ELEMENTS OF THE THEORY OF COMPUTATION

Second Edition

Harry R. Lewis
Gordon McKay Professor of Computer Science
Harvard University
and Dean of Harvard College
Cambridge, Massachusetts

Christos H. Papadimitriou
C. Lester Hogan Professor of Electrical Engineering
and Computer Science
University of California
Berkeley, California

PRENTICE-HALL, Upper Saddle River, New Jersey 07458

Introduction

Look around you. Computation happens everywhere, all the time, initiated by everybody, and affecting us all. Computation can happen because computer scientists over the past decades have discovered sophisticated methods for managing computer resources, enabling communication, translating programs, designing chips and databases, creating computers and programs that are faster, cheaper, easier to use, more secure.

As it is usually the case with all major disciplines, the practical successes of computer science build on its *elegant and solid foundations.* At the basis of physical sciences lie fundamental questions such as *what is the nature of matter?* and *what is the basis and origin of organic life?* Computer science has its own set of fundamental questions: *What is an algorithm? What can and what cannot be computed? When should an algorithm be considered practically feasible?* For more than sixty years (starting even before the advent of the electronic computer) computer scientists have been pondering these questions, *and coming up with ingenious answers that have deeply influenced computer science.*

The purpose of this book is to introduce you to these fundamental ideas, models, and results that permeate computer science, the *basic paradigms* of our field. They are worth studying, for many reasons. First, much of modern computer science is based more or less explicitly on them —and much of the rest should... Also, these ideas and models are powerful and beautiful, excellent examples of mathematical modeling that is elegant, productive, and of lasting value. Besides, they are so much a part of the history and the "collective subconscious" of our field, that it is hard to understand computer science without first being exposed to them.

It probably comes as no surprise that these ideas and models are *mathematical* in nature. Although a computer is undeniably a physical object, it is also

true that very little that is useful can be said of its physical aspects, such as its molecules and its shape; the most useful abstractions of a computer are clearly mathematical, and so the techniques needed to argue about them are necessarily likewise. Besides, practical computational tasks require the ironclad guarantees that only mathematics provides (we want our compilers to translate correctly, our application programs to eventually terminate, and so on). However, the mathematics employed in the theory of computation is rather different from the mathematics used in other applied disciplines. It is generally *discrete*, in that the emphasis is not on real numbers and continuous variables, but on finite sets and sequences. It is based on very few and elementary concepts, and draws its power and depth from the careful, patient, extensive, layer-by-layer manipulation of these concepts —just like the computer. In the first chapter you will be reminded of these elementary concepts and techniques (sets, relations, and induction, among others), and you will be introduced to the style in which they are used in the theory of computation.

The next two chapters, Chapters 2 and 3, describe certain restricted models of computation capable of performing very specialized string manipulation tasks, such as telling whether a given string, say the word *punk*, appears in a given text, such as the collective works of Shakespeare; or for testing whether a given string of parentheses is properly balanced —like () and (())(), but not)(). These restricted computational devices (called *finite-state automata* and *pushdown automata,* respectively) actually come up in practice as very useful and highly optimized components of more general systems such as circuits and compilers. Here they provide fine warm-up exercises in our quest for a formal, general definition of an algorithm. Furthermore, it is instructive to see how the power of these devices waxes and wanes (or, more often, is preserved) with the addition or removal of various features, most notably of *nondeterminism*, an intriguing aspect of computation which is as central as it is (quite paradoxically) unrealistic.

In Chapter 4 we study general models of algorithms, of which the most basic is the *Turing machine,*[†] a rather simple extension of the string-manipulating devices of Chapters 2 and 3 which turns out to be, surprisingly, a general frame-

---

[†] Named after Alan M. Turing (1912–1954), the brilliant English mathematician and philosopher whose seminal paper in 1936 marked the beginning of the theory of computation (and whose image, very appropriately, adorns the cover of this book). Turing also pioneered the fields of artificial intelligence and chess-playing by computer, as well as that of morphogenesis in biology, and was instrumental in breaking *Enigma*, the German naval code during World War II. For more on his fascinating life and times (and on his tragic end in the hands of official cruelty and bigotry) see the book *Alan Turing: The Enigma*, by Andrew Hodges, New York: Simon Schuster, 1983.

work for describing arbitrary algorithms. In order to argue this point, known as the *Church-Turing thesis*, we introduce more and more elaborate models of computation (more powerful variants of the Turing machine, even a *random access Turing machine* and *recursive definitions of numerical functions*), and show that they are all precisely equivalent in power to the basic Turing machine model.

The following chapter deals with *undecidability*, the surprising property of certain natural and well-defined computational tasks to lie *provably* beyond the reach of algorithmic solution. For example, suppose that you are asked whether we can use tiles from a given finite list of basic shapes to tile the whole plane. If the set of shapes contains a square, or even any triangle, then the answer is obviously "yes." But what if it consists of a few bizarre shapes, or if some of the shapes are mandatory, that is, they *must* be used at least once for the tiling to qualify? This is surely the kind of complicated question that you would like to have answered by a machine. In Chapter 5 we use the formalism of Turing machines to prove that this and many other problems *cannot be solved by computers at all.*

Even when a computational task is amenable to solution by *some* algorithm, it may be the case that there is no *reasonably fast, practically feasible* algorithm that solves it. In the last two chapters of this book we show how real-life computational problems can be categorized in terms of their *complexity:* Certain problems can be solved within reasonable, *polynomial* time bounds, whereas others seem to require amounts of time that grow astronomically, *exponentially.* In Chapter 7 we identify a class of common, practical, and notoriously difficult problems that are called $\mathcal{NP}$*-complete* (the traveling salesman problem is only one of them). We establish that all these problems are *equivalent* in that, if one of them has an efficient algorithm, then all of them do. It is widely believed that all $\mathcal{NP}$-complete problems are of inherently exponential complexity; whether this conjecture is actually true is the famous $\mathcal{P} \neq \mathcal{NP}$ problem, one of the most important and deep problems facing mathematicians and computer scientists today.

This book is very much about algorithms and their formal foundations. However, as you are perhaps aware, the subject of algorithms, their analysis and their design, is considered in today's computer science curriculum quite separate from that of the theory of computation. In the present edition of this book we have tried to restore some of the unity of the subject. As a result, this book also provides a decent, if somewhat specialized and unconventional, introduction to the subject of algorithms. Algorithms and their analysis are introduced informally in Chapter 1, and are picked up again and again in the context of the restricted models of computation studied in Chapters 2 and 3, and of the natural computational problems that they spawn. This way, when general models of algorithms are sought later, the reader is in a better position to appreciate the scope of the quest, and to judge its success. Algorithms play a

major role in our exposition of complexity as well, because there is no better way to appreciate a complex problem than to contrast it with another, amenable to an efficient algorithm. The last chapter culminates in a section on *coping with $\mathcal{NP}$-completeness*, where we present an array of algorithmic techniques that have been successfully used in attacking $\mathcal{NP}$-complete problems (approximation algorithms, exhaustive algorithms, local search heuristics, and so on).

Computation is essential, powerful, beautiful, challenging, ever-expanding —and so is its theory. This book only tells the beginning of an exciting story. It is a modest introduction to a few basic and carefully selected topics from the treasure chest of the theory of computation. We hope that it will motivate its readers to seek out more; the references at the end of each chapter point to good places to start.

1 Sets, Relations, and Languages

1.1 SETS

They say that mathematics is the language of science —it is certainly the language of the theory of computation, the scientific discipline we shall be studying in this book. And the language of mathematics deals with *sets*, and the complex ways in which they overlap, intersect, and in fact take part themselves in forming new sets.

A **set** is a collection of objects. For example, the collection of the four letters a, b, c, and d is a set, which we may name L; we write $L = \{a, b, c, d\}$. The objects comprising a set are called its **elements** or **members**. For example, b is an element of the set L; in symbols, $b \in L$. Sometimes we simply say that b is in L, or that L contains b. On the other hand, z is not an element of L, and we write $z \notin L$.

In a set we do not distinguish repetitions of the elements. Thus the set $\{\text{red}, \text{blue}, \text{red}\}$ is the same set as $\{\text{red}, \text{blue}\}$. Similarly, the order of the elements is immaterial; for example, $\{3, 1, 9\}$, $\{9, 3, 1\}$, and $\{1, 3, 9\}$ are the same set. To summarize: Two sets are equal (that is, the same) if and only if they have the same elements.

The elements of a set need not be related in any way (other than happening to be all members of the same set); for example, $\{3, \text{red}, \{d, \text{blue}\}\}$ is a set with three elements, one of which is itself a set. A set may have only one element; it is then called a **singleton**. For example, $\{1\}$ is the set with 1 as its only element; thus $\{1\}$ and 1 are quite different. There is also a set with no element at all. Naturally, there can be only one such set: it is called the **empty** set, and is denoted by $\emptyset$. Any set other than the empty set is said to be nonempty.

So far we have specified sets by simply listing all their elements, separated by commas and included in braces. Some sets cannot be written in this way,

because they are infinite. For example, the set $\mathbf{N}$ of natural numbers is infinite; we may suggest its elements by writing $\mathbf{N} = \{0, 1, 2, \ldots\}$, using the three dots and your intuition in place of an infinitely long list. A set that is not infinite is finite.

Another way to specify a set is by referring to other sets and to properties that elements may or may not have. Thus if $I = \{1, 3, 9\}$ and $G = \{3, 9\}$, G may be described as the set of elements of I that are greater than 2. We write this fact as follows.

$$G = \{x : x \in I \text{ and } x \text{ is greater than } 2\}.$$

In general, if a set A has been defined and P is a property that elements of A may or may not have, then we can define a new set

$$B = \{x : x \in A \text{ and } x \text{ has property } P\}.$$

As another example, the set of odd natural numbers is

$$O = \{x : x \in \mathbf{N} \text{ and } x \text{ is not divisible by } 2\}.$$

A set A is a **subset** of a set B —in symbols, $A \subseteq B$— if each element of A is also an element of B. Thus $O \subseteq \mathbf{N}$, since each odd natural number is a natural number. Note that any set is a subset of itself. If A is a subset of B but A is not the same as B, we say that A is a **proper subset** of B and write $A \subset B$. Also note that the empty set is a subset of every set. For if B is any set, then $\emptyset \subseteq B$, since each element of $\emptyset$ (of which there are none) is also an element of B.

To prove that two sets A and B are equal, we may prove that $A \subseteq B$ and $B \subseteq A$. Every element of A must then be an element of B and vice versa, so that A and B have the same elements and $A = B$.

Two sets can be combined to form a third by various *set operations*, just as numbers are combined by arithmetic operations such as addition. One set operation is **union**: the union of two sets is that set having as elements the objects that are elements of at least one of the two given sets, and possibly of both. We use the symbol $\cup$ to denote union, so that

$$A \cup B = \{x : x \in A \text{ or } x \in B\}.$$

For example,

$$\{1, 3, 9\} \cup \{3, 5, 7\} = \{1, 3, 5, 7, 9\}.$$

The **intersection** of two sets is the collection of all elements the two sets have in common; that is,

$$A \cap B = \{x : x \in A \text{ and } x \in B\}.$$

For example,

$$\{1,3,9\} \cap \{3,5,7\} = \{3\},$$

and

$$\{1,3,9\} \cap \{a,b,c,d\} = \emptyset.$$

Finally, the **difference** of two sets A and B, denoted by $A - B$, is the set of all elements of A that are not elements of B.

$$A - B = \{x : x \in A \text{ and } x \notin B\}.$$

For example,

$$\{1,3,9\} - \{3,5,7\} = \{1,9\}.$$

Certain properties of the set operations follow easily from their definitions. For example, if A, B, and C are sets, the following laws hold.

Idempotency	$A \cup A = A$
	$A \cap A = A$
Commutativity	$A \cup B = B \cup A$
	$A \cap B = B \cap A$
Associativity	$(A \cup B) \cup C = A \cup (B \cup C)$
	$(A \cap B) \cap C = A \cap (B \cap C)$
Distributivity	$(A \cup B) \cap C = (A \cap C) \cup (B \cap C)$
	$(A \cap B) \cup C = (A \cup C) \cap (B \cup C)$
Absorption	$(A \cup B) \cap A = A$
	$(A \cap B) \cup A = A$
DeMorgan's laws	$A - (B \cup C) = (A - B) \cap (A - C)$
	$A - (B \cap C) = (A - B) \cup (A - C)$

Example 1.1.1: Let us prove the first of De Morgan's laws. Let

$$L = A - (B \cup C)$$

and

$$R = (A - B) \cap (A - C);$$

we are to show that $L = R$. We do this by showing (a) $L \subseteq R$ and (b) $R \subseteq L$.

(a) Let x be any element of L; then $x \in A$, but $x \notin B$ and $x \notin C$. Hence x is an element of both $A - B$ and $A - C$, and is thus an element of R. Therefore $L \subseteq R$.

(b) Let $x \in R$; then x is an element of both $A - B$ and $A - C$, and is therefore in A but in neither B nor C. Hence $x \in A$ but $x \notin B \cup C$, so $x \in L$.

Therefore $R \subseteq L$, and we have established that $L = R$.$\diamondsuit$

Two sets are **disjoint** if they have no element in common, that is, if their intersection is empty.

It is possible to form intersections and unions of more than two sets. If S is any collection of sets, we write $\bigcup S$ for the set whose elements are the elements of all the sets in S. For example, if $S = \{\{a, b\}, \{b, c\}, \{c, d\}\}$ then $\bigcup S = \{a, b, c, d\}$; and if $S = \{\{n\} : n \in \mathbf{N}\}$, that is, the collection of all the singleton sets with natural numbers as elements, then $\bigcup S = \mathbf{N}$. In general,

$$\bigcup S = \{x : x \in P \text{ for some set } P \in S\}.$$

Similarly,

$$\bigcap S = \{x : x \in P \text{ for each set } P \in S\}.$$

The collection of all subsets of a set A is itself a set, called the **power set** of A and denoted 2^A. For example, the subsets of $\{c, d\}$ are $\{c, d\}$ itself, the singletons $\{c\}$ and $\{d\}$ and the empty set $\emptyset$, so

$$2^{\{c,d\}} = \{\{c, d\}, \{c\}, \{d\}, \emptyset\}.$$

A **partition** of a nonempty set A is a subset Π of 2^A such that $\emptyset$ is not an element of Π and such that each element of A is in one and only one set in Π. That is, Π is a partition of A if Π is a set of subsets of A such that

(1) each element of Π is nonempty;
(2) distinct members of Π are disjoint;
(3) $\bigcup \Pi = A$.

For example, $\{\{a, b\}, \{c\}, \{d\}\}$ is a partition of $\{a, b, c, d\}$, but $\{\{b, c\}, \{c, d\}\}$ is not. The sets of even and odd natural numbers form a partition of $\mathbf{N}$.

Problems for Section 1.1

1.1.1. Determine whether each of the following is true or false.

(a) $\emptyset \subseteq \emptyset$
(b) $\emptyset \in \emptyset$
(c) $\emptyset \in \{\emptyset\}$
(d) $\emptyset \subseteq \{\emptyset\}$
(e) $\{a, b\} \in \{a, b, c, \{a, b\}\}$
(f) $\{a, b\} \subseteq \{a, b, \{a, b\}\}$
(g) $\{a, b\} \subseteq 2^{\{a,b,\{a,b\}\}}$
(h) $\{\{a, b\}\} \in 2^{\{a,b,\{a,b\}\}}$
(i) $\{a, b, \{a, b\}\} - \{a, b\} = \{a, b\}$

1.1.2. What are these sets? Write them using braces, commas, and numerals only.
(a) $(\{1,3,5\} \cup \{3,1\}) \cap \{3,5,7\}$
(b) $\bigcup\{\{3\},\{3,5\},\bigcap\{\{5,7\},\{7,9\}\}\}$
(c) $(\{1,2,5\} - \{5,7,9\}) \cup (\{5,7,9\} - \{1,2,5\})$
(d) $2^{\{7,8,9\}} - 2^{\{7,9\}}$
(e) $2^{\emptyset}$

1.1.3. Prove each of the following.
(a) $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
(b) $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
(c) $A \cap (A \cup B) = A$
(d) $A \cup (A \cap B) = A$
(e) $A - (B \cap C) = (A - B) \cup (A - C)$

1.1.4. Let $S = \{a, b, c, d\}$.
(a) What partition of S has the fewest members? The most members?
(b) List all partitions of S with exactly two members.

