Chapter 2
Mathematical Preliminaries

"If your only tool is a hammer ... "

This chapter is devoted to reviewing a broad range of mathematical concepts that are central to our approach to developing computation theory. As we develop these concepts, we shall repeatedly observe instances of the following "self-evident truth" (which is what "axiom" means).

The conceptual axiom. *One's ability to think deeply about a complicated concept is always enhanced by having more than one way to think about the concept.*

We shall harvest only small benefits from this axiom within this chapter, but we shall gather an abundant harvest in the remaining chapters.

2.1 Sets and Their Operations

Sets are probably the most basic object of mathematical discourse. We assume, therefore, that the reader knows what a set is and recognizes that some sets are finite, while others are infinite. Sample finite sets are, for example, the set of words in this book or the set of characters in any JAVA program. Some familiar infinite sets that will appear somewhere in our discussions in this book are:[1]

- the set of *nonnegative integers*, which we denote by $\mathbb{N}$,
- the set of *positive integers*, which we denote by $\mathbb{N}^+$,
- the set of *all integers*, which we denote by $\mathbb{Z}$,
- the set of nonnegative *rational numbers*—which are quotients of integers,
- the set of nonnegative *real numbers*—which can be viewed computationally as the set of numbers that admit infinite decimal expansions,

[1] We assume prior familiarity with all of these sets. We include them here just to establish notation and terminology.

A.L. Rosenberg, *The Pillars of Computation Theory*, Universitext,
DOI 10.1007/978-0-387-09639-1_2, © Springer Science+Business Media, LLC 2010

- the set of nonnegative *complex numbers*—which can be viewed as ordered pairs of real numbers,
- the set of *all* finite-length binary strings—i.e., strings of 0's and 1's—which we denote $\{0,1\}^\star$.

When discussing computer-related matters, one often calls each 0 and 1 that occurs in a binary string a *bit*, (for *binary digit*), which leads to the term *"bit string"* as a synonym of "binary string." With respect to general set-related notions, a source such as [34] will supply more than enough background for the topics we discuss in this book. Despite this assumption, we devote this short section to reviewing some basic concepts concerning sets and operations thereon. (Others will appear as needed throughout the book.)

For any finite set S, we denote by $|S|$ the *cardinality* of S, which is the number of elements in S. Finite sets having three special cardinalities are singled out with special names. If $|S| = 0$—i.e., if S has no elements—then we call S the *empty set* and denote it $\emptyset$. The empty set will reappear myriad times throughout the book, as a limiting case of set-defined entities. If $|S| = 1$—i.e., if S has just one element—then we call S a *singleton*; and if $|S| = 2$—i.e., if S has precisely two elements—then we call S a *doubleton*. In many of our discussions throughout the book, the sets of interest will be subsets of some fixed "universal" set U.

> We use the term "universal" as in "universe of discourse," not in the self-referencing sense of a set that contains all other sets. (Bertrand Russell has shown us in [91] [Chapter X, section 100] that the latter notion leads to mind-bending paradoxes.)

Two universal sets that will appear often are the two sample infinite sets mentioned earlier, $\mathbb{N}$ and $\{0,1\}^\star$. Given a universal set U and a *subset* $S \subseteq U$ (the notation meaning that every element of S—if there are any—is also an element of U), we note that the set inequalities

$$\emptyset \subseteq S \subseteq U$$

always hold.

It is often useful to have a convenient term and notation for *the set of all subsets of a set S*. This bigger set—it contains $2^{|S|}$ elements when S is finite—is denoted by $\mathscr{P}(S)$ and is called the *power set* of S.[2] You should satisfy yourself that the biggest and smallest elements of $\mathscr{P}(S)$ are, respectively, the set S itself and the empty set $\emptyset$.

> Let's pause for a moment. *Why does the power set $\mathscr{P}(S)$ of a finite set S contain $2^{|S|}$ elements?*
>
> The **conceptual axiom** will help us answer this question. We begin by taking an arbitrary finite set S—say of n elements—and laying its elements out in a line. We thereby establish a correspondence between S's elements and positive integers: there is the first element, which we associate with the integer 1, the second element, which we associate with the integer 2, and so on, until the last element along the line gets associated with the integer n.
>
> Next, let's note that we can specify any subset S' of S by specifying a length-n *binary string*, i.e., a string of 0's and 1's. The translation is as follows. If an element s of S appears in the subset S', then we look at the integer we have associated with s (via our linearization of S),

[2] The name "power set" arises from the relative cardinalities of S and $\mathscr{P}(S)$ for finite S.

and we set the corresponding bit-position of our binary string to 1; otherwise, we set this bit-position to 0. In this way, we get a distinct subset of S for each distinct binary string, and a distinct binary string for each distinct subset of S. In particular: *the number of length-n binary strings is the same as the number of elements in the power set of S!*

> The binary string that we have constructed to represent each set of integers $N \subseteq \{0,1,\ldots,n-1\}$ is called the *(length-n) characteristic vector of the set N*. Of course, the finite set N has characteristic vectors of all finite lengths. Generalizing this idea, *every* set of integers $N \subseteq \mathbb{N}$, whether finite or infinite, has an *infinite* characteristic vector, which is formed in precisely the same way as are finite characteristic vectors, but now using the set $\mathbb{N}$ as the base set.

We are making progress, but let's look at an example before pressing onward. Let us focus on the set $S = \{a,b,c\}$. Just to make life more interesting, let us lay S's elements out in the order b,a,c, so that b has associated integer 1, a has associated integer 2, and c has associated integer 3. We depict the elements of $\mathscr{P}(S)$ and the corresponding binary strings in the following table.

