11. So $\mathbf{S} \not\models \mathfrak{P}^3_1(x_9, x_1, x_4)[\mathbf{a}]$.

For the other side of the disjunction, we need to consider whether $\mathbf{S} \not\models \mathfrak{P}^3_2(c_1, x_4, c_7)[\mathbf{a}]$ is the case. Again, $\mathbf{S} \not\models \mathfrak{P}^3_2(c_1, x_4, c_7)[\mathbf{a}]$ *iff* $\langle I(c_3)[\mathbf{a}], I(x_4)[\mathbf{a}], I(c_7)[\mathbf{a}]\rangle \notin I(\mathfrak{P}^3_2)$. We then calculate each term value:

1. $I(c_3)[\mathbf{a}] = I(c_3) = 3 \times 2 = 6$;

2. $I(x_4)[\mathbf{a}] = \mathbf{a}(x_4) = 4 \times 2 = 8$;

3. $I(c_7)[\mathbf{a}] = I(c_7) = 7 + 7 = 14$.

Substituting back, we get $\langle 6, 8, 14\rangle$, which is not in $I(\mathfrak{P}^3_2)$, because $6 \times 8 \neq 14$. Thus, $\mathbf{S} \not\models \mathfrak{P}^3_2(c_1, x_4, c_7)[\mathbf{a}]$. So it is the case that $\mathbf{S} \models \neg\mathfrak{P}^3_2(c_1, x_4, c_7)[\mathbf{a}]$.

So $\mathbf{S} \models \neg\mathfrak{P}^3_2(c_1, x_4, c_7) \vee \mathfrak{P}^3_1(x_9, x_1, x_4)[\mathbf{a}]$.

6.4 On quantifiers and bound variables

We now know how to deal with any formula of $\mathcal{L}_1$ that does not have any quantifiers occurring in it. As we just saw, in such cases, the assignment function does not do much, and you may very well wonder why there is even an assignment function distinct from an interpretation function when they function very much the same.

Well, the reason why we do need a separate variable assignment function over and above the interpretation function for constants and predicates is because sometimes, we want to actually *vary* the variable assignment, *without* varying the interpretation of the constants and the predicates. And indeed, using quantifiers and binding variables is a way to systematically incorporate these variations into the meaning of complex expressions.

How to do this was already briefly sketched above. Let's look at it in the simplest case, now from a more formal point of view. First, take $\exists x P(x)$, where x is a a variable and P is a one-place predicate. You can read this as "There exists an x such that $P(x)$", or "There is an x such that x is P", or some variation of the above. At any rate, what this formula (and indeed, sentence) says is that *there is* a way of assigning a value to the variable x (there is an assignment) such that $P(x)$ is satisfied *under that assignment*. In other words, it says that there is a member of the domain that is in the interpretation of P. Or in yet other words, the interpretation of P is *non-empty*. Put as before, it might also be taken to say that $P(x)$ has a solution.

Suppose $\mathbf{S} = \langle \mathbf{D}, I \rangle$ is a structure with (non-empty) domain $\mathbf{D}$ being the set of all living things on the planet (at this moment), and $I(P)$ is a subset of the domain consisting of all pandas. Then, we can ask: is the *sentence* $\exists x P(x)$ *true* in $\mathbf{S}$? Well, the answer is yes if there is an assignment $\mathbf{a}$ that renders $P(x)$ satisfied under $\mathbf{S}$ and $\mathbf{a}$, and otherwise, the answer is no. Thankfully, at the moment of writing this book (and hopefully, at the moment of reading it), there are pandas among the living things on the planet. Thus, $I(P)$ is a non-empty set, and so there should be an assignment that manages to assign an actual panda to x, thereby satisfying $P(x)$ in $\mathbf{S}$.

Figure 6.1: The giant panda Tián Tián (pictured above) is in the set $I(P)$ — The Land, CC BY-SA 3.0, via Wikimedia Commons

For example, in Figure 6.4, it is noted that the panda Tián Tián $\in I(P)$. Thus, we *can* find an assignment that makes $P(x)$ satisfied in $\mathbf{S}$, namely, any assignment that assigns Tián Tián to x. In fact, making use of x-variance, we can say that given *any* assignment $\mathbf{a}$, the x-variant assignment $\mathbf{a}^x_{\text{Tián Tián}}$ satisfies $P(x)$ in $\mathbf{S}$. In other words, $\mathbf{S} \models P(x)[\mathbf{a}^x_{\text{Tián Tián}}]$, since $I(x)[\mathbf{a}^x_{\text{Tián Tián}}] = \mathbf{a}^x_{\text{Tián Tián}}(a) = \text{Tián Tián}$ and Tián Tián $\in I(P)$.

Now remember that what we were evaluating originally is whether $\mathbf{S} \models \exists x P(x)$, i.e., whether $\exists x P(x)$ is true in $\mathbf{S}$. And we said that it *is* true *if* we can find and appropriate assignment under which $P(x)$ is satisfied in $\mathbf{S}$. Since we *have* found such an assignment,

$\mathbf{S} \models \exists x P(x)$. Note that this is really just a precise way of saying: there is a panda!

Clearly, this is a lot of words, and as we have seen, in formal logic, it is possible to say a lot of things in very few words (symbols). In general, what we can say is the following:

$\mathbf{S} \models \exists x P(x)[\mathbf{a}]$ *iff* there is an x-variant assignment $\mathbf{a}'$ such that $\mathbf{S} \models P(x)[\mathbf{a}']$.

Clearly, our previous reasoning does conform to this definition, since $\mathbf{a}^x_{\text{Tián Tián}}$ is such an assignment, as we have shown.

Now let's see the same type of reasoning with the universal quantifier $\forall$. In fact, we can take the formula $\forall x P(x)$ relative to the structure $\mathbf{S}$, as before. In this case, the sentence can be read "For every x, $P(x)$", or "Every x is P", or some variant of this. Now if $\exists x P(x)$ meant that there is an assignment under which $P(x)$ is satisfied in $\mathbf{S}$, then $\forall x P(x)$ says under *every* assignment, $P(x)$ is satisfied in $\mathbf{S}$. Again, we can express what $\forall x P(x)$ says in various ways. For example, it says that *every* member of the domain is in the interpretation of P, or that the domain $D = I(P)$, or that $P(x)$ has *only* solutions in $\mathbf{S}$. Or again, it says: everything is a panda!

Clearly, not everything is a panda in general, but focusing on $\mathbf{S}$, it is also not true that every living thing is a panda. Thus, it should be the case that $\forall x P(x)$ comes out false in $\mathbf{S}$. And indeed, this is the case. First, we have the general case where:

$\mathbf{S} \models \forall x P(x)[\mathbf{a}]$ *iff* for every x-variant assignment $\mathbf{a}'$, $\mathbf{S} \models P(x)[\mathbf{a}']$.

But again, is it true that for every x-variant assignment $\mathbf{a}'$, $\mathbf{S} \models P(x)[\mathbf{a}']$? Clearly not. For example, among the currently living things in the world, there is Kanzi the bonobo – that is, Kanzi $\in D$. Now Kanzi is not a panda, so Kanzi $\notin I(P)$. So if we take an x-variant assignment $\mathbf{a}^x_{\text{Kanzi}}$ (no matter what $\mathbf{a}$ was initially), then $P(x)$ is clearly not satisfied in $\mathbf{S}$. That is, $\mathbf{S} \not\models P(x)[\mathbf{a}^x_{\text{Kanzi}}]$. So it is not the case that every x-variant assignment $\mathbf{a}'$ is such that $\mathbf{S} \models P(x)[\mathbf{a}']$. So $\mathbf{S} \not\models \forall x P(x)[\mathbf{a}]$, as expected. In other words, $\forall x P(x)$ is false – not every living thing is a panda.

To reiterate what we have seen here, when we have $\exists x P(x)$ true in a structure, it means we can find *at least one* assignment under which $P(x)$ is satisfied, while if we have $\forall x P(x)$ true in a structure, it means we can find that *every* assignment of a value to x satisfies $P(x)$.

Exercise 6.4.1. For now, let's stay with our structure $\mathbf{S}$ of living things in the world with the predicate P standing for pandas as specified. Then, think of how you would decide whether the following are true:

1. $\mathbf{S} \models \neg\exists x P(x)[\mathbf{a}]$;

2. $\mathbf{S} \models \neg\exists x \neg P(x)[\mathbf{a}]$;

3. $\mathbf{S} \models \exists x \neg P(x)[\mathbf{a}]$;

4. $\mathbf{S} \models \neg\forall x P(x)[\mathbf{a}]$;

5. $\mathbf{S} \models \neg\forall x \neg P(x)[\mathbf{a}]$;

6. $\mathbf{S} \models \forall x \neg P(x)[\mathbf{a}]$.

6.5 On truth and satisfaction, again

Above, we have noted that sentences are either true or false relative to a structure, while open formulas (which are not sentences) are satisfied or not relative to a structure *and* an assignment. But we also saw that assignments are used in the calculations when we are dealing with quantified sentences. What gives?

In fact, there is no mistake here. How we bridge this apparent problem is by saying that a *sentence* (whether quantified or not) is *true* relative to a structure provided it is satisfied under *every* assignment (for that structure), no matter what. In other words, no matter what assignment you start out with, the sentence comes out satisfied under that assignment relative to the structure. This is trivially true for sentences *without* quantifiers, since there, assignments are completely superfluous. But as it turns out, the *initial* assignment is also completely superfluous when it comes to sentences *with* quantifiers.

We can return to our example $\exists x P(x)$ relative to the structure above. Is it the case that $\mathbf{S} \models P(x)$, regardless of the assignment? In fact, it is, since no matter which initial assignment we take, there will always be an x-variant assignment that sends x to Tián Tián. In particular, for *any* initial assignment $\mathbf{a}$, we can just take $\mathbf{a}^x_{\text{Tián Tián}}$, which will satisfy $P(x)$ relative to $\mathbf{S}$. And the same goes for the universal quantified sentence $\forall x P(x)$ (and falsity).

Another way to think about this is to note that every assignment assigns a value to every variable, not just x (or y, or whatever). But when it comes to a quantifier with a variable x, like $\exists x$ or $\forall x$, we are only interested in the possible values x may take, which is where x-variant assignments come into the picture. But we can just disregard which values the other variables get, because it is irrelevant for our concerns. So again, the initial assignment drops out of the picture.

These considerations are why, as was briefly noted above, if we are considering a sentence X relative to a structure $\mathbf{S}$, $\mathbf{S} \models X$ or $\mathbf{S} \not\models X$ (without specifying an assignment), and we can say that X is either *true* in $\mathbf{S}$ or it is *false* in $\mathbf{S}$. Thus, we can say:

A formula X is true in $\mathbf{S}$, written $\mathbf{S} \models X$, provided under every assignment $\mathbf{a}$, $\mathbf{S} \models X[\mathbf{a}]$. That is, provided under every assignment $\mathbf{a}$, X is satisfied under $\mathbf{a}$ relative to $\mathbf{S}$.

And a formula X is false in $\mathbf{S}$, written $\mathbf{S} \not\models X$, provided under every assignment $\mathbf{a}$, $\mathbf{S} \not\models X[\mathbf{a}]$. That is, provided under every assignment $\mathbf{a}$, X is not satisfied under $\mathbf{a}$ relative to $\mathbf{S}$.

6.6 Dealing with multiple quantifiers

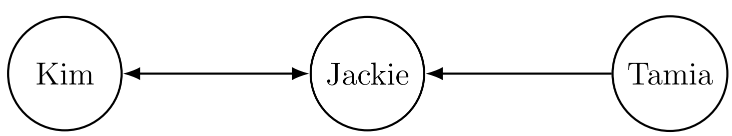
Let's jump a step up in complexity. So far, we considered quantified sentences in which only a single quantifier occurred. But of course, our language has more elaborate formulas in which several quantifiers can occur in various places. This introduces a lot of complexity, and the need to be very careful when calculating with these.

There is a prototypical example here that logicians *love* talking about; the example of a two-place predicate L, glossed as the relation 'loves' for simplicity.

More precisely, let $\mathbf{D} = \{Jackie, Kim, Tamia\}$ be the domain of the structure $\mathbf{S}$, and let $I(L) = \{\langle \mathbf{c}, \mathbf{d} \rangle \mid \mathbf{c} \text{ loves } \mathbf{d}\}$. So $I(L)$ is the set of all pairs of people $\mathbf{c}$, $\mathbf{d}$, such that $\mathbf{c}$ loves $\mathbf{d}$. Of course, this is a made up example, so let's specify that

$$I(L) = \{\langle \text{Jackie}, \text{Kim} \rangle, \langle \text{Kim}, \text{Jackie} \rangle, \langle \text{Tamia}, \text{Jackie} \rangle\}.$$

In other words, Jackie loves Kim, Kim loves Jackie, Tamie loves Jackie, but Jackie does not love Tamia, and Kim and Tamia do not love each other. This can also be represented with the following graph:



Given our language, we can then express simple things like:

$$\exists x \exists y L(x, y) \tag{6.5}$$

$$\exists x \forall y L(x, y) \tag{6.6}$$

$$\forall x \exists y L(x, y) \tag{6.7}$$

$$\forall x \forall y L(x, y) \tag{6.8}$$

And as it turns out, all of these mean completely different things! In order to see this, however, we need to see how to go on calculating, once we passed one quantifier, and moved to another.

