Discovering Logic

Iliano Cervesato
Carnegie Mellon University

Draft of November 30, 2015

Contents

Preface

This document collects some course notes for a new experimental course introduced on the Qatar Campus of Carnegie Mellon University in the Spring semester of 2010. *Discovering Logic* (15-199) is an introduction to logic for computer science majors in their freshman year. It targets students who have had little or no exposure to logic and has the objective of preparing them for sophomore classes which require proficiency with understanding formal statements expressed in English, elementary reasoning skills, and a sense of mathematical rigor.

These notes currently cover just propositional and predicate logic, and they do so at a motivational level only: they present some elementary concepts and techniques on the basis of intuition, but shy away from technicism. In particular, these notes do not contain proofs, except for a few simple examples introduced for motivational purpose. The tone is intentionally lightweight and fun, without however compromising on rigor.

I. Cervesato
Doha, 28 April 2010

Chapter 1

What is Logic?

Logic is the study of reasoning. But what is *reasoning*? We can think about reasoning in two ways, at least:[1]

- Reasoning is extending what we know already by giving reasons. This is *learning* new things from old things, discovery from what is known if you want. Then, logic gives us a methodology to carry that out.
- Reasoning is making sense of the reasons that justify an argument. This is *analysis* of the known to make sure it is correct. Then, logic is the study of what counts as good reasons for what we know and learn, and why.

These two meanings are interconnected: the reasons why we can produce new knowledge are what makes it correct. Both are well summarized in one of the earliest attempts at defining logic, attributed to the first logician ever according to some: Aristotle (384–322BC). In his (translated) words,

> *Logic is new and necessary reasoning.*

"New" gives the sense of learning, while "necessary" presupposes correctness. This ancient definition, over 2,300 years old, will be the beginning of our discovery of logic.

1.1 Inferences

Consider the following sentence:

> *Doha is in Qatar and Sidra is in Doha, so she is in Qatar.* (1.1)

This is a convincing argument, isn't it? Let's dissect it. It has two parts separated by the word "so". This word, like many synonyms like "therefore", "consequently", etc,

[1] Like with all somewhat vague concepts, people have different opinions.

is very important, so important that we will underline it, writing "so": it separates the known from the unknown, the old from the new. This word is an indication of reason. Let's look at what comes before and after it:

- The part before "so" lists the things we know already, here *"Doha is in Qatar"* and *"Sidra is in Doha"*. They are called the *premises*. We use words like "and" to separate them, and will sometimes underline them too.
- The part after "so" is the new thing that we learn when the premises are true, here *"Sidra is in Qatar"*. It is called the *conclusion*.

Going from things that we know to things that we learn is called an *inference*. So, "Doha is in Qatar and Sidra is in Doha, so she is in Qatar" is an inference. Inferences have a number of premises (zero or more) and a single conclusion.

Reasoning is making inferences: learning new things from things that we previously knew using inferences such as the above. Logic is therefore the study of inferences.

1.2 Valid Inferences

Now, the above inference makes intuitive sense, doesn't it? It is a *valid inference*. But what does this mean? What makes an inference valid? This is where logic really enters the stage. Let's look at another example:

Doha is in Qatar and Sidra is in Doha, so she went for a walk. (1.2)

This inference does not ring right. What does Sidra's walk have to do with her being in Doha or Doha being in Qatar? In (1.1), the conclusion was a necessary consequence of the premises: given that "Doha is in Qatar" and that "Sidra is in Doha", there is no way she can be in any country besides Qatar. Here it is not necessary: Sidra could very well be in Doha, which is in Qatar, and watching television on a sofa — no walking involved. This inference is not valid.

What is the difference between (1.1) and (1.2)? In the first case, the premises were true and the conclusion had to be true: the conclusion was *necessary* given the premises (ah! Aristotle again!). In the second example, premises and conclusions were independent: the conclusion could be false even when the premises were true. Its truth was not necessary given true premises. This gives us a basis for a tentative definition of valid inference:

> An inference is *valid* if it is not possible that its premises are true and its conclusion is false.

Now, what happens when the premises are not true? Consider another inference:

Doha is in France and Sidra is in Doha, so she is in France. (1.3)

This is clearly wrong ... or is it? Well, if Doha *were* in France, then Sidra being in Doha would mean that she is in France. The first premise, *"Doha is in France"*, is clearly false (to the displeasure of the French, who could do with some hydrocarbons reserves on their own soil), but if it *were* true, the conclusion would be true. So the inference is valid.

This means that an inference can be valid even if one (or more) of its premises is false. What matters for validity is that *it is impossible that all the premises be true and yet the conclusion be false* (as in our last example). So our definition of valid inference does hold water: we will keep it as what it means for an inference to be valid.

Logic is the study of valid inferences, specifically of what makes them valid and what we can do with them.

1.3 Formal Reasoning

Consider yet another inference about Sidra's travels:

Mikulat is in Kiribati and Sidra is in Mikulat, so she is in Kiribati. (1.4)

Chances are that you have no idea whether Mikulat and Kiribati are real places, let alone where they are located.[2] Yet, this inference sounds alright, even if you have never heard about these places, and therefore do not know anything about whether the premises and conclusion are true or false. The reason it sounds alright is that it has exactly the same shape as inferences (1.1) and (1.3), with just the names of the places replaced in a consistent manner.

What we are discovering here is that we can tell whether an inference is valid even if we don't fully understand what it is about. Let's take an even more extreme example:

東京 *is in* 日本 *and Sidra is in* 東京, *so she is in* 日本. (1.5)

We have no idea what 東京 and 日本 are, yet this sentence has again the same structure, which makes it valid.[3]

We can indeed generalize the above inferences by abstracting away the name of the places used in these sentences — logic is big on generalization and abstraction. We obtain the following *inference pattern*:

"X is in Y and Sidra is in X, so she is in Y" (1.6)

which is valid no matter what we write for X and Y. Whatever we choose X and Y to be, the conclusion cannot be false if the premises are true.

[2] As a matter of fact, they do exist: I used Google Maps to pick the smallest village I could find in Tanzania — that's Mikulat, and then I went to Wikipedia to see if there was any country in the world that I had never heard about — there were plenty and the island nation of Kiribati in the Southern Pacific was one of the choices there.

[3] In case you are wondering (but maybe you are not, which is good), 東京 is Japanese for Tokyo and 日本 for Japan.

The above inferences are not valid because of what they say (which we can't always understand), but because of the way they look. *What matters is the structure of these sentences, not their meaning.* Indeed, logic starts with the study of *linguistic patterns* of valid reasoning, and abstracts them into mathematical symbols. Why? Because the result is less ambiguous than natural language, ultimately simpler, and possibly mechanizable. The word "logic" is often qualified to make this clear. We speak of

formal logic to indicate that we are interested in the *form* of an argument, not in its contents. Going back to Sidra's travels above, we pounded on that argument so much that what it actually said didn't matter any more, and in the end we just looked at the form of the sentence.

symbolic logic to stress that language about facts and ideas is replaced with *symbols*. Symbols *represent* facts and ideas, but logic manipulates just the symbols.

In the rest of this book, we will abandon meaning in favor of form.

1.4 Types of Reasoning

Having traveled with Sidra all around the world, let's use another inference as an example:

> *If the burglar had broken through the kitchen window, there would be footprints outside, but there are no footprints outsides, so the burglar didn't break in through the kitchen window.* (1.7)

Intuitively, this is another valid inference.

The particular form of reasoning that it and the previous examples embody is called *deductive reasoning*. This inference allows us to *deduce* the conclusion when we know that the premises are true: then the conclusion is new knowledge of which we are 100% sure (this knowledge is necessary). This is the form of reasoning we will be interested in throughout this book.

There are other forms of reasoning however. Here's another example:

> *Tom coughs a lot and has yellow stains on his fingers, so he is a smoker.* (1.8)

This makes sense too! Note however that now the relationship between the premise and the conclusion is quite different from our earlier examples. Here, the conclusion is not so much a consequence of the premises (Tom may have a cold and touched turmeric) but a probable explanation of the premises. This form of reasoning is called *inductive reasoning*. Doctors, mechanics and lawyers use it all the time, and students too, sometimes, when they are working on an assignment. For how widespread as it may be, this book will not be concerned with inductive reasoning.

1.5 Logic and Language

Deductive reasoning is *universal*: inferences (1.1) and (1.3–5) will convince anybody in the world with basic mental capabilities. The language in which they are expressed does not matter for their validity. Reasoning is rooted in language and yet it transcends language. It is an innate human ability, like walking, say.

Can we then separate reasoning from language? This is precisely what logic does. Remember? we called it "formal" and "symbolic". That's because it focuses on the form of an argument, not the way we formulate it.

This has a lot of advantages:

- First of all, logic is *language independent*, so that a German, an Arab and a Cantonese getting together will understand the exact same inferences as valid, even if they cannot communicate very well.
- Second, a same argument can be expressed in many different ways even in English. There are many sentences that say the same thing. Logic is more orderly.
- Logic completely resolves the problem of ambiguity. There is no way to understand the same thing in multiple different ways.
- As any linguist will confirm, logic is a lot simpler than any natural language.
- Finally, since computers are good at manipulating symbols, maybe by focusing on symbols we can automate reasoning.

Logic has also a lot of shortcomings:

- Natural language is incredibly flexible and has a lot of reasoning possibilities. By contrast logic tends to concentrate on a few, very common, forms of reasoning (more on this later).
- Logic is not natural: not only are you taking a course on it, but your little brother can do a lot of reasoning without having a clue of all the symbols we will be introducing shortly.

By and large, logic has a place in the world, especially the world of Mathematics and Computer Science. It will never replace language, not even to make inferences, but it can help quite a bit.

1.6 Which logic?

We said earlier that logic isolates the linguistic patterns of valid reasoning. How do we go about it? What linguistic patterns should we look at? Well, there are a lot of them. Consider the following simple sentence:

> *The sun shines.*

This is the description of a basic fact that can be either true or false. No sweat here. Well, let's extend it a bit:

The sun shines
somewhere in the world.

This introduces a spatial element. This means that this sentence can now participate in reasoning about locations. This requires isolating linguistic patterns to deal with space, which leads to a spatial logic. Let's extend it further:

The sun shines
somewhere in the world
at any time.

Now we have also a temporal element. To make inferences using this sentence involves devising a logic that allows reasoning about time, a temporal logic. But we can do even more:

Sidra believes that
the sun shines
somewhere in the world
at any time.

Now, this sentence forces us to reason about people's point of view: a sentence is not universally true or false, but it depends on what each individual believes. Instilling reasoning mechanisms to deal with beliefs yields what is called an epistemic logic.

This is already very complicated, and we have barely scratched the surface. Reasoning has many dimensions, and many many linguistic patterns can be isolated into a logic. Which do we pick?

Rather than trying to study all possible linguistic patterns at once, logic tends to proceed using a divide-and-conquer strategy:[4]

- It starts with very simple, very common linguistic patterns.
- It then studies the different dimensions separately.

Therefore, there is not one logic, but many. Not only this, but logic as a field is very alive, with dozens of conferences on it every year.

In this book, we will look at two such logics: *propositional logic* and *predicate logic*. Both belong to the first category above, the one that looks into "very simple, very common linguistic patterns". Indeed,

- they are very simple,
- they are very well understood (but remember that logic is alive: many logicians are working on understanding them even better as you read this sentence!),

[4]This is a bit of a simplification, but not too far from the way logicians actually do things.

- predicate logic is powerful enough to make sense of all of mathematics.[5]

This last justification is puzzling to say the least: if predicate logic is so powerful, why bother studying all those more complicated spatial, temporal, epistemic, whatever logics? Great questions! Here are a couple of answers:

1. *For the same reason we don't normally program in assembly language.* Predicate logic is very low level and it is often convenient to work at a higher level of abstraction.

2. *To get better automation.* Predicate logic at large is not easy to automate (we will see what this means), while more specialized logics behave better.

3. *To look for more fundamental concepts.* If we compare reasoning to physics, predicate logic gets us to the level of the atoms: this is great because we can use it to describe all the objects around us. However, atoms are themselves built out of smaller entities, like electrons and neutrons, which are in turn composed of quarks and other particles. The same happens with reasoning: predicate logic can itself be understood on the basis of smaller logical particles, which also explain a lot of other forms of reasoning.

1.7 *Exercises*

1. Is the following a valid inference?

   > *To get an* A *in this class I need to get at least 90% overall, I can get 90% in each homework and 95% in every other thing that is graded. Therefore I will get an* A *in this class.*

   What are the premises and conclusions? What kind of dimension to you need to work with on top of a core logic about truth and falsehood to determine if it is valid?

2. We defined an inference as having *zero* or more premises and one conclusion. What can you say of an inference with zero premises? Can you think of an example?

3. For this exercise, you will need to read the book *"Logicomix: an Epic Search for Truth"* [4] (yes, the entire book) and write a letter to a friend about it (or if your prefer a review for Amazon.com).

   The first part of your letter should describe the book to your friend. Tell him/her what the story is about, how the book is structured, what the main characters, the main themes, the historical context are, etc. You are encouraged to expand on this list — the more creative you are the better! For example, does some character remind you of somebody you know?

[5] Specifically, the form of predicate logic known as *second-order logic*. It is really cool, but a bit beyond the scope of this book.

In the second part of the letter, share your personal impressions with your friend. What parts did you like best? Why? Which did you dislike? Why? Which aroused your curiosity? Why? Make sure to motivate your arguments: writing just "I liked it" won't cut it.

In the third part, formulate at least three questions that you would like to see answered in this course. They can be about anything you expect the course to be about or anything in the book that aroused your curiosity. The suggested length of your letter is between 3 and 5 pages. It's about quality, not quantity!

Here are some evaluation criteria you should aim toward as you write your letter:

- *Form*: Your letter is expected to be grammatical and to develop a sensible argument. Your sentences and paragraphs should flow into each other without any rough edges. Your ideas should be developed progressively. You are welcome to use quotes and citations as long as you reference them.
- *Completeness*: Your letter should do a good job at describing the book to your friend so that he/she can form a well-informed opinion about whether he/she wants to read it. There are a lot of aspects that are clearly important: you are expected to cover them all.
- *Creativity*: Do not just state the obvious. Your letter should provoke thinking and curiosity, and it should be pleasant to read.

Part I

Propositional Logic

Chapter 2

Propositional logic

Having established that logic isolates common linguistic patterns used in reasoning, we begin our investigation with some of the most elementary patterns, and in a sense the most profound. It took logicians several thousands of years to properly identify them and understand how they work. This is the language of *propositional logic*. In this chapter, we introduce it and see how we can use it to express sentences and inferences. We will see how to make sure those inferences are valid in the next chapters.

2.1 Statements

Recall what we are trying to do: reason about when inferences are valid. We have defined an inference to consist of zero or more premises and one conclusion, and for an inference to be valid, it must not be possible that all premises be true and yet the conclusion be false. This suggests that we need to look at phrases that can be either true or false. Sentences that can be either true or false are called *statements* or *propositions*. When we speak or write, we use statements all the time. Note however that we also say things that are not statements. For example, a question is neither true nor false: it is just a question. The following table shows some examples in each category.

Statements:

- Sidra is in Doha
- Doha is in France
- The sun shines
- $10 < 7$
- Sidra is in class and she is paying attention
- All humans are mortal
- Every prime number greater than two is odd

Non-statements:

- How are you?
- Where is the exit?

Questions

- Bang!
- Idiot!

Exclamations

- Sidra ...
- Hey you!

Invocations

- Go away
- Finish your lunch

Orders

If we look carefully at the left column, we see that some statements are simpler than others.

- Take *"Sidra is in Doha"*. Clearly, it can be either true or false, but no part of it taken by itself is a sentence that is true or false. Indeed, removing any word yields an ungrammatical phrase. Statements like this are called *atomic* or *elementary*. Other atomic propositions in our list are *"Doha is in France"*, *"The sun shines"* and "$10 < 7$".

- By contrast, *"Sidra is in class and she is paying attention"* is composed two (elementary) statements, *"Sidra is in class"* and *"she is paying attention"* (both of which are either true or false) combined into a larger statement by the word *"and"*. This is a *composite statement*. It will be convenient to highlight the parts of a composite proposition that are elementary, here:

  " *Sidra is in class* and *she is paying attention* "

  The last two examples are also composite statements, but in a different way: they don't simply combine smaller statements into bigger statements, but they generalize them somehow.

In language, we have ways to combine basic statements that are true or false into bigger statements that are also true or false. There are a lot of ways to do so. *Propositional logic* looks at a couple of very simple ways statements are formed from atomic propositions that come up all the time. It focuses on simple connectives like *"and"*. Some common forms of generalization will be handled by predicate logic in Chapter 5.

2.2 Propositional Connectives

What are those simple connectives that allow building bigger statements from smaller statements? You can come up with a lot of them, but there are a few that stand out: they are very common and they can express all others (in a sense to be made precise later). They are known as the *Boolean connectives* in honor of the mathematician and philosopher George Boole (1815–1864) who studied them in depth (for the time). Here they are:

- *"and"* and synonyms like *"furthermore"*, *"moreover"*, *"but"*, *"additionally"*, and many more. This connective is called *conjunction* and connects two sentences in a bigger sentence: it is a *binary* connective. Most logicians use the infix symbol $\wedge$ to denote it.

- *"or"* and linguistic constructions such as *"maybe ... maybe ..."* and *"either ... or ..."*. This binary connective is called *disjunction* and logicians like to write it $\vee$.

- *"not"* (also encountered as *"it is not the case that ..."* and *"... is false"*). This connective, which applies to single formulas — it is *unary*, is called *negation* and is typically written $\neg$ prefix.

- *"implies"* and its synonyms like *"if ..., then ..."*, *"therefore"*, *"only if"*, *"... is a necessary condition for ..."*, *"... is a sufficient condition for ..."*, and many many others. This binary connective is known to logicians the world around as *implication* and they write it infix as $\rightarrow$ (some write $\supset$, but we won't use this notation).

  All these synonyms for *"implies"* are confusing, aren't they? Take *"only if"*: does it mean "implies"? does it mean "is implied by"? And what about *"if"* in the middle of a sentence? Not to say anything about those necessary and sufficient conditions math books are so fond of. Let's make sense of all this.

  - Consider the phrase *"The sun shines if it is noon"*. What it says is that whenever it is noon, the sun is out there shining, which is the same as *"If it is noon, then the sun shines"* or *"It is noon implies that the sun shines"*. Another way to look at it is that if we know that it is noon, then the sun must be shining: it being noon is a sufficient condition for the sun to shine. Note that these phrases do not exclude that the sun shines at other times, for example at 2pm: all they say is that at noon we are guaranteed that it shines.
  - Next, take the phrase *"The sun shines only if it is noon"*. This time, the meaning is that it'd better be noon for the sun to shine, i.e., the sun won't be shining unless it's noon. We can express this as *"If the sun shines, then it is noon"* and *"The sun shines implies it is noon"*. That's the opposite of the previous example — the difference one little word can make! Note that this phrase does not guarantee that the sun will be shining at noon —

it could be overcast for example. Therefore, it being noon is a necessary condition for the sun tho shine.

- *"if and only if"* is known as *biimplication* and is written infix using the symbol $\leftrightarrow$. Biimplication looks like the combination of an *"if"* and an *"only if"*. In fact, the phrase *"The sun shines if and only if it is noon"* means that the sun shines exactly when it is noon, always. It being noon is a necessary and sufficient condition for the sun to shine.

Although not properly connectives, it is useful to introduce notation for the statement that is always true and the statement that is always false:

- *Truth* is generally written $\top$.
- *Falsehood* is often written $\bot$.

2.3 Turning Sentences into Logic

Now, we can start turning sentences into logical formulas (that is, linguistic patterns such that only their form matters). Consider the following sentence:

Sidra is in class and she is paying attention.

We immediately recognize "and" as a linguistic pattern for conjunction, while the two sentences on either side are just atomic. We can then rewrite it as follows in propositional logic:

$$\boxed{\textit{Sidra is in class}} \wedge \boxed{\textit{she is paying attention}}.$$

As we do this, we will often expand pronouns, rewriting the contents of the second box as *"Sidra is paying attention"*, for example.

Let's do the same not with a sentence, but with a whole inference. Consider again inference (1.7) from last chapter:

If the burglar had broken through the kitchen window, there would be footprints outside, but there are no footprints outsides, so the burglar didn't break in through the kitchen window.

Here, *"If . . . then . . . "* is a linguistic pattern for implication, and there are a couple of negations. This gets us the following translation in propositional logic:

$$\boxed{\textit{the burglar had broken in through the kitchen window}}$$
$$\rightarrow \boxed{\textit{there would be footprints outside}} \ \underline{\textit{and}} \ \neg \boxed{\textit{there were footprints outsides}},$$
$$\underline{\textit{so}} \ \neg \boxed{\textit{the burglar broke in through the kitchen window}}.$$

What about the word *"and"*? Isn't it a conjunction? Here we have a choice: we can see it either as a way to separate the various premises of an inference (that's what we did just now) or we can see it as a connective, in which case we would get the following inference:

$($ *the burglar had broken in through the kitchen window*
$\rightarrow$ *there would be footprints outside* $) \wedge \neg$ *there were footprints outsides*,
so $\neg$ *the burglar broke in through the kitchen window*.

As we will see, the two translations are equivalent in a sense that we will make precise in chapters to come.

So far, we have replaced the linguistic patterns for conjunction, disjunction, etc, with propositional connectives, leaving the atomic statements as English sentences. This quickly becomes a pain: they are pretty long and, as you can notice, they contain small linguistic variations among sentences meant to represent the same thing (e.g., *"there* would be *footprints outside"* and *"there* were *footprints outside"*). Logicians, being the practical types they are (sometimes), have introduced abbreviations. Rather than using those long sentences, they like to represent statements with short strings, often a single letter. In the first example, they could define

$$\begin{array}{rcl} C & = & \text{"Sidra is in class"} \\ A & = & \text{"Sidra is paying attention"} \end{array}$$

This is sometimes called a *dictionary* or a *reading key*, and C and A are called *propositional letters*. Then, the sentence in the first example becomes

$$C \wedge A$$

Notice that, by introducing propositional letters as abbreviations, we have completely eliminated the contents of the formula, focusing uniquely on its form. This becomes even more useful when looking at full inferences. Consider our second example with the following dictionary:

$$\begin{array}{rcl} B & = & \text{"the burglar broke in through the kitchen window"} \\ F & = & \text{"there were footprints outside"} \end{array}$$

Then (our second reading of) that inference becomes

$$(B \rightarrow F) \wedge \neg F \quad \underline{\text{so}} \quad \neg B$$

This is so much shorter!!! This is the kind of abstract inferences we will look at from now on.

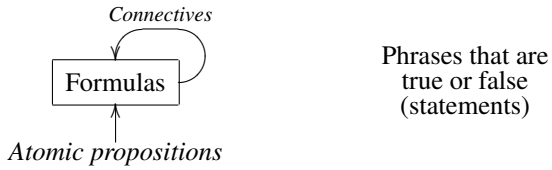
2.4 Propositional Formulas

A propositional formula is a piece of mathematics that we write using propositional connectives, propositional letters and possibly parentheses. We can define them formally as follows (note that "formally" implies that we are interested in what they look

like, not what they mean). We will write a generic propositional formula using the Greek letters φ ("phi", pronounced *"fAi"*) and ψ ("psi", pronounced *"sAi"*), possibly annotated with subscripts and primes — that will help us keep things straight.

- Any atomic proposition (propositional letter) A, B, ... is a propositional formula.
- $\top$ and $\bot$ are propositional formulas.
- If φ is a propositional formula, then $(\neg\varphi)$ is also a propositional formula.
- If φ and ψ are propositional formulas, then $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \to \psi)$, and $(\varphi \leftrightarrow \psi)$ are all propositional formulas.

Nothing else is a propositional formula. This definition is summarized by the following schema, which shows that formulas are obtained by combining other formulas using connectives starting from atomic propositions. We can think of the propositional constants $\top$ and $\bot$ as connectives that take zero arguments.



For practice, here is a well-formed propositional formulas:

$$((A \vee B) \leftrightarrow (\neg((\neg A) \wedge (\neg B))))$$

That's a lot of parentheses! Fortunately, logicians have developed conventions that allow putting parentheses only where necessary. First, the connectives have been assigned the following standard precedences:

$$1.\ \neg \qquad 2.\ \wedge,\ \vee \qquad 3.\ \to \qquad 4.\ \leftrightarrow$$

This means that $\neg$ has the highest precedence and $\leftrightarrow$ the lowest, so that $A \to A \vee B$ is understood as $(A \to (A \vee B))$ while $A \to A \leftrightarrow B$ stands for $((A \to A) \leftrightarrow B)$. This allows us to simplify the above formula as

$$A \vee B \leftrightarrow \neg(\neg A \wedge \neg B)$$

Now, only the parentheses around the conjunction are needed, otherwise the $\neg$ before them would cling to the formula $\neg A$.

The second widely adopted convention is to consider conjunction and disjunction to be associative, so that we can write $A \vee B \vee C$ instead of either $((A \vee B) \vee C)$ or $(A \vee (B \vee C))$.

Let's practice turning statements into formulas. This time, we will skip the boxes and also consider slightly more complicated examples. As a warm up, consider the following inference:

If chicken is on the menu, you shouldn't order fish, but you should have either fish or salad, so if chicken is on the menu you should have salad. (2.1)

Its premise and conclusion are composite statements. Let us define a dictionary that associates propositional letters with the elementary statements therein:

$$\begin{array}{rcl} C & = & \text{“chicken is on the menu”} \\ F & = & \text{“you should order fish”} \\ S & = & \text{“you should order salad”} \end{array}$$

Then, the above inference translates to the following simple form:

$$(C \to \neg F) \wedge (F \vee S) \;\; \underline{so} \;\; C \to S$$

As in the burglar's example, we could also have interpreted the word "but" as the separator between two premises rather than as a conjunction.

With all we know, we can finally write Sidra's travel inference logically. Here it is again:

Doha is in Qatar and Sidra is in Doha, so she is in Qatar. (2.2)

To express it in propositional logic, it is convenient to rewrite it slightly into:

If Sidra is in Doha then she is in Qatar and Sidra is in Doha, so she is in Qatar. (2.3)

Now, the implication is clearly visible. Then, it doesn't take much of a dictionary to produce the propositional inference

$$D \to Q \;\; \underline{and} \;\; D \;\; \underline{so} \;\; Q$$

As our last example, let's try to make logical sense of a simple (and not very useful) computer program. Here we use the C/Java syntax, where `&&` and `||` indicates a Boolean conjunction and disjunction, respectively.

```
if (x < 7 || (1 <= y && y <= 5)) {
  if (x < 7)
    y >= 6
} else y > 0
```

Observe that `1 <= y` is the opposite of `y > 0`, and similarly for `y <= 5` and `y >= 6` (if `y` is an integer). Then, we can take our dictionary to be

$$\begin{array}{rcl} x_7 & = & \texttt{x < 7} \\ y_0 & = & \texttt{y > 0} \\ y_5 & = & \texttt{y < 5} \end{array}$$

Translating *"if ... then ..."* is easy, but what about the *"else"* part? Well, the *"else"* part should apply exactly when the condition is false, so that *"if c then X else Y"* has the exact same meaning as *"if c then X, and if not c then Y"*. This gives us the key to translating the above program into logic. We obtain

$$\begin{array}{rl} & (x_7 \vee (\neg y_0 \wedge y_5) \to (x_7 \to \neg y_5)) \\ \wedge & (\neg(x_7 \vee (\neg y_0 \wedge y_5)) \to y_0) \end{array}$$

2.5 Valid Inferences

Now that we have abstracted the linguistic patterns we care about into a logic, let's go back to the original reason for doing all this. What can we say about when inferences are valid?

One approach is to list all the valid symbolic inferences we know of, putting them in a big dictionary. Then, to check whether some inference is valid, we simply look it up in this dictionary: if it is there good, otherwise it is not valid. (This is similar to the way we used to check the spelling of words before spell-checkers — a long time ago!) There are some obvious problems with this approach:

- There are infinitely many valid inferences. For example, if you think about it, the inference "$A_1 \wedge \ldots \wedge A_n$ so A_i" is valid for any positive value of n and for every $1 \leq i \leq n$. So, the very idea of building this dictionary is hopeless.

- Even if it were somehow possible, this does not address the issue of how we convince ourselves whether an inference is valid or not. How do we decide which inferences to insert in the dictionary?

Clearly, that won't work.

One really powerful observation that will help us out here is that the meaning of a logical formula is given by the meaning of its connectives and propositional letters, neither of which changes within the formula. Furthermore, the meaning of the connectives themselves is fixed whatever the formula: $\wedge$ combines subformulas in the same way no matter where it appears. So, all that matters is the behavior of the connectives. Therefore, the key observation to determine which inferences are valid is that any reasoning can be reduced to reasoning about the connectives in isolation. This is a key observation not just for propositional logic but for any logical language.

How do we use this observation? Remember our definition of valid inference: *it cannot be that all the premises are true but the conclusion is false*. This suggests two ways of proceedings, which will be explored in more detail in the next two chapters.

1. If we have a way to determine whether formulas are true or false based on the truth or falsehood of the propositional letters in them, then the above definition gives us an easy way to check whether an inference is valid: that's exactly when *it cannot be that all the premises are true but the conclusion is false*. This is what the *truth tables* will allow us to do in Chapter 3.

2. Alternatively, we can describe valid inferences as the composition of elementary valid inferences for each connective. That is, we build a dictionary that contains all the valid inferences for each connective in isolation, and then combine them to justify inferences for complex formulas. This is the *derivation method*, studied in Chapter 4.

2.6 *Exercises*

1. Consider the following passage:

   > *If it got dark because the sun was setting, it would be getting cool, but instead it is still hot even though the there was no more light.*

   - Identify the atomic statements *[Hint: there are just 3 of them]* and define a dictionary that allows you to abbreviate them into propositional letters.
   - Identify the connectives and, using your dictionary, express the passage in propositional logic.
   - Is this an inference? Is it valid?

2. Let A, B, C and D be the following atomic statements:

   - A = "The villain is French"
   - B = "The hero is Egyptian"
   - C = "The heroine is British"
   - D = "The movie is good"

   Translate the following statements into propositional formulas:

   (a) The hero is Egyptian and the movie is good.

   (b) Although the villain is French, the movie is good.

   (c) If the movie is good, then either the hero is Egyptian or the heroine is British.

   (d) The hero is not Egyptian but the villain is French.

   (e) When the heroine is British, the movie is good only if the villain is French or the hero is not Egyptian.

3. Using propositional letters for the atomic statements, translate the following statements into propositional formulas:

   (a) If prices go up, then housing will be plentiful and expensive, but if housing is not expensive, then it will still be plentiful.

   (b) Either going to bed or going swimming is a sufficient condition for changing clothes, however changing clothes does not mean going swimming.

   (c) Either it will rain or it will snow, but not both.

   (d) Whether Janet wins or loses, she will be tired.

   (e) Either Janet will win or, if she loses she will be tired.

4. Using propositional letters for the atomic statements, translate the following statements into propositional formulas:

   (a) If the horse is fresh, then the knight will win.

(b) The knight will win only if the horse is fresh and the armor is strong.

(c) Either a fresh horse or a strong armor is a necessary condition for the knight to win.

(d) The knight will win if and only if the armor is strong but the horse is not fresh.

(e) Neither the horse being fresh nor the armor being strong alone is a sufficient condition for the knight to win, but if both the horse is fresh and the armor is strong, then the knight shall win.

5. Same exercise again, but this time the sentences are in *Italian*. All you need to know is that *"e"* means "and", that *"o"* means "or", that *"se"* means "if", that *"allora"* means "then", that *"solo se"* means "only if", and finally that *"non"* means "not".

(a) *Se Anita vince le elezioni, allora le tasse verranno ridotte.*

(b) *Le tasse verranno ridotte solo se Anita vince le elezioni e l'economia rimane forte.*

(c) *Le tasse verrano ridotte se Anita non vince le elezioni.*

(d) *L'economia rimane forte se e solo se Anita vince le elezioni o le tasse verranno ridotte.*

(e) *Se l'economia non rimane forte e le tasse non verranno ridotte, allora Anita vince le elezioni.*

6. Let's translate some more sentences in propositional logic. This time the sentences will be in Chinese. Chinese??? All you need to know to do this exercise is that "和" means "and", that "或" means "or", that "如果" means "if", that "就" means "then", that "只有" means "only if", and finally that "沒" means "not".

(a) 如果下雨, 就撐雨傘

(b) 如果整數或偶數, 就乘二

(c) 如果沒溼, 就沒灑水

(d) 拿滿分只有他和你

Write the formula corresponding to each of these sentences. You don't need to say which character strings correspond to what propositional letter: just write the result. If you really want to show the mapping, use cut-and-paste to write the Chinese characters (unless you can type in Chinese, that is).

7. Let A, B and C be the following atomic propositions:

- A = "Roses are red"
- B = "Violets are blue"
- C = "Sugar is sweet"

Rewrite the following atomic formulas in English:

(a) $B \vee \neg C$
(b) $\neg B \vee (A \rightarrow C)$
(c) $(C \wedge \neg A) \leftrightarrow B$
(d) $C \wedge (\neg A \leftrightarrow B)$
(e) $\neg(B \wedge \neg C) \rightarrow A$
(f) $A \vee (B \wedge \neg C)$
(g) $(A \vee B) \wedge \neg C$
(h) $(A \rightarrow B) \wedge (\neg A \rightarrow C)$
(i) $A \rightarrow (B \rightarrow C)$
(j) $(A \rightarrow B) \rightarrow C$

8. Consider the following programming problem: you are given two arrays `arrA` and `arrB`, their elements are sorted in ascending order, and you want to merge them into a third array, `arrC`, whose elements are also ordered. How would you complete the condition of the `if` statement in the code snippet below to get the correct result? The atomic conditions you can work with are "`i<arrA.length`", "`j<arrB.length`" and "`arrA[i]<arrB[j]`". You can assume that `arrC` has already been created for you and that `arrA` and `arrB` do not have common elements.

```
int i=0;
int j=0;
int k=0;
for (; (i<arrA.length)||(j<arrB.length);) {
  if (...) {
    arrC[k] = arrA[i];
    i++;
  } else {
    arrC[k] = arrB[j];
    j++;
  }
  k++;
}
```

If this is easier for you, feel free to define an auxiliary variable for your test condition and use it in the "`if`". (Macho programmers do not define auxiliary variables, but nobody is able to read their code — good programmers do a lot of things that macho programmers don't!) If you find a better way to write this loop, feel free to include it in your answer.

9. One daily activity that allows you to use propositional logic is searching the Internet. Surprised? Look it up! Go to your favorite search engine and look for how it supports using Boolean connectives to guide the search. Write a short essay that lists what connectives are available in that search engine, gives a few examples of queries that use them, and compare the results. Does this engine support other mechanisms besides Boolean operators to guide the search? As usual with essays, go beyond the obvious and help me learn something I didn't know. For example, some of you may want to write about a search engine that is not Google, or about Google searches that are not about web pages. Your essay

should spell-check, be grammatically correct, be pleasant to read, and especially be your own.

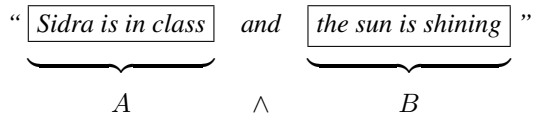
Chapter 3

Truth Tables

There is a simple way to determine whether a propositional formula is true or false on the basis of the truth or falsehood of the propositional letters that appear in it. This is based on a construction known as *truth tables*. It is then a short jump to deciding whether an inference is valid: by applying the definition, *it is valid if it never happens that the conclusion is false when all premises are true*. In this chapter, we will examine how to make this intuition precise.

3.1 Evaluating Formulas

Consider the English statement *"Sidra is in class and the sun is shining"*, which we would have decomposed as follows in Chapter 2:



Being a statement, it is either true or false, and so are the atomic statements *"Sidra is in class"* (A) and *"the sun is shining"* (B) that appear in it. Intuitively, the truth of the composite sentence depends on the truth of its elementary propositions. Let's see:

- The overall sentence, $A \wedge B$, is certainly true if Sidra is sitting on her seat and sun rays are basking the floor of the classroom, that is $A \wedge B$ is true if A is true and at the same time B is true.
- If she is in her seat and it rains outside, A is true but B is false. At the same time, the overall sentence $A \wedge B$ is false: it is not the case that *"Sidra is in class and the sun is shining"*.
- The overall sentence is similarly false when Sidra is sick and it is sunny, that is A is false but B is true.