Binary string	Set of integers	Subset of S
000	$\emptyset$	$\emptyset$
001	$\{3\}$	$\{c\}$
010	$\{2\}$	$\{a\}$
011	$\{2,3\}$	$\{a,c\}$
100	$\{1\}$	$\{b\}$
101	$\{1,3\}$	$\{b,c\}$
110	$\{1,2\}$	$\{a,b\}$
111	$\{1,2,3\}$	$\{a,b,c\} = S$

So, we need now only establish that there are 2^n binary strings of length n. This is accomplished most simply by noting that there are always twice as many binary strings of length n as there are of length $n-1$. This is because we can form the set of binary strings of length n by taking the set A of binary strings of length $n-1$, duplicating A to obtain two equinumerous sets A_1 and A_2, and appending 0 to every string in A_1 and appending 1 to every string in A_2. The thus-amended sets A_1 and A_2 collectively contain all binary strings of length n.

We now have the desired result.

Given two sets S and T, we denote by:

- $S \times T$ the *direct product* of S and T, which is the set of all ordered pairs whose first coordinate contains an element of S and whose second coordinate contains an element of T.
- $S \cap T$ the *intersection* of S and T, which is the set of elements that occur in *both* S and T.
- $S \cup T$ the *union* of S and T, which is the set of elements that occur in S, or in T, *or in both*. (Because of the "or both" qualifier, this operation is sometimes called *inclusive union*.)
- $S \setminus T$ the *difference* of S and T, which is the set of elements that occur in S but not in T. (Particularly in the United States, one often finds "$S - T$" instead of "$S \setminus T$.")

We exemplify the preceding operations with the sets $S = \{a, b, c\}$ and $T = \{c, d\}$. For these sets:

$$\begin{aligned} S \times T &= \{\langle a,c\rangle, \langle b,c\rangle, \langle c,c\rangle, \langle a,d\rangle, \langle b,d\rangle, \langle c,d\rangle\}, \\ S \cap T &= \{c\}, \\ S \cup T &= \{a,b,c,d\}, \\ S \setminus T &= \{a,b\}. \end{aligned}$$

When studying the several contexts that involve a universal set U that all other sets are subsets of, we include also the operation

- $\overline{T} = U \setminus T$, the *complement* of T (relative to the universal set U).
  For instance, the set of odd positive integers is the complement of the set of even positive integers, relative to the set of all positive integers.

We note a number of basic identities involving sets and operations on them. Working on verifying them will cement your understanding:

- $S \setminus T = S \cap \overline{T}$,
- If $S \subseteq T$, then
  1. $S \setminus T = \emptyset$,
  2. $S \cap T = S$,
  3. $S \cup T = T$.

Note, in particular, that[3]

$$[S = T] \text{ iff } \Big[[S \subseteq T] \text{ and } [T \subseteq S]\Big] \text{ iff } \Big[(S \setminus T) \cup (T \setminus S) = \emptyset\Big].$$

The operations union, intersection, and complementation—and operations formed from them, such as set difference—are usually called the *Boolean (set) operations*, acknowledging the seminal work of the nineteenth-century English mathematician George Boole.[4] There are several important identities involving the Boolean set operations. Among the most frequently invoked are the two "laws" attributed to the nineteenth-century French mathematician Auguste De Morgan:

$$\text{For all sets } S \text{ and } T\text{:} \quad \begin{cases} \overline{S \cup T} = \overline{S} \cap \overline{T}, \\ \overline{S \cap T} = \overline{S} \cup \overline{T}. \end{cases} \tag{2.1}$$

While we have focused here on Boolean operations on *sets*, there are "logical" analogues of these operations for logical sentences and their logical "truth values" 0 and 1:

---

[3] "iff" abbreviates the common mathematical phrase, "if and only if."

[4] One often encounters the lowercase adjective "boolean." Such is the price of fame.

- The logical analogue of complementation is (logical) not, which we shall denote by an overline;[5] e.g., $[\overline{0} = 1]$, and $[\overline{1} = 0]$.
- The logical analogue of union is (logical) or, which is also called *disjunction* or *logical sum*. Texts often denote "or" in expressions by "$\vee$"; e.g., $[X \vee Y = 1]$ iff $[X = 1]$ or $[Y = 1]$ or both.
- The logical analogue of intersection is (logical) and, which is also called *conjunction* or *logical product*. Texts often denote "and" in expressions by "$\wedge$"; e.g., $[X \wedge Y = 1]$ iff both $[X = 1]$ and $[Y = 1]$

We end this section with a set-theoretic definition that recurs often throughout our study. Let $\mathscr{C}$ be any (finite or infinite) collection of sets, and let S and T be two elements of $\mathscr{C}$. (Note that $\mathscr{C}$ is a set whose elements are sets.) Focus, just for example, on the set-theoretic operation of intersection; you should be able to extrapolate easily to other operations. We say that $\mathscr{C}$ is *closed* under intersection if whenever sets S and T (which could be the same set) both belong to $\mathscr{C}$, the set $S \cap T$ also belongs to $\mathscr{C}$. As one instance of the desired extrapolation, $\mathscr{C}$'s being closed under union would mean that the set $S \cup T$ belongs to $\mathscr{C}$.

2.2 Binary Relations

2.2.1 The Formal Notion of Binary Relation

Given sets S and T, a *relation on S and T* (in that order) is any subset

$$R \subseteq S \times T.$$

When $S = T$, we often call R a *binary relation on (the set) S* ("*binary*" because there are *two* sets being related). Relations are so common that we use them in every aspect of our lives without even noticing them. The relations "equal," "less than," and "greater than or equal to" are simple examples of binary relations on the integers. These same three relations apply also to other familiar number systems such as the rational and real numbers; only "equal," though, holds (in the natural way) for the complex numbers. Some subset of the three relations "is a parent of," "is a child of," and "is a sibling of" probably are binary relations on (the set of people constituting) your family. To mention just one relation with distinct sets S and T, the relation "A is taking course X" is a relation on (the set of all students) $\times$ (the set of all courses).