1.2 RELATIONS AND FUNCTIONS

Mathematics deals with statements about objects and the relations between them. It is natural to say, for example, that "less than" is a relation between objects of a certain kind —namely, numbers— which holds between 4 and 7 but does not hold between 4 and 2, or between 4 and itself. But how can we express relations between objects in the only mathematical language we have available at this point —that is to say, the language of sets? We simply think of a relation as being itself a set. The objects that belong to the relation are, in essence, the combinations of individuals for which that relation holds in the intuitive sense. So the less-than relation is the set of all *pairs* of numbers such that the first number is less than the second.

But we have moved a bit quickly. In a pair that belongs to a relation, we need to be able to distinguish the two parts of the pair, and we have not explained how to do so. We cannot write these pairs as sets, since $\{4,7\}$ is the same thing as $\{7,4\}$. It is easiest to introduce a new device for grouping objects called an **ordered pair**.†

We write the ordered pair of two objects a and b as (a,b); a and b are called the **components** of the ordered pair (a,b). The ordered pair (a,b) is not the same as the set $\{a,b\}$. First, the order matters: (a,b) is different from (b,a),

† True fundamentalists would see the ordered pair (a,b) not as a new kind of object, but as identical to $\{a,\{a,b\}\}$.

whereas $\{a, b\} = \{b, a\}$. Second, the two components of an ordered pair need not be distinct; $(7, 7)$ is a valid ordered pair. Note that two ordered pairs (a, b) and (c, d) are equal only when $a = c$ and $b = d$.

The **Cartesian product** of two sets A and B, denoted by $A \times B$, is the set of all ordered pairs (a, b) with $a \in A$ and $b \in B$. For example,

$$\{1, 3, 9\} \times \{b, c, d\} = \{(1, b), (1, c), (1, d), (3, b), (3, c), (3, d), (9, b), (9, c), (9, d)\}.$$

A binary relation on two sets A and B is a subset of $A \times B$. For example, $\{(1, b), (1, c), (3, d), (9, d)\}$ is a binary relation on $\{1, 3, 9\}$ and $\{b, c, d\}$. And $\{(i, j) : i, j \in \mathbf{N} \text{ and } i < j\}$ is the less-than relation; it is a subset of $\mathbf{N} \times \mathbf{N}$ —often the two sets related by a binary relation are identical.

More generally, let n be any natural number. Then if $a_1, \ldots, a_n$ are any n objects, not necessarily distinct, $(a_1, \ldots, a_n)$ is an **ordered tuple**; for each $i = 1, \ldots, n$, a_i is the ith component of $(a_1, \ldots, a_n)$. An ordered m-tuple $(b_1, \ldots, b_m)$, where m is a natural number, is the same as $(a_1, \ldots, a_n)$ if and only if $m = n$ and $a_i = b_i$, for $i = 1, \ldots, n$. Thus $(4, 4)$, $(4, 4, 4)$, $((4, 4), 4)$, and $(4, (4, 4))$ are all distinct. Ordered 2-tuples are the same as the ordered pairs discussed above, and ordered 3-, 4-, 5-, and 6-tuples are called **ordered triples**, **quadruples**, **quintuples**, and **sextuples**, respectively. On the other hand, a **sequence** is an ordered n-tuple for some unspecified n (the **length** of the sequence). If $A_1, \ldots, A_n$ are any sets, then the **n-fold Cartesian product** $A_1 \times \cdots \times A_n$ is the set of all ordered n-tuples $(a_1, \ldots, a_n)$, with $a_i \in A_i$, for each $i = 1, \ldots, n$. In case all the A_i, are the same set A, the n-fold Cartesian product $A \times \cdots \times A$ of A with itself is also written A^n. For example, $\mathbf{N}^2$ is the set of ordered pairs of natural numbers. An **n-ary relation** on sets $A_1, \ldots, A_n$ is a subset of $A_1 \times \cdots \times A_n$; 1-, 2-, and 3-ary relations are called **unary**, **binary**, and **ternary relations**, respectively.

Another fundamental mathematical idea is that of a *function*. On the intuitive level, a function is an association of each object of one kind with a unique object of another kind: of persons with their ages, dogs with their owners, numbers with their successors, and so on. But by using the idea of a binary relation as a set of ordered pairs, we can replace this intuitive idea by a concrete definition. A **function** from a set A to a set B is a binary relation R on A and B with the following special property: for each element $a \in A$, there is *exactly one* ordered pair in R with first component a. To illustrate the definition, let C be the set of cities in the United States and let S be the set of states; and let

$$R_1 = \{(x, y) : x \in C, y \in S, \text{ and } x \text{ is a city in state } y\},$$

$$R_2 = \{(x, y) : x \in S, y \in C, \text{ and } y \text{ is a city in state } x\}.$$

Then R_1 is a function, since each city is in one and only one state, but R_2 is not a function, since some states have more than one city.†

In general, we use letters such as f, g, and h for functions and we write $f : A \mapsto B$ to indicate that f is a function from A to B. We call A the **domain** of f. If a is any element of A we write $f(a)$ for that element b of B such that $(a, b) \in f$; since f is a function, there is exactly one $b \in B$ with this property, so $f(a)$ denotes a unique object. The object $f(a)$ is called the **image** of a under f. To specify a function $f : A \mapsto B$, it suffices to specify $f(a)$ for each $a \in A$; for example, to specify the function R_1 above, it suffices to specify, for each city, the state in which it is located. If $f : A \mapsto B$ and A' is a subset of A, then we define $f[A'] = \{f(a) : a \in A'\}$ (that is, $\{b : b = f(a)$ for some $a \in A'\}$). We call $f[A']$ the **image** of A' under f. The **range** of f is the image of its domain.

Ordinarily, if the domain of a function is a Cartesian product, one set of parentheses is dropped. For example, if $f : \mathbf{N} \times \mathbf{N} \mapsto \mathbf{N}$ is defined so that the image under f of an ordered pair (m, n) is the sum of m and n, we would write $f(m, n) = m+n$ rather than $f((m, n)) = m+n$, simply as a matter of notational convenience.

If $f : A_1 \times A_2 \times \ldots \times A_n \mapsto B$ is a function, and $f(a_1, \ldots, a_n) = b$, where $a_i \in A_i$ for $i = 1, \ldots, n$ and $b \in B$, then we sometimes call $a_1, \ldots, a_n$ the **arguments** of f and b the corresponding **value** of f. Thus f may be specified by giving its value for each n-tuple of arguments.

Certain kinds of functions are of special interest. A function $f : A \mapsto B$ is **one-to-one** if for any two distinct elements $a, a' \in A$, $f(a) \neq f(a')$. For example, if C is the set of cities in the United States, S is the set of states, and $g : S \mapsto C$ is specified by

$$g(s) = \text{the capital of state } s$$

for each $s \in S$, then g is one-to-one since no two states have the same capital. A function $f : A \mapsto B$ is **onto** B if each element of B is the image under f of some element of A. The function g just specified is not onto C, but the function R_1 defined above is onto S since each state contains at least one city. Finally a mapping $f : A \mapsto B$ is a **bijection** between A and B if it is both one-to-one and onto B; for example, if C_0 is the set of capital cities, then the function $g : S \mapsto C_0$ specified, as before, by

$$g(s) = \text{the capital of state } s$$

is a bijection between S and C_0.

---

† We consider Cambridge, Massachusetts, and Cambridge, Maryland, not the same city, but different cities that happen to have the same name.

The **inverse** of a binary relation $R \subseteq A \times B$, denoted $R^{-1} \subseteq B \times A$, is simply the relation $\{(b,a) : (a,b) \in R\}$. For example, the relation R_2 defined above is the inverse of R_1. Thus, the inverse of a function need not be a function. In the case of R_1 its inverse fails to be a function since some states have more than one city; that is, there are distinct cities c_1 and c_2 such that $R_1(c_1) = R_1(c_2)$. A function $f : A \mapsto B$ may also fail to have an inverse if there is some element $b \in B$ such that $f(a) \neq b$ for all $a \in A$. If $f : A \mapsto B$ is a bijection, however, neither of these eventualities can occur, and f^{-1} is a function —indeed, a bijection between B and A. Moreover $f^{-1}(f(a)) = a$ for each $a \in A$, and $f(f^{-1}(b)) = b$ for each $b \in B$.

When a particularly simple bijection between two sets has been specified, it is sometimes possible to view an object in the domain and its image in the range as virtually indistinguishable: the one may be seen as a renaming or a way of rewriting the other. For example, singleton sets and ordered 1-tuples are, strictly speaking, different, but not much harm is done if we occasionally blur the distinction, because of the obvious bijection f such that $f(\{a\}) = (a)$ for any singleton $\{a\}$. Such a bijection is called a *natural isomorphism*; of course this is not a formal definition since what is "natural" and what distinctions can be blurred depend on the context. Some slightly more complex examples should make the point more clearly.

Example 1.2.1: For any three sets A, B, and C, there is a natural isomorphism of $A \times B \times C$ to $(A \times B) \times C$, namely

$$f(a,b,c) = ((a,b),c)$$

for any $a \in A$, $b \in B$, and $c \in C$.$\diamondsuit$

Example 1.2.2: For any sets A and B, there is a natural isomorphism ϕ from

$$2^{A \times B},$$

that is, the set of all binary relations on A and B, to the set

$$\{f : f \text{ is a function from } A \text{ to } 2^B\}.$$

Namely, for any relation $R \subseteq A \times B$, let $\phi(R)$ be that function $f : A \mapsto 2^B$ such that

$$f(a) = \{b : b \in B \text{ and } (a,b) \in R\}.$$

For example, if S is the set of states and $R \subseteq S \times S$ contains any ordered pair of states with a common border, then the naturally associated function $f : S \mapsto 2^S$ is specified by $f(s) = \{s' : s' \in S \text{ and } s' \text{ shares a border with } s\}$.$\diamondsuit$

Example 1.2.3: Sometimes we regard the inverse of a function $f : A \mapsto B$ as a function even when f is not a bijection. The idea is to regard $f^{-1} \subseteq B \times A$ as a function from B to 2^A, using the natural isomorphism described under Example 1.2.2. Thus $f^{-1}(b)$ is, for any $b \in B$, the set of all $a \in A$ such that $f(a) = b$. For example, if R_1 is as defined above —the function that assigns to each city the state in which it is located— then $R_1^{-1}(s)$, where s is a state, is the set of all cities in that state.

If Q and R are binary relations, then their composition $Q \circ R$, or simply QR, is the relation $\{(a, b) :$ for some c, $(a, c) \in Q$ and $(c, b) \in R\}$. Note that the composition of two functions $f : A \mapsto B$ and $g : B \mapsto C$ is a function h from A to C such that $h(a) = g(f(a))$ for each $a \in A$. For example, if f is the function that assigns to each dog its owner and g assigns to each person his or her age, then $f \circ g$ assigns to each dog the age of its owner.$\diamondsuit$

Problems for Section 1.2

1.2.1. Write each of the following explicitly.
(a) $\{1\} \times \{1, 2\} \times \{1, 2, 3\}$
(b) $\emptyset \times \{1, 2\}$
(c) $2^{\{1,2\}} \times \{1, 2\}$

1.2.2. Let $R = \{(a, b), (a, c), (c, d), (a, a), (b, a)\}$. What is $R \circ R$, the composition of R with itself? What is R^{-1}, the inverse of R? Is R, $R \circ R$, or R^{-1} a function?

1.2.3. Let $f : A \mapsto B$ and $g : B \mapsto C$. Let $h : A \mapsto C$ be their composition. In each of the following cases state necessary and sufficient conditions on f and g for h to be as specified.
(a) Onto.
(b) One-to-one.
(c) A bijection.

1.2.4. If A and B are any sets, we write B^A for the set of all functions from A to B. Describe a natural isomorphism between $\{0, 1\}^A$ and 2^A.

1.3 SPECIAL TYPES OF BINARY RELATIONS

Binary relations will be found over and over again in these pages; it will be helpful to have convenient ways of representing them and some terminology for discussing their properties. A completely "random" binary relation has no significant internal structure; but many relations we shall encounter arise out

of specific contexts and therefore have important regularities. For example, the relation that holds between two cities if they belong to the same state has certain "symmetries" and other properties that are worth noting, discussing, and exploiting.

In this section we study relations that exhibit these and similar regularities. We shall deal only with binary relations on a set and itself. Thus, let A be a set, and $R \subseteq A \times A$ be a relation on A. The relation R can be represented by a **directed graph**. Each element of A is represented by a small circle —what we call a **node** of the directed graph— and an arrow is drawn from a to b if and only if $(a, b) \in R$. The arrows are the **edges** of the directed graph. For example, the relation $R = \{(a, b), (b, a), (a, d), (d, c), (c, c), (c, a)\}$ is represented by the graph in Figure 1-1. Note in particular the loop from c to itself, corresponding to the pair $(c, c) \in R$. From a node of a graph to another there is either no edge, or one edge —we do not allow "parallel arrows."

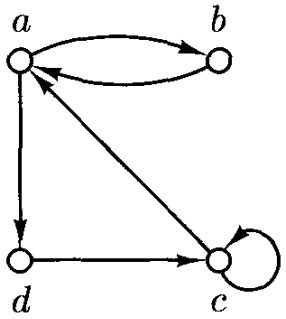


Figure 1-1

There is no formal distinction between binary relations on a set A and directed graphs with nodes from A. We use the term *directed graph* when we want to emphasize that the set on which the relation is defined is of no independent interest to us, outside the context of this particular relation. Directed graphs, as well as the *undirected graphs* soon to be introduced, are useful as models and abstractions of complex systems (traffic and communication networks, computational structures and processes, etc.). In Section 1.6, and in much more detail in Chapters 6 and 7, we shall discuss many interesting *computational problems* arising in connection with directed graphs.

For another example of a binary relation/directed graph, the less-than-or-equal-to relation $\leq$ defined on the natural numbers is illustrated in Figure 1-2. Of course, the entire directed graph cannot be drawn, since it would be infinite.

A relation $R \subseteq A \times A$ is **reflexive** if $(a, a) \in R$ for each $a \in A$. The directed graph representing a reflexive relation has a loop from each node to itself. For example, the directed graph of Figure 1-2 represents a reflexive relation, but that of Figure 1-1 does not.

A relation $R \subseteq A \times A$ is **symmetric** if $(b, a) \in R$ whenever $(a, b) \in R$.

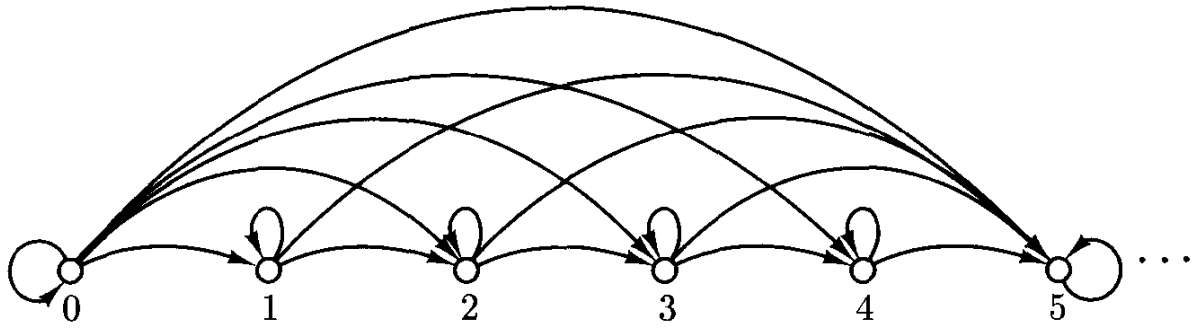


Figure 1-2

In the corresponding directed graph, whenever there is an arrow between two nodes, there are arrows between those nodes in both directions. For example, the directed graph of Figure 1-3 represents a symmetric relation. This directed graph might depict the relation of "friendship" among six people, since whenever x is a friend of y, y is also a friend of x. The relation of friendship is not reflexive, since we do not regard a person as his or her own friend. Of course, a relation could be both symmetric and reflexive; for example, $\{(a, b) : a$ and b are persons with the same father$\}$ is such a relation.

