Chapter 1
Introduction

1.1 The Origins of Mathematical Logic

Logic formalizes valid methods of reasoning. The study of logic was begun by the ancient Greeks whose educational system stressed competence in reasoning and in the use of language. Along with rhetoric and grammar, logic formed part of the *trivium*, the first subjects taught to young people. Rules of logic were classified and named. The most widely known set of rules are the *syllogisms*; here is an example of one form of syllogism:

Premise All rabbits have fur.
Premise Some pets are rabbits.
Conclusion Some pets have fur.

If both premises are true, the rules ensure that the conclusion is true.

Logic must be formalized because reasoning expressed in informal natural language can be flawed. A clever example is the following 'syllogism' given by Smullyan (1978, p. 183):

Premise Some cars rattle.
Premise My car is some car.
Conclusion My car rattles.

The formalization of logic began in the nineteenth century as mathematicians attempted to clarify the foundations of mathematics. One trigger was the discovery of non-Euclidean geometries: replacing Euclid's parallel axiom with another axiom resulted in a different theory of geometry that was just as consistent as that of Euclid. Logical systems—axioms and rules of inference—were developed with the understanding that different sets of axioms would lead to different theorems. The questions investigated included:

Consistency A logical system is consistent if it is impossible to prove both a formula and its negation.
Independence The axioms of a logical system are independent if no axiom can be proved from the others.

DOI 10.1007/978-1-4471-4129-7_1, © Springer-Verlag London 2012

Soundness All theorems that can be proved in the logical system are true.
Completeness All true statements can be proved in the logical system.

Clearly, these questions will only make sense once we have formally defined the central concepts of *truth* and *proof*.

During the first half of the twentieth century, logic became a full-fledged topic of modern mathematics. The framework for research into the foundations of mathematics was called *Hilbert's program*, (named after the great mathematician David Hilbert). His central goal was to prove that mathematics, starting with arithmetic, could be axiomatized in a system that was both consistent and complete. In 1931, Kurt Gödel showed that this goal cannot be achieved: any consistent axiomatic system for arithmetic is incomplete since it contains true statements that cannot be proved within the system.

In the second half of the twentieth century, mathematical logic was applied in computer science and has become one of its most important theoretical foundations. Problems in computer science have led to the development of many new systems of logic that did not exist before or that existed only at the margins of the classical systems. In the remainder of this chapter, we will give an overview of systems of logic relevant to computer science and sketch their applications.

1.2 Propositional Logic

Our first task is to formalize the concept of the *truth* of a statement. Every statement is assigned one of two values, conventionally called *true* and *false* or T and F. These should be considered as arbitrary symbols that could easily be replaced by any other pair of symbols like 1 and 0 or even ♣ and ♠.

Our study of logic commences with the study of *propositional logic* (also called the *propositional calculus*). The *formulas* of the logic are built from *atomic propositions*, which are statements that have no internal structure. Formulas can be combined using *Boolean operators*. These operators have conventional names derived from natural language (*and*, *or*, *implies*), but they are given a formal meaning in the logic. For example, the Boolean operator *and* is defined as the operator that gives the value *true* if and only if applied to two formulas whose values are *true*.

Example 1.1 The statements 'one plus one equals two' and 'Earth is farther from the sun than Venus' are both true statements; therefore, by definition, so is the following statement:

'one plus one equals two' *and* 'Earth is farther from the sun than Venus'.

Since 'Earth is farther from the sun than Mars' is a false statement, so is:

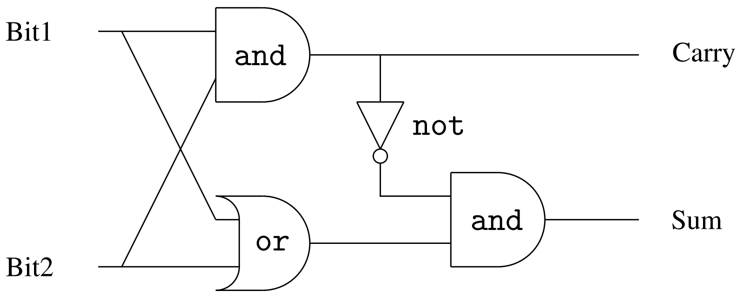
'one plus one equals two' *and* 'Earth is farther from the sun than Mars'. ■

Rules of *syntax* define the legal structure of formulas in propositional logic. The *semantics*—the meaning of formulas—is defined by *interpretations*, which assign one of the *(truth) values* T or F to every atomic proposition. For every legal way that a formula can be constructed, a semantical rule specifies the truth value of the formula based upon the values of its constituents.

