Preliminaries

What is a logic?

A logic is a language equipped with rules for deducing the truth of one sentence from that of another. Unlike natural languages such as English, Finnish, and Cantonese, a logic is an artificial language having a precisely defined syntax. One purpose for such artificial languages is to avoid the ambiguities and paradoxes that arise in natural languages. Consider the following English sentence.

> Let n be the smallest natural number that cannot be defined in fewer than 20 words.

Since this sentence itself contains fewer than 20 words, it is paradoxical. A logic avoids such pitfalls and streamlines the reasoning process. The above sentence cannot be expressed in the logics we study. This demonstrates the fundamental tradeoff in using logics as opposed to natural languages: to gain precision we necessarily sacrifice expressive power.

In this book, we consider classical logics: primarily first-order logic but also propositional logic, second-order logic and variations of these three logics. Each logic has a notion of *atomic formula*. Every sentence and formula can be constructed from atomic formulas following precise rules. One way that the three logics differ is that, as we proceed from propositional logic to first-order logic to second-order logic, there is an increasing number of rules that allow us to construct increasingly complex formulas from atomic formulas. We are able to express more concepts in each successive logic.

We begin our study with propositional logic in Chapter 1. In the present section, we provide background and prerequisites for our study.

What is logic?

Logic is defined as the study of the principles of reasoning. The study of logics (as defined above) is the part of this study known as symbolic logic. Symbolic logic is a branch of mathematics. Like other areas of mathematics, symbolic logic flourished during the past century. A century ago, the primary aim of symbolic logic was to provide a foundation for mathematics. Today, foundational studies are just one part of symbolic logic. We do not discuss foundational issues in this

book, but rather focus on other areas such as model theory, proof theory, and computability theory. Our goal is to introduce the fundamentals and prepare the reader for further study in any of these related areas of symbolic logic. Symbolic logic views mathematics and computer science from a unique perspective and supplies distinct tools and techniques for the solution of certain problems. We highlight many of the landmark results in logic achieved during the past century.

Symbolic logic is exclusively the subject of this book. Henceforth, when we refer to "logic" we always mean "symbolic logic."

Time complexity

Logic and computer science share a symbiotic relationship. Computers provide a concrete setting for the implementation of logic. Logic provides language and methods for the study of theoretical computer science. The subject of complexity theory demonstrates this relationship well. Complexity theory is the branch of theoretical computer science that classifies problems according to how difficult they are to solve. For example, consider the following problem:

> **The Sum 10 Problem:** Given a finite set of integers, does some subset add up to 10?

This is an example of a decision problem. Given input as specified (in this case, a finite set of integers) a decision problem asks a question to be answered with a "yes" or "no." Suppose, for example, that we are given the following set as input:

$$\{-26, -16, -12, -8, -4, -2, 7, 8, 27\}.$$

The problem is to decide whether or not this set contains a subset of numbers that add up to 10. One way to resolve this problem is to check every subset. Since 10 is not in our set, such a subset must contain more than one number. We can check to see if the sum of any two numbers is 10. We can then check to see if the sum of any three numbers is 10, and so forth. This method will eventually provide the correct answer to the question, but it is not efficient. We have $2^9 = 512$ subsets to check. In general, if the input contains n integers, then there are 2^n subsets to check. If the input set is large, then this is not feasible. If the set contains 23 numbers, then there are more than 8 million subsets to check. Although this is a lot of subsets, this is a relatively simple task for a computer. If, however, there are more than, say, 100 numbers in the input set, then, even for the fastest computer, the time required to check each subset exceeds the lifespan of earth.

Time complexity is concerned with the amount of time it takes to answer a problem. To answer a decision problem, one must produce an algorithm that, given any suitable input, will result in the correct answer of "yes" or "no." An algorithm is a step-by-step procedure. The "amount of time" is measured by how many steps it takes to reach the answer. Of course, the bigger the input, the longer it will take to reach a conclusion. An algorithm is said to be *polynomial-time* if there is some number k so that, given any input of size n, the algorithm reaches its conclusion in fewer than n^k steps. The class of all decision problems that can be solved by a polynomial-time algorithm is denoted by **P**. We said that complexity theory classifies problems according to how difficult they are to solve. The complexity class **P** contains problems that are relatively easy to solve.