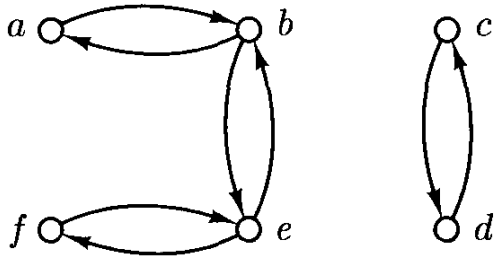


Figure 1-3

A symmetric relation *without pairs of the form* (a, a) is represented as an **undirected graph**, or simply a **graph**. Graphs are drawn without arrowheads, combining pairs of arrows going back and forth between the same nodes. For example, the relation shown in Figure 1-3 could also be represented by the graph in Figure 1-4.

A relation R is **antisymmetric** if whenever $(a, b) \in R$ and a and b are distinct, then $(b, a) \notin R$. For example, let P be the set of all persons. Then

$$\{(a, b) : a, b \in P \text{ and } a \text{ is the father of } b\}$$

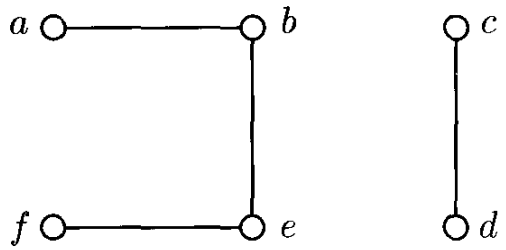


Figure 1-4

is antisymmetric. A relation may be neither symmetric nor antisymmetric; for example, the relation

$$\{(a, b) : a, b \in P \text{ and } a \text{ is the brother of } b\}$$

and the relation represented in Figure 1-1 are neither.

A binary relation R is **transitive** if whenever $(a, b) \in R$ and $(b, c) \in R$, then $(a, c) \in R$. The relation

$$\{(a, b) : a, b \in P \text{ and } a \text{ is an ancestor of } b\}$$

is transitive, since if a is an ancestor of b and b is an ancestor of c, then a is an ancestor of c. So is the less-than-or-equal relation. In terms of the directed graph representation, transitivity is equivalent to the requirement that whenever there is a sequence of arrows leading from an element a to an element z, there is an arrow directly from a to z. For example, the relation illustrated in Figure 1-5 is transitive.

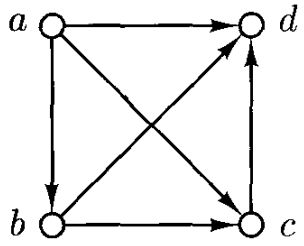


Figure 1-5

A relation that is reflexive, symmetric, and transitive is called an **equivalence relation**. The representation of an equivalence relation by an undirected graph consists of a number of *clusters*; within each cluster, each pair of nodes is connected by a line (see Figure 1-6). The "clusters" of an equivalence relation are called its **equivalence classes**. We normally write $[a]$ for the equivalence class containing an element a, provided the equivalence relation R is understood by the context. That is, $[a] = \{b : (a, b) \in R\}$, or, since R is symmetric, $[a] = \{b : (b, a) \in R\}$. For example, the equivalence relation in Figure 1-6 has three equivalence classes, one with four elements, one with three elements, and one with one element.

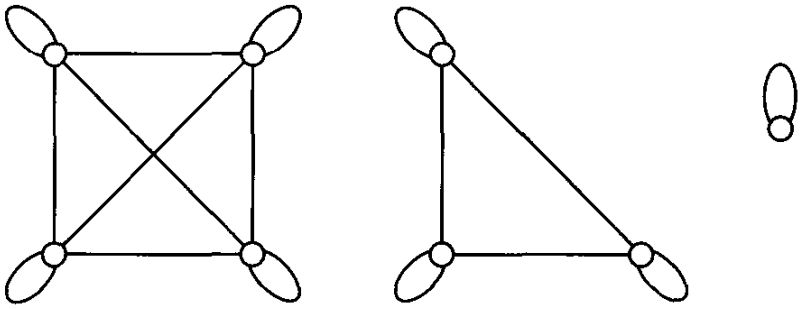


Figure 1-6

Theorem 1.3.1: *Let R be an equivalence relation on a nonempty set A. Then the equivalence classes of R constitute a partition of A.*

Proof: Let $\Pi = \{[a] : a \in A\}$. We must show that the sets in Π are nonempty, disjoint, and together exhaust A. All equivalence classes are nonempty, since $a \in [a]$ for all $a \in A$, by reflexivity. To show that they are disjoint, consider any two distinct equivalence classes $[a]$ and $[b]$, and suppose that $[a] \cap [b] \neq \emptyset$. Thus there is an element c such that $c \in [a]$ and c $\in [b]$. Hence $(a, c) \in R$ and $(c, b) \in R$; since R is transitive, $(a, b) \in R$; and since R is symmetric, $(b, a) \in R$. But now take any element $d \in [a]$; then $(d, a) \in R$ and, by transitivity, $(d, b) \in R$. Hence $d \in [b]$, so that $[a] \subseteq [b]$. Likewise $[b] \subseteq [a]$. Therefore $[a] = [b]$. But this contradicts the assumption that $[a]$ and $[b]$ are distinct.

To see that $\bigcup \Pi = A$, simply notice that each element a of A is in some set in Π —namely, $a \in [a]$, by reflexivity. ■

Thus starting from an equivalence relation R, we can always construct a corresponding partition Π. For example, if

$$R = \{(a, b) : a \text{ and } b \text{ are persons and } a \text{ and } b \text{ have the same parents}\},$$

then the equivalence classes of R are all groups of siblings. Note that the construction of Theorem 1.3.1 can be reversed: from any partition, we can construct a corresponding equivalence relation. Namely, if Π is a partition of A, then

$$R = \{(a, b) : a \text{ and } b \text{ belong in the same set of } \Pi\}$$

is an equivalence relation. Thus there is a natural isomorphism between the set of equivalence relations on a set A and the set of partitions of A.

A relation that is reflexive, antisymmetric, and transitive is called a **partial order**. For example,

$$\{(a, b) : a, b \text{ are persons and } a \text{ is an ancestor of } b\}$$

is a partial order (provided we consider each person to be an ancestor of himself or herself). If $R \subseteq A \times A$ is a partial order, an element $a \in A$ is called **minimal** if the following is true: $(b, a) \in R$ only if $a = b$. For example, in the ancestor relation defined above, Adam and Eve are the only minimal elements. A finite partial order must have at least one minimal element, but an infinite partial order need not have one.

A partial order $R \subseteq A \times A$ is a **total order** if, for all $a, b \in A$, either $(a, b) \in R$ or $(b, a) \in R$. Thus the ancestor relation is not a total order since not any two people are ancestrally related (for example, siblings are not); but the less-than-or-equal-to relation on numbers is a total order. A total order cannot have two or more minimal elements.

A **path** in a binary relation R is a sequence $(a_1, \ldots, a_n)$ for some $n \geq 1$ such that $(a_i, a_{i+1}) \in R$ for $i = 1, \ldots, n-1$; this path is said to be from a_1 to a_n. The **length** of a path $(a_1, \ldots, a_n)$ is n. The path $(a_1, \ldots, a_n)$ is a **cycle** if the a_i's are all distinct and also $(a_n, a_1) \in R$.

Problems for Section 1.3

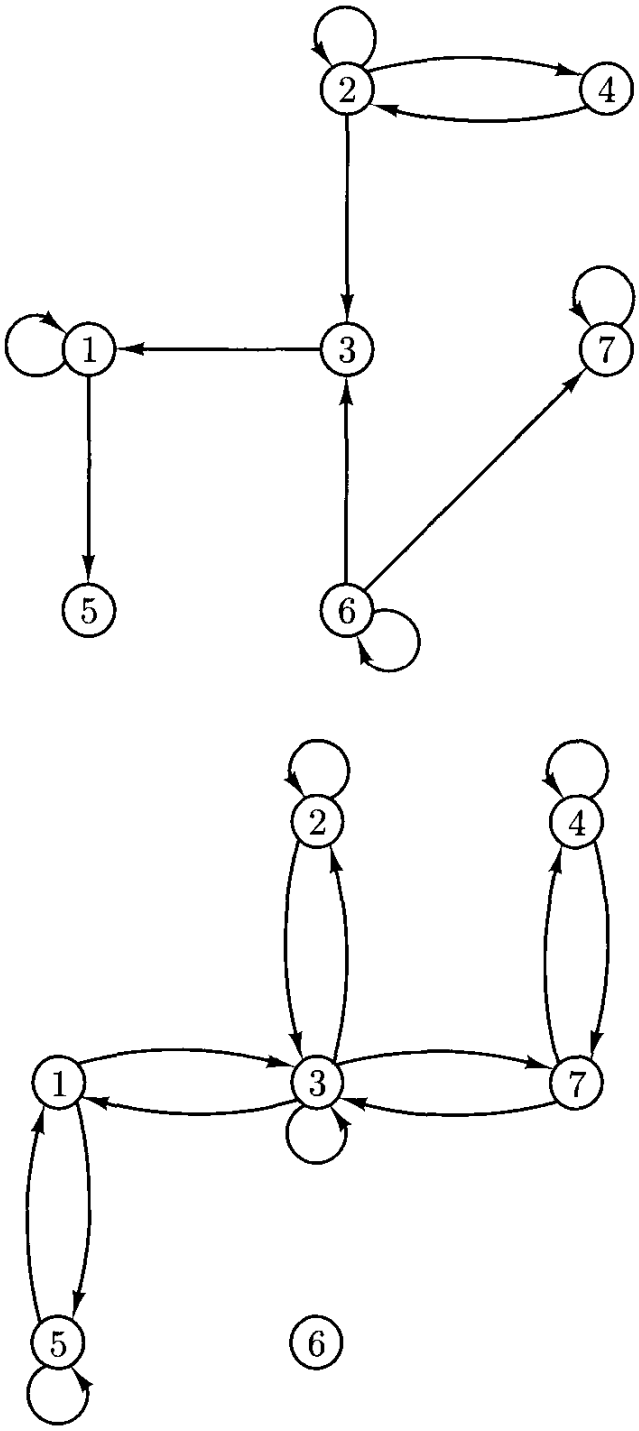
1.3.1. Let $R = \{(a, \mathrm{c}), (\mathrm{c}, e), (e, e), (e, b), (d, b), (d, d)\}$. Draw directed graphs representing each of the following.
(a) R
(b) R^{-1}
(c) $R \cup R^{-1}$
(d) $R \cap R^{-1}$

1.3.2. Let R and S be the binary relations on $A = \{1, \ldots, 7\}$ with the graphical representations shown in the next page.
(a) Indicate whether each of R and S is (i) symmetric, (ii) reflexive, and (iii) transitive.
(b) Repeat (a) for the relation $R \cup S$.

1.3.3. Draw directed graphs representing relations of the following types.
(a) Reflexive, transitive, and antisymmetric.
(b) Reflexive, transitive, and neither symmetric nor antisymmetric.

1.3.4. Let A be a nonempty set and let $R \subseteq A \times A$ be the empty set. Which properties does R have?
(a) Reflexivity.
(b) Symmetry.
(c) Antisymmetry.
(d) Transitivity.

1.3.5. Let $f : A \mapsto B$. Show that the following relation R is an equivalence relation on A: $(a, b) \in R$ if and only if $f(a) = f(b)$.



1.3.6. Let $R \subseteq A \times A$ be a binary relation as defined below. In which cases is R a partial order? a total order?

(a) $A =$ the positive integers; $(a, b) \in R$ if and only if b is divisible by a.

(b) $A = \mathbf{N} \times \mathbf{N}$; $((a, b)(c, d)) \in R$ if and only if $a \leq c$ or $b \leq d$.

(c) $A = \mathbf{N}$; $(a, b) \in R$ if and only if $b = a$ or $b = a + 1$.

(d) A is the set of all English words; $(a, b) \in R$ if and only if a is no longer than b.

(e) A is the set of all English words; $(a, b) \in R$ if and only if a is the same as b or occurs more frequently than b in the present book.

1.3.7. Let R_1 and R_2 be any two partial orders on the same set A. Show that $R_1 \cap R_2$ is a partial order.

1.3.8. (a) Prove that if S is any collection of sets, then $R_S = \{(A, B) : A, B \in S \text{ and } A \subseteq B\}$ is a partial order.
(b) Let $S = 2^{\{1,2,3\}}$. Draw a directed graph representing the partial order R_S defined in (a). Which are the minimal elements of R_S?

1.3.9. Under what circumstances does a directed graph represent a function?

1.3.10. Show that any function from a finite set to itself contains a cycle.

1.3.11. Let S be any set, and let $\mathcal{P}$ be the set of all partitions of S. Let R be the binary relation on $\mathcal{P}$ such that $(\Pi_1, \Pi_2) \in R$ if and only if for every $S_1 \in \Pi_1$, there is an $S_2 \in \Pi_2$ such that $S_1 \subseteq S_2$; if $(\Pi_1, \Pi_2) \in R$ we say that Π_1 **refines** Π_2. Show that R is a partial order on $\mathcal{P}$. What elements of $\mathcal{P}$ are maximal and minimal? Suppose that $\mathcal{P}$ were an arbitrary collection of subsets of 2^S, which need not be partitions of S. Would R necessarily be a partial order?

1.4 FINITE AND INFINITE SETS

A basic property of a finite set is its *size*, that is, the number of elements it contains. Some facts about the sizes of finite sets are so obvious they hardly need proof. For example, if $A \subseteq B$, then the size of A is less than or equal to that of B; the size of A is zero if and only if A is the empty set.

However, an extension of the notion of "size" to infinite sets leads to difficulties if we attempt to follow our intuition. Are there more multiples of 17 $(0, 17, 34, 51, 68, \ldots)$ than there are perfect squares $(0, 1, 4, 9, 16, \ldots)$? You are welcome to speculate on alternatives, but experience has shown that the only satisfactory convention is to regard these sets as having *the same size.*

We call two sets A and B **equinumerous** if there is a bijection $f : A \mapsto B$. Recall that if there is a bijection $f : A \mapsto B$, then there is a bijection $f^{-1} : B \mapsto A$; hence equinumerosity is a symmetric relation. In fact, as is easily shown, it is an equivalence relation. For example, $\{8, \text{red}, \{\emptyset, b\}\}$ and $\{1, 2, 3\}$ are equinumerous; let $f(8) = 1$, $f(\text{red}) = 2$, $f(\{\emptyset, b\}) = 3$. So are the multiples of 17 and the perfect squares; a bijection is given by $f(17n) = n^2$ for each $n \in \mathbf{N}$.

In general, we call a set **finite** if, intuitively, it is equinumerous with $\{1, 2, \ldots, n\}$ for some natural number n. (For $n = 0$, $\{1, \ldots, n\}$ is the empty set, so $\emptyset$ is finite, being equinumerous with itself.) If A and $\{1, \ldots, n\}$ are equinumerous, then we say that the cardinality of A (in symbols, $|A|$) is n. The cardinality of a finite set is thus the number of elements in it.

A set is **infinite** if it is not finite. For example, the set $\mathbf{N}$ of natural numbers is infinite; so are sets such as the set of integers, the set of reals, and the set of perfect squares. However, *not all infinite sets are equinumerous.*

A set is said to be **countably infinite** if it is equinumerous with $\mathbf{N}$, and **countable** if it is finite or countably infinite. A set that is not countable is **uncountable**. To show that a set A is countably infinite we must exhibit a bijection f between A and $\mathbf{N}$; equivalently, we need only suggest a way in which A can be enumerated as

$$A = \{a_0, a_1, a_2, \ldots\},$$

and so on, since such an enumeration immediately suggests a bijection —just take $f(0) = a_0, f(1) = a_1, \ldots$

For example, we can show that the union of any finite number of countably infinite sets is countably infinite. Let us only illustrate the proof for the case of three pairwise disjoint, countably infinite sets; a similar argument works in general. Call the sets A, B, and C. The sets can be listed as above: $A = \{a_0, a_1, \ldots\}$, $B = \{b_0, b_1, \ldots\}$, $C = \{c_0, c_1, \ldots\}$, Then their union can be listed as $A \cup B \cup C = \{a_0, b_0, c_0, a_1, b_1, c_1, a_2, \ldots\}$. This listing amounts to a way of "visiting" all the elements in $A \cup B \cup C$ by alternating between different sets, as illustrated in Figure 1-7. The technique of interweaving the enumeration of several sets is called "dovetailing" (for reasons that any carpenter can give after looking at Figure 1-7).

