CHAPTER 1

INTRODUCTION

We all have our own intuitions about what is true and what is false. For example it is the case that $1 + 2 = 3$ is true while $2 + 2 = 3$ is false, and there are formal ways to show that this is the case. There are other entities that we can deem to be true or false, for example "someone likes chocolate" might be true—it depends on whether we can find evidence of the existence of such a person.

These kinds of expressions are called atomic sentences and we will always assume that we know the truth value of these. From atomic sentences we can build more complicated ones; for example "someone likes chocolate and $1 + 2 = 3$". We reason about the truth of such sentences by looking at the truth value of the components and reasoning about the way they are connected together. In this example we used the connective "and"; other ones that we will use are "or", "not" and "implies" (if ... then ...). All of these connectives follow our intuitions: "$A$ and $B$" is true just when both $A$ and $B$ are true; "$A$ or $B$" is true if either of $A$ or $B$ is true; "not $A$" is true if $A$ is false; "$A$ implies $B$" is true if when $A$ is true then $B$ is also true. We shall see later in fact that "implies" can be rather subtle.

Rules that allow us to deduce more facts from previously known facts are called deduction rules—if the premises of a rule are true, then the conclusion of the rule is also true (provided of course that the rule is valid). For example, if $A$ and $B$ are true, we can deduce that $A$ is true, since if both $A$ and $B$ are true, then it must be the case that $A$ is true—it is one of the assumptions! Notice that we are not at all interested in what $A$ and $B$ are—they could be anything at all—we are only interested in the form of the sentences that we write.

In addition to the above connectives, there are quantifiers that allow us to specify the domain of discourse. These are "for all" which specifies a universal quantification and "there exists" which specifies an existential quantification. Think of "for all" as really meaning all, and "there exists" as meaning at least one. For example, $A$ implies $A$ is always true no matter what $A$ is, so we could deduce that "for all $A$, $A$ implies $A$". We could also say that "there exists $A$, $A$ implies $A$", since if it is the case that the sentence is true for all $A$, then it is certainly true for one such $A$.

One should take great care with quantifiers. Consider the sentence "if $n$ is a number, then there is another number $m$ such that $n^2 = m$". Intuitively, what we mean here is that for all numbers $n$, there exists a number $m$ such that $n^2 = m$; and we certainly don’t mean that there exists a number $n$ such that for all $m$, $n^2 = m$.

It is a fruitful exercise for the reader to try to write down some sentences, in
natural language, that are true using these connectives.

During logical reasoning there is an element of structural reasoning that allows us to manipulate a sentence. For example we would expect the truth value of "A and B" to be the same as "B and A". However, what about this sentence: I watch the TV and I break the TV. I would certainly have difficulty in watching the TV after I broke it! Hence there are circumstances in which we would prefer not to have the commutativity rule.

"Roses are red and violets are blue therefore we can deduce that I like logic". In this sentence, the premises are true, and the conclusion is true, so is it a valid deduction? The answer depends on whether we are required to use the assumptions (the information) during the deduction or not. Different kinds of logic have been introduced to express these kinds of entities. It is not the formal power of different logics that is at issue, but rather how useful they are at representing different real life situations.

Systems of formal logic have been developed so that we can reason about sentences built from these kinds of connectives in a precise way. There are a collection of formal systems that capture different concepts. Here are just a few of the well known ones:

- Classical logic
- Intuitionistic logic
- Temporal logic
- Modal logic
- Linear logic

Think of these as a set of tools that can be applied to different situations. The right one to use depends on which concept we are interested in modeling. Classical logic can be regarded as everyday logic, and fits with most of our daily reasoning; it is based wholly on the notion of truth. Intuitionistic logic on the other hand is more about proof than truth—a sentence is true when we can provide a constructive proof of the statement. The example that one finds to distinguish intuitionistic logic from classical logic is the sentence ‘A or not A’. Classically, this is always true, since whatever A is (true or false), the resulting sentence is true. In intuitionistic logic we are unable to prove this sentence, since in general it is not the case that we can find a proof of either ‘A’ or ‘not A’. Temporal logic captures the notion of time in a proof and allows us to express ideas such that sentences become true at a certain point in time. More generally, modal logics explore alternative modes of truth, where truth may depend on the state we are in, whether it be time, a set of beliefs or the current state of a machine. Linear logic is a refined logic that captures the notion of resources in a proof. A proof of a sentence in linear logic requires that we have
correctly used all the assumptions. All of these logics play a major rôle in computer science, for example: intuitionistic logic connects strongly with type systems for functional programming; temporal logic is used for concurrent system specification and verification; modal logic for artificial intelligence; and linear logic is currently being used to study the dynamics of the evaluation of programs.

Here we are not so concerned with modeling real life situations in a logical system, but more on understanding the logical systems themselves. For example, we might have a rule that says: under the assumption that the atomic sentences \( A \) and \( B \) are true, we can deduce that the sentence \( "A \text{ and } B" \) is true; why \( A \) and \( B \) are true is not part of our reasoning. To express sentences like this we use a particular syntax for the logical connectives. In general we will write \( \land \) for "and", \( \lor \) for "or", \( \neg \) for "not", \( \Rightarrow \) for "implies", \( \forall \) for "for all", and \( \exists \) for "there exists". We will also introduce some additional connectives upon demand.

The kind of rules that we desire to reason about such sentences are going to tell us how we can introduce these connectives, and also how to reason about the components of the sentences by eliminating the connectives.

