Learning Logic Backwards

Peter Susanszky

Chapter 1

Formal languages

The purpose of this textbook is to teach you some of the ins and outs of formal logic. However, if you have ever come across an introductory logic textbook, you will see that this one is nothing like it. Logic is a very abstract science, and this abstract nature is very intimidating at first. It certainly was for me when I first started! I think this is hindered, not helped by immediately trying to relate the formal stuff to bigger, and even harder, topics like natural language (think English) and the nature of human reasoning.

Instead, we will start by getting familiar with the formal stuff itself. The idea is to make seemingly complicated ideas easy to grasp and understand (or at least, eas*ier*). After all, in formal logic, being formal is very important. It is part of what it means to be rigorous. On the other hand, some of this rigour can make things seem more complicated than they really are, and that gap is exactly what we will be focusing on.

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:

●

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:

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:

Proposition 1. *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 PRODUCE 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)

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 A more elaborate writing game

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 $\mathcal{L}_{\mathbb{Z}}$?

1.3.1 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. 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 1.3.2. 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. $\{\}$
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 1.3.3. 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 1.3.4. Look at the first formula of Exercise 1.3.3 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 1.3.2 and 1.3.3, there were two sequences of symbols of the alphabet that are not formulas of the language[1] Here, it actually 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 1.3.5. Derive all the other formulas of the language $\mathcal{L}_f$ from Exercises 1.3.2 and 1.3.3. 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.

[1] Specifically, number 4 in each.

Then, we get the following formulas from above:

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

Exercise 1.3.6. Rewrite all the formulas of $\mathcal{L}_f$ from Exercises 1.3.2 and 1.3.3 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 3 folders, the second says there is only one.

Exercise 1.3.7. How many files or folders are there in root if the formula specifies the file structure $f_1, f_2, \{\{f_2, f_4\}, \{f_3, f_3\}\}$?

1.4 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 1.4.1. 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.

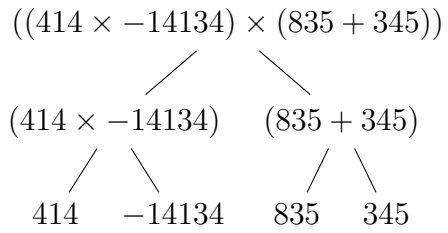
Exercise 1.4.1. Give an example of an arithmetic expressions with three (not necessarily distinct) numbers connected by two (not necessarily distinct) arithmetic operations where how the parentheses are placed does not matter. Then, give another example where how the parentheses are placed does matter. 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

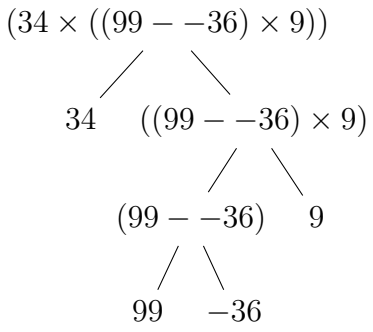
we get to the desired formula. Moreover, more complex expressions are formed by putting together two simpler expressions with an arithmetic operator, flanked by the left and right parentheses.

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 have a formula can be broken down into its constituent parts.

There is a question which formulas to 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 1.4.2. Decide which is the main operator in each of these formulas:

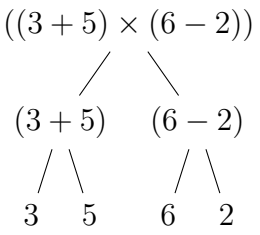
1. $((93 + 73) \times (43 - 15))$
2. $(975 \times (44 + 1))$
3. $((77 \times 3) - 33)$
4. $(((3 \times (5 + 4)) \times 9) - 7)$
5. $((((3 \times 5) + 4) \times 9) - 7)$
6. $(((3 \times 5) + 4) \times (9 - 7))$

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 1.4.3. Construct syntax trees for each of the six formulas in Exercise 1.4.2. 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:

$$\begin{array}{c} ((3+5)\times(6-2)) \\ \diagup \quad \diagdown \\ 8 \quad (6-2) \\ \quad\quad \diagup \quad \diagdown \\ \quad\quad 6 \quad 2 \end{array}$$

Then, we compute the right side, getting:

$$\begin{array}{c} ((3+5)\times(6-2)) \\ \diagup \quad \diagdown \\ 8 \quad 4 \end{array}$$

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.

1.5 The language $\mathcal{L}_0$

1.5.1 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: $\mathfrak{c}_1$ is the first constant, $\mathfrak{c}_2$ is the second constant, $\mathfrak{c}_3$ is the third constant, 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 succintly, 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.

1.5.2 Predicates

Second, similar to constants, we also have infinitely many symbols called *predicates* in our alphabet. These will be denoted by $\mathfrak{P}$. 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:

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

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

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

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

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

Just as before, this may be represented a lot more succintly by simply saying: $\mathfrak{P}_n^k$ 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.

Exercise 1.5.1. 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. $\mathfrak{P}_{1000000}^1$

1.5.3 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. 9), 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 $\mathfrak{P}_n^k$ (of arity k), the connectives $\wedge, \vee, \neg, \rightarrow$, the left '(' and right ')' parentheses, and the comma ','.

1.5.4 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 $\mathfrak{P}_i^k$ is a predicate of arity k, then $\mathfrak{P}_i^k(\mathfrak{c}_{n_1}, ..., \mathfrak{c}_{n_k})$ 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 $\mathfrak{P}_n^k$ wears on its sleeves how many individual constants it demands – namely, k many! Note however that these individual constants need not be distinct. For example, for $\mathfrak{P}_5^2$, we may write $\mathfrak{P}_5^2(\mathfrak{c}_5, \mathfrak{c}_5)$ just as well as $\mathfrak{P}_5^2(\mathfrak{c}_5, \mathfrak{c}_3)$.