- Finally, $A \wedge B$ is undoubtedly false on those rainy days where Sidra is sick, i.e., when both A and B are false.

In all cases, we could determine the truth or falsehood of $A \wedge B$ on the basis of the truth or falsehood of A and of B. If we think about "true" and "false" as *values*, we have determined the value of $A \wedge B$ in pretty much the same way as we determine the value of $x + y$ knowing the value of x and the value of y. Just like $+$, $\wedge$ is an *operator*. Given values for its arguments, it allows us to *calculate* the value of the conjunction.

Conjunctions and the other propositional connectives operate on values "true" and "false", which we will abbreviate as T and F, respectively. They are called *truth values* for obvious reasons. They differ from the numerical values in that there are just two of them, rather than infinitely many. This makes it easy to describe how each Boolean connective operates: for example we can build a "conjunction table" similar to the addition table that we have learned in elementary school:

B \ A	T	F
T	T	F
F	F	F

It is however convenient to format such tables slightly differently: we use one row for each of the four possible combinations of the truth values for A and B, and list the corresponding value of their conjunction next to it on that row. What we obtain in this way is what is known as the *truth table* of conjunction. While we are at it, it is convenient to add another column for disjunction, another for implication, and so on for all other propositional connectives. We obtain the following composite truth table for all connectives:

A	B	$\neg A$	$A \wedge B$	$A \vee B$	$A \to B$	$A \leftrightarrow B$
T	T	F	T	T	T	T
T	F	F	F	T	F	F
F	T	T	F	T	T	F
F	F	T	F	F	T	T

All are pretty intuitive, except for $\to$. Why should it be true when the antecedent is false? We will see this in a minute. We have not displayed the trivial columns for $\top$ and $\bot$, which are always T and always F, respectively. Truth tables, pretty much in the format just shown, were devised by the logician Ludwig Wittgenstein (1889–1951).

In mathematics, once we know how to do additions and multiplications, it is easy to calculate the value of a polynomial, say $-2x^2 + 3x - 1$, for any given value of the variable x: for instance, for $x = 5$ this polynomial has value -36. In the same way, since truth tables give us a way to "do" conjunctions, disjunction, etc, we can *calculate* the truth value of a complex propositional formula for any truth value of its

propositional letters. For example, take the formula $\neg(A \vee \neg B)$ and let's calculate its truth value when A is T and B is F.

$$\begin{array}{ccccccc} & & T & & F & & \\ & & \Downarrow & & \Downarrow & & \\ \neg & (& A & \vee & \neg & B &) \\ & & & & & \underbrace{}_{T} & \\ & & & \underbrace{}_{T} & & & \\ & \underbrace{}_{F} & & & & & \end{array}$$

The truth table for $\neg$ tells us that $\neg B$ evaluates to T when B is F. Then, the truth table for $\vee$ gives us a way to calculate that the truth value of $A \vee \neg B$ is T since both arguments are T. Finally, the truth table for $\neg$ gives us the final result, F, because its argument has value T.

Plotting the values of the polynomial $-2x^2 + 3x - 1$ on a graph as the value of x varies gives us an overall picture. We can do the same with any formula φ: determine its truth value for all the possible combinations of the truth values of its propositional letters. That's the *truth table* of φ. For example, take again the formula $\neg(A \vee \neg B)$ and let's determine its truth value for all possible truth values of the atomic propositions A and B. We obtain the following table, with the overall result displayed in bold:

A	B	$\neg$	$(A$	$\vee$	$\neg$	$B)$
T	T	**F**	T	T	F	T
T	F	**F**	T	T	T	F
F	T	**T**	F	F	F	T
F	F	**F**	F	T	T	F
0	0	4	1	3	2	1

This is the truth table of the formula $\neg(A \vee \neg B)$. To understand the way we built this table, follow the indices at the bottom of every column:

0. We started by just listing all possible combinations of values for A and B under the two leftmost columns.

1. Then, we copied the A-column under every occurrence of A in the formula, and similarly for B.

2. The only subformula we could calculate knowing just the values of A and of B is $\neg B$. We used the truth table for $\neg$ to compute it and wrote the result under that instance of $\neg$.

3. Knowing the truth values of A and $\neg B$, we used again the truth tables, this time for disjunction, to determine the truth value of $A \vee \neg B$, and wrote the result under $\vee$.

4. Finally, we used again the truth table for negation and computed the possible truth values of $\neg(A \vee \neg B)$ on the basis of the truth values for $A \vee \neg B$ we just obtained.

This table corresponds to the graph of the polynomial since it gives us an overall view of the behavior of this propositional formula as A and B vary.

Clearly, we can apply this technique to any formula. If it has just two propositional letters, we proceed pretty much as we just did, simply applying the truth table for the appropriate connective at each step. If it has three, say A, B and C, then we will need $8 = 2^3$ rows to account for all the possible combinations of their truth values. In general, if we have n letters we get 2^n rows.

3.2 Interpretations and Models

That was easy! But what exactly are we doing when calculating the above truth table for $\neg(A \vee \neg B)$? We are imagining all possible ways the world could be with respect to A and B, and computing the truth value of the formula. Each combination of values for A and B is called an *interpretation* (or a *world*), and the corresponding row gives us the truth value of the formula in that world. Here, it is as if we had four planets: on the first both A and B are true and correspondingly $\neg(A \vee \neg B)$ is false; on the second A is true but B is false, and also there $\neg(A \vee \neg B)$ is false; on the third it is the opposite, A is false and B is true, but there $\neg(A \vee \neg B)$ is true; and finally on the fourth planet, where both A and B are false, the formula is false too.

Let's introduce some notation. Let $\mathcal{M}$ be an interpretation (again, that's just an assignment of truth values to propositional letters) and φ a formula. We write

$$\mathcal{M} \models \varphi$$

if φ has value T in the row of the truth table corresponding to $\mathcal{M}$. Then, this world is called a *model* for φ. For example, consider the world $\mathcal{M} = \{A \mapsto F, B \mapsto T\}$, then $\mathcal{M} \models \neg(A \vee \neg B)$ because in the third row of the above table this formula is true. We sometimes write $\mathcal{M} \not\models \varphi$ for a world $\mathcal{M}$ that is not a model for φ, i.e., if φ has value F in the row corresponding to $\mathcal{M}$. Consider, for example, the second row in the above table: it says that $\{A \mapsto T, B \mapsto F\} \not\models \neg(A \vee \neg B)$. We can indeed summarize the truth table for $\neg(A \vee \neg B)$ as follows:

$$\begin{array}{ll} \{A \mapsto T, B \mapsto T\} & \not\models \neg(A \vee \neg B) \\ \{A \mapsto T, B \mapsto F\} & \not\models \neg(A \vee \neg B) \\ \{A \mapsto F, B \mapsto T\} & \models \neg(A \vee \neg B) \\ \{A \mapsto F, B \mapsto F\} & \not\models \neg(A \vee \neg B) \end{array}$$

Notice that each row of the truth table for a formula is independent from the other rows: to check whether a given world is a model for a formula, we do not need to generate the whole table — the row corresponding to that world is enough. This is good when we know exactly what world we are in: even if the formula is very big, this gives us a fast and easy way to calculate the truth value of the formula.

Before going back to valid inferences, let's look at implication again. What do we mean by "A implies B" (or "if A, then B")? Well, we mean that it should never be the case that A is true and B is false. But isn't that what the formula $\neg(A \wedge \neg B)$ expresses?

Indeed, $A \to B$ and $\neg(A \wedge \neg B)$ should be *equivalent*: for every given values of A and B (i.e, for every interpretation), they should have the same truth value. That is, they should have the same truth table. Let's build the truth table of $\neg(A \wedge \neg B)$:

A	B	$\neg$	$(A$	$\wedge$	$\neg$	$B)$
T	T	**T**	T	F	F	T
T	F	**F**	T	T	T	F
F	T	**T**	F	F	F	T
F	F	**T**	F	F	T	F

And indeed! That's exactly the truth table of $A \to B$. This explains why implication has this strange truth table. In truth, there are other explanations of why things work this way.

3.3 Tautologies

One more thing. Some formulas are always true no matter what the truth values of the propositional letters in them. For example here is a very simple one:

A	A	$\vee$	$\neg$	A
T	T	**T**	F	T
F	F	**T**	T	F

These formulas are called *tautologies*. If a formula φ is a tautology, it is true in every world. We write this as

$$\models \varphi$$

In fact, every interpretation is a model of a tautology: if $\models \varphi$ then $\mathcal{M} \models \varphi$ for every $\mathcal{M}$.

A formula that is always false is instead called a *contradiction*. Here's an equally simple example:

A	A	$\wedge$	$\neg$	A
T	T	**F**	F	T
F	F	**F**	T	F

Clearly, a formula φ is a tautology if and only if $\neg\varphi$ is a contradiction, and vice versa. Therefore, φ is a contradiction iff $\models \neg\varphi$. Note that there are formulas that are

neither tautologies nor contradictions, the simplest being the propositional letter A by itself.

To check that a formula is a tautology (or a contradiction), we have to build its entire truth table — all the rows! That can be very time consuming: if the formula contains n propositional letters, its truth table will have 2^n rows. On the other hand, the truth value of each row can be calculated independently from the other rows, so the work of checking that a formula is tautology could be done in parallel: we could recruit 2^n people, give each of them one of the interpretations, ask each to compute the truth value of the formula for his/her interpretation, and finally check whether all values they came up with are T.

3.4 Equivalences

One particularly useful type of tautologies are the ones with $\leftrightarrow$ as their main connective. When $\varphi \leftrightarrow \psi$ is a tautology, then φ and ψ always have the same truth value: they are *interchangeable*. The formulas φ and ψ are *equivalent* in the sense of Section 3.2.

For example, consider the formula $A \vee B \leftrightarrow \neg(\neg A \wedge \neg B)$. It has the following truth table:

A	B	A	$\vee$	B	$\leftrightarrow$	$\neg$	$(\neg$	A	$\wedge$	$\neg$	$B)$
T	T	T	T	T	**T**	T	F	T	F	F	T
T	F	T	T	F	**T**	T	F	T	F	T	F
F	T	F	T	T	**T**	T	T	F	F	F	T
F	F	F	F	F	**T**	F	T	F	T	T	F

(This is one of de Morgan's laws — see below). It says that $A \vee B$ is equivalent to $\neg(\neg A \wedge \neg B)$. In particular, we could take $\neg(\neg A \wedge \neg B)$ as a definition for $A \vee B$. This means that, if we have negation and conjunction, we can simulate disjunction: we do not really need it.

Try it yourself! Show that $(A \leftrightarrow B) \leftrightarrow (A \rightarrow B) \wedge (B \rightarrow A)$ is a tautology. This means that $\leftrightarrow$ itself is definable.

There are a number of interesting equivalences that give logic an algebraic flavor. Consider the following:

- [Unit] $A \wedge \top \leftrightarrow A$
- [Neutral] $A \wedge \bot \leftrightarrow \bot$
- [Commutativity] $A \wedge B \leftrightarrow B \wedge A$
- [Associativity] $A \wedge (B \wedge C) \leftrightarrow (A \wedge B) \wedge C$
- [Distributivity] $A \wedge (B \vee C) \leftrightarrow (A \vee C) \wedge (A \vee C)$

- [Idempotence] $A \wedge A \leftrightarrow A$
- [De Morgan law] $A \wedge B \leftrightarrow \neg(\neg A \vee \neg B)$

The first says that $\wedge$ and $\top$ behave a little bit like multiplication and 1: 1 is the identity element of multiplication, so that $x \times 1 = x$ for any value of x. The second says that $\bot$ is the neutral element of $\wedge$, and it reminds us of $x \times 0 = 0$. The next three are self-explanatory, while the last two have no counterparts in arithmetic.

You may want to verify that these formulas are indeed tautologies by writing their truth table. An even more interesting fact is that if we take the exact same formulas, but switch $\wedge$ and $\vee$ on the one hand, and $\top$ and $\bot$ on the other, we obtain another set of equivalences. This is an instance of a phenomenon called *duality*. Of course, there are many other equivalences. One that is particularly important is that negating a proposition twice is the same as doing nothing:

- [Double negation] $\neg\neg A \leftrightarrow A$

The first mathematician to notice that the propositional connectives had algebraic properties similar to (but not quite the same as) arithmetic was George Boole (1815–1864). Algebras whose operators have these same properties are called Boolean algebras. Other examples include sets and their operations, which was noticed diagrammatically by John Venn (1834–1923), and digital circuits, whose gates were shown to behave like connectives by Claude Shannon (1916–2001)[1].

Notice that we have stated all these equivalences based on propositional letters (A, B and C). They remain equivalences whatever formula we use instead of these letters (as long as every occurrence of A is replaced by the same propositional formula, and so on). For example, the commutativity of $\wedge$ is generalized to the equivalence

$$\varphi \wedge \psi \leftrightarrow \psi \wedge \varphi$$

where φ and ψ stand for any formula we wish. This a general property of tautologies: we can substitute arbitrary formulas for the propositional letters appearing in them and they remain tautologies.

As a final note, observe that, to determine that two formulas are equivalent, one has to build the entire truth table of both. That's again a lot of work, although it can be done in parallel.

3.5 Valid Inferences

Let's go back to valid inferences. An inference is valid if *it cannot be the case that its premises are true but the conclusion is false*. Truth tables give us a mechanism to verify

[1] Shannon made this discovery as part of his master's thesis at the Massachusetts Institute of Technology in 1937. There, he demonstrated that electrical circuits built out of gates that behave like the Boolean connectives could represent and carry out all common logical and numerical computation. It has been claimed to be the most important master's thesis of all times.

that an inference is valid: write the truth table for all the premises and the conclusion, and check that on every row where all premise are true, also the conclusion is true.

Here's an example: we want to verify that Sidra's inference is valid, that is that "A and $A \to B$ so B" is valid. Here is the combined truth table, where we have removed the explicit columns for A and B on the left (they are there anyway under the various occurrences of A and B) and highlighted the rows where all premises are true.

	A	A	$\to$	B	B	
▶	**T**	T	**T**	T	**T**	✓
	T	T	**F**	F	**F**	
	F	F	**T**	T	**T**	
	F	F	**T**	F	**F**	

There is exactly one row where all premises are true, the first row, and on this row, the conclusion is also true. Therefore we can conclude that this inference is valid. All the other rows have some premise that is false, and therefore we don't need to care about what the truth value of the conclusion is: all that matters is what happens when all the premises are true. Observe that the rows of the table still correspond to all possible interpretations with respect to A and B: as we are building the above table, we are examining all possible situations. In particular, the interpretation of each propositional letter remains constant on each row.

Logicians have a special notation for denoting inferences that are valid in the sense we just examined. Rather than writing "If φ_1 and … and φ_n so ψ is a valid inference", they write

$$\varphi_1, \ldots, \varphi_n \models \psi$$

Here, we have that $A, A \to B \models B$.[2]

Now, it is interesting to look at what happens when an inference is not valid. Take for example "$\neg A$ and $A \to B$ so $\neg B$" (isn't it actually valid?) and let's go through the same process:

	$\neg$	A	A	$\to$	B	$\neg$	B	
	F	T	T	**T**	T	**F**	T	
	F	T	T	**F**	F	**T**	F	
▶	**T**	F	F	**T**	T	**F**	T	!!!
▶	**T**	F	F	**T**	F	**T**	F	✓

This time, there are two rows where all premises are true: the third and the fourth are highlighted. In the fourth, the conclusion is true, which satisfies our criterion. Instead, the third row features a conclusion that is false. This row is called a *counterexample* to

[2] This notation originated in another way of reading our definition of validity: for every interpretation $\mathcal{M}$, if $\mathcal{M}$ is a model of every premise (that is if $\mathcal{M} \models \varphi_1$ and … and $\mathcal{M} \models \varphi_n$) then $\mathcal{M}$ must also be a model of the conclusion (that is $\mathcal{M} \models \psi$).

the validity of this inference. It pinpoints a precise interpretation where it fails: for A true and B false, the premises are true but the conclusion is false. By our definition, this means that the inference as a whole is not valid (even if many people think it is). Using the notation we just introduced, $\neg A, A \to B \not\models \neg B$.

Establishing that an inference is valid requires building the truth table of each premise and of the conclusion. As noted earlier, that can be expensive for formulas containing a lot of propositional letters.

It is important at this point to look back and reflect on the road we have traveled. In Chapter 1, when we first encountered Sidra's inference, "Doha is in Qatar and Sidra is in Doha, so she is in Qatar", we intuitively knew it was valid, but we had no way to convince somebody else that it was. Actually, we said it was valid because it was universal: anybody in the world with basic mental capabilities would accept it.

By contrast, we have devised a method to mechanically show that the inference "A and $A \to B$ so B" is valid: simply check that for all four interpretations of A and B, each one that is a model of the premises is also a model of the conclusion. Intuition is gone. In particular, this mechanical approach is independent of the particular meaning we assign to the propositional letters A and B: only the structure of the involved logical formulas (their form) matters. In particular, any English inference that translates to "A and $A \to B$ so B" is automatically valid: that's because of its form, not its meaning. Is "If Paul is hungry he stops making sense and Paul is hungry so he stops making sense" valid? Sure it is: it has the form "A and $A \to B$ so B". Is "If 18 is prime, then it has exactly two divisor and 18 is prime so 18 has exactly two divisors" valid? Absolutely, and for the same reason. By expressing inferences in logic and studying their validity there, we have abstracted from the specific domain they are about (Sidra's whereabouts, Paul's physiology and numbers), from whether specific premises may be true or false (Sidra was in Qatar when I first used that example, 18 is certainly not prime, and we don't know whether Paul is hungry — whoever Paul is), and even from the language in which those sentences where spoken.

3.6 *Exercises*

1. Determine whether the following inferences are valid by using the truth table method:

   - $B \vee A$ and $C \vee \neg A$, so $B \wedge C$
   - $\neg A$ and B, so $B \vee \neg A$
   - $A \wedge \neg A$ and C, so $\neg C$
   - $\neg A \to B$, so $A \vee \neg B$
   - $A \wedge B \to C$ and $A \to B$, so $A \to C$

2. Show that the following formulas are tautologies:

   - $A \to B \to A \wedge B$

- $(A \to B) \vee (B \to A)$
- $A \wedge (B \vee C) \leftrightarrow (A \vee B) \wedge (A \vee C)$
- $\neg(A \leftrightarrow \neg A)$
- $((A \to B) \to A) \to A$

3. Four machines, A, B, C and D, are connected to a computer network. It is feared that a computer virus may have infected the network. Your security team runs its diagnostic tools and informs you of the following state of affairs:

   (a) If D is infected, then so is C.

   (b) If C is infected, then A is infected too.

   (c) If D is clean, then B is clean but C is infected.

   (d) If A is infected, then either B is infected or C is clean.

   You trust your security team and have confidence that all these statements are true. What can you conclude about the individual machines A, B, C and D? Use the truth table method to figure out their possible state of infection.

4. The CEO of a software company makes the following declaration during the shareholders meeting:

   > *If security is a problem, then regulation will increase. If security is not a problem, then business on the Web will grow. Therefore, if regulation is not increased, then business on the Web will grow.*

   At the same meeting, the CFO of the company declares the following:

   > *The sad reality was that if the government did not give us a tax break, we would not meet our 3rd quarter target. Fortunately however, the government just gave us our tax break. So it is my pleasure to announce that we will meet our 3rd quarter target!*

   Write the component statements of each declaration as propositional formulas and express the overall declarations as inference. Is each inference valid? If so, use the truth table method to show it. If not, show a model that violates it.

5. Disjunction as we defined it is *inclusive*: $A \vee B$ is true also when both A and B are true. The connective $\oplus$, pronounced *exclusive or* or *"x" or*, disallows this possibility: $A \oplus B$ is true when either A or B is true, but not both.

   - Write the truth table for $\oplus$.
   - Find one (or more) formulas that are equivalent to $A \oplus B$, i.e., that have the same truth table.

6. In this exercise, we will be interested in what happens when we take a formula φ which contains a propositional letter A and replace A with another formula ψ everywhere in φ. To avoid long-winded sentences, we will write $[\psi/A]\varphi$ for the resulting formula.

(a) An *extension* of an interpretation $\mathcal{M}$ is any interpretation $\mathcal{M}'$ that contains all the truth assignments of $\mathcal{M}$, and possibly a few more — recall that this is just an assignment of truth values to propositional letters. For example, $\{A \mapsto T, B \mapsto F, C \mapsto F\}$ is an extension of $\{A \mapsto T, B \mapsto F\}$. As a warm-up, show that, for any extension $\mathcal{M}'$ of $\mathcal{M}$,

- if $\mathcal{M} \models \varphi$, then $\mathcal{M}' \models \varphi$;
- if $\mathcal{M} \not\models \varphi$, then $\mathcal{M}' \not\models \varphi$.

This means that it doesn't hurt to throw in a few extra propositional letters when considering the interpretations of a formula.

(b) If φ contains A, we can write an interpretation $\mathcal{M}$ of φ as $\mathcal{M}' \cup \{A \mapsto v\}$ where v is the truth value that $\mathcal{M}$ assigns to A and $\mathcal{M}'$ is the truth assignment of the other propositional letters in $\mathcal{M}$. An interpretation of $[\psi/A]\varphi$ has the form $\mathcal{M}' \cup \mathcal{M}''$. Explain why this is the case. What is $\mathcal{M}''$? [Hint: ψ may contain propositional letters that are not in φ, propositional letters that are in φ, and even A itself!] This same characterization also holds if A does not appear in φ. Why?

(c) Show that the following results hold, where $\mathcal{M}^* = \mathcal{M}' \cup \mathcal{M}''$ from the previous question:

- if $v = T$ and $\mathcal{M}^* \models \psi$, then $\mathcal{M}^* \models [\psi/A]\varphi$ if and only if $\mathcal{M} \models \varphi$;
- if $v = F$ and $\mathcal{M}^* \not\models \psi$, then $\mathcal{M}^* \models [\psi/A]\varphi$ if and only if $\mathcal{M} \models \varphi$.

What does this mean in words? What does it say about how the truth table for $[\psi/A]\varphi$ is related to the truth table for φ?

(d) Consider the (valid) inference $A \vee B, \neg B \models A$. How would you use the above result to show that for all φ and ψ the inference $\varphi \vee \psi, \neg\psi \models \varphi$ is valid?

(e) More generally, show that for any formulas $\varphi, \varphi_1, \ldots, \varphi_n$ and ψ,

if $\varphi_1, \ldots, \varphi_n \Gamma \models \varphi$, *then* $[\psi/A]\varphi_1, \ldots, [\psi/A]\varphi_n \models [\psi/A]\varphi$,

(f) Show that for any formulas φ, ψ and ξ, the inference

$\psi \vee \varphi$ <u>and</u> $\xi \vee \neg\varphi$, <u>so</u> $\psi \wedge \xi$

is valid.

7. A *Boolean function* is any n-ary function G that takes n Boolean variables $A_1, \ldots, A_n$ as input (the possible value of each A_i is either T or F) and returns T or F. One way to describe a Boolean function is to give a truth table for it: it shows the value of G for each possible way of assigning T or F to its inputs. Consider the following example:

A	B	G
T	T	F
T	F	T
F	T	T
F	F	F

Now, we can express any Boolean function by means of a Boolean formula that uses just $\wedge$, $\vee$ and $\neg$.

To start with, for each row j, build the conjunction R_j as follows:

- if input A_i has value T, add the conjunct A_i to R_j
- if input A_i has value F, add the conjunct $\neg A_i$ to R_j

Then, combine the formulas R_j you just obtained for each row j into the *disjunctive* Boolean formula Pos built as follows:

- if the output of G has value T on row j, add the disjunct R_j to Pos
- (if no output of G has value T, Pos is just $\bot$)

For instance, in the example above, we get

$$Pos_G = (A \wedge \neg B) \vee (\neg A \wedge B)$$

Another way to express the Boolean function G is to form the *conjunctive* Boolean formula Neg from the row formulas R_j as follows:

- if the output of G has value F on row j, add the conjunct $\neg R_j$ to Neg
- (if no output of G has value F, Neg is just $\top$)

For instance, in the above example, we get

$$Neg_G = \neg(A \wedge B) \wedge \neg(\neg A \wedge \neg B)$$

(a) Check that this works by verifying that the formula Pos_G we just obtained has the same truth table as G above. Then, explain in your own words why the construction for Pos works.

(b) Check that this works by verifying that the formula Neg_G too has the same truth table as G above. Then, explain in your own words why the construction for Neg works.

(c) Check that also $Pos_G \wedge Neg_G$ has the same truth table as G and explain why. Notice that this allows to omit the special cases where Pos is $\bot$ and where Neg is $\top$.

(d) Notice that the construction of Neg only uses $\neg$ and $\wedge$ (and possibly $\top$). By relying on the de Morgan equivalences, modify the construction for Pos so that it only uses $\neg$ and $\vee$.

(e) The above discussion implies that every Boolean function can be expressed by a Boolean formula that only uses $\neg$ and $\wedge$, or $\neg$ and $\vee$. Show that is possible to do so also with only $\neg$ and $\rightarrow$.

8. We saw in Section 3.4 that the de Morgan law, $A \wedge B \leftrightarrow \neg(\neg A \vee \neg B)$, is a tautological equivalence that allows us to define disjunction in terms of negation and conjunction. This means that any formula that contains disjunctions can be rewritten into an equivalent formula without it. Said differently, disjunction

is not necessary. In this exercise, we will discover what other connectives are definable in this way. We will also look for the smallest set of connectives that can define all others.

(a) Consider the standard set of connectives: $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$. Find equivalences that define

- $\leftrightarrow$ in terms of $\{\neg, \wedge, \vee, \rightarrow\}$,
- $\rightarrow$ in terms of $\{\neg, \wedge, \vee\}$,
- $\wedge$ in terms of $\{\neg, \vee\}$.

For each of these equivalences, show that they are tautologies by displaying their truth tables.

(b) By the previous exercise, $\{\neg, \vee\}$ seems to be a good candidate as a "smallest set of connectives" for $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$. Make use of this property to rewrite the following formulas into equivalent formulas that contain just $\neg$ and $\vee$. Do so in steps and justify each step by writing which equivalence you are using.

- $A \vee B \leftrightarrow \neg(\neg A \wedge \neg B)$
- $A \rightarrow B \rightarrow A \wedge B$
- $(A \rightarrow C) \wedge (B \rightarrow C) \rightarrow (A \vee B \rightarrow C)$
- $(A \rightarrow (B \rightarrow C)) \rightarrow (A \rightarrow B) \rightarrow (A \rightarrow C)$
- $((A \rightarrow B) \leftrightarrow (A \rightarrow \neg B)) \rightarrow \neg A$

We used the standard precedence conventions to limit parentheses proliferation.

(c) Given a set of connectives C, a subset C' of C is *minimal* if every connective in C can be defined in terms of just the connectives in C', but this is not true anymore if we remove any element of C'.

We have just seen that any formula that uses $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$ is equivalent to a formula over just $\{\neg, \vee\}$. Now, show that $\{\neg, \vee\}$ is minimal, i.e., that there are formulas that we cannot express using $\neg$ alone or $\vee$ alone.

(d) The set $\{\neg, \vee\}$ is not the only minimal set of connectives with respect to $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$. Give at least two other 2-element sets of connectives that are minimal with respect to $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$.

(e) Let us define a new connective. $A \uparrow B = \neg(A \wedge B)$. This new connective is called *nand* and therefore $A \uparrow B$ is read *"A nand B"*.

- Write the truth table of $A \uparrow B$.
- Show that nand is sufficient to define all connectives in $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$. Therefore, $\{\uparrow\}$ is a minimal set of connectives with respect to $\{\uparrow, \neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$.
- Define another connective with the same property.

9. When using truth tables to describe propositional logic, atomic symbols and complex propositions can have one of two values: T or F. This makes this

system a *two-valued logic*. This is a convenient way to describe *objective* reality (after all, either the sun is shining in Mexico City in this moment, or it is not — there is no alternative), but *subjectively* things are different: unless I look on the web, I have no idea whether the sun is or is not shining in Mexico City I simply don't know, do you? Let us try to adapt propositional logic to deal with subjective realities of this type. All we will do will be to modify the way we build truth tables with a third value: U for "unknown". We are therefore dealing with a *three-valued logic*. Here are the updated truth tables for negation, conjunction and disjunction:

A	B	$\neg A$	$A \wedge B$	$A \vee B$
T	T	F	T	T
T	U	F	U	T
T	F	F	F	T
U	T	U	U	T
U	U	U	U	U
U	F	U	F	U
F	T	T	F	T
F	U	T	F	U
F	F	T	F	F

(a) If all we care about are interpretations where the propositional symbols can only have value T or F (U is not allowed), can we use this 3-valued truth table instead of our original 2-valued truth table? Explain why. If the answer is "yes", the 3-valued logic is said to be *conservative* over the 2-valued logic.

(b) Viewing U as "unknown", explain why it is reasonable to define $\neg U = U$ and $\top \wedge U = U$ and also $\bot \vee U = U$.

(c) What is the 3-valued truth table of implication if we still want $A \to B$ to be equivalent to $\neg(A \wedge \neg B)$? By the same token, what does the 3-valued truth table for $A \leftrightarrow B$ look like?

(d) Suppose the statement "Flight 237 is on time" is true, the statement "Runway conditions are icy" is false, and the truth value of the statement "Flight 51 is on-time" is unknown. Find the truth value of the following statements:

i. Runway conditions are not icy and flight 51 is on time.

ii. Flight 51 is on time and flight 237 is not.

iii. Flight 51 is not on time or runway conditions are not icy.

iv. If the runway conditions are icy then flight 51 is not on time

v. Flight 237 is on time if and only if the runway conditions are not icy and flight 51 is not on time.

10. Using the truth table method, show that the following inferences are valid.

(a) $A \wedge B, A \to (B \to C) \models A \wedge C$

(b) $A \to (A \to B) \models A \to B$

(c) $\neg A \vee B, B \to C \models A \to C$

(d) $\models (A \vee A \to A) \wedge (A \to A \vee A)$

(e) $A \to B, \neg(B \vee A) \models B \to C$

11. In Python, Java and other programming languages, *Boolean expressions* are used in the condition of `if` statements and to control loops. They are kind of like propositional formulas in the sense that they evaluate to true or false, but in reality they have a richer set of possible behaviors: they can raise exceptions and their evaluation may not terminate. In this exercise, we will look at this latter property, called *divergence* (but we ignore exceptions).

    We will study a form of propositional logic where atomic formulas can take the values T (true), F (false) and $\nearrow$ (loop forever) — you can think of this as another type of 3-valued logic. Composite formulas are built using standard connectives (typically $\wedge$, $\vee$ and $\neg$), and their evaluation proceeds left-to-right. This gives rise to the following truth table:

A	B	$\neg A$	$A \wedge B$	$A \vee B$
T	T	F	T	T
T	F	F	F	T
T	$\nearrow$	F	$\nearrow$	T
F	T	T	F	T
F	F	T	F	F
F	$\nearrow$	T	F	$\nearrow$
$\nearrow$	T	$\nearrow$	$\nearrow$	$\nearrow$
$\nearrow$	F	$\nearrow$	$\nearrow$	$\nearrow$
$\nearrow$	$\nearrow$	$\nearrow$	$\nearrow$	$\nearrow$

    (a) Using your favorite programming language, write a Boolean condition containing a variable `x` such that this condition evaluates to true when `x>0`, to false when `x=0`, and loops forever for `x<0`. Feel free to define any auxiliary code you may need.

    (b) Some of the standard propositional equivalences do not hold in this logic. Give two examples, one with $\wedge$ and the other with $\vee$ and explain why they are not valid.

    (c) Extend the above truth table with columns for the Boolean expressions $A \to B$ and $A \leftrightarrow B$. Justify your answer on the basis of well-known equivalences.

    (d) Some programming languages include two additional Boolean connectives: *parallel-and* (written $A \sqcap B$) and *parallel-or* (written $A \sqcup B$). These operators behave just like $\wedge$ and $\vee$ but they evaluate A and B at the same time

(in parallel). If one of them terminates and that's enough to determine the overall value of the expression, it can kill the execution of the other. Write the truth tables for $A \sqcap B$ and $A \sqcup B$. Have you seen these truth tables before?

12. In natural language, the statement *"if it rains, then the road is wet"* is closely related to the inference *"it rains so the road is wet"*. This is not a coincidence (and it helps explain why implication is such a weird connective). Validity and implication are indeed related by a result the property that $\Gamma \models \varphi \to \psi$ iff $\Gamma, \varphi \models \psi$ for any set of formulas $\Gamma = \varphi_1 \dots \varphi_n$. In this exercise, we will take inspiration to this result to look at the way validity works.

    (a) Using the truth table method, show that if $\Gamma \models \varphi \to \psi$, then $\Gamma, \varphi \models \psi$.

    (b) In the same way, show that, conversely, if $\Gamma, \varphi \models \psi$, then $\Gamma \models \varphi \to \psi$.

    (c) Use similar arguments to show that $\Gamma, \varphi', \varphi'' \models \psi$ iff $\Gamma, \varphi' \wedge \varphi'' \models \psi$.

    (d) Based on these results, any valid inference $\Gamma \models \psi$ can be rewritten as $\models (\wedge\ \Gamma) \to \psi$, where $(\wedge\ \Gamma)$ is the formula obtained by taking the conjunction of every formula in Γ. This says that the inference $\Gamma \models \psi$ is valid if and only if the formula $(\wedge\ \Gamma) \to \psi$ is a tautology.

13. Write your favorite inference in symbolic notation and show that it is valid. The only constraint on this "favorite" is that it not be one of the inferences seen so far in this book.

Chapter 4

Derivations

Truth tables are great, but they don't scale up:

- Their size is exponential in the number of atomic propositions, so they get big very quickly. If we want to make inferences about addition in a 64-bit processor and represent each bit with a propositional letter, the 128 bit of input suddenly yield a table that has 2^{128} rows. This is enormous! Even if you are very fast, it would take you more than the age of the universe to write it down!
- We cannot quite use truth tables for more complex logics, like *predicate logic* (see Chapter 5).
- Basing inferences on truth and falsehood is simplistic. There are other options that cannot be expressed using truth tables.

Enter . . . *derivations*.

4.1 Elementary Derivations

Because all is needed to understand inferences is the way the individual connectives in them work, the idea is to list all the elementary inferences that involve each connective in isolation — there is a small number of them. Then, we get all the other, more complicated, inferences by composing these elementary inferences. This is what *theorem proving* is based upon.

4.1.1 Conjunction

Let's take conjunction. If we know that $A \wedge B$ is true, what inferences can we make? Well, if $A \wedge B$ is true, for sure A is true. This is the type of elementary inference we are talking about. Do we know anything else? Of course! If $A \wedge B$ is true, then B must be true also. Anything else? Not really: every other inference with $A \wedge B$ as a

premise has a conclusion that is at least as complicated as $A \wedge B$ itself. Good, let's list these inferences:

- $\wedge_{\mathsf{E1}}$: $A \wedge B$, so A.
- $\wedge_{\mathsf{E2}}$: $A \wedge B$, so B.

($\wedge_{\mathsf{E1}}$ and $\wedge_{\mathsf{E2}}$ are the names we will use to refer to them). Observe the way these inferences work: their premise contains the connective we are analyzing (here conjunction) and their conclusion tells us something simpler that is true if the premises are true. Inferences of this form are called *elimination rules* (that's why their name has "E" in it).

Now, there is another type of elementary inference about conjunction we can be interested in: when is it that we can conclude that $A \wedge B$ is true? Well, for $A \wedge B$ to be true, it must be the case that both A is true and B is true. This gives rise to another inference:

- $\wedge_{\mathsf{I}}$: A and B, so $A \wedge B$.

Elementary inferences of this type, where the connective we are examining appears in the conclusion, are called *introduction rules* (that explains the "I" in the name).