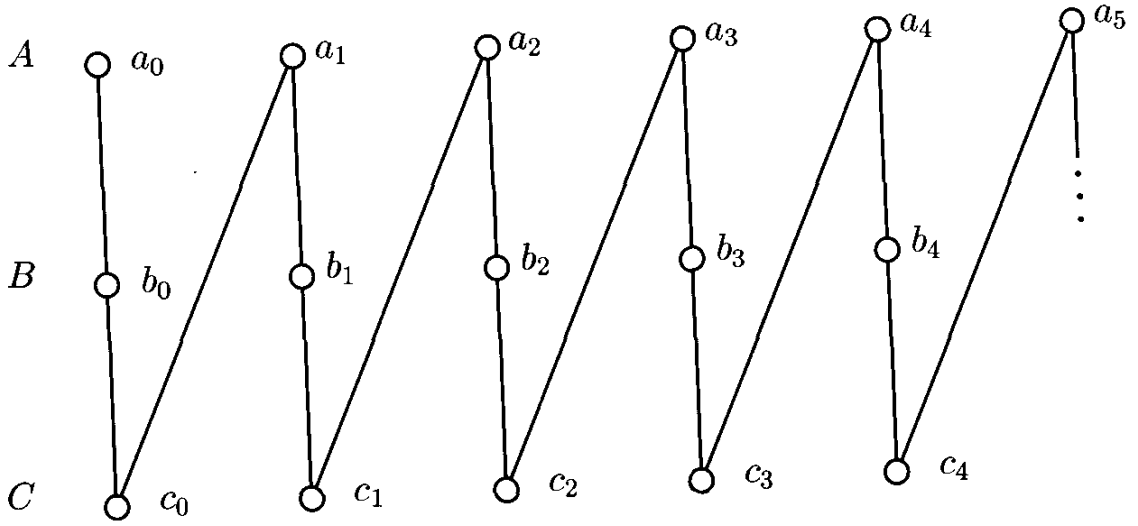


Figure 1-7

The same idea can be used to show that the union of a countably infinite collection of countably infinite sets is countably infinite. For example, let us show that $\mathbf{N} \times \mathbf{N}$ is countably infinite; note that $\mathbf{N} \times \mathbf{N}$ is the union of $\{0\} \times \mathbf{N}$, $\{1\} \times \mathbf{N}$, $\{2\} \times \mathbf{N}$, and so on, that is, the union of a countably infinite collection of countably infinite sets. Dovetailing must here be more subtle than in the

example above: we cannot, as we did there, visit one element from each set before visiting the second element of the first set, because with infinitely many sets to visit we could never even finish the first round! Instead we proceed as follows (see Figure 1-8).

(1) In the first round, we visit one element from the first set: $(0,0)$.
(2) In the second round, we visit the next element from the first set, $(0,1)$, and also the first element from the second set, $(1,0)$.
(3) In the third round we visit the next unvisited elements of the first and second sets, $(0,2)$ and $(1,1)$, and also the first element of the third set, $(2,0)$.
(4) In general, in the nth round, we visit the nth element of the first set, the $(n-1)$st element of the second set, and the first element of the nth set.

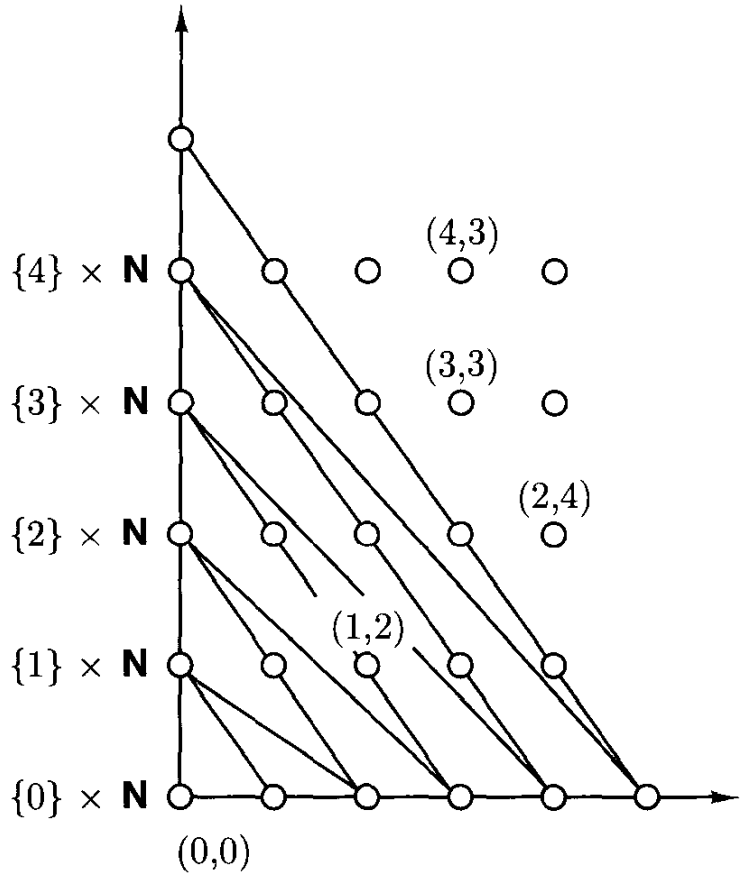


Figure 1-8

Another way of viewing this use of dovetailing is to observe that the pair (i,j) is visited mth, where $m = \frac{1}{2}[(i+j)^2 + 3i + j]$; that is to say, the function $f(i,j) = \frac{1}{2}[(i+j)^2 + 3i + j]$ is a *bijection* from $\mathbf{N} \times \mathbf{N}$ to $\mathbf{N}$ (see Problem 1.4.4).

At the end of the next section, we present a technique for showing that two infinite sets are *not* equinumerous.

Problems for Section 1.4

1.4.1. Prove that the following are countable.

(a) The union of any three countable sets, not necessarily infinite or disjoint.

(b) The set of all finite subsets of $\mathbf{N}$.

1.4.2. Explicitly give bijections between each of the following pairs.

(a) $\mathbf{N}$ and the odd natural numbers.

(b) $\mathbf{N}$ and the set of all integers.

(c) $\mathbf{N}$ and $\mathbf{N} \times \mathbf{N} \times \mathbf{N}$.

(We are looking for formulas that are as simple as possible and involve only such operations as addition and multiplication.)

1.4.3. Let C be a set of sets defined as follows,

1. $\emptyset \in C$
2. If $S_1 \in C$ and $S_2 \in C$, then $\{S_1, S_2\} \in C$.
3. If $S_1 \in C$ and $S_2 \in C$, then $S_1 \times S_2 \in C$.
4. Nothing is in C except that which follows from (1), (2), and (3).

(a) Explain carefully why it is a consequence of (1–4) that $\{\emptyset, \{\emptyset\}\} \in C$.

(b) Give an example of a set S of ordered pairs such that $S \in C$, and $|S| > 1$.

(c) Does C contain any infinite sets? Explain.

(d) Is C countable or uncountable? Explain.

1.4.4. Show that the dovetailing method of Figure 1-8 visits the pair (i, j) mth, where

$$m = \frac{1}{2}[(i+j)^2 + 3i + j].$$

1.5 THREE FUNDAMENTAL PROOF TECHNIQUES

Every proof is different, since every proof is designed to establish a different result. But like games of chess or baseball, observation of many leads one to realize that there are patterns, rules of thumb, and tricks of the trade that can be found and exploited over and over again. The main purpose of this section is to introduce three fundamental principles that recur, under various disguises, in many proofs: *mathematical induction*, the *pigeonhole principle*, and *diagonalization*.

The Principle of Mathematical Induction: *Let A be a set of natural numbers such that*

(1) $0 \in A$, *and*
(2) for each natural number n, *if* $\{0, 1, \ldots, n\} \subseteq A$, *then* $n + 1 \in A$.

Then $A = \mathbf{N}$.

In less formal terms, the principle of mathematical induction states that any set of natural numbers containing zero, and with the property that it contains $n + 1$ whenever it contains all the numbers up to and including n, must in fact be the set of all natural numbers.

The justification for this principle should be clear intuitively; every natural number must wind up in A since it can be "reached" from zero in a finite succession of steps by adding one each time. Another way to argue the same idea is by contradiction; suppose (1) and (2) hold but $A \neq \mathbf{N}$. Then some number is omitted from A. In particular, let n be the first number among $0, 1, 2, \ldots$ that is omitted from $\mathbf{N}$.† Then n cannot be zero, since $0 \in A$ by (1); and since $0, 1, \ldots, n-1 \subseteq A$ by the choice of n, then $n \in A$ by (2), which is a contradiction.

In practice, induction is used to prove assertions of the following form: "For all natural numbers n, property P is true." The above principle is applied to the set $A = \{n : P \text{ is true of } n\}$ in the following way.

(1) In the *basis step* we show that $0 \in A$, that is, that P is true of 0.
(2) The *induction hypothesis* is the assumption that for some fixed but arbitrary $n \geq 0$, P holds for each natural number $0, 1, \ldots, n$.
(3) In the *induction step* we show, using the induction hypothesis, that P is true of $n + 1$. By the induction principle, A is then equal to $\mathbf{N}$, that is, P holds for every natural number.

Example 1.5.1: Let us show that for any $n \geq 0$, $1 + 2 + \cdots + n = \frac{n^2+n}{2}$.

Basis Step. Let $n = 0$. Then the sum on the left is zero, since there is nothing to add. The expression on the right is also zero.

Induction Hypothesis. Assume that, for some $n \geq 0$, $1 + 2 + \cdots + m = \frac{m^2+m}{2}$ whenever $m \leq n$.

---

† This is a use of another principle, called the *least number principle*, that is actually equivalent to the principle of mathematical induction, so we are not really "proving" the principle of mathematical induction. The least number principle is: If $A \subseteq \mathbf{N}$ and $A \neq \mathbf{N}$, then there is a unique least number $n \in \mathbf{N} - A$; that is, a unique number n such that $n \notin A$ but $0, 1, \ldots, n-1 \in A$. A somewhat frivolous example of the least number principle is the fact that *there are no uninteresting numbers*. For suppose there were; then there would have to be a least such number, say n. But then n would have the remarkable property of being the least uninteresting number, which would surely make n interesting...

Induction Step.

$$\begin{aligned} 1+2+\cdots+n+(n+1) &= (1+2+\cdots+n)+(n+1) \\ &= \frac{n^2+n}{2}+(n+1) \quad \text{(by the induction hypothesis)} \\ &= \frac{n^2+n+2n+2}{2} \\ &= \frac{(n+1)^2+(n+1)}{2} \end{aligned}$$

as was to be shown.$\diamondsuit$

Example 1.5.2: For any finite set A, $|2^A| = 2^{|A|}$; that is, the cardinality of the power set of A is 2 raised to a power equal to the cardinality of A. We shall prove this statement *by induction on the cardinality of A.*

Basis Step. Let A be a set of cardinality $n = 0$. Then $A = \emptyset$, and $2^{|A|} = 2^0 = 1$; on the other hand, $2^A = \{\emptyset\}$, and $|2^A| = |\{\emptyset\}| = 1$.

Induction Hypothesis. Let $n > 0$, and suppose that $|2^A| = 2^{|A|}$ provided that $|A| \leq n$.

Induction Step. Let A be such that $|A| = n+1$. Since $n > 0$, A contains at least one element a. Let $B = A - \{a\}$; then $|B| = n$. By the induction hypothesis, $|2^B| = 2^{|B|} = 2^n$. Now the power set of A can be divided into two parts, those sets containing the element a and those sets not containing a. The latter part is just 2^B, and the former part is obtained by introducing a into each member of 2^B. Thus

$$2^A = 2^B \cup \{C \cup \{a\} : C \in 2^B\}.$$

This division in fact partitions 2^A into two disjoint equinumerous parts, so the cardinality of the whole is twice $2^{|B|}$, which, by the induction hypothesis, is $2 \cdot 2^n = 2^{n+1}$, as was to be shown.$\diamondsuit$

We next use induction to establish our second fundamental principle, the *pigeonhole principle.*

The Pigeonhole Principle: *If A and B are finite sets and $|A| > |B|$, then there is no one-to-one function from A to B.*

In other words, if we attempt to pair off the elements of A (the "pigeons") with elements of B (the "pigeonholes"), sooner or later we will have to put more than one pigeon in a pigeonhole.

Proof: *Basis Step.* Suppose $|B| = 0$, that is, $B = \emptyset$. Then there is *no function* $f : A \mapsto B$ *whatsoever*, let alone a one-to-one function.

Induction Hypothesis. Suppose that f is not one-to-one, provided that $f : A \mapsto B$, $|A| > |B|$, and $|B| \leq n$, where $n \geq 0$.

Induction Step. Suppose that $f : A \mapsto B$ and $|A| > |B| = n + 1$. Choose some $a \in A$ (since $|A| > |B| = n + 1 \geq 1$, A is nonempty, and therefore such a choice is possible). If there is another element of A, say a', such that $f(a) = f(a')$, then obviously f is not a one-to-one function, and we are done. So, suppose that a is the only element mapped by f to $f(a)$. Consider then the sets $A - \{a\}$, $B - \{f(a)\}$, and the function g from $A - \{a\}$ to $B - \{f(a)\}$ that agrees with f on all elements of $A - \{a\}$. Now the induction hypothesis applies, because $B - \{f(a)\}$ has n elements, and $|A - \{a\}| = |A| - 1 > |B| - 1 = |B - \{f(a)\}|$. Therefore, there are two distinct elements of $A - \{a\}$ that are mapped by g (and therefore by f) to the same element of $B - \{b\}$, and hence f is not one-to-one. ■

This simple fact is of use in a surprisingly large variety of proofs. We present just one simple application here, but point out other cases as they arise in later chapters.

Theorem 1.5.1: *Let R be a binary relation on a finite set A, and let $a, b \in A$. If there is a path from a to b in R, then there is a path of length at most $|A|$.*

Proof: Suppose that $(a_1, a_2, \ldots, a_n)$ is the *shortest* path from $a_1 = a$ to $a_n = b$, that is, the path with the smallest length, and suppose that $n > |A|$. By the pigeonhole principle, there is an element of A that repeats on the path, say $a_i = a_j$ for some $1 \leq i < j \leq n$. But then $(a_1, a_2, \ldots, a_i, a_{j+1}, \ldots, a_n)$ is a shorter path from a to b, contradicting our assumption that $(a_1, a_2, \ldots, a_n)$ is the shortest path from a to b. ■

Finally, we come to our third basic proof technique, the *diagonalization principle.* Although it is not as widely used in mathematics as the other two principles we have discussed, it seems particularly well-suited for proving certain important results in the theory of computation.

The Diagonalization Principle: *Let R be a binary relation on a set A, and let D, the diagonal set for R, be $\{a : a \in A \text{ and } (a, a) \notin R\}$. For each $a \in A$, let $R_a = \{b : b \in A \text{ and } (a, b) \in R\}$. Then D is distinct from each R_a.*

If A is a finite set, then R can be pictured as a square array; the rows and columns are labeled with the elements of A and there is a cross in the box with row labeled a and column labeled b just in case $(a, b) \in R$. The diagonal set D corresponds to the complement of the sequence of boxes along the main diagonal, boxes with crosses being replaced by boxes without crosses, and vice versa. The sets R_a correspond to the rows of the array. The diagonalization principle can then be rephrased: the complement of the diagonal is different from each row.

Example 1.5.3: Let us consider the relation $R = \{(a,b),(a,d),(b,b),(b,c),(c,c),(d,b),(d,c),(d,e),(d,f),(e,e),(e,f),(f,a),(f,c),(f,d),(f,e)\}$; notice that $R_a = \{b,d\}$, $R_b = \{b,c\}$, $R_c = \{c\}$, $R_d = \{b,c,e,f\}$, $R_e = \{e,f\}$ and $R_f = \{a,c,d,e\}$. All in all, R may be pictured like this:

	a	b	c	d	e	f
a		×		×		
b		×	×			
c			×			
d		×	×		×	×
e					×	×
f	×		×	×	×	

The sequence of boxes along the diagonal is

	×	×		×	

Its complement is

×			×		×

which corresponds to the diagonal set $D = \{a,d,f\}$. Indeed, D is different from each row of the array; for D, because of the way it is constructed, differs from the first row in the first position, from the second row in the second position, and so on.◇

The diagonalization principle holds for infinite sets as well, for the same reason: The diagonal set D always differs from the set R_a on the question of whether a is an element, and hence cannot be the same as R_a for any a.