6.6.1 Existential-existential

In the simplest case, we have $\exists x \exists y L(x, y)$. This may be read as "There is an x and a y such that x loves y", or in other words, there is at least one pair of people such that one loves the other.

As usual, the question is whether $\exists x \exists y L(x, y)$ is true in $\mathbf{S}$ or not. That is, is it the case that $\mathbf{S} \models \exists x \exists y L(x, y)$? By the above, we know that this should come out true, but how?

The reasoning is a simple generalization of the previous one. Previously, for atomic formulas with a unary predicate, we had:

$\mathbf{S} \models \exists x P(x)[\mathbf{a}]$ *iff* there is an x-variant assignment $\mathbf{a}'$ such that $\mathbf{S} \models P(x)[\mathbf{a}']$.

We can generalize this to *any* formula Y of $\mathcal{L}_1$ so that it says:

$\mathbf{S} \models \exists x Y[\mathbf{a}]$ *iff* there is an x-variant assignment $\mathbf{a}'$ such that $\mathbf{S} \models Y[\mathbf{a}']$.

Note that Y here may be *any* formula whatsoever, including one that already has quantifiers in it. This allows us to calculate our way through multiple quantifiers in a formula.

As we discussed before, semantic rules for formulas are always formulated in parallel to the syntactic rules. Since we have a syntactic rule that says for any formula Y, you can put $\exists x$ ($\forall x$) in front to get a quantified formula $\exists x Y$ ($\forall x Y$), we needed a semantic rule in just those terms. The above is exactly that, in terms of the existential quantifier. Thus, once more, our semantic calculations will stepwise follow the steps of the syntactic breakdown of any formula.

Let's return to our sentence $\exists x \exists y L(x, y)$. As mentioned, this should come out true. Let's see how this comes about using our definitions. First, take any $\mathbf{a}$ whatsoever, since this is a sentence. Then, let's see when it is the case that $\mathbf{S} \models \exists x \exists y L(x, y)[\mathbf{a}]$. Well, by the above, we have that:

$\mathbf{S} \models \exists x \exists y L(x, y)[\mathbf{a}]$ *iff* there is an x-variant assignment $\mathbf{a}'$ such that $\mathbf{S} \models \exists y L(x, y)[\mathbf{a}']$.

Note that we got rid of the first quantifier. However, we are now working with a hypothesis, choosing a value for x that we think will make the resulting formula $\exists y L(x,y)$ satisfied under $\mathbf{a}'$ (and not $\mathbf{a}$ anymore). Since we see the definition of I as regards L, we know that Jackie and Kim is a good pair to choose, since by $I(L)$, they love each other (and specifically, Jackie loves Kim). So let's choose the x-variant assignment $\mathbf{a}^x_{\text{Jackie}}$. Then, we need to plug these new values into the definition to calculate again. In particular:

> $\mathbf{S} \models \exists y L(x,y)[\mathbf{a}^x_{\text{Jackie}}]$ *iff* there is a y-variant (!) assignment $(\mathbf{a}^x_{\text{Jackie}})'$ such that $\mathbf{S} \models L(x,y)[(\mathbf{a}^x_{\text{Jackie}})']$.

Again, the hypothesis is that if x evaluates to Jackie, then we should find a value for y such that $L(x,y)$ becomes satisfied in $\mathbf{S}$. Of course, this happens precisely when y evaluates to Kim. So we take the y-variant assignment of $\mathbf{a}^x_{\text{Jackie}}$ (and not $\mathbf{a}$!) that still assigns Jackie to x, but now also explicitly assigns Kim to y. We can represent this as $(\mathbf{a}^x_{\text{Jackie}})^y_{\text{Kim}}$. This is the assignment that is just like $\mathbf{a}$ except x is assigned Jackie, and y is assigned Kim. Thus, we have that:

$$\mathbf{S} \models L(x,y)[(\mathbf{a}^x_{\text{Jackie}})^y_{\text{Kim}}].$$

Since there are no more quantifiers to deal with, and there were no connectives to begin with, this is where we bottom out. Thus, it is shown that $\mathbf{S} \models \exists x \exists y L(x,y)[\mathbf{a}]$.

Here are the steps again, in succession:

$$\begin{aligned}
\mathbf{S} &\models \exists x \exists y L(x,y)[\mathbf{a}] \\
\mathbf{S} &\models \exists y L(x,y)[\mathbf{a}^x_{\text{Jackie}}] \\
\mathbf{S} &\models L(x,y)[(\mathbf{a}^x_{\text{Jackie}})^y_{\text{Kim}}].
\end{aligned}$$

Again, as you can see, we are working parallel to the syntactic analysis of the formula, namely, taking off a quantifier at each turn until we get to the atomic formula. And semantically, what we are trying to make sure at each turn is to have the final (now open) formula satisfied under the new assignment we constructed.

Thinking in terms of mathematics, you may think of this as finding a solution to the final expression $L(x,y)$ by finding an appropriate value for x, and for y, that makes $L(x,y)$ come out correct. And thinking intuitively, we just found two people in the domain of the structure such that one loves the other, and derived that the sentence "There are two people such that one loves the other" is therefore true relative to the structure.

6.6.2 Existential-universal

When it comes to the use of the universal quantifier, things get more interesting. In the case of the existential quantifier, we only had to look at single solutions. But when it comes to the universal quantifier, we have to check whether everything is a solution. Again, we can generalize the previous definition for the universal to cover every formula as such:

> $\mathbf{S} \models \forall x Y[\mathbf{a}]$ *iff* for every x-variant assignment $\mathbf{a}'$, $\mathbf{S} \models Y[\mathbf{a}']$.

As you can see, this says that $\forall x Y$ is satisfied in $\mathbf{S}$ relative to $\mathbf{a}$ *iff* *every* x-variant assignment $\mathbf{a}'$ satisfies Y in $\mathbf{S}$. This gets more complicated when we examine how this relates to the other quantifiers in a certain formula. Of course, in general, the calculation steps are always given by the definitions, but carrying them out correctly is crucial.

Let's take the next formula, then, $\exists x \forall y L(x, y)$. Using our usual gloss, this means that there is *someone* in our domain such that they love *everyone*. Thus, if we are trying to figure out whether this is true or false, we need to consider every person, and see whether any of them is such that they love everyone. As it turns out, this is *false*. In fact, you have to be careful here, since loving *everyone* would mean that they love *themselves* too! By that fact alone, there is no one in the domain that loves everyone. Why is this case?

Well, again, we add an arbitrary assignment $\mathbf{a}$ to start with, and get to the instance of the $\exists$-definition:

> $\mathbf{S} \models \exists x \forall y L(x, y)[\mathbf{a}]$ *iff* there is an x-variant assignment $\mathbf{a}'$ such that $\mathbf{S} \models \forall y L(x, y)[\mathbf{a}']$.

Thus, we need to find a value for x that would make the open formula $\forall y L(x, y)$ satisfied in the structure $\mathbf{S}$, which in turn would mean the open formula $L(x, y)$ satisfied for *every* value of y in turn. Since this is a finite structure, this can be checked mechanically, by taking each value one by one.

In particular, any possible solution would need to assign a value to x to start. Plugging these into our $\forall$-definition, we have that:

1. $\mathbf{S} \models \forall y L(x, y)[\mathbf{a}^x_{\text{Jackie}}]$ *iff* for every y-variant assignment $(\mathbf{a}^x_{\text{Jackie}})'$, $\mathbf{S} \models L(x, y)[(\mathbf{a}^x_{\text{Jackie}})']$;
2. $\mathbf{S} \models \forall y L(x, y)[\mathbf{a}^x_{\text{Kim}}]$ *iff* for every y-variant assignment $(\mathbf{a}^x_{\text{Kim}})'$, $\mathbf{S} \models L(x, y)[(\mathbf{a}^x_{\text{Kim}})']$;
3. $\mathbf{S} \models \forall y L(x, y)[\mathbf{a}^x_{\text{Tamia}}]$ *iff* for every y-variant assignment $(\mathbf{a}^x_{\text{Tamia}})'$, $\mathbf{S} \models L(x, y)[(\mathbf{a}^x_{\text{Tamia}})']$.

The next question is if we can make the right hand side of *any* of these true. And the answer, again, is no. To each of these, there is a counterexample. In particular, each counterexample shows that for every choice of x, we can find a value for y such that x does *not* love y, thereby making sure that there is no one that loves everyone.

In particular, we have that:

1. $\mathbf{S} \not\models L(x, y)[(\mathbf{a}^x_{\text{Jackie}})^y_{\text{Jackie}}]$;
2. $\mathbf{S} \not\models L(x, y)[(\mathbf{a}^x_{\text{Kim}})^y_{\text{Kim}}]$;
3. $\mathbf{S} \not\models L(x, y)[(\mathbf{a}^x_{\text{Tamia}})^y_{\text{Tamia}}]$.

In other words, Jackie dos not love Jackie, Kim does not love Kim, Tamia does not love Tamia, and therefore, there is no one that loves everyone. So we can conclude that:

$$\mathbf{S} \not\models \exists x \forall y L(x, y).$$

Before moving forward, we should note that the above way of checking the truth of a quantified sentence is not always possible, since some structures are infinite. For example, if $\mathbf{S}' = \langle \mathbb{N}, I \rangle$, where $I(L) = \{\langle j, k \rangle \mid j \leq k\}$, then it is true that $\exists x \forall y L(x, y)$. That is, there is a particular natural number that is less than or equal to *every* other natural number. That number is 0. The problem is that the relevant instance of this is:

$$\mathbf{S} \models \forall y L(x, y)[\mathbf{a}^x_0] \textit{ iff } \text{for every } y\text{-variant assignment } (\mathbf{a}^x_0)', \mathbf{S} \models L(x, y)[(\mathbf{a}^x_0)']$$

The above is clearly true, since 0 is less than or equal to every number (including itself!). But you cannot *mechanically* check this for *every* number one by one, since there are infinitely many of them! Thankfully, mathematicians have ways of proving facts about infinitely many things that do not require an infinite amount of time to carry out. We shall not go into this, and freely refer to facts like 0 is less than or equal to every other number.

6.6.3 Universal-existential

Now let's consider the formula $\forall x \exists y L(x, y)$. Does this mean the same thing as $\exists x \forall y L(x, y)$? In fact, it does not! Note that one says that there is an x which is in the relation L for *every* y, and the other one says every x is in the relation L with *some* y. These two are not, in general, equivalent. For example, if we read L as 'loves', $\exists x \forall y L(x, y)$ says, as we have just seen, that there is someone who loves everyone. On the other hand, if we take $\forall x \exists y L(x, y)$, this says that everyone loves someone. In fact, if we check the second sentence, unlike the first, it will come out *true* in $\mathbf{S}$!

Here, we start with the $\forall$-definition, since the first quantifier is a universal. Thus:

$$\mathbf{S} \models \forall x \exists y L(x,y)[\mathbf{a}] \textit{ iff } \text{for every } x\text{-variant assignment } \mathbf{a}', \mathbf{S} \models \exists y L(x,y)[\mathbf{a}'].$$

Again, we need to continue with the right hand side. In particular, we need to check whether for every choice of value for x, there is a choice of y for which $L(x, y)$ is satisfiable in $\mathbf{S}$.

We only have three people in our domain, so checking for every value of x is not a *huge* pain. In particular, we need to check each of:

1. $\mathbf{S} \models \exists y L(x,y)[\mathbf{a}^x_{\text{Jackie}}]$ *iff* there is a y-variant assignment $(\mathbf{a}^x_{\text{Jackie}})'$ such that $\mathbf{S} \models L(x,y)[(\mathbf{a}^x_{\text{Jackie}})']$;
2. $\mathbf{S} \models \exists y L(x,y)[\mathbf{a}^x_{\text{Kim}}]$ *iff* there is a y-variant assignment $(\mathbf{a}^x_{\text{Kim}})'$ such that $\mathbf{S} \models L(x,y)[(\mathbf{a}^x_{\text{Kim}})']$;
3. $\mathbf{S} \models \exists y L(x,y)[\mathbf{a}^x_{\text{Tamia}}]$ *iff* there is a y-variant assignment $(\mathbf{a}^x_{\text{Tamia}})'$ such that $\mathbf{S} \models L(x,y)[(\mathbf{a}^x_{\text{Tamia}})']$.

Notice that unlike before, we want each of the right hand sides to hold, since we need to consider *every* possible value for x in the domain. On the other hand, for each of the three, we only need to find a single value for y for which $L(x, y)$ is satisfied. This can be done easily:

1. $\mathbf{S} \models L(x,y)[(\mathbf{a}^x_{\text{Jackie}})^y_{\text{Kim}}]$;
2. $\mathbf{S} \models L(x,y)[(\mathbf{a}^x_{\text{Kim}})^y_{\text{Jackie}}]$;
3. $\mathbf{S} \models L(x,y)[(\mathbf{a}^x_{\text{Tamia}})^y_{\text{Jackie}}]$.