Logicians like to write rules such as the above slightly differently: rather than using the word "so", they write a horizontal line with the premises above the line and the conclusion below it. Here is how a logician would write the above three rules:

$$\frac{A \wedge B}{A}\wedge_{\mathsf{E1}} \qquad \frac{A \wedge B}{B}\wedge_{\mathsf{E2}} \qquad \frac{A \quad B}{A \wedge B}\wedge_{\mathsf{I}}$$

Each of these is an *inference rule*, or simply *rule*, which is what logicians call elementary inferences.

How do we use these rules? Let us show that a more complex inference is valid: this will be "$A \wedge B$, so $B \wedge A$" — conjunction is commutative (that may be obvious, but there is nothing in the rules that hints at this!). We will start with the premise, and then use the rules to infer a series of intermediate formulas and eventually produce the conclusion. We need to be careful to always say which rule we are using on which formula.

1.	$A \wedge B$	assumption
2.	A	by $\wedge_{\mathsf{E1}}$ on (1)
3.	B	by $\wedge_{\mathsf{E2}}$ on (1)
4.	$B \wedge A$	by $\wedge_{\mathsf{I}}$ on (3) and (2)

Success! This series of formulas together with the justification for every step is called a *derivation*. We have derived that $A \wedge B$ so $B \wedge A$ is a valid inference, or, said equivalently, that $B \wedge A$ is derivable from $A \wedge B$. Logicians have a special notation for this too: they write the symbol "$\vdash$" with the premises on the left and the conclusion on the right, so we have obtained that

$$A \wedge B \vdash B \wedge A$$

Sometimes, logicians find it convenient to write derivations as trees by stacking the rules. The above derivation would then look like this:

$$\dfrac{\dfrac{A \wedge B}{B}\wedge_{\mathsf{E2}} \quad \dfrac{A \wedge B}{A}\wedge_{\mathsf{E1}}}{B \wedge A}\wedge_{\mathsf{I}}$$

The conclusion of this derivation, $B \wedge A$, is at the bottom and the assumptions (here two occurrences of $A \wedge B$) are at the top without any horizontal line above them.

When writing your own derivations, you can choose either the sequential step format or the tree format: whatever works for you.

4.1.2 Truth

While we learned a lot of things, conjunction alone gets boring after a while. Let's look for other rules.

The next logical entity we will look at is the truth constant, $\top$. When is it true? Well, always, by definition! This immediately yields the following introduction rule:

$$\dfrac{}{\top}\top_{\mathsf{I}}$$

It says that $\top$ is always true. What would the elimination rule(s) be? What can we deduce knowing that something is always true? Nothing interesting, really: knowing that $\top$ is true, we get at most that $\top$ is true, which is nothing new. Indeed, there is no elimination rule for $\top$.

Let us remind ourselves how rules work (phrased a bit differently than before): the rules for a connective describe all the ways we can unpack the information held in this connective (the elimination rules) and all the ways we can put together a formula using that connective (the introduction rules).

4.1.3 Implication

The elimination rule for implication is easy — it is Sidra's rule: "A and $A \to B$ so B" — and it takes the following form:

$$\dfrac{A \quad A \to B}{B}\to_{\mathsf{E}}$$

(while this rule is very dear to Sidra, it is known to logicians and philosophers worldwide as the *modus ponens*).

What about inferences that introduce implication? Let's think about the way we prove things of the form "*if* ... *then* ..." in our math classes. Take the following (silly) example:

If x is odd, then $x + x$ is even

The way you would probably proceed is as follows:

Let's assume that x is odd, then $x = 2y + 1$. Under this assumption, $x + x = (2y + 1) + (2y + 1)$, but this is equal to $4y + 2 = 2(2y + 1)$ which is an even number.

Here, we assumed that the antecedent ("x is odd") was true, as if it were a temporary assumption, and we used this assumption to show that the consequent ("$x + x$ is even") is true. This suggests the following (strange) rule for introducing an implication:

$$\dfrac{\begin{array}{c}\overline{A}\\ \vdots\\ B\end{array}}{A \to B}\to_{\mathsf{I}}$$

This rules adds A as an assumption, but we can only use it while deriving B and once we use $\to_{\mathsf{I}}$, we strike it out: it is not an assumption anymore. Let's see how this works on an actual example. Let's show that $A \to B, B \to C \vdash A \to C$ (remember? this stands for the inference "$A \to B$ and $B \to C$ so $A \to C$": i.e., implication is transitive).

$$\dfrac{\dfrac{\dfrac{\overline{A} \quad A \to B}{B}\to_{\mathsf{E}} \qquad B \to C}{C}\to_{\mathsf{E}}}{A \to C}\to_{\mathsf{I}}$$

Notice that the assumptions of this derivation (the formulas without a horizontal bar above them) are just $A \to B$ and $B \to C$. Indeed A was a temporary assumption until we used rule $\to_{\mathsf{I}}$. Let's try to write the same derivation in the sequential steps style.

1.	A	temporary assumption of (6)
2.	$A \to B$	assumption
3.	B	by $\to_{\mathsf{E}}$ on (1) and (2)
4.	$B \to C$	assumption
5.	C	by $\to_{\mathsf{E}}$ on (3) and (4)
6.	$A \to C$	by $\to_{\mathsf{I}}$ on (5) assuming (1)

Whether using truth tables or derivations, implication is always weird!

4.1.4 Disjunction

The introduction rules for disjunction are easy: if we know that A is true, then $A \vee B$ is true whatever B is. This yields the following pair of symmetric rules:

$$\frac{A}{A \vee B}\vee_{\mathsf{I1}} \qquad\qquad \frac{B}{A \vee B}\vee_{\mathsf{I2}}$$

The elimination rule will again be strange. What can we do if we know $A \vee B$? Again, we will take inspiration from the way we do simple proofs. Consider another silly example:

> *Whether x is even or odd, $x + x$ is even.*

To prove this fact, you would consider two cases, one where x is even and the other where x is odd, and for each of them prove that the conclusion is true. Let's do it.

> *If x is even, then $x = 2y$. In this case, $x + x = 2y + 2y$, which is equal to $4y = 2(2y)$ and so $x + x$ is even.*
>
> *If instead x is odd, then $x = 2y+1$. In this case, $x+x = (2y+1)+(2y+1)$, but this is equal to $4y + 2 = 2(2y + 1)$ and so $x + x$ is even.*

In this proof, we reasoned by *cases*, assuming each disjunct in turn and showing that under that (temporary) assumption we could prove the conclusion. We capture this way of reasoning in the following inference:

$$\frac{\begin{array}{ccc} & \overline{A} & \overline{B} \\ & \vdots & \vdots \\ A \vee B & C & C \end{array}}{C}\vee_{\mathsf{E}}$$

Let's read it out loud: knowing that $A \vee B$ is true ("x is either even or odd"), and if we are able to show C (that "$x + x$ is even") both in the case A were true ("x is even") and in the case B were true ("x is odd"), then C must be true.

For practice, let's give a derivation for $A \vee B \vdash B \vee A$:

$$\frac{A \vee B \qquad \dfrac{\overline{A}}{B \vee A}\vee_{\mathsf{I2}} \qquad \dfrac{\overline{B}}{B \vee A}\vee_{\mathsf{I1}}}{B \vee A}\vee_{\mathsf{E}}$$

or linearly:

1. $A \vee B$ assumption
2. A temporary assumption of (6)
3. $B \vee A$ by $\vee_{\mathsf{I2}}$ on (2)
4. B temporary assumption of (6)
5. $B \vee A$ by $\vee_{\mathsf{I1}}$ on (4)
6. $B \vee A$ by $\vee_{\mathsf{E}}$ on (1), (3) assuming (2), and (5) assuming (4)

4.1.5 Falsehood

While we are on the subject of weird rules, let's look at falsehood: $\bot$ is never true. An immediate consequence of this is that there cannot be any introduction rule for $\bot$: it simply cannot be true!

There is an elimination rule instead, and a strange one: if we were able somehow to derive $\bot$, then we could derive anything we want:

$$\frac{\bot}{A}\,\bot_{\mathsf{E}}$$

To make sense of this, think of a situation where you have been able to derive $\bot$ as some kind of absurd state, where everything is possible. Have you seen the movie *Alice*? Once Alice gets to Wonderland (an absurd place), then everything becomes possible (cats fly, mice talk, playing cards defend castles, etc) [2].

Interestingly, this absurd state has an expression in many languages. For example,

- French has "la semaine des quatre Jeudis" (the week with 4 Thursdays) and "quand les poules auront des dents" (when chickens will grow teeth)
- Friulan (a Latin language) has "il trentedoi di Mai" (May 32nd)

When you are a kid (and sometimes an adult), these are the dates where all your wishes will be realized ...

If that's still strange to you (and it should be), check the way truth tables work when the premises are always false: whatever the conclusion is, the inference is valid.

4.1.6 Negation

Let's end with a bang! Negation is the weirdest of all connectives and the one that has caused most head scratching among logicians and philosophers. I will make it simple for you and take $\neg A$ to stand as an abbreviation for $A \to \bot$, that is $\neg A$ is true if A implies the absurd state, i.e., if A cannot be.

Under this definition, it is easy to obtain the rules for $\neg$ by simply using the rules for $\to$ and $\bot$ (check it as an exercise!):

$$\frac{\begin{array}{c}\overline{A}\\ \vdots\\ \bot\end{array}}{\neg A}\,\neg_{\mathsf{I}} \qquad\qquad \frac{A \quad \neg A}{B}\,\neg_{\mathsf{E}}$$

The introduction rule on the left corresponds to the proof technique based on *contradiction*: to show that $\neg A$ is true, assume temporarily that A is true instead and if you get a contradiction ($\bot$), then it must be the case that $\neg A$ is true. The elimination rule on the right says that if we are ever able to derive something and its negation, then we are in an absurd state where anything can be true.

Let us practice by showing the validity of the inference $A \vee B, \neg B \vdash A$:

$$\dfrac{A \vee B \quad \overline{A} \qquad \dfrac{\overline{B} \qquad \neg B}{A}\,\neg_{\mathsf{E}}}{A}\,\vee_{\mathsf{E}}$$

Having just shown that $A \vee B, \neg B \vdash A$, we know that "$A \vee B$, $\neg B$ so A" is a valid inference, which means that we should be able to use it in other inferences: after all, it is just an abbreviation of the above derivation. We can pretend that it is a more complex type of rule (these are called *derived rules*) and we can write it as

$$\dfrac{A \vee B \quad \neg B}{A}\,\mathsf{AE}$$

We have called it AE (alternative excluded) so that we can use it later.

Let's go back to rule $\neg_{\mathsf{E}}$. Can it be right that from a premise and its negation we can derive whatever we want? Knowing that "the sun shines" and "the sun doesn't shine" deducing that "camels can fly" looks like a stretch. Let's derive that "the sun shines", ¬"the sun shines" ⊢ "camels can fly" in a different way:

$$\dfrac{\dfrac{\text{“the sun shines”}}{\text{“the sun shines” } \vee \text{ “camels can fly”}}\,\vee_{\mathsf{I1}} \qquad \neg\text{“the sun shines”}}{\text{“camels can fly”}}\,\mathsf{AE}$$

This is not the end of the story with negation: I told you it was weird. We need one more rule, a rule that kind of breaks the pattern we have seen so far, where there is a single mention of a logical connective or constant in each rule:

$$\dfrac{\neg\neg A}{A}\,\neg^{2}$$

This rule is called *double negation elimination*. It says that if we know that it is not true that A is not true, then A must be true, as in "it is not true that $1 + 1 \neq 2$" which means that $1 + 1 = 2$. We will see what role this rule plays in a minute.

4.2 Soundness and Completeness

We have given two definitions of what a valid inference "φ_1 and … and φ_n so φ" is:

- using truth tables, we have defined it as $\varphi_1, \ldots, \varphi_n \models \varphi$, and
- using derivations, we have defined it as $\varphi_1, \ldots, \varphi_n \vdash \varphi$.

How do we know both definitions get us the same notion of valid inference, i.e., that they are equivalent?

Showing equivalences like these is big business in logic: if you prove it, you get not one, but two theorems with fancy names.

For succinctness, let's abbreviate $\varphi_1, \ldots, \varphi_n$ as Γ (the Greek letter Gamma). Showing that if "Γ so φ" is valid according to the derivation method, then it is also valid with the truth table method is called *soundness*, and we have the following *soundness theorem*:

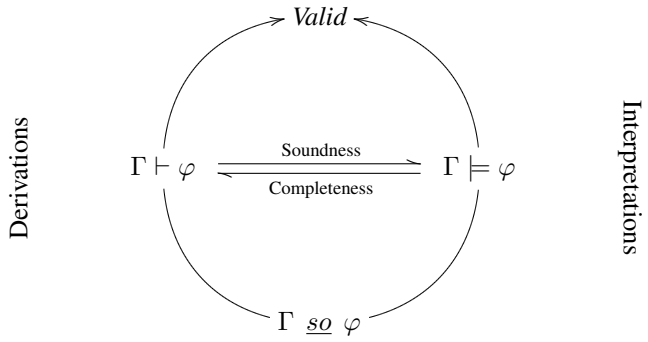
Theorem 4.1 (Soundness) *If* $\Gamma \vdash \varphi$*, then* $\Gamma \models \varphi$.

Showing the converse, that if "Γ so φ" is valid according to the derivation method, then it is also valid using the truth table method is called *completeness* and yields the following *completeness theorem*:

Theorem 4.2 (Completeness) *If* $\Gamma \models \varphi$*, then* $\Gamma \vdash \varphi$.

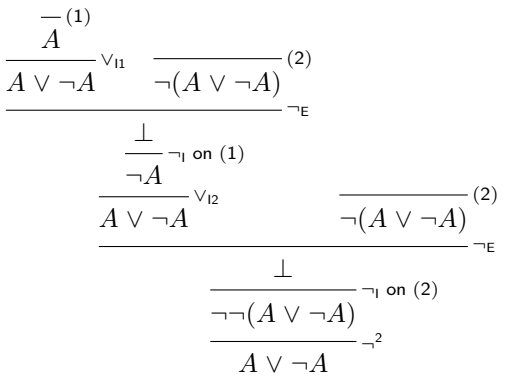
Together, these two theorems tell us that the two methods for characterizing valid inferences are equivalent. Good to know!!!

Here is a cool diagram that illustrates the soundness and completeness theorems:



One thing logicians like doing is trying to get rid of rules and see what happens. Here, if we remove any rule, then the completeness theorem stops holding. Yet, some logicians have been really queasy about the double negation rule: it causes a bunch of strange inferences to be valid. These logicians, who called themselves *intuitionists*, decided to get rid of it nonetheless, and that led to one of the most powerful forms of logic used in computer science: *intuitionistic logic* (also called *constructive logic*).

Without double negation elimination (rule $\neg^2$), the formula $\neg\neg A \to A$ is not derivable, that is $\nvdash \neg\neg A \to A$, while it is very easy to write a derivation for it if rule $\neg^2$ is available. What other examples are there of valid formulas that are not valid intuitionistically? An important one is the *law of excluded middle*, $\vdash A \vee \neg A$, which states that either a formula or its negation must always be true (to which the intuitionists retort that this is useless unless we have a way to know which one). It is interesting to write a derivation for $A \vee \neg A$ using our rule set. As you will see, derivations that involve $\neg^2$ often take a bizarre shape, which is sometimes seen as a kind of contorted reasoning.



Confused? That may give you a sense of how come the intuitionists did not think highly of double negation elimination and similar rules.[1]

4.3 *Exercises*

1. Using the derivation method, show that the following inferences are valid.

   (a) $A \wedge B, A \to (B \to C) \vdash A \wedge C$

   (b) $A \to (A \to B) \vdash A \to B$

   (c) $\neg A \vee B, B \to C \vdash A \to C$

   (d) $\vdash (A \vee A \to A) \wedge (A \to A \vee A)$

   (e) $A \to B, \neg(B \vee A) \vdash B \to C$

2. At a trial, a defense attorney gave the following argument:

   > *If my client is guilty, then the knife was in the drawer. Either the knife was not in the drawer or Jason Pritchard saw the knife. But if Jason Pritchard saw the knife, then it follows that the cabinet was unlocked on October 10. Furthermore, if the cabinet was unlocked on October 10, then the knife was in the drawer, the chimney was blocked and also the hammer was in the barn. But we all know that the hammer was not in the barn. Therefore, ladies and gentlemen of the jury, my client is innocent.*

   Typical lawyer talk! But if you are in the jury, should you be convinced by this argument? Let's bring propositional logic to the rescue. Identify the atomic propositions in this text, express the individual parts as formulas and the overall argument as an inference. Finally, write a derivation that shows that this inference is valid. Also, say why it is much more convenient to use derivations than the truth table method in this case.

[1] The reasons are actually deeper than this and go beyond the scope of this book.

3. Having defined $\neg A$ as $A \to \bot$ in Section 4.1.6, show that the rules $\neg_{\mathsf{I}}$ and $\neg_{\mathsf{E}}$ we have given can be derived from the rules for $\to$ and $\bot$.

4. Did we forget about if-and-only-if??? We did indeed! Just like we understood $\neg A$ as an abbreviation of $A \to \bot$, logicians who work on derivations (they are called *proof theorists*) view $A \leftrightarrow B$ as an abbreviation for $(A \to B) \wedge (B \to A)$. Taking inspiration to what we did for negation, write the elementary derivations for $A \leftrightarrow B$.

5. In natural language, the statement *"if it rains, then the road is wet"* is closely related to the inference *"it rains so the road is wet"*. This is not a coincidence (and it helps explain why implication is such a weird connective). Derivability and implication are indeed related by the *deduction theorem*, which states that $\Gamma \vdash \varphi \to \psi$ iff $\Gamma, \varphi \vdash \psi$ for any set of formulas $\Gamma = \varphi_1 \dots \varphi_n$. In this exercise, we will take inspiration to this result to look at the way derivability works. You may not use the deduction theorem to solve the following questions.

   (a) Using the inference rules you know, show that if you have a derivation of $\Gamma \vdash \varphi \to \psi$, then you can extend it into a derivation of $\Gamma, \varphi \vdash \psi$.

   (b) Similarly, use the inference rules you know to show that if $\Gamma, \varphi \vdash \psi$, then $\Gamma \vdash \varphi \to \psi$.

   (c) Use similar arguments to show that $\Gamma, \varphi', \varphi'' \vdash \psi$ iff $\Gamma, \varphi' \wedge \varphi'' \vdash \psi$. Be careful because now you are working on the premises of the inference.

   (d) Based on these results, any valid inference $\Gamma \vdash \psi$ can be rewritten as $\vdash (\wedge\ \Gamma) \to \psi$, where $(\wedge\ \Gamma)$ is the formula obtained by taking the conjunction of every formula in Γ. Show how this applies to the following inference: *"if it rains then the road is wet"*, *"it rains"* $\vdash$ *"the road is wet"*. Specifically, express this inference in logical notation, rewrite it as just described, and translate it back into English.

   The results obtained in this exercise can also be justified using the truth table method. This was the subject of exercise 3.12 in Chapter 3.

6. A first step toward proving the soundness theorem in Section 4.2 is to show that each derivation rules is a valid inference using the truth table method. Let's explore this a bit.

   - Show that $\wedge_{\mathsf{E1}}$ is sound, i.e., show that the inference $A \wedge B \models A$ is valid using the truth table method.
   - Similarly, show that $\to_{\mathsf{E}}$ and $\neg_{\mathsf{E}}$ are sound.
   - The rules with temporary assumption are a bit more tricky: for them, you first write the truth table as if the temporary assumptions were not there. Then, for every premise that has a temporary assumption, you need to strike out the rows where the temporary assumption is false and the premise is true. Finally, use the method we saw in Chapter 3 on what is left of the truth table. Your turn: show that $\to_{\mathsf{I}}$ and $\vee_{\mathsf{E}}$ are sound. Just to be sure, do the same for rule $\neg_{\mathsf{I}}$.

- At this point, you have shown that most of the derivation rules are sound. Do you want to do the rest?
- How would you finish off the proof of the soundness theorem? That is, knowing that $\Gamma \vdash \varphi$, show that it must be the case that $\Gamma \models \varphi$?

7. One property that is particularly useful when studying derivations is the *substitution lemma*. It says that, for any formulas φ and ψ and any set of formulas Γ, if $\Gamma \vdash \psi$ and $\Gamma, \psi \vdash \varphi$, then $\Gamma \vdash \varphi$. In words, if we can derive φ assuming ψ, but we have a way to derive ψ on its own, they we can build a derivation of φ directly, without ψ as an assumption. What does this direct derivation of φ look like? [Hint: Drawing pictures is a great way to answer this question.] Do you want to prove it? One of the previous exercises gives you the tools to do precisely that!

8. Checking that an inference is valid is easy with the truth table method. With the derivation method, it does not seem as obvious. We will now examine some of the vexations involved and mention solutions for them.

   - One of the most vexing things about derivations is deciding what rule to apply when. Here is a simple heuristic that always works: apply elimination rules to assumptions (and formulas obtained from them by applying elimination rules); apply introduction rules to conclusions (and formulas obtained from them by applying introduction rules); apply introduction rules also to the leftmost premise of rules $\to_{\mathsf{E}}$ and $\neg_{\mathsf{E}}$ and to the two rightmost premises of rule $\vee_{\mathsf{E}}$. By doing so, elimination and introduction rules will always meet at propositional letters. Derivations that follow this heuristic are called *canonical*.
   - Say we have a hunch that to build a derivation for an inference $\Gamma \vdash \varphi$ formula we need to apply rule $\to_{\mathsf{E}}$. To use this rule, however, we need to come up with a formula ψ to put in its premises. A priori, ψ could be any formula. That's infinitely many formulas to try! How do we narrow down the hunt? One really useful result in this case is the *subformula property*. The subformula property tells us that the only candidates for ψ we ever need to consider as subformulas of the formulas in Γ and φ. The same applies to all other rules that apparently ask us to pull a formula out of our sleeve.
   - One last annoyance about derivations is that it is never clear when to stop applying rules and declare that an inference is not derivable. One simple method is to give up on an attempt if we encounter a formula twice on the same branch of a derivation *as long as no new temporary assumptions were made in between*.

   Your turn now. Using these tricks, carry out the following tasks:

   - Verify that all the example inferences in this chapter were canonical and abide by the subformula property.

- Give a derivation of $A \to C \vdash (B \to C) \to (A \vee B \to C)$.
- Show that the inference $A \to A \vdash A$ is not valid.
- Show that the inference $A, \neg B \vdash C$ is not valid.
- Determine whether the inference $\neg A \to \bot \vdash A$ is valid.
- Explain how one can exploit these heuristics to *always* determine whether an inference is valid using the derivation method, and in finite time (although it could take quite a long time).

9. One important property of propositional logic is that it is *consistent*, i.e., that there is no formula φ such that both $\vdash \varphi$ and $\vdash \neg\varphi$ are derivable.

   - Why would it be bad if such a formula existed?
   - The above definition of consistency asks for derivations of φ and $\neg\varphi$ that don't have any premises. Give an example of a set of assumptions Γ and a formula φ such that $\Gamma \vdash \varphi$ and $\Gamma \vdash \neg\varphi$ are both derivable.
   - Using the soundness and/or completeness theorems, give an argument for why propositional logic is consistent.

10. Write a favorite inference in symbolic notation and show that it is valid using the derivation method. The only constraint on this "favorite" is that it not be one of the inferences seen so far in this book.

Part II

Predicate Logic

Chapter 5

Predicate Logic

Looking back, propositional logic is pretty straightforward once we get a grip on how implication works, and showing that inferences are valid is a breeze using truth tables (derivations may seem a bit mysterious at first, but all it takes is some practice). Judging by the exercises we gave, it is also quite useful. Yes, but it doesn't take long before we feel the need to capture more linguistic patterns within logic in order to express more of the world around us.

Predicate logic builds on propositional logic: it retains all the ideas and the methods that we saw in the last three chapters, and extends them. It brings out the structure of atomic formulas and introduces quantifiers to leverage this structure. This yields an enormous jump in what can be expressed in logic and in the type of reasoning we can do. However, it also significantly complicates showing that an inference is valid. In this and the next two chapters, we will extend the material in each of the previous three chapters with what it takes to accommodate these concepts.

5.1 Beyond Propositional Inferences

So far, we have looked at inferences in propositional logic: elementary sentences that are true or false, and that can be combined with the propositional connectives ($\wedge$, $\vee$, $\neg$, $\rightarrow$ and $\leftrightarrow$). We were very successful in this but let's go back where we started, the study of the linguistic patterns of valid inferences. Are there valid inferences that are commonly used and yet that cannot be described in propositional logic? What about the following?

All 15-199 students love logic and Sidra is a 15-199 student
so Sidra loves logic. (5.1)

Is this a valid inference? Sure! However, we cannot use propositional logic to show that it is. Indeed, we have little choice but using a dictionary like the following:

$$\begin{array}{rcl} A & = & \text{“All 15-199 students love logic”} \\ B & = & \text{“Sidra is a 15-199 student”} \\ C & = & \text{“Sidra loves logic”} \end{array}$$

which yield the formal inference

$$A \underline{\textit{and}}\ B\ \underline{\textit{so}}\ C$$

which is clearly not valid since A and B can both be true while C is false. Yet, the above informal inference *is* valid: everybody with basic mental capabilities anywhere in the world will agree that it is valid.

To understand what is going on, let's make some experiments and vary this inference by making various statements in it false. Consider the following:

> *All 15-199 students love logic <u>and</u> Johnny Depp is a 15-199 student <u>so</u> Johnny Depp loves logic.* (5.2)

We all would love to see Johnny Depp — maybe in his pirate garb — sitting in class with us, but he is not, so the second premise is false. Yet, the overall inference is valid, just as a similar propositional experiment was in Chapter 1. Let's make yet another little change:

> *All 15-199 students love logic <u>and</u> Johnny Depp is a 15-199 student <u>so</u> Sidra loves logic.* (5.3)

This time, the inference cannot be valid because the conclusion has nothing to do with the premises: back to Aristotle's definition, it is not necessary knowledge. The fact that the second premise is false does not influence this.

The good news is that our definition of valid inference is safe. We just saw that inferences such as the above behave exactly like propositional inferences: intuitively, they are valid precisely when *there is no way the conclusion can be false if the premises are true*.

The bad news is that propositional logic is not expressive enough to capture what makes inferences (5.1) and (5.2) valid, and (5.3) invalid. We need to come up with an understanding of these new linguistic patterns and with a logic to reason about them correctly. That will be *predicate logic*. Specifically, we will look at a fragment known as *first-order logic*.

5.2 The Structure of Atomic Propositions

What has changed then?

In the previous chapters, we could break candidate inferences at the level of the elementary statements that occurred in them: to reason about the inference *“If Sidra is*

in Doha then she is in Qatar and Sidra is in Doha so she is in Qatar", all was needed was to find the basic propositions, here *"Sidra is in Doha"* and *"She is in Qatar"*, and to interpret the words around them as propositional connectives, obtaining *"$A \to B$ and A so B"*. Validity came from the way common pieces, here A and B, were used in different parts of the inference.

When looking at inferences such as (5.1), we need to look *inside* atomic propositions, at their *structure*. The reason this inference is valid has to do with the fact that the word *"Sidra"* occurs in both *"Sidra is a 15-199 student"* and *"Sidra loves logic"*, that the phrase *"loves logic"* appears in both *"All 15-199 students love logic"* and *"Sidra loves logic"*, and so on. To explore this idea, we need to build a little bit of infrastructure which will set the stage for developing a logic that accounts for this deeper structure of atomic propositions — predicate logic.

Look at the sentence

"Sidra is a 15-199 student"

To begin with, we need two extensions:

- be able to *name* the people, objects, entities, we want to talk about (here "Sidra", earlier "Johnny Depp");
- be able to *describe* them (here "is a 15-199 student" and earlier "loves logic").

Let's look into these two ingredients in turn, and then look at how they fit together.

5.2.1 Names

Names are labels for the entities (people, things, places, etc) that we refer to in sentences. Here are a few examples:

- "Sidra" is the name of the person Sidra. Note that "Sidra", a string that we write on a page, or even the sound that we make when calling her, is not the same as Sidra the person. They are rather ways to refer to her.
- "Qatar" is a the name of a country. Again, it is not the country but just an agreed upon way to refer to it. Indeed in ancient history, it was known by other names.
- "Logicomix" is the name of a book. Again, it is not the book, nor any specific copy we can hold in our hands.
- "Five" is the name of the number 5. Notice that "5" too is a name for this number in the European numeral system. Once more, neither "five" nor "5" are the number 5 (which is an abstract idea) but representations of it.

From now on, we will write names in *italics*, reserving quotes for English phrases, or pieces of phrases, that we are analyzing.

5.2.2 Predicates

A *predicate* is a way to describe something. It is a proposition with one or more pieces missing, pieces that we will denote as "…" for the time being. Here are some examples on the left-hand side of this table:

- "… is a 15-199 student" $\rightsquigarrow$ $student(\ldots)$
- "… love logic" $\rightsquigarrow$ $logicLover(\ldots)$
- "… is taller than …" $\rightsquigarrow$ $taller(\ldots,\ \ldots)$
- "… is a multiple of …" $\rightsquigarrow$ $multiple(\ldots,\ \ldots)$
- "… is between … and …" $\rightsquigarrow$ $between(\ldots,\ \ldots,\ \ldots)$
- "it is raining" $\rightsquigarrow$ $raining$

The first two have just one missing piece, they are *unary predicates*. The next two have two missing pieces, which makes them *binary* predicates. The fifth is an example of a *ternary* predicate. In general, if a predicate has n missing pieces, it is called an n-ary predicate. Some predicates have no missing pieces at all, for instance the last example above, "it is raining": it is a *nullary* predicate.

Just like we abbreviated atomic statements with propositional letters in Chapter 2, it is customary to abbreviate predicates by means of a short string such as $student$ or $logicLover$, followed by a list of the missing pieces in parentheses. Here, $student$ and $logicLover$ are called *predicate symbols*. And just as with which propositional letter to associate to a sentence, it is entirely our choice which predicate symbol we use for each English predicate. The right-hand side of the above list gives a predicate form for each of our examples. Notice we associated the nullary predicate "it is raining" with the symbol $raining$, which has no list of missing parts behind it.

For n-ary predicates with $n \geq 2$, we need to agree on which missing piece stands for what in the English version of the predicate. This is typically specified based on position. These conventions are expressed by giving a *reading* of the symbolic predicate. We will return to this later when we have a better way to distinguish missing pieces.

5.2.3 Elementary Propositions

By plugging the name $Sidra$ in the symbolic predicate $student(\ldots)$, we obtain the expression $student(Sidra)$, which naturally corresponds to the statement *"Sidra is a 15-199 student"* since the predicate $student(\ldots)$ stood for *"… is a 15-199 student"*. Schematically,

"Sidra"	+	*"… is a 15-199 student"*	=	*"Sidra is a 15-199 student"*
$\updownarrow$		$\updownarrow$		$\updownarrow$
$Sidra$	+	$student(\ldots)$	=	$student(Sidra)$

Because *"Sidra is a 15-199 student"* is an atomic proposition, a sentence which can be either true or false, we call a symbolic predicate with all the missing pieces replaced by names an *elementary proposition*. We can do the same with any predicate, with any number of missing pieces. This works even with nullary predicates: $raining$ corresponds to the statement *"it is raining"* without plugging anything since there are no missing pieces.

Thus, names are used to name objects, and predicates describe properties or relations of objects by referring to their names. Together they comprise the structure of an atomic proposition such as *"Sidra is a 15-199 student"*.[1]

Having obtained elementary propositions by plugging names for all the missing pieces in a predicate, we can combine them with propositional connectives as if they were propositional letters. For example, we can write

$$student(Sidra) \wedge logicLover(Sidra)$$

to express the composite statement *"Sidra is a 15-199 student and she loves logic"*.

With names and predicates, we have brought out the structure of elementary statements, those minimal true/false sentences that we associated to propositional letters in Chapter 2. We can then combine them by means of the usual propositional connectives. So far, we have not done anything we couldn't do in propositional logic. The additional structure however opens the door to writing formulas that take advantage of the fact that the same names appear in different elementary statements within a formula.

5.3 Quantifiers

We often have structured sentences that are true or false without referring to any specific name. Consider the first premise of inference (5.1):

"All 15-199 students love logic"

The part *"15-199 students"* is just the predicate *"... is a 15-199 student"* and similarly *"loves logic"* is the predicate *"... love logic"*. What binds them together is the word *"all"* which kind of acts like a name, but a common name between the two predicates. There are a lot of other words and phrases like *"all"* that play the same role. Here are a few:

"All	*15-199 students*	*love logic"*
"Some	*15-199 students*	*love logic"*
"No	*15-199 student*	*love logic"*
"Exactly two	*15-199 students*	*love logic"*
"Most	*15-199 students*	*love logic"*
"A few	*15-199 students*	*love logic"*

[1] Can we bring out even more structure? We can, and indeed that's part of the standard recipe of first-order logic. We are omitting one ingredient to keep things really simple. We will add it back in Chapter 8.

In a sense, these words and phrases describe the *quantity* of people having the various properties. They are called *quantifiers*.

Although there are lots of quantifiers out there, predicate logic deals with just two of them, two very important ones:

- the *universal quantifier*, which appears in common language as the phrases *"for all"*, *"all"*, *"each"*, *"every"*, *"any"* and many more. Logicians like to write it as an upside-down "A", specifically $\forall$.

- the *existential quantifier*, which you encounter as the phrases *"there exists"*, *"some"*, *"there are"*, and several others. Logicians write it $\exists$, an inside-out "E".

That takes care of two of our examples. The quantifier *"no"* is easily expressible using the universal and/or the existential quantifier since the sentence *"No 15-199 student loves logic"* has the same meaning as *"All 15-199 students don't love logic"* and of *"There doesn't exist a 15-199 student who loves logic"*. Quantifiers like *"exactly two"* and *"at most seven"* are a little bit more tricky because they force us to know when two things (or people) are equal — we will get back to this in Chapter 9. Other quantifiers, such as *"most"* and *"a few"*, are outside of predicate logic: they are linguistic patterns that predicate logic is unable to reason about.

To get started using quantifiers in formulas, we need to be able to talk about generic entities rather than specific names. For this, we introduce *variables* such as x, y and z. With variables, we can now write predicates that are *parametric*, like

$$student(x)$$

This parametric predicate says that *"x is a 15-199 student"*, for some unspecified x — this is the *reading* of that predicate. By contrast, the predicate $student(\ldots)$ focused on the property of "being a 15-199 student" — a subtle difference in emphasis.

Now, depending on the name we plug in for x, we get elementary propositions that can be either true or false. For example,

$$student(Sidra)$$

is true, while

$$student(JohnnyDepp)$$

is false. Now notice that $student(x)$ by itself is neither true nor false, however. It is suspended in a sense.

Having variables inside predicates, we can use the same variable in different parts of a formula. For example, we can write

$$student(x) \rightarrow logicLover(x)$$

which reads *"if x is a 15-199 student, then x loves logic"*. Notice that, again, it is neither true nor false: it depends on who x is. Said this, we can prefix this parametric

formula with quantifiers to obtain some of the statements we considered at the beginning of this section. For example, using the universal quantifier $\forall$, we can close this sentence over x as

$$\forall x.\, student(x) \rightarrow logicLover(x)$$

which reads *"for all x, if x is a 15-199 student, then x loves logic"*, which is a verbose way to say "*All 15-199 students love logic"*, the sentence we started with at the beginning of this section. Notice that, differently from $student(x) \rightarrow logicLover(x)$, this time we have a statement that *is* either true or false.

A simple variant of this statement gives us the reading *"No 15-199 student loves logic"* — simply negate the consequent of the implication:

$$\forall x.\, student(x) \rightarrow \neg logicLover(x)$$

or more literally, *"for all x, if x is a 15-199 student, then x does not love logic"*.

By using the existential quantifier, $\exists$, in a similar way, we can write

$$\exists x.\, student(x) \wedge logicLover(x)$$

that is *"there is x such that x is a 15-199 student and x loves logic"*, or more simply "*Some 15-199 students love logic"*, which is another true/false statement.