We illustrate the use of diagonalization by a classic theorem of Georg Cantor (1845–1918).

Theorem 1.5.2: *The set* $2^{\mathbf{N}}$ *is uncountable.*

Proof: . Suppose that $2^{\mathbf{N}}$ is countably infinite. That is, we assume that that there is a way of enumerating all members of $2^{\mathbf{N}}$ as

$$2^{\mathbf{N}} = \{R_0, R_1, R_2, \ldots\}$$

(notice that these are the sets R_a in the statement of the diagonalization principle, once we consider the relation $R = \{(i, j) : j \in R_i\}$). Now consider the set

$$D = \{n \in \mathbf{N} : n \notin R_n\}$$

(this is the the diagonal set). D is a set of natural numbers, and therefore it should appear somewhere in the enumeration $\{R_0, R_1, R_2, \ldots\}$ But D cannot be R_0, because it differs from it with respect to containing 0 (it does if and only if R_0 does not); and it cannot be R_1 because it differs from it with respect to 1; and so on. We must conclude that D does not appear on the enumeration at all, and this is a contradiction.

To restate the argument a little more formally, suppose that $D = R_k$ for some $k \geq 0$ (since D is a set of natural numbers, and $\{R_0, R_1, R_2, \ldots\}$ was supposed to be a complete enumeration of all such sets, such a k must exist). We obtain a contradition by asking whether $k \in R_k$:

(a) Suppose the answer is yes, $k \in R_k$. Since $D = \{n \in \mathbf{N} : n \notin R_n\}$, it follows that $k \notin D$; but $D = R_k$, a contradiction.
(b) Suppose the answer is no, $k \notin R_k$; then $k \in D$. But D is R_k, so $k \in R_k$, another contradiction.

We arrived at this contradiction starting from the assumption that $2^{\mathbf{N}}$ is countably infinite, and continuing by otherwise impeccably rigorous mathematical reasoning; we must therefore conclude that this asumption was in error. Hence $2^{\mathbf{N}}$ is uncountable. ■

For a different rendering of this proof, in terms of establishing that the set of real numbers in the interval $[0, 1]$ is uncountable, see Problem 1.5.11.

Problems for Section 1.5

1.5.1. Show by induction that

$$1 \cdot 2 \cdot 3 + 2 \cdot 3 \cdot 4 + \cdots + n \cdot (n+1) \cdot (n+2) = \frac{n \cdot (n+1) \cdot (n+2) \cdot (n+3)}{4}.$$

1.5.2. Show by induction that $n^4 - 4n^2$ is divisible by 3 for all $n \geq 0$.

1.5.3. What is wrong with the following purported proof that all horses are the same color?
Proof by induction on the number of horses:
Basis Step. There is only one horse. Then clearly all horses have the same color.
Induction Hypothesis. In any group of up to n horses, all horses have the same color.
Induction Step. Consider a group of $n+1$ horses. Discard one horse; by the induction hypothesis, all the remaining horses have the same color. Now put that horse back and discard another; again all the remaining horses have the same color. So all the horses have the same color as the ones that were not discarded either time, and so they all have the same color.

1.5.4. Show that, if A and B are any finite sets, then there are $|B|^{|A|}$ functions from A to B.

1.5.5. Prove by induction: Every partial order on a nonempty finite set has at least one minimal element. Need this statement be true if the requirement of finiteness is lifted?

1.5.6. Show that in any group of at least two people there are at least two persons that have the same number of acquaintances within the group. (Use the pigeonhole principle.)

1.5.7. Suppose we try to prove, by an argument exactly parallel to the proof of Theorem 1.5.2, that the set of all finite subsets of $\mathbf{N}$ is uncountable. What goes wrong?

1.5.8. Give examples to show that the intersection of two countably infinite sets can be either finite or countably infinite, and that the intersection of two uncountable sets can be finite, countably infinite, or uncountable.

1.5.9. Show that the difference of an uncountable set and a countable set is uncountable.

1.5.10. Show that if S is any set, then there is a one-to-one function from S to 2^S, but not vice versa.

1.5.11. Show that the set of all real numbers in the interval $[0, 1]$ is uncountable. (*Hint:* It is well known that each such number can be written in binary notation as an infinite sequence of 0s and 1s —such as .0110011100000... Assume that an enumeration of these sequences exists, and create a "diagonal" sequence by "flipping" the ith bit of the ith sequence.)

1.6 CLOSURES AND ALGORITHMS

Consider the two directed graphs R and R^* in Figure 1-9(a) and (b). R^* contains R; also, R^* is reflexive and transitive (whereas R is neither). In fact, it is easy to see that R^* is *the smallest possible directed graph* that has these properties —that is, contains R, is reflexive, and is transitive (by "smallest" we mean the one with the fewest edges). For this reason, R^* is called the *reflexive transitive closure of* R. We next define this useful concept formally:

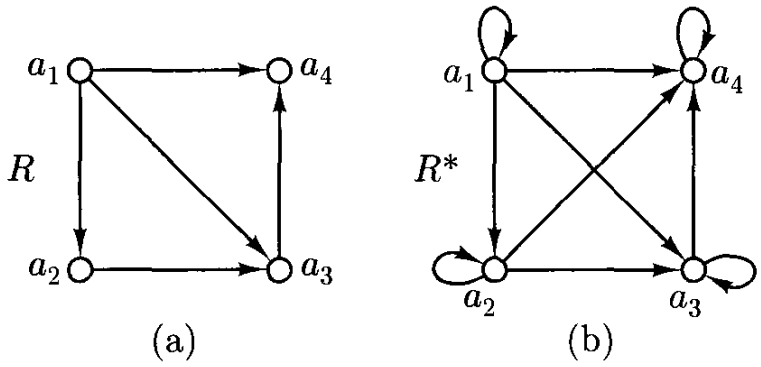


Figure 1-9

Definition 1.6.1: Let $R \subseteq A^2$ be a directed graph defined on a set A. The **reflexive transitive closure** of R is the relation

$$R^* = \{(a, b) : a, b \in A \text{ and there is a path from } a \text{ to } b \text{ in } R.\}$$

Notice the interesting contrast between the definition "from below" articulated in Definition 1.6.1 and the informal definition "from above" whereby we introduced the reflexive transitive closure (the smallest relation that contains R and is reflexive and transitive). It is perhaps intuitively clear that the two definitions are equivalent. Towards the end of this section we shall study more closely such "definitions from above," and the reason why they always correspond to an alternative definition "from below."

Algorithms

Definition 1.6.1 immediately suggests an *algorithm* for *computing* the reflexive transitive closure R^* of any given binary relation R over some finite set $A = \{a_1, a_2, \ldots, a_n\}$:

Initially $R^* := \emptyset$
for $i = 1, \ldots, n$ do
  for each i-tuple $(b_1, \ldots, b_i) \in A^i$ do
  if $(b_1, \ldots, b_i)$ is a path in R then add (b_1, b_i) to R^*.

The rigorous study of algorithms is in some sense what this book is all about; but to give a formal definition of an algorithm at this point would be to spoil the nice story we have to tell. Until a formal general model for algorithms is introduced, we shall be treating algorithms informally and somewhat nonrigorously, gradually increasing our insight and familiarity. It will then be clear that our definition of an algorithm, when its time comes, indeed captures correctly this important concept. Accordingly, this section only contains an intuitive treatment of algorithms and an informal introduction to their analysis.

Fortunately, it is easy to tell an algorithm when you see one. The procedure we describe above for computing R^* is a detailed and unambiguous sequence of instructions that produces a result —what we call R^*. It consists of elementary steps which we may reasonably assume are easy to carry out: Initializing R^* to $\emptyset$, adding new elements to R^*, testing whether $(b_j, b_{j+1}) \in R$ —this latter has to be done $i - 1$ times in the last line of the algorithm to test whether $(b_1, \ldots, b_i)$ is a path of R. We assume that somehow this algorithm operates on elements of A and R directly, so we need not specify how such sets and relations are *represented* for manipulation by the algorithm.

We shall next argue that the relation R^* computed by this algorithm is indeed the reflexive transitive closure of R (that is, the algorithm is *correct*). The reason is that our algorithm is just a straightforward *implementation* of Definition 1.6.1. It adds to R^*, initially empty, all pairs of elements of A that are connected by a path in R. Possible paths are checked one by one, in increasing length. We stop at sequences of length n because, by Theorem 1.5.1, if two nodes of A are connected by a path, then there is a path of length n or less that connects them.

It is thus clear that our algorithm will eventually terminate with the correct answer. One question that will prove most important and relevant to our concerns in this book is, *after how many steps will it terminate?* In the last part of the book we shall develop a whole theory of how the answer to such questions is calculated, and when the result is deemed satisfactory. But let us proceed informally for now.

What we need is an indication of how many elementary steps, how much "time," the algorithm requires when presented with a relation R as an input. As it is reasonable to expect that the algorithm will take more time on larger input relations, the answer will depend on how large the input relation is —more concretely, it will depend on the number n of elements in the set A. Thus, we are seeking a function $f : \mathbf{N} \mapsto \mathbf{N}$ such that, for each $n \geq 1$, if the algorithm is

presented with a binary relation $R \subseteq A \times A$ with $|A| = n$, it will terminate after at most $f(n)$ steps. As is typical in the analysis of algorithms, we shall allow $f(n)$ to be a rough overestimate, as long as it has the correct *rate of growth.* Rates of growth are thus our next topic.

The Growth of Functions

Of these three functions from the set of natural numbers to itself, which is the largest?

$$f(n) = 1,000,000 \cdot n; \quad g(n) = 10 \cdot n^3; \quad h(n) = 2^n$$

Although for all values of n up to about a dozen we have $f(n) > g(n) > h(n)$, it should be intuitively clear that the correct ranking is the exact opposite; if we take n *large enough* we shall eventually have $f(n) < g(n) < h(n)$. In this subsection we shall develop the concepts necessary for ranking such functions according to their ultimate potential for producing large values.

Definition 1.6.2: Let $f : \mathbf{N} \mapsto \mathbf{N}$ be a function from the natural numbers to the natural numbers. The **order of** f, denoted $\mathcal{O}(f)$, is the set of all functions $g : \mathbf{N} \mapsto \mathbf{N}$ with the property that there are positive natural numbers $c > 0$ and $d > 0$ such that, for all $n \in \mathbf{N}$, $g(n) \leq c \cdot f(n) + d$. If in fact this inequality holds for all n, we say that $g(n) \in \mathcal{O}(f(n))$ **with constants** c **and** d.

If for two functions $f, g : \mathbf{N} \mapsto \mathbf{N}$ we have that $f \in \mathcal{O}(g)$ and $g \in \mathcal{O}(f)$, then we write $f \asymp g$. It is clear that the relation $\asymp$ defined on functions is an equivalence relation: It is reflexive (because always $f \in \mathcal{O}(f)$, with constants 1 and 0) and it is symmetric (because the roles of f and g are completely interchangeable in the definition of $\asymp$). Finally, it is transitive. Because suppose that $f \in \mathcal{O}(g)$ with constants c, d, and $g \in \mathcal{O}(h)$ with constants c', d'. Then for all n,

$$f(n) \leq c \cdot g(n) + d \leq c \cdot (c' \cdot h(n) + d') + d = (c \cdot c')h(n) + (d + c \cdot d').$$

Thus $f \in \mathcal{O}(h)$ with constants $c \cdot c'$ and $d + c \cdot d'$.

Thus, all functions from the set of natural numbers to itself are partitioned by $\asymp$ into equivalence classes. The equivalence class of f with respect to $\asymp$ is called the **rate of growth** of f.

Example 1.6.1: Consider the *polynomial* $f(n) = 31n^2 + 17n + 3$. We claim that $f(n) \in \mathcal{O}(n^2)$. To see this, notice that $n^2 \geq n$, and so $f(n) \leq 48n^2 + 3$, and thus $f(n) \in \mathcal{O}(n^2)$ with constants 48 and 3. Of course, also $n^2 \in \mathcal{O}(f(n))$ —with constants 1 and 0. Hence $n^2 \asymp 31n^2 + 17n + 3$, and the two functions have the same rate of growth.

Similarly, for any polynomial of degree d with nonnegative coefficients

$$f(n) = a_d n^d + a_{d-1} n^{d-1} + \cdots + a_1 n + a_0$$

where $a_i \geq 0$ for all i, and $a_d > 0$, it is easy to prove that $f(n) \in \mathcal{O}(n^d)$ —with constants $\sum_{i=1}^{d} a_i$ and a_0. *All polynomials of the same degree have the same rate of growth.*

Consider then two polynomials with different degrees; do they also have the same rate of growth? The answer here is negative. Since all polynomials with the same degree have the same rate of growth, it suffices to consider the two simplest representatives, namely two polynomials of the form n^i and n^j, with $0 < i < j$. Obviously, $n^i \in \mathcal{O}(n^j)$ with constants 1 and 0. We shall prove that $n^j \notin \mathcal{O}(n^i)$.

Suppose for the sake of contradiction that indeed $n^j \in \mathcal{O}(n^i)$ with constants c and d. That is, for all $n \in \mathbf{N}$ $n^j \leq cn^i + d$. But this is easily found to be absurd by trying $n = c + d$:

$$c(c+d)^i + d < (c+d)^{i+1} \leq (c+d)^j.$$

To summarize, for any two polynomials f and g, if they have the same degree then $f \asymp g$. Otherwise, if g has larger degree than f, then $f \in \mathcal{O}(g)$ but $g \notin \mathcal{O}(f)$; that is, g *has higher rate of growth* than f.$\diamondsuit$

Example 1.6.2: The previous example suggests that the rate of growth of a polynomial is captured by its degree. The larger the degree of f the higher the rate of growth. But there are functions with rate of growth higher than that of any polynomial. A simple example is the *exponential* function 2^n.

Let us first establish that, for all $n \in \mathbf{N}$, $n \leq 2^n$. We shall use induction on n. The result certainly holds when $n = 0$. So, suppose that it holds for all natural numbers up to and including n. Then we have

$$n + 1 \leq 2^n + 1 \leq 2^n + 2^n = 2^{n+1},$$

where in the first inequality we used the induction hypothesis, and in the second we used the fact that $1 \leq 2^n$ for all n.

We shall now extend this easy fact to all polynomials. We shall show that for any $i \geq 1$, $n^i \in \mathcal{O}(2^n)$; that is,

$$n^i \leq c2^n + d \tag{1}$$

for appropriate constants c and d. Take $c = (2i)^i$ and $d = (i^2)^i$. There are two cases: If $n \leq i^2$, then the inequality holds because $n^i \leq d$. If on the other hand $n \geq i^2$, then we shall show that the inequality (1) again holds because now $n^i \leq c2^n$. In proof, let m be the quotient of n divided by i —the unique integer

such that $im \leq n < im + i$. Then we have

$$\begin{aligned} n^i \leq & (im+i)^i \quad \text{by the definition of } m \\ = & i^i(m+1)^i \\ \leq & i^i(2^{m+1})^i \quad \text{by the inequality } n \leq 2^n \text{ applied to } n = m+1 \\ \leq & c2^{mi} \quad \text{by recalling that } c = (2i)^i \\ \leq & c2^n \quad \text{again by the definition of } m. \end{aligned}$$

Thus the rate of growth of any polynomial is no faster than that of 2^n. Can a polynomial have the same rate of growth as 2^n? If so, then any polynomial of larger degree would have the same rate of growth as well; and we saw in the previous example that polynomials of different degrees have different rates of growth. We conclude that 2^n *has a higher rate of growth than any polynomial.* Other exponential functions, such as 5^n, n^n, $n!$, 2^{n^2}, or, worse, 2^{2^n}, have even higher rates of growth.$\Diamond$

Analysis of Algorithms

Polynomial and exponential rates of growth arise very naturally in the *analysis of algorithms.* For example, let us derive a rough estimate of the number of steps taken by the algorithm for the reflexive transitive closure introduced in the beginning of this section.