We shall see later (Section 7.1) that there is a formal sense in which binary relations are all we ever need consider: 3-set (*ternary*) relations—which are subsets of $S_1 \times S_2 \times S_3$—and 4-set (*quaternary*) relations—which are subsets of $S_1 \times S_2 \times S_3 \times S_4$—and so on (for any finite "arity"), can all be expressed as binary relations of binary relations ... of binary relations. As examples: For ternary relations, we can

[5] Context will always make it clear when we are talking about set complementation and when we are talking about logical not.

replace any subset R of $S_1 \times S_2 \times S_3$ by the obvious corresponding subset R' of $S_1 \times (S_2 \times S_3)$: for each element $\langle s_1, s_2, s_3 \rangle$ of R, the corresponding element of R' is $\langle s_1, \langle s_2, s_3 \rangle \rangle$. Similarly, for quaternary relations, we can replace any subset R'' of $S_1 \times S_2 \times S_3 \times S_4$ by the obvious corresponding subset R''' of $S_1 \times (S_2 \times (S_3 \times S_4))$: for each element $\langle s_1, s_2, s_3, s_4 \rangle$ of R'', the corresponding element of R''' is $\langle s_1, \langle s_2, \langle s_3, s_4 \rangle \rangle \rangle$.

> You should convince yourself that we could achieve the desired correspondence also by replacing $S_1 \times (S_2 \times S_3)$ with $(S_1 \times S_2) \times S_3$ and by replacing $S_1 \times S_2 \times S_3 \times S_4$ by either $((S_1 \times S_2) \times S_3) \times S_4$ or $(S_1 \times S_2) \times (S_3 \times S_4)$.

By convention, with a binary relation $R \subseteq S \times T$, we often write "sRt" in place of the more conservative "$\langle s,t \rangle \in R$." For instance, in "real life," we write "$5 < 7$" rather than the strange-looking (but formally correct) "$\langle 5,7 \rangle \in <$."

The following operation on relations occurs in many guises, in almost all mathematical theories. Let P and P' be binary relations on a set S. The *composition* of P and P' (in that order) is the relation

$$P'' \stackrel{\text{def}}{=} \Big\{ \langle s,u \rangle \in S \times S \mid (\exists t \in S) \Big[[sPt] \text{ and } [tP'u] \Big] \Big\}.$$

(Note how we have used both of our notational conventions for relations here. Note also a new notational device that will recur frequently throughout the book: We use the compound symbol "$\stackrel{\text{def}}{=}$" as a shorthand for introducing notation. The sentence "$X \stackrel{\text{def}}{=} Y$" should be read "$X$ is, by definition, Y.")

There are two special classes of binary relations that play such a central role in computation theory—and elsewhere!—that we must single them out immediately, in the next two subsections.

2.2.2 Equivalence Relations

A binary relation R on a set S is an *equivalence relation* if it enjoys the following three properties:

1. R is *reflexive:* for all $s \in S$, we have sRs.
2. R is *symmetric:* for all $s, s' \in S$, we have sRs' whenever $s'Rs$.
3. R is *transitive:* for all $s, s', s'' \in S$, whenever we have sRs' and $s'Rs''$, we also have sRs''.

Sample familiar equivalence relations are:

- The equality relation, $=$, on a set S which relates each $s \in S$ with itself but with no other element of S.
- The relations $\equiv_{12}$ and $\equiv_{24}$ on integers, where[6]

[6] As usual, $|x|$ is the *absolute value*, or, *magnitude* of the number x. That is, if $x \geq 0$, then $|x| = x$; if $x < 0$, then $|x| = -x$.

1. $n_1 \equiv_{12} n_2$ if and only if $|n_1 - n_2|$ is divisible by 12.
2. $n_1 \equiv_{24} n_2$ if and only if $|n_1 - n_2|$ is divisible by 24.

We use relation $\equiv_{12}$ (without formally knowing it) whenever we tell time using a 12-hour clock and relation $\equiv_{24}$ whenever we tell time using a 24-hour clock.

Closely related to the notion of an equivalence relation on a set S is the notion of a *partition* of S. A partition of S is a nonempty collection of subsets $S_1, S_2, \ldots$ of S that are

1. *mutually exclusive:* for distinct indices i and j, $S_i \cap S_j = \emptyset$;
2. *collectively exhaustive:* $S_1 \cup S_2 \cup \cdots = S$.

We call each set S_i a *block* of the partition.

One verifies as follows that *a partition of a set S and an equivalence relation on S are just two ways of looking at the same concept*. To see this, we note the following.

Getting an equivalence relation from a partition. Given any partition $S_1, S_2, \ldots$ of a set S, define the following relation R on S:

sRs' if and only if s and s' belong to the same block of the partition.

Relation R is an equivalence relation on S. To wit, R is reflexive, symmetric, and transitive because collective exhaustiveness ensures that each $s \in S$ belongs to some block of the partition, while mutual exclusivity ensures that it belongs to only one block.

Getting a partition from an equivalence relation. For the converse, focus on any equivalence relation R on a set S. For each $s \in S$, denote by $[s]_R$ the set

$$[s]_R \stackrel{\text{def}}{=} \{s' \in S \mid sRs'\};$$

we call $[s]_R$ *the equivalence class of s under relation R*.