These are typical ways we use the logical quantifiers $\forall$ and $\exists$ to express English sentences that have to do with quantities. As you can see, it is not as direct as with propositional connectives (or better, English and other natural languages have very succinct ways of expressing these concepts). It will take a little bit of practice to become comfortable with them and the way they interact with other connectives. For instance, a formula involving $\forall$ very often also includes $\rightarrow$, as in the above examples, while $\exists$ is a good friend of $\wedge$.

5.4 The Language of Predicate Logic

We have covered a lot of ground so far in this chapter. Let us summarize all the ingredients of first-order logic we have surveyed by describing the language we will write formulas in, and then give a few additional definitions.

5.4.1 Terms and Formulas

To begin with, we need a number of *names* (also called *constants*), which we generically denote as a, b, c, possibly with subscripts and primes. It is a good thing to have enough names to talk about all the people, places and other entities we are interested in. Then, we need a supply of *variables*, denoted x, y, z also with subscripts and primes if needed. We make heavy use of variables, so we will assume there are infinitely many of them. Together, constants and variables are called *terms*.

We also need a collection of *predicate symbols* to represent the properties or relations we are interested about. When talking abstractly, we denote them p, q, etc, again

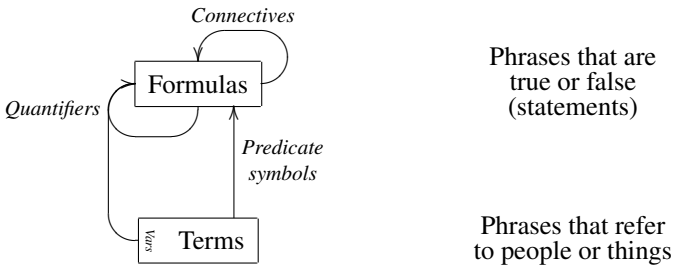
with subscripts and primes if necessary. Each predicate symbol takes a certain number of arguments — this number is called its *arity*. An *atomic formula*, or *elementary predicate*, is then just an n-ary predicate symbol p applied to n terms $t_1, \ldots, t_n$, so that it looks like $p(t_1, \ldots, t_n)$.

Before we define formulas, let's agree on how to denote them: we use the Greek letters φ and ψ decorated with the usual primes and subscripts when needed, and sometimes write $\varphi(x)$ to indicate a formula φ that may contain the variable x (more on this below). Then a *formula* is either atomic, a Boolean combination of formulas or a quantified formula. Specifically,

- Any atomic formula $p(t_1, \ldots, t_n)$ is a formula.
- If φ and ψ are formulas, then $\neg\varphi$ and $\varphi \wedge \psi$ and $\varphi \vee \psi$ and $\varphi \to \psi$ and $\varphi \leftrightarrow \psi$ are all formulas, and so are $\top$ and $\bot$.
- If $\varphi(x)$ is a formula, then $\forall x.\, \varphi(x)$ and $\exists x.\, \varphi(x)$ are also formulas.

These entities constitute a fragment of the language of *first-order logic* (we will be introduced to one last ingredient in Chapter 8). First-order logic is a type of predicate logic — in fact there are other ways to use terms, predicates and quantifiers.

A lot is going on in the above definition. The following schema describes how the various parts fit together.



First-order logic turns two types of phrases from everyday speech into symbols: phrases that indicate entities we want to talk about are expressed as terms, and phrases that are true or false (what we called *statements*) become formulas. Atomic predicates are where terms enter formulas, as arguments of predicate symbols. They enable us to express elementary statements about these entities. As in propositional logic, we can combine formulas using connectives. Additionally, quantifiers tighten up a formula by fixing how we understand their variables across atomic formulas. Notice that, in first-order logic, terms occur inside formulas, but there is no way to embed a formula inside a term.

Although the above definition prescribes that an atomic predicate have the form $p(t_1, \ldots, t_n)$, as in $student(Sidra)$ earlier, logicians are often a lot more relaxed, especially when writing formulas about mathematics. For example, the predicate for the phrase "x *is less than* y" is typically written $x < y$ rather than $lessThan(x, y)$ — the familiar $x < y$ is actually just $<(x, y)$ where the predicate symbol $<$ is written between

its arguments (or infix as this is called). Similarly, we can write $x + 1 \geq 0$ for "*the successor of x is greater than or equal to zero*".[2]

5.4.2 Binding and Scope

When writing $\forall x.\, \varphi(x)$ and $\exists x.\, \varphi(x)$, the formula $\varphi(x)$ is called the *scope* of the quantifier. This means that this quantifier refers to the occurrences of the variable x inside $\varphi(x)$ and nowhere else.

In Chapter 2 we relied on precedences and other conventions to simplify writing propositional formulas. We keep these conventions and actually extend them to make it convenient to write quantified formulas too. In particular, we assume that quantifiers have precedence lower than all other operators and that the scope of a quantifier extends to the end of the current formula. Just like with connectives, we use parentheses to override these conventions — this means that parentheses can make the scope of a quantifier as small as we want.

A variable x that occurs in the scope of a quantifier $\forall x.$ or $\exists x.$ is said to be *bound* — the quantifier *binds* it. A variable that is not bound is called *free*. Let's look at an example:

$$\forall x.\, student(y) \to logicLover(x)$$

The scope of the quantifier $\forall x$ is the formula $student(y) \to logicLover(x)$. The occurrence of x in $logicLover(x)$ is bound by this quantifier. The occurrence of the variable y in $student(y)$ is instead free. A sure way to tell that a variable is free in a formula is if there is no quantifier in the formula that binds this same variable.

A formula that has no free variables is said to be *closed*, and *open* otherwise. The formula in our last example is open because the variable y in it is free, and so is the formula $student(x) \to logicLover(x)$ in the last section. The formula $\forall x.\, student(x) \to logicLover(x)$ is instead closed. Notice that only closed formulas correspond to English sentences: we cannot say if an open formula is true or false because it depends on what the free variables stands for. From now on, we will be exclusively interested in closed formulas.

5.5 Turning Sentences into Predicate Logic

Consider a sentence we are now very familiar with:

Doha is in Qatar and Sidra is in Doha, so she is in Qatar.

That's our original inference from Chapter 1. How do we express it in first-order logic? Well, we first need to decide on what the entities we are interested in talking about are and what we want to say about them. This means fixing names and predicates. Already in this simple sentence, we have some choices.

[2] We will discover a better way to express this statement in Chapter 8.

- A reasonable setup is that we are interested in drawing inferences about various people and places. We then need to give them names so that we can refer to them. In this simple sentence, these names are $Sidra$, $Doha$ and $Qatar$, but we may have more if it were part of a larger description. The one relation used in that sentence is that of a person being in a place. This suggests a binary predicate $isIn(x, y)$ which we read as "person x is in place y". The above sentence is then translated into the following formal inference of first-order logic:

$$\begin{array}{ll} & isIn(Sidra, Doha) \rightarrow isIn(Sidra, Qatar) \\ \underline{and} & isIn(Sidra, Doha) \\ \underline{so} & isIn(Sidra, Qatar) \end{array}$$

- Say however that we are interested in the travels of Sidra and nobody else. Then, we do not even need to bother giving her a name since everything is about her. We still need to name the places she goes, here $Doha$ and $Qatar$, and we need to have a predicate that describes where she is — let's pick $sidraIn(y)$ to mean "Sidra is in place y". The above sentence is then translated into first-order logic as follows:

$$\begin{array}{ll} & sidraIn(Doha) \rightarrow sidraIn(Qatar) \\ \underline{and} & SidraIn(Doha) \\ \underline{so} & SidraIn(Qatar) \end{array}$$

- On the other hand, what we are interested in may be the coming and going of people in Doha and in Qatar, specifically. We need to give names to people, here $Sidra$ for Sidra, but we can use specific relations about the places. We then need two predicates: $inDoha(x)$ to mean that "person x is in Doha", and $inQatar(x)$ to mean that "x is in Qatar". In this scenario, our sentence if formalized as:

$$\begin{array}{ll} & inDoha(Sidra) \rightarrow inQatar(Sidra) \\ \underline{and} & inDoha(Sidra) \\ \underline{so} & inQatar(Sidra) \end{array}$$

- Finally, we may be interested in just Sidra and only about whether she is in Doha or in Qatar. We do not need any names in this case, just two predicates with no arguments: $sidraInDoha$ meaning that "Sidra is in Doha" and $sidraInQatar$ to mean that "Sidra is in Qatar". Our sentence becomes:

$$\begin{array}{ll} & sidraInDoha \rightarrow sidraInQatar \\ \underline{and} & SidraInDoha \\ \underline{so} & SidraInQatar \end{array}$$

Which logical representation should we choose, then? It very much depends on the situation we are trying to describe. If we need to refer to many people and places, then the last description is too poor. On the other hand, if we care only about Sidra, the first one, which would include her name in every atomic proposition, is overkill (although not wrong).

One thing that can force our hand is quantification: say we are asked to interpret the premise "Doha is in Qatar" as "Everybody in Doha is in Qatar". This forces us to quantify over people, thereby excluding our second and fourth options above. In fact, our choices for turning this premise into a formula would then be limited to either $\forall x.\, isIn(x, Doha) \rightarrow isIn(x, Qatar)$ or $\forall x.\, inDoha(x) \rightarrow inQatar(x)$.

As we were expressing the sentence "Doha is in Qatar and Sidra is in Doha, so she is in Qatar" above, we freely chose names and predicate symbols that reminded us of what we wanted them to mean. We could have used any symbols however — remember? we are in the business of *symbolic* logic. For example, in our first translation, we could have picked

$$\begin{array}{rcl} a & = & \text{"Sidra"} \\ b & = & \text{"Doha"} \\ c & = & \text{"Qatar"} \\ p(x,y) & = & \text{"}x \text{ is in } y\text{"} \end{array}$$

which would have given us the following first-order logic inference:

$$\begin{array}{ll} & p(a,b) \rightarrow p(a,c) \\ \underline{and} & p(a,b) \\ \underline{so} & p(a,c) \end{array}$$

Similarly in our last rendering, we could have chosen D and Q instead of $sidraInDoha$ and $sidraInQatar$, right? We would have gotten:

$$\begin{array}{ll} & D \rightarrow Q \\ \underline{and} & D \\ \underline{so} & Q \end{array}$$

But wait! This is exactly the propositional inference we wrote in Chapter 2!! A nullary predicate, one with no missing pieces, is just a propositional letter: it can be either true or false.

5.6 Valid Inferences

Let's go back to our original example, inference (5.1):

> *All 15-199 students love logic <u>and</u> Sidra is a 15-199 student*
> *<u>so</u> Sidra loves logic.* (5.4)

We can now easily express it in predicate logic as:

$$\begin{array}{ll} & \forall x.\, student(x) \rightarrow logicLover(x) \\ \underline{and} & student(Sidra) \\ \underline{so} & logicLover(Sidra) \end{array}$$

How do we show it is valid? Just like in the propositional case, we will have two ways:

1. We can show that in all situations where the premises are true, the conclusion must also be true. This will be akin to the truth table method, but as we will see not quite as easy.

2. We can give elementary derivations for the universal and existential quantifiers, just like we did with the propositional connectives.

We will look into these alternatives in the next two chapters.

5.7 *Exercises*

1. For each of the following expressions built out of the predicates $tender(x)$, $wolf(x)$, $sheep(x)$, $cabbage(x)$ and $eats(x, y)$, determine whether it is a well formed first-order formula. If it is not, explain what is wrong with it:

   - $\forall x.\, \neg tender(x) \rightarrow sheep(x) \wedge wolf(x)$
   - $eats(sheep(x), cabbage(x))$
   - $\exists x.\, \forall x.\, eats(wolf(x)) \leftrightarrow eats(x, sheep(x))$
   - $\forall x.\, \neg eats(cabbage(x), wolf(x))$
   - $\forall x.\, \forall y.\, \neg eats(\neg tender(x), y)$

2. Express each of the following English sentences as formulas in first-order logic :

   - *No math class is exciting, except logic.*
   - *Every weekday, the traffic is terrible at all hours of the mornings but only some hours in the evening.*

3. Consider the following first-order formula:

   $$\forall x.\, p(x) \rightarrow (\forall y.\, q(y, x) \rightarrow r) \wedge (\exists z.\, s(z, x) \wedge r)$$

   Is it closed or open? What is the scope of the variables x, y and z?

4. Consider the following reading for the predicates $C(x)$, $P(x)$, $F(x)$ and $S(x, y)$:

   $$\begin{array}{rcl} C(x) & = & \text{“}x \text{ is a Corvette”} \\ P(x) & = & \text{“}x \text{ is a Porsche”} \\ F(x) & = & \text{“}x \text{ is a Ferrari”} \\ S(x, y) & = & \text{“}x \text{ is slower than } y\text{”} \end{array}$$

   Translates the following formulas in good fluent English (we do not normally say things like “for all x such that x is a Corvette, …”):

   (a) $\forall x.\, \neg(C(x) \wedge F(x))$

   (b) $\exists x.\, P(x) \wedge \forall y.\, F(y) \rightarrow S(x, y)$

   (c) $\forall x.\, \forall y.\, P(x) \wedge S(x, y) \rightarrow C(y)$

(d) $\neg\exists x.\, C(x) \wedge F(x)$

(e) $\exists x.\, \forall y.\, F(y) \wedge C(x) \wedge S(x, y)$

(f) $\exists x.\, C(x) \wedge \forall y.\, F(y) \rightarrow S(x, y)$

(g) $\forall x.\, \forall y.\, \forall z.\, C(x) \wedge F(y) \wedge P(z) \rightarrow S(x, y) \wedge S(y, z)$

(h) $\forall x.\, P(x) \wedge \forall y.\, F(y) \vee C(y) \rightarrow S(x, y)$

(i) $\forall x.\, \forall y.\, \forall z.\, S(x, y) \wedge S(y, z) \rightarrow S(x, z)$

(j) $\forall x.\, P(x) \wedge \exists y.\, F(y) \vee C(y) \rightarrow \neg S(x, y)$

5. Consider the following reading of the constants q and o and of the predicates $G(x)$, $M(x)$ and $N(x, y)$:

$$\begin{array}{rcl} q & = & \text{“Qatar”} \\ o & = & \text{“Oman”} \\ G(x) & = & \text{“}x\text{ is a GCC country”} \\ M(x) & = & \text{“}x\text{ has mountains”} \\ N(x, y) & = & \text{“}x\text{ is north of }y\text{”} \end{array}$$

Translates the following formulas in good fluent English that a 6-year old can understand:

(a) $\forall x.\, G(x) \rightarrow M(x)$

(b) $\forall x.\, \forall y.\, \forall z.\, N(x, y) \wedge N(y, z) \rightarrow N(x, z)$

(c) $\exists x.\, \exists y.\, G(x) \wedge \neg G(y) \wedge N(y, x) \wedge N(x, q)$

(d) $\forall x.\, \exists y.\, G(x) \wedge N(x, y) \rightarrow M(y)$

(e) $\neg\exists x.\, N(o, x) \wedge G(x)$

(f) $\exists x.\, G(x) \wedge \forall y.\, M(y) \rightarrow N(x, y)$

6. Given the following readings for the constants s, h and c and for the predicates $L(x)$, $P(x)$ and $T(x, y)$,

$$\begin{array}{rcl} s & = & \text{“Prof. Sieg”} \\ h & = & \text{“Hilbert”} \\ c & = & \text{“Cantor”} \\ L(x) & = & \text{“}x\text{ is a logician”} \\ P(x) & = & \text{“}x\text{ is a philosopher”} \\ T(x, y) & = & \text{“}x\text{ talks about }y\text{”} \end{array}$$

translates the following formulas in simple English that a 6-year old can understand:

(a) $\forall x.\, T(s, x) \rightarrow L(x)$

(b) $\forall x.\, T(h, x) \rightarrow \neg T(x, x)$

(c) $\forall x.\, \forall y.\, T(x, y) \wedge T(y, x) \wedge L(x) \rightarrow P(y)$

(d) $\exists x.\, \forall y.\, T(y, x) \rightarrow \forall z.\, T(z, h)$

(e) $\forall x.\, \exists y.\, T(y, x) \to \forall z.\, T(z, h) \vee T(z, c)$

(f) $\exists x.\, T(s, x) \to x = h \vee x = c$

7. Consider the following constant and predicates:

$$\begin{array}{rcl} \mathbf{1} & = & \text{“the number 1”} \\ x < y & = & \text{“}x\text{ is smaller than }y\text{”} \\ \mathit{Prime}(x) & = & \text{“}x\text{ is a prime number”} \\ \mathit{Div}(x, y) & = & \text{“}x\text{ is divisible by }y\text{”} \end{array}$$

Translates the following formulas in good fluent English that a 10-year old can understand:

(a) $\forall x.\, \forall z.\, \exists y.\, x < z \to x < y \wedge y < z$

(b) $\forall x.\, \exists y.\, x < y \wedge \neg \exists y.\, \forall x.\, x < y$

(c) $\forall x.\, \exists y.\, y < x \wedge \neg \exists x.\, \forall y.\, y < x$

(d) $\forall x.\, \forall y.\, \forall z.\, \mathit{Div}(x, y) \to \mathit{Div}(y, z) \to \mathit{Div}(x, z)$

(e) $\forall x.\, \mathbf{1} < x \to \exists y.\, \neg(x = 1) \wedge \mathit{Div}(y, x)$

(f) $\forall x.\, \mathit{Prime}(x) \leftrightarrow (\forall y.\, \mathit{Div}(x, y) \to x = 1 \vee x = y)$

(g) $\forall x.\, \forall x'.\, \exists y.\, \mathit{Prime}(x) \wedge \mathit{Prime}(x') \wedge \forall z.\, \mathit{Div}(y, z) \to z = x \vee z = x'$

8. Using the following predicates with the given meanings,

$$\begin{array}{rcl} S(x) & = & \text{“}x\text{ is a spy novel”} \\ L(x) & = & \text{“}x\text{ is long”} \\ M(x) & = & \text{“}x\text{ is a mystery”} \\ B(x, y) & = & \text{“}x\text{ is better than }y\text{”} \end{array}$$

express the following English sentences in predicate logic:

(a) All spy novels are long.

(b) Not every mystery is a spy novel.

(c) Only mysteries are long.

(d) Some spy novels are mysteries.

(e) Spy novels are better than mysteries.

(f) Some mysteries are better than all spy novels.

(g) Only spy novels are better than mysteries.

(h) Some mysteries are better than others.

(i) Some mysteries are better than spy novels and others are worse.

(j) Nothing is better than a spy novel.

9. Using the following predicates with the given meanings,

$$\begin{array}{rcl} a & = & \text{“April 1st”} \\ P(x) & = & \text{“}x\text{ is a person”} \\ T(x) & = & \text{“}x\text{ is a time”} \\ F(x,y) & = & \text{“}x\text{ is fooled at }y\text{”} \end{array}$$

express the following English sentences in predicate logic:

(a) You can fool some people all the time.

(b) One can fool everybody on April 1st.

(c) Some people get fooled only on April 1st.

(d) You can't fool all the people all the time.

(e) Some people get fooled on April 1st, but not others.

(f) Some people get fooled on April 1st, while other people get fooled on other days.

10. Using the following predicates with the given meanings,

$$\begin{array}{rcl} \mathit{Judge}(x) & = & \text{“}x\text{ is a judge”} \\ \mathit{Chemist}(x) & = & \text{“}x\text{ is a chemist”} \\ \mathit{Lawyer}(x) & = & \text{“}x\text{ is a lawyer”} \\ \mathit{Young}(x) & = & \text{“}x\text{ is young”} \\ \mathit{Admires}(x,y) & = & \text{“}x\text{ admires }y\text{”} \end{array}$$

express the following English sentences in predicate logic:

(a) There are no young chemists.

(b) No young person is both a lawyer and a chemist.

(c) Chemists admire everybody.

(d) Some lawyers admire only judges.

(e) Every chemist admires some old lawyer.

(f) A judge admires a young person if and only if this person admires all judges.

(g) All lawyers admire any old judge who admires some chemist.

(h) Chemists only admires themselves.

(i) Judges admire all lawyers who don't admire themselves.

(j) Old people don't admire anybody.

11. Using the following predicates with the given meanings,

$$\begin{array}{rcl} \emptyset & = & \text{“the empty set”} \\ x \in y & = & \text{“}x\text{ is an element of }y\text{”} \\ x \subseteq y & = & \text{“}x\text{ is a subset of }y\text{”} \\ \mathit{Set}(x) & = & \text{“}x\text{ is a set”} \end{array}$$

express the following English sentences in predicate logic:

(a) The empty set is a subset of every set.

(b) The empty set is a superset of itself only.

(c) The empty set contains no elements.

(d) For any two sets, there is a third set each of whose elements belong to both.

(e) There is no set that contains all sets.

(f) Given a set and a subset, there is a set containing all the elements of the set that are not in the subset.

(g) Any set is an element of the set of its subsets.

(h) Every set except the empty set has an element.

12. Some English sentences are ambiguous: they can be translated into logic in several ways that mean different things. For example "Chemists only admires themselves" in exercise 2 can be interpreted as "Chemists only admire other chemists", or "Each chemist admires just him/herself". Each of the following sentences has multiple meanings. Express each of these meanings in predicate logic (the number of meanings is in brackets).

(a) Chemists only admire themselves. [2]

(b) Every Java expression has a type. [2]

(c) People in Doha speak two languages. [2]

(d) Everybody has a father and a mother. [4]

13. The Greek philosopher Aristotle (384-322 BC) studied under Plato and tutored Alexander the Great. His early investigations of logic influenced philosophers for over 2,000 years, that is until formal logic came about in the 1800's. He cataloged four "perfect" syllogisms that medieval scholars named *Barbara*, *Celarent*, *Darii* and *Ferio*. Translate each of them below into logical inferences of the form 'premises so conclusion'.

(a) [Barbara] All Q are R. All P are Q. Therefore, all P are R.

(b) [Celarent] No Q are R. All P are Q. Therefore, no P are R.

(c) [Darii] All Q are R. Some P are Q. Therefore, some P are R.

(d) [Ferio] No Q are R. Some P are Q. Therefore, some P are not R.

14. When writing code, an *assertion* is a formula that is true at the point in the program where it occurs. Good programmers often write assertions as comments in their code to justify why they wrote a condition or a statement in a certain way, and to communicate this reasoning to whoever will maintain the code. Consider as an example the following commented code for the function that computes the factorial a number (not that it is complicated, but it is a good example):

```
int factorial(int n) { /*  Assumption: n ≥ 0              */
  int x = 1;           /*  x = 1 ∧ n ≥ 0                  */
  int f = 1;           /*  x = 1 ∧ n ≥ 0 ∧ f = 1          */
  while (x < n) {      /*  1 ≤ x ∧ x < n ∧ f = (x − 1)!   */
    f = x * f;         /*  1 ≤ x ∧ x < n ∧ f = x!         */
    x++;               /*  1 < x ∧ x ≤ n ∧ f = x!         */
  }                    /*  x = n ∧ n ≥ 0 ∧ f = n!         */
  return f;            /*  x = n ∧ n ≥ 0 ∧ f = n!         */
}
```

The first assertion is an *assumption*: it states that this function will work for positive inputs but we don't promise anything for negative inputs (use at your own risk in that case!). Notice that the assertions inside the loop must work during all iterations: we cannot say more about `x` other than it is between the bounds defined in the program. Observe also that these formulas, although they make use of the predicates "$=$", "$<$", "$\leq$", etc, do not use quantifiers: the variables get their value from the program at that point during execution. Finally, pay particular attention to how each assertion is obtained from the previous one and what is going on in the program.

Your job in this exercise will be to add annotations to the following program that implements the bubble sort algorithm.

```
void bubblesort(int[] A) {          /*                    */
  int n = A.length;                 /*                    */
  boolean swapped = true;           /*                    */
  while (swapped) {                 /*                    */
    swapped = false;                /*                    */
    n--;                            /*                    */
    for (int i = 0; i < n; i++){    /*                    */
      if (A[i] > A[i+1]) {          /*                    */
        int tmp = A[i];             /*                    */
        A[i] = A[i+1];              /*                    */
        A[i+1] = tmp;               /*                    */
        swapped = true;             /*                    */
      }                             /*                    */
    }                               /*                    */
  }                                 /*                    */
}                                   /*                    */
```

Because this exercise is about sorting, you will find it convenient to use the predicate "$sorted(\mathtt{A}[j-k])$" which expresses the fact that the portion of array `A` starting at index j and ending at index k (both included) is sorted.

Chapter 6

First-Order Interpretations

It would be nice to determine whether an inference in predicate logic is valid by directly using the definition: *the conclusion cannot be false when all the premises are true*. To do this, we need a way to tell whether a formula in predicate logic is true or false. In propositional logic, this was easy: the truth tables were a simple construction to do just that.

To deal with quantifiers, we need to refine the simple truth assignments that constituted the rows of a truth table into a more elaborate *first-order interpretation* that takes into account the structure of predicate formulas. This will complicate matters significantly. In fact, not only will the necessary definitions be more complex, but determining the truth or falsehood of a formula, and therefore the validity of an inference, will not always be practically feasible in predicate logic.

6.1 Evaluating Elementary Formulas

Is the elementary formula

$$student(Sidra)$$

true or false?

I don't know. These are just a bunch of symbols on the page, in the same way as a propositional letter A is just a symbol — by itself it can be either true or false depending on what we choose it to stand for. In Chapter 3, we assigned A a truth value by defining an interpretation for it (remember? an interpretation was just a row in the truth table, an assignment of truth values to each propositional letter in a formula). We will need to proceed in the same way here and assign an *interpretation* to $student(Sidra)$, but "interpretation" will mean something different.

One idea is to treat elementary formulas such as $student(Sidra)$ just like propositional letters. Then, we can assign truth values to them and use the truth tables to calculate the truth value of larger formulas that combine them using the Boolean connectives. But how to compute the truth value of formulas that start with a quantifier? How do

we tell if $\forall x.\ student(x) \to logicLover(x)$ is true or if it is false? Having exposed the structure of atomic statements, we will need to assign a meaning to all of their parts: the name $Sidra$ refers to the same entity in both $student(Sidra)$ and $logicLover(Sidra)$, doesn't it? Similarly, the predicate symbol $student$ in $student(Sidra)$ and in the above universal formula should have a consistent meaning. A first-order interpretation makes this intuition precise.

Let's start with the *intended interpretation* of the formula $student(Sidra)$: in our mind, the name $Sidra$ stands for the person Sidra, and the predicate symbol $student$ for the unary predicate *"... is a 15-199 student"*. According to this interpretation, $student(Sidra)$ is certainly true since she is a registered student of 15-199 and shows up regularly to class.

However this is not the only possible interpretation of these symbols: the name $Sidra$ could stand for the nearby hospital of the same name and $student$ for the predicate *"... is a music student"*. Then, $student(Sidra)$ would make little sense and therefore be false. Less intuitive interpretations are possible too: $Sidra$ could be a code name for the number 5 and $student$ be a hidden way to refer to the predicate *"... is a prime number"*, in which case $student(Sidra)$ is true. In Chapter 3, we likened the rows of a truth table to far away planets, each with its own view of the truth or falsehood of the propositional letters. Same thing here: in our world, $Sidra$ is the name of a particular student and $student$ stands for the predicate *"... is a 15-199 student"*. But these same symbols can indicate something different on other planets (or even in different parts of our own world). We will need to account for all the meanings that the inhabitants of the countless places in the universe can give to these symbols.

All this makes intuitive sense. Let's come up with a general definition of what constitutes an interpretation for a formula in predicate logic. We need three ingredients:

- **Domain of interpretation.** First of all, we need to agree on what our formulas talk about. In our ongoing example, we have been talking about people doing various things, like being students and liking logic. The set of the entities we are talking about, like people, places or numbers, is called the *domain of interpretation*, or sometimes *domain of discourse*. It is a set that will form the basis of our interpretation — its importance will become fully apparent when we start looking into determining the truth value of a quantified formula. We often write $\mathbb{D}$ for the domain of an interpretation, especially when we want to stay generic.

  We can choose the domain of an interpretation to be what we want. In Sidra's example, we could choose a specific set of students, the set of all people on earth, or the set of all things in the universe, even a set of abstract ideas like the real numbers would do. Often, we have a specific domain in mind when writing a formula, here the set of people involved with the 15-199 course for example. That's the *intended domain*.

  As we develop our example, we will use the following domain of interpretation, where we use small pictures to represent various people:

  $\mathbb{D} = \{$   $\cdots$     $\}$

- **Interpretation of names.** The second part of an interpretation is an understanding of what the names stand for. As we saw, $Sidra$ is just a symbol. The *interpretation of names* is what allows us to say that by $Sidra$ we mean the person Sidra. Mathematically, it is just a function that maps every symbolic name we want to use, like $Sidra$, to an element in the domain of discourse, here Sidra the person. Again, it is our choice how we define it, although we generally have an *intended interpretation of names* in mind.

  In our example, the following table maps names like $Sidra$ and $JohnnyDepp$ to pictures in the above domain of interpretation:

Name	$\mathbb{D}$
$Sidra$	
$Hanan$	
⋮	⋮
$Iliano$	
$JohnnyDepp$	
$TomCruise$	

  Note that there may be elements of the domain of interpretation we do not have a name for, like  here.

- **Interpretation of predicates.** The third part of an interpretation is an understanding of for which members of the domain of discourse the predicates are true. For example, having interpreted the names $Sidra$ and $JohnnyDepp$ as the student Sidra () and the actor Johnny Depp (), we want to specify that $student(Sidra)$ is true but instead $student(JohnnyDepp)$ is false, so that the predicate $student(x)$ actually means *"x is a 15-199 student"* in our world. The *interpretation of predicates* does exactly that: for each predicate, it says on which elements of the domain of interpretation it is true and on which it is false. When all predicates are unary, it is convenient to display it as a table like the following, where the leftmost column lists the elements of the domain of interpretation (including those we don't have a name for, like ) and each remaining column stands for a predicate:

$x \in \mathbb{D}$	$student(x)$	$logicLover(x)$
	T	T
	T	T
⋮	⋮	⋮
	F	T
	F	T
	F	F
	F	F

> Mathematically, the interpretation of predicates is a function that maps every predicate name, like $student$ or $logicLover$, to the subset of the domain of discourse $\mathbb{D}$ where it is true.[1]

These three ingredients, a domain of discourse $\mathbb{D}$, an interpretation of names on $\mathbb{D}$, and an interpretation of predicates on $\mathbb{D}$, define a *first-order interpretation*. The notion of interpretation, pretty much as just described, was invented by the logician Alfred Tarski (1901–1983). We will write $\mathcal{M}$ to denote an interpretation.

Just as in the propositional case, we will write $\mathcal{M} \models \varphi$ if the (closed) formula φ is true in interpretation $\mathcal{M}$ (in which case $\mathcal{M}$ is called a *model* of φ), and $\mathcal{M} \not\models \varphi$ otherwise.

How do we check whether $student(Sidra)$ is true or false, then? In the interpretation we have been building (call it $\mathcal{M}$), we look for the element of the domain of discourse the name $Sidra$ corresponds to (that's ), and then we check whether the interpretation of the predicate $student$ is true or false for this element. Since $student(x)$ is true for $x =$  , we have that

$$\mathcal{M} \models student(Sidra)$$

Doing the same thing for $student(JohnnyDepp)$ tells us that $student(x)$ is false for $x =$  , which is the interpretation of the name $JohnnyDepp$. Therefore,

$$\mathcal{M} \not\models student(JohnnyDepp)$$

Given an interpretation $\mathcal{M}$, we can always determine the truth value of any elementary predicate in this way, as long as it does not contain variables.

6.2 Evaluating Propositional Connectives

Having a way to know the truth value of elementary formulas, we can use the truth tables from Chapter 3 to determine the truth value of any Boolean combination of elementary formulas, i.e., of any formula obtained by means of the standard propositional connectives on elementary formulas. For example, in the interpretation $\mathcal{M}$ described in the above tabular form, we have that

$student(Sidra) \to logicLover(Sidra)$	is true
$student(JohnnyDepp) \to logicLover(JohnnyDepp)$	is also true

or using the notation we have just (re)introduced,

$$\mathcal{M} \models student(Sidra) \to logicLover(Sidra)$$
$$\mathcal{M} \models student(JohnnyDepp) \to logicLover(JohnnyDepp)$$

[1] That's for unary predicates. In general, it maps an n-ary predicate to the subset of $\mathbb{D}^n$ where it is true. In particular, this means that each nullary predicate is either always true or always false, just like propositional letters were in Chapter 3.

6.3 Evaluating Quantified Formulas

What we are left with is finding a way to determine the truth value of a quantified formula. How do we go about it?

Consider the formula

$$\forall x.\, student(x) \rightarrow logicLover(x)$$

We just learned how to determine the truth value of $student(x) \rightarrow logicLover(x)$ for any specific value of x. Then, $\forall x.\, student(x) \rightarrow logicLover(x)$ shall be true simply if $student(x) \rightarrow logicLover(x)$ is true for all values of x.

Now, what are the legal values the variable x can assume? Intuitively, that's what the interpretation associates with names like $Sidra$ or $JohnnyDepp$. We will go one step further and let x range over the entire domain of interpretation, that is we will check that $student(x) \rightarrow logicLover(x)$ is true where x is replaced with any element in $\mathbb{D}$ (even the ones that don't have a name associated with them).[2]

Let's try it relative to the interpretation $\mathcal{M}$ (partially) displayed above: everybody is a student except for the professor and the actors, and everybody loves logic except Tom Cruise (let's assume). Then, we can extend the table with one more column for the formula $student(x) \rightarrow logicLover(x)$ and use the standard truth tables to determine its value for each element of the domain. Here is what we obtain:

x	$student(x)$	$logicLover(x)$	$student(x) \rightarrow logicLover(x)$
	T	T	**T**
	T	T	**T**
$\vdots$	$\vdots$	$\vdots$	$\vdots$
	F	T	**T**
	F	T	**T**
	F	F	**T**
	F	F	**T**

This shows that $student(x) \rightarrow logicLover(x)$ is true for every row in the table. Therefore, for this particular interpretation, the quantified formula $\forall x.\, student(x) \rightarrow logicLover(x)$ is true, i.e.,

$$\mathcal{M} \models \forall x.\, student(x) \rightarrow logicLover(x)$$

Had there been a student in the class who hated logic (Sidra maybe?), the implication would be false for this student and with it $\forall x.\, student(x) \rightarrow logicLover(x)$.

This same technique works equally well for existentially quantified formulas. For example, $\exists x.\, student(x) \wedge \neg logicLover(x)$ would yield a similar table, from which we could read off its truth value.

[2] In general, different variables can have different domains of interpretation. For example, consider a binary predicate $student'(x, y)$ whose intended reading is *"x is a student in class y"*. Then, the domain of discourse of x (or more precisely of the first argument of $student'$) is some set of people, while y will range over possible class numbers. So, the domain of discourse of an interpretation can be the union of various sets. In this case we have what is called a *multi-sorted interpretation*.

6.4 The Trouble with Infinity

Determining the truth value of the quantified formulas in the above examples was fine, actually easy, because there were just a handful of elements in the domain of interpretation. For larger sets, for instance all people in the university, it becomes more laborious but still possible. We have a problem however, when the domain of discourse is infinite ... Infinite? How can that be? Consider the following formula

$$\forall x.\, x > 0 \rightarrow x - 1 \geq 0$$

It is true of any of the standard sets of numbers we learn about in school, for example the natural numbers. So, let's fix the domain of discourse to be $\mathbb{N}$ and use the above method to show that it is true with respect to that domain.

x	$x > 0$	$x - 1 \geq 0$	$x > 0 \rightarrow x - 1 \geq 0$
0	F	T	**T**
1	T	T	**T**
2	T	T	**T**
$\vdots$	$\vdots$	$\vdots$	$\vdots$

There is a problem: this table is infinite! So, how can we make sure in a finite time that the column on the right-hand side only contains T? We cannot use the above method to determine whether the formula $\forall x.\, x > 0 \rightarrow x - 1 \geq 0$ is true or false.[3] In general we cannot show that a universal formula such as the above is true when the domain of interpretation is infinite.

