Steven Homer • Alan L. Selman

Computability and Complexity Theory

Second Edition

Springer

Preface to the First Edition 2001

The theory of computing provides computer science with concepts, models, and formalisms for reasoning about both the resources needed to carry out computations and the efficiency of the computations that use these resources. It provides tools to measure the difficulty of combinatorial problems both absolutely and in comparison with other problems. Courses in this subject help students gain analytic skills and enable them to recognize the limits of computation. For these reasons, a course in the theory of computing is usually required in the graduate computer science curriculum.

The harder question to address is which topics such a course should cover. We believe that students should learn the fundamental models of computation, the limitations of computation, and the distinctions between feasible and intractable. In particular, the phenomena of NP-completeness and NP-hardness have pervaded much of science and transformed computer science. One option is to survey a large number of theoretical subjects, typically focusing on automata and formal languages. However, these subjects are less important to theoretical computer science, and to computer science as a whole, now than in the past. Many students have taken such a course as part of their undergraduate education. We chose not to take that route because computability and complexity theory are the subjects that we feel deeply about and that we believe are important for students to learn. Furthermore, a graduate course should be scholarly. It is better to treat important topics thoroughly than to survey the field.

This textbook is intended for use in an introductory graduate course in theoretical computer science. It contains material that should be core knowledge in the theory of computation for all graduate students in computer science. It is self-contained and is best suited for a one-semester course. Most of the text can be covered in one semester by moving expeditiously through the core material of Chaps. 1 through 5 and then covering parts of Chap. 6. We will give more details about this below.

As a graduate course, students should have some prerequisite preparation. The ideal preparation would be the kind of course that we mentioned above: an undergraduate course that introduced topics such as automata theory, formal languages, computability theory, or complexity theory. We stress, however, that

there is nothing in such a course that a student needs to know before studying this text. Our personal experience suggests that we cannot presume that all of our students have taken such an undergraduate course. For those students who have not, we advise that they need at least some prior exposure that will have developed mathematical skills. Prior courses in mathematical logic, algebra (at the level of groups, rings, or fields), or number theory, for example, would all serve this purpose.

Despite the diverse backgrounds of our students, we have found that graduate students are capable of learning sophisticated material when it is explained clearly and precisely. That has been our goal in writing this book.

This book also is suitable for advanced undergraduate students who have satisfied the prerequisites. It is an appropriate first course in complexity theory for students who will continue to study and work in this subject area.

The text begins with a preliminary chapter that gives a brief description of several topics in mathematics. We included this in order to keep the book self-contained and to ensure that all students have a common notation. Some of these sections simply enable students to understand some of the important examples that arise later. For example, we include a section on number theory and algebra that includes all that is necessary for students to understand that primality belongs to NP.

The text starts properly with classical computability theory. We build complexity theory on top of that. Doing so has the pedagogical advantage that students learn a qualitative subject before advancing to a quantitative one. Also, the concepts build from one to the other. For example, although we give a complete proof that the satisfiability problem is NP-complete, it is easy for students to understand that the bounded halting problem is NP-complete, because they already know that the classical halting problem is c.e.-complete.

We use the terms *partial computable* and *computably enumerable (c.e.)* instead of the traditional terminology, *partial recursive* and *recursively enumerable (r.e.)*, respectively. We do so simply to eliminate confusion. Students of computer science know of "recursion" as a programming paradigm. We do not prove here that Turing-computable partial functions are equivalent to partial recursive functions, so by not using that notation, we avoid the matter altogether. Although the notation we are using has been commonplace in the computability theory and mathematical logic community for several years, instructors might want to advise their students that the older terminology seems commonplace within the theoretical computer science community. Computable functions are defined on the set of words over a finite alphabet, which we identify with the set of natural numbers in a straightforward manner. We use the term *effective*, in the nontechnical, intuitive sense, to denote computational processes on other data types. For example, we will say that a set of Turing machines is "effectively enumerable" if its set of indices is computably enumerable.