Thus, for *every* choice of x, there is a choice for y such that $L(x, y)$ is satisfied in $\mathbf{S}$. So as expected:

$$\mathbf{S} \models \forall x \exists y L(x,y).$$

6.6.4 Universal-universal

If we take our calculations stepwise, it is the formula $\forall x \forall y L(x, y)$ that will take the most amount of time, since we need to check $L(x, y)$ in $\mathbf{S}$ for *every* pair of values x and y. Since we have three people in our domain, this will mean checking $L(x, y)$ for 9 pairs of values (an exponential growth!). For what the sentence $\forall x \forall y L(x, y)$ says relative to $\mathbf{S}$ is that *everyone* loves *everyone*. This means that for each person, we have to check that person's relation to every other person.

Of course, as you may see, this sentence is *false* in $\mathbf{S}$. In fact, for every person, there are two people whom they do not love. Kim does not love Tamia, Jackie does not love Tamia, and Tamia does not love Kim, and nobody loves themselves. Any one of these is sufficient to conclude that not everyone loves everyone. But if we want to check it directly, we need to consider all 9 y-variant-of-the-x-variant-of-$\mathbf{a}$. We can represent all these assignments in three subsequent graphs for each choice of x, to make it more vivid. Thus:

$$\mathbf{a}^x_{\text{Jackie}}$$

$$(\mathbf{a}^x_{\text{Jackie}})^y_{\text{Jackie}} \quad (\mathbf{a}^x_{\text{Jackie}})^y_{\text{Kim}} \quad (\mathbf{a}^x_{\text{Jackie}})^y_{\text{Tamia}}$$

$$\mathbf{a}^x_{\text{Kim}}$$

$$(\mathbf{a}^x_{\text{Kim}})^y_{\text{Kim}} \quad (\mathbf{a}^x_{\text{Kim}})^y_{\text{Jackie}} \quad (\mathbf{a}^x_{\text{Kim}})^y_{\text{Tamia}}$$

$$\mathbf{a}^x_{\text{Tamia}}$$

$$(\mathbf{a}^x_{\text{Tamia}})^y_{\text{Tamia}} \quad (\mathbf{a}^x_{\text{Tamia}})^y_{\text{Jackie}} \quad (\mathbf{a}^x_{\text{Tamia}})^y_{\text{Kim}}$$

Again, as mentioned above, relative to any of the 9 leaves in the above trees, $L(x, y)$ should come out satisfied in $\mathcal{L}_1$. So there are 6 assignments, 2 for each tree, that entail individually that

$$\mathbf{S} \not\models \forall x \forall y L(x, y).$$

6.7 Putting it all together

Using the above techniques, one can calculate the truth-value of any sentence in $\mathcal{L}_1$, involving any number of quantifiers in any possible distribution. And in fact, one can also calculate the satisfaction of every formula of $\mathcal{L}_1$ relative to an initial assignment $\mathbf{a}$ in the structure $\mathbf{S}$, involving open formulas with quantifiers. At each step, one just has to apply the relevant definition for the quantifier or connective until none are left, and then calculate the satisfiability of each resulting atomic formula relative to the relevant assignments. As we shall see soon, there are various techniques that make this easier than it first appears. But first, let's formulate the semantics for our language $\mathcal{L}_1$ as a whole.

Again, the semantics is *compositional*, where the meaning of more complex expressions will always be calculable from the meaning of their constituent parts. Thus, parallel to how we introduced the semantics above, our definition will also start with assigning meaning to the predicates and terms of the language, then to atomic formulas, and then to more complex formulas.

Returning to satisfaction and truth once more, truth will be defined as a special case of satisfaction. Namely, satisfaction is relativized to structures and variable assignments, while truth will be assigned to those formulas in a structure that are satisfied under *every* possible variable assignment in the domain of that structure. In other words, only some formulas satisfiable in a structure will be said to be true, those that are satisfied under every variable assignment in the domain of the structure.

Definition 6.7.1 (Semantics of $\mathcal{L}_1$)**.** A first-order structure $\mathbf{S}$ is a pair $\langle \mathbf{D}, I\rangle$ where $\mathbf{D}$ is a non-empty set called the *domain*, and I is the *interpretation function* such that for every $c \in \mathsf{CONS}_{\mathcal{L}_1}$, $I(c) \in \mathbf{D}$, and for every predicate of arity n, $I(P^n) \subseteq \mathbf{D}^n$.

The function $\mathbf{a} : \mathsf{VAR}_{\mathcal{L}_1} \to \mathbf{D}$ is a *variable assignment* in $\mathbf{D}$. If $\mathbf{a}'$ is a variable assignment in $\mathbf{D}$ just like $\mathbf{a}$, except possibly for some x, $\mathbf{a}'(x) \neq \mathbf{a}(x)$, we call $\mathbf{a}'$ an x-variant variable assignment of $\mathbf{a}$. If $\mathbf{a}'(x) = d$ ($d \in \mathbf{D}$), i.e., $\mathbf{a}'$ is the x-variant variable assignment of $\mathbf{a}$ such that x is sent to d by $\mathbf{a}'$, we may also write $\mathbf{a}^x_d$.

For each term $t \in \mathsf{TERM}_{\mathcal{L}_1}$, we define the value of t in the structure $\mathbf{S}$ relative to the variable assignment $\mathbf{a}$ as follows:

1. if $t = x$ for some $x \in \mathsf{VAR}_{\mathcal{L}_1}$, $I(t)[\mathbf{a}] = \mathbf{a}(x)$;

2. if $t = c$ for some $c \in \mathsf{CONS}_{\mathcal{L}_1}$, $I(t)[\mathbf{a}] = I(c)$.

We define satisfaction for a formula X relative to $\mathbf{S}$ under $\mathbf{a}$, in symbols, $\mathbf{S} \models X[\mathbf{a}]$ as follows:

1. For the base: if $X = P^n(t_1, ..., t_n)$, then $\mathbf{S} \models X[\mathbf{a}]$ *iff* $\langle I(t_1)[\mathbf{a}], ..., I(t_n)[\mathbf{a}]\rangle \in I(P^n)$.

2. For the connectives:

   (a) if $X = \neg Y$, $\mathbf{S} \models X[\mathbf{a}]$ *iff* $\mathbf{S} \not\models Y[\mathbf{a}]$;

   (b) if $X = (Y \wedge Z)$, $\mathbf{S} \models X[\mathbf{a}]$ *iff* $\mathbf{S} \models Y[\mathbf{a}]$ and $\mathbf{S} \models Z[\mathbf{a}]$;

   (c) if $X = (Y \vee Z)$, $\mathbf{S} \models X[\mathbf{a}]$ *iff* $\mathbf{S} \models Y[\mathbf{a}]$ or $\mathbf{S} \models Z[\mathbf{a}]$ (or both);

   (d) if $X = (Y \to Z)$, $\mathbf{S} \models X[\mathbf{a}]$ *iff* if $\mathbf{S} \models Y[\mathbf{a}]$, then $\mathbf{S} \models Z[\mathbf{a}]$.

3. For the quantifiers:

   (a) if $X = \exists x Y$, then $\mathbf{S} \models X[\mathbf{a}]$ *iff* there is an x-variant assignment $\mathbf{a}'$ such that $\mathbf{S} \models Y[\mathbf{a}']$;

   (b) if $X = \forall x Y$, then $\mathbf{S} \models X[\mathbf{a}]$ *iff* for every x-variant assignment $\mathbf{a}'$, $\mathbf{S} \models Y[\mathbf{a}']$.

We say that a formula X is *true in* (or *relative to*) $\mathbf{S}$ *iff* for every assignment $\mathbf{a}$ (in $\mathbf{D}$ of $\mathbf{S}$), $\mathbf{S} \models X[\mathbf{a}]$. We say that a formula X is *false in* (or *relative to*) $\mathbf{S}$ *iff* for every assignment $\mathbf{a}$ (in $\mathbf{D}$ of $\mathbf{S}$), $\mathbf{S} \not\models X[\mathbf{a}]$.

You may see something strange with this definition of falsity. In particular, we defined a formula X being false *not* as it not being true, but as a separate condition (namely, being unsatisfied under every assignment in $\mathbf{S}$). The reason why this is the case is precisely because some formulas cannot be said to be true or false, as they depend on the assignment function for their evaluation. Accordingly, if we specified that the formulas that are not true are false, we would have had to deem false those formulas that are sometimes satisfied, sometimes aren't in $\mathbf{S}$. It should be noted however that as far as sentences are concerned, a sentence not being true *does* entail it being false, so we are not violating the Law of Excluded Middle.

6.8 The full force of the semantics

Now that we have everything in place, we can start appreciating how expressive our language $\mathcal{L}_1$ is. So far, in our examples, we only considered ourselves with complex formulas without quantifiers, and with multiple quantifiers in front of an atomic formula, with some negations thrown in here and there. But of course, the real magic happens when we use the quantifiers and the connectives together to form complex sentences.

To make this more vivid, we may consider a possible real life application; a webshop's database of items. The website sells clothing items in the following categories: shoes, bottoms, tops, and accessories. Moreover, each item has one or more associated color (black, white, red, green, blue), and one or more associated material (cotton, polyester, polycotton, leather). All in all, the database has information about 10 pieces of clothing items. These, we can refer to by their associated 'code': 1, 2, 3, and so on.

All of this we can formally represent in a first-order structure. To make the particulars more easily readable, we will represent them in a table as follows:

Item	**Category**	**Color(s)**	**Fabric**
1	top	red, green	cotton
2	top	white	cotton
3	top	red, blue	cotton
4	top	blue	cotton
5	bottom	green	cotton
6	bottom	red, white	polycotton
7	shoes	white	polyester, cotton
8	shoes	blue	polyester, cotton
9	shoes	white	polyester, cotton
10	accessory	blue, white	polyester, cotton

Then, we can use the following predicates, with their interpretation specified relative to the table above:

1. Category predicates:
   (a) $I(Top) = \{x \mid x \text{ is a top}\}$
   (b) $I(Bot) = \{x \mid x \text{ is a bottom}\}$
   (c) $I(Sho) = \{x \mid x \text{ is a pair of shoes}\}$
   (d) $I(Acc) = \{x \mid x \text{ is an accessory}\}$
2. Color predicates:
   (a) $I(Red) = \{x \mid x \text{ is red}\}$
   (b) $I(Gre) = \{x \mid x \text{ is green}\}$
   (c) $I(Blu) = \{x \mid x \text{ is blue}\}$
   (d) $I(Whi) = \{x \mid x \text{ is white}\}$
   (e) $I(Bla) = \{x \mid x \text{ is black}\}$
3. Fabric predicates:
   (a) $I(Cot) = \{x \mid x \text{ is made of cotton}\}$
   (b) $I(Pol) = \{x \mid x \text{ is made of polyester}\}$
   (c) $I(Pc) = \{x \mid x \text{ is made of polycotton}\}$
   (d) $I(Lea) = \{x \mid x \text{ is made of leather}\}$

☞ Here is an interesting fabric fact. Some items are made out of polycotton, while some items are made out of polyester *and* cotton. We can distinguish between these two as follows. Polycotton is a fabric blend that is made out of both polyester *and* cotton. For example, many t-shirts are made entirely out of polycotton. But there are also some items that are made of pure cotton and pure polyester in some way combined. For example, you might buy some cotton shoes that have a polyester brand label. This does not make the shoes polycotton.

Let's call the structure determined by the above information **W** (for a change). Once **W** is fixed, we can try and express a number of complex facts about **W**, and in particular, about the various relations between these items and their properties.

6.8.1 Quantifier placement

Let's start with some simple facts. First, you might want to express something like the following: "There is a red top and a white top". You might hesitate between the following two options:

$$\exists x((Red(x) \wedge Top(x)) \wedge (Whi(x) \wedge Top(x)))$$
$$\exists x(Red(x) \wedge Top(x)) \wedge \exists x(Whi(x) \wedge Top(x))$$

In such cases, the correct option is the second one. For note that what the first option says is that there is a red and white top (that is a top). You can easily see this if you do the calculations explicitly, for in the first case, we need to find a single x-variant assignment that would satisfy the formula $((Red(x) \wedge Top(x)) \wedge (Whi(x) \wedge Top(x)))$. This is clearly impossible, for there is no one item that is a top, and both red and white. In the second case, however, we consider the two conjuncts separately, and thus we *can* consider two distinct x-variant assignments (two distinct objects). Accordingly, in **W**, the first sentence is false, while the second is true. We can see how the second sentence is true in **W** as follows:

$$\begin{array}{c}
\mathbf{W} \models \exists x(Red(x) \wedge Top(x)) \wedge \exists x(Whi(x) \wedge Top(x))[\mathbf{a}] \\
\begin{array}{cc}
\mathbf{W} \models \exists x(Red(x) \wedge Top(x))[\mathbf{a}] & \mathbf{W} \models \exists x(Whi(x) \wedge Top(x))[\mathbf{a}] \\
| & | \\
\mathbf{W} \models Red(x) \wedge Top(x)[\mathbf{a}^x_1] & \mathbf{W} \models Whi(x) \wedge Top(x)[\mathbf{a}^x_2] \\
\mathbf{W} \models Red(x)[\mathbf{a}^x_1] \quad \mathbf{W} \models Top(x)[\mathbf{a}^x_1] & \mathbf{W} \models Whi(x)[\mathbf{a}^x_2] \quad \mathbf{W} \models Top(x)[\mathbf{a}^x_2]
\end{array}
\end{array}$$

A similar problem arises if, for example, we change the conjunction to a disjunction, and the existential quantifier to a universal one. Suppose we want to express the fact that every item in the database is either made (at least partly) of polycotton, or made (at least partly) of cotton. Again, you have two options:

$$\forall x(Pc(x) \vee Cot(x))$$
$$\forall x Pc(x) \vee \forall x Cot(x)$$

Now the first sentence is the correct choice, and the second is the incorrect one. For note that what the second sentence says is really that either everything is made of polycotton, or everything is made of cotton. This sentence is a disjunction, so it is true if one of the disjuncts is true. But taken separately, it is *not* true that everything is made of polycotton, and it is also not true that everything is made of cotton. In particular, we can refer to the fact that 1 is not made of polycotton, while 6 is not made of cotton. Thus:

$$\begin{array}{ccc} & \mathbf{W} \not\models \forall x Pc(x) \vee \forall x Cot(x)[\mathbf{a}] & \\ \swarrow & & \searrow \\ \mathbf{W} \not\models \forall x Pc(x)[\mathbf{a}] & & \mathbf{W} \not\models \forall x Cot(x)[\mathbf{a}] \\ | & & | \\ \mathbf{W} \not\models Pc(x)[\mathbf{a}^x_1] & & \mathbf{W} \not\models Cot(x)[\mathbf{a}^x_6] \end{array}$$

But again, we do have that $\mathbf{S} \models \forall x(Pc(x) \vee Cot(x))$, since for every x-variant assignment $\mathbf{a}'$, we have that $\mathbf{S} \models Pc(x) \vee Cot(x)[\mathbf{a}']$.