The equivalence classes under R form a partition of S. To wit: R's reflexivity ensures that the equivalence classes collectively exhaust S; R's symmetry and transitivity ensure that equivalence classes are mutually disjoint.

The *index* of the equivalence relation R is its number of classes—which can be finite or infinite.

Let[7] $\equiv_1$ and $\equiv_2$ be two equivalence relations on a set S. We say that the relation $\equiv_1$ *is a refinement of* (or *refines*) the relation $\equiv_2$ just when each block of $\equiv_1$ is a subset of some block of $\equiv_2$. A couple of basic facts:

- The equality relation, $=$, on S refines every equivalence relation on S. (In this sense, it is the finest equivalence relation on S.)
- Say that the equivalence relation $\equiv_1$ refines the equivalence relation $\equiv_2$ and that $\equiv_2$ has finite index I_2. Then either $\equiv_1$ also has finite index $I_1 \geq I_2$, or $\equiv_1$ has infinite index.

[7] Conforming to common usage, we typically use the symbol $\equiv$, possibly with an embellishing subscript, to denote an equivalence relation.

2.3 Functions

One learns early in school that a function from a set A to a set B is a rule that assigns a unique value from B to every value from A. Yet, as one grows in (mathematical) sophistication, one finds that this notion of function is more restrictive than necessary. A simple example will illustrate our point. Our first example concerns division. We learn that division, like multiplication, is a function that assigns a number to a given pair of numbers. Yet we are warned almost immediately not to "divide by 0": The quotient upon division by 0 is "undefined." So, division is not quite a function as envisioned in the definition that begins this section. Indeed, in contrast to an expression such as "$4 \div 2$," which should lead to the result 2 in any programming environment,[8] expressions such as "$4 \div 0$" will lead to wildly different results in different programming environments. How can one deal with this situation? As presenters of computation theory, we are going to use an approach that is quite distinct from those of programming environments. We are going to broaden the definition of "function" in a way that behaves like the definition that begins this section in "well-behaved" situations and that extends the notion in an intellectually consistent way within "ill-behaved" situations. Let us begin to get formal.

A *(partial) function from set S to set T* is a relation $F \subseteq S \times T$ that is *single-valued:* for each $s \in S$, there is *at most* one $t \in T$ such that sFt. We traditionally write "$F : S \rightarrow T$" as shorthand for the assertion, "F is a function from the set S to the set T"; we also traditionally write "$F(s) = t$" for the more conservative "sFt." (The single-valuedness of F makes the nonconservative notation safe.) We often call the set S the *source (set)* and T the *target (set)* for function F. When there is always a (perforce, unique) $t \in T$ for each $s \in S$, then we call F a *total* function. Note that our terminology is a bit unexpected: *Every total function is a partial function;* that is, "partial" is the generic term, and "total" is a special case.

You may be surprised that we make partial functions our default domain of discourse. This is because most of the functions you deal with daily are *total* functions. Our mathematical ancestors had to do some fancy footwork in order to make your world so neat. Their choreography took two complementary forms.

1. They expanded the target set T on numerous occasions. As just two instances:
   - They appended both 0 and the negative integers to the preexisting positive integers[9] in order to make subtraction a total function.
   - They appended the rationals to the preexisting integers in order to make division (by nonzero numbers!) a total function.

   The irrational algebraic numbers, the nonalgebraic real numbers, and the nonreal complex numbers were similarly appended, in turn, to our number system in order to make certain (more complicated) functions total.

[8] We are, of course, ignoring demons such as round-off error.

[9] The great mathematician Leopold Kronecker said, "God made the integers, all else is the work of man"; cf. [3]. Kronecker was referring, of course, to the *positive* integers.

2. They adapted the function. In programming languages, in particular, undefinedness is anathema, so such languages typically have ways of making functions total, via devices such as "integer division" (so that odd integers can be "divided by 2") as well as various ploys for accommodating "division by 0."

We are going to be less pragmatic than our ancestors, because computation theory is traditionally a theory of functions on nonnegative integers (or, as we shall see, some transparent encoding thereof). The price for such "pureness" is that we must allow functions to be undefined on some arguments. Simple examples of such *nontotal* functions are "division by 2" and "taking square roots." Both of these functions are defined only on subsets of the positive integers (the even integers and the perfect squares, respectively).

Three special classes of functions merit explicit mention. For each, we give both a down-to-earth name and a more scholarly Latinate one.

A function $F : S \to T$ is:

1. *one-to-one* (or *injective*) if for each $t \in T$, there is at most one $s \in S$ such that $F(s) = t$;
   Example: "multiplication by 2" is injective; "integer division by 2" is not (because, e.g., 3 and 2 yield the same answer).
   An injective function F is called an *injection*.
2. *onto* (or *surjective*) if for each $t \in T$, there is at least one $s \in S$ such that $F(s) = t$;
   Example: "subtraction of 1" is surjective, as is "taking the square root"; "addition of 1" is not (because, e.g., 0 is never the sum), and "squaring" is not (because, e.g., 2 is not the square of any integer).
   A surjective function F is called a *surjection*.
3. *one-to-one, onto* (or *bijective*) if for each $t \in T$, there is precisely one $s \in S$ such that $F(s) = t$.
   Example: The (total) function $F : \{0,1\}^\star \to \{0,1\}^\star$ defined by:

$$(\forall w \in \{0,1\}^\star)\ F(w) \ = \ (\text{the reversal of } w)$$

   is a bijection. The (total) function $F' : \{0,1\}^\star \to \mathbb{N}$ defined by

$$(\forall w \in \{0,1\}^\star)\ F(w) \ = \ (\text{the integer that is represented by } w \text{ viewed as a numeral})$$

   is *not* a bijection, due to the possibility of leading 0's.