The same problem emerges when trying to show that an existential formula is false with respect to an infinite domain of discourse. Take for example the formula $\exists x.\, x > 0 \wedge prime(x^x)$, where the reading of $prime(x)$ is *"x is a prime number"*. In $\mathbb{N}$, the table is infinite and the value of $x > 0 \wedge prime(x^x)$ for progressively larger values of x is F, but we cannot conclude that there is no value of x that makes it true.[4]

6.5 Counterexamples and Witnesses

Based on the above experiments, it looks like we cannot say anything useful about quantified formulas when the domain of discourse is infinite. This is not quite correct. Consider the formula

$$\forall x.\, x > 0 \rightarrow even(x)$$

where the reading of the predicate $even(x)$ is *"x is an even number"*. Let's again use the natural numbers, $\mathbb{N}$, as our domain of discourse. The interpretation table for the

[3] The formula is actually always true in $\mathbb{N}$, but we would need to use mathematical induction for example to show it. We will explore induction in Chapter 10.

[4] The formula $\exists x.\, x > 0 \wedge prime(x^x)$ is actually false in $\mathbb{N}$, which is proved by considering three cases: $x = 0$, $x = 1$, and $x \geq 1$.

subformula $x > 0 \to even(x)$ is infinite, of course, but when we start writing it down, we quickly find a number that makes it false (actually lots of them):

x	$x > 0$	$even(x)$	$x > 0 \to even(x)$	
0	F	T	**T**	
1	T	F	**F**	◀
2	T	T	**T**	
3	T	F	**F**	◀
⋮	⋮	⋮	⋮	

Because there are instances of $x > 0 \to even(x)$ that are false, the formula $\forall x.\, x > 0 \to even(x)$ is false too since that subformula is not true for all elements of the domain of discourse. If we call this interpretation $\mathcal{M}$:

$$\mathcal{M} \not\models \forall x.\, x > 0 \to even(x)$$

Domain elements such as 1 and 3 above are called *counterexamples*. We found them by writing progressively more rows of this table until one of them gave us the truth value F.[5]

The same approach allows us to determine that an existential formula is true. For example, to show that $\exists x.\, x > 0 \wedge even(x)$ is true, we could write a table similar to the last one and we would quickly see that $x > 0 \wedge even(x)$ for $x = 2$. This makes the overall formula true. This value of x is called a *witness* in the case of existential formulas (the word "counterexample" is used only for universal formulas that are false).

The following table summarizes what we can hope to achieve when evaluating a quantified formula with respect to an infinite domain of discourse like $\mathbb{N}$. The construction just described can be used for true existential formulas (as we will eventually encounter a witness) and false universal formulas (as some row will expose a counterexample), as long as the formula inside the quantifier can be evaluated in finite time. By contrast, true universal and false existential formulas are hopeless using this method because we would need to generate the whole table, which is infinite.

	$\forall$	$\exists$
T	×	✓
F	✓	×

6.6 Validating Inferences

Given an interpretation $\mathcal{M}$, a combination of the techniques we have just examined allows us to determine whether a generic formula φ is true in $\mathcal{M}$, although it does not

[5]This construction is based on $\mathbb{N}$ being *enumerable*: we can write down all the natural numbers one after the next — this is what enables us to list them in a table, albeit an infinite one. By contrast, $\mathbb{R}$ is *not* enumerable: there is no "next" element in $\mathbb{R}$ after π, so that we cannot make a table of all real numbers. Consequently, finding a counterexample in $\mathbb{R}$ or other non-enumerable domain is a matter of trial and error.

work too well in practice when $\mathcal{M}$ is infinite. Ignoring this "little detail", we can use the definition of validity to check whether a given inference is valid in $\mathcal{M}$: *it is never the case that the premises are all true but the conclusion is false*.

This seems easy. Consider an interpretation whose domain contains just $Sidra$, a well known student and logic lover. Then, the following inference is valid with respect to this interpretation:

$$\forall x.\, student(x) \;\; \underline{so} \;\; \forall x.\, logicLover(x)$$

This seems strange: it says that whenever everybody is a student, then everybody loves logic. What about those students who hate logic? There aren't any in this interpretation, but we can certainly think about other interpretations where there are students who do not like logic.

Our tentative definition above referred to a single interpretation. This is not enough: it would be like defining the validity of a propositional formula on the basis of a single row of its truth table. Instead, we know that we need to consider every row, i.e, every interpretation. The same happens in predicate logic: we need to consider every interpretation to tell whether an inference is valid or not. *All interpretations?* Even those nasty infinite ones? Yes!

An inference in predicate logic is valid if every interpretation that makes all its premises true also makes its conclusion true. Using symbols, an inference φ_1 and $\ldots$ and φ_n so ψ is valid if for every interpretation $\mathcal{M}$ such that $\mathcal{M} \models \varphi_1$ and $\ldots$ and $\mathcal{M} \models \varphi_n$, we have that $\mathcal{M} \models \psi$. As in the propositional case, we then write

$$\varphi_1, \ldots, \varphi_n \models \psi$$

As usual, we often use the Greek letter Γ as an abbreviation for the premises $\varphi_1, \ldots, \varphi_n$, obtaining $\Gamma \models \psi$. The special case where there are no premises, i.e., $\models \psi$ means that the formula ψ is true in every interpretation. This is the predicate logic equivalent of the propositional notion of tautology.

So, how do we go about showing that an inference is valid in practice? This is tough. Usually one reasons abstractly over a completely generic domain. We will not do this here, but it is not difficult to show for example that

$$p(a), \;\; \forall x.\, p(x) \to q(x) \;\; \models \;\; q(a)$$

where a is some name and p and q are predicate symbols. This is good to know because that's just a generic representation of Sidra's inference, so

$$student(Sidra), \forall x.\, student(x) \to logicLover(x) \models logicLover(Sidra)$$

It also matches weirder inferences like

$$3 > 0, \;\; \forall x.\, x > 0 \to even(x) \;\; \models \;\; even(3)$$

which seem wrong until we realize the second premise is false.

The above definition is however very useful to show that an inference is *not* valid: simply find some interpretation that makes all premises true but where the conclusion is false. For instance, let's show that

$$\not\models \forall x.\, (\exists y.\, p(x, y) \rightarrow \forall z.\, p(x, z))$$

Since there are no premises, all it takes is to show that the conclusion is false in some interpretation. Take the domain of discourse $\mathbb{D}$ to be the natural numbers, $\mathbb{N}$, and consider the predicate $p(x, y)$ to be $x < y$. It is then easier to make sense of it if we rewrite it accordingly as $\forall x.\, (\exists y.\, x < y \rightarrow \forall z.\, x < z)$. Now, this is clearly invalid: for $x = 3$, we get $\exists y.\, 3 < y \rightarrow \forall z.\, 3 < z$ and, for any value of y that makes the antecedent true, for example $y = 4$, we can find a value of z that makes the consequent false, e.g., $z = 2$. The formula, and therefore the above inference, is false.

6.7 Model Checking

We have just seen that verifying that an inference is valid, i.e., that $\Gamma \models \psi$, for premises $\Gamma = \varphi_1, \ldots, \varphi_n$ is really hard in general. Yet, it has been very successful in many cases where the intended domain of interpretation if finite, or can be turned into a finite domain using some tricks. Doesn't that go against having to consider all possible interpretations? It doesn't actually because it is often possible to write the premises of an inference so that only very specific interpretations will make them true. When this works well, this technique goes by the name of *model checking*.

One famous use of model checking has to do with designing correct digital circuits such as microprocessors. The circuitry of some component of interest in the microprocessor is expressed as a predicate logic formula that exactly describes its input/output behavior. It is used as the premise of an inference. The conclusion is some logical description of a desired correctness property this component should obey. Then, showing that the inference is valid corresponds to showing that the component actually has this property. If instead the inference is not valid, the counterexample that pops out indicates exactly what went wrong, which can be corrected by the engineering team before trying again.

Model checking is nowadays commonly used in industrial settings that have to do with hardware development, and is now spreading to other fields. Logicians and computer scientists all over the world are working on optimizations and generalizations that allow further broadening its range of application.

6.8 *Exercises*

1. Construct a first-order interpretation whose domain of discourse contains the members of your immediate family and defines two predicates for them, $male(x)$ and $female(x)$. Using this interpretation, check that the formula $\forall x.\, male(x) \vee female(x)$ is true (in most families it is). Then, extend the domain of discourse so that this same formula is false in this updated interpretation.

2. Consider the first-order interpretation whose domain of discourse is the set $\mathbb{Z}_3 = \{0, 1, 2\}$, that interprets names $\mathbf{0}$ as 0 and $\mathbf{1}$ as 1, and interprets the predicates $\mathit{diff}(x, y)$ as "x is not equal to y" and $\mathit{mult}(x, y, z)$ as "x times y modulo 3 is equal to z. Compute the truth value of the formula

$$\forall x.\, \mathit{diff}(x, \mathbf{0}) \to \exists y.\, \mathit{mult}(x, y, \mathbf{1})$$

3. Find a first-order interpretation where the formula

$$\forall x.\, \mathit{diff}(x, \mathbf{0}) \to \exists y.\, \mathit{mult}(x, y, \mathbf{1})$$

is *not* true.

4. We saw in Section 6.4 that we cannot tell that the formula $\exists x.\, x > 0 \wedge \mathit{prime}(x^x)$ is false in the expected interpretation based on the natural numbers: we would have to check that all the rows of an infinite table contain false. One idea is to use an equivalence you may have heard about, $\exists x.\, \varphi(x) \equiv \neg\forall x.\, \neg\varphi(x)$ — note that we are not yet able to verify that it is indeed an equivalence. Does rewriting the above formula as $\neg\forall x.\, \neg(x > 0 \wedge \mathit{prime}(x^x))$ solve our problem? Explain.

5. We know that propositional logic is a special case of first-order logic, right? Then it make sense that propositional interpretations (that is the rows of a truth table) are just a special case of first-order interpretations. Explain why this is the case. How do you reconcile the fact that there are infinitely many first-order interpretations to look at but it is enough to consider just 2^n interpretations for a formula (or inference) with n propositional letters?

6. The simple minded approach to establishing validity seen in this chapter is not practical because it asks us to consider infinitely many interpretations. What if we consider instead a *generic* interpretation? In some case, that's just what we need. Show that the inference $p(a), \forall x.\, p(x) \to q(x) \models q(a)$ is valid in a generic interpretation, for which you do not know any detail.

7. Sometimes, when making inferences, we have interpretations with certain characteristics in mind. Consider the candidate inference

$$\forall x.\, p(x), \forall x.\, p(x) \to q(x) \models \exists x.\, q(x)$$

and find an interpretation in which it is not valid. Now, by adding a simple premise that involves just the predicate symbol p to the left of $\models$, we can make this inference valid. This amounts to forcing the interpretations we consider to have certain characteristics.

8. If a formula contains only universal quantifiers, one way to show that it is not true is to simply find values for the quantified variables that make the inner formula false. For example, in $\mathbb{Z}$, $\forall x.\, x^2 > 0$ is false because for $x = 0$, we obtain $0^2 > 0$, which is false. The value $x = 0$ is a counterexample. Given the following predicates with the indicated meanings:

$$\begin{aligned} \mathit{prime}(x) &= \text{"}x \text{ is a prime number"} \\ \mathit{odd}(x) &= \text{"}x \text{ is an odd number"} \end{aligned}$$

find counterexamples to the following formulas where the variables range over $\mathbb{N}$:

(a) $\forall x.\, prime(x) \to odd(x)$

(b) $\forall x.\, x > 1 \to 2^x > 2x$

(c) $\forall x.\, prime(x) \to prime(2^x - 1)$

(d) $\forall x.\, \neg prime(x) \wedge x > 1 \to prime(2^x - 3)$

(e) $\forall x.\, \forall y.\, odd(x + y) \to \forall z.\, prime(z) \to odd(xz) \vee odd(yz)$

9. Do the same exercise, but this time with the variables ranging over the given domain.

(a) In $\mathbb{Z}$: $\forall x.\, \forall y.\, (x + y)^2 > x^2 + y^2$

(b) In $\mathbb{Z}$: $\forall x.\, x^2 - 1 = 0 \to x = 1$

(c) In $\mathbb{Z}$: $\forall x.\, \forall y.\, x - y < 0 \to y > x^2$

(d) In $\mathbb{R}$: $\forall x.\, 1/(1/x) = x$

(e) In $\mathbb{R}$: $\forall x.\, \log_2 x > 0 \leftrightarrow x > 0$

10. A more general way to show that a universal formula is not true is to replace some of its universal variables with carefully chosen values from the domain of discourse and give an argument as to why the rest of the formula cannot be true. For example, in $\mathbb{Z}$ the formula $\forall x.\, \forall z.\, \exists y.\, x < y \wedge y < z$ is false because for $x = 0$ and $z = 1$, this reduces to $\exists y.\, 0 < y \wedge y < 1$, but there are no integers between 0 and 1. Use this approach to show that the following formulas are false in the given domain.

(a) In $\mathbb{N}$: $\forall x.\, \exists y.\, y < x$

(b) In $\mathbb{Z}$: $\forall x.\, x < 0 \to \exists y.\, y > 0 \wedge \forall z.\, x + y = z$

(c) In $\mathbb{N}$: $\forall x.\, \exists y.\, x + y = 0$

(d) In $\mathbb{R}$: $\forall a.\, \forall b.\, \forall c.\, \exists x.\, ax^2 + bx + c = 0$

(e) In $\mathbb{R}$: $\forall x.\, x > 0 \to \exists y.\, y > x \wedge y = \log_2 x$

Chapter 7

Derivations in Predicate Logic

As we saw in the last chapter, the definition of valid inference is useful in specific domains (that's where model checking rules) or for showing non-validity. However, it is not a practical approach in general because of the necessity to consider all possible interpretations, especially those that are infinite.

The other way we can show that an inference is valid is by forgetting about interpretations and instead building a derivation for it. Following the recipe we used in the propositional case, all that appears to be needed is to give the elementary inference rules that govern the universal and existential quantifiers, and then combine them with the rules for the propositional connectives from Chapter 4. *Et voilà!* We will do precisely this, and it will be very easy. However, it will not completely solve the problem: although the infrastructure will be much lighter than the first-order interpretations of the last chapter, finding a derivation for a valid inference sometimes requires substantial ingenuity, to the point that no automatic tool can do this for us, in general.

7.1 Universal Quantifier

Recall from Chapter 4 that the elementary derivations for a logical operator fall in two categories: the elimination rules tell us what new information we can deduce if we know that a formula with this operator is true, and the introduction rules describe all possible ways to build a formula with this operator. Recall also that we write $\Gamma \vdash \psi$ to indicate that the inference with premises Γ and conclusion ψ is valid according to this definition, i.e., that there is a derivation of the formula ψ from the formulas Γ.

Let's start with the elimination rule for $\forall$, which is very intuitive. If we know that $\forall x.\, \varphi(x)$ is true, what can we conclude? Well, if $\varphi(x)$ holds for every x, then it shall hold for any name a we may care to substitute for x in it. More generally, it will hold for any term t we substitute for x in $\varphi(x)$ — recall that t is can be either a name or a variable, and a variable stands for a name (the more general form will be particularly convenient in Chapter 8). This intuition is formally expressed by the

following inference rule:

$$\frac{\forall x.\, \varphi(x)}{\varphi(t)}\,\forall_{\mathsf{E}}$$

This rule already gives us a simple way to show that the inference $p(a),\ \ \forall x.\, p(x) \to q(x) \ \vdash\ q(a)$ from Chapter 6 is valid:

$$\dfrac{p(a) \qquad \dfrac{\forall x.\, p(x) \to q(x)}{p(a) \to q(a)}\,\forall_{\mathsf{E}}}{q(a)}\,\to_{\mathsf{E}}$$

Recall that this is a generic version of Sidra's inference: we have indeed just shown that

$$\begin{array}{ll} & \forall x.\, student(x) \to logicLover(x) \\ \underline{and} & student(Sidra) \\ \underline{so} & logicLover(Sidra) \end{array}$$

is valid. This had eluded us so far.

When can we conclude $\forall x.\, \varphi(x)$? To make sense of this, let's look at how we write proofs of the form *"For all x, ..."*. Take as an example the simple property

> *For every natural number x, if x is odd, then $x + x$ is even*

A standard proof would go as follows:

> *Let n be any number and assume that n is odd. Then there exists m such that $n = 2m + 1$. Under this assumption, $n + n = (2m + 1) + (2m + 1)$, which is equal to $4m + 2 = 2(2m + 1)$ which is an even number.*

Here, we have replaced the variable x with a new symbol that we called n. We used n to denote an arbitrary number (it is not a nickname for some number we already know about). We used it as a *temporary name* in the proof of the property. This suggests one of those strange rules where we assume stuff specifically for the purpose of deriving a formula, but cannot use it for other purposes:

$$\frac{\begin{array}{c}(a)\\ \vdots \\ \varphi(a)\end{array}}{\forall x.\, \varphi(x)}\,\forall_{\mathsf{I}}$$

This rule looks a little bit like, say, the implication introduction rule from Chapter 4, but there is a big difference: what we are assuming is not a formula (a temporary premise) but a symbol (a temporary name). Again, we are not allowed to use a outside of the subderivation of $\varphi(a)$.

7.2 Existential Quantifier

The introduction rule for the existential quantifier is easy: when can we conclude that $\exists x.\, \varphi(x)$ is true? Well, if we know that $\varphi(a)$ for some name a. Just as in rule $\forall_{\mathsf{E}}$, we can use a generic term t, including a variable. This immediately leads to the following rule:

$$\frac{\varphi(t)}{\exists x.\, \varphi(x)}\,\exists_{\mathsf{I}}$$

Here, the term t corresponds to the *witness* mentioned when discussing interpretations.

Maybe not too surprisingly, the elimination rule for the existential quantifier is not as straightforward. It is actually pretty strange. To understand how it works, consider how we would go about proving a property of the form *"if there exists x such that ..., then ..."*. Let's work with a somewhat silly statement:

> *If there is x such that $x^2 = -1$, then there is a y such that $y^2 + 4 = 0$.*

A detailed proof of this property would proceed as follows:

> *Assume that there is an x such that $x^2 = -1$ and call it i. Thus, we are assuming that $i^2 = -1$. Now, let $j = 2i$. Then, $j^2 = (2i)^2 = 2^2 \times i^2 = 4 \times (-1) = -4$ and therefore $j^2 + 4 = 0$. Therefore there exists a y such that $y^2 + 4 = 0$.*

There is a lot going on here, so let's pick the various pieces apart. We first assumed the antecedent of the implication (essentially applying rule $\rightarrow_{\mathsf{I}}$). Then, we created a new name, i, for the value x that we assumed existed. We then used i to define another value j and used it as a witness for proving the consequent of the application (using rule $\exists_{\mathsf{I}}$ in fact). Rephrasing this explanation, assuming that the existential formula in the antecedent was true, we defined a temporary name i for the witness and used it to prove the consequent. The rule that emerges has the following form:

$$\frac{\exists x.\, \varphi(x) \qquad \begin{array}{c}\overline{\varphi(a)}^{\,(a)}\\ \vdots \\ \psi\end{array}}{\psi}\,\exists_{\mathsf{E}}$$

In words, knowing that $\exists x.\, \varphi(x)$ is true, we give a name a to this x that is supposed to exist and assume that $\varphi(a)$ holds. If from this we can prove some other formula ψ, then ψ can be proved just on the basis of $\exists x.\, \varphi(x)$. The name a is temporary and can be used only for the purpose of showing that ψ holds. It cannot appear in ψ itself. The assumption $\varphi(a)$ is also temporary.

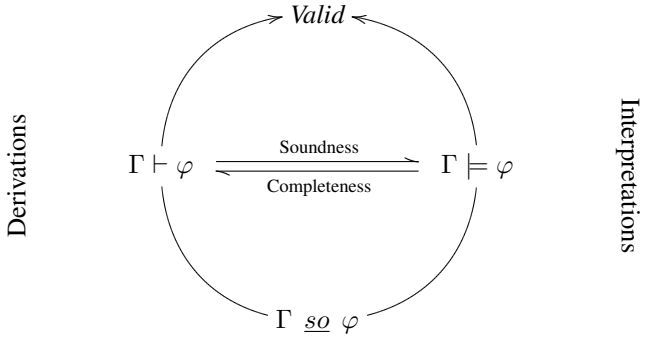
It is interesting to look at the shape of the derivation that proves the simple property we used as motivation. Ignoring the arithmetic into some vertical dots, this is what it

looks like:

$$\dfrac{\dfrac{\exists x.\, x^2 = -1 \qquad \dfrac{\dfrac{\dfrac{}{i^2 = -1}\,(i)}{\vdots\ (j=2i)}}{j^2 + 4 = 0}\,\exists_{\mathsf{I}}}{\exists y.\, y^2 + 4 = 0}\,\exists_{\mathsf{E}}}{\exists x.\, x^2 = -1 \to \exists y.\, y^2 + 4 = 0}\,\to_{\mathsf{I}}$$

7.3 Soundness and Completeness

As seen already in Chapter 4, the truth-based method and the derivation-based approach are two different ways to define what it means for an inference to be valid, and logicians prove soundness and completeness theorems to show that they are actually the same. Such theorems actually hold for predicate logic too, which assures us that we have given consistent definitions of validity. The same picture we saw in Chapter 4 applies also for first-order logic:



In words, the soundness theorem says that every inference that is shown valid using derivations is always valid when using the truth-based method.

Theorem 7.1 (Soundness) *If* $\Gamma \vdash \varphi$*, then* $\Gamma \models \varphi$.

The completeness theorem says that it also works the other way around: truth-based validity implies derivability.

Theorem 7.2 (Completeness) *If* $\Gamma \models \varphi$*, then* $\Gamma \vdash \varphi$.

The completeness theorem was first proved by the logician Kurt Gödel (1906–1978). This was quite an achievement at the time.

7.4 Is my Inference Valid?

So, do we have a method for determining whether an inference in first-order logic is valid or not? Let's see. Say this inference has premises $\Gamma = \varphi_1, \ldots, \varphi_n$ and conclusion φ. Here is what we know:

- To show that this inference is valid using the derivation method, i.e., that $\Gamma \vdash \varphi$, all we need to do is to come up with a derivation of φ from Γ. A *single* derivation is enough — that's promising!

  However, the derivation method is of little use to show that an inference is *not* valid. In fact, based on what we know, it is not obvious at all to convince oneself that no derivation exists for an inference.

- We know from Chapter 6 that we can use the truth-based method to show that $\Gamma \nvDash \varphi$, i.e., that our inference is *not* valid. We do so by producing an interpretation where each of the premises $\varphi_1, \ldots, \varphi_n$ in Γ is true, but the conclusion φ is false. A single interpretation will do — that's promising too!

  On the other hand, Chapter 6 gives us little hope that we can use this method in practice to show that an inference *is* valid: we would have to consider all possible interpretations, and there are infinitely many of them. Definitely not practical.

But maybe we can use them together to establish validity... Here's an idea:

- Try building a derivation for $\Gamma \vdash \varphi$.

- At the same time, try finding an interpretation $\mathcal{M}$ that shows that $\Gamma \nvDash \varphi$.

The first that succeeds determines the outcome: if we find a derivation for $\Gamma \vdash \varphi$ then the inference is valid. If we find an interpretation $\mathcal{M}$ such that $\mathcal{M} \models \varphi_i$ for all φ_i in Γ but $\mathcal{M} \nvDash \varphi$, we know it is not valid. This all hinges on soundness and completeness ensuring that $\Gamma \vdash \varphi$ iff $\Gamma \models \varphi$.

Should we declare victory?

Unfortunately, life is not this simple. The problem is with the second part of our idea. How do we find the interpretation $\mathcal{M}$, among all the infinitely many interpretations that are out there? We were able to do so for a small example using our ingenuity, but how to do this in general? There is no mechanical method to do so.

In fact, while it is possible in the propositional case to write a program that always tells us whether an inference is valid or not, no such program can be written for inferences in predicate logic. It is not a matter of finding a sufficiently smart programmer: no such program can possibly exist!

The failure of both the truth- and derivation-based methods to always tell whether a generic inference is valid actually touches on a deep property of predicate logic. No method will ever be able to determine whether an arbitrary inference is valid or not. Properties of this type are called *undecidability results*.

7.5 Theorem Proving

And yet, there are plenty of people writing programs to determine whether $\Gamma \vdash \varphi$ given premises Γ and conclusion φ. Why? Isn't it impossible to write such a program? Undecidability says that it is impossible to write a program that will *always* give us an answer for *all* Γ and φ. That leaves plenty of room to maneuver.

We can give up on the expectation that we will *always* get a result. One easy way to do so is to systematically apply derivations rules to Γ and φ. If a derivation exists (and we are really systematic), we will eventually find it. Otherwise we may end up trying rules for ever.

We can also restrict the kind of formulas that we allow. For example, if Γ and φ do not contain any free or bound variables (and therefore no quantifiers either), then we can build truth tables for the atomic formulas in them. We can then use the method seen in Chapter 3 to determine validity — always.

These programs are called *theorem provers*. Although they can establish validity only partially, they are useful for all kinds of purposes, from helping mathematicians prove theorems (that's why they are called theorem provers) to verifying that programs and systems work as expected. They are starting to be used in industrial settings too.

Theorem provers operate in all kind of different ways: some work completely on their own (they are called *automated* theorem provers) while others rely on interaction with users to guide them through the proof process (they are called *proof assistants*). How they establish validity also differs greatly. Some try to build derivations (but typically in a much smarter way than what we have seen), some make use of sophisticated forms of model checking, some combine the two, and some rely on completely different methods. It's a wild exciting world out there!

7.6 *Exercises*

1. Show that the following inferences are valid by giving a derivation:
   (a) $\vdash \forall x.\, (\forall y.\, p(x, y)) \to p(x, x)$
   (b) $\forall x.\, p(x, x), \forall y.\, \forall z.\, p(z, y) \to q(y, z) \vdash \forall w.\, q(w, w)$
   (c) $\exists x.\, p(x) \land q(x) \vdash \exists x.\, q(x)$
   (d) $\exists x.\, \forall y.\, p(y) \vdash \forall y.\, p(y)$
   (e) $\forall x.\, \exists y.\, p(x) \to q(y), \forall y.\, \exists z.\, q(y) \to r(z) \vdash \forall x.\, \exists z.\, p(x) \to r(z)$
2. Using derivations, show that the following inferences are valid.
   (a) $\exists x.\, \varphi(x), \forall y.\, \varphi(y) \to \psi(y) \vdash \exists z.\, \psi(z)$
   (b) $\exists x.\, \varphi(x) \to \psi(x) \vdash \forall x.\, \varphi(x) \to \exists x.\, \psi(x)$
   (c) $\neg\forall x.\, \varphi(x) \vdash \exists x.\, \neg\varphi(x)$
   (d) $\vdash \forall x.\, \forall y.\, \varphi(x, y) \to \forall y.\, \forall x.\, \varphi(x, y)$
   (e) $\vdash \exists x.\, \exists y.\, \varphi(x, y) \to \exists y.\, \exists x.\, \varphi(x, y)$

where φ and ψ are generic formulas that may use the variable shown in parentheses.

3. Differently from the truth-based method, we can build a derivation for an inference even if it mentions (unbound) variables. Check it out for yourself by giving a derivation for $p(x), p(x) \to q(x) \vdash q(x)$. Knowing this, show that whenever there is a derivation for a generic inference

$$\varphi_1(x), \ldots, \varphi_n(x) \vdash \varphi(x)$$

where each formula may contain variable x (and possibly others), then there is a derivation of the inference

$$\forall x.\, \varphi_1(x), \ldots, \forall x.\, \varphi_n(x) \vdash \forall x.\, \varphi(x)$$

4. Using any of the methods seen in this and last chapter, determine whether the following inference is valid: $p(c)$, $\exists x.\, p(x) \to q(x) \vdash q(c)$.

5. Here's a sure-fire idea for showing that an inference $\Gamma \vdash \varphi$ is valid using just the derivation method:

> *Try to build a derivation both for* $\Gamma \vdash \varphi$ *and for* $\Gamma \vdash \neg\varphi$*. If the former succeeds, the inference is valid. If the latter succeeds, it is not valid because then* $\Gamma \nvdash \varphi$.

There is a flaw in this argument. Can you find it?

Once you have done so, express this argument as an inference (yes, that will be an inference about inferences) and show that it is invalid. [Hint: Exercise 5 from Chapter 4 may turn useful.]

6. When it comes to writing a mechanical procedure that can always tell whether an inference is valid or not, the trouble is with the quantifiers, and not even all of them. Indeed, if we ban the existential quantifier and allow the universal quantifier to appear only on the outside of any connective, then determining whether an inference is valid becomes decidable. By applying rules in a systematic way we can always find a derivation if there is one. Where this restriction becomes helpful is into building an interpretation where the inference does not hold. Try it out for yourself. Consider the following inference

$$p(a) \text{ and } \forall x.\, p(x) \to \neg q(x), \text{ so } q(b)$$

Find an interpretation that shows that it is not valid.

Chapter 8

Function Symbols

Predicate logic, as we defined it in the last three chapters, is very expressive. Can we extend it to capture even more linguistic patterns from everyday speech? You bet! and logicians all over the world are busy doing precisely that. While there are plenty of extensions worth looking into, we will focus on just one, one that happens to be part of the standard definition of first-order logic. This will be enough of a workout. After all, last time we extended logic (going from propositional to first-order) we made things pretty complicated, didn't we?

We will augment the terms of first-order logic with *function symbols*. Up to now, the only way we had to refer to an object was by giving it a name. This is fine, but there are plenty of things we don't give names to: we indicate them relative to other things. Consider for example the phrase "the front cover of the book": if I give you a book, you know exactly what I mean by its front cover, even though that part of this specific book does not have an official name. Function symbols allow us to express indirect references such as "the front cover of".

8.1 Indirect References

Let's go back to Sidra's love for logic, and maybe learn about where it comes from. Consider the following statement

Sidra's father loves logic.

How do we express it in first-order logic?

Sidra knows the name of her father for sure, but chances are that you don't — I am myself at loss. One thing we can certainly do is make up a name for him, for example $SidrasDad$. Then, we would write this statement as the following first-order logic formula:

$$LogicLover(SidrasDad).$$

This would do, but what about the following sentence, which states that passion for logic is inherited from one's father:

Everybody whose father loves logic also loves logic.

To express this, we need to be able to refer to the father of a generic person. We cannot do this conveniently in the language of first-order logic, at least as of now.[1]

Now, we don't know the name of Sidra's father, but we can certainly talk about him relative to Sidra by referring to him as "Sidra's father". Here, "father" is kind of a function that identifies one person based on another one. What a great insight for extending logic to capture more linguistic patterns! What we need is a way to express the phrase "*the father of* $\ldots$". We do so by extending first-order logic with *function symbols*. A function symbol is like a name, except that we can give it arguments that correspond to whatever we replace "$\ldots$" with in an actual phrase like "the father of Sidra". We used the same device in Chapter 5 when we introduced predicate symbols. Let's pick the function symbol $father$ with a single argument, so that

$$father(x) \qquad \text{stands for} \qquad \text{"}\textit{the father of } x\text{"}.$$

Then, if we plug an actual name in there, we get a way to refer to that person's father. For example $father(Sidra)$ becomes our way to talk about Sidra's father, although we don't know his name. Schematically,

$$\begin{array}{ccccc} \textit{"the father of } \ldots\textit{"} & + & \textit{"Sidra"} & = & \textit{"the father of Sidra"} \\ \updownarrow & & \updownarrow & & \updownarrow \\ father(\ldots) & + & Sidra & = & father(Sidra) \end{array}$$

With $father$ as a function symbol, we do not need to have a name for Sidra's father: by writing $father(Sidra)$ we are able to refer to him indirectly relative to Sidra. Just as we used the name $Sidra$ to refer to the person Sidra, we can use the *function symbol* $father$ to refer to somebody's father.

Our statement about the love of logic being inherited can now be expressed by the formula

$$\forall x.\, LogicLover(father(x)) \rightarrow LogicLover(x).$$

Here, $father$ is just a symbol. It allows us to represent fatherhood, but nothing forces us to read it that way. In fact, nobody prevents us from using $father(x)$ to mean "*the smallest prime larger than* x". Conversely, we can use other symbols to express this statement. For example, if $p(x)$ is "x loves logic" and $f(x)$ is "x's father", then our sentence becomes $\forall x.\, p(f(x)) \rightarrow p(x)$. Remember? We are in the business of *formal* and *symbolic* logic: we want to determine the validity of an inference on the basis of its form, not the meaning we assign to the symbols in it.

Since for us $father(x)$ refers to the father of x, we can take x to be $father(Sidra)$ to refer to her paternal grandfather as

$$father(father(Sidra)).$$

We can repeat this as many times as we want, obtaining a term that allows us to refer to the paternal grand grand grand $\ldots$ grandfather of Sidra. At some point, not even Sidra will know the name of that person: we have a way to refer to an entity without having a name for it!

[1] There are ways to do it, but they are convoluted. We will touch on them a bit later in this chapter.

8.2 Using Function Symbols

We called $father$ a *function* symbol, which is fitting because everyone has exactly one father: the relation between a child and his/her father is a function (relative to the child), in fact a *total* function. Now, two questions arise. What about the cases where we are not dealing with functions but with generic relations? And what if we have a function, but it is not total? Let's look into that.

8.2.1 When Relations are not Functions

Consider sisterhood. What if I want to express the sentence

Anyone whose sister loves logic also loves logic.

Can we use a function symbol $sister(x)$ to denote a sister of x? If we could, then the above sentence would be translated into the formula

$$\forall x.\, LogicLover(sister(x)) \rightarrow LogicLover(x).$$

The problem is that what we mean is not clear: do we mean that if *any* sister of x loves logic then x loves logic? Or maybe *all* sisters of x must love logic to guarantee that x loves it too?

First-order logic stays clear of these complications. It allows us to use function symbols in a term only to express relationships that are really functions, when the indirect reference identifies a unique entity. So, if we use the function symbol $sister$ to represent sisterhood, then $sister(x)$ shall stand for *the* sister of x (implying that x has exactly one sister), not for *a* sister of x (the more general case where x may have any number of sisters). Note that the article we use, whether "the" or "a", is a good indication as to whether we can use a function symbol to express an indirect reference: "the" implies that the relationship is functional, "a" that it is not.

Now, how do we deal with sisters loving logic? One way to do so is use a *predicate* $sister(x, y)$ to stand for "x *is a sister of* y". As a predicate, it is true of false, not an indirect way to refer to somebody. Then we can write the two meanings of the above sentence as

$$\forall x.\, \exists y.\, sister(x, y) \wedge LogicLover(x) \rightarrow LogicLover(y)$$
$$\forall x.\, \forall y.\, sister(x, y) \wedge LogicLover(x) \rightarrow LogicLover(y)$$

respectively. By using predicates, we are forced to be explicit about the quantification. Notice however that this forces us to have a name for every sister, something function symbols make unnecessary when they can be used.

8.2.2 Partial Functions

Consider an indirect reference such as "the father-in-law of x". Married people have fathers-in-law, but unmarried people do not. So, father-in-law-hood is a *partial* function. We can still use function symbols in these situations, for example $fatherInLaw(x)$

in this case. We have to be careful not to write nonsense, though. One way to do so is to guard our formulas so that we will only refer to the father-in-law of married people. For example, if we have a predicate $married(x)$ that we understand as "x is married", then the phrase "everybody whose father-in-law loves logic also loves logic" is expressed as

$$\forall x.\, married(x) \wedge LogicLover(fatherInLaw(x)) \rightarrow LogicLover(x).$$

The other way is to make sure that a formula that mentions $fatherInLaw$ will never be true for unmarried people. For example,

$$\forall x.\, LogicLover(fatherInLaw(x)) \rightarrow LogicLover(x)$$

would work if all the people we are dealing with are married.

While we are talking about family relations, we can use a function symbol with two arguments to denote the first child of two people: $firstBorn(x, y)$ denote "the first-born child of x and y". Now, x and y may not have children — in fact they may have never met — so this is another partial function.