6.8.2 Restricted quantification

Let's try and formulate some more interesting facts about our database. For example, we might want to capture some interrelations between various properties of our objects. One such fact would be the following: "If something is blue, then it is made of cotton".

The first question here would be: which quantifier are we supposed to use? From the above formulation, this is not clear, and indeed, the use of 'something' might be misleading. Suppose you went with:

$$\exists x(Blu(x) \to Cot(x))$$

This sentence is true in $\mathbf{W}$ if there is a value for x relative to which $Blu(x) \to Cot(x)$ is satisfied. There are several problems with this option. First, since this is an existentially quantified sentence, it is sufficient for a single object to satisfy $Blu(x) \to Cot(x)$ for the sentence to be true. This does not seem like what we are trying to express, which is a general rule applying to *every* thing that is blue.

But there is another misleading element here: the use of a conditional after an existential quantifier. As mentioned several times already, a conditional is *only* unsatisfied (false) if its antecedent is satisfied (true), but its consequent is unsatisfied (false). In this case, this means that for any value for x, $Blu(x) \to Cot(x)$ is satisfied relative to that value provided either $Blu(x)$ and $Cot(x)$ are both satisfied, or $Blu(x)$ is *not* satisfied (regardless of whether $Cot(x)$ is satisfied or not). This means that item 6 being red, white, and polycotton is by itself sufficient to make $\exists x(Blu(x) \to Cot(x))$ true in $\mathbf{W}$, for it is *not* made of cotton. This is usually not something we want to express. In general, if we wanted to express the fact that there is something that is blue and is made of cotton, then we can just say: $\exists x(Blu(x) \wedge Cot(x))$.

Returning to our initial fact under consideration, it was *not* that there is something that is blue and cotton, it was that whenever something is blue, then it is made of cotton. The correct way to capture this fact is with a universal quantifier. In particular:

$$\forall x(Blu(x) \to Cot(x))$$

Interestingly, once we change the quantifier to a universal one, the use of a conditional proceeding it will capture precisely what we wanted to express. In particular, the only fact that would make this sentence *false* is if there were a blue item that was *not* cotton. In other words, as long as there is no item that is both blue and not cotton, the sentence is true, which seems to be what we wanted to express.

Let's choose a blue and a non-blue item from the domain, and see how this functions. First, take item number 4, which is blue. Then, we have that:

$$\begin{aligned}
\mathbf{W} &\models Blu(x)[\mathbf{a}_4^x] \\
\mathbf{W} &\models Cot(x)[\mathbf{a}_4^x] \\
\mathbf{W} &\models Cot \to Blu(x)[\mathbf{a}_4^x]
\end{aligned}$$

But now taking item number 5, which is not blue, we can reason, based on the definition of satisfaction as it relates to $\to$, that:

$$\begin{aligned}
\mathbf{W} &\not\models Blu(x)[\mathbf{a}_5^x] \\
\mathbf{W} &\models Cot(x)[\mathbf{a}_5^x] \\
\mathbf{W} &\models Cot \to Blu(x)[\mathbf{a}_5^x]
\end{aligned}$$

Naturally, we would need to check whether $Blu(x) \to Cot(x)$ is satisfied in $\mathbf{W}$ relative to every possible x-variant assignment, not just these two. But it is easy to see that this will indeed hold, for again, every blue thing is cotton, and the non-blue things cannot make the sentence false.

If we zoom out for a bit, we can say that in sentences like $\forall x(Blu(x) \to Cot(x))$, the universal quantifier is *restricted.* What it is restricted to is the set of things satisfying the antecedent of the conditional. For in general, the universal quantifier 'ranges over' the whole domain. For example, if we had $\forall x Cot(x)$, the truth of this sentence depends on *every* possible assignment of value to x, so every object in the domain is relevant here. On the other hand, as mentioned, once we put $Blu(x)$ in front with a conditional, we are no longer concerned with the whole domain. Rather, we *restrict* our attention to just the blue things in the domain, and we say *of them* that each must be cotton. Once quantification is restricted, any member of the domain not captured by the antecedent condition becomes irrelevant.

As a rule, the universal quantification of Z, $\forall xZ$ is thus restricted (w.r.t. Y) with a conditional in the form $\forall x(Y \rightarrow Z)$. We can also talk about restricting the existential quantification of Z, $\exists xZ$ (w.r.t. Y), which is done with a conjunction (as mentioned briefly above), in the form $\exists x(Y \wedge Z)$.

One problematic aspect of this is *vacuous quantification*. Note that it may be the case that *nothing* satisfies the antecedent of the conditional after the universal quantifier. As demonstrated above, if something does not satisfy the antecedent of the conditional, the conditional as a whole is automatically satisfied. Unfortunately, this entails that if *nothing* satisfies the antecedent of the conditional, then relative to each value of x, the conditional as a whole is satisfied. So its universal quantification is true. For example, $\mathbf{W} \models \forall x((Gre(x) \wedge Sho(x)) \rightarrow (Blu(x) \wedge \neg Blu(x)))$, for the sole (pun unintended) reason that there are no green shoes.

6.8.3 Quantifying into multiple positions

Finally, let's add even more complexity by introducing a two-place predicate Col, defined such that:

$$I(Col) = \{\langle x, y\rangle, \mid x \text{ is part of the same collection as } y\}.$$

Let's further specify that in our structure $\mathbf{W}$, there are four collections:

1. Collection 1: items 2, 6, and 9
2. Collection 2: items 4, 7 and 10
3. Collection 3: items 1, 3, and 8
4. Collection 4: item 5

We can represent these collections in terms of the relation $I(Col)$ as in Figure 6.2. As you can see, the relation allows us to separate the domain into 4 distinct, non-overlapping subsets, one for each collection, based on how the relation $I(Col)$ is distributed between the members (as represented by the arrows). The basic idea is that for any two members of the *same* collection, $I(Col)$ holds, while for any two items in *distinct* collections, they are never related in terms of $I(Col)$.

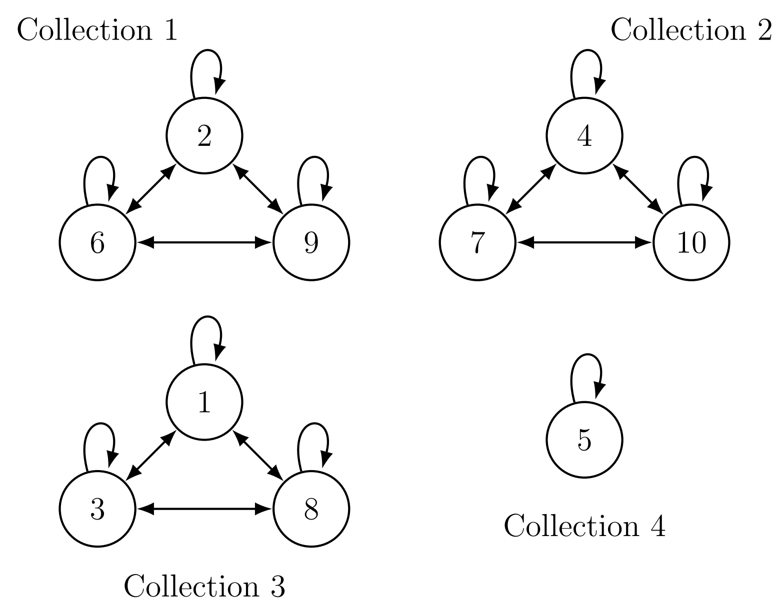


Figure 6.2: Partitioning the domain relative to $I(Col)$

Note that as it is specified, $I(Col)$ is a reflexive, symmetric, and transitive relation. Reflexive, because everything is in the same collection as itself. Symmetric, because if x is in the same collection as y, then y is in the same collection as x. And transitive, because if x is in the same collection as y, and y is in the same collection as z, then x is in the same collection as z. As it turns out, such relations always *partition* the domain into non-overlapping subsets, like our collections above.

We can then use multiple quantifiers to 'quantify into' the two 'positions' (argument places) of the binary predicate Col. For example, we might want to express the fact that there is an item for which it is true that if another item is in the same collection as it is, then that second item is white. Put another way, there is an item which is in a collection of white things. This can be captured as follows:

$$\exists x \forall y (Col(x, y) \to Whi(y))$$

How is this sentence to be evaluated relative to **W**? We need to check if we can find a value for x, such that for any value we can find for y, it is satisfied that if x and y are in the same collection, then y is white.

Here is what's *not* going to work. Suppose you choose for x the value 4. After all, *other than* 4, every member of collection 2 is white. The problem here is that we have to check *every* item in the same collection as 4, which includes 4 itself. And 4 is not white! So we have that:

$$\begin{aligned}
\mathbf{W} &\not\models \forall y(Col(x,y) \to Whi(y))[\mathbf{a}_4^x] \\
\mathbf{W} &\not\models Col(x,y) \to Whi(y)[(\mathbf{a}_4^x)_4^y] \\
\mathbf{W} &\models Col(x,y)[(\mathbf{a}_4^x)_4^y] \\
\mathbf{W} &\not\models Whi(y)[(\mathbf{a}_4^x)_4^y]
\end{aligned}$$

(Note that we are not saying that $\mathbf{S} \not\models \exists x \forall y(Col(x,y) \to Whi(y))$, we are saying that choosing 4 as the value for x would not satisfy $\forall y(Col(x,y) \to Whi(y))$. This does not entail that the initial formula is false, and indeed, it is true, we just have to find the right value for x!)

On the other hand, we may choose any one item of collection 1, for there, it is guaranteed that every member of the collection will be white (including the chosen member). For example:

$$\begin{aligned}
\mathbf{W} &\models \exists x \forall y(Col(x,y) \to Whi(y))[\mathbf{a}] \\
\mathbf{W} &\models \forall y(Col(x,y) \to Whi(y))[\mathbf{a}_2^x] \\
\mathbf{W} &\models Col(x,y) \to Whi(y)[(\mathbf{a}_2^x)_2^y] \\
\mathbf{W} &\models Col(x,y) \to Whi(y)[(\mathbf{a}_2^x)_6^y] \\
\mathbf{W} &\models Col(x,y) \to Whi(y)[(\mathbf{a}_2^x)_9^y]
\end{aligned}$$

As before, these are only the relevant y-variant assignments that *could* make a difference. In particular, if we were to send y to any member other than 2, 6 or 9, the conditional would immediately be satisfied by the fact that for no other assignment to y do we have $Col(x,y)$ (when x is sent to 2).

Exercise 6.8.1. From the above, it does not follow that $\mathbf{W} \models \forall x(\exists y Col(x,y) \to Whi(y))$. Nor does it follow that $\mathbf{W} \models \forall x \forall y(Col(x,y) \to Whi(y))$ Explain why these fail to be true in $\mathbf{W}$.

6.8.4 Translating the untranslatable

As our final example, let's try and capture something trickier. Suppose we want to capture the fact that there is a collection with a red and a blue item in it. At first glance, it seems like you could easy express this fact by using an existential quantifier and a predicate for 'is a collection'. The problem is that we have no such predicate! We have the predicate Col, but that predicate says not that something is a collection, but that two items are in the *same* collection. There is a reason for not including a predicate for 'is a collection'. For note that one-place predicates are evaluated in our semantics as subsets of our domain, and our domain does not have collections as members in it. It only has clothing items! And if you check $Col(x, y)$, it applies to clothing items, and *not* the collections, for it says that two *items* x, y are in the same collection.

On the other hand, we *can* capture the above fact without talking explicitly about collections, just by using the predicate Col. For there being a collection with a red and a blue item in it is equivalent to saying that there are two items, one red and blue, such that they are in the same collection. If this is true, then there is a collection with a red and a blue item in it. Conversely, if there is a collection with a red and a blue item in it, then there are two items, one red, one blue that are in the same collection. (For everything in this paragraph, you should remember that there may only be a single object that has both properties.)