A *numeral* is a sequence of digits that is the "name" of a number. The numerical value of a numeral x depends on the *number base*, which is a positive integer $b > 1$ that is used to create x. Much of our focus will be on *binary* numerals—which are binary strings—for which the base is $b = 2$. For a general number base b, the integer denoted by the numeral $\beta_n\beta_{n-1}\ldots\beta_1\beta_0$, where each $\beta_i \in \{0,1,\ldots,b-1\}$, is

$$\sum_{i=0}^{n} \beta_i b^i.$$

We say that bit β_i has *lower order* in the numeral than does β_{i+1}, because β_i is multiplied by b^i in evaluating the numeral, whereas β_{i+1} is multiplied by b^{i+1}.

A bijective function F is called a *bijection*.

2.4 Formal Languages

2.4.1 The Notion of Language in Computation Theory

Let Σ be a finite set of (atomic) symbols. Reflecting the linguistic antecedents of computation theory (one of the theory's many ancestors), we often call the set Σ an **alphabet** and its constituent symbols **letters**. For each nonnegative integer k, we denote by Σ^k the set of all length-k strings—or sequences—of elements of Σ. For instance, if $\Sigma = \{a,b\}$, then:

$$\begin{aligned}
\Sigma^0 &= \{\varepsilon\} \ \ (\varepsilon \text{ is the } \textit{null string:} \text{ the unique string of length } 0),\\
\Sigma^1 &= \Sigma = \{a,b\},\\
\Sigma^2 &= \{aa,ab,ba,bb\},\\
\Sigma^3 &= \{aaa,aab,aba,abb,baa,bab,bba,bbb\}.
\end{aligned}$$

We denote by $\Sigma^\star$ *the set of all finite-length strings of elements of* Σ; symbolically,

$$\Sigma^\star = \bigcup_{k\in\mathbb{N}} \Sigma^k.$$

Again nodding to the theory's linguistic antecedents, we often call elements of $\Sigma^\star$ *words*, although we also often call them *strings*.

Notes. (a) If $\Sigma \neq \emptyset$, then $\Sigma^\star$ is infinite.

(b) Because $\Sigma^0 \subseteq \Sigma^\star$, $\Sigma^\star$ is never empty. Indeed, $\Sigma^\star$ is finite iff $\Sigma = \emptyset$, in which case $\Sigma^\star$ contains the single word ε.

(c) Be careful when reasoning about the null string ε (just as you should be careful when reasoning about the null list as a data structure). Specifically, despite ε's lack of letters, it *is* an object, so, for instance, the set $\{\varepsilon\}$ is *not* empty.

(d) The alphabet $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ is a *set*; hence, it has no *intrinsic* order. However, in many situations, Σ is endowed with an *extrinsic* order. For instance, if Σ is the Latin alphabet, then we all "know" that "a" precedes "b," which precedes "c," and so on. Similarly, if $\Sigma = \{0,1\}$, then we all "know" that "0" precedes "1." For such ordered alphabets, there is the important notion of *lexicographic order,* which is a total order on $\Sigma^\star$. Given any two words from $\Sigma^\star$,

$$x = \sigma'_1\sigma'_2\cdots\sigma'_k \quad \text{and} \quad y = \sigma''_1\sigma''_2\cdots\sigma''_\ell,$$

we say that *x precedes y in lexicographic order* precisely when one of the following holds:

- x is a *proper prefix* of y (meaning that $y = xz$ for some nonnull $z \in \Sigma^\star$);
- there exists an index $i \leq \min(k,\ell)$ such that
  - $\sigma'_j = \sigma''_j$ for all $j < i$
  - $\sigma'_i < \sigma''_i$ in the extrinsic order on Σ.

A **language over** the alphabet Σ is *any* subset $L \subseteq \Sigma^\star$.

A language L over Σ can be as "small" as $\emptyset$ or as "big" as $\Sigma^\star$, since, being a set, L satisfies

$$\emptyset \subseteq L \subseteq \Sigma^\star.$$

We denote by $\ell(w)$ the *length* of the word $w \in \Sigma^\star$. Hence, $\ell(\varepsilon) = 0$, and $\ell(w) = k$ for all $w \in \Sigma^k$.

The *concatenation* of words $x \in \Sigma^\star$ and $y \in \Sigma^\star$, which we denote by juxtaposition of x and y—namely, xy—is obtained by appending the string y after the string x. For instance, given two strings $x = 01001$ and $y = 110111$ over the alphabet $\{0,1\}$, the concatenation of x and y is the string $xy = 01001110111$. Occasionally—but only occasionally—for emphasis, we actually insert an operation symbol to denote concatenation, by writing $x \cdot y$ in place of xy.

The operation of concatenation is often called the *complex product* within an algebraic setting. In our context, the underlying algebra is the so-called *free semigroup* over the alphabet Σ, which is just an esoteric way of talking about the semigroup of words over Σ, viewing concatenation as a type of multiplication.

The operation of concatenation is *associative*, which means that for all strings x, y, and z from $\Sigma^\star$, we have

$$x \cdot (y \cdot z) \;=\; (x \cdot y) \cdot z.$$

We leave the inductive argument that establishes this fact to the reader.

Associativity allows us to write long expressions without parentheses. We have been doing this "forever" with binary operations such as addition and multiplication. We are now just noting that we can do this also with this new, string-oriented, type of multiplication.

Equivalence relations on $\Sigma^\star$, specifically "right-invariant" ones, cast a broad shadow in the theory.