The algorithm examines each sequence $(b_1, \ldots, b_i)$ of length up to n to determine whether it is a path of R, and, if the answer is "yes," it adds (b_1, b_i) to R^*. Each such repetition can definitely be carried out in n or fewer "elementary operations" —testing whether $(a, b) \in R$, adding (a, b) to R^*. Thus, the total number of operations can be no more than

$$n \cdot (1 + n + n^2 + \cdots + n^n),$$

which is in $\mathcal{O}(n^{n+1})$. We have thus determined that the rate of growth of the *time requirements* of this algorithm is $\mathcal{O}(n^{n+1})$.

This outcome is rather disappointing because it is an *exponential* function, with an even higher rate of growth than 2^n. This algorithm is not efficient at all!

The question then arises, is there a faster algorithm for computing the reflexive transitive closure? Consider the following:

Initially $R^* := R \cup \{(a_i, a_i) : a_i \in A\}$
while there are three elements $a_i, a_j, a_k \in A$ such that
$(a_i, a_j), (a_j, a_k) \in R^*$ but $(a_i, a_k) \notin R^*$ do:
  add (a_i, a_k) to R^*.

This algorithm is perhaps even more natural and intuitive than the first one. It starts by adding to R^* all pairs in R and all pairs of the form (a_i, a_i) —thus guaranteeing that R^* contains R and is reflexive. It then repeatedly looks for violations of the transitivity property. Presumably this search for violations proceeds in some organized way not spelled out exactly in the algorithm, say by running through all values of a_i, for each of them through all values of a_j, and finally of a_k. If such a violation is discovered then it is corrected by adding an appropriate pair to R^*, and the search for a violation must start all over again. If at some point all triples are examined and no violation is found, the algorithm terminates.

When the algorithm terminates, R^* will certainly contain R, and it will be reflexive; besides, since no violation was found in the last iteration of the while loop, R^* must be transitive. Furthermore, R^* is the smallest relation that has all these properties (and it is thus the sought reflexive transitive closure of R). To see why, suppose that there is a proper subset of R^*, call it R_0, that contains R, and is reflexive and transitive. Consider then the first pair that does not belong to R_0, and still was added to R^* by the algorithm. It cannot be a pair in R, or a pair of the form (a_i, a_i), because all these pairs do belong to R_0. Thus, it must be a pair (a_i, a_k) such that both (a_i, a_j) and (a_j, a_k) belong to R^* at that point —and therefore also to R_0, since (a_i, a_k) was the first pair in which the two relations differ. But then R_0, by hypothesis a transitive relation, must also contain (a_i, a_k) —a contradiction. Therefore, the result will be the reflexive transitive closure of R and the algorithm is correct.

But when will the algorithm terminate? The answer is *after at most n^2 iterations of the while loop.* This is because after each successful iteration a pair (a_i, a_k) is added to R^* that was not there before, and there are at most n^2 pairs to be added. Hence the algorithm will terminate after at most n^2 iterations. And since each iteration can be carried out in $\mathcal{O}(n^3)$ time (all triples of elements of A need to be searched), the complexity of the new algorithm is $\mathcal{O}(n^2 \times n^3) = \mathcal{O}(n^5)$ —a *polynomial* rate of growth, and thus a spectacular improvement over our exponential first attempt.

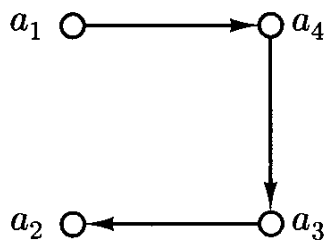


Figure 1-10

Example 1.6.3: Let us see how the new algorithm performs on the graph in Figure 1-10. We start in the first line with the graph plus all self-loops. Suppose

that the search for triples (a_i, a_j, a_k) that violate transitivity is conducted in the "natural" order

$$(a_1,a_1,a_1),(a_1,a_1,a_2),\ldots,(a_1,a_1,a_4),(a_1,a_2,a_1),(a_1,a_2,a_2),\ldots,(a_4,a_4,a_4).$$

The first violation thus discovered is (a_1, a_4, a_3), so edge (a_1, a_3) is added. We next start checking all triples from the beginning —because the introduction of (a_1, a_3) may have caused a violation of transitivity in a triple that was examined in the previous iteration. Indeed, the next violation discovered is (a_1, a_3, a_2), and so (a_1, a_2) is added. We start once again from the beginning, this time to discover a violation for (a_4, a_3, a_2) —edge (a_4, a_2) is added. In the next iteration we go through all triples without discovering a violation, and hence we conclude that we have computed the reflexive transitive closure of the graph.$\Diamond$

This example illustrates the source of our algorithm's relative inefficiency, reflected in the steep n^5 growth of its complexity: A triple (a_i, a_j, a_k) must be tested again and again for possible violation of transitivity, since a newly inserted pair may create new violations *in triples that have been already checked.*

Can we do better? In particular, is there a way to order the triples so that newly added pairs never introduce transitivity violations in the already examined triples? Such an ordering would yield an $\mathcal{O}(n^3)$ algorithm, because each triple would then have to be examined once and only once.

As it turns out, such an ordering is possible: We order all triples (a_i, a_j, a_k) to be searched in *increasing* j —the middle index! We first look systematically for violations of transitivity of the form (a_i, a_1, a_k); those that are found are corrected by adding the appropriate pairs, and the search continues from the point it stopped. Once all triples of the form (a_i, a_1, a_k) have been examined, we look at all triples of the form (a_i, a_2, a_k), then (a_i, a_3, a_k), and so on. The precise order in which the triples within each group are examined is immaterial. Once we examine the last group, all triples of the form (a_i, a_n, a_k), then we stop, having computed the reflexive transitive closure. The full algorithm is this:

Initially $R^* := R \cup \{(a_i, a_i) : a_i \in A\}$
for each $j = 1, 2, \ldots, n$ do
  for each $i = 1, 2, \ldots, n$ and $k = 1, 2, \ldots, n$ do
  if $(a_i, a_j), (a_j, a_k) \in R^*$ but $(a_i, a_k) \notin R^*$ then add (a_i, a_k) to R^*.

Why does this idea work? Consider a path $(a_{i_0}, a_{i_1}, \ldots, a_{i_{k-1}}, a_{i_k})$ from a_{i_0} to a_{i_k}, where $1 \leq i_j \leq n$ for all j. Define the *rank* of the path to be the largest integer among $i_1, \ldots, i_{k-1}$; that is, the largest index appearing in any intermediate node. Trivial paths, such as edges and self-loops, have rank zero, as they have no intermediate nodes.

With this terminology we can argue about the correctness of our latest algorithm. In particular, we can prove the following statement: *For each* $j =$

$0, \ldots, n$, *immediately after the* j*th execution of the outer loop,* R^* *contains all pairs* (a_i, a_k) *such that there is a path of rank* j *or less from* a_i *to* a_k *in* R. Notice that, since all paths have rank at most n, this implies that in the end all pairs joined by any path will have been added in R^*.

The proof of this statement is by induction on j. The result is certainly true when $j = 0$ —no iterations yet, and accordingly R^* contains only pairs connected by trivial paths, of rank 0. For the induction hypothesis, suppose that the result holds for j up to some value, say $m < n$, and consider the $m+1$st iteration. We must show that the $m+1$st iteration adds to R^* precisely those pairs that are connected in R by paths of rank equal to $m+1$, that is, in which the highest-indexed node is a_{m+1}. If two nodes a_i and a_k are connected by such a path, then they must be connected by such a path in which a_{m+1} appears exactly once —if a_{m+1} appears more than once, then omit the portion of the path between the first and the last appearance— while all other intermediate nodes are indexed m or less. And such a path would consist of a path of rank m or less from a_i to a_{m+1}, followed by a path of rank m or less from a_{m+1} to a_k. By the induction hypothesis, both pairs (a_i, a_{m+1}) and (a_{m+1}, a_k) must be in R^* at this point. Therefore, the algorithm will discover that $(a_i, a_{m+1}), (a_{m+1}, a_k) \in R^*$, but $(a_i, a_k) \notin R^*$, and will add the pair (a_i, a_k) to R^*. Conversely, the $m+1$st iteration will add to R^* only pairs (a_i, a_j) that are connected by a path of rank exactly $m+1$. The algorithm is correct.

Example 1.6.3 (continued): If we apply this algorithm to the graph in Figure 1-10, no edges are added when $j = 1$ and $j = 2$. Then when $j = 3$ the edge (a_4, a_2) is added, and when $j = 4$ the edges (a_1, a_3) and (a_1, a_2) are added. ◇

Closure Properties and Closures

The transitive closure of a relation is just one instance of an important style of defining larger sets (or relations) starting from smaller ones.

Definition 1.6.3: Let D be a set, let $n \geq 0$, and let $R \subseteq D^{n+1}$ be a $(n+1)$-ary relation on D. Then a subset B of D is said to be **closed under** R if $b_{n+1} \in B$ whenever $b_1, \ldots, b_n \in B$ and $(b_1, \ldots, b_n, b_{n+1}) \in R$. Any property of the form "the set B is closed under relations $R_1, R_2, \ldots, R_m$" is called a **closure property** of B.

Example 1.6.4: The set of a person's ancestors is closed under the relation

$$\{(a, b) : a \text{ and } b \text{ are persons, and } b \text{ is a parent of } a\},$$

since the parent of an ancestor is also an ancestor. ◇

Example 1.6.5: Let A be a fixed set. We say that set S satisfies the *inclusion property associated with* A if $A \subseteq S$. Any inclusion property is a closure property, by taking the relation R to be the unary relation $\{(a) : a \in A\}$ (notice that we must take $n = 0$ in Definition 1.6.3).$\diamondsuit$

Example 1.6.6: We shall occasionally say that a set $A \subseteq D$ is closed under a *function* $f : D^k \mapsto D$. There should be no mystery as to what this means, since a function is a special kind of relation. For example, we may say that the natural numbers are *closed under addition.* We mean that for any $m, n \in \mathbf{N}$ we also have $m + n \in \mathbf{N}$ —since $(m, n, m + n)$ is a triple in the "addition relation" over the natural numbers. $\mathbf{N}$ is also closed under multiplication, but it is not closed under subtraction.$\diamondsuit$

Example 1.6.7: Since relations are sets, we can speak of one relation as being closed under one or more others. Let D be a set, let Q be the ternary relation on D^2 (that is, a subset of $(D \times D)^3$) such that

$$Q = \{((a, b), (b, c), (a, c)) : a, b, c \in D\}.$$

Then a relation $R \subseteq D \times D$ is closed under Q if and only if it is transitive. We conclude that transitivity is a closure property. On the other hand, reflexivity is a closure property, because it is the inclusion property of the set $\{(d, d) : d \in D\}$.$\diamondsuit$

A common type of mathematical construction is to pass from a set A to *the minimal set B that contains A and has property P.* By "minimal set B" we mean "a set B that does not properly include any other set B' that also contains A and has property P." Care must be taken, when a definition of this form is used, that the set B is well defined, that is, that there exists only one such minimal set. Since a set of sets can have many minimal elements or none at all, whether B is well defined depends on the nature of property P. For example, if P is the property "has either b or c as an element" and $A = \{a\}$, then B is not well defined since both $\{a, b\}$ and $\{a, c\}$ are minimal sets with A as a subset and with property P.

However, as the following result guarantees, if P is a closure property, then the set B is always well defined:

Theorem 1.6.1: *Let P be a closure property defined by relations on a set D, and let A be a subset of D. Then there is a unique minimal set B that contains A and has property P.*

Proof: Consider the set of all subsets of D that are closed under $R_1, \ldots, R_m$ and that have A as a subset. We call this set of sets $\mathcal{S}$. We need to show that $\mathcal{S}$

has a unique minimal element B. It is easy to see that $\mathcal{S}$ is nonempty, since it contains the "universe" D —itself trivially closed under each R_i, and certainly containing A.

Consider then the set B which is the intersection of all sets in $\mathcal{S}$,

$$B = \bigcap \mathcal{S}.$$

First, B is well defined, because it is the intersection of a non-empty collection of sets. Also, it is easy to see that it contains A —since all sets in $\mathcal{S}$ do. We next claim that B is closed under all R_i's. Suppose that $a_1, \ldots, a_{n_i-1} \in B$, and $(a_1, \ldots, a_{n_i-1}, a_{n_i}) \in R_i$. Since B is the intersection of all sets in $\mathcal{S}$, it follows that all sets in $\mathcal{S}$ contain $a_1, \ldots, a_{n_i-1}$. But since all sets in $\mathcal{S}$ are closed under R_i, they all also contain a_{n_i}. Therefore B must contain a_{n_i}, and hence B is closed under R_i. Finally B is minimal, because there can be no proper subset B' of B that has these properties (B' contains A and is closed under the R_i's). Because then B' would be a member of $\mathcal{S}$, and thus it would include B. ■

We call B in Theorem 1.6.1 the **closure** of A under the relations $R_1, \ldots, R_m$.

Example 1.6.8: The set of all your ancestors (where we assume that each person is an ancestor of her- or himself) is the closure of the singleton set containing only yourself under the relation $\{(a, b) : a$ and b are persons, and b is a parent of $a\}$.◇

Example 1.6.9: The set of natural numbers **N** is the closure under addition of the set $\{0, 1\}$. **N** is closed under addition and multiplication, but not under subtraction. The set of integers (positive, negative, and zero) is the closure of **N** under subtraction.◇

Example 1.6.10: The reflexive, transitive closure of a binary relation R over some finite set A defined as

$$R^* = \{(a, b) : \text{there is a path in } R \text{ from } a \text{ to } b\}$$

(recall Definition 1.6.1) richly deserves its name: It turns out that it is the closure of R under transitivity and reflexivity —both closure properties.

First, R^* is reflexive and transitive; for there is a trivial path from a to a for any element a, and if there is a path from a to b, and a path from b to c, then there is a path from a to c. Also, clearly $R \subseteq R^*$, because there is a path from a to b whenever $(a, b) \in R$.

Finally, R^* is minimal. For let $(a, b) \in R^*$. Since $(a, b) \in R^*$, there is a path $(a = a_1, \ldots, a_k = b)$ from a to b. It follows by induction on k that (a, b) must belong to any relation that includes R and is transitive and reflexive.

The reflexive, transitive closure of a binary relation is only one of several possible closures. For example, the *transitive closure* of a relation R, denoted R^+, is the set of all (a, b) such that there is a *nontrivial* path from a to b in R —it need not be reflexive. And the *reflexive, symmetric, transitive closure* of any relation (there is no special symbol) is always an equivalence relation. As we shall show next, there are polynomial algorithms for computing all of these closures.$\diamondsuit$

Any closure property over a finite set can be computed in polynomial time! Suppose that we are given relations $R_1 \subseteq D^{r_1}, \ldots, R_k \subseteq D^{r_k}$ of various arities over the finite set D, and a set $A \subseteq D$; we are asked to compute the closure A^* of A under $R_1, \ldots, R_k$. This can be carried out in polynomial time by a straightforward generalization of the $\mathcal{O}(n^5)$ algorithm we devised for the transitive closure problem in the last subsection:

Initially $A^* := A$
while there is an index i, $1 \leq i \leq k$, and r_i elements $a_{j_1}, \ldots, a_{j_{r_i-1}} \in A^*$
and $a_{j_{r_i}} \in D - A^*$ such that $(a_{j_1}, \ldots, a_{j_{r_i}}) \in R_i$ do:
add $a_{j_{r_i}}$ to A^*.

It is a straightforward extension of our argument for the transitive closure algorithm, which we leave as an exercise (Problem 1.6.9), to show that the above algorithm is correct, and that it terminates after $\mathcal{O}(n^{r+1})$ steps, where $n = |D|$ and r is the largest integer among $r_1, \ldots, r_k$. It follows that the closure of any given finite set under any closure property defined in terms of given relations *of fixed arity* can be computed in polynomial time. As a matter of fact, in Chapter 7 we shall prove a very interesting *converse* statement: *Any* polynomial-time algorithm can be rendered as the computation of the closure of a set under some relations of fixed arity. In other words, the polynomial algorithm for closure shown above is *the mother of all polynomial algorithms.*

Problems for Section 1.6

1.6.1. Are the following sets closed under the following operations? If not, what are the respective closures?
(a) The odd integers under multiplication.
(b) The positive integers under division.
(c) The negative integers under subtraction.
(d) The negative integers under multiplication.
(e) The odd integers under division.