> First-order languages, though very expressive, are still in some respects limited. In particular, they are limited by the fact that the quantifiers only range over members of the domain, predicates only have as interpretations subsets of the domain, and so on. On the other hand, many times (though certainly not always!), there are clever ways to turn an 'untranslatable' claim into a translatable one. The touchstone for these is 'truth-conditional equivalence'. All this means is that the two sentences should be true (false) under the the same circumstances.

Accordingly, we can represent the fact that there is a collection with a red and a blue item in it as:

$$\exists x \exists y((Red(x) \wedge Blu(y)) \wedge Col(x, y))$$

In particular, item 1 is red, item 3 is blue, and they are both in collection 3. So we have:

$$\mathbf{W} \models \exists x \exists y((Red(x) \wedge Blu(y)) \wedge Col(x, y))[\mathbf{a}]$$

$$\mathbf{W} \models \exists y((Red(x) \wedge Blu(y)) \wedge Col(x, y))[\mathbf{a}^x_1]$$

$$\mathbf{W} \models (Red(x) \wedge Blu(y)) \wedge Col(x, y)[(\mathbf{a}^x_1)^y_3]$$

$$\mathbf{W} \models (Red(x) \wedge Blu(y))[(\mathbf{a}^x_1)^y_3] \qquad \mathbf{W} \models Col(x, y)[(\mathbf{a}^x_1)^y_3]$$

$$\mathbf{W} \models Red(x)[(\mathbf{a}^x_1)^y_3] \qquad \mathbf{W} \models Blu(y)[(\mathbf{a}^x_1)^y_3]$$

Now you try it!

Exercise 6.8.2. Try and formulate in $\mathcal{L}_1$ the fact that the relation $I(Col)$ is reflexive, symmetric, and transitive. In particular:

1. Every item is in the same collection as itself.
2. For every two items, if the first is in the same collection as the second, then the second is in the same collection as the first one.
3. For every three items, if the first is in the same collection as the second, and the second is in the same collection as the third, then the first is in the same collection as the third.

Exercise 6.8.3. Try to formulate in $\mathcal{L}_1$ the following facts about $\mathbf{W}$:

1. If something is made of polycotton, then it is a red and white bottom.
2. Every green item is either a top or a bottom.
3. Every item is either an accessory or not blue and white.
4. If something is both polyester and cotton, then it is not a top.
5. If something is a top, then it is only made of one material.

   [Hint: something being made of one material cannot be expressed directly, but you can express each possible situation in which something is made of one thing and not the other two. One of these situations must hold if something is made of a single material.]

In each case, explain why the fact as you translated it into $\mathcal{L}_1$ is true.

Remark 6.8.1. Here is how your answers should look like. Suppose you want to express that no item is made of leather.

The translation of the above fact is: $\neg\exists x Lea(x)$, since we want to express that there is no x such that x is made of leather. Now $\mathbf{W} \models \neg\exists Lea(x)$, since $\mathbf{W} \models \neg\exists x Lea(x)[\mathbf{a}]$ *iff* $\mathbf{W} \not\models \exists x Lea(x)[\mathbf{a}]$. In turn, $\mathbf{W} \not\models \exists x Lea(x)[\mathbf{a}]$ if there is *no* x-variant assignment $\mathbf{a}'$ such that $\mathbf{W} \models Lea(x)[\mathbf{a}']$. This means that for *every* x-variant assignment $\mathbf{a}'$, $\mathbf{W} \not\models Lea(x)[\mathbf{a}']$. But this is the case, since nothing is made of leather, that is, $I(Lea) = \emptyset$.

Exercise 6.8.4. Write four true sentences about $\mathbf{W}$ that each contain *at least* 2 quantifiers and a connective. In each case, express what fact you are capturing about $\mathbf{W}$ by the formula.

Chapter 7

Doing first-order logic

Now that we are familiar with some of the ins and outs of first-order syntax and semantics, we can start covering the central notions of first-order logical systems. As you will see, the account will be eerily similar to the zeroth-order one above. In fact, first-order logic is an *extension* of zeroth-order logic, in the sense that every valid argument of zeroth-order logic is also valid in first-order logic (but *not* vice versa!). As with zeroth-order logic, logical notions can be formulated in two ways; semantically and syntactically. Thankfully, since first-order logic is *sound* and *complete*, just as zeroth-order logic is, we can be sure that validity and satisfiability will coincide, whether formulated semantically or syntactically.[1]

7.1 First-order logic, semantically speaking

To save space and time, we will immediately formulate our notions for *sets* of formulas. As noted above, this is not a problem since single formulas are just special sets of formulas. Namely, for any formula X, there is a singleton set $\{X\}$.

7.1.1 Satisfiability

As before, we will start with the notion of satisfiability for sets of formulas (including singleton sets), and define validity using the notion of satisfiability. Thus, we have:

Definition 7.1.1 (Satisfiability). A set S of formulas of $\mathcal{L}_1$ is first-order *satisfiable* or *semantically consistent iff* there is a structure $\mathbf{S}$ and an assignment $\mathbf{a}$ such that $\mathbf{S} \models X[\mathbf{a}]$ for

[1]It is important to note here that this is not a universal feature of logical systems. In fact, second-order logic is *not* complete, meaning that there are some arguments that are semantically valid, but not syntactically so. In other words, they cannot be deduced, no matter what system one uses, or how hard one tries.

each $X \in S$. We say S is first-order *unsatisfiable* or *semantically inconsistent* provided it is not semantically consistent.

Exercise 7.1.1. By the above, it follows that a set of formulas is semantically inconsistent provided there is no structure $\mathbf{S}$ and assignment $\mathbf{a}$ under which every formula in the set is simultaneously satisfiable. Explain why this is the case in your own words using the definition above.

Checking whether a set of formulas is first-order satisfiable is checking whether there is a structure and an assignment that assigns a member of the domain of that structure to the variables of the language such that every formula in the set comes out satisfied. This sounds very complicated, but as usual, the underlying idea is very simple.

Take, for example, the following formula (considered as a singleton set):

$$P(x) \wedge \forall y P(y)$$

Is this formula satisfiable? To answer this in the positive, we need to find a structure and a variable assignment in which it *is* satisfied. In fact, there are many structures and assignments that satisfy this formula, so it *is* satisfiable. Here is one.

Let $\mathbf{S}$ be a structure such that $\mathbf{D} = \{\mathsf{a}\}$, and $\mathbf{I}(P) = \{\mathsf{a}\}$. For any variable x, let $\mathbf{a}(x) = \mathsf{a}$. So there is a single object a in the structure, and that single object is in the set of things that are P. Moreover, every variable is assigned a under the variable assignment $\mathbf{a}$.

Based on this, we have that $\mathbf{S} \models P(x)[\mathbf{a}]$, since $\mathbf{a}(x) \in \mathbf{I}(P)$. Moreover, we also have that $\mathbf{S} \models \forall y P(y)[\mathbf{a}]$, since under every y-variant assignment to $\mathbf{a}$, $P(y)$ is satisfied. In other words (and some symbols), $\mathbf{S} \models P(y)[\mathbf{a}']$ for every y-variant assignment $\mathbf{a}'$, simply because every assignment points to a (as there is nothing else in the domain). So since both are satisfied under $\mathbf{a}$ in $\mathbf{S}$, their conjunction $P(x) \wedge \forall y P(y)$ is also satisfied in the same structure under the same assignment. So the formula $P(x) \wedge \forall y P(y)$ is satisfiable.

There are many other formulas that are satisfiable because they are satisfied in the above structure $\mathbf{S}$ and assignment $\mathbf{a}$.

Exercise 7.1.2. Show that each of the following formulas are first-order satisfiable by showing that they are satisfied in $\mathbf{S}$ and $\mathbf{a}$, as defined above.

1. $\exists x_1 P(x) \wedge \neg\forall y \neg P(y)$

2. $\forall x(P(x) \vee \neg P(x))$

3. $\forall x P(x) \vee \neg P(x)$

4. $\neg P(y) \to \neg P(y)$

5. $\forall x(P(x) \to \exists y P(y))$

Remark 7.1.1. Note that this entails that the set of formulas $\{\exists x_1 P(x) \wedge \neg\forall y \neg P(y), \forall x(P(x) \vee \neg P(x)), \forall x P(x) \vee \neg P(x), \neg P(y) \to \neg P(y), \forall x(P(x) \to \exists y P(y))\}$ is satisfiable, since you just showed that they are each satisfied in the same structure under the same assignment.

Exercise 7.1.3. Write five formulas using only the predicate P and no constants such that they are each satisfied in $\mathbf{S}$ under $\mathbf{a}$, and thus *satisfiable* in first-order logic.

Of course, some sets of formulas require more elaborate structures and assignments to be satisfied, and hence be satisfiable. And in general, just because a set of formulas is not satisfied in a given structure under a given assignment does not entail that it is *not* satisfiable. For the latter, what you would have to show is that there is *no* structure and assignment whatsoever under which the set of formulas are satisfied.

Take, for example, the formula $P(x) \wedge \neg P(y)$. This is *not* satisfied in $\mathbf{S}$ under any variable assignment $\mathbf{a}^*$, simply because it requires an object to *not* be P. So $\mathbf{S} \not\models P(x) \wedge \neg P(y)[\mathbf{a}^*]$ for any $\mathbf{a}^*$. On the other hand, this does *not* mean that $P(x) \wedge \neg P(y)$ is *unsatisfiable*. Indeed, it is easy to think of a structure and an assignment that satisfies it.

Let $\mathbf{W}$ be the structure such that $\mathbf{D} = \{\mathsf{a}, \mathsf{b}\}$, $\mathbf{I}(P) = \{\mathsf{a}\}$. Moreover, let b be $\mathbf{b}(x) = \mathsf{a}$ and $\mathbf{b}(y) = \mathsf{b}$. Now $\mathbf{W} \models P(x) \wedge \neg P(y)[\mathbf{b}]$. Why? It is because $\mathbf{W} \models P(x)[\mathbf{b}]$, for $\mathbf{b}(x) \in \mathbf{I}(P)$, and $\mathbf{W} \models \neg P(y)[\mathbf{b}]$, since $\mathbf{b}(y) \notin \mathbf{I}(P)$. So $P(x) \wedge \neg P(y)$ *is* satisfiable, since there is a structure and an assignment that satisfy it, as just demonstrated.

Note that when it comes to open formulas, assignments matter! On the other hand, when it comes to sentences, we have an easier time. You may remember that a sentence is either satisfied under every assignment in a structure or none, and therefore, is true or false in that structure. So the only things you need to care about in these cases is whether the sentence is true in the structure or not. This is because if the sentence is true in a structure, it is satisfied under every assignment in that structure by definition, so it is satisfiable, as per our definition of satisfiability. Not coincidentally, this is exactly how zeroth-order satisfiability was defined, as zeroth-order languages do not have open formulas.

Here is a simple example. Suppose you have a first-order sentence $P(b) \vee \neg P(a)$. Extend the interpretation $\mathbf{I}$ of $\mathbf{S}$ so that $\mathbf{I}(b) = \mathbf{I}(a) = \mathsf{a}$ (for there is nothing else in the domain of $\mathbf{S}$). Now since $\mathsf{a} \in \mathbf{I}(P)$, $\mathbf{S} \models P(b)$, so $\mathbf{S} \models P(b) \vee \neg P(a)$, and the sentence is *true* in $\mathbf{S}$. So the sentence is satisfiable in

> Remember: for a set of formulas to be satisfiable, it is enough to find a *single* structure and variable assignment under which each formula is satisfied. Moreover, if a sentence (closed formula) is true in a structure, by definition, it is satisfied under every assignment in that structure, and thus it is satisfied in a structure under an assignment, and thus it is satisfiable full stop. So if a set of sentences are all true in a structure, they are satisfied in that structure, so they are first-order satisfiable.

So some sets of formulas are satisfiable, and for each set of satisfiable formulas, there needs to be only one structure and assignment under which each formula in the set is satisfied (in the same structure, under the same assignment). Now let's inspect some examples of sets of formulas that are *not* satisfiable. In other words, sets of formulas for which a structure and assignment cannot be found under which they are all satisfied. As noted, these formulas we will call (first-order) unsatisfiable or semantically inconsistent.

As a simple example, take the sentence $P(a) \wedge \neg P(a)$. You may already see that this is a sentence of form $X \wedge \neg X$, which is the most basic form of a contradiction. As we have demonstrated previously, there is no structure $\mathbf{S}$ in which it could be true, simply by the way $\wedge$ and $\neg$ is defined. But this also means that no matter what structure $\mathbf{S}^*$ and assignment $\mathbf{a}^*$ we choose, $\mathbf{S}^* \not\models P(a) \wedge \neg P(a)[\mathbf{a}^*]$, so indeed, this formula is first-order unsatisfiable.

Note that the same goes for first-order *sentences* that include quantifiers, like $\exists x P(x) \wedge \neg \exists x P(x)$. This sentence is unsatisfiable precisely for the same reason as $P(a) \wedge \neg P(a)$ is unsatisfiable. Simply because if $\exists x P(x) \wedge \neg \exists x P(x)$ were satisfied in a structure under an assignment, then $\exists x P(x)$ would have to be satisfied, and $\neg \exists x P(x)$ would also have to be satisfied. But by definition, if $\exists x P(x)$ is satisfied, $\neg \exists x P(x)$ is not, and if $\neg \exists x P(x)$ is satisfied, $\exists x P(x)$ is not. So there is no way to satisfy this formula. In general, any formula of form $X \wedge \neg X$, where X is any formula, is first-order unsatisfiable unsatisfiable for this reason.