An equivalence relation $\equiv$ on $\Sigma^\star$ is **right-invariant** if for all $z \in \Sigma^\star$, $[xz \equiv yz]$ whenever $[x \equiv y]$.

Two simple examples illustrate right-invariance. (1) Consider first the finest equivalence relation $\equiv_1$, namely, equality:

$$[x \equiv_1 y] \text{ if and only if } [x = y].$$

This relation is right-invariant because if x and y are identical, then appending the same string z to both leaves you with identical strings, xz and yz. (2) Consider next the equivalence relation $\equiv_2$ that "identifies" binary strings that have the same number of 1's:

$$[x \equiv_2 y] \text{ if and only if the number of 1's in } x \text{ equals the the number of 1's in } y$$

(You should prove that $\equiv_2$ is indeed an equivalence relation.) This relation is right-invariant because if x and y share the same number of 1's, then so also do xz and yz, no matter what string z is.

A major focus in our development of the theory will be the following specific right-invariant equivalence relation on $\Sigma^\star$, which is defined in terms of a given language $L \subseteq \Sigma^\star$:

$$\text{For all } x, y \in \Sigma^\star : \quad [x \equiv_L y] \ \text{ iff } \ (\forall z \in \Sigma^\star)[[xz \in L] \Leftrightarrow [yz \in L]]. \tag{2.2}$$

The following important result is left as an exercise.

Lemma 2.1. *For any alphabet Σ and language $L \subseteq \Sigma^\star$, the equivalence relation $\equiv_L$ is right-invariant.*

2.4.2 Languages as Metaphors for Computational Problems

This section is devoted to an important example of how one can think about computations in nonobvious ways—a somewhat subtler instance of the **conceptual axiom** than we have observed to this point.

Every language $L \subseteq \Sigma^\star$ has an associated function that allows us to step back and forth between the world of functions and the world of languages.

We may initially be a bit uncomfortable hopping in this way between formal notions that are quite unrelated in day-to-day discourse. However, the historical antecedents of computation theory more or less force us to, especially if we want access to primary sources in the development of the theory.

The *characteristic function* of the set/language L is the function κ_L defined as follows:

$$(\forall x \in \Sigma^\star) : \ \kappa_L(x) \ = \ \begin{cases} 1 \text{ if } \ x \in L, \\ 0 \text{ if } \ x \notin L. \end{cases}$$

Dually, every function $f : \Sigma^\star \to \{0, 1\}$ has an associated language L_f defined as follows:

$$L_f \ = \ \{x \in \Sigma^\star \mid f(x) = 1\}.$$

One can study a large range of computational issues involving two-valued functions by focusing on the languages associated with the functions; and one can study a large range of computational issues involving languages by focusing on the languages' characteristic functions. One thus finds three distinct notions talked about interchangeably within the theory:

1. a *language* L;
2. the *computational problem*: to compute L's characteristic function;
3. the *system property*: to decide, given $x \in \Sigma^\star$, whether $x \in L$.

Interestingly, we shall encounter situations in which we shall be able to compute only L's *semicharacteristic function* κ'_L, which is a partial function that tells us when a given $x \in \Sigma^\star$ belongs to L but gives no response when $x \notin L$:

$$(\forall x \in \Sigma^\star): \ \kappa'_L(x) \ = \ \begin{cases} 1 \ \text{if} \ \ x \in L, \\ \text{undefined if} \ \ x \notin L. \end{cases}$$

Alan M. Turing's world-changing demonstration of the existence of computational problems that cannot be solved algorithmically [104] in fact exhibited a language L whose *semicharacteristic function* is computable but whose *characteristic function* is not.

A more concrete example of the duality between functions and languages involves an arbitrary function

$$g : \{0,1\}^\star \times \{0,1\}^\star \to \{0,1\}^\star. \tag{2.3}$$

(Think of g as being addition or multiplication, for instance.) One often studies the problem of computing g via the following language-recognition problem. We define the language[10] $L(g)$ as follows. $L(g)$ is a language over the alphabet $\Sigma \stackrel{\text{def}}{=} \{0,1\} \times \{0,1\}$ whose letters are *ordered pairs* of bits. For each $n \in \mathbb{N}$, the n-letter word

$$\langle \alpha_0, \beta_0 \rangle \langle \alpha_1, \beta_1 \rangle \cdots \langle \alpha_{n-1}, \beta_{n-1} \rangle$$

in Σ^n belongs to the language $L(g)$ precisely when the nth bit of the bit-string

$$g(\alpha_{n-1} \cdots \alpha_0, \ \beta_{n-1} \cdots \beta_0)$$

is a 1.

Note that we reverse the orders of bit-strings so that the index of a bit-position equals the power of 2 that we use to convert the bit-string to an integer. Using this notational convention, the bit-string $\alpha_{n-1} \cdots \alpha_0$ is the numeral[11] for the integer $\sum_{i=0}^{n-1} \alpha_i 2^i$.

2.5 Graphs and Trees

A *directed graph* (*digraph*, for short) $\mathscr{G}$ is given by a set of *nodes* $\mathscr{N}_\mathscr{G}$ and a set of *arcs* (or *directed edges*) $\mathscr{A}_\mathscr{G}$. Each arc has the form $(u \to v)$, where $u, v \in \mathscr{N}_\mathscr{G}$; we say that this arc goes *from u to v*. A *path* in $\mathscr{G}$ is a sequence of arcs that share adjacent endpoints, as in the following path from node u_1 to node u_n:

$$(u_1 \to u_2), \ (u_2 \to u_3), \ \ldots, \ (u_{n-2} \to u_{n-1}), \ (u_{n-1} \to u_n). \tag{2.4}$$

It is sometimes useful to endow the arcs of a digraph with labels from an alphabet Σ. When so endowed, the path (2.4) would be written

$$(u_1 \stackrel{\lambda_1}{\to} u_2), \ (u_2 \stackrel{\lambda_2}{\to} u_3), \ \ldots, \ (u_{n-2} \stackrel{\lambda_{n-2}}{\to} u_{n-1}), \ (u_{n-1} \stackrel{\lambda_{n-1}}{\to} u_n),$$

[10] We avoid the notation "L_g" to avoid any confusion with languages and their characteristic functions.