Exercise 1.5.2. Determine if the following are (atomic) formulas of the language $\mathcal{L}_0$. In each case, explain your reasoning.

1. $\mathfrak{P}_1^3(\mathfrak{c}_3, \mathfrak{c}_5, \mathfrak{c}_1)$
2. $\mathfrak{P}_4^1(\mathfrak{c}_{66})$
3. $\mathfrak{P}_5^5(\mathfrak{c}_5, \mathfrak{c}_4, \mathfrak{c}_3, \mathfrak{c}_2, \mathfrak{c}_1)$
4. $\mathfrak{P}_6^2(\mathfrak{c}_2, \mathfrak{c}_2)$
5. $\mathfrak{P}_6^2(\mathfrak{c}_2)$
6. $\mathfrak{P}_3^2(\mathfrak{c}_4, \mathfrak{c}_9, \mathfrak{c}_1)$

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

1.5.5 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 somewhow.

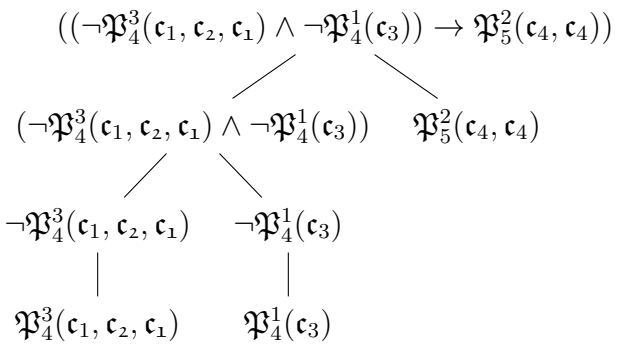
As an example, let's analyze the formula:

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

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:



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

1.5.6 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.

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 $\mathfrak{P}^3_n$ 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: $\mathfrak{P}^3_n(\mathfrak{c}_k, \mathfrak{c}_j, \mathfrak{c}_l)$ 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 $\mathfrak{P}^2_n(\mathfrak{c}_k, \mathfrak{c}_k)$ 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.

1.6 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:

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

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

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

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:

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

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:

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

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, and S and Q are identical.

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 2.2.1. 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 examplify 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 succintly 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 $S = \{Leonardo, Taylor\}$ with itself?
2. Is the set $Q = \{\langle Taylor, Leonardo \rangle, \langle Leonardo, Leonardo \rangle\}$ 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 situtation). But using this blueprint, any relation can be represented, starting from such pairs to more elaborate ones, like 'x is in berween 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.

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 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 $\langle \text{NYC}, \text{LA}\rangle \in A$, and $\langle \text{LA}, \text{Tokyo}\rangle \in A$, then $\langle \text{NYC}, \text{LA}\rangle \in A$. 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).

☞ 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, i.e., 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*.

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:

$$\langle 0, 2\rangle$$
$$\langle 1, 3\rangle$$
$$\langle 2, 4\rangle$$
$$\langle 3, 5\rangle$$
$$\langle 4, 6\rangle$$
$$\vdots$$

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:

$$\langle \text{Leonardo DiCaprio}, 49 \rangle$$
$$\langle \text{Dua Lipa}, 28 \rangle$$
$$\langle \text{Curtis Jackson}, 38 \rangle$$
$$\langle \text{Taylor Swift}, 34 \rangle$$
$$\langle \text{Christian Bale}, 49 \rangle$$
$$\vdots$$

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 Leonardo DiCaprio and Christian Bale are the same age at the moment of writing this book (though only for 30 days each year). 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 numbers i, j such that $n \times 2 = i$ and $n \times 2 = j$. 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, $\{1, 2, ..., n\}$.

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. Let $f : B \to A$ just like in Q. Is f surjective and/or injective? Is f one-to-one?

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?

2.7 Set-builder notation and Russell's paradox

There is a convention 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.

2.7.1 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 gave 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^{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's paradox. Suppose you are in a town that has a barber with a seemingly straightfoward rule. The barber shave 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

Semantics

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 formulation of expressions in our syntax.

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 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.

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 meaning to our atomic formulas, we first have to specify the meaning of constants and predicates. Then, we shall be able to compute the meaning of our atomic formulas. In turn, this will allow us to specify the meaning 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 what 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 side of semantics, 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 bijectivity, 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, **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 **D**, let's call it **P**, which consists of just the physicists in our domain. In set-builder notation, we can say:

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

P here is a subset of the domain **D**, since by definition, every $x \in \mathbf{P}$ is also in **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)
$\mathfrak{P}_1^1$	$\mapsto$	**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 **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')
$\mathfrak{P}_2^1$	$\mapsto$	**S** ('is a singer')
$\mathfrak{P}_3^1$	$\mapsto$	**N** ('is a number')
$\mathfrak{P}_4^1$	$\mapsto$	**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 $\mathfrak{P}_1^2$, 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)
$\mathfrak{P}_1^2$	$\mapsto$	**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)
$\mathfrak{P}_1^2$	$\mapsto$	**L** ('loves')
$\mathfrak{P}_2^2$	$\mapsto$	**F** ('is the favorite number of')
$\mathfrak{P}_3^2$	$\mapsto$	**B** ('is a sibling of')
$\mathfrak{P}_4^2$	$\mapsto$	**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 give 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 a lot of 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...

3.2.4 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.

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 $\mathbf{S}$ 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 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 $\mathbf{D}$ and an interpretation $\mathbf{I}$ that would tell me what these formulas mean! 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 $\mathbf{S}$.