To answer the Sum 10 Problem, we gave the following algorithm: check every subset. If some subset adds up to 10, then output "yes." Otherwise, output "no." This algorithm is not polynomial-time. Given input of size n, it takes at least 2^n steps for the algorithm to reach a conclusion and, for any k, $2^n > n^k$ for sufficiently large n. So this decision problem is not necessarily in **P**. It is in another complexity class known as **NP** (nondeterministic polynomial-time). Essentially, a decision problem is in **NP** if a "yes" answer can be obtained in polynomial-time by guessing. For example, suppose we somehow guess that the subset $\{-26, -4, -2, 7, 8, 27\}$ sums up to 10. It is easy to check that this guess is indeed correct. So we quickly obtain the correct output of "yes."

So the Sum 10 Problem is in **NP**. It is not known whether it is in **P**. The algorithm we gave is not polynomial-time, but perhaps there exists a better algorithm for this problem. In fact, maybe every problem in **NP** is in **P**. The question of whether $\mathbf{P} = \mathbf{NP}$ is not only one of the big questions of complexity theory, it is one of the most famous unanswered questions of mathematics. The Clay Institute of Mathematics has chosen this as one of its seven Millennium Problems. The Clay Institute has put a bounty of one million dollars on the solution for each of these problems.

What does this have to do with logic? Complexity theory will be a recurring theme throughout this book. From the outset, we will see decision problems that naturally arise in the study of logic. For example, we may want to know whether or not a given sentence of propositional logic is sometimes true (likewise, we may ask if the sentence is always true or never true). This decision problem, which we shall call the *Satisfiability Problem*, is in **NP**. It is not known whether it is in **P**. In Chapter 7, we show that the Satisfiability Problem is **NP**-complete. This means that if this problem is in **P**, then so is every problem in **NP**. So if we can find a polynomial time algorithm for determining whether or not a given sentence of propositional logic is sometimes true, or if we can show that no such algorithm exists, then we will resolve the $\mathbf{P} = \mathbf{NP}$ problem.

In Chapter 10, we turn this relationship between complexity and logic on its head. We show that, in a certain setting (namely, graph theory) the complexity classes of **P** and **NP** (and others) can be defined as logics. For example, Fagin's Theorem states that (for graphs) **NP** contains precisely those decision problems that can be expressed in second-order existential logic. So the **P** = **NP** problem and related questions can be rephrased as questions of whether or not two logics are equivalent.

From the point of view of a mathematician, this makes the **P** = **NP** problem more precise. Our above definitions of **P** and **NP** may seem hazy. After all, our definition of these complexity classes depends on the notion of a "step" of an algorithm. Although we could (and will) precisely define what constitutes a "step," we utterly avoid this issue by defining these classes as logics. From the point of view of a computer scientist, on the other hand, the relationship between logics and complexity classes justifies the study of logics. The fact that the familiar complexity classes arise from these logics is evidence that these logics are natural objects to study.

Clearly, we are getting ahead of ourselves. Fagin's Theorem is not mentioned until the final chapter. In fact, no prior knowledge of complexity theory is assumed in this book. Some prior knowledge of algorithms may be helpful, but is not required. We do assume that the reader is familiar with sets, relations, and functions. Before beginning our study, we briefly review these topics.

Sets and structures

We assume that the reader is familiar with the fundamental notion of a *set*. We use standard set notation:

$x \in A$ means x is an element of set A,
$x \notin A$ means x is not an element of A,
$\emptyset$ denotes the unique set containing no elements,
$A \subset B$ means every element of set A is also an element of set B,
$A \cup B$ denotes the union of sets A and B,
$A \cap B$ denotes the intersection of sets A and B, and
$A \times B$ denotes the Cartesian product of sets A and B.

Recall that the *union* $A \cup B$ of A and B is the set of elements that are in A or B (including those in both A and B), whereas the *intersection* $A \cap B$ is the set of only those elements that are in both A and B. The Cartesian product $A \times B$ of A and B is the set of ordered pairs (a, b) with $a \in A$ and $b \in B$. We simply write A^2 for $A \times A$. Likewise, for $n > 2$, A^n denotes the Cartesian product of A^{n-1} and A. This is the set of n-tuples $(a_1, a_2, \ldots, a_n)$ with each $a_i \in A$. For convenience, A^1 (the set of 1-tuples) is an alternative notation for A itself.