8.3 The Language of First-Order Logic

What we dealt with until this chapter was a simplified version of the language of predicate logic. First-order logic, as commonly used by mathematicians and computer scientists the world around, comes equipped with function symbols. Let's therefore update the definition from Chapter 5 to include them. All we need to do, really, is to spruce up the definition of what a *term* is:

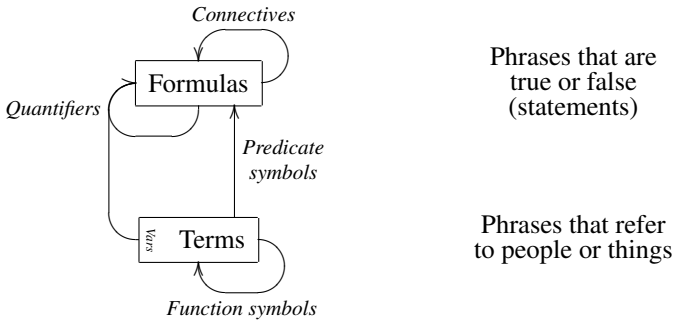
- A variable is a term.
- A name is a term.
- If $t_1, \ldots, t_n$ are terms and f is a function symbol that takes n arguments, then $f(t_1, \ldots, t_n)$ is a term.

Nothing else is a term. We continue writing x, y and z for variables and a, b, c, etc, for generic names (which we also call constants). We will use the letters f, g and h to denote generic function symbols. The number n of arguments that a function symbol f takes is called its *arity* — just like for predicate symbols.

The last line is what we added with respect to Chapter 5. It says that we can build a term by applying a function symbol f that takes n arguments to n terms $t_1, \ldots, t_n$, obtaining $f(t_1, \ldots, t_n)$. Although the t_i's can themselves contain function symbols, such subterms will eventually have to be just a variable, x say, or just a constant, a for example.

The rest of the infrastructure of first-order logic remains exactly the same as what we saw in Chapter 5: an atomic formula is still a predicate symbol applied to terms (in number equal to its arity), and composite formulas are still constructed using the same connectives and quantifiers.

Altogether, the way these ingredients are combined to form the full language of first-order logic is described in the following diagram.



The main change with respect to Chapter 5 is that we have a richer language of terms. Terms still enter formulas as the arguments of predicate symbols, and formulas are still build on top of atomic propositions using the Boolean connectives and the quantifiers.

Function symbols open the doors to all kinds of new inferences. Consider the following:

$$\begin{array}{ll} & \mathit{LogicLover}(\mathit{father}(\mathit{father}(\mathit{Sidra}))) \\ \underline{\textit{and}} & \forall x.\, \mathit{LogicLover}(\mathit{father}(x)) \rightarrow \mathit{LogicLover}(x) \\ \underline{\textit{so}} & \mathit{logicLover}(\mathit{Sidra}) \end{array}$$

It says that assuming Sidra's paternal grandfather loves logic and that the love logic is inherited from one's father, then it must be the case that Sidra loves logic too. It is a bit more complicated than previous inferences, but it makes sense: intuitively, it is valid.

To make things shorter and to emphasize that father and $\mathit{LogicLover}$ are just symbols, we will replace them with f and p, respectively (we will keep the name Sidra however). We get

$$\begin{array}{ll} & p(f(f(\mathit{Sidra}))) \\ \underline{\textit{and}} & \forall x.\, p(f(x)) \rightarrow p(x) \\ \underline{\textit{so}} & p(\mathit{Sidra}) \end{array}$$

Notice that this is the exact same inference: we have just changed some symbols. Intuition does not help much any more however. We will be using this inference as our work horse through the rest of the chapter.

How do we show that it is actually valid? The usual suspects will come to the rescue: interpretations and derivations.

8.4 Interpretations

Recall that an interpretation $\mathcal{M}$ has (so far) three components: a domain of discourse $\mathbb{D}$ that lists all the individuals we are talking about, an interpretation of names that tells us

which individual in $\mathbb{D}$ each name refers to, and an interpretation of predicate symbols that tells us whether a predicate is true or false on each individual (or combination of individuals) in $\mathbb{D}$.

Now, all we need to add is an *interpretation of function symbols* which tells us which individual we are referring to when using a function symbol on terms that represent some individuals in $\mathbb{D}$. This new component associates to each function symbol, such as $father(x)$, an actual function in the domain of discourse.

Let's make things concrete and define a small interpretation for our example. Here's our domain of discourse:

$\mathbb{D} = \{$  ,  ,  , ? $\}$

We will get back to ? in just a minute.

Let's say that the only two names we care about are $Sidra$ and $Hanan$. Here's an interpretation for these names:

Name	$\mathbb{D}$
$Sidra$	
$Hanan$	

Note in particular that we do not have names associated with either  or ?.

The interpretation of the function symbol f could then be given by the following table:

$x \in \mathbb{D}$	$f(x)$
	
	?
	?
?	?

This interpretation of f says that  is the father of  (which is associated with $Sidra$ according to our interpretation of names). We use ? to stand for the father of the people we don't know who their father is. Here, we don't know who Hanan's father is, nor who the father of Sidra's father is. We also don't know who the father of the unknown person is. Note that the interpretation of f is a total function over the domain of discourse: ? allows us to deal with incomplete knowledge, or in general with genuinely partial functions.

To complete $\mathcal{M}$, all we need is an interpretation for the predicate symbol p, which we do through the following table, for example:

$x \in \mathbb{D}$	$p(x)$
	T
	F
	T
?	F

At this point, we can extend this interpretation to all the formulas that appear in our inference. We get the following table:

$x \in \mathbb{D}$	$p(x)$	$p(f(x))$	$p(f(f(x)))$	$p(f(x)) \to p(x)$
	T	T	F	T
	F	F	F	T
	T	F	F	F
?	F	F	F	T

Because the last column contains an F, this table tells us that, in our example interpretation, the formula

$$\forall x.\, p(f(x)) \to p(x)$$

is false. The first premise, $p(f(f(Sidra)))$ is also false in this interpretation, while the conclusion, $p(Sidra)$ is true. Therefore, our inference is valid *with respect to this interpretation*.

However, this does not tell us that the inference is valid in general. For this we would have to check all possible interpretations, including those with infinite domains of discourse. Just like in Chapter 6, before we had function symbols in logic, it would take for ever to check that a universal formula is true or that an existential formula is false. On the other hand, a single interpretation is sufficient to find counterexamples to a universal formula (if it is false) or a witness to a universal formula (if it is true).

In summary, given premises $\Gamma = \varphi_1, \ldots, \varphi_n$ and conclusion φ, using the definition of valid inference ($\Gamma \models \varphi$ if and only if φ is true in all the — infinitely many — interpretations where each formula in Γ is also true) remains highly impractical.

8.5 Derivations

So, what kind of complicated extension will we need to inflict on derivations to incorporate function symbols?

The surprising answer is ... *none*!

The two elementary rules that dealt with terms in Chapter 7 are $\forall_{\mathsf{E}}$ and $\exists_{\mathsf{I}}$:

$$\frac{\forall x.\, \varphi(x)}{\varphi(t)}\,\forall_{\mathsf{E}} \qquad\qquad \frac{\varphi(t)}{\exists x.\, \varphi(x)}\,\exists_{\mathsf{I}}$$

They do not care about what kind of term we pass to them.

Can we then write a derivation for the inference

$$p(f(f(Sidra))), \forall x.\, p(f(x)) \to p(x) \vdash p(Sidra)$$

? We can! Here it is:

$$\dfrac{p(f(f(Sidra))) \quad \dfrac{\dfrac{\forall x.\, p(f(x)) \to p(x)}{p(f(f(Sidra))) \to p(f(Sidra))}\forall_{\mathsf{E}} \qquad p(f(f(Sidra)))}{p(f(Sidra))}\to_{\mathsf{E}} \quad \dfrac{\dfrac{\forall x.\, p(f(x)) \to p(x)}{p(f(Sidra)) \to p(Sidra)}\forall_{\mathsf{E}} \qquad p(f(Sidra))}{p(Sidra)}\to_{\mathsf{E}}$$

In the leftmost use of $\forall_{\mathsf{E}}$, we instantiated the variable x with the term $f(Sidra)$. Instead, in the rightmost use of $\forall_{\mathsf{E}}$, we instantiated this variable with $Sidra$. Therefore, this inference *is* valid!

Not bad, eh!

8.6 Soundness and Completeness

The two roads to defining a valid inference lead once more to the same notion. This is witnessed through the familiar soundness and completeness theorems.

Theorem 8.1 (Soundness) *If* $\Gamma \vdash \varphi$, *then* $\Gamma \models \varphi$.

Theorem 8.2 (Completeness) *If* $\Gamma \models \varphi$, *then* $\Gamma \vdash \varphi$.

This is the actual completeness theorem that Kurt Gödel (1906–1978) proved. Logicians were using function symbols even back then.

Function symbols do not make determining whether an inference is valid any simpler. In fact, they complicate things further by introducing new avenues by which undecidability can sneak in.

8.7 *Exercises*

1. Consider the following phrases. Identify which ones can and which ones cannot be expressed using function symbols in logic. Explain why.

   (a) "page of book b"

   (b) "preface of book b"

   (c) "n-th chapter of book b"

   (d) "author of book b"

   (e) "publisher of book b"

   (f) "n-th edition of the book by author a"

   (g) "number of copies of book b sold"

   (h) "first book of author a"

(i) "book of a who sold a million copies"

(j) "n-th chapter of book b by author a"

For those that cannot be expressed using a function symbol, propose a similar phrase that can.

2. Consider the following names, function symbols and predicate symbols with the given readings:

$$\begin{array}{rcl} \mathit{Nile} & = & \text{"the Nile river"} \\ \mathit{Ganges} & = & \text{"the Ganges river"} \\ \mathit{Mouth}(x) & = & \text{"the mouth of river } x\text{"} \\ \mathit{Source}(x) & = & \text{"the source of river } x\text{"} \\ \mathit{North}(x, y) & = & \text{"}x \text{ is to the north of } y\text{"} \end{array}$$

For each of the following expressions, determine whether it is a well-formed first-order formula:

(a) $\forall x.\, \mathit{Mouth}(x) \rightarrow \mathit{North}(\mathit{Source}(x))$

(b) $\neg\exists x.\, \neg\mathit{Mouth}(x) \lor \neg\mathit{Source}(x)$

(c) $\exists x.\, \mathit{North}(\mathit{Mouth}(x), \mathit{Source}(x))$

(d) $\forall x.\, \mathit{Mouth}(\mathit{Source}(x)) \rightarrow \neg x$

(e) $\forall x.\, \mathit{Source}(x) \rightarrow \mathit{Mouth}(x)$

(f) $\exists x.\, \mathit{North}(x, \mathit{Mouth}(x))$

(g) $\forall x.\, \exists y.\, \mathit{North}(\mathit{Source}(x), \mathit{Mouth}(y))$

3. Define an interpretation for the inference in Section 8.4 that makes all the premises true. Is there a "smallest" such interpretation? What does it look like?

4. Find an interpretation where the following inference does not hold:

$$\forall x.\, p(f(x, x), x) \vdash p(f(1, 0), 0)$$

5. Function symbols are often used to represent mathematical operations like addition and functions like squaring. We will explore how to do so for infinite sets like the natural numbers in Chapter 10. For now, we will limit ourselves to operations modulo 3 in the finite set $\mathbb{Z}_3 = \{0, 1, 2\}$. Using the standard interpretation of the operations $x + y$ (addition), $x - y$ (subtraction), $x * y$ (multiplication), of the function x^2 (squaring) and of the predicate $x = y$ (equality) — all modulo 3, determine whether the following inferences are valid and if not give a counterexample.

(a) $\models \forall x.\, x * x = x^2$

(b) $\models \forall x.\, \exists y.\, x = y^2$

(c) $\models \forall x.\, \exists y.\, x * y = 1$

(d) $\models \forall x.\, \forall y.\, \forall z.\, x + y = z \rightarrow x - z = y$

(e) $\exists x.\, \forall y.\, x * y = 2 \models \forall x.\, \exists y.\, x + y = 0$

6. Give a derivation of the following inference:

$$\begin{array}{rr} \forall x.\, \forall z. & p(x, f(x, z)) \\ \forall x.\, \forall y.\, \forall z.\, p(x, z) \rightarrow & p(x, f(y, z)) \end{array} \vdash p(2, f(1, f(2, f(3, end))))$$

7. Constants, or names as we called them, are often viewed as function symbols that take zero arguments. That is, the name a is viewed as the term $a()$ where we are passing no arguments to the function symbol a. Modify the definition of term to account for this view. How do interpretations change if you take this perspective? What about derivations?

8. Are function symbols really necessary? In Section 8.2 we saw that we can describe relations like "x is a sister of y" without the help of function symbols. But, as we learned in our algebra classes, functions are special cases of relations, aren't they? Could we use predicates to express functional references too? Let's try it out:

   (a) By taking inspiration to our treatment of sisterhood in Section 8.2.1, express the inference

   > *Everybody whose father loves logic also loves logic and Sidra's father loves logic, so Sidra loves logic*

   in first-order logic without using function symbols or a name for Sidra's father. Be careful about how you translate the second premise.

   (b) Based on your knowledge of derivations, explain how using a predicate symbol instead of a function symbol achieves our goal in this example. [Hint: somehow a unique name gets assigned to Sidra's father.]

   (c) What happens if you use this approach on a partial function, like $fatherInLaw$?

   (d) For something a bit more challenging, give a general recipe to translate a formula with function symbols into a formula without.

Part III

Beyond the Basics

Chapter 9

Equality

One form of inference we engage in very frequently is about things being *equal*. We do so when talking about people, for example "the first man to walk on the moon" and "Neil Armstrong" refer to the same person. We do so also when checking that we got the right change from a cashier: for example if we pay for an item that costs $7.83 with a $10 bill, we expect the change we get back to be equal to $2.17. Even finding your car in the parking lot has something to do with equality. So, equality seems to be one of those recurrent linguistic patterns worth being able to reason about within logic.

9.1 When are Two Things the Same?

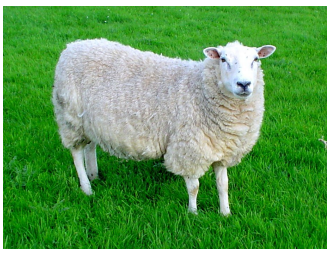
Now, what do we mean by "equal"?

Consider these two sentences:

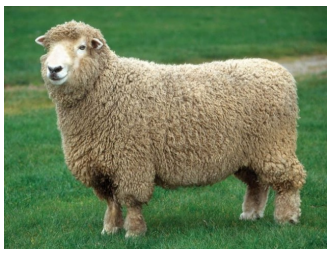
"Sidra loves logic" *and* "Sidra loves logic"

Are they the same? Well, it depend. Certainly they consist of the exact same sequence of characters. However, if you look very closely, possibly with a magnifying lens, you will notice that they are made of slightly different ink patterns on the page. But who reads books on paper nowadays? Well, if you are reading this on a computer of some kind, you cannot avoid noticing that they occupy different positions on your screen. Are they really the same? Hmm ...

Consider the following two pictures of sheep on a field:



and



Are they the same? Chances are that you will answer "no". But they depict two sheep in the same way as "Sidra loves logic" and "Sidra loves logic" are two individual sentences. So, yes they identify the same thing, if we mean the abstract idea of "sheep", but no they do not depict the same specific animal.

Next, consider the last time you looked at yourself in the mirror (maybe this morning) and the time before that (yesterday?). Did you see the same person? You would hope so! But, if you think of it, the time was different, the light was probably not the same, a few of your cell had been replaced. So, is it really the same person? This is getting philosophical.

But maybe people and sheep are too complicated to deal with. Then, consider the following two C functions:

```
int inc (x:int) {
    x+1;
}
```

and

```
int inc (y:int) {
    y+1;
}
```

Are they the same? They look different on the page, but arguably they will return the same output for every input. So they are the same ... but earlier we argued that the two copies of "Sidra loves logic" were the same because they were the exact same sequence of characters while these two programs are not the same sequence of characters ... so are they different?

This is all complicated! Let's turn to logic to make sense of it all.

9.2 First-order Equality

In first-order logic, as we defined it so far, when is it that two things are equal? Consider the inference $p(a), p(a) \to q(a) \vdash q(a)$, which is shown valid by a single use of rule $\to_{\mathsf{E}}$. To apply this rule, we recognize that the first premise, $p(a)$, and the antecedent of the second premise $p(a) \to q(a)$, also $p(a)$, are the same proposition. First-order logic views two atomic formulas as equal when they have the same predicate symbol and it is applied to the same terms.[1] This is a very crude notion of equality, which falls short of capturing the nuances we witnessed in the last section.

First-order logic provides one way to represent individual entities, and that is by using terms (possibly with function symbols if we want to refer to them indirectly). Therefore, equality between entities amounts to determining when two terms are equal. All we need to do this is to provide a single binary predicate symbol to represent equality. Following what we are used to in mathematics, we will denote it $=$, written infix. Therefore, $t_1 = t_2$ is an atomic proposition that we understand as asking whether (or specifying that) terms t_1 and t_2 are equal.

[1] Equality among generic formulas is a bit more complicated because quantified variables can be renamed without affecting the meaning of the formula.

By itself, this is just a notation. It does not say anything about when two terms are equal: we have *expressed* equality, but not *defined* it. Before we do so, let's look at an inference that uses equality:

> *Sidra's father is called Ali, Ali loves logic, and everybody whose father loves logic also loves logic, so Sidra loves logic.*

This is an inference, and it makes use of the fact that Ali and Sidra's father are the same person. Let's write it in first-order logic:

$$\begin{array}{ll} & f(Sidra) = Ali \\ \underline{and} & p(Ali) \\ \underline{and} & \forall x.\, p(f(x)) \rightarrow p(x) \\ \underline{so} & p(Sidra) \end{array}$$

Here, the first premise, $f(Sidra) = Ali$, tells us that we consider the term $f(Sidra)$ and the term Ali to be equal. Recall that we are writing $f(x)$ for "*the father of* x".

Up to now, $x = y$ is just a binary predicate, no different from $q(x, y)$: it does not have any meaning from the point of view of logic. Therefore, if we just use the definitions seen up to last chapter, this inference cannot be shown to be valid. Yet, it is intuitively valid if we think about "=" as equality.

The way we will be able to show that this inference, and others like it, are valid is by refining the notions of interpretation and derivation, so that "=" has a *special meaning*. A predicate symbol whose meaning is given by the infrastructure used to define validity is called *interpreted*. Therefore, "=" will be an interpreted predicate symbol. All predicates we have seen so far were *uninterpreted* because interpretations and derivations did not treat them specially.

So, what will $t_1 = t_2$ mean for us? It will hold, or be true, exactly when t_1 and t_2 can be shown to be equal ... that sounds like a circular definition! We will specify what things we want to consider equal by means of premises that contain "=" — that's how we deal with sheep and mirrors, and the other nuances we saw in the previous section. Using this information, we will be able to infer new equalities (ah! Aristotle again: *logic is new and necessary reasoning*) and also to draw conclusions that leverage term equality.

Now, the usual question: how to we check that an inference that uses equality is valid? This begets the usual answer: either by means of interpretations or of derivations.

9.3 Interpreting Equality

What assigns a meaning to predicates in an interpretation is the component that we called the interpretation of predicates. So, all we need to do is define the meaning of "=" since it is a predicate symbol. Now, given that an interpretation $\mathcal{M}$ associates every term in a first-order formula to an element of the domain of discourse, and that the interpretation of predicates tells us when each predicate is true of its elements, it is

natural to require that "=" be true exactly when its two arguments are the same element of the domain of discourse.

Let's see how this works with respect to the example interpretation we saw in the last chapter. Our domain of discourse was

$\mathbb{D} = \{$  ,  ,  , ? $\}$

We interpreted names and our one function symbol f as

Name	$\mathbb{D}$
Sidra	
Hanan	
Ali	

$x \in \mathbb{D}$	$f(x)$
	
	?
	?
?	?

We just added an entry for the name Ali, which we didn't have in the last chapter. Back then we had a single predicate symbol to interpret, p, and we did so as follows:

$x \in \mathbb{D}$	$p(x)$
	T
	F
	T
?	F

We shall now extend the interpretation of predicates with the following table for $x = y$, the *interpretation of equality* in $\mathbb{D}$:

		$y \in \mathbb{D}$			
	$=$				?
$x \in \mathbb{D}$		T	F	F	F
		F	T	F	F
		F	F	T	F
	?	F	F	F	T

The one difference between the interpretation of the predicate p and the interpretation of "=" is that the latter is *fixed* while the former could be arbitrary. Specifically, given any domain of discourse $\mathbb{D}$, the interpretation of $t_1 = t_2$ in $\mathbb{D}$ will always be true exactly when t_1 and t_2 correspond to the same elements of $\mathbb{D}$, and false when they are different: because "=" is binary, its interpretation will be a two-dimensional matrix; whatever the domain of discourse, its diagonal will always contain T while all other entries will have F in them. It is in this respect that "=" is an interpreted symbol: it has a fixed meaning.

This is all we need to add to our definition of first-order interpretation — very little compared to what we had to do for function symbols, for example.

Now, with this, we can check the validity of our example inference in the above interpretation $\mathcal{M}$. Here is how the reasoning goes:

- The premise $f(Sidra) = Ali$ is true because the name $Sidra$ is interpreted to  and the interpretation of the function symbol f maps it to  . Now, the name Ali also corresponds to  . The two sides of "=" are therefore the same, so the table for "=" says that this premise is true.
- The premise $p(Ali)$ is true according to the interpretation of predicate symbol p on the individual  , which is the interpretation of the name Ali.
- The premise $\forall x.\, p(f(x)) \to p(x)$ is false, for the reasons seen in the last chapter.
- The conclusion $p(Sidra)$ is true since the interpretation of p on  is true.

Altogether, the inference is valid in $\mathcal{M}$ because it is indeed the case that if all premises are true (they are not) then the conclusion is also true.

Again, it holds in $\mathcal{M}$, but to ensure that it is valid, we would have to examine all infinite possible interpretations, which is not directly feasible. As we will see shortly, this particular inference *is* valid. Therefore,

$$\begin{array}{r} f(Sidra) = Ali \\ p(Ali) \\ \forall x.\, p(f(x)) \to p(x) \end{array} \models\ p(Sidra)$$

9.4 Derivations

To use derivations, we need to give elimination rule(s) to describe what we can infer knowing that two terms are equal, and introduction rule(s) to tell us when we can establish that two terms are equal.

The elimination rule is actually pretty easy to define. How can we use the knowledge that $t_1 = t_2$? Well, if I have a formula that mentions t_2, then the formula obtained by replacing some of the occurrences of t_2 with t_1 should also be a derivable formula. This suggests the following elimination rule:

$$\frac{t_1 = t_2 \quad \varphi(t_2)}{\varphi(t_1)} =_{\mathsf{E}}$$

This rule is sufficient to show that our example inference in this chapter is derivable. Here's a derivation for it:

$$\dfrac{\dfrac{f(Sidra) = Ali \quad p(Ali)}{p(f(Sidra))} =_{\mathsf{E}} \quad \dfrac{\forall x.\, p(f(x)) \to p(x)}{p(f(Sidra)) \to p(Sidra)} \forall_{\mathsf{I}}}{p(Sidra)} \to_{\mathsf{E}}$$

This shows that this inference is valid.

Note that in rule $=_{\mathsf{E}}$ we do not need to replace *all* occurrences of t_2 with t_1: only the ones we choose to. Only in this way can we build a derivation for $c = d, p(c, c) \vdash p(c, d)$, which may represent the inference "Ali is Sidra's father and Ali sees himself in the mirror, so Ali sees Sidra's father in the mirror".

Now, how do we establish that two terms are equal? Well, it should always be the case that $Sidra = Sidra$. In fact, this should hold for every term. This suggests the rule

$$\frac{}{t = t}\,\mathsf{refl}$$

This is one of the properties of equality we learn in school. It is called *reflexivity*. Equality is a reflexive relation.

Are there other circumstances where two terms are equal? Definitely. Say that we know that $Ali = f(Sidra)$. We should be able to infer that $f(Sidra) = Ali$, shouldn't we? Can we do so on the basis of the rules we have so far? Not quite. This suggests adding another of the properties we learn in school: *symmetry*. Symmetry says that we can swap the order of two terms in an equality. It is defined by the following rule:

$$\frac{t_2 = t_1}{t_1 = t_2}\,\mathsf{sym}$$

It says that one way to learn that $t_1 = t_2$ is if we already know that $t_2 = t_1$. Note that without this rule, there is no way to establish the validity of the following variant of the inference in our ongoing example:

$$\begin{array}{r} Ali = f(Sidra) \\ p(Ali) \\ \forall x.\, p(f(x)) \to p(x) \end{array} \;\vdash\; p(Sidra)$$

This is our original inference with the sides of the first premise reversed.

Now, when learning about the properties of equality in school, *transitivity* comes next right after reflexivity and symmetry. This is how it would be expressed as an inference rule:

$$\frac{t_1 = t_2 \quad t_2 = t_3}{t_1 = t_3}\,\mathsf{trans}$$

It says that one way to learn that two terms are equal is to know that they are both equal to a third term. With transitivity, we can learn that Sidra's father is the president of the logic lover society knowing that Sidra's father is Ali and that Ali is the president of the logic lover society:

$$\begin{array}{r} f(Sidra) = Ali \\ Ali = g(LLS) \end{array} \;\vdash f(Sidra) = g(LLS)$$

Here the function symbol $g(x)$ stands for "the president of x" and the name LLS indicates the logic lover society.

When learning about equality in school, there is one last property that is often mentioned: *congruence*. Congruence states that we can replace equals for equals. This

is how we solve equations for example. Should we have a rule for congruence? We do already! That's our elimination rule $=_{\mathsf{E}}$. Let's rename it cong to make school teachers around the world happy:

$$\frac{t_1 = t_2 \quad \varphi(t_2)}{\varphi(t_1)}\,\mathsf{cong}$$

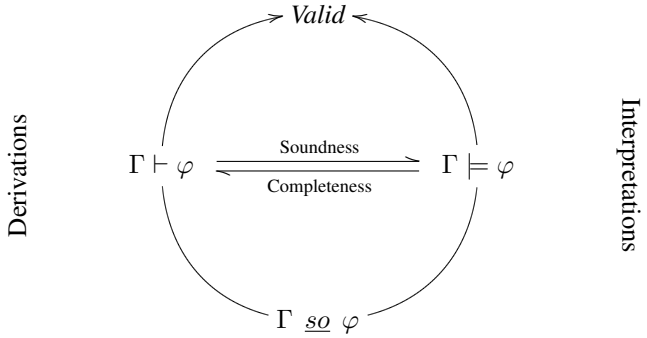
9.4.1 Soundness and Completeness

Looking back, we gave very different meanings to the predicate symbol "=" in the interpretation world and in the derivation world. Do the two methods tell us that the same inferences are valid? The maybe surprising answer is "yes!". And we have soundness and completeness theorems to show for that!

Theorem 9.1 (Soundness) *If* $\Gamma \vdash \varphi$*, then* $\Gamma \models \varphi$*.*

Theorem 9.2 (Completeness) *If* $\Gamma \models \varphi$*, then* $\Gamma \vdash \varphi$*.*

We will not prove that they hold, but generations of logicians have done that (typically as logic students). I told you that proving soundness and completeness is big business in logic! Here they are in pictorial form.



9.5 Definitions

One common use of equality, for example in math textbooks (but also in everyday life), is to give a name as an abbreviation of something that takes long to say or write. This is called a *definition*.

9.5.1 Defining Terms

We have given many definitions in this book. But let's start with a simple example. We don't have a name for Sidra's paternal grandfather: as of now, we can refer to him as either $father(father(Sidra))$ or $father(Ali)$.[2] Both are long-winded. What we

[2] As we introduce more function symbols, for clarity we are reverting to using the function symbol $father$ for somebody's father .

would typically do is to give him a nickname, say $Gido$. To do so, we would write, for example,

$$Gido = father(father(Sidra))$$

and use it as a premise. By doing so, we have defined the name $Gido$ to be an abbreviation for $father(father(Sidra))$. Using the derivation method, we can then show that the inference

$$\begin{array}{rl} Gido = father(father(Sidra)) & \\ p(Gido) & \vdash p(Sidra) \\ \forall x.\, p(father(x)) \rightarrow p(x) & \end{array}$$

is valid. This inference looks very similar to the main example in this chapter, which had the premise $f(Sidra) = Ali$: beside being the name of Sidra's father, Ali can also be seen as an abbreviation of $f(Sidra)$.

Now, the paternal grandfather of anybody is that person's father's father. We can then capture the idea of "the grandfather of x" by defining him as "the father of the father of x", for any x. We have just defined "grandfather" as an abbreviation that is *parametric* in x. The following formula then defines the symbol $grandfather$ in terms of $father$:

$$\forall x.\, grandfather(x) = father(father(x)) \tag{9.1}$$

This is a definition of the function symbol $grandfather$. This means that whenever we see the term $grandfather(t)$, we can replace it with the term $father(father(t))$ and vice versa. We abbreviated the complex term $father(father(x))$ to $grandfather(x)$, where $grandfather$ is a function symbol of convenience.

Now, let's do something a bit more challenging. Siblings have the same father. Given the names x and y of two siblings, we want to define the function symbol $commonFather(x, y)$ to refer to the name of their common father. Here is one of the various possible ways to write this definition:[3]

$$\begin{array}{rcl} \forall x.\, \forall y.\, \forall z.\, commonFather(x, y) = z & \leftrightarrow & father(x) = father(y) \\ & & \wedge\ z = father(x) \end{array}$$

It defines $commonFather(x, y)$ to be $father(x)$, but only if $father(x) = father(y)$. This last part imposes a condition, or constraint, on the individuals $commonFather(x, y)$ refers to. Therefore, this is a *conditional definition*. Such conditions require making use of connectives in the definition.

Using connectives opens lots of new opportunities. Just like we used the function symbol $father$ to refer to somebody's father, let's use $mother$ to refer to someone's mother. Then the following definition should capture the general concept of grandfather (as opposed to just paternal grandfather):

$$\begin{array}{rcl} \forall x.\, \forall y.\, grandfather(x) = y & \leftrightarrow & y = father(father(x)) \\ & & \vee\ y = father(mother(x)) \end{array} \tag{9.2}$$

Or does it? If we use this formula as the definition of grandfather, we can draw some surprising inferences. Say that we know that $father(father(Sidra)) = Gido$ and

[3] Can you find a simpler way to write it?

$father(mother(Sidra)) = Abraham$. Then the following inference has a derivation:

$$\frac{\begin{array}{c}\text{(formula 9.2)}\\ father(father(Sidra)) = Gido\end{array} \quad father(mother(Sidra)) = Abraham}{\vdash Gido = Abraham}$$

It concludes $Gido$ and $Abraham$ are the same person! What has gone wrong? The problem here is that formula (9.2) allows us to conclude both

$$grandfather(Sidra) = Gido \qquad \textit{and} \qquad grandfather(Sidra) = Abraham$$

Then, symmetry and transitivity combine them into the above unexpected conclusion, $Gido = Abraham$.

The root cause of the problem is that grandfather-hood is not a function: everybody has *two* grandfathers, not one. Therefore we cannot use a function symbol to define somebody's grandfather!

9.5.2 Defining Predicates

Is there any way to define the notion of generic grandfather in logic? There is, but not using terms. Because each person has more than one grandfather, we need to define $grandfather$ as a predicate symbol rather than as a function symbol. Here it is:

$$\begin{array}{lll}\forall x.\, \forall y.\, grandfather(x, y) & \leftrightarrow & y = father(father(x))\\ & & \vee\ y = father(mother(x))\end{array}$$

This definition says that the grandfather of x is y if either y is x's father's father or if it is x's mother's father.

Here, we have defined the predicate $grandfather$. It is again an abbreviation. Not an abbreviation of a (parametric or conditional) term this time, but an abbreviation of a formula. Specifically the formula on the right-hand side of $\leftrightarrow$. Note that x and y act as parameters in this definition.

The same device allows us to define not just grandfather-hood, but the general notion of ancestor. Here it is, limited to the paternal line for simplicity:

$$\begin{array}{lll}\forall x.\, \forall y.\, ancestor(x, y) & \leftrightarrow & y = father(x)\\ & & \vee\ ancestor(father(x), y)\end{array}$$

Differently from the previous ones, $ancestor$ is a *recursive* definition because it refers to itself.

As we will see in Chapter 11, predicate definitions can be leveraged as a means to use logic to carry out computations. Those definitions will not even need to use equality. Another way to carry out computation relies on nothing more than term equality, as examined in the previous section. This goes by the name of *term rewriting*. We will not discuss if further in this book, however.

9.6 *Exercises*

1. Give derivations for the following inferences that involve both function symbols and equality.

   (a) $a = f(b), b = f(c) \vdash a = f(f(c))$

   (b) $\vdash (\forall x.\, f(x) = g(x)) \wedge (\forall y.\, f(y) = h(y)) \rightarrow \forall z.\, h(z) = g(z)$

   (c) $\left[\begin{array}{r} \forall x.\, \forall y.\, f(x,y) = f(y,x) \\ \forall x.\, f(x,c) = x \end{array}\right] \vdash\ \forall y.\, f(c,y) = y$

   (d) $\left[\begin{array}{r} \exists y.\, \forall x.\, f(x) = y \\ \forall y.\, \exists z.\, g(y) = z \end{array}\right] \vdash\ \forall x.\, \exists z.\, g(f(x)) = z$

   (e) $\left[\begin{array}{r} \forall x.\, \forall y.\, f(x,y) = 1 \leftrightarrow x = 1 \vee y = 1, \\ \forall z.\, z = 0 \vee z = 1, \\ \neg(0 = 1) \end{array}\right] \vdash\ \forall x.\, f(0,x) = 0$

2. If you think about it, it looks like we could do away with the derivation rule for reflexivity, refl, if we replace it with the premise $\forall x.\, x = x$. Can we use this approach to replace all derivation rules for equality? Give similar premises for those rules for which this idea works (if any), and for those for which it doesn't work (if any) explain in detail what is going wrong.

3. Continuing on the last exercise, if you have a premise of the form $\forall x.\, x = x$, is it still necessary to give a special interpretation to the predicate "=" in the truth-based approach? Explain why or why not.

4. Do we actually need transitivity as an inference rule? Show that we can always replace rule trans with a combination of the other rules that deal with equality. This means we don't really need to have transitivity as a primitive rule. It is a derived rule.

5. In the previous exercise, we saw that we don't really need a rule for transitivity. Can we get rid of other rules as well? Not quite, but we can swap them for different rules. Show that the rule

   $$\frac{t_1 = t_2 \quad t_3 = t_2}{t_1 = t_3}\,\mathsf{ac}$$

   can replace commutativity (and transitivity). To do so, give a derivation of each of the following inferences using rule ac, but without using either sym or trans:

   - $t_1 = t_2 \vdash t_2 = t_1$
   - $t_1 = t_2, t_2 = t_3 \vdash t_1 = t_3$

   where t_1, t_2 and t_3 are generic terms.

6. Using the definition of paternal grandfather (9.1) from Section 9.5, show that the following conclusion is valid using the derivation method

   $$\forall x.\, father(grandfather(x)) = grandfather(father(x))$$

7. The first common ancestor of two people x and y, written $fca(x, y)$, is x if they are the same person. It is also x if x is the father of y. Similarly, it is y if y is the father of x. Otherwise it is the first common ancestor of their fathers. Give a definition of the function symbol $fca(x, y)$ in first-order logic. Be careful not to let your definition collapse distinct names (as we did in Section 9.5). [Hint: make sure that each disjunct is exclusive.]

8. Here are a few of the standard family relations. Identify the ones that can be expressed in logic by a function definition, and the ones that require a predicate definition. Then give these definitions.

   (a) Daughter

   (b) Oldest son

   (c) Grandparent (one's father or mother's father or mother)

   (d) First cousin (somebody sharing a grandparent)

   (e) Aunt (the daughter of a grandparent)

   (f) Eldest sister

   (g) Related (two people sharing a common ancestor)

   (h) Youngest uncle

   As you do so, you may assume that the function symbols $father(x)$ and $mother(x)$ and the predicate symbol $older(x, y)$ — x is older than y — have been predefined for you.

9. It is common practice to "define" exclusive-or (written $\oplus$) as

$$\varphi \oplus \psi \leftrightarrow (\varphi \vee \psi) \wedge \neg(\varphi \wedge \psi)$$

   Why is this not an acceptable definition according to what we saw in this chapter? Can we define exclusive-or in first-order logic?