1.6.2. What is the reflexive transitive closure R^* of the relation $R = \{(a, b), (a, c), (a, d), (d, c), (d, e)\}$? Draw a directed graph representing R^*.

1.6.3. Is the transitive closure of the symmetric closure of a binary relation necessarily reflexive? Prove it or give a counterexample.

1.6.4. Let $R \subseteq A \times A$ be any binary relation.

(a) Let $Q = \{(a,b) : a,b \in A$ and there are paths in R from a to b and from b to $a\}$. Show that Q is an equivalence relation on A.

(b) Let Π be the partition of A corresponding to the equivalence relation Q. Let $\mathcal{R}$ be the relation $\{(S,T) : S,T \in \Pi$ and there is a path in R from some member of S to some member of $T\}$. Show that $\mathcal{R}$ is a partial order on Π.

1.6.5. Give an example of a binary relation that is not reflexive but has a transitive closure that is reflexive.

1.6.6. Recall the three functions in the beginning of the subsection on rates of growth:

$$f(n) = 1,000,000 \cdot n; \quad g(n) = 10 \cdot n^3; \quad h(n) = 2^n.$$

What are appropriate constants c and d for the inclusions $f(n) \in \mathcal{O}(g(n))$, $f(n) \in \mathcal{O}(h(n))$, and $g(n) \in \mathcal{O}(h(n))$? What is the smallest integer n such that the values $f(n) \leq g(n) \leq h(n)$?

1.6.7. Arrange these functions in order of increasing rate of growth. Identify any functions with the same rate of growth:

$$n^2, 2^n, n\lceil \log n \rceil, n!, 4^n, n^n, n^{\lceil \log n \rceil}, 2^{2^n}, 2^{2n}, 2^{2^{n+1}}.$$

1.6.8. You have five algorithms for a problem, with these running times:

$$10^6 n, \quad 10^4 n^2, \quad n^4, \quad 2^n, \quad n!$$

(a) Your computer executes 10^8 steps per second. What is the largest size n you can solve by each algorithm in a second?

(b) In a day? (Assume that a day is 10^5 seconds).

(c) How would the numbers in (a) and (b) change if you bought a computer ten times faster?

1.6.9. Show that the algorithm given in the end of this section correctly computes the closure of a set $A \subseteq D$ under the relations $R_1 \subseteq D^{r_1}, \ldots, R_k \subseteq D^{r_k}$ in $\mathcal{O}(n^r)$ time, where $n = |D|$, and r is the maximum of the arities $r_1, \ldots, r_k$. (*Hint:* The argument is a straightforward generalization of the one for the $\mathcal{O}(n^5)$ transitive closure algorithm.)

1.7 ALPHABETS AND LANGUAGES

The algorithms we studied informally in the last section have much that is left vague. For example, we have not specified exactly how the relations R and R^* that need to be accessed and modified are represented and stored. In computational practice, such *data* are encoded in the computer's memory as strings of bits or other symbols appropriate for manipulation by a computer. The mathematical study of the theory of computation must therefore begin by understanding *the mathematics of strings of symbols.*

We start with the notion of an **alphabet**: a finite set of **symbols**. An example is, naturally, the Roman alphabet $\{a, b, \ldots, z\}$. An alphabet particularly pertinent to the theory of computation is the **binary alphabet** $\{0, 1\}$. In fact, any object can be in an alphabet; from a formal point of view, an alphabet is simply a finite set of any sort. For simplicity, however, we use as symbols only letters, numerals, and other common characters such as \$, or #.

A **string** over an alphabet is a finite sequence of symbols from the alphabet. Instead of writing strings with parentheses and commas, as we have written other sequences, we simply juxtapose the symbols. Thus *watermelon* is a string over the alphabet $\{a, b, \ldots, z\}$, and 0111011 is a string over $\{0, 1\}$. Also, using the natural isomorphism, we identify a string of only one symbol with the symbol itself; thus the symbol a is the same as the string a. A string may have no symbols at all; in this case it is called the **empty string** and is denoted by e. We generally use u, v, x, y, z, and Greek letters to denote strings; for example, we might use w as a name for the string *abc*. Of course, to avoid confusion it is a good practice to refrain from using as symbols letters we also use as names of strings. The set of all strings, including the empty string, over an alphabet Σ is denoted by Σ^*.

The **length** of a string is its length as a sequence; thus the length of the string *acrd* is 4. We denote the length of a string w by $|w|$; thus $|101| = 3$ and $|e| = 0$. Alternatively (that is, via a natural isomorphism) a string $w \in \Sigma^*$ can be considered as a function $w : \{1, ..., |w|\} \mapsto \Sigma$; the value of $w(j)$, where $1 \leq j \leq |w|$, is the symbol in the jth position of w. For example, if $w = accordion$, then $w(3) = w(2) = c$, and $w(1) = a$. This alternative viewpoint brings out a possible point of confusion. Naturally, the symbol c in the third position is identical to that in the second. If, however, we need to distinguish identical symbols at different positions in a string, we shall refer to them as different **occurrences** of the symbol. That is, the symbol $\sigma \in \Sigma$ occurs in the jth position of the string $w \in \Sigma^*$ if $w(j) = \sigma$.

Two strings over the same alphabet can be combined to form a third by the operation of **concatenation**. The concatenation of strings x and y, written $x \circ y$ or simply xy, is the string x followed by the string y; formally, $w = x \circ y$ if

and only if $|w| = |x| + |y|$, $w(j) = x(j)$ for $j = 1, \ldots, |x|$, and $w(|x| + j) = y(j)$ for $j = 1, \ldots, |y|$. For example, $01 \circ 001 = 01001$, and $beach \circ boy = beachboy$. Of course, $w \circ e = e \circ w = w$ for any string w. And concatenation is associative: $(wx)y = w(xy)$ for any strings w, x, and y. A string v is a **substring** of a string w if and only if there are strings x and y such that $w = xvy$. Both x and y could be e, so every string is a substring of itself; and taking $x = w$ and $v = y = e$, we see that e is a substring of every string. If $w = xv$ for some x, then v is a **suffix** of w; if $w = vy$ for some y, then v is a **prefix** of w. Thus *road* is a prefix of *roadrunner*, a suffix of *abroad*, and a substring of both these and of *broader*. A string may have several occurrences of the same substring; for example, *ababab* has three occurrences of *ab* and two of *abab*.

For each string w and each natural number i, the string w^i is defined as

$$\begin{aligned} w^0 &= e, \quad \text{the empty string} \\ w^{i+1} &= w^i \circ w \quad \text{for each } i \geq 0 \end{aligned}$$

Thus $w^1 = w$, and $do^2 = dodo$.

This definition is our first instance of a very common type: **definition by induction**. We have already seen *proofs* by induction, and the underlying idea is much the same. There is a basis case of the definition, here the definition of w^i for $i = 0$; then when that which is being defined has been specified for all $j \leq i$, it is defined for $j = i + 1$. In the example above, w^{i+1} is defined in terms of w^i. To see exactly how any case of the definition can be traced back to the basis case, consider the example of do^2. According to the definition (with $i = 1$), $(do)^2 = (do)^1 \circ do$. Again according to the definition (with $i = 0$) $(do)^1 = (do)^0 \circ do$. Now the basis case applies: $(do)^0 = e$. So $(do)^2 = (e \circ do) \circ do = dodo$.

The **reversal** of a string w, denoted by w^R, is the string "spelled backwards": for example, $reverse^R = esrever$. A formal definition can be given by induction on the length of a string:

(1) If w is a string of length 0, then $w^R = w = e$.
(2) If w is a string of length $n + 1 > 0$, then $w = ua$ for some $a \in \Sigma$, and $w^R = au^R$.

Let us use this definition to illustrate how a proof by induction can depend on a definition by induction. We shall show that for any strings w and x, $(wx)^R = x^R w^R$. For example, $(dogcat)^R = (cat)^R (dog)^R = tacgod$. We proceed by induction on the length of x.

Basis Step. $|x| = 0$. Then $x = e$, and $(wx)^R = (we)^R = w^R = ew^R = e^R w^R = x^R w^R$.

Induction Hypothesis. If $|x| \leq n$, then $(wx)^R = x^R w^R$.

Induction Step. Let $|x| = n + 1$. Then $x = ua$ for some $u \in \Sigma^*$ and $a \in \Sigma$ such that $|u| = n$.

$$\begin{aligned}
(wx)^R &= (w(ua))^R && \text{since } x = ua \\
&= ((wu)a)^R && \text{since concatenation is associative} \\
&= a(wu)^R && \text{by the definition of the reversal of } (wu)a \\
&= au^R w^R && \text{by the induction hypothesis} \\
&= (ua)^R w^R && \text{by the definition of the reversal of } ua \\
&= x^R w^R && \text{since } x = ua
\end{aligned}$$

Now we move from the study of individual strings to the study of finite and infinite *sets* of strings. The simple models of computing machines we shall soon be studying will be characterized in terms of regularities in the way they handle many different strings, so it is important first to understand general ways of describing and combining classes of strings.

Any set of strings over an alphabet Σ —that is, any subset of Σ^*— will be called a **language**. Thus Σ^*, $\emptyset$, and Σ are languages. Since a language is simply a special kind of set, we can specify a finite language by listing all its strings. For example, $\{aba, czr, d, f\}$ is a language over $\{a, b, \ldots, z\}$. However, most languages of interest are infinite, so that listing all the strings is not possible. Languages that might be considered are $\{0, 01, 011, 0111, \ldots\}$, $\{w \in \{0,1\}^* : w \text{ has an equal number of 0's and 1's}\}$, and $\{w \in \Sigma^* : w = w^R\}$. Thus we shall specify infinite languages by the scheme

$$L = \{w \in \Sigma^* : w \text{ has property } P\},$$

following the general form we have used for specifying infinite sets.

If Σ is a finite alphabet, then Σ^* is certainly infinite; but is it a countably infinite set? It is not hard to see that this is indeed the case. To construct a bijection $f : \mathbf{N} \mapsto \Sigma^*$, first fix some ordering of the alphabet, say $\Sigma = \{a_1, \ldots, a_n\}$, where $a_1, \ldots, a_n$ are distinct. The members of Σ^* can then be enumerated in the following way.

(1) For each $k \geq 0$, all strings of length k are enumerated before all strings of length $k + 1$.
(2) The n^k strings of length exactly k are enumerated **lexicographically**, that is, $a_{i_1} \ldots a_{i_k}$ precedes $a_{j_1} \ldots a_{j_k}$, provided that, for some m, $0 \leq m \leq k - 1$, $i_\ell = j_\ell$ for $\ell = 1, \ldots, m$, and $i_{m+1} < j_{m+1}$.

For example, if $\Sigma = \{0, 1\}$, the order would be as follows:

$$e, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \ldots$$

If Σ is the Roman alphabet and the ordering of $\Sigma = \{a_1, \ldots, a_{26}\}$ is the usual one $\{a, \ldots, z\}$, then the lexicographic order for strings of equal length is the order used in dictionaries; however, the ordering described by (1) and (2) for all strings in Σ^* differs from the dictionary ordering by listing shorter strings before longer ones.

Since languages are sets, they can be combined by the set operations of union, intersection, and difference. When a particular alphabet Σ is understood from context, we shall write $\overline{A}$ —the **complement** of A— instead of the difference $\Sigma^* - A$.

In addition, certain operations are meaningful only for languages. The first of these is the **concatenation of languages**. If L_1 and L_2 are languages over Σ, their concatenation is $L = L_1 \circ L_2$, or simply $L = L_1 L_2$, where

$$L = \{w \in \Sigma^* : w = x \circ y \text{ for some } x \in L_1 \text{ and } y \in L_2\}.$$

For example, if $\Sigma = \{0, 1\}$, $L_1 = \{w \in \Sigma^* : w \text{ has an even number of 0's}\}$ and $L_2 = \{w : w \text{ starts with a 0 and the rest of the symbols are 1's}\}$, then $L_1 \circ L_2 = \{w : w \text{ has an odd number of 0's}\}$.

Another language operation is the **Kleene star** of a language L, denoted by L^*. L^* is the set of all strings obtained by concatenating zero or more strings from L. (The concatenation of zero strings is e, and the concatenation of one string is the string itself.) Thus,

$$L^* = \{w \in \Sigma^* : w = w_1 \circ \cdots \circ w_k \text{ for some } k \geq 0 \text{ and some } w_1, \ldots, w_k \in L\}.$$

For example, if $L = \{01, 1, 100\}$, then $110001110011 \in L^*$, since $110001110011 = 1 \circ 100 \circ 01 \circ 1 \circ 100 \circ 1 \circ 1$, and each of these strings is in L.

Note that the use of Σ^* to denote the set of all strings over Σ is consistent with the notation for the Kleene star of Σ, regarded as a finite language. That is, if we let $L = \Sigma$ and apply the definition above, then Σ^* is the set of all strings that can be written as $w_1 \circ \cdots \circ w_k$ for some $k \geq 0$ and some $w_1, \ldots, w_k \in \Sigma$. Since the w_i are then simply individual symbols in Σ, it follows that Σ^* is, as originally defined, the set of all finite strings whose symbols are in Σ.

For another extreme example, observe that $\emptyset^* = \{e\}$. For let $L = \emptyset$ in the above definition. The only possible concatenation $w_1 \circ w_2 \circ \cdots \circ w_k$ with $k \geq 0$ and $w_1, \ldots, w_k \in L$ is that with $k = 0$, that is, the concatenation of zero strings; so the sole member of L^* in this case is e!

As a final example, let us show that if L is the language $\{w \in \{0,1\}^* : w \text{ has an unequal number of 0's and 1's}\}$, then $L^* = \{0,1\}^*$. To see this, first note that for any languages L_1 and L_2, if $L_1 \subseteq L_2$, then $L_1^* \subseteq L_2^*$ as is evident from the definition of Kleene star. Second, $\{0,1\} \subseteq L$, since each of 0 and 1, regarded as a string, has an unequal number of 0's and 1's. Hence $\{0,1\}^* \subseteq L^*$; but $L^* \subseteq \{0,1\}^*$ by definition, so $L^* = \{0,1\}^*$.

We write L^+ for the language LL^*. Equivalently, L^+ is the language

$$\{w \in \Sigma^* : w = w_1 \circ w_2 \circ \cdots \circ w_k \text{ for some } k \geq 1 \text{ and some } w_1, \ldots, w_k \in L\}.$$

Notice that L^+ can be considered as the *closure* of L under the function of concatenation. That is, L^+ is the smallest language that includes L and all strings that are concatenations of strings in L.

Problems for Section 1.7

1.7.1. (a) Prove, using the definition of concatenation given in the text, that concatenation of strings is associative.
(b) Give an inductive definition of the concatenation of strings.
(c) Using the inductive definition from (b), prove that the concatenation of strings is associative.

1.7.2. Prove each of the following using the inductive definition of reversal given in the text.
(a) $(w^R)^R = w$ for any string w.
(b) If v is a substring of w, then v^R is a substring of w^R.
(c) $(w^i)^R = (w^R)^i$ for any string w and $i \geq 0$.

1.7.3. Let $\Sigma = \{a_1, \ldots, a_{26}\}$ be the Roman alphabet. Carefully define the binary relation $<$ on Σ^* such that $x < y$ if and only if x would precede y in a standard dictionary.

1.7.4. Show each of the following.
(a) $\{e\}^* = \{e\}$
(b) For any alphabet Σ and any $L \subseteq \Sigma^*$, $(L^*)^* = L^*$.
(c) If a and b are distinct symbols, then $\{a, b\}^* = \{a\}^*(\{b\}\{a\}^*)^*$.
(d) If Σ is any alphabet, $e \in L_1 \subseteq \Sigma^*$ and $e \in L_2 \subseteq \Sigma^*$, then $(L_1\Sigma^*L_2)^* = \Sigma^*$.
(e) For any language L, $\emptyset L = L\emptyset = \emptyset$.