Example 1 Let $A = \{\alpha, \beta, \gamma\}$ and let $B = \{\beta, \delta, \epsilon\}$. Then

$$A \cup B = \{\alpha, \beta, \gamma, \delta, \epsilon\},$$
$$A \cap B = \{\beta\},$$
$$A \times B = \{(\alpha, \beta), (\alpha, \delta), (\alpha, \epsilon), (\beta, \beta), (\beta, \delta), (\beta, \epsilon), (\gamma, \beta), (\gamma, \delta), (\gamma, \epsilon)\},$$
$$\text{and } B^2 = \{(\beta, \beta), (\beta, \delta), (\beta, \epsilon), (\delta, \beta), (\delta, \delta), (\delta, \epsilon), (\epsilon, \beta), (\epsilon, \delta), (\epsilon, \epsilon)\}.$$

Two sets are equal if and only if they contain the same elements. Put another way, $A = B$ if and only if both $A \subset B$ and $B \subset A$. In particular, the order and repetition of elements within a set do not matter. For example,

$$A = \{\alpha, \beta, \gamma\} = \{\gamma, \beta, \alpha\} = \{\beta, \beta, \alpha, \gamma\} = \{\gamma, \alpha, \beta, \beta, \alpha\}.$$

Note that $A \subset B$ includes the possibility that $A = B$. We say that A is a *proper* subset of B if $A \subset B$ and $A \neq B$ and $A \neq \emptyset$.

A set is essentially a database that has no structure. For an example of a database, suppose that we have a phone book listing 1000 names in alphabetical order along with addresses and phone numbers. Let T be the set containing these names, addresses, and phone numbers. As a set, T is a collection of 3000 elements having no particular order or other relationships. As a database, our phone book is more than merely a set with 3000 entries. The database is a *structure*: a set together with certain relations.

Definition 2 Let A be a set. A **relation** R on A is a subset of A^n (for some natural number n). If $n = 1, 2$, or 3, then the relation R is called **unary**, **binary**, or **ternary** respectively. If n is bigger than 3, then we refer to R as an n-ary relation. The number n is called the **arity** of R.

As a database, our phone book has several relations. There are three types of entries in T: names, numbers, and addresses. Each of these forms a subset of T, and so can be viewed as a unary relation on T. Let N be the set of names in T, P be the set of phone numbers in T, and A be the set of addresses in T. Since a "relation" typically occurs between two or more objects, the phrase "unary relation" is somewhat of an oxymoron. We continue to use this terminology, but point out that a "unary relation" should be viewed as a predicate or an adjective describing elements of the set.

We assume that each name in the phone book corresponds to exactly one phone number and one address. This describes another relation between the elements of T. Let R be the ternary relation consisting of all 3-tuples (x, y, z) of elements in T^3 such that x is a name having phone number y and address z. Yet another relation is the order of the names. The phone book, unlike the set T, is in alphabetical order. Let the symbol $<$ represent this order. If x and y

are elements of N (that is, if they are names in T), then $x < y$ means that x precedes y alphabetically. This order is a binary relation on T. It can be viewed as the subset of T^2 consisting of all ordered pairs (x, y) with $x < y$.

Structures play a primary role in the study of first-order logic (and other logics). They provide a context for determining whether a given sentence of the logic is true or false. First-order structures are formally introduced in Chapter 2. In the previous paragraphs, we have seen our first example of a structure: a phone book. Let D denote the database we have defined. We have

$$D = (T|N, P, A, <, R).$$

The above notation expresses that D is the structure having set T and the five relations N, P, A, $<$, and R on T.

Although a phone book may not seem relevant to mathematics, the objects of mathematical inquiry often can be viewed as structures such as D. Number systems provide familiar examples of infinite structures studied in mathematics. Consider the following sets:

$\mathbb{N}$ denotes the set of natural numbers: $\mathbb{N} = \{1, 2, 3, ...\}$,
$\mathbb{Z}$ denotes the set of integers: $\mathbb{Z} = \{..., -3, -2, -1, 0, 1, 2, 3, ...\}$, and
$\mathbb{Q}$ denotes the set of rational numbers: $\mathbb{Q} = \{a/b|a, b \in \mathbb{Z}\}$.
$\mathbb{R}$ denotes the set of real numbers: $\mathbb{R}$ is the set of all decimal expansions of the form $z.a_1a_2a_3\cdots$ where z and each a_i are integers and $0 \leq a_i \leq 9$.
$\mathbb{C}$ denotes the set of complex numbers: $\mathbb{C} = \{a+bi|a, b \in \mathbb{R}\}$ where $i = \sqrt{-1}$.

Note that $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, and $\mathbb{C}$ each represents a set. These number systems, however, are more than sets. They have much structure. The structure includes relations (such as $<$ for less than) and functions (such as $+$ for addition). Depending on what our interests are, we may consider these sets with any number of various functions and relations.