Chapter 4 concludes with a short list of topics that students should know from the chapters on computability theory before proceeding to study complexity theory. We advise instructors who wish to minimize coverage of computability theory to refer to this list. Typically, we do not cover the second section on the recursion theorem (Sect. 3.10) in a one-semester course. Although we do not recommend it,

it is possible to begin the study of complexity theory after learning the first five sections of Chap. 4 and at least part of Sect. 3.9 on oracle Turing machines, Turing reductions, and the arithmetical hierarchy.

In Chap. 5, we treat general properties of complexity classes and relationships between complexity classes. These include important older results such as the space and time hierarchy theorems, as well as the more recent result of Immerman and Szelepcsényi that space-bounded classes are closed under complements. Instructors might be anxious to get to NP-complete problems (Chap. 6) and NP-hard problems (Chap. 7), but students need to learn the basic results of complexity theory and it is instructive for them to understand the relationships between P, NP, and other deterministic and nondeterministic, low-level complexity classes. Students should learn that nondeterminism is not well understood in general, that P =? NP is not an isolated question, and that other classes have complete problems as well (which we take up in Chap. 7). Nevertheless, Chap. 5 is a long chapter. Many of the results in this chapter are proved by complicated Turing-machine simulations and counting arguments, which give students great insight, but can be time-consuming to cover. For this reason, instructors might be advised to survey some of this material if the alternative would mean not having sufficient time for the later chapters.

Homework exercises are an important part of this book. They are embedded in the text where they naturally arise, and students should not proceed without working on them. Many are simple exercises, whereas others are challenging. Often we leave important but easy-to-prove propositions as exercises. We provide additional problems at the end of chapters, which extend and apply the material covered there.

Once again, our intent has been to write a text that is suitable for all graduate students, that provides the right background for those who will continue to study complexity theory, and that can be taught in one semester. There are several important topics in complexity theory that cannot be treated properly in a one-semester course. Currently we are writing a second part to this text, which will be suitable for an optional second semester course, covering nonuniform complexity (Boolean circuits), parallelism, probabilistic classes, and interactive protocols.

Preface to the Second Edition 2011

At long last we have written this second part, and here it is. We corrected a myriad of typos, a few technical glitches, and some pedagogical issues in the first part.

Chapter 8, the first of the new chapters, is on nonuniformity. Here we study Boolean circuits, advice classes, define P/poly, and establish the important result of Karp–Lipton. Then we define and show basic properties of Schöning's low and high hierarchies. We need this for results that will come later in Chap. 10. Specifically, in that chapter we prove that the Graph Nonisomorphism Problem (GNI) is in the operator class $\mathrm{BP} \cdot \mathrm{NP}$ and that the Graph Isomorphism Problem (GI) is in the low hierarchy. Then it follows immediately that GI cannot be NP-complete unless the polynomial hierarchy collapses. Of course, primarily this chapter studies properties of the fundamental probabilistic complexity classes.

We study the alternating Turing machine and uniform circuit classes, especially NC, in Chap. 9. In the next chapter, we introduce counting classes and prove the famous results of Valiant and Vazirani and of Toda. The text ends with a thorough treatment of the proof that IP is identical to PSPACE, including worked-out examples. We include a section on Arthur-Merlin games and point out that $\mathrm{BP} \cdot \mathrm{NP} = \mathrm{AM}$, thereby establishing some of the results that classify this class.

We have found that the full text can be taught in one academic year. We hope that the expanded list of topics gives instructors flexibility in choosing topics to cover.

Thanks to everyone who sent us comments and corrections, and a special thanks to our students who demonstrated solvability of critical homework exercises.

Boston and Buffalo

Steven Homer
Alan Selman

Chapter 1
Preliminaries

We begin with a limited number of mathematical notions that a student should know before beginning with this text. This chapter is short because we assume some earlier study of data structures and discrete mathematics.

1.1 Words and Languages

In the next chapter we will become familiar with models of computing. The basic data type of our computers will be "symbols," for our computers manipulate symbols. The notion of symbol is undefined, but we define several more concepts in terms of symbols.

A finite set $\Sigma = \{a_1, \ldots, a_k\}$ of symbols is called a finite *alphabet*. A *word* is a finite sequence of symbols. The *length* of a word w, denoted $|w|$, is the number of symbols composing it. The *empty* word is the unique word of length 0 and is denoted as λ. Note that λ is *not* a symbol in the alphabet. The empty word is not a set, so do not confuse the empty word λ with the empty set $\emptyset$.