> Note: just as any set of *sentences* is first-order satisfiable if it is *true* in a structure, any set of sentences is first-order unsatisfiable if it is *false* in *every* structure. So in both cases, variable assignments may be disregarded, as there are no free variables.

It is important to note that some first-order unsatisfiable sets of formulas are unsatisfiable not solely because of how the connectives are defined (which coincides with zeroth-order unsatisfiability), but essentially because of how the variables and quantifiers function. For example, take $P(x) \wedge \forall y \neg P(y)$. This formula is not of the form $X \wedge \neg X$, or any zeroth-order contradiction for that matter, yet it *is* first-order unsatisfiable. It is not hard to see why this is the case.

Suppose $P(x) \wedge \forall y \neg P(y)$ *were* first-order satisfiable. Then, there would be a structure $\mathbf{S}$ and assignment $\mathbf{a}$ such that $\mathbf{S} \models P(x) \wedge \forall y \neg P(y)[\mathbf{a}]$. In turn, this would mean that $\mathbf{S} \models P(x)[\mathbf{a}]$, that is, that $\mathbf{a}(x) \in \mathbf{I}(P)$. On the other hand, we would also have that $\mathbf{S} \models \forall y \neg P(y)[\mathbf{a}]$, which would mean that for every y-variant $\mathbf{a}'$, $\mathbf{S} \models \neg P(y)[\mathbf{a}']$, so $\mathbf{S} \not\models P(y)[\mathbf{a}']$ for every y-variant $\mathbf{a}'$, and so $\mathbf{a}'(y) \notin \mathbf{I}(P)$ for every y-variant $\mathbf{a}'$. But this is only true if $\mathbf{I}(P)$ is empty, otherwise we would be able to find a y-variant assignment $\mathbf{a}'$ such that $\mathbf{a}'(y) \in \mathbf{I}(P)$. On the other hand, we also showed that $\mathbf{a}(x) \in \mathbf{I}(P)$, so it should not be empty! In other words, these two conjuncts cannot be satisfied at the same time, since one requires $\mathbf{I}(P)$ to have at least one member, and the other requires it to be empty. So $P(x) \wedge \forall y \neg P(y)$ cannot be satisfied, and so is not first-order satisfiable.

Exercise 7.1.4. Show that the first-order sentence $\exists x P(x) \wedge \forall y \neg P(y)$ is unsatisfiable. Hint: it is unsatisfiable for essentially the same reasons as $P(x) \wedge \forall y \neg P(y)$, with a slight twist.

7.2 Validity

Now that we have a grasp on first-order satisfiability, we are in the position to formulate what it means for an argument to be first-order valid, and in connection, what it means for a formula to be a first-order validity. As we did with zeroth-order validity, we will define first-order validity using the notion of first-order satisfiability. In particular, we shall say:

Definition 7.2.1 (Validity, first-order)**.** Let $A = \{X_1, X_2, ...\}$ be any set of formulas and let Y be any formula of $\mathcal{L}_1$. Then, the argument from *premises* A to the *conclusion* Y is first-order *valid iff* the set $\{X_1, X_2, ..., \neg Y\}$ is not first-order satisfiable. If an argument from A to Y is first-order *valid*, we write $A \models Y$, or say that Y is a (first-order) *semantic consequence* of, or (first-order) *semantically entailed* by A. If the argument is not valid, we say it is *invalid*, and write $A \not\models Y$. If $A = \emptyset$ and $A \models Y$, we say Y is a first-order validity, and simply write $\models Y$.

Similar to our zeroth-order definition, the reasoning behind this is as follows. An argument is valid if whenever its premises are satisfied, its conclusion is also satisfied. This is equivalent to saying that it is *impossible* for an argument's premises to be satisfied but its conclusion to be unsatisfied. This latter means that if an argument is valid, we should not be able to find a structure and a variable assignment which satisfies all its premises but does not satisfy its conclusion. Put another way, we should not be able to find a structure and variable assignment under which the premises and the *negation* of the conclusion is satisfied. So put another way, we should not find that the set of the premises and the negation of the conclusion together is satisfiable. Which is exactly what the definition says.

As far as first-order sentences are concerned, the definition of validity essentially reduces to that of zeroth-order validity. In particular, a set of sentences $A = \{X_1, X_2, ...\}$ entails a sentence Y provided in every structure in which A is true, Y is also true, which is once again equivalent to $\{X_1, X_2, ..., \neg Y\}$ not being true in any structure. Keep in mind, however, that these sentences may include quantifiers, as long as every variable is bound. The only additional complication when including open formulas is that we need to include into our calculations the assignments. But again, this just means that if an argument is valid, then whenever its premises are satisfied in a structure under an assignment, its conclusion will also always be satisfied in that structure under that assignment.

Note that by definition, an argument is invalid if it not valid. And validity for an argument with premise set $\{X_1, X_2, ...\}$ and conclusion Y is defined by the set $\{X_1, X_2, ..., \neg Y\}$ being *unsatisfiable*. From this, it readily follows that an argument is *not* valid (or simply, invalid) if $\{X_1, X_2, ..., \neg Y\}$ *is* satisfiable. In other words, if you want to show that an argument is invalid, you need to provide a structure and an assignment under which the premises are satisfied, but the conclusion is not (or equivalently, its negation is).

Exercise 7.2.1. Show that the following facts hold in first-order logic:

1. $\exists x \neg P(x) \not\models \forall y \neg P(y)$
2. $\neg\forall x \neg P(x) \not\models \neg\exists x \neg P(x)$
3. $\neg\forall x P(x) \not\models \neg\exists x P(x)$
4. $\forall x P(x) \rightarrow \forall x Q(x) \not\models \forall x (P(x) \rightarrow Q(x))$
5. $\exists x (P(x) \wedge Q(x)) \not\models \exists x P(x) \wedge \exists x Q(x)$
6. $\forall x (P(x) \vee Q(x)) \not\models \forall x P(x) \vee \forall x Q(x)$
7. $\exists x (P(x) \rightarrow Q(a)) \not\models \exists x P(x) \rightarrow Q(a)$
8. $\{\exists x (P(x) \rightarrow Q(x)), \exists x P(x)\} \not\models \exists x Q(x)$

In general, every zeroth-order valid argument is first-order valid. In fact, zeroth-order satisfiability, unsatisfiability and validity entail first-order satisfiability, unsatisfiability and validity, respectively (for zeroth-order arguments). But once again, there are first-order valid arguments whose validity essentially depends on the particular use of variables and quantifiers. Most importantly, there are two fundamental properties of quantifiers, connected to negation:

Proposition 14 (Negation push-through; $\exists$)**.** $\neg\exists xY \models \forall x\neg Y$ *for any formula* Y *of* $\mathcal{L}_1$*.*

Proof. Suppose $\mathbf{S} \models \neg\exists xY[\mathbf{a}]$. By definition, $\mathbf{S} \not\models \exists xY[\mathbf{a}]$. Since $\mathbf{S} \models \exists xY[\mathbf{a}]$ means that for some x-variant $\mathbf{a}'$, $\mathbf{S} \models Y[\mathbf{a}']$, its negation means that there is no x-variant $\mathbf{a}'$ for which we have $\mathbf{S} \models Y[\mathbf{a}']$. That is, for every x-variant $\mathbf{a}'$, $\mathbf{S} \not\models Y[\mathbf{a}']$. That is, for every x-variant $\mathbf{a}'$, $\mathbf{S} \models \neg Y[\mathbf{a}']$. But this just means that $\mathbf{S} \models \forall x\neg Y[\mathbf{a}]$. □

It is easy to see intuitively why this holds. Simply, if there is no x such that Y, then obviously, every x must not be Y, for otherwise, there would be an x that is Y. For example, if among a group of students, *there is no* student who received a C on their exam, then for *every* student, it is true that they did *not* receive a C on their exam. That is, if $\neg\exists xC(x)$, then $\forall x\neg C(x)$.

Note also that Y may be any formula above, not just an atomic one. So it is also the case, for example, that if *there is no* person who has a car *and* and has a sister, then for *every* person, it is *not* the case that they have a car and a sister. So if $\neg\exists x(H(x) \wedge S(x))$, then $\forall x\neg(H(x) \wedge S(x))$. It is important here where the parentheses are. For no one having a car *and* a sister does not mean that no one has a car *or* a sister. It only means that they cannot have both. And indeed, this follows, for by DeMorgan's law, $\forall x\neg(H(x) \wedge S(x))$ entails $\forall x(\neg H(x) \vee \neg S(x))$. That is, it is true of everyone that they either do not have a car, *or* they do not have a sister, or they neither have a car nor a sister. The only possible configuration excluded by this sentence is that they have both have a car *and* a sister.

Next, we also have the converse of the above. Namely, that if it is *not* true for all x that Y, then there must be at least one x for which Y *is* the case.

Proposition 15 (Negation push-through; $\forall$)**.** $\neg\forall xY \models \exists x\neg Y$ *for any formula* Y *of* $\mathcal{L}_1$*.*

Proof. Suppose $\mathbf{S} \models \neg\forall xY[\mathbf{a}]$. By definition, $\mathbf{S} \models \neg\forall xY[\mathbf{a}]$ *iff* $\mathbf{S} \not\models \forall xY[\mathbf{a}]$. Now $\mathbf{S} \models \forall xY[\mathbf{a}]$ provided for every x-variant assignment $\mathbf{a}'$, we have $\mathbf{S} \models Y[\mathbf{a}']$. So on the contrary, $\mathbf{S} \not\models \forall xY[\mathbf{a}]$ provided there is some x-variant assignment $\mathbf{a}'$ for which we have $\mathbf{S} \not\models Y[\mathbf{a}']$. So there is some x-variant assignment $\mathbf{a}'$ such that $\mathbf{S} \models \neg Y[\mathbf{a}']$. But then by definition, $\mathbf{S} \models \exists x\neg Y[\mathbf{a}]$. □

Now again, the intuitive content of this is pretty straightforward. For example, if *not everyone* got an A on their exam, then *there is* someone who did *not* get an A on their exam. That is, if $\neg\forall xA(x)$, then $\exists x\neg A(x)$. And of course, once again, this holds for any formula whatsoever. So for example, if *not everyone* has a car and a sister, then *there is* someone who does not have a car and a sister. That is, if $\neg\forall x(H(x) \wedge S(x))$, then $\exists x\neg(H(x) \wedge S(x))$. This then entails, by DeMorgan's law, $\exists x\neg H(x) \vee \neg S(x))$. That is, if not everyone has both a car and a sister, then there must be someone who either does not have a car, or does not have a sister, or neither has a car nor a sister.

The above two propositions also work backwards. In particular:

Proposition 16 (Negation pull-through; $\exists$)**.** $\forall x \neg Y \models \neg \exists x Y$ *for any formula* Y *of* $\mathcal{L}_1$*.*

Proposition 17 (Negation pull-through; $\forall$)**.** $\exists x \neg Y \models \neg \forall x Y$ *for any formula* Y *of* $\mathcal{L}_1$*.*

Exercise 7.2.2. Prove the two propositions above using the proofs for their push-through variants as blueprint. Hint: think backwards.

Note that the use of DeMorgan's law above was not accidental. The connectives are defined in first-order logic exactly the same way as they are in zeroth-order logic. Accordingly, the valid argument forms of zeroth-order logic carry over to first-order logic. For example, here is a clever utilization of the above propositions and some zeroth-order logic.

Proposition 18. $\exists x Y \models \neg \forall x \neg Y$ *and* $\neg \forall x \neg Y \models \exists x Y$

Proof. For the first, suppose $\mathbf{S} \models \exists x Y[\mathbf{a}]$. Then by zeroth-order reasoning, $\mathbf{S} \models \neg\neg \exists x Y[\mathbf{a}]$. Then, by pushing the negation through $\exists$, we have $\mathbf{S} \models \neg \forall \neg x Y[\mathbf{a}]$. So $\exists x Y \models \neg \forall x \neg Y$.

For the second, suppose $\mathbf{S} \models \neg \forall x \neg Y[\mathbf{a}]$. Then, pulling the negation through $\forall$, we get $\mathbf{S} \models \neg\neg \exists x Y[\mathbf{a}]$. Then, by zeroth-order reasoning, we get $\mathbf{S} \models \exists x Y[\mathbf{a}]$. So $\neg \forall x \neg Y \models \exists x Y$. □

Proposition 19. $\forall x Y \models \neg \exists x \neg Y$ *and* $\neg \exists x \neg Y \models \forall x Y$

Exercise 7.2.3. Prove the above proposition. Hint: take the proof of its inverted version above as blueprint.