The interplay between mathematical structures and formal languages is the subject of model theory. First-order logic, containing various relations and functions, is the primary language of model theory. We study model theory in Chapters 4–6. As we shall see, the perspective of model theory sheds new light on familiar structures such as the real and complex numbers.

Functions

The notation $f : A \to B$ expresses that f is a function from set A to a subset of set B. This means that, given any $a \in A$ as input, f yields at most one output $f(a) \in B$. It is possible that, given $a \in A$, f yields no output. In this case we say that $f(a)$ is undefined. The set of all $a \in A$ for which f does produce an output is

called the *domain* of f. The *range* of f is the set of all $b \in B$ such that $b = f(a)$ for some $a \in A$. If the range of f is all of B, then the function is said to be *onto* B.

The *graph* of $f\colon A \to B$ is the subset of $A \times B$ consisting of all ordered pairs (a, b) with $f(a) = b$. If A happens to be B^n for some $n \in \mathbb{N}$, then we say that f is a function *on* B and n is the *arity* of f. In this case, the graph of f is an $(n+1)$-ary relation on B. The *inverse graph* of $f\colon A \to B$ is obtained by reversing each ordered pair in the graph of f. That is, (b, a) is in the inverse graph of f if and only if (a, b) is in the graph of f. The inverse graph does not necessarily determine a function. If it does determine a function $f^{-1}\colon B \to A$ (defined by $f^{-1}(b) = a$ if and only if (b, a) is in the inverse graph of f) then f^{-1} is called the *inverse function* of f and f is said to be *one-to-one*.

The concept of a function should be quite familiar to anyone who has completed a course in calculus. As an example, consider the function from $\mathbb{R}$ to $\mathbb{R}$ defined by $h(x) = 3x^2 + 1$. This function is defined by a rule. Put into words, this rule states that, given input x, h squares x, then multiplies it by 3, and then adds 1. This rule allows us to compute $h(0) = 1$, $h(3) = 28$, $h(72) = 15553$, and so forth. In addition to this rule, we must be given two sets. In this example, the real numbers serve as both sets. So h is a unary function on the real numbers. The domain of h is all of $\mathbb{R}$ since, given any x in $\mathbb{R}$, $3x^2 + 1$ is also in $\mathbb{R}$. The function h is not one-to-one since $h(x)$ and $h(-x)$ both equal the same number for any x. Nor is h onto $\mathbb{R}$ since, given $x \in \mathbb{R}$, $h(x) = 3x^2 + 1$ cannot be less than 1.

Other examples of functions are provided by various buttons on any calculator. Scientific calculators have buttons $\boxed{x^2}$, $\boxed{\log x}$, $\boxed{\sin x}$, and so forth. When you put a number into the calculator and then push one of these buttons, the calculator outputs at most one number. This is exactly what is meant by a "function." The key phrase is "at most one." As a nonexample, consider the square root. Given input 4, there are two outputs: 2 and -2. This is not a function. If we restrict the output to the positive square root (as most calculators do), then we do have a function. It is possible to get less than one output: you may get an ERROR message (say you input -35 and then push the $\boxed{\log x}$ button). The domain of a function is the set of inputs for which an ERROR does not occur. We can imagine a calculator that has a button $\boxed{\text{h}}$ for the function h defined in the previous paragraph. When you input 2 and then push $\boxed{\text{h}}$, the output is 13. This is what $\boxed{\text{h}}$ does: it squares 2, multiplies it by 3, and adds 1. Indeed, if we have a programmable calculator we could easily make it compute h at the push of a button.

Intuitively, any function behaves like a calculator button. However, this analogy must not be taken literally. Although calculators provide many examples of familiar functions, most functions cannot be programmed into a calculator.

Definition 3 A function f is **computable** if there exists a computer program that, given input x,

- outputs $f(x)$ if x is in the domain of f, and
- yields no output if x is not in the domain of f.

As we will see in Section 2.5, most functions are not computable. However, it is hard to effectively demonstrate a function that is not computable. How can we uniquely describe a particular function without providing a means for its computation? As we will see, logic provides many examples. Computability theory is a subject of Chapter 7. Odd as it may seem, computability theory studies things that cannot be done by a computer. This subject arose from Gödel's proof of his famous Incompleteness theorems. Proved in 1931, Gödel's theorems rank among the great mathematical achievements of the past century. They imply that there is no computer algorithm to determine whether or not a given statement of arithmetic is true or false. Again, we are getting way ahead of ourselves. We will come to Gödel's theorems (as well as Fagin's theorem) and state them precisely in due time. Let us now end our preliminary ramblings and begin our study of logic.