Proof is another syntactical concept. A proof is a deduction of a formula from a set of formulas called *axioms* using *rules of inference*. The central theoretical result that we prove is the soundness and completeness of the axiom system: the set of provable formulas is the same as the set of formulas which are always true.

Propositional logic is central to the design of computer hardware because hardware is usually designed with components having two voltage levels that are arbitrarily assigned the symbols 0 and 1. Circuits are described by idealized elements called *logic gates*; for example, an `and`-gate produces the voltage level associated with 1 if and only if both its input terminals are held at this same voltage level.

Example 1.2 Here is a *half-adder* constructed from `and`, `or`- and `not`-gates.



The half-adder adds two one-bit binary numbers and by joining several half-adders we can add binary numbers composed of many bits. ■

Propositional logic is widely used in software, too. The reason is that any program is a finite entity. Mathematicians may consider the natural numbers to be infinite $(0, 1, 2, \ldots)$, but a word of a computer's memory can only store numbers in a finite range. By using an atomic proposition for each bit of a program's state, the meaning of a computation can be expressed as a (very large) formula. Algorithms have been developed to study properties of computations by evaluating properties of formulas in propositional logic.

1.3 First-Order Logic

Propositional logic is not sufficiently expressive for formalizing mathematical theories such as arithmetic. An arithmetic expression such as $x + 2 > y - 1$ is neither true nor false: (a) its truth depends on the values of the *variables* x and y; (b) we need to formalize the meaning of the operators $+$ and $-$ as *functions* that map a pair of numbers to a number; (c) *relational* operators like $>$ must be formalized as mapping pairs of numbers into truth values. The system of logic that can be interpreted by values, functions and relations is called *first-order logic* (also called *predicate logic* or the *predicate calculus*).

The study of the foundations of mathematics emphasized first-order logic, but it has also found applications in computer science, in particular, in the fields of automated theorem proving and logic programming. Can a computer carry out the work of a mathematician? That is, given a set of axioms for, say, number theory, can we write software that will find proofs of known theorems, as well as statements and proofs of new ones? With luck, the computer might even discover a proof of Goldbach's Conjecture, which states that every even number greater than two is the sum of two prime numbers:

$$4 = 2 + 2, \quad 6 = 3 + 3, \quad \ldots,$$
$$100 = 3 + 97, \quad 102 = 5 + 97, \quad 104 = 3 + 101, \quad \ldots.$$

Goldbach's Conjecture has not been proved, though no counterexample has been found even with an extensive computerized search.

Research into automated theorem proving led to a new and efficient method of proving formulas in first-order logic called *resolution*. Certain restrictions of resolution have proved to be so efficient they are the basis of a new type of programming language. Suppose that a theorem prover is capable of proving the following formula:

> Let A be an array of integers. Then there exists an array A' such that the elements of A' are a permutation of those of A, and such that A' is ordered: $A'(i) \leq A'(j)$ for $i < j$.

Suppose, further, that given any specific array A, the theorem prover constructs the array A' which the required properties. Then the formula is a *program* for sorting, and the proof of the formula generates the *result*. The use of theorem provers for computation is called *logic programming*. Logic programming is attractive because it is *declarative*—you just write what you *want* from the computation—as opposed to classical programming languages, where you have to specify in detail *how* the computation is to be carried out.

1.4 Modal and Temporal Logics

A statement need not be absolutely true or false. The statement 'it is raining' is sometimes true and sometimes false. *Modal logics* are used to formalize statements where finer distinctions need to be made than just 'true' or 'false'. Classically, modal logic distinguished between statements that are *necessarily* true and those that are *possibly* true. For example, $1 + 1 = 2$, as a statement about the natural numbers, is necessarily true because of the way the concepts are defined. But any historical statement like 'Napoleon lost the battle of Waterloo' is only possibly true; if circumstances had been different, the outcome of Waterloo might have been different.