Here is another example of how one can reason about validity in first-order logic:

Proposition 20. $\{\forall x(P(x) \to Q(x)), \exists y P(y)\} \models \exists y Q(y)$

Proof. Suppose $\mathbf{S} \models \forall x(P(x) \to Q(x))[\mathbf{a}]$, and $\mathbf{S} \models \exists y Q(y)[\mathbf{a}]$. By the latter, we know that there is a y-variant assignment $\mathbf{a}'$ such that $\mathbf{S} \models Q(y)[\mathbf{a}']$. In other words, $\mathbf{a}'(y) \in \mathbf{I}(Q)$. Let's denote the object $\mathbf{a}'(y)$ by c, so that we know $\mathsf{c} \in \mathbf{I}(Q)$. Now because $\mathbf{S} \models \forall x(P(x) \to Q(x))[\mathbf{a}]$, we know that for *every* x-variant assignment $\mathbf{a}''$, $\mathbf{S} \models P(x) \to Q(x)[\mathbf{a}'']$. So we also have that $\mathbf{S} \models P(x) \to Q(x)[\mathbf{a}^x_{\mathsf{c}}]$ (the x-variant assignment where x is sent to c). Now we already know that $\mathbf{S} \models P(x)[\mathbf{a}^x_{\mathsf{c}}]$, since we established that $\mathsf{c} \in \mathbf{I}(P)$. But then $\mathbf{S} \models Q(x)[\mathbf{a}^x_{\mathsf{c}}]$ as well, by the conditional. So $\mathsf{c} \in \mathbf{I}(Q)$. This, in turn, means that $\mathbf{S} \models Q(y)[\mathbf{a}^y_{\mathsf{c}}]$. So by definition, $\mathbf{S} \models \exists y Q(y)[\mathbf{a}]$. □

Now as you can see, reasoning about validity is rather painful when done semantically. So it is time to extend our tableau system with new rules so that it can handle arguments in first-order logic.

7.3 First-order tableau

Just like our first-order syntax and semantics is an extension of zeroth-order syntax and semantics, our first-order tableau system will be an extension of our zeroth-order one. In particular, and again paralleling the syntactic and semantic treatment, the only additional rules we need to introduce for our system are the ones which will allow us to deal with quantified sentences.

Our tableau system only deals with first-order sentences, that is, closed formulas. Accordingly, open formulas will not occur at any part of any of the deductions to follow.

Accordingly, we still have the same rules for the connectives as introduced for zeroth-order tableau. In particular, we have:

1. the premise rule Pr;
2. the double negation rule $\neg\neg$;
3. the conjunctiom rule $\wedge$ and the negated conjunction rule $\neg\wedge$;
4. the disjunction rule $\vee$ and the negated disjunction rule $\neg\vee$;
5. the conditional rule $\rightarrow$ and the negated conditional rule $\neg \rightarrow$.

We also retain the general approach to how we construct our trees. So in particular, if we want to show that $\{X_1, X_2, ...\} \vdash Y$, we put the formula $\neg Y$ (the *negation* of Y) as the starting point. Then, if the argument is indeed syntactically valid, we should be able to close each branch of the tree (sooner or later) by finding some formula Z and its negation $\neg Z$ for each branch.

Again, the only change is that we need rules to deal with certain additional formula forms. In particular:

1. a rule $\forall$ for universally quantified sentences of form $\forall x Y$;
2. a rule $\neg\forall$ for negated universally quantified sentences of form $\neg\forall x Y$;
3. a rule $\exists$ for existentially quantified sentences of form $\exists x Y$;
4. a rule $\neg\exists$ for negated existentially quantified sentences of form $\neg\exists x Y$.

Accordingly, our definition of what it means for something to constitute a proof of validity is as follows:

Definition 7.3.1. Let $A = \{X_1, X_2, ...\}$ be any set of sentences and let Y be any sentence of $\mathcal{L}_1$. Then, the set of premises A *syntactically entails* Y, or Y is a *syntactic consequence* of A, *iff* there is a closed tree with $\neg Y$ at its origin using the zeroth-order rules, plus the rules $\forall$, $\neg\exists$, $\exists$, and $\neg\forall$. In such cases, we write $A \vdash Y$, and call the closed tree a (first-order) *proof* of Y from the premise set (or simply, premises) A. If there is a closed tableau with $\neg Y$ at its origin and without the use of the rule Pr, we say Y is a first-order theorem and write $\vdash Y$.

7.3.1 The rule $\forall$

We start with the simplest first-order tableau rule, $\forall$. What the rule $\forall$ captures, in semantic terms, is that if for some structure $\mathbf{S}$, $\mathbf{S} \models \forall x Y$, then if we take Y by itself and replace each variable x originally bound by the quantifier $\forall$ in front of the formula with an arbitrary name a, then the resulting sentence will also be true in the structure. Remember that we have a particular notation for this. Namely, what we claim is that if $\mathbf{S} \models \forall x Y$, then for *any* name a, we have that $\mathbf{S} \models Y(a/x)$. Unsurprisingly, this is a semantically valid inference in first-order logic. Moreover, it is intuitively quite simple.

For example, it is true of every natural number that it is either less than or equal 5, or larger than 5. That is, $\forall x(x \leq 5 \vee x > 5)$. But then we can substitute for x any specific number, and the resulting sentence will also be true. For example, it is true that $4 \leq 5 \vee 4 > 5$, since $4 \leq 5$. But it is also true that $7 \leq 5 \vee 7 > 5$, since $7 > 5$. And the sentence will also be true if we substitute 45, 2, or 83574634 for x in $\forall x(x \leq 5 \vee x > 5)$.

Accordingly, we get the following *non-branching* rule:

$$\begin{array}{lll} 1. & \forall x Y(x) & \\ 2. & Y(a/x) & \forall\ 1 \end{array}$$

(For *any* name a)

Figure 7.1: Universal quantification rule

The following example is an application of the $\forall$ rule:

$$\begin{array}{lll} 1. & \forall x \exists y(R(x,y) \wedge \exists z R(x,z)) & \\ 2. & \exists y(R(a,y) \wedge \exists z R(a,z)) & \forall\ 1 \end{array}$$

As you can see, we simply took the universally quantified sentence $\forall x \exists y(R(x,y) \wedge \exists z R(x,z))$, and using the $\forall$ rule, derived from it the sentence $\exists y(R(a,y) \wedge \exists z R(a,z))$, which results from taking off the quantifier expression $\forall x$ and substituting the name a for every occurrence of x bound by $\forall$. And since the rule works with *any* name whatsoever, we could also use it like this:

$$\begin{array}{lll} 1. & \forall x \exists y (R(x,y) \wedge \exists z R(x,z)) & \\ 2. & \exists y (R(c,y) \wedge \exists z R(c,z)) & \forall\ 1 \end{array}$$

Of course, in practice, which name we choose is not quite this arbitrary, since we are always aiming to close the branches of the tree. And this is usually only possible with some specific choice for a name to be substituted in.

For example, take $\{\forall x(P(x) \to Q(x)), P(b)\} \vdash Q(b)$. We can prove this as follows:

$$\begin{array}{lcl} 1. & \neg Q(b) & \text{Start} \\ 2. & P(b) & \text{Pr.} \\ 3. & \forall x(P(x) \to Q(x)) & \text{Pr.} \\ 4. & P(b) \to Q(b) & \forall\ 3 \\ 5. & \neg P(b) \quad\quad Q(b) & \to 4 \\ & \underset{2,5}{\otimes} \quad\quad\quad \underset{1,5}{\otimes} & \end{array}$$

As you can see, the only reason we were able to close the branches of the tree is because we substituted b for x when applying $\forall$ on line 3. If we substituted any other name for x, the branches would not have closed. Thus, when you are using the rule $\forall$, you need to be mindful which name you choose. Remember that the goal is always to be able to close each branch of the tree.

7.4 The rule $\neg\exists$

Just like the rules $\wedge$ and $\neg\vee$ function similarly, the $\forall$ rule functions similar to the $\neg\exists$ rule. This is no accident given the negation push-through result we showed earlier. In particular, if $\mathbf{S} \models \neg\exists x Y(x)$, then $\mathbf{S} \models \forall x \neg Y(x)$. And as noted above, from $\mathbf{S} \models \forall x \neg Y(x)$, it follows that $\mathbf{S} \models \neg Y(a/x)$ for any name a. So from $\neg\exists x Y(x)$, we should be able to immediately derive $\neg Y(a/x)$.

Again, intuitively, this is easy to see. For example, it is true at this point in time at least that there is no one taller than 10 feet. From this, it follows that everyone is *not* taller than 10 feet. So substituting any person's name, it will also be true that, e.g., 'Dwayne Johnson is not taller than 10 feet', 'Donald Trump is not taller than 10 feet', and 'Beyonce is not taller than 10 feet'. In symbols, from $\neg\exists x T(x)$, it follows that $\forall x \neg T(x)$, from which it follows that $\forall x \neg T(a)$ for any name a. This is precisely the type of reasoning the $\neg\exists$ rule codifies.

1.	$\neg\exists x Y(x)$	
2.	$\neg Y(a/x)$	$\neg\exists$ 1

(For *any* name a)

Figure 7.2: Negated existential quantification rule

For example, take $\neg\exists x(P(x) \to Q(x)) \vdash \neg Q(d) \wedge P(d)$. We can prove this as follows:

1.	$\neg(\neg Q(d) \wedge P(d))$		Start
2.	$\neg\exists x(P(x) \to Q(x))$		Pr.
3.	$\neg(P(d) \to Q(d))$		$\neg\exists$ 2
4.	$P(d)$		$\neg\to$ 3
5.	$\neg Q(d)$		$\neg\to$ 3
6.	$\neg\neg Q(d)$	$\neg P(d)$	$\neg\wedge$ 1
	$\otimes$ 5, 6	$\otimes$ 4, 6	

Once again, it was important when using the rule $\neg\exists$ that we substituted d for x, since it was d that occurred in the conclusion, and hence was necessary to close both branches of the tree.

7.5 The rule $\exists$

The rule $\exists$ comes with an important caveat that you should always keep in mind. Semantically speaking, a sentence like $\exists x P(x)$ gives us some information, but in some sense, that information is quite limited. Namely, it tells us that there exists an x such that it is P, but it does not tell us *which* thing is the one that is P. Accordingly, unlike with $\forall$, you *cannot* just substitute any name for x in $P(x)$ to get a sentence that is true. In fact, you cannot be sure that you can substitute *any* name for x in $P(x)$, for it might be the case that the object that makes $\exists x P(x)$ true does not have a name (that is, no name is interpreted such that its value is the object that is P).

For example, suppose you are in a classroom the first day of classes at your college, and there are 20 other students with you. However, you only know two of them by name, your friends Jade and Ali. Then, the professor tells you: ‘There is a senior in this classroom’. Can you infer that ‘Jade is a senior’, or that ‘Ali is the senior’, *based on this information alone*? The answer is ‘no’, since you do not know who the senior is, only that there is one. And the situation is not much better if the professor gives you the names of every person in the room, for again, you still do not know which name is the name of the senior.

However, sometimes you do want to have a name for an object in the domain that makes an existential statement true, and to then reason about that specific object. For example, the professor might tell you later on that every person who received an A was a senior, from which you can reason that the person who is a senior got an A. Now what you can do in this case is simple; introduce a *new* name with the explicit intention that it refer to that specific object. In the above example, we could introduce a new name like ‘Senior’, and say that that person is a senior. The important thing is that ‘Senior’ must be an entirely new name, not shared by anyone in the class. Then, you can infer that ‘Senior is a senior’, and from then, that ‘Senior got an A’, and so on.

Based on the above, our $\exists$ rule is specified as follows:

$$\begin{array}{lll}
1. & \exists x Y(x) & \\
2. & Y(a/x) & \exists\ 1
\end{array}$$

(For a name a that does not occur in the premise set or on the branch of the tree)

Figure 7.3: Existential quantification rule

We are now in the position to show Proposition 20 syntactically, namely, that $\{\forall x(P(x) \to Q(x)), \exists y P(y)\} \vdash \exists y Q(y)$.

$$\begin{array}{lccl}
1. & \neg\exists y Q(y) & & \text{Start}\\
2. & \exists y P(y) & & \text{Pr.}\\
3. & \forall x(P(x) \to Q(x)) & & \text{Pr.}\\
4. & P(e) & & \exists\ 2\\
5. & P(e) \to Q(e) & & \forall\ 3\\
6. & \neg P(e) & Q(e) & \to 5\\
7. & \underset{4,6}{\otimes} & \neg Q(e) & \neg\exists\ 1\\
 & & \underset{6,7}{\otimes} &
\end{array}$$

Notice the interplay between our three rules, $\exists$, $\forall$, and $\neg\exists$ above. First, we used the $\exists$ rule on $\exists y P(y)$. For this, we needed to find a new name that did not occur on any of the branches or in the premise set. This was trivial, since no name occurred in either. So we chose a random one, e. So we deduced that e is P. Next, we considered $\forall x(P(x) \to Q(x))$. With $\forall$, we can substitute for x any name whatsoever, but of course, we chose e specifically. So we got that if e is P, then e must be Q. With $\to$, this left us two options: either e is not P, or e must be Q. But we already knew that e *is* P, so that branch closed. Moreover, from $\neg\exists y Q(y)$, with the rule $\neg\exists$, we can substitute for y any name whatsoever. So in particular, we again substituted e for y, getting $\neg Q(e)$, and closed the other branch.