10. A string is a sequence of characters. Once we fix the set of characters we can draw from (for example the uppercase letters of the English alphabet), strings can be described on the basis of the operation of concatenation, written "$\cdot$" infix , so that $s_1 \cdot s_2$ is the result of placing the string s_2 right after s_1, and the empty string ϵ. They obey the following laws:

    - Concatenation is associative, i.e., $s_1 \cdot (s_2 \cdot s_3)$ is the same string as $(s_1 \cdot s_2) \cdot s_3$ for any strings s_1, s_2 and s_3.
    - The empty string is the left and right identity of concatenation, that is $\epsilon \cdot s$ is just s and so is $s \cdot \epsilon$, for any string s.

    These laws apply anywhere inside a string, not just on the outside. Mathematical structures that work this way are called *semi-groups*.

    Can you use "$=$", as defined in this chapter, to express when two strings are equal according to these these laws? If so, do it. Otherwise, give another representation of the notion of equality over strings.

11. We could define "paternal grandfather" using a predicate symbol (rather than a function symbol as in Section 9.5) exactly in the same way as for (generic) "grandfather". Try it out for yourself.

    So, it seems that any function definition could be replaced by a predicate definition. Is this the case? If so, are there drawbacks to doing this? If not, describe this transformation.

Chapter 10

Numbers

One of the things we reason most often about are *numbers*. I am not just talking about those interminable math classes, but everyday activities like getting change, deciding if it's hot enough outside to leave a sweater at home, etc.

In the last chapter, equality allowed us to check that the \$2.17 we got back in change when making a \$7.83 purchase with a \$10 bill is the same as the \$2.17 change we needed. But bow do we determine that \$2.17 is the expected change for \$7.83 from \$10? Sure, we learn how to do subtraction in elementary school, but how do we get logic to do it?

10.1 Logic with Natural Numbers

So, let's take the plunge! We will be dealing with the simplest type of numbers you have been exposed to: the *natural numbers*, $0, 1, 2, \ldots$ (yes, they start at 0). In math classes, we denote them altogether as the set $\mathbb{N}$. What about the other numbers? Some, like $\mathbb{Z}$ and $\mathbb{Q}$ are fairly simple adaptations of what we will see. Others, like $\mathbb{R}$ and $\mathbb{C}$ rely on very different machinery — we will not look at them in this book.

If you want to use logic to draw inferences about numbers, our first task is to express them in logic. How do we do that? Since they are entities we want to talk about (as opposed to things that are true or false), they will have to be terms. How do we express "thirteen"? One idea is to have a separate constant for each number. The problem with this idea is that we would need infinitely many constants. Why is that a problem? Think about what the addition table would look like: an infinite matrix! Think about what it would take to define new operations on numbers? The operation that doubles a number, for example, would have an infinite description.

But this is not the way we use numbers in everyday life. Instead, we rely on a method that allows us to describe any number out of finitely many symbols and simple rules to arrange them. We use the decimal system, which consists of 10 symbols (or digits) and their relative positions to represent powers of 10. Then, thirteen is represented as "13", that is $1\times10^1+3\times10^0$. This is not the only way of representing num-

bers. You have probably studied the binary representation, which works pretty much in the same way, but with just two digits, 1 and 0. Here, thirteen is represented as "1101", that is $1\times2^3+1\times2^2+0\times2^1+1\times2^0$. A different example is the roman numerals, where thirteen is represented as "XIII". Roman numerals are however a pain to work with.

We will follow this overall tradition, but be as lazy as we can — you will appreciate that in no time! We will use the *unary system*, which represents each number by as many marks — think about prisoners in the movies making scratches in their cell walls to count the number of days they've been in. To do so in logic, we need just two symbols:

- One constant to represent the number zero. We will write it $\mathsf{0}$.
- One function symbol to represent the successor of a number. We will use the function symbol s and write $\mathsf{s}(n)$ for the successor of n.

The representation of a number in this way is called a *numeral*. Therefore, zero is simply $\mathsf{0}$, one is represented as the numeral $\mathsf{s}(\mathsf{0})$, five as the numeral $\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{0})))))$ and thirteen as the very long numeral $\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{0}))))))))))))$ — count the s's. This is not pretty, but it will be convenient.

It will also be useful to have a predicate that is true of all and only the natural numbers. This predicate, read "x is a natural number", will be written $\mathsf{nat}(x)$.

A logic that, like the one we are looking at, deals with natural number is called *arithmetic*. We are therefore trying to define and reason about arithmetic.

10.2 Inferences and Definitions

So, now that we have a way to describe the natural numbers, what should we do with them? For a start, we will be want to show that interesting formulas are valid, i.e., always true, without any premises. One of them is

$$\forall x.\, \mathsf{nat}(x) \to x = \mathsf{0} \vee \exists y.\, \mathsf{nat}(y) \wedge x = \mathsf{s}(y)$$

It says that every natural number is either zero or the successor of some other number.

We will also want to define useful operations, like doubling and addition — but we already know how to do that as of last chapter (we will see other ways in chapters to come). For example, here is a definition of the function that doubles its input, using the function symbol double:

$$\begin{cases} \mathit{double}(\mathsf{0}) &= \mathsf{0} \\ \forall x.\, \mathit{double}(\mathsf{s}(x)) &= \mathsf{s}(\mathsf{s}(\mathit{double}(x))) \end{cases}$$

This says that the double of zero is zero — in common math $2\times0=0$ — and the double of the successor of a number is double that number plus two — i.e., $2(x+1)=2x+2$. Notice that the definition we have given for double is recursive.

For practice, let's define the addition function, written $\mathit{plus}(x,y)$:

$$\begin{cases} \forall y.\quad \mathit{plus}(\mathsf{0},y) &= y \\ \forall x.\,\forall y.\, \mathit{plus}(\mathsf{s}(x),y) &= \mathsf{s}(\mathit{plus}(x,y)) \end{cases}$$

Simply put, zero plus any number is that number — $0 + y = y$ — and adding the successor of a number to another is the same as taking the successor of their sum — $(x + 1) + y = (x + y) + 1$.

With these definitions in place, we will be interested in showing that properties like

$$\forall x.\, plus(x, x) = double(x)$$

are valid. Note that when dealing with numbers, we tend to talk about "properties" rather than "inferences". Clearly we are interested in showing that the above formula is the conclusion of a valid inference with no premises.

As usual, we have two ways to show that an inference is valid. One uses interpretations, the other derivations. Let's look at both.

10.3 Arithmetic Interpretations

For us, 0, s and nat have a special meaning. They are interpreted symbols like "=" was in the last chapter. This will inform the various ingredients of an interpretation for formulas about numbers. Let's examine these ingredients:

Domain of discourse: Because we are doing inferences about the natural numbers, we want the domain of interpretation $\mathbb{D}$ to contain the natural numbers. Therefore

$$\mathbb{N} \subseteq \mathbb{D}$$

Note that there may be other elements in $\mathbb{D}$ besides numbers. In the next few examples, we will consider the following domain of discourse:

$$\mathbb{D} = \mathbb{N} \cup \{\text{  }, \text{  }\}$$

Interpretation of names: The symbol 0 is a name, and we will naturally interpret it as the natural number zero ($0 \in \mathbb{N}$). If our other name is *Sidra*, we have

Name	$\mathbb{D}$
0	0
Sidra	

Interpretation of functions: We want the symbol s to be interpreted as the successor function for the elements of $\mathbb{D}$ that are natural numbers. The following interpretation of this symbol has this characteristic:

$x \in \mathbb{D}$	$\mathsf{s}(x)$
0	1
1	2
2	3
⋮	⋮
	
	42

Note that for elements of $\mathbb{D}$ that are not numbers, s can be interpreted as anything we want.

Clearly, we could have other function symbols in our logic, which would have their own interpretation.

Interpretation of predicates: In an interpretation of arithmetic, the predicate symbol nat must be true of all the natural numbers, and false of every other element of $\mathbb{D}$. So, its interpretation is as follows in our example:

$x \in \mathbb{D}$	$\mathsf{nat}(x)$
0	T
1	T
2	T
$\vdots$	$\vdots$
	F
	F

We may have other predicates to interpret. In the discussion so far, we used "=" which is interpreted as in the previous chapter. There may be others.

Note that nat is the only way to keep us on the straight and narrow: when we intend to talk about numbers (and numbers only), we should guard our statements and definitions with it, or we may not be expressing what we have in mind.

Take the definition of $plus$ above. You would like to be able to prove that

$$\forall x.\, \forall y.\, \forall z.\, plus(x, y) = z \rightarrow \mathsf{nat}(z)$$

But it doesn't hold! Take x to be 0 and y and z to be  . The first part of the definition of $plus$ tells us that, for these values, $plus(x, y) = z$ is true, but  is not a natural number, so that the above formula is altogether false.

But we can use nat to make sure the variable y in this formula is a natural number. We get the following corrected definition:

$$\begin{cases} \forall y.\, \mathsf{nat}(y) \rightarrow & plus(\mathsf{0}, y) &=& y \\ \forall x.\, \forall y. & plus(\mathsf{s}(x), y) &=& \mathsf{s}(plus(x, y)) \end{cases}$$

which makes sure that we won't try to say anything about adding things that are not numbers. But shouldn't we guard x and y similarly in the second formula? We could, but in this case this is not necessary.[1]

We have rediscovered something we knew already: interpretations are good for showing that an inference does not hold. As usual, showing validity is a lot more complicated. Note in particular that here the domain $\mathbb{D}$ is necessarily infinite.

[1] A different approach to making sure that variables can take only certain values is to categorize terms using *sorts*, which are similar to types in programming languages. Then by specifying the sort of every variable we use, we would prevent it from assuming values we are not interested in — just like when writing a program in Java, say.

10.4 Derivations

So, let's turn to derivations, which we have seen are a tractable method for checking validity (although not for showing that an inference does not hold).

We need to come up with elementary inference rules that describe how to introduce and eliminate our one interpreted predicate, $\mathsf{nat}(x)$. The introduction rules are easy:

$$\frac{}{\mathsf{nat}(\mathsf{0})}\,\mathsf{nat_{I1}} \qquad \frac{\mathsf{nat}(t)}{\mathsf{nat}(\mathsf{s}(t))}\,\mathsf{nat_{I2}}$$

They simply say that $\mathsf{0}$ is a natural number, and that $\mathsf{s}(t)$ is a natural number whenever we can show that t is one too. Using them, we can easily show that $\mathsf{nat}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{0})))))))$ is derivable — i.e., that five is indeed a natural number. That's reassuring!

Next, how can we use the information that n is a natural number? Here is the elimination rule for $\mathsf{nat}(x)$, and you tell me what it says.

$$\frac{\mathsf{nat}(n) \quad \varphi(\mathsf{0}) \qquad \begin{array}{c}\overline{\mathsf{nat}(x)} \quad \overline{\varphi(x)}\,(x)\\ \vdots\\ \varphi(\mathsf{s}(x))\end{array}}{\varphi(n)}\,\mathsf{nat_E}$$

Let's see. The premises require that n be a natural number, that $\varphi(0)$ be provable, and that $\varphi(\mathsf{s}(x))$ hold for an arbitrary natural number x such that $\varphi(x)$ holds. The conclusion tells us that, in this case, $\varphi(n)$ holds.

Have you seen this before?

The last two premises are the cases of a proof by *mathematical induction* for φ. You would use them to derive $\forall x.\, \varphi(x)$, from which $\varphi(n)$ holds by instantiating the variable x to n — that's using rule $\forall_{\mathsf{E}}$.

So, yes, that rule corresponds to the principle of mathematical induction. You have probably seen it written in a different way:

$$\varphi(0) \land (\forall x'.\, \varphi(x') \to \varphi(\mathsf{s}(x'))) \to \forall x.\, \varphi(x)$$

where $\varphi(x)$ is any formula (usually called "property") about the natural number x. Given what we experienced earlier, we'd better be very precise about φ's argument really ranging over the natural numbers. Let's use nat to guard it:

$$\varphi(0) \land (\forall x'.\, \mathsf{nat}(x') \land \varphi(x') \to \varphi(\mathsf{s}(x'))) \to \forall x.\, (\mathsf{nat}(x) \to \varphi(x))$$

This makes it very close to the above rule. Note that this formula incorporates all kinds of quantifiers and connectives, while rule $\mathsf{nat_E}$ is a pure elimination rule: it only mentions the formula construction being eliminated, nat.

You may have seen the principle of mathematical induction written as follows (let's drop the nat guards for clarity):

$$\forall P.\, (P(0) \land \forall x'.\, P(x') \to P(\mathsf{s}(x'))) \to \forall x.\, P(x)$$

Here, we have replaced the formula φ with a universally quantified formula that we called P. Doing so takes us outside first-order logic, where we can only quantify on variables representing terms. In fact, the ability of quantifying over formulas would bring us into *second-order logic*. Second-order logic is a lot more powerful than first-order logic, but it is also a lot more complicated. We will largely stay away from it.

So, let's see the above rules in action by showing that the formula

$$\forall x.\, \mathsf{nat}(x) \to x = 0 \lor \exists y.\, \mathsf{nat}(y) \land x = \mathsf{s}(y)$$

is indeed derivable (from no premises). Here is a derivation for it:

$$\dfrac{\dfrac{}{\mathsf{nat}(n)}(1) \quad \dfrac{\dfrac{}{0 = 0}\,\mathsf{refl}}{0 = 0 \lor \exists y.\, \mathsf{nat}(y) \land 0 = \mathsf{s}(y)}\lor_{\mathsf{I1}} \quad \dfrac{\dfrac{\dfrac{}{\mathsf{nat}(n')}(2) \quad \dfrac{}{\mathsf{s}(n') = \mathsf{s}(n')}\,\mathsf{refl}}{\mathsf{nat}(n') \land \mathsf{s}(n') = \mathsf{s}(n')}\land_{\mathsf{I}}}{\exists y.\, \mathsf{nat}(y) \land \mathsf{s}(n') = \mathsf{s}(y)}\exists_{\mathsf{I}}(\mathsf{n}')}{\mathsf{s}(n') = 0 \lor \exists y.\, \mathsf{nat}(y) \land \mathsf{s}(n') = \mathsf{s}(y)}\lor_{\mathsf{I2}}}{n = 0 \lor \exists y.\, \mathsf{nat}(y) \land n = \mathsf{s}(y)}\,\mathsf{nat}_{\mathsf{E}}(2,3)$$

$$\dfrac{\dfrac{n = 0 \lor \exists y.\, \mathsf{nat}(y) \land n = \mathsf{s}(y)}{\mathsf{nat}(n) \to n = 0 \lor \exists y.\, \mathsf{nat}(y) \land n = \mathsf{s}(y)}\to_{\mathsf{I}}(1)}{\forall x.\, \mathsf{nat}(x) \to x = 0 \lor \exists y.\, \mathsf{nat}(y) \land x = \mathsf{s}(y)}\forall_{\mathsf{I}}(n)$$

Wow! And this was a simple property! The moment we start doing interesting things with natural numbers, the derivations get pretty big. This calls for automation, but we know that no program can reliably determine validity for all first-order logic inferences. What to do then? It turns out that, for many common inferences, automatic theorem provers can give an answer quite quickly. For others, if it is important that we show validity (e.g., an important theorem or a property of a safety-critical system), then people will simply try hard, with the help of automated proof assistants.

10.5 Soundness and ... Completeness?

At this point, we are quite used to expecting that the two notions of valid inferences we have given are equivalent, expressed as soundness and completeness results. The good news is that soundness holds. Indeed, the following theorem holds, and is not all that hard to prove.

Theorem 10.1 (Soundness) *If* $\Gamma \vdash \varphi$*, then* $\Gamma \models \varphi$.

The opposite is however not true: completeness does not hold in general. Do you want a counterexample? What about the inference with no premises and $\neg(0 = \mathsf{s}(0))$ as its conclusion?

In any interpretation with the characteristics we have given in Section 10.3, it is very easy to show that

$$\models \neg(0 = \mathsf{s}(0))$$

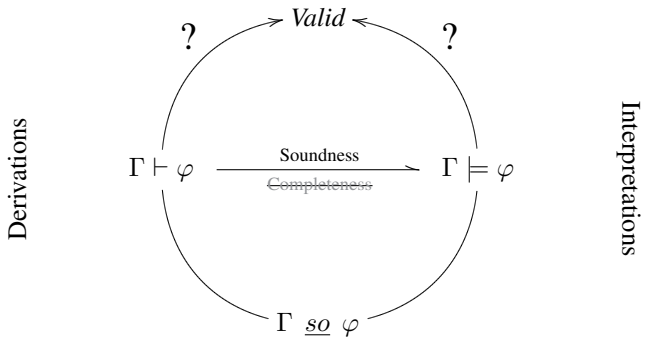
Indeed, $\mathsf{0}$ is mapped to 0, $\mathsf{s}(\mathsf{0})$ is mapped to 1, the interpretation of equality ensures that $0 = 1$ is false, which means that $\neg(0 = 1)$ is true. So the above inference is valid in the world of interpretations.

However, there is no way to derive

$$\vdash \neg(\mathsf{0} = \mathsf{s}(\mathsf{0}))$$

This may not be evident, but trust me on this! In fact, nothing in our derivation rules prohibits that various numerals refer to the same number. Everything we have done would work fine for arithmetic modulo 5 for example.

Our pictorial illustration of the relationship between derivation- and interpretation-based validity now looks as follows:



But, which of these two notions should be pick as the official definition of validity?

If what we have in mind are the natural numbers as we learn them in school, with $0 \neq 1$, then we want validity to coincide with the interpretation given in Section 10.3. Then, to recover completeness, we can strengthen our notion of derivation to match interpretations. To do so, we need to add just two rules:

$$\frac{}{\neg(\mathsf{s}(t) = \mathsf{0})} \qquad \frac{\mathsf{s}(t_1) = \mathsf{s}(t_2)}{t_1 = t_2}$$

Note however that these rules are not in the standard form of elementary inference rules: they are neither introduction nor elimination rules. Altogether, the resulting rules constitute what is known as *Peano arithmetic* after the logician Giuseppe Peano (1858–1932).

But we may be interested in the more liberal notion of number that the derivation rules in Section 10.4 gave us. We then get completeness back by making our requirements on interpretations for numbers less stringent by simply relaxing the status of $\mathsf{0}$ and s to uninterpreted symbols. We would keep our interpretation of nat the same, however.

10.6 Axiomatization of Arithmetic

The derivation rules given in the Section 10.4 are but one of the ways to incorporate arithmetic into logic. Another way, extremely popular at the turn of the 20-th century, is to capture everything needed to work with numbers in a set of premises. These sets of premises were called *axioms*.[2]

10.6.1 Presburger Arithmetic

One of the simplest axiomatic descriptions of arithmetic has the sole purpose of describing the way the natural numbers and the operation of addition work. It retains the connectives and quantifiers of first-order logic, but disallows all predicate symbols except for equality (with its usual rules), all function symbols except s for the successor operations and "$+$" for addition, and all constants except $\mathsf{0}$ for zero. Then, the natural numbers with addition are completely described by just four axioms and the axiom schema of mathematical induction. This axiomatization is due to the mathematician Mojżesz Presburger (1904–1943) and therefore know as *Presburger arithmetic*.

The four axioms are as follows:

$$\begin{array}{ll} \forall x. & \neg(\mathsf{0} = \mathsf{s}(x)) \\ \forall x.\, \forall y. & \mathsf{s}(x) = \mathsf{s}(y) \to x = y \\ \forall x. & x + \mathsf{0} = x \\ \forall x.\, \forall y. & x + \mathsf{s}(y) = \mathsf{s}(x + y) \end{array}$$

The first two correspond to the rules we added at the end of the last section to restore completeness: zero is not the successor of any number, and if two successor numbers are equal then they are the successors of equal numbers. The last two are exactly our definition of addition from Section 10.2. Note that, with such a restricted symbol set, we do not need to guard these axioms with a predicate such as nat.

The axiom schema of mathematical induction has the following form:

$$\left(\begin{array}{l} \varphi(\mathsf{0}) \\ \land\ \forall x.\, \varphi(x) \to \varphi(\mathsf{s}(x)) \end{array} \right) \to \forall y.\, \varphi(y)$$

where $\varphi(x)$ is any first-order formula containing the free variable x. This is an axiom *schema* rather than a simple axiom: we get an actual axiom for every choice of the formula φ. It therefore describes an infinite set of axioms, albeit a very predictable one. In particular the premise set of Presburger arithmetic is infinite.

Presburger arithmetic is rather weak: unless enriched with further axioms or additional function or predicate symbols, it allows us only to express simple properties of

[2] Until the end of the 19-th century, arithmetic and other parts of mathematics (for example geometry) were thought to obey fundamental laws that the mathematicians of the time called *axioms*. Axioms were therefore laws of nature, fundamental truths that simply had to be and on which all of mathematics was founded. Nineteenth century mathematicians started questioning this view, and explored what happens if these laws were replaced with others. This led to an enormous jump in our understanding of mathematics, and the birth of modern logic. Nowadays, the word "axiom" is just a synonym for assumption or premise.

addition. In particular, there is no way to define or reason about multiplication in it, let alone more advanced concepts such as prime numbers. Yet, it is sound and complete with respect to the interpretation with domain $\mathbb{N}$ (without extra elements), 0 mapped to 0, and s and $+$ corresponding to the successor and addition operations over the natural numbers.

The study of which rules or axioms are needed to obtain a certain interpretation we have in mind is an area of logic called *model theory*. Presburger arithmetic is an excellent example of this.

10.6.2 Peano Arithmetic

In its simplest form, Peano arithmetic, which we encountered in Section 10.5 already, is axiomatized by extending Presburger arithmetic with multiplication and two defining axioms. This simple extension makes Peano arithmetic much more powerful than Presburger's variant however. In fact, it is so powerful that it can express the whole of logic.

Here is an axiomatization meant to coexist with additional symbols. For this reason, we guard variables standing for natural numbers with the predicate symbol nat, which the first two axioms define similarly to rules $\mathsf{nat}_{\mathsf{I1}}$ and $\mathsf{nat}_{\mathsf{I2}}$. The axiom schema of mathematical induction, shown last, is adapted similarly.

$$
\begin{array}{lll}
(1) & & \mathsf{nat}(0) \\
(2) & \forall x. & \mathsf{nat}(x) \to \mathsf{nat}(\mathsf{s}(x)) \\
(3) & \forall x. & \mathsf{nat}(x) \to \neg(0 = \mathsf{s}(x)) \\
(4) & \forall x.\, \forall y. & \mathsf{nat}(x) \wedge \mathsf{nat}(y) \wedge \mathsf{s}(x) = \mathsf{s}(y) \to x = y \\
(5) & \forall x. & \mathsf{nat}(x) \to x + 0 = x \\
(6) & \forall x.\, \forall y. & x + \mathsf{s}(y) = \mathsf{s}(x + y) \\
(7) & \forall x. & \mathsf{nat}(x) \to x \times 0 = 0 \\
(8) & \forall x.\, \forall y. & x \times \mathsf{s}(y) = x + (x \times y)
\end{array}
$$

$$
\left(\begin{array}{l} \varphi(0) \\ \wedge\ \forall x.\, \mathsf{nat}(x) \wedge \varphi(x) \to \varphi(\mathsf{s}(x)) \end{array} \right) \to \forall y.\, \mathsf{nat}(y) \to \varphi(y)
$$

10.7 *Exercises*

1. Show that the inference $\models \forall x.\, \mathsf{nat}(\mathsf{s}(x)) \to \mathsf{nat}(x)$ is not valid.

2. Let Γ_{double} be the set of formulas that defines $double$. Show that there is a derivation of
$$\Gamma_{double} \vdash \forall x.\, \mathsf{nat}(x) \to \mathsf{nat}(double(x))$$

3. Show that the principle of mathematical induction,
$$\varphi(0) \wedge \forall x'.\, (\mathsf{nat}(x') \wedge \varphi(x') \to \varphi(\mathsf{s}(x'))) \to \forall x.\, (\mathsf{nat}(x) \to \varphi(x))$$

is valid in the derivation system.

4. Taking inspiration to the definitions of $double$ and $plus$, give definitions for the following functions on natural numbers:

   - $mult$ so that $mult(x, y)$ is equal to $x \times y$.
   - exp so that $exp(x, y)$ is equal to x^y.
   - $fact$ so that $fact(x)$ is equal to $x!$ (the factorial of x).
   - fib so that $fib(x)$ is equal to the x-th Fibonacci number.

   As you do so, make sure that your definitions apply *only* to natural numbers.

5. Using the same method, give a definition of subtraction, $x - y$. Note that this operation is not defined on all pairs of natural numbers.

6. In the last two exercises, we have been defining functions. Let's try our hand at defining predicates instead. Give a definition of the following predicates:

   - $divides$ so that $divides(x, y)$ is true when x divides y exactly — this is often written as $x \mid y$ in math textbooks. For example $3 \mid 6$ is true but $3 \mid 5$ is false.
   - $prime$ so that $prime(x)$ is true exactly when x is a prime number (whose only divisors are 1 and itself).

7. We need three symbols to express the natural numbers in binary notation within first-order logic: the constant $\mathsf{0}$ for zero, and the function symbols e and o so that $\mathsf{e}(x)$ and $\mathsf{o}(x)$ represent $2x$ and $2x + 1$ respectively.

   - What is the representation of thirteen?
   - Give a definition of the operation that returns the successor of a number? Can you also define the double of a number? What about addition?
   - How would you express the decimal system for the natural numbers in first-order logic? Do you want to give it a try and define successor, doubling and addition? I don't blame you!

   You will have observed by now that the more digits, the shorter the numerals get and the bigger the definitions become.

8. One appealing way to express the *integers* (i.e., the elements of the set $\mathbb{Z}$, which contains the negative numbers) is to use the function symbol p so that $\mathsf{p}(x)$ corresponds to the predecessor of x. This is in addition to $\mathsf{0}$ and s. Doing so and nothing else has one drawback. What is this drawback? How would you fix it?

   Your fix, if it is what I expect, does not work in first-order logic without equality. Can you think of a representation of the integers that does not depend on equality?

9. Give a derivation for the following inference

$$\vdash \forall x.\, \mathsf{nat}(x) \to \neg(x = \mathsf{s}(x))$$

Can you do so with just the introduction and elimination rules?

A term ω such that $\omega = \mathsf{s}(\omega)$ would come very close to representing infinity itself: a number that is its own successor. That's neat, isn't it? We need to make a single change to our elementary derivation rules to accommodate it, and still be able to show that $\neg(n = m)$ when n and n are different numerals. Can you identify this one change? Can you then express the principle of strong induction as an axiom schema?

10. Peano arithmetic can be extended in many ways. One such way is to add a predicate symbol that captures the "less than" relation between number. As in common mathematics, it is written $x < y$. Give a definition for this predicate. How would you extend the axioms of Peano arithmetic to incorporate it?

11. The discussion in this chapter centered around the natural numbers but all concepts we saw apply to many constructions used every day in mathematics and computer science. Lists, or sequences, are one of them — for simplicity, we will consider only lists of natural numbers. To represent them, we need one predicate symbol, list so that $\mathsf{list}(x)$ is true exactly when x is a list of natural numbers, one constant, nil to represent the empty list, and one binary function symbol, cons so that $\mathsf{cons}(x, l)$ is the list with head x and tail l. Your job will be to redo pretty much everything we did in this chapter, but with lists instead of numbers. Specifically,

    - Define the operation of appending a list to another list, denote $append(l_1, l_2)$, which is equal to the list that contains all the elements of l_1 followed by all the elements of l_2, in the order they occur in these lists.
    - Define the operation that returns the length of a list.
    - Describe what an interpretation must be like to account for lists.
    - Give introduction and elimination rules for lists by taking inspiration to what we did in the case of the natural numbers. The introduction rules are easy. The elimination rule (yes, there is just one) is another induction principle, this time on lists.
    - Does completeness hold? What would you need to add to achieve it?

Chapter 11

Computing with Logic

Work in Progress

```
parent(abraham, herb).    parent(mona,    herb).
parent(abraham, homer).   parent(mona,    homer).

parent(clancy,  marge).   parent(jackie,  marge).
parent(clancy,  patty).   parent(jackie,  patty).
parent(clancy,  selma).   parent(jackie,  selma).

parent(homer,   bart).    parent(marge,   bart).
parent(homer,   lisa).    parent(marge,   lisa).
parent(homer,   maggie).  parent(marge,   maggie).

parent(selma,   ling).

female(mona).       female(lisa).       male(abraham).
female(jackie).     female(maggie).     male(clancy).
female(marge).      female(ling).       male(herb).
female(patty).                          male(homer).
female(selma).                          male(bart).
```

11.1 *Exercises*

1. Define the following family relations on the basis of the predicates used in this chapter:
   - `uncle(X,Y)` which holds if `X` is the uncle of `Y`.
   - `cousin(X,Y)` which hold if `X` is a cousin of `Y`.

Figure 11.1: The Simpsons' Family Tree

You will notice that everybody and his brother (literally) is his own cousin. You can say that two people should be different by using the built-in predicate `X \= Y`.

2. `X \= Y` is one of those predicates where Prolog is at odds with logic. Consider the following variant of the definition of `member` earlier in this chapter.

```
member2(X, [X|_]).
member2(X, [Y|L]) :- X \= Y, member2(X,L).
```

Some queries will give you unexpected answers. Find an example and try to explain why it doesn't work.

Chapter 12

Meta-Logic

So, we have seen that we can get logic to help us check the validity of inferences about people being in Doha, about things being equal, and about numbers. We can adapt these techniques to similarly reason about a lot of other entities of interest to computer scientists, like strings, lists, trees, programs, and much much more.

What about logic itself? Let's give it a try and see whether we learn something interesting along the way.

12.1 Reasoning about Logic

What does it mean to make inferences about logic? In a way, this is what we have been doing all along in this book. We have done so using words, saying things like

"if φ is true and $\varphi \to \psi$ is true, then ψ is true" is a valid inference.

This is a statement about a logical inference, and specifically about its validity. Our goal, then, is to use logic to make inferences about such statements. As in this example, we will be extremely interested in using logic to make inferences about validity. To do so, we need to define a formula $\mathit{Valid}(x, y)$ such that $\mathit{Valid}(\Gamma, \varphi)$ is true if and only if the inference Γ so φ is valid.

But we cannot do that! We are allowed to replace variables only with terms, but Γ is a *set of formulas* (the premises of the inference) and φ is a *formula* (its conclusion).

One thing we can do, however, is to *encode* formulas and sets of formulas as terms, and then we will have a shot at defining Valid. Let's do that, and see where it gets us.

12.2 Logic in Logic

We have two tasks at hand. The first one is to express the language of logic using terms, so that we can then refer to logical expressions in the arguments of predicates

such as Valid. The second is to define Valid itself so that $\mathit{Valid}(\ulcorner\Gamma\urcorner, \ulcorner\varphi\urcorner)$ is true, or derivable, exactly when the inference Γ so φ is valid, where $\ulcorner\Gamma\urcorner$ and $\ulcorner\varphi\urcorner$ are the encodings of Γ and φ we came up with.

12.2.1 Encoding Logic

First-order logic has a lot of ingredients: variables, symbols, terms, formulas and even sets of formulas. We need to give an *encoding*, or *representation*, for each of them. Let's roll up our sleeves.

Propositional Connectives

Let's start with something really simple, the propositional constant $\top$, which is always true. If we want to represent $\top$ as a term, we can just pick a name for it and use this name whenever we mean $\top$. Let's choose the name true for this purpose, since it is easy to remember. In general, we will write $\ulcorner\varphi\urcorner$ for the encoding of a formula φ. Then

$$\ulcorner\top\urcorner = \mathit{true}.$$

That was easy. We can do something similar for more complex formulas. Take conjunction for example. If we want to represent $\varphi_1 \wedge \varphi_2$, then we can first encode φ_1 and φ_2, obtaining $\ulcorner\varphi_1\urcorner$ and $\ulcorner\varphi_2\urcorner$, and then fix a binary function symbol, say and, and apply it to them. Therefore

$$\ulcorner\varphi_1 \wedge \varphi_2\urcorner = \mathit{and}(\ulcorner\varphi_1\urcorner, \ulcorner\varphi_2\urcorner).$$

We can proceed in the same way for all the other connectives, using binary function symbols or, imp and iff to encode $\vee$, $\rightarrow$ and $\leftrightarrow$ respectively, the unary function symbol not for $\neg$, and the constant false for $\bot$.

Up to now, we can encode propositional formulas without atomic propositions, like $\top \wedge (\bot \rightarrow \top \vee \bot)$, which gets represented as the term

$$\mathit{and}(\mathit{true}, \mathit{imp}(\mathit{false}, \mathit{or}(\mathit{true}, \mathit{false}))).$$

Atomic Formulas

What about the atomic formulas $p(t_1, \ldots, t_n)$? A few things are necessary to encode them:

- We need to give an encoding to each term t — let's denote it $\ulcorner t\urcorner$.
- We need to give a representation $\ulcorner t_1, \ldots, t_n\urcorner$ to tuples of terms $t_1, \ldots, t_n$.
- Finally, we need to associate some term, for example a constant, to each predicate symbol p. How we do this doesn't really matter, as long as the association is unique. We will write the representation of p, whatever we choose it to be, as $\ulcorner p\urcorner$.

We will see in a bit how to achieve the first two. Once we have them, then all we need is to pick a binary function symbol, say $atomic$, and define

$$\ulcorner p(t_1, \ldots, t_n) \urcorner = atomic(\ulcorner p \urcorner, \ulcorner t_1, \ldots, t_n \urcorner).$$

Before we proceed any further, let's see how we can encode tuples. We will turn them into the ordered lists of their constituents. To do this, we need one constant, say nil, to denote the empty list, and one binary function symbol, say $cons$, to encode the extension of a list with an element. Therefore, if x is an element and l is a list, then $cons(x, l)$ is the list with head x and tail l. Here is the full encoding:

$$\begin{cases} \ulcorner t_1, t_2, \ldots, t_n \urcorner &= cons(\ulcorner t_1 \urcorner, \ulcorner t_2, \ldots, t_n \urcorner) \\ \ulcorner . \urcorner &= nil \end{cases}$$

Did you notice it is recursive?

Now, how do we encode terms? For terms starting with a function symbol, say $f(t_1, \ldots, t_n)$, we can use the same idea as for atomic predicates: pick a binary function symbol fun that takes a representation of f and a representation the tuple $t_1, \ldots, t_n$. Therefore

$$\ulcorner f(t_1, \ldots, t_n) \urcorner = fun(\ulcorner f \urcorner, \ulcorner t_1, \ldots, t_n \urcorner).$$

We can actually use this same representation for names by considering them as function symbols that take no arguments:

$$\ulcorner c \urcorner = fun(\ulcorner c \urcorner, nil),$$

Again, the representation of each name or function symbol can be any constant we want as long as it doesn't clash with other constants.

The last kind of terms we need to give an encoding for are variables. We can do so in several ways. For example, we can assign a distinct number to every variable and use a unary function symbol var applied to this number as the encoding of this variable. Therefore, if x is given number 42, then $\ulcorner x \urcorner = var(42)$. We could also use the string "x". What matters is that every occurrence of x is encoded in the same way and that we can tell different variables apart.

Good! We know how to encode terms and tuples of terms.

At this point, we are able to represent any first-order formula that does not make use of quantifiers. Take for example $LogicLover(father(Sidra)) \rightarrow LogicLover(Sidra)$ — remember? we used it to express that if Sidra's father loves logic, then Sidra must love logic too. It is encoded as follows on the left-hand side:

$$\begin{array}{l} imp(atomic(p\_LogicLover, \\ \quad cons(fun(f\_father, \\ \quad\quad cons(fun(f\_Sidra, nil), \\ \quad\quad nil)), \\ \quad nil)), \\ \quad atomic(p\_LogicLover, \\ \quad cons(fun(f\_Sidra, nil), \\ \quad nil)) \end{array} \qquad \begin{array}{l} LogicLover\,(\\ \quad father\,(\\ \quad\quad Sidra \\ \quad) \\ ) \\ \rightarrow \quad LogicLover\,(\\ \quad Sidra \\ ) \end{array}$$

where we have used the constant $p\_LogicLover$ to represent the predicate symbol $LogicLover$, and similarly used the constants f_father and f_Sidra to represent the function symbol $father$ and the name $Sidra$, respectively. The shaded right-hand side relates each component of this encoding to the corresponding part of our formula (except for the infix connective $\rightarrow$, which is encoded as the prefix function symbol imp).