Modal logics have turned out to be extremely useful in computer science. We will study a form of modal logic called *temporal logic*, where 'necessarily' is interpreted as *always* and 'possibly' is interpreted as *eventually*. Temporal logic has turned out to be the preferred logic for program verification as described in the following section.

1.5 Program Verification

One of the major applications of logic to computer science is in *program verification*. Software now controls our most critical systems in transportation, medicine, communications and finance, so that it is hard to think of an area in which we are not dependent on the correct functioning of a computerized system. Testing a program can be an ineffective method of verifying the correctness of a program because we test the scenarios that we think will happen and not those that arise unexpectedly. Since a computer program is simply a formal description of a calculation, it can be verified in the same way that a mathematical theorem can be verified using logic.

First, we need to express a *correctness specification* as a formal statement in logic. Temporal logic is widely used for this purpose because it can express the dynamic behavior of program, especially of *reactive* programs like operating systems and real-time systems, which do not compute an result but instead are intended to run indefinitely.

Example 1.3 The property '*always* not deadlocked' is an important correctness specification for operating systems, as is 'if you request to print a document, *eventually* the document will be printed'. ■

Next, we need to formalize the semantics (the meaning) of a program, and, finally, we need a formal system for deducing that the program fulfills a correctness specification. An axiomatic system for temporal logic can be used to prove concurrent programs correct.

For sequential programs, verification is performed using an axiomatic system called *Hoare logic* after its inventor C.A.R. Hoare. Hoare logic assumes that we know the truth of statements of the program's domain like arithmetic; for example, $-(1-x) = (x-1)$ is considered to be an axiom of the logic. There are axioms and rules of inference that concern the structure of the program: assignment statements, loops, and so on. These are used to create a proof that a program fulfills a correctness specification.

Rather than deductively prove the correctness of a program relative to a specification, a *model checker* verifies the truth of a correctness specification in every possible state that can appear during the computation of a program. On a physical computer, there are only a finite number of different states, so this is always possible. The challenge is to make model checking feasible by developing methods and algorithms to deal with the very large number of possible states. Ingenious algorithms and data structures, together with the increasing CPU power and memory of modern computers, have made model checkers into viable tools for program verification.

1.6 Summary

Mathematical logic formalizes reasoning. There are many different systems of logic: propositional logic, first-order logic and modal logic are really families of logic with many variants. Although systems of logic are very different, we approach each logic in a similar manner: We start with their syntax (what constitutes a formula in the logic) and their semantics (how truth values are attributed to a formula). Then we describe the method of *semantic tableaux* for deciding the validity of a formula. This is followed by the description of an axiomatic system for the logic. Along the way, we will look at the applications of the various logics in computer science with emphasis on theorem proving and program verification.

1.7 Further Reading

This book was originally inspired by Raymond M. Smullyan's presentation of logic using semantic tableaux. It is still worthwhile studying Smullyan (1968). A more advanced logic textbook for computer science students is Nerode and Shore (1997); its approach to propositional and first-order logic is similar to ours but it includes chapters on modal and intuitionistic logics and on set theory. It has a useful appendix that provides an overview of the history of logic as well as a comprehensive bibliography. Mendelson (2009) is a classic textbook that is more mathematical in its approach.

Smullyan's books such as Smullyan (1978) will exercise your abilities to think logically! The final section of that book contains an informal presentation of Gödel's incompleteness theorem.

1.8 Exercise

1.1 What is wrong with Smullyan's 'syllogism'?

References

E. Mendelson. *Introduction to Mathematical Logic* (*Fifth Edition*). Chapman & Hall/CRC, 2009.

A. Nerode and R.A. Shore. *Logic for Applications* (*Second Edition*). Springer, 1997.

R.M. Smullyan. *First-Order Logic*. Springer-Verlag, 1968. Reprinted by Dover, 1995.

R.M. Smullyan. *What Is the Name of This Book?—The Riddle of Dracula and Other Logical Puzzles*. Prentice-Hall, 1978.