[11] Recall that a *numeral* is the string-name for a number.

where the λ_i denote symbols from Σ. If $u_1 = u_n$, then we call the preceding path a *cycle*.

An *undirected graph* is obtained from a directed graph by removing the directionality of the arcs; the thus-beheaded arcs are called *edges*. Whereas we say:

the *arc* (u,v) goes *from* node u *to* node v

we say:

the undirected edge $\{u,v\}$ goes *between* nodes u and v

or, more simply:

the undirected edge $\{u,v\}$*connects* nodes u and v.

Undirected graphs are usually the default concept, in the following sense: *When $\mathscr{G}$ is described as a "graph," with no qualifier "directed" or "undirected," it is understood that $\mathscr{G}$ is an undirected graph.*

One specific genre of digraph merits separate mention: *rooted trees*, which are a class of *acyclic* digraphs. Paths in trees that start at the root are often called *branches*. The *acyclicity* of a tree $\mathscr{T}$ means that for any branch of $\mathscr{T}$ of the form (2.4), we cannot have $u_1 = u_n$, for this would create a cycle. Each rooted tree $\mathscr{T}$ has a designated *root node* $r_{\mathscr{T}} \in \mathscr{N}_{\mathscr{T}}$. A node $u_n \in \mathscr{N}_{\mathscr{T}}$ that resides at the end of a branch (2.4) that starts at $r_{\mathscr{T}}$ (so $u_1 = r_{\mathscr{T}}$) is said to reside at *depth* $n-1$ in $\mathscr{T}$; by convention, $r_{\mathscr{T}}$ is said to reside at depth 0. $\mathscr{T}$'s root $r_{\mathscr{T}}$ has some number (possibly 0) of arcs that go from $r_{\mathscr{T}}$ to its *children,* each of which thus resides at depth 1 in $\mathscr{T}$; in turn, each child has some number of arcs (possibly 0) to its children, and so on. (Think of a family tree.) For each arc $(u \to v) \in A_{\mathscr{T}}$, we call u a *parent* of v, and v a *child* of u, in $\mathscr{T}$; clearly, the depth of each child is one greater than the depth of its parent. Every node of $\mathscr{T}$ except for $r_{\mathscr{T}}$ has precisely one parent; $r_{\mathscr{T}}$ has no parents. A childless node of a tree is a *leaf*. The transitive extensions of the parent and child relations are, respectively, the *ancestor* and *descendant* relations. The *degree* of a node v in a tree is the number of children that the node has, call it c_v. If every nonleaf node in a tree has the same degree c, then we call c the *degree of the tree*.

It is sometimes useful to have a symbolic notation for the ancestor and descendant relations. To this end, we write $(u \Rightarrow v)$ to indicate that node u is an ancestor of node v, or equivalently, that node v is a descendant of node u. If we decide for some reason that we are not interested in really distant descendants of the root of tree $\mathscr{T}$, then we can *truncate* $\mathscr{T}$ at a desired depth d by removing all nodes whose depths exceed d. We thereby obtain the *depth-d prefix* of $\mathscr{T}$. (We encounter in Theorem 13.3 a situation in which we truncate a tree.)

Figure 2.1 depicts an arc-labeled rooted tree $\mathscr{T}$ whose arc labels come from the alphabet $\{a,b\}$. $\mathscr{T}$'s arc-induced relationships are listed in Table 2.1.

2.6 Useful Quantitative Notions

Although our main focus will be on logical relationships among computation-theoretic concepts, we shall now and then have occasion to discuss quantitive concepts. This section reviews a couple of basic definitions involving such concepts.

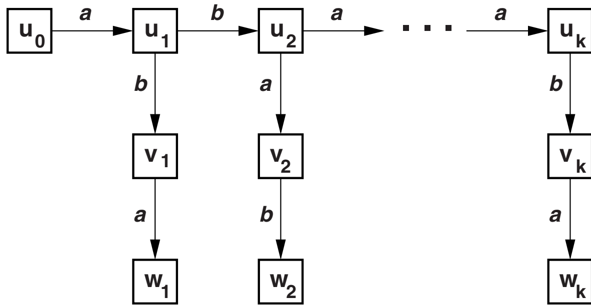

Fig. 2.1 An arc-labeled rooted tree $\mathscr{T}$ whose arc labels come from the alphabet $\{a,b\}$. (Arc labels have no meaning; they are just for illustration.)

The arc-labeled rooted tree $\mathscr{T}$ of Figure 2.1

Node	Children	Parent	Descendants	Ancestors
$r_{\mathscr{T}} = u_0$	u_1	none	$u_1, u_2, \ldots, u_k, v_1, v_2, \ldots, v_k, w_1, w_2, \ldots, w_k$	none
u_1	u_2, v_1	u_0	$u_2, \ldots, u_k, v_1, v_2, \ldots, v_k, w_1, w_2, \ldots, w_k$	u_0
u_2	u_3, v_2	u_1	$u_3, \ldots, u_k, v_2, \ldots, v_k, w_2, \ldots, w_k$	u_0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
u_k	v_k	u_{k-1}	v_k, w_k	$u_0, u_1, \ldots, u_{k-1}$
v_1	w_1	u_1	w_1	u_0, u_1
v_2	w_2	u_2	w_2	u_0, u_1, u_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
v_k	w_k	u_k	w_k	$u_0, u_1, \ldots, u_k$
w_1	none	v_1	none	u_0, u_1, v_1
w_2	none	v_2	none	u_0, u_1, u_2, v_2
w_k	none	v_k	none	$u_0, u_1, \ldots, u_k, v_k$

Table 2.1 A tabular description of the rooted tree $\mathscr{T}$ of Figure 2.1.