Σ^* denotes the set of all words over the alphabet Σ. A *language* is a set of words. That is, L is a language if and only if $L \subseteq \Sigma^*$. A *prefix* of a word is a substring that begins the word.

Example 1.1. Let $w = abcce$. The prefixes of w are

$$\lambda, a, ab, abc, abcc, abcce.$$

Define *suffixes* similarly.

The *concatenation* of two words x and y is the word xy. For any word w, $\lambda w = w\lambda = w$. If $x = uvw$, then v is a subword of x. If u and w are not both λ, then v is a proper subword.

S. Homer and A.L. Selman, *Computability and Complexity Theory*,
Texts in Computer Science, DOI 10.1007/978-1-4614-0682-2_1,

Some operations on languages:

union $L_1 \cup L_2$
intersection $L_1 \cap L_2$
complement $\overline{L} = \Sigma^* - L$
concatenation $L_1 L_2 = \{xy \mid x \in L_1 \text{ and } y \in L_2\}$.

The *powers* of a language L are defined as follows:

$$\begin{aligned} L^0 &= \{\lambda\}, \\ L^1 &= L, \\ L^{n+1} &= L^n L, \text{ for } n \geq 1. \end{aligned}$$

The *Kleene closure* of a language L is the language

$$L^* = \bigcup_{i=0}^{\infty} L^i.$$

Note that $\lambda \in L^*$, for all L. Applying this definition to $L = \Sigma$, we get, as we said above, that Σ^* is the set of all words. Note that $\emptyset^* = \{\lambda\}$.

Define $L^+ = \bigcup_{i=1}^{\infty} L^i$. Then, $\lambda \in L^+ \Leftrightarrow \lambda \in L$.

Theorem 1.1. *For any language S, $S^{**} = S^*$.*

Homework 1.1 *Prove Theorem 1.1.*

The *lexicographic* ordering of Σ^* is defined by $w < w'$ if $|w| < |w'|$ or if $|w| = |w'|$ and w comes before w' in ordinary dictionary ordering.

If A is a language and n is a positive integer, $A^{=n} = A \cap \Sigma^n$ denotes the set of words of length n that belongs to A.

1.2 *K*-adic Representation

Let N denote the set of all natural numbers, i.e., $N = \{0, 1, 2, 3, \ldots\}$. We need to represent the natural numbers as words over a finite alphabet. Normally we do this using binary or decimal notation, but k-adic notation, which we introduce here, has the advantage of providing a one-to-one and onto correspondence between Σ^* and N.

Let Σ be a finite alphabet with k symbols. Call the symbols $1, \ldots, k$. Every word over Σ will denote a unique natural number.

Let $x = \sigma_n \cdots \sigma_1 \sigma_0$ be a word in Σ^*. Define

$$\begin{aligned} N_k(\lambda) &= 0, \\ N_k(x) &= N_k(\sigma_n \cdots \sigma_1 \sigma_0) \\ &= \sigma_n * k^n + \cdots + \sigma_1 * k^1 + \sigma_0. \end{aligned}$$

$N_k(x)$ is the number that the word x represents.

Example 1.2. Let $\Sigma = \{1,2,3\}$. The string 233 denotes the integer

$$N_3(233) = 2*3^2+3*3^1+3*3^0 = 18+9+3 = 30.$$

Also,

$$\begin{aligned} N_k(\lambda) &= 0, \\ N_k(xa) &= k*N_k(x)+a \end{aligned}$$

is a recursive definition of N_k.

To see that N_k maps Σ^* onto the natural numbers, we need to show that every natural number has a k-adic representation. Given m, we want a word $s_n \ldots s_1 s_0$ such that $m = s_n * k^n + \cdots + s_1 * k^1 + s_0$. Note that $m = [s_n * k^{n-1} + \cdots + s_1] * k + s_0$. Let $a_0 = s_n * k^{n-1} + \cdots + s_1$. Then, $a_0 k = \max\{ak \mid ak < m\}$. Use this equation to find a_0. Then, $s_0 = m - a_0 k$. Iterate the process with a_0 until all values are known.