That's getting pretty hairy, isn't it? Writing

$$\ulcorner LogicLover(father(Sidra)) \rightarrow LogicLover(Sidra)\urcorner$$

for this term (without bothering about the sausage-making) makes it easier to see what's going on.

Quantifiers

To finish off the encoding of formulas, all we need is a representation of the quantifiers. We can use the technique we employed earlier, but we have to be careful with the variables. Let's pick the binary function symbol all to encode the universal quantifier. Its first argument will be the quantification variable and the second will be an encoding of the quantified formula. Then,

$$\ulcorner \forall x.\, \varphi \urcorner = all(\ulcorner x \urcorner, \ulcorner \varphi \urcorner).$$

The same encoding used for x in the first argument should be used for each occurrence of x in φ, at least the ones bound by $\forall x$.

We can do the same for the existential quantifier:

$$\ulcorner \exists x.\, \varphi \urcorner = exists(\ulcorner x \urcorner, \ulcorner \varphi \urcorner).$$

As an example, the formula $\forall x.\, LogicLover(father(x)) \rightarrow LogicLover(x)$ is encoded as:

$$\begin{array}{lll}
all(var(29), & & \forall x. \\
\quad imp(atomic(p_LogicLover, & & \quad LogicLover\,(\\
\qquad cons(fun(f_father, & & \quad\; father\,(\\
\qquad\quad cons(var(29), & & \quad\;\; x \\
\qquad\quad nil)), & & \quad\;) \\
\qquad nil)), & &) \\
\quad atomic(p_LogicLover, & \rightarrow & \quad LogicLover\,(\\
\qquad cons(var(29), & & \quad\; x \\
\qquad nil)) & &)
\end{array}$$

where we represented the variable x as the term $var(29)$.

Sets of Formulas

Since we now know about lists, we can use them to encode the premises of an inference. Therefore, if $\Gamma = \varphi_1, \ldots, \varphi_n$, we define $\ulcorner \Gamma \urcorner$ as the list of the encodings $\ulcorner \varphi_1 \urcorner, \ldots, \ulcorner \varphi_n \urcorner$.

This completes the representation of the language of first-order logic as terms. That was not too bad, was it? Observe that none of our encodings involved introducing new predicates. We were able to represent terms, formulas and sequences of each as terms by introducing new function symbols and using them in a judicious way.

12.2.2 Defining Derivability

Now that we have a way to encode formulas and sequences of formulas, we have a shot at reasoning about validity in logic. The idea of a formula $\mathit{Valid}(\ulcorner\Gamma\urcorner, \ulcorner\varphi\urcorner)$ is now meaningful. Next we need to define it so that it is true exactly when the inference Γ so φ is valid. How to go about it? In fact, what do we mean by "valid"? We have seen that we have two ways to define validity, one in term of interpretations and the other via derivations. For many logics (but not all as we learned in Chapter 10), soundness and completeness results ensure us that they coincide. The first of these two approaches forces us to consider infinitely many interpretations to define validity — doing so in logic sounds like a daunting task. The second is instead based on just a handful of rules and simple ways to combine them into derivations. This is more promising: we will base our logical description of validity on derivablility. To emphasize this, we will use the predicate symbol $\mathit{Derivable}$ rather than Valid.

Let's then define the predicate $\mathit{Derivable}(x, y)$ so that $\mathit{Derivable}(\ulcorner\Gamma\urcorner, \ulcorner\varphi\urcorner)$ is true, or derivable, exactly when $\Gamma \vdash \varphi$, i.e., when φ is derivable from Γ. We will collect the definition of $\mathit{Derivable}$, as well as the definition of a few auxiliary predicates, in a set of premises that we call Γ_{FOL}.

To define $\mathit{Derivable}$, we are going to go rule by rule. Let's start with rule $\top_{\mathsf{I}}$, which introduces $\top$:

$$\frac{}{\top}\,\top_{\mathsf{I}}$$

It says that $\top$ is always derivable, whatever assumptions are around. Then, the formula

$$\forall y.\, \mathit{Derivable}(y, \mathit{true})$$

defines the derivability of $\ulcorner\top\urcorner$ (that's the constant true) from any set of assumptions (that's y).

Next, consider the rule $\wedge_{\mathsf{I}}$,

$$\frac{\varphi_1 \quad \varphi_2}{\varphi_1 \wedge \varphi_2}\,\wedge_{\mathsf{I}}$$

It tells us that, if we can derive φ_1 from the current assumptions, say Γ, and φ_2 from Γ, then we can derive of $\varphi_1 \wedge \varphi_2$ from Γ. Then, the following first-order formula captures exactly what we just described:

$$\begin{array}{l}\forall x_1.\, \forall x_2.\, \forall y.\\ \qquad \mathit{Derivable}(y, x_1)\\ \quad \wedge\ \mathit{Derivable}(y, x_2)\\ \rightarrow \mathit{Derivable}(y, \mathit{and}(x_1, x_2))\end{array}$$

If we instantiate the variables to $\ulcorner\varphi_1\urcorner$, $\ulcorner\varphi_2\urcorner$ and $\ulcorner\Gamma\urcorner$ respectively, we get

$$\begin{array}{c} \mathit{Derivable}(\ulcorner\Gamma\urcorner, \ulcorner\varphi_1\urcorner) \\ \wedge\ \mathit{Derivable}(\ulcorner\Gamma\urcorner, \ulcorner\varphi_2\urcorner) \\ \rightarrow \mathit{Derivable}(\ulcorner\Gamma\urcorner, \mathit{and}(\ulcorner\varphi_1\urcorner, \ulcorner\varphi_2\urcorner)) \end{array}$$

which is exactly the above process of showing that $\varphi_1 \wedge \varphi_2$ are derivable, but at the level of our representation of first-order logic.

We can do this for every derivation rule of propositional logic. The rules for the quantifiers make use of some additional operations however. The one that will be most interesting for us in this chapter is substitution, the operation of replacing every occurrence of a variable x with a term t in a formula $\varphi(x)$ — we denoted the result as $\varphi(t)$ in rules $\forall_\mathsf{E}$ and $\exists_\mathsf{I}$ in Chapter 7. We express substitution as the predicate $\mathit{subst}(x, y, z)$, defined in such a way that $\mathit{subst}(\ulcorner\varphi(x)\urcorner, t, T)$ is true exactly when $T = \ulcorner\varphi(t)\urcorner$, i.e., when the term T is the representation of the formula obtained by substituting every occurrence of x in formula $\varphi(x)$ with term t.[1]

Encoding derivability for the quantifiers makes use of a second auxiliary predicate, fresh, which produces a new name, as required by rules $\forall_\mathsf{I}$ and $\exists_\mathsf{E}$. One last auxiliary predicate is needed to draw the conclusion of an inference from its assumptions: the predicate *lookup* is such that $\mathit{lookup}(\ulcorner\varphi\urcorner, \ulcorner\Gamma\urcorner)$ is true exactly when $\varphi \in \Gamma$.

Spelling out all the details of the definition of $\mathit{Derivable}$ is not hard but a bit tedious. We will skip it. What really matters is what this definition is able to do. It captures the statement "$\Gamma \vdash \varphi$ is derivable" very precisely. Specifically, $\mathit{Derivable}$ has the following two properties:

- *If* $\Gamma \vdash \varphi$, *then* $\Gamma_{FOL} \vdash \mathit{Derivable}(\ulcorner\Gamma\urcorner, \ulcorner\varphi\urcorner)$. (12.1)
- *If* $\Gamma \nvdash \varphi$, *then* $\Gamma_{FOL} \nvdash \mathit{Derivable}(\ulcorner\Gamma\urcorner, \ulcorner\varphi\urcorner)$. (12.2)

Note that the second line says that whenever $\Gamma \vdash \varphi$ is not derivable, neither is $\Gamma_{FOL} \vdash \mathit{Derivable}(\ulcorner\Gamma\urcorner, \ulcorner\varphi\urcorner)$. It does not say that $\Gamma_{FOL} \vdash \neg\mathit{Derivable}(\ulcorner\Gamma\urcorner, \ulcorner\varphi\urcorner)$ is derivable, which is a much stronger property, as we will see. This is because of the undecidability result we mentioned in Chapter 7.

12.3 Meta-Mathematics

We did it! We have defined when an inference is valid using first-order logic. We are all set to reason about logic in logic. This is big business in logic, to the point that it has a fancy name: it is called *meta-mathematics* when reasoning about aspects of mathematics in logic, or more narrowly *meta-logic* when focusing on just logic.

What kind of reasoning tasks shall we do, then? Let us start with some simple properties, and then get to something a bit more fancy.

[1] We treat x as a distinguished variable — typically it is the only free variable in φ. It is easy to generalize subst to take a fourth argument, the representation of which variable to substitute with t.

12.3.1 Simple Properties

Say that you have established that some inference Γ so φ is valid and you have a derivation to show for it. What happens if a friend, or a professor, gives you one more premise? Can you still build a derivation of φ even with this additional premise? Sure! You just ignore it. This simple observation is called the *weakening lemma*. Here is its statement.

Lemma 12.1 (Weakening) *If $\Gamma \vdash \varphi$, then $\Gamma, \psi \vdash \varphi$ for any ψ.*

where ψ is your new premise, so that Γ, ψ is the updated premise set.

How do we know this lemma is true? Well, we prove it. But what is this proof? By and large, it is an English justification with some math symbols here and there. But now that we can encode logic in logic, we should be able to do better. We can express the weakening lemma as an inference in logic as follows:[2]

$$\Gamma_{FOL} \vdash \mathit{Derivable}(\ulcorner\Gamma\urcorner, \ulcorner\varphi\urcorner) \to \mathit{Derivable}(\ulcorner\Gamma, \psi\urcorner, \ulcorner\varphi\urcorner)$$

Now, our informal proof that the lemma is valid can be replaced with a derivation for this inference. In fact, were we to give an encoding to derivations as well (and indeed we can), we could ask a computer to check that it is a valid proof of our lemma.

The weakening lemma is one of the most elementary properties of first-order logic. Here is another one, the *substitution lemma*, which says that if a formula ψ is derivable from some assumptions Γ, and a second formula φ is derivable from Γ together with ψ, then φ is derivable from Γ alone without the need for ψ. Here it is as a mathematical statement.

Lemma 12.2 (Substitution) *If $\Gamma \vdash \psi$ and $\Gamma, \psi \vdash \varphi$, then $\Gamma \vdash \varphi$.*

If you look at it, this property is the justification for using lemmas in a proof: ψ is a lemma used in the proof of φ. This result says that lemmas are a convenient way of partitioning a large proof into smaller, more tractable parts, but you don't really need them.

Can we express this lemma too as an inference within logic? Here it is:

$$\Gamma_{FOL} \vdash \mathit{Derivable}(\ulcorner\Gamma\urcorner, \ulcorner\psi\urcorner) \wedge \mathit{Derivable}(\ulcorner\Gamma, \psi\urcorner, \ulcorner\varphi\urcorner) \to \mathit{Derivable}(\ulcorner\Gamma\urcorner, \ulcorner\varphi\urcorner).$$

The same considerations as for the weakening lemma apply here as well.

12.3.2 On to the Limits of Logical Reasoning

There are a lot of statements like the above that we can express and reason about within logic. In fact, entire logic courses worth of statements.

Rather than doing that — take those courses if you are interested! — we will use the machinery we have developed in this chapter to explore how much we can really do with logic.

[2] This is slightly simplified. The actual inference for the weakening lemma would use universally quantified variables rather than terms such as $\ulcorner\Gamma\urcorner$ and $\ulcorner\varphi\urcorner$ and additional components would force these variables to stand for the representation of the various entities.

Consistency

One important property of first-order logic is that there is no formula such that both it and its negation are derivable from no premises. Logicians call this property *consistency*. Here is its mathematical statement:

Property 1 (Consistency) *There is no formula φ such that $\cdot \vdash \varphi$ and $\cdot \vdash \neg\varphi$.*

If it didn't hold, rule $\neg_{\mathsf{E}}$ would be able to derive every formula we can think of. This would be absurd: we could show that both "Sidra loves Logic" and "Sidra does not love logic" are true statements out of thin air. An inconsistent logic, where everything is true, is not very useful. Good thing first-order logic is consistent!

We can internalize consistency in first-order logic just like we did for the weakening and substitution lemmas. It is expressed as follows:

$$\Gamma_{FOL} \vdash \neg\exists x.\, \mathit{Derivable}(\ulcorner\cdot\urcorner, x) \wedge \mathit{Derivable}(\ulcorner\cdot\urcorner, \mathit{not}(x)) \tag{12.3}$$

Consistency is retained even if we add Γ_{FOL} as premises. This means that Γ_{FOL} is itself consistent, which is expressed by the following lemma.

Lemma 12.3 (Consistency of Γ_{FOL}) *There is no formula φ such that $\Gamma_{FOL} \vdash \varphi$ and $\Gamma_{FOL} \vdash \neg\varphi$.*

Diagonalization

Recall that we introduced the predicate symbol subst to capture the way substitution works. In particular, $\Gamma_{FOL} \vdash \mathit{subst}(\ulcorner\varphi(x)\urcorner, t, T)$ is derivable exactly when T is $\ulcorner\varphi(t)\urcorner$, the representation of the result of substituting t for x in $\varphi(x)$. One consequence of this is that the inference

$$\Gamma_{FOL} \vdash \exists z.\, \mathit{subst}(\ulcorner\varphi(x)\urcorner, t, z)$$

is always derivable. This is by using rule $\exists_{\mathsf{I}}$ with witness $\ulcorner\varphi(t)\urcorner$ for the variable z.

That's pretty simple stuff, isn't it? True, so let's add a twist. Consider the following formula $\psi(x)$ where $\varphi(z)$ is any formula you want:

$$\underbrace{\exists z.\, \varphi(z) \wedge \mathit{subst}(x, x, z)}_{\psi(x)}$$

Stare at the second part, $\mathit{subst}(x, x, z)$, and you will realize it's wicked! The first argument of subst stands for the representation of a formula. Instantiating x to the representation of some formula $\chi(x)$, we get the formula $\psi(\ulcorner\chi(x)\urcorner)$ which contains $\mathit{subst}(\ulcorner\chi(x)\urcorner, \ulcorner\chi(x)\urcorner, z)$ — this formula wants to substitute x in $\chi(x)$ with $\chi(x)$'s own representation! By the above discussion, this has a derivation if and only if $z = \ulcorner\chi(\ulcorner\chi(x)\urcorner)\urcorner$. This is all a bit twisted, but it kind of makes sense, in a weird way.

Let's then add another twist: we will pick $\psi(x)$ itself as the formula $\chi(x)$. That is, we are looking at the formula δ given by

$$\delta = \psi(\ulcorner\psi(x)\urcorner)$$

This is $\psi(x)$ applied to its own representation. Now that's twisted!! Let's unfold it like we did earlier:

$$\begin{aligned}\delta &= \psi(\ulcorner\psi(x)\urcorner) \\ &= \exists z.\, \varphi(z) \wedge \mathit{subst}(\ulcorner\psi(x)\urcorner, \ulcorner\psi(x)\urcorner, z)\end{aligned}$$

Now, δ is derivable from Γ_{FOL} exactly when $\exists z.\, \varphi(z) \wedge \mathit{subst}(\ulcorner\psi(x)\urcorner, \ulcorner\psi(x)\urcorner, z)$ is, since they are the same formula. For this formula to be derivable, it must be the case that there is a term T such that $\varphi(T) \wedge \mathit{subst}(\ulcorner\psi(x)\urcorner, \ulcorner\psi(x)\urcorner, T)$, which in turn requires that both

$$\varphi(T) \qquad \textit{and} \qquad \mathit{subst}(\ulcorner\psi(x)\urcorner, \ulcorner\psi(x)\urcorner, T)$$

be derivable in Γ_{FOL}.

By the previous observation, the formula on the right is derivable in Γ_{FOL} exactly for $T = \ulcorner\psi(\ulcorner\psi(x)\urcorner)\urcorner$. But notice that this is $\ulcorner\delta\urcorner$! This means that δ will be derivable in Γ_{FOL} exactly when $\varphi(\ulcorner\delta\urcorner)$ is.

Starting from an arbitrary formula $\varphi(z)$, we were able to construct a formula δ that is derivable exactly when $\varphi(z)$ applied to δ's very representation is derivable. Logicians found this result so remarkable that they gave it a name, the *diagonalization lemma*. It is usually expressed in the following way, which is equivalent to what we have just obtained.

Lemma 12.4 (Diagonalization) *For every formula $\varphi(z)$ there exists a formula δ such that*

$$\Gamma_{FOL} \vdash \delta \leftrightarrow \varphi(\ulcorner\delta\urcorner)$$

The formula δ is a self-referential sentence that can be viewed as saying of itself that it has the property φ.

Incompleteness

The diagonalization lemma holds for every formula $\varphi(z)$. So, it holds also if we choose the formula $\neg \mathit{Derivable}(\ulcorner\Gamma_{FOL}\urcorner, z)$ as $\varphi(z)$. It then tells us that there exists a formula δ_G such that

$$\Gamma_{FOL} \vdash \delta_G \leftrightarrow \neg \mathit{Derivable}(\ulcorner\Gamma_{FOL}\urcorner, \ulcorner\delta_G\urcorner) \tag{12.4}$$

This is weird: we are applying $\mathit{Derivable}$ on the representation of the set of premises Γ_{FOL} that we used to define $\mathit{Derivable}$. Is this another of those twisted sentences? Let's see what it means. We know that $\mathit{Derivable}(\ulcorner\Gamma_{FOL}\urcorner, \ulcorner\delta_G\urcorner)$ is derivable when δ_G is derivable from Γ_{FOL}. Then, this inference says that δ_G is true if and only if it is not derivable in Γ_{FOL}, and all this has a derivation in Γ_{FOL}!!!

Wicked! But is δ_G true? Is it false? We've got to check. Since δ_G can use all kind of things defined in Γ_{FOL}, what we are really asking is whether $\Gamma_{FOL} \vdash \delta_G$ or $\Gamma_{FOL} \vdash \neg\delta_G$.

Assume $\Gamma_{FOL} \vdash \delta_G$:

Since by property (12.1) $\mathit{Derivable}$ represents derivability, for sure we have that $\Gamma_{FOL} \vdash \mathit{Derivable}(\ulcorner\Gamma_{FOL}\urcorner, \ulcorner\delta_G\urcorner)$.

On the other hand, since $\Gamma_{FOL} \vdash \delta_G$, instance (12.4) of the diagonalization lemma entails that $\Gamma_{FOL} \vdash \neg Derivable(\ulcorner\Gamma_{FOL}\urcorner, \ulcorner\delta_G\urcorner)$.

But putting these two things together would mean that Γ_{FOL} is inconsistent! We know this is not the case by lemma 12.3. So the assumption $\Gamma_{FOL} \vdash \delta_G$ got us into trouble: we have reached a contradiction. Therefore it must be the case that $\Gamma_{FOL} \nvdash \delta_G$.

Assume that $\Gamma_{FOL} \vdash \neg\delta_G$:

Then clearly $\Gamma_{FOL} \nvdash \delta_G$, or Γ_{FOL} would once more be inconsistent, which lemma 12.3 assures us it is not. Now, by property (12.2), we then have that $\Gamma_{FOL} \nvdash Derivable(\ulcorner\Gamma_{FOL}\urcorner, \ulcorner\delta_G\urcorner)$. One consequence of this is that $\Gamma_{FOL} \nvdash \neg\delta_G$, because otherwise we could infer that $\Gamma_{FOL} \vdash Derivable(\ulcorner\Gamma_{FOL}\urcorner, \ulcorner\delta_G\urcorner)$ by diagonalization instance (12.4), which is the opposite of what we just obtained, namely $\Gamma_{FOL} \nvdash Derivable(\ulcorner\Gamma_{FOL}\urcorner, \ulcorner\delta_G\urcorner)$.

But $\Gamma_{FOL} \nvdash \neg\delta_G$ contradicts our assumption! This means that this case cannot be either!

Altogether, we have found that neither $\Gamma_{FOL} \vdash \delta_G$ nor $\Gamma_{FOL} \vdash \neg\delta_G$ are derivable in first-order logic. This is odd, because the diagonalization lemma constructed δ_G with exactly the kind of things that Γ_{FOL} talks about. This oddity becomes even more evident if we apply the property (12.2) of Γ_{FOL} on these statements: we get that $\Gamma_{FOL} \nvdash Derivable(\ulcorner\Gamma_{FOL}\urcorner, \ulcorner\delta_G\urcorner)$ and $\Gamma_{FOL} \nvdash \neg Derivable(\ulcorner\Gamma_{FOL}\urcorner, \ulcorner\delta_G\urcorner)$.

So, although Γ_{FOL} is a representation of derivability of first-order logic, there is a statement of first-order logic for which it is unable to give us an answer about whether it is derivable or not. This inability is called *incompleteness*. Here is a slightly more general formulation as a theorem.

Theorem 12.5 (Incompleteness of Γ_{FOL}) *There exists a set of formulas Γ and a formula φ such that*

- $\Gamma_{FOL} \nvdash Derivable(\ulcorner\Gamma\urcorner, \ulcorner\varphi\urcorner)$
- $\Gamma_{FOL} \nvdash \neg Derivable(\ulcorner\Gamma\urcorner, \ulcorner\varphi\urcorner)$

Said differently, there is an inference $\Gamma \vdash \varphi$ such that Γ_{FOL} is unable to tell us whether it is valid or not. But we defined Γ_{FOL} precisely for the purpose of reasoning about logic inferences! How can it fail so miserably?

12.4 Gödel's Incompleteness Theorems

Although I didn't show you all the details, we were pretty careful when putting together Γ_{FOL} — trust me. But maybe we made a mistake somewhere. Maybe there is a better encoding of first-order logic that is immune from incompleteness.

Unfortunately there isn't. It is not just Γ_{FOL}, but *any* encoding of first-order logic that is incomplete: one will always be able to find a formula about first-order logic,

like δ_G, for which this encoding cannot tell us whether it is derivable or not derivable. First-order logic is itself incomplete.

This is not even just a shortcoming of first-order logic, but of any logic that is consistent and expressive enough to represent itself, in particular its own notion of validity. This property was first proved by the logician Kurt Gödel in 1930 and it caused quite a stir in the logic circles of the time. It is called *Gödel's first incompleteness theorem*.[3]

Theorem 12.6 (Gödel's First Incompleteness Theorem) *Any consistent logical system expressive enough to capture its own notion of validity has statements about itself that it can neither prove nor disprove.*

Gödel's first incompleteness theorem is one of the most important results in modern logic. It establishes fundamental limitations about what logic can do, and therefore about what we can expect from it. It essentially says that we cannot use a logic to determine the validity of all of its own statements (however some "stronger" logics can prove all statements of "weaker" logics).

Gödel's approach to proving this result was similar to what we saw in Section 12.2, but with some interesting differences. His starting point was any logic that contained Peano arithmetic from Chapter 10. Rather than encoding this logic as a set of premises Γ_{FOL}, Gödel represented terms, formulas, everything as numbers — that's why he needed arithmetic in the first place. This process is now known as *Gödel numbering* or Gödelization (a word now used for any encoding of logic as terms). Numbers are enough to encode all of logic! The rest of Gödel's original proof is very similar to what we saw, with the exception that, for technical reasons, he encoded derivations and the statement "$\mathcal{D}$ is a derivation of $\Gamma \vdash \varphi$" as a jumping board to defining derivability.

First incompleteness theorem? Does that mean there are more? Indeed! Gödel showed with his *second incompleteness theorem* that no consistent logic expressive enough to capture its own notion of validity can prove that it is in fact consistent. This means in particular that we cannot use first-order logic to prove that first-order logic is consistent! So, inference (12.3) does not have a derivation even if it expresses a true statement about first-order logic!! We need a more expressive logic for this ... and an even more expressive logic to prove the consistency of that logic ... and so on ... for ever.

Theorem 12.7 (Gödel's Second Incompleteness Theorem) *No consistent logical system expressive enough to capture its own notion of validity can prove that it is in fact consistent.*

Gödel's incompleteness theorems establish fundamental limitation about what logic can do. For example, no theorem prover that mechanizes first-order logic will be able

[3] We met Gödel before: he is the one who proved that first-order logic was *complete*. Confused? You are right to be! Completeness as in Theorem 8.2 and incompleteness as in Theorem 12.5 have nothing to do with each other. The first property shows that interpretations and derivations capture the same notion of validity. The second exposes the inability of many logical systems to tell whether their own inferences are valid.

This unfortunate similarity in names has confused generations of students, and even some professors. Not only that, but both were proved by the same person!

to decide all first-order inferences, and definitely not consistency. The same applies to more powerful logics. These limitations of mechanical reasoning also anticipate limitations of what computers can do, a notion that was established a few years after Gödel's results by the logician Alan Turing (1912–1954).

12.5 *Exercises*

1. Give an encoding of the following formulas:
   - $\forall x.\, \exists y.\, \mathit{sister}(x, y) \land \mathit{logicLover}(x) \to \mathit{logicLover}(y)$
   - $\forall x.\, \mathit{plus}(\mathsf{s}(\mathsf{0}), x) = \mathsf{s}(x)$
2. Complete the encoding of the language of first-order logic in Section 12.2.1 by giving a precise mathematical definition of $\ulcorner t \urcorner$ for a term t, of $\ulcorner \varphi \urcorner$ for a first-order formula φ and of $\ulcorner \Gamma \urcorner$ for a set of formulas $\Gamma = \varphi_1, \ldots, \varphi_n$.
3. Give a definition of the substitution predicate $\mathit{subst}(x, y, z)$ discussed in Section 12.2.2 in first-order logic. Your definition will need to make use of auxiliary predicates for terms and tuples — define them as well. As you do so, it will be clear that you need to augment subst with a fourth argument.
4. Define the predicates
   - *lookup* such that $\mathit{lookup}(\ulcorner \Gamma \urcorner, \ulcorner \varphi \urcorner)$ is true if and only if φ occurs in Γ.
   - *fresh* such that $\mathit{fresh}(\ulcorner \Gamma \urcorner, \ulcorner \varphi \urcorner, a)$ is true if and only if a is the representation of a constant that does not occur in Γ and φ. To do so, you need to commit to a specific encoding of names.
5. Now that you have all the ingredients, complete the definition of $\mathit{Derivable}$.
6. In this exercise, we will explore a variant of $\mathit{Derivable}$ that expresses the statement "$\mathcal{D}$ is a derivation of $\Gamma \vdash \varphi$" rather than just "$\Gamma \vdash \varphi$ is derivable". This is quite close to how Gödel proved his first incompleteness theorem. We will encode this statement by means of a ternary predicate $\mathit{Derives}$ such that

   $$\Gamma_{FOL} \vdash \mathit{Derives}(\ulcorner \mathcal{D} \urcorner, \ulcorner \Gamma \urcorner, \ulcorner \varphi \urcorner) \qquad \text{iff} \qquad \mathcal{D} \text{ is a derivation of } \Gamma \vdash \varphi$$

   where $\ulcorner \mathcal{D} \urcorner$ is a representation of the derivation $\mathcal{D}$. Just like $\mathit{Derivable}$, $\mathit{Derives}$ is defined rule by rule. Here are two examples. Rule $\top_{\mathsf{I}}$

   $$\frac{}{\top}\,\top_{\mathsf{I}}$$

   is a derivation of $\top$. Thus, if we pick the constant TrueI to represent this rule, then

   $$\forall y.\, \mathit{Derives}(\mathit{TrueI}, y, \mathit{true})$$

   defines the derivability of $\ulcorner \top \urcorner$ from any set of assumptions using TrueI as the representation of the inference consisting of just rule $\top_{\mathsf{I}}$.

Next, consider the rule $\wedge_\mathsf{I}$,

$$\frac{\varphi_1 \quad \varphi_2}{\varphi_1 \wedge \varphi_2}\wedge_\mathsf{I}$$

It tells us that, if we have a derivation, say $\mathcal{D}_1$, of φ_1 from the current assumptions, say Γ, and a derivation $\mathcal{D}_2$ of φ_2 from Γ, then we obtain a derivation of $\varphi_1 \wedge \varphi_2$ from Γ by combining $\mathcal{D}_1$ and $\mathcal{D}_2$ by means of rule $\wedge_\mathsf{I}$. Let's introduce a function symbol $andI$ to represent the use of $\wedge_\mathsf{I}$ to combine $\mathcal{D}_1$ and $\mathcal{D}_2$ into the derivation of $\varphi_1 \wedge \varphi_2$. Then, the following first-order formula captures exactly the process just described:

$$\begin{aligned}&\forall x_1.\, \forall x_2.\, \forall y.\, \forall D_1.\, \forall D_2.\\ &\qquad Derives(D_1, y, x_1)\\ &\quad \wedge\ Derives(D_2, y, x_2)\\ &\rightarrow Derives(andI(D_1, D_2), y, and(x_1, x_2))\end{aligned}$$

If we instantiate the variables to $\ulcorner\varphi_1\urcorner$, $\ulcorner\varphi_2\urcorner$, $\ulcorner\Gamma\urcorner$, $\ulcorner\mathcal{D}_1\urcorner$ and $\ulcorner\mathcal{D}_2\urcorner$ respectively, we get

$$\begin{aligned}&Derives(\ulcorner\mathcal{D}_1\urcorner, \ulcorner\Gamma\urcorner, \ulcorner\varphi_1\urcorner)\\ &\wedge\ Derives(\ulcorner\mathcal{D}_2\urcorner, \ulcorner\Gamma\urcorner, \ulcorner\varphi_2\urcorner)\\ &\rightarrow Derives(andI(\ulcorner\mathcal{D}_1\urcorner, \ulcorner\mathcal{D}_2\urcorner), \ulcorner\Gamma\urcorner, and(\ulcorner\varphi_1\urcorner, \ulcorner\varphi_2\urcorner))\end{aligned}$$

which is exactly the above process of constructing a derivation using $\wedge_\mathsf{I}$, but at the level of our representation of first-order logic.

Your turn. Define $Derives$ for the remaining rules of first-order logic — you will see a lot of similarities with the definition of $Derivable$.

How would you define $Derivable$ in terms of $Derives$? You can do so in a single line.

7. The consistency of first-order logic states that there is no formula φ such that $\cdot \vdash \varphi$ and $\cdot \vdash \neg\varphi$. Note in particular that the set of assumptions is empty (written "$\cdot$"). This definition does not work in general if we do not make this restriction. Find a set of formulas Γ and a formula φ such that $\Gamma \vdash \varphi$ and $\Gamma \vdash \neg\varphi$ are both derivable.

8. A logic that is not incomplete in the sense of Section 12.2.2 is called *complete* (yes, I know, it's confusing). This doesn't mean however that for every formula φ and a consistent Γ, either $\Gamma \vdash \varphi$ or $\Gamma \vdash \neg\varphi$ are derivable. Completeness only deals with situations where Γ encodes validity for this logic and φ is a statement about it. To make this point clear, find a first-order formula φ such that $\cdot \nvdash \varphi$ and $\cdot \nvdash \neg\varphi$ *even if first-order logic were complete*.

Appendix A

Collected Rules

This appendix collects the elementary derivation rules presented in chapters 4, 7, 9 and 10.

A.1 First-Order Logic

$$\frac{\varphi_1 \wedge \varphi_2}{\varphi_1}\,\wedge_{\mathsf{E1}} \qquad \frac{\varphi_1 \wedge \varphi_2}{\varphi_2}\,\wedge_{\mathsf{E2}} \qquad \frac{\varphi_1 \quad \varphi_2}{\varphi_1 \wedge \varphi_2}\,\wedge_{\mathsf{I}}$$

$$\text{(no } \top_{\mathsf{E}}\text{)} \qquad \frac{}{\top}\,\top_{\mathsf{I}}$$

$$\frac{\varphi_1 \vee \varphi_2 \qquad \begin{array}{c}\overline{\varphi_1}\\ \vdots\\ \psi\end{array} \qquad \begin{array}{c}\overline{\varphi_2}\\ \vdots\\ \psi\end{array}}{\psi}\,\vee_{\mathsf{E}} \qquad \frac{\varphi_1}{\varphi_1 \vee \varphi_2}\,\vee_{\mathsf{I1}} \qquad \frac{\varphi_2}{\varphi_1 \vee \varphi_2}\,\vee_{\mathsf{I2}}$$

$$\frac{\bot}{\psi}\,\bot_{\mathsf{E}} \qquad \text{(no } \bot_{\mathsf{I}}\text{)}$$

$$\frac{\varphi \quad \varphi \to \psi}{\psi}\,\to_{\mathsf{E}} \qquad \frac{\begin{array}{c}\overline{\varphi}\\ \vdots\\ \psi\end{array}}{\varphi \to \psi}\,\to_{\mathsf{I}}$$

$$\frac{\varphi \quad \neg\varphi}{\psi}\,\neg_{\mathsf{E}} \qquad \frac{\begin{array}{c}\overline{\varphi}\\ \vdots\\ \bot\end{array}}{\neg\varphi}\,\neg_{\mathsf{I}}$$

$$\frac{\neg\neg\varphi}{\varphi}\,\neg^{2}$$

$$\frac{\forall x.\,\varphi(x)}{\varphi(t)}\,\forall_{\mathsf{E}} \qquad \frac{\begin{array}{c}(a)\\ \vdots\\ \varphi(a)\end{array}}{\forall x.\,\varphi(x)}\,\forall_{\mathsf{I}}$$

$$\frac{\exists x.\,\varphi(x) \qquad \begin{array}{c}\overline{\varphi(a)}^{\,(a)}\\ \vdots\\ \psi\end{array}}{\psi}\,\exists_{\mathsf{E}} \qquad \frac{\varphi(t)}{\exists x.\,\varphi(x)}\,\exists_{\mathsf{I}}$$

A.2 Equality

$$\frac{t_1 = t_2 \quad \varphi(t_2)}{\varphi(t_1)}\,\mathsf{cong}$$

$$\frac{}{t = t}\,\mathsf{refl} \qquad \frac{t_2 = t_1}{t_1 = t_2}\,\mathsf{sym} \qquad \frac{t_1 = t_2 \quad t_2 = t_3}{t_1 = t_3}\,\mathsf{trans}$$

A.3 Arithmetic

$$\frac{}{\mathsf{nat}(0)}\,\mathsf{nat}_{\mathsf{I1}} \qquad \frac{\mathsf{nat}(t)}{\mathsf{nat}(\mathsf{s}(t))}\,\mathsf{nat}_{\mathsf{I2}} \qquad \frac{\mathsf{nat}(n) \quad \varphi(0) \qquad \begin{array}{c}\overline{\mathsf{nat}(x)} \quad \overline{\varphi(x)}^{\,(x)}\\ \vdots\\ \varphi(\mathsf{s}(x))\end{array}}{\varphi(n)}\,\mathsf{nat}_{\mathsf{E}}$$

Bibliography

[1] Deborah Bennett. *Logic Made Easy: How to Know When Language Deceives You*. W.W. Norton & Company, 2005.

[2] Lewis Carroll. Alice's Adventures in Wonderland, 1865.

[3] Martin Davis. *The Universal Computer: The Road From Leibniz to Turing*. A K Peters/CRC Press, 2nd edition, 2011.

[4] Apostolos Doxiadis and Christos H. Papadimitriou. *Logicomix: An Epic Search for Truth*. Bloomsbury, 2009.

[5] Judith L. Gersting. *Mathematical Structures for Computer Science*. W. H. Freeman, 6th edition, 2006.

[6] Douglas R. Hofstadter. *Gödel, Escher, Bach: An Eternal Golden Braid*. Basic Books, 1999. First published in 1979.

[7] Graham Priest. *Logic: A Very Short Introduction*. Oxford University Press, 2001.

[8] Greg Restall. *Logic: An Introduction*. McGill-Queen's University Press, 2004.

[9] Raymond M. Smullyan. *The Gödelian Puzzle Book: Puzzles, Paradoxes and Proofs*. Dover Publications, 2013.

Index