Mathematics gains a lot from a formal syntax. We have seen some examples above that were written in English, but we could do much better in a language that is more suited to express our ideas. Roughly speaking, we want to get rid of words that are not important; just capturing the parts of the sentence that we are interested in. For example, the following equation from set theory, expressing distributivity of intersection over union:

\[
X \cap (Y \cup Z) = (X \cap Y) \cup (X \cap Z)
\]

could be written in English as:

Given three sets \( X, Y \) and \( Z \), then the set obtained by taking the union of the sets \( Y \) and \( Z \) and then the intersection of this set with the set \( X \) is equal to the set obtained by first taking the intersection of the sets \( X, Y \) and \( X, Z \) and then taking the union of these two sets.

The gains are self evident: mathematical syntax is briefer, more concise and more precise (at least when we have understood what the mathematical symbols mean!). The same phenomenon arises in logic too; it is a mathematical discipline. Generally, one can write a logical sentence representing a natural language one by arranging it in a more structured way.

Here is an example of a logical sentence:

\[
\forall n, \text{divisible-by-}2(n) \Rightarrow \text{even}(n)
\]

which represents the natural language sentence:

All numbers that are divisible by 2 are even numbers.
Suppose further that we know that divisible-by-2(10) is true, then we can deduce that even(10). This kind of reasoning is fundamental to logical reasoning. More generally this rule follows the form 'given both $A$ and $A \Rightarrow B$ then we can deduce $B$', and is called *modus ponens*. Rules of this form will generally be written in the form:

$$
\begin{array}{c}
A \\
A \Rightarrow B
\end{array}

\Rightarrow

\begin{array}{c}
B
\end{array}
$$

where we write the assumptions above the line (the things that we know are true) and the conclusion below the line (the things that we can deduce to be true).

There are other kinds of rules like this that allow us to reason. You will find many of these throughout this book. Here is one more to give the flavour of the kinds of rules that we will use:

$$
\begin{array}{c}
\neg B \\
A \Rightarrow B
\end{array}

\Rightarrow

\begin{array}{c}
\neg A
\end{array}
$$

This rule says that if we know that if $A$ is true then $B$ is also true, and we also know that $B$ is not true, then it must be the case that $A$ is also not true. This rule is called *modus tollens*. To understand this rule a little better, we give an example. Suppose we have the following facts: “If it is raining then I am inside”, and “I am outside”. Then the above rule states that we can deduce that it is not raining.

The purpose of a formal system is to allow us to obtain *proofs* of sentences. There are many different proof systems that have been developed to express these kinds of rules and to reason about them in different ways. We will study many of these including Tableaux, Resolution and Natural Deduction. Common to all these methods is the production of a proof of a particular sentence in some logic. The different systems allow us to manipulate the sentence in different ways to construct a proof. It is worth mentioning that proofs themselves are worthy of study. The formal study of proofs is called *proof theory*, which is really the heart of logic, or *the Logic of Logic*. It is essentially a *syntactical* discipline in that we are interested in the structure and properties of manipulation at a syntactical level.

Reasoning in a formal system of course must be founded upon a set of rules that are correct; that is each rule of the formal system must preserve the mathematical truth value of the sentence. For this we need a notion of *semantics*—a mathematical object that we use to interpret the logical system. The property that all the rules are correct (or *valid*) with respect to some semantics is called *soundness* and is one of the most important properties of a proof system.

There are many different kinds of semantics that we can use to model proof systems, and we shall see several of these in this book. Different logics lend themselves towards different kinds of semantics. For example classical logic and truth tables, intuitionistic logic and Heyting (functional) semantics, etc.
There is another property that we may require of our logical system, and that is completeness; that is to say we can prove all the sentences that are true in the system. Having a logical system that is sound and complete with respect to some mathematical model gives us total freedom to use the proof system in any way that we like—it is not possible to apply a rule to change the meaning of the sentence, and (if we can find it!) there is always a proof of a true sentence.

It is therefore not so hard to believe that the process of producing proofs in these frameworks is a mechanical process that can be automated. We can represent logical sentences as data types in a programming language, and express the rules of deduction as functions over these data types. This is the topic of this book—automated deduction. Of course, some proof methods are going to be more appropriate than others, for example, resolution is one of the most successful implementations, and is the basis of the programming language Prolog. We will discuss many proof techniques and mention implementation difficulties.

Logic plays a major rôle in computer science. We can use it for specifications of programs (pre-conditions, post-conditions, loop invariants, etc.) as a way of reasoning about computation and proving certain properties of programs with respect to the specification. Here we are interested in using a much stronger connection between logic and computation where the logic is utilised much more directly. There are two paradigms interfacing logic and computation:

1. Logic Programming

   • Programs are theories (over some base logic, for example Horn clauses).
   • Computation is proof search.

A computer is given a program (information in the form of logical sentences) and a query (a logical sentence) on this information. Computation is then the process of finding a proof of the query; in other words: is there sufficient information in the program to determine this query. The most significant language to follow this paradigm is Prolog.

2. Typed Functional Programming

   • Types of programs are logical formulas.
   • Programs are proofs.
   • Computation is the normalisation (or cut-elimination) process of proofs.

The idea is that a computer is given a program in terms of a proof — a proof object corresponding to a sentence — (following the intuition that “proving is like programming”) and we require as output the normal form of that
Proof Theory and Automated Deduction
Goubault-Larrecq, J.; Mackie, I.
1997, 444 p., Softcover
ISBN: 978-1-4020-0368-4