1.3 Partial Functions

Suppose that P is a program whose input values are natural numbers. It is possible that P does not halt on all possible input values. Suppose that P is designed to compute exactly one output value, again a natural number, for each input value on which it eventually halts. Then P computes a *partial function* on the natural numbers. This is the fundamental data type that is studied in computability theory.

The partial function differs somewhat from the function of ordinary mathematics. If f is a partial function defined on N, then for some values of $x \in N$, $f(x)$ is well defined; i.e., there is a value $y \in N$ such that $y = f(x)$. For other values of $x \in N$, $f(x)$ is undefined; i.e., $f(x)$ does not exist. When $f(x)$ is defined, we say $f(x)$ *converges* and we write $f(x)\downarrow$. When $f(x)$ is undefined, we say $f(x)$ *diverges* and we write $f(x)\uparrow$.

Given a partial function f, we want to know whether, given values x, does $f(x)$ converge and if so, what is the value of $f(x)$? Can the values of f be computed (by a computer program), and if so can f be efficiently computed?

We will also be concerned with subsets of the natural numbers and with relations defined on the natural numbers.

Given a set A (i.e., $A \subseteq N$), we want to know, for values x, whether $x \in A$. Is there an algorithm that for all x, will determine whether $x \in A$? For relations, the question is essentially the same. Given a k-ary relation R and values $x_1, \ldots, x_k$, is $R(x_1, \ldots, x_k)$ true? Is there a computer program that for all input tuples will decide the question? If so, is there an efficient solution?

This discussion assumed that the underlying data type is the set of natural numbers, N. As we just learned, it is equivalent to taking the underlying data type to be Σ^*, where Σ is a finite alphabet. We will pun freely between these two points of view.

1.4 Graphs

A *graph* is a pair $G = (V, E)$ consisting of a finite, nonempty set V of vertices and a set E of *edges*. An *edge* is an unordered pair of distinct vertices. (For $v \in V$, (v, v) cannot be an edge because the vertices are not distinct.) If (u, v) is an edge, then u and v are vertices; we say that u and v are *adjacent*. A graph is *complete* if every pair of distinct vertices is connected by an edge.

A *subgraph* of $G = (V, E)$ is a graph $G' = (V', E')$ such that

1. $V' \subseteq V$, and
2. E' consists of edges (v, w) in E such that both v and w are in V'.

If E' consists of all edges (v, w) in E such that both v and w are in V', then G' is called an *induced subgraph* of G.

In most contexts (v, w) denotes an ordered pair, but when discussing graphs, we abuse notation by using (v, w) to denote edges, which are unordered pairs.

A *path* is a sequence of vertices connected by edges. The length of a path is the number of edges on the path. (A single vertex is a path of length 0.) A *simple path* is a path that does not repeat any vertex or edge, except possibly the first and last vertices. A *cycle* is a simple path of length at least 1 that starts and ends in the same vertex. Observe that the length of a cycle must be at least 3 because v and u, v, u are not cycles. A *Hamiltonian circuit* is a cycle that contains every vertex in the graph.

Example 1.3. The sequence $1, 2, 4, 3, 5, 1$ is a Hamiltonian circuit of the graph in Fig. 1.1.

A graph is *connected* if every two vertices has a path between them. The number of edges at a vertex is the *degree* of the vertex.

A *directed graph* (*digraph*) consists of a set of *vertices* and a set of *arcs*. An arc is an ordered pair. Figure 1.2 gives an example.

A *path* in a digraph is a sequence of vertices $v_1, \ldots, v_n$ such that for every i, $1 \leq i < n$, there is an arc from v_i to v_{i+1}. A digraph is *strongly connected* if there is a path from any vertex to any other vertex.