7.6 The rule $\neg\forall$

Given the above, it is now easy to understand the rule $\neg\forall$. First, we can again refer to negation push-through. In particular, semantically, if $\mathbf{S} \models \neg\forall x Y(x)$, then $\mathbf{S} \models \exists x \neg Y(x)$. That is, if it is not true of everything that Y, then there is something for which it is false that Y. But then again, we can use the reasoning we provided for the rule $\exists$, and derive from $\exists x \neg Y(x)$ the sentence $\neg Y(a/x)$ provided a is a new name introduced specifically for the object that makes $\exists x \neg Y(x)$ true. Thus, we simply have:

1.	$\neg\forall x Y(x)$	
2.	$\neg Y(a/x)$	$\neg\forall$ 1

(For a name a that does not occur in the premise set or on the branch of the tree)

Figure 7.4: Negated universal quantification rule

Here is an illustration of how this works in practice. Take the following:

$$\neg\forall x Q(x), \exists y S(y) \to \forall z Q(z) \vdash \neg\exists u S(u)$$

We can show that this is indeed the case as follows:

1.	$\neg\neg\exists u S(u)$		Start
2.	$\neg\forall x Q(x)$		Pr.
3.	$\exists y S(y) \to \forall z Q(z)$		Pr.
4.	$\neg Q(e)$		$\neg\forall$ 2
5.	$\neg\exists y S(y)$	$\forall z Q(z)$	$\to$ 3
6.	$\exists u S(u)$	$Q(e)$	$\neg\neg$ 1; $\forall$ 5
7.	$S(d)$	$\otimes$ 4, 6	$\exists$ 6
8.	$\neg S(d)$		$\neg\exists$ 5
	$\otimes$ 7, 8		

Notice something extremely important above. As noted, the name introduced with either $\exists$ or $\neg\forall$ must always be one that does not occur in the premise set *or* on the branch that we are working on. On line 4, we introduced the name e, using $\neg\forall$ on $\neg\forall x Q(x)$. Then, on the left branch, at line 6, we came across $\exists u S(u)$. This meant that we needed a new name to apply $\exists$. But now there were a name already occurring on the branch, namely, the name e. So e was off limits, and we needed to find a new one. So we chose d. Notice also that when it came to $\neg\exists$ and $\forall$, we chose the names d and e, respectively. There is no mystery here.

We could have chosen any name whatsoever, including the inverse selection. But since we wanted to close both branches, we chose the names in accordance with that goal.

Exercise 7.6.1. Show that the following hold in first-order logic:

1. $\neg\forall x P(x) \vdash \exists x \neg P(x)$
2. $\exists x \neg P(x) \vdash \neg\forall x P(x)$
3. $\neg\exists x P(x) \vdash \forall x \neg P(x)$
4. $\forall x \neg P(x) \vdash \neg\forall x P(x)$
5. $\neg\forall x \neg Q(x) \vdash \exists x Q(x)$
6. $\exists x Q(x) \vdash \neg\forall x \neg Q(x)$
7. $\neg\exists x \neg Q(x) \vdash \forall x Q(x)$
8. $\forall x Q(x) \vdash \neg\exists x \neg Q(x)$

Here is the first one:

$$\begin{array}{lcl}
1. & \neg\exists x \neg P(x) & \text{Start} \\
2. & \neg\forall x P(x) & \text{Pr.} \\
3. & \neg P(a) & \neg\forall\ 2 \\
4. & \neg\neg P(a) & \neg\exists\ 1 \\
 & \underset{3,4}{\otimes} &
\end{array}$$

Exercise 7.6.2. Show that the following hold in first-order logic:

1. $\forall x \exists y L(x, y) \vdash \exists z L(a, z)$
2. $\exists x \forall y L(y, x) \vdash \forall x \exists y L(x, y)$
3. $\forall x \neg L(a, x) \vdash \exists x \exists y \neg L(x, y)$
4. $\{\forall x(L(a, x) \to P(x)), \neg P(b)\} \vdash \neg L(a, b)$
5. $\{\forall x(L(a, x) \vee \exists y(L(a, y) \wedge L(y, x)))\} \vdash L(a, b) \vee \exists x(L(a, x) \wedge L(x, b))$
6. $\{\forall x L(x, b) \vee \forall x \neg L(x, b), L(a, b)\} \vdash L(c, b)$
7. $\{\forall x L(j, x) \wedge \neg\exists y L(b, y)\} \vdash L(j, b) \wedge \neg L(b, j)$

8. $\{\forall x(L(j,x) \to \forall y L(y,x)), \neg L(a,b)\} \vdash \neg L(j,b)$

9. $\{\forall x(\exists y L(y,x) \to \exists y L(x,y))\} \vdash \neg\exists x L(a,x) \to \neg L(c,a)$

10. $\{\neg L(a,a), \forall x \forall y(L(x,y) \to L(y,y))\} \vdash \neg\exists x L(x,a)$

7.7 Quantifiers and the conditional

An important thing we noted in the last chapter was the interplay between the quantifiers and the conditional. The reason quantifiers and the conditional can result in unintuitive consequences is the zeroth-order equivalence between $X \to Y$ and $\neg X \vee Y$, and how pushing and pulling $\neg$ through the quantifiers functions. The following exercise demonstrates some unexpected equivalences in first-order logic regarding the often misleading sentence $\exists x(P(x) \to Q(x))$, which is neither equivalent to $\exists x(P(x) \wedge Q(x))$, nor to $\forall x(P(x) \to Q(x))$.

Exercise 7.7.1. Show that the following facts hold in first-order logic:

1. $\exists x(P(x) \to Q(x)) \vdash \exists x(\neg P(x) \vee Q(x))$ and $\exists x(\neg P(x) \vee Q(x)) \vdash \exists x(P(x) \to Q(x))$

2. $\exists x(\neg P(x) \vee Q(x)) \vdash \exists x\neg(P(x) \wedge \neg Q(x))$ and $\exists x\neg(P(x) \wedge \neg Q(x)) \vdash \exists x(\neg P(x) \vee Q(x))$

3. $\exists x\neg(P(x) \wedge \neg Q(x)) \vdash \neg\forall x(P(x) \wedge \neg Q(x))$ and $\neg\forall x(P(x) \wedge \neg Q(x)) \vdash \exists x\neg(P(x) \wedge \neg Q(x))$

4. $\exists x(P(x) \to Q(x)) \vdash \neg\forall x(P(x) \wedge \neg Q(x))$ and $\neg\forall x(P(x) \wedge \neg Q(x)) \vdash \exists x(P(x) \to Q(x))$

A related problem arises when we think about quantifier placement regarding conditionals. In particular, it should be noted that putting a quantifier in front of a conditional and in the antecedent of a conditional will result in formulas that do not mean the same thing. For example, $\forall x \exists y(R(x,y) \to P(x))$ and $\forall x(\exists y R(x,y) \to P(x))$ mean different things. This is, again, because in some sense, there is a hidden negation in a conditional $X \to Y$, given its equivalence to $\neg X \vee Y$, and pulling the quantifier over that negation changes it to the other quantifier, by negation pull-through. Accordingly, we can prove the following:

Exercise 7.7.2. Show that the following facts hold in first-order logic:

1. $\forall x \exists y(R(x,y) \to P(x)) \vdash \forall x(\forall y R(x,y) \to P(x))$

2. $\forall x(\forall y R(x,y) \to P(x)) \vdash \forall x \exists y(R(x,y) \to P(x))$

3. $\exists x \forall y(R(x,y) \to P(x)) \vdash \exists x(\exists y R(x,y) \to P(x))$

4. $\exists x(\exists y R(x,y) \to P(x)) \vdash \exists x \forall y(R(x,y) \to P(x))$

Here is the first one:

$$
\begin{array}{lcl}
1. & \neg\forall x(\forall y R(x,y) \to P(x)) & \text{Start} \\
2. & \forall x \exists y(R(x,y) \to P(x)) & \text{Pr.} \\
3. & \neg(\forall y R(a,y) \to P(a)) & \neg\forall\ 1 \\
4. & \forall y R(a,y) & \neg \to 3 \\
5. & \neg P(a) & \neg \to 3 \\
6. & \exists y(R(a,y) \to P(a)) & \forall\ 2 \\
7. & R(a,b) \to P(a)) & \exists\ 6 \\
8. & R(a,b) & \forall\ 4 \\
9. & \neg R(a,b) \qquad P(a) & \to 7 \\
 & \underset{8,9}{\otimes} \qquad\quad \underset{5,9}{\otimes} &
\end{array}
$$

7.8 Properties of relations

Finally, let's return to properties of relations. We already saw how we can formulate these in first-order logic. In particular:

1. Reflexivity: $\forall x R(x,x)$
2. Symmetricity: $\forall x \forall y(R(x,y) \to R(y,x))$
3. Transitivity: $\forall x \forall y \forall z((R(x,y) \land R(y,z)) \to R(x,z))$

Using these formulations and first-order logic, we are now able to reason about these properties, and see what they entail.

One obvious thing we can show is that given these general properties, particular instances follow. For example, if reflexivity holds, then for any particular object a, we have $R(a,a)$. That is, $\forall x R(x,x) \vdash R(a,a)$ for any a.

$$
\begin{array}{lcl}
1. & \neg R(a,a) & \text{Start} \\
2. & \forall x R(x,x) & \text{Pr.} \\
3. & R(a,a) & \forall\ 2 \\
 & \underset{1,3}{\otimes} &
\end{array}
$$

We also have that $\{\forall x \forall y(R(x,y) \to R(y,x)), R(a,b)\} \vdash R(b,a)$.

$$
\begin{array}{llll}
1. & & \neg R(b,a) & \text{Start} \\
2. & & \forall x \forall y(R(x,y) \to R(y,x)) & \text{Pr.} \\
3. & & R(a,b) & \text{Pr.} \\
4. & & \forall y(R(a,y) \to R(y,a)) & \forall\ 2 \\
5. & & R(a,b) \to R(b,a) & \forall\ 4 \\
6. & \neg R(a,b) & R(b,a) & \to 5 \\
 & \otimes_{3,6} & \otimes_{1,6} &
\end{array}
$$

Finally, we also have

$$\{\forall x \forall y \forall z((R(x,y) \land R(y,z)) \to R(x,z)), R(a,b), R(b,c) \vdash R(a,c)\}$$

Exercise 7.8.1. Show that this fact holds in first-order logic.

In addition, we can also show some more general facts about relations. For example, if a relation R is symmetric and transitive, and we know that some object x relates by R to some object y, then from this, it follows that some object relates to itself. That is:

$$\{\forall x \forall y(R(x,y) \to R(y,x)), \forall x \forall y \forall z((R(x,y) \land R(y,z)) \to R(x,z)), \exists x \exists y R(x,y)\} \vdash \exists x R(x,x)$$

Thankfully, given our tableau system, this is not a particularly hard fact to prove, but it is somewhat tedious. The only trick is to realize that since universal quantifiers can be instantiated with any name independent of one another, the following is an instance of transitivity: $(R(a,b) \land R(b,a)) \to R(a,a)$.

1.	$\neg\exists xR(x,x)$				Start
2.	$\forall x\forall y(R(x,y) \to R(y,x))$				Pr.
3.	$\forall x\forall y\forall z((R(x,y) \land R(y,z)) \to R(x,z))$				Pr.
4.	$\exists x\exists yR(x,y)$				Pr.
5.	$\exists yR(a,y)$				$\exists$ 4
6.	$R(a,b)$				$\exists$ 5
7.	$\forall y(R(a,y) \to R(y,a))$				$\forall$ 2
8.	$R(a,b) \to R(b,a))$				$\forall$ 7
9.	$\neg R(a,b)$	$R(b,a)$			$\to$ 8
10.	$\otimes$ $6,9$	$\forall y\forall z((R(a,y) \land R(y,z)) \to R(a,z))$			$\forall$ 3
11.		$\forall z((R(a,b) \land R(b,z)) \to R(a,z))$			$\forall$ 10
12.		$(R(a,b) \land R(b,a)) \to R(a,a)$			$\forall$ 11
13.		$\neg(R(a,b) \land R(b,a))$		$R(a,a)$	$\to$ 12
14.		$\neg R(a,b)$	$\neg R(b,a)$		$\neg\land$ 13
15.		$\otimes$ $6,14$	$\otimes$ $9,14$	$\neg R(a,a)$	$\neg\exists$ 1
				$\otimes$ $13,15$	

Exercise 7.8.2. If we rearrange the above a bit, we get the following:

$$\{\forall x\forall y(R(x,y) \to R(y,x)), \forall x\forall y\forall z((R(x,y) \land R(y,z)) \to R(x,z)), \neg\exists xR(x,x)\}$$
$$\vdash \neg\exists x\exists yR(x,y)$$

Show that this holds in first-order logic.

Exercise 7.8.3. Show that the following holds in first-order logic:

$$\{\forall x\forall y(R(x,y) \to R(y,x)), \forall x\forall y\forall z((R(x,y) \land R(y,z)) \to R(x,z)), R(a,b), \neg R(a,c)\}$$
$$\vdash \exists x\neg R(x,b)$$

This is a hard one! If you can solve this, you have mastered first-order tableau deductions. Here is a hint if you need one. Note that we have Symmetricity, Transitivity, $R(a,b)$ and $\neg R(a,c)$. Moreover, the negation of the conclusion entails that $\forall xR(x,b)$, so in particular, $R(c,b)$. Now if $R(c,b)$, then by Symmetricity, $R(b,c)$ follows. Then, by Transitivity, $R(a,c)$ follows (given $R(a,b)$). But $\neg R(a,c)$ is a premise, so we have a contradiction.