1.7.5. Give some examples of strings in, and not in, these sets, where $\Sigma = \{a, b\}$.
(a) $\{w :$ for some $u \in \Sigma\Sigma$, $w = uu^Ru\}$
(b) $\{w : ww = www\}$
(c) $\{w :$ for some $u, v \in \Sigma^*$, $uvw = wvu\}$
(d) $\{w :$ for some $u \in \Sigma^*$, $www = uu\}$

1.7.6. Under what circumstances is $L^+ = L^* - \{e\}$?

1.7.7. The Kleene star of a language L is the closure of L under which relations?

1.8 FINITE REPRESENTATIONS OF LANGUAGES

A central issue in the theory of computation is the representation of languages by finite specifications. Naturally, any finite language is amenable to finite representation by exhaustive enumeration of all the strings in the language. The issue becomes challenging only when infinite languages are considered.

Let us be somewhat more precise about the notion of "finite representation of a language." The first point to be made is that any such representation must itself be a string, a finite sequence of symbols over some alphabet Σ. Second, we certainly want different languages to have different representations, otherwise the term *representation* could hardly be considered appropriate. But these two requirements already imply that the possibilities for finite representation are severely limited. For the set Σ^* of strings over an alphabet Σ is countably infinite, so the number of possible representations of languages is countably infinite. (This would remain true even if we were not bound to use a particular alphabet Σ, so long as the total number of available symbols was countably infinite.) On the other hand, the set of all possible languages over a given alphabet Σ —that is, 2^{Σ^*}— is uncountably infinite, since $2^{\mathbf{N}}$, and hence the power set of any countably infinite set is not countably infinite. With only a countable number of representations and an uncountable number of things to represent, we are unable to represent all languages finitely. Thus, the most we can hope for is to find finite representations, of one sort or another, for at least some of the more interesting languages.

This is our first result in the theory of computation: No matter how powerful are the methods we use for representing languages, only countably many languages can be represented, so long as the representations themselves are finite. There being uncountably many languages in all, the vast majority of them will inevitably be missed under any finite representational scheme.

Of course, this is not the last thing we shall have to say along these lines. We shall describe several ways of describing and representing languages, each more powerful than the last in the sense that each is capable of describing languages the previous one cannot. This hierarchy does not contradict the fact that all these finite representational methods are inevitably limited in scope for the reasons just explained.

We shall also want to derive ways of exhibiting particular languages that cannot be represented by the various representational methods we study. We know that the world of languages is inhabited by vast numbers of such unrepresentable specimens, but, strangely perhaps, it can be exceedingly difficult to catch one, put it on display, and document it. Diagonalization arguments will eventually assist us here.

To begin our study of finite representations, we consider expressions —

strings of symbols— that describe how languages can be built up by using the operations described in the previous section.

Example 1.8.1: Let $L = \{w \in \{0,1\}^* : w$ has two or three occurrences of 1, the first and second of which are not consecutive$\}$. This language can be described using only singleton sets and the symbols $\cup$, $\circ$, and * as

$$\{0\}^* \circ \{1\} \circ \{0\}^* \circ \{0\} \circ \{1\} \circ \{0\}^* \circ ((\{1\} \circ \{0\}^*) \cup \emptyset^*).$$

It is not hard to see that the language represented by the above expression is precisely the language L defined above. The important thing to notice is that the only symbols used in this representation are the braces { and }, the parentheses (and), $\emptyset$, 0, 1, *, $\circ$, and $\cup$. In fact, we may dispense with the braces and $\circ$ and write simply

$$L = 0^*10^*010^*(10^* \cup \emptyset^*).$$

$\Diamond$

Roughly speaking, an expression such as the one for L in Example 1.8.1 is called a **regular expression**. That is, a regular expression describes a language exclusively by means of single symbols and $\emptyset$, combined perhaps with the symbols $\cup$ and *, possibly with the aid of parentheses. But in order to keep straight the expressions about which we are talking and the "mathematical English" we are using for discussing them, we must tread rather carefully. Instead of using $\cup$, *, and $\emptyset$, which are the names in this book for certain operations and sets, we introduce special symbols U, *, and $\oslash$, which should be regarded for the moment as completely free of meaningful overtones, just like the symbols a, b, and 0 used in earlier examples. In the same way, we introduce special symbols (and) instead of the parentheses (and) we have been using for doing mathematics. The regular expressions over an alphabet Σ^* are all strings over the alphabet $\Sigma \cup \{(,), \oslash, \mathsf{U}, ^\star\}$ that can be obtained as follows.

(1) $\oslash$ and each member of Σ is a regular expression.
(2) If α and β are regular expressions, then so is $(\alpha\beta)$.
(3) If α and β are regular expressions, then so is $(\alpha \mathsf{U} \beta)$.
(4) If α is a regular expression, then so is $\alpha^\star$.
(5) Nothing is a regular expression unless it follows from (1) through (4).

Every regular expression represents a language, according to the interpretation of the symbols U and * as set union and Kleene star, and of juxtaposition of expressions as concatenation. Formally, the relation between regular expressions and the languages they represent is established by a function $\mathcal{L}$, such that if α is any regular expression, then $\mathcal{L}(\alpha)$ is the language represented by α. That is, $\mathcal{L}$ is a function from strings to languages. The function $\mathcal{L}$ is defined as follows.

(1) $\mathcal{L}(\oslash) = \emptyset$, and $\mathcal{L}(a) = \{a\}$ for each $a \in \Sigma$.
(2) If α and β are regular expressions, then $\mathcal{L}((\alpha\beta)) = \mathcal{L}(\alpha)\mathcal{L}(\beta)$.
(3) If α and β are regular expressions, then $\mathcal{L}((\alpha \mathsf{U} \beta)) = \mathcal{L}(\alpha) \cup \mathcal{L}(\beta)$.
(4) If α is a regular expression, then $\mathcal{L}(\alpha^\star) = \mathcal{L}(\alpha)^*$.

Statement 1 defines $\mathcal{L}(\alpha)$ for each regular expression α that consists of a single symbol; then (2) through (4) define $\mathcal{L}(\alpha)$ for regular expressions of some length in terms of $\mathcal{L}(\alpha')$ for one or two regular expressions α' of smaller length. Thus every regular expression is associated in this way with some language.

Example 1.8.2: What is $\mathcal{L}(((a \mathsf{U} b)^\star a))$? We have the following.

$$\begin{aligned}
\mathcal{L}(((a \mathsf{U} b)^\star a)) &= \mathcal{L}((a \mathsf{U} b)^\star)\mathcal{L}(a) \text{ by(2)} \\
&= \mathcal{L}((a \mathsf{U} b)^\star)\{a\} \text{ by (1)} \\
&= \mathcal{L}((a \mathsf{U} b))^*\{a\} \text{ by (4)} \\
&= (\mathcal{L}(a) \cup \mathcal{L}(b))^*\{a\} \text{ by (3)} \\
&= (\{a\} \cup \{b\})^*\{a\} \text{ by (1) twice} \\
&= \{a, b\}^*\{a\} \\
&= \{w \in \{a, b\}^* : w \text{ ends with an } a\}
\end{aligned}$$

$\Diamond$

Example 1.8.3: What language is represented by $(c^\star(a \mathsf{U}(bc^\star))^\star)$? This regular expression represents the set of all strings over $\{a, b, c\}$ that do not have the substring ac. Clearly no string in $\mathcal{L}((c^\star(a \mathsf{U}(bc^\star))^\star))$ can contain the substring ac, since each occurrence of a in such a string is either at the end of the string, or is followed by another occurrence of a, or is followed by an occurrence of b. On the other hand, let w be a string with no substring ac. Then w begins with zero or more c's. If they are removed, the result is a string with no sub-string ac and not beginning with c. Any such string is in $\mathcal{L}((a \mathsf{U}(bc^\star)))$; for it can be read, left to right, as a sequence of a's, b's, and c's, with any blocks of c's immediately following b's (not following a's, and not at the beginning of the string). Therefore $w \in \mathcal{L}((c^\star(a \mathsf{U}(bc^\star))^\star))$.$\Diamond$

Example 1.8.4: $(0^\star \mathsf{U}(((0^\star(1 \mathsf{U}(11)))((00^\star)(1 \mathsf{U}(11)))^\star)0^\star))$ represents the set of all strings over $\{0, 1\}$ that do not have the substring 111.$\Diamond$

Every language that can be represented by a regular expression can be represented by infinitely many of them. For example, α and $(\alpha \mathsf{U} \emptyset)$ always represent the same language; so do $((\alpha \mathsf{U} \beta) \mathsf{U} \gamma)$ and $(\alpha \mathsf{U}(\beta \mathsf{U} \gamma))$. Since set union

and concatenation are associative operations —that is, since $(L_1 \cup L_2) \cup L_3 = L_1 \cup (L_2 \cup L_3)$ for all L_1, L_2, L_3, and the same for concatenation— we normally omit the extra (and) symbols in regular expressions; for example, we treat $a\mathsf{U}b\mathsf{U}c$ as a regular expression even though "officially" it is not. For another example, the regular expression of Example 1.8.4 might be rewritten as $0^*\mathsf{U}0^*(1\mathsf{U}11)(00^*(1\mathsf{U}11))^*0^*$.

Moreover, now that we have shown that regular expressions and the languages they represent can be defined formally and unambiguously, we feel free, when no confusion can result, to blur the distinction between the regular expressions and the "mathematical English" we are using for talking about languages. Thus we may say at one point that a^*b^* is the set of all strings consisting of some number of a's followed by some number of b's —to be precise, we should have written $\{a\}^* \circ \{b\}^*$. At another point, we might say that a^*b^* is a regular expression representing that set; in this case, to be precise, we should have written (a^*b^*).

The class of **regular languages** over an alphabet Σ is defined to consist of all languages L such that $L = \mathcal{L}(\alpha)$ for some regular expression α over Σ. That is, regular languages are all languages that can be described by regular expressions. Alternatively, regular languages can be thought of in terms of *closures*. The class of regular languages over Σ is precisely the closure of the set of languages

$$\{\{\sigma\} : \sigma \in \Sigma\} \cup \{\emptyset\}$$

with respect to the functions of union, concatenation, and Kleene star.

We have already seen that regular expressions do describe some nontrivial and interesting languages. Unfortunately, we cannot describe by regular expressions some languages that have very simple descriptions by other means. For example, $\{0^n1^n : n \geq 0\}$ will be shown in Chapter 2 *not* to be regular. Surely any theory of the finite representation of languages will have to accommodate at least such simple languages as this. Thus regular expressions are an inadequate specification method in general.

In search of a general method for finitely specifying languages, we might return to our general scheme

$$L = \{w \in \Sigma^* : w \text{ has property } P\}.$$

But which properties P should we entail? For example, what makes the preceding properties, "w consists of a number of 0's followed by an equal number of 1's" and "w has no occurrence of 111" such obvious candidates? The reader may ponder about the right answer; but let us for now allow *algorithmic properties*, and only these. That is, for a property P of strings to be admissible as a specification of a language, there must be an algorithm for deciding whether a given string belongs to the language. An algorithm that is specifically designed,

for some language L, to answer questions of the form "Is string w a member of L?" will be called a **language recognition device**. For example, a device for recognizing the language

$$L = \{w \in \{0,1\}^* : w \text{ does not have } 111 \text{ as a substring}\}.$$

by reading strings, a symbol at a time, from left to right, might operate like this:

Keep a count, which starts at zero and is set back to zero every time a 0 is encountered in the input; add one every time a 1 is encountered in the input; stop with a No answer if the count ever reaches three, and stop with a Yes answer if the whole string is read without the count reaching three.

An alternative and somewhat orthogonal method for specifying a language is to describe how a generic specimen in the language is produced. For example, a regular expression such as $(e \cup b \cup bb)(a \cup ab \cup abb)^*$ may be viewed as a way of *generating* members of a language:

To produce a member of L, first write down either nothing, or b, or bb; then write down a or ab, or abb, and do this any number of times, including zero; all and only members of L can be produced in this way.

Such **language generators** are not algorithms, since they are not designed to answer questions and are not completely explicit about what to do (how are we to choose which of a, ab, or abb is to be written down?) But they are important and useful means of representing languages all the same. The relation between language recognition devices and language generators, both of which are types of finite language specifications, is another major subject of this book.

Problems for Section 1.8

1.8.1. What language is represented by the regular expression $(((a^\star a)b)\mathsf{U}b)$?

1.8.2. Rewrite each of these regular expressions as a simpler expression representing the same set.

(a) $\oslash^\star \mathsf{U} a^\star \mathsf{U} b^\star \mathsf{U} (a \mathsf{U} b)^\star$

(b) $((a^\star b^\star)^\star (b^\star a^\star)^\star)^\star$

(c) $(a^\star b)^\star \mathsf{U} (b^\star a)^\star$

(d) $(a \mathsf{U} b)^\star a (a \mathsf{U} b)^\star$

1.8.3. Let $\Sigma = \{a, b\}$. Write regular expressions for the following sets:

(a) All strings in Σ^* with no more than three a's.

(b) All strings in Σ^* with a number of a's divisible by three.

(c) All strings in Σ^* with exactly one occurrence of the substring aaa.

1.8.4. Prove that if L is regular, then so is $L' = \{w : uw \in L \text{ for some string } u\}$. (Show how to construct a regular expression for L' from one for L.)

1.8.5. Which of the following are true? Explain.
(a) $baa \in a^*b^*a^*b^*$
(b) $b^*a^* \cap a^*b^* = a^* \cup b^*$
(c) $a^*b^* \cap b^*c^* = \emptyset$
(d) $abcd \in (a(cd)^*b)^*$

1.8.6. The **star height** $h(\alpha)$ of a regular expression α is defined by induction as follows.

$$\begin{aligned} h(\varnothing) &= 0 \\ h(a) &= 0 \text{ for each } a \in \Sigma \\ h(\alpha \mathsf{U} \beta) &= h(\alpha\beta) = \text{the maximum of } h(\alpha) \text{ and } h(\beta). \\ h(\alpha^\star) &= h(\alpha) + 1 \end{aligned}$$

For example, if $\alpha = (((ab)^\star \mathsf{U} b^\star)^\star \mathsf{U} a^\star)$, then $h(\alpha) = 2$. Find, in each case, a regular expression which represents the same language and has star height as small as possible.
(a) $((abc)^\star ab)^\star$
(b) $(a(ab^\star c)^\star)^\star$
(c) $(c(a^\star b)^\star)^\star$
(d) $(a^\star \mathsf{U} b^\star \mathsf{U} ab)^\star$
(e) $(abb^\star a)^\star$

1.8.7. A regular expression is in **disjunctive normal form** if it is of the form $(\alpha_1 \mathsf{U} \alpha_2 \mathsf{U} \cdots \mathsf{U} \alpha_n)$ for some $n \geq 1$, where none of the α_i's contains an occurrence of U. Show that every regular language is represented by one in disjunctive normal form.

REFERENCES

An excellent source on informal set theory is the book

- P. Halmos *Naive Set Theory,* Princeton, N.J.: D. Van Nostrand, 1960.

A splendid book on mathematical induction is

- G. Polya *Induction and Analogy in Mathematics,* Princeton, N.J.: Princeton University Press, 1954.

A number of examples of applications of the pigeonhole principle appear in the first chapter of

- C. L. Liu *Topics in Combinatorial Mathematics,* Buffalo, N.Y.: Mathematical Association of America, 1972.

Cantor's original diagonalization argument can be found in

- G. Cantor *Contributions to the Foundations of the Theory of Transfinite Numbers* New York: Dover Publications, 1947.

The O-notation and several variants were introduced in

- D. E. Knuth "Big omicron and big omega and big theta," *ACM SIGACT News, 8* (2), pp. 18–23, 1976.

The $O(n^3)$ algorithm for the reflexive-transitive closure is from

- S. Warshall "A theorem on Boolean matrices," *Journal of the ACM, 9,* 1, pp. 11-12, 1962.

Two books on algorithms and their analysis are

- T. H. Cormen, C. E. Leiserson, R. L. Rivest *Introduction to Algorithms,* Cambridge, Mass.: The MIT Press., 1990, and
- G. Brassard, P. Bratley *Fundamentals of Algorithms,* Englewood Cliffs, N.J.: Prentice Hall, 1996.

Two advanced books on language theory are

- A. Salomaa *Formal Languages* New York: Academic Press, 1973.
- M. A. Harrison *Introduction to Formal Language Theory,* Reading, Massach.: Addison-Wesley, 1978.