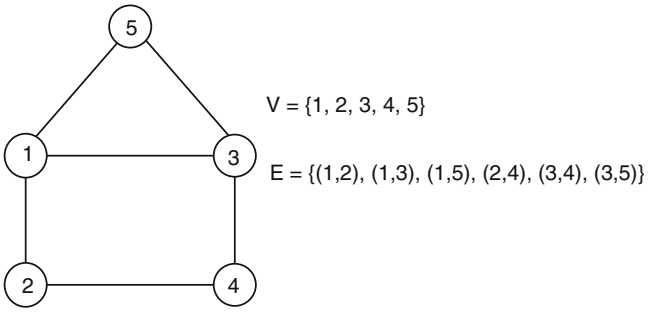

Fig. 1.1 A graph $G = (V, E)$

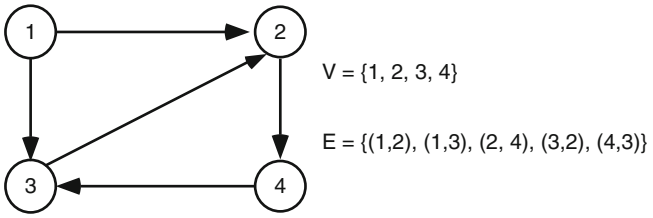

Fig. 1.2 A digraph $G = (V, E)$

An (undirected) *tree* is a connected graph with no cycles.

For a directed graph we define a cycle just as for an undirected graph. Note that there can be cycles of length 2 in directed graphs. For directed graphs we define a tree as follows:

1. There is exactly one vertex, called the root, that no arcs enter;
2. Every other vertex is entered by exactly one arc; and
3. There is a path from the root to every vertex.

If (u, v) is an arc in a tree, then u is a *parent* of v, and v is a *child* of u. If there is a path from u to v, then u is an *ancestor* of v, and v is a *descendant* of u. A vertex with no children is called a *leaf*. A *vertex u* together with all its descendants is a *subtree*, and u is the root of that subtree.

The *depth* of a vertex u in a tree is the length of the path from the root to u. The *height* of u is the length of a longest path from u to a leaf. The *height of the tree* is the height of the root. Finally, when the children of each vertex are ordered, we call this an *ordered tree*. A *binary tree* is a tree such that each child of a vertex is either a *left* child or a *right* child, and no vertex has more than one left child or right child.

1.5 Propositional Logic

Propositional logic provides a mathematical formalism that is useful for representing and manipulating statements of fact that are either true or false.

Let $U = \{u_1, u_2, u_3, \ldots\}$ be a set of Boolean variables (i.e., ranging over $\{0, 1\}$, where we identify 0 with False and 1 with True). We associate the binary Boolean connectives $\wedge$ and $\vee$, and the unary connective $\neg$, with AND, inclusive-OR, and NOT, respectively. However, their exact semantic meaning will be given shortly. For now, they are purely syntactic.

The class of *propositional formulas* is defined inductively as follows:

1. Every propositional variable is a propositional formula;
2. If A and B are propositional formulas, then the expressions $(A \wedge B)$, $(A \vee B)$, and $(\neg A)$ are propositional formulas.

When convenient, we will eliminate parentheses in propositional formulas in accordance with the usual precedence rules.

Definition 1.1. Let F be a propositional formula and let $\mathrm{VAR}(F)$ be the set of variables that occur in F. An *assignment* (or *truth-assignment*) t is a function

$$t : \mathrm{VAR}(F) \to \{0,1\}.$$

An assignment induces a truth-value to the formula F by induction, as follows:

1.
$$t((A \wedge B)) = \begin{cases} 1 \text{ if } t(A) = t(B) = 1; \\ 0 \text{ otherwise.} \end{cases}$$

2.
$$t((A \vee B)) = \begin{cases} 0 \text{ if } t(A) = t(B) = 0; \\ 1 \text{ otherwise.} \end{cases}$$

3.
$$t((\neg A)) = \begin{cases} 1 \text{ if } t(A) = 0; \\ 0 \text{ otherwise.} \end{cases}$$

Using these rules, given any formula F and an assignment t to $\mathrm{VAR}(F)$, we can evaluate $t(F)$ to determine whether the assignment makes the formula True or False. Also, these rules ascribe meaning to the connectives. It is common to present the truth-values of a formula F under all possible assignments as a finite table, called a *truth-table*.

If $u \in U$ is a Boolean variable, it is common to write $\overline{u}$ in place of $(\neg u)$. Variables u and negated variables $\overline{u}$ are called *literals*.