Floors and ceilings. Given any real number x, we denote by $\lfloor x \rfloor$ the *floor* (or *integer part*) of x, which is the largest integer that that does not exceed x. Symmetrically, we denote by $\lceil x \rceil$ the *ceiling* of x, which is the smallest integer that is at least as large as x. For any nonnegative integer n,

$$\lfloor n \rfloor \;=\; \lceil n \rceil \;=\; n;$$

for any positive rational number $n + p/q$, where n, p, and q are positive integers and $p < q$,

$$\lfloor n + p/q \rfloor \;=\; n, \;\text{ and }\; \lceil n + p/q \rceil \;=\; n + 1.$$

Logarithms and exponentials. Given any integer $b > 1$ (for "base"), the *base-b logarithm* function $\log_b(\bullet)$ maps positive reals to reals and is defined by either of the following inverse relations:

$$(\forall x > 0)[x \; = \; b^{\log_b x} \; = \; \log_b b^x].$$

Taking logarithms is, thus, inverse to exponentiating. When $b = 2$, a particularly common special case within computation theory, we usually elide the base 2 and just write $\log x$.

Big-O, Big-Ω, and Big-Θ notation. It is convenient to have terminology and a notation that allows us to talk about the rate of growth of one function as measured by the rate of growth of another. We are interested in the exact growth rate, as well as upper and lower bounds on the growth rate. We do have appropriate such language for certain rates of growth. We can talk, for instance, about a linear growth rate or a quadratic rate or an exponential rate, to name just a few—and we get the desired bounds using the prefixes "sub" or "super," as in "subexponential" and "superlinear"—but our repertoire of such terms is quite limited. Mathematicians working in the theory of numbers in the late nineteenth century established a notation that gives us an unlimited repertoire of descriptors for growth rates, via what has come to be called the big-O, big-Ω, and big-Θ notations, which are collectively sometimes called *asymptotic notation*.

Let f and g be total functions from the nonnegative real numbers to the real numbers. We define the following notation:

$f(x) = O(g(x))$	means	$(\exists c > 0)(\exists x^{\#})(\forall x > x^{\#})[f(x) \leq c \cdot g(x)]$
$f(x) = \Omega(g(x))$	means	$f(x) = O(g(x))$,
	i.e.,	$(\exists c > 0)(\exists x^{\#})(\forall x > x^{\#})[f(x) \geq c \cdot g(x)]$
$f(x) = \Theta(g(x))$	means	$[f(x) = O(g(x))]$ and $[f(x) = \Omega(g(x))]$,
	i.e.,	$(\exists c_1 > 0)(\exists c_2 > 0)(\exists x^{\#})(\forall x > x^{\#})[c_1 \cdot g(x) \leq f(x) \leq c_2 \cdot g(x)]$

Note that all three of the rate specifications are *eventual*—or *asymptotic*—because of the "$(\forall x > x^{\#})$" quantifier. Thus, in contrast to the more familiar completely determined assertions such as "$f(x) \leq g(x)$," the assertion "$f(x) = O(g(x))$" has built-in uncertainty, regarding both the size of the scaling factor $c > 0$ and the threshold $x^{\#}$ at which the asserted relationship between $f(x)$ and $g(x)$ kicks in. In mathematical terms, it is best to think about these three asymptotic bounding assertions as establishing *envelopes* for $f(x)$:

- Say that $f(x) = O(g(x))$. If one draws the graphs of the functions $f(x)$ and $c \cdot g(x)$, then as one traces the graphs going rightward (letting x increase), one eventually reaches a point $x^{\#}$ beyond which the graph of $f(x)$ never enters the territory *above* the graph of $c \cdot g(x)$.
- Say that $f(x) = \Omega(g(x))$. This situation is the up-down mirror image of the preceding one: just replace the highlighted "*above*" with "*below*."
- Say that $f(x) = \Theta(g(x))$. We now have a two-sided envelope: beyond $x^{\#}$, the graph of $f(x)$ never enters the territory *above* the graph of $c_1 \cdot g(x)$ and never enters the territory *below* the graph of $c_2 \cdot g(x)$.

In addition to allowing one to make familiar growth-rate comparisons such as "$n^{14} = O(n^{15})$" and "$1.001^n = \Omega(n^{1000})$," we can now also make assertions such as "$\sin x = \Theta(1)$," which are much clumsier to explain in words.

There are "small"-letter analogues of the preceding "big"-letter asymptotic notations, but they are not encountered frequently in computation theory (although they do arise in the analysis of algorithms). In order to prepare the reader for Section 8.4.2, the one place in this book that employs the small-o notation, we note that the notation $o(1)$ refers to any function $f(n)$ that tends to the limit 0 as n grows without bound.

We refer the reader to a text such as [20] for the full repertoire of asymptotic notations that are useful when studying algorithms.

http://www.springer.com/978-0-387-09638-4

The Pillars of Computation Theory
State, Encoding, Nondeterminism
Rosenberg, A.L.
2010, XVIII, 326 p. 49 illus., Softcover
ISBN: 978-0-387-09638-4