Example 1.4. The propositional formula $(u_1 \vee \overline{u_2}) \wedge (\overline{u_1} \vee u_2)$ has the following truth-table:

u_1	u_2	$(u_1 \vee \overline{u_2}) \wedge (\overline{u_1} \vee u_2)$
1	1	1
1	0	0
0	1	0
0	0	1

Definition 1.2. An assignment t *satisfies* a formula F if $t(F) = 1$. A formula F is *satisfiable* if there exists an assignment to its variables that satisfies it.

We will learn in Chap. 6 that the satisfiable formulas play an exceedingly important role in the study of complexity theory.

Definition 1.3. A formula is *valid* (or is a *tautology*) if every assignment to its variables satisfies it.

Proposition 1.1. *A formula F is a tautology if and only if $(\neg F)$ is not satisfiable.*

Homework 1.2 *Prove Proposition 1.1.*

Definition 1.4. Two formulas F and G are *equivalent* if for every assignment t to $\mathrm{VAR}(F) \cup \mathrm{VAR}(G)$, $t(F) = t(G)$.

Next we define two special syntactic "normal" forms of propositional formulas. We will show that every formula is equivalent to one in each of these forms.

A formula is a *conjunction* if it is of the form $(A_1 \wedge A_2 \wedge \cdots A_n)$, where each A_i is a formula, and we often abbreviate this using the notation $\bigwedge_{1 \le i \le n} A_i$. Similarly, a *disjunction* is a formula of the form $(A_1 \vee A_2 \vee \cdots A_n)$, which we can write as $\bigvee_{1 \le i \le n} A_i$.

A *clause* is a disjunction of literals. (For example, $u_1 \vee \overline{u_3} \vee u_8$ is a clause.) Observe that a clause is satisfied by an assignment if and only if the assignment makes at least one of its literals true.

Definition 1.5. A propositional formula G is in *conjunctive normal form* if it is a conjunction of clauses.

Example 1.5. $(u_1 \vee \overline{u_2}) \wedge (\overline{u_1} \vee u_2)$ is in conjunctive normal form, the assignment $t(u_1) = t(u_2) = 1$ satisfies the formula, but it is not a tautology.

Example 1.6. $(u_1 \wedge \overline{u_1})$ is in conjunctive normal form and has no satisfying assignment.

Homework 1.3 *Show that every formula is equivalent to one in conjunctive normal form. (You will need to use elementary laws of propositional logic such as DeMorgan's laws, which state that* $\neg(A \wedge B)$ *is equivalent to* $(\neg A \vee \neg B)$ *and that* $\neg(A \vee B)$ *is equivalent to* $(\neg A \wedge \neg B)$.*)*

Since a formula in conjunctive normal form is a conjunction of clauses, it is a conjunction of a disjunction of literals. Analogously, we define a formula to be in *disjunctive normal form* if it is a disjunction of a conjunction of literals.

Example 1.7. $(u_1 \wedge \overline{u_2}) \vee (\overline{u_1} \wedge u_2)$ is in disjunctive normal form.

Using the technique of Homework 1.3, every propositional formula is equivalent to one in disjunctive normal form.

1.5.1 Boolean Functions

A *Boolean function* is a function $f : \{0,1\}^n \to \{0,1\}$, where $n \ge 1$. A truth-table is just a tabular presentation of a Boolean function, so every propositional formula defines a Boolean function by its truth-table.

Conversely, let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function. Then, we can represent f by the following formula F_f in disjunctive normal form whose truth-table is f. For each n-tuple $(a_1, \ldots, a_n) \in \{0,1\}^n$ such that $f(a_1, \ldots, a_n) = 1$, write the conjunction of literals $(l_1 \wedge \cdots \wedge l_n)$, where $l_i = u_i$ if $a_i = 1$ and $l_i = \overline{u_i}$ if $a_i = 0$ (where $1 \le i \le n$, and $u_1, \ldots, u_n$ are Boolean variables). Then, define F_f to be the disjunction of each such conjunction of literals.

1.6 Cardinality

The *cardinality* of a set is a measure of its size. Two sets A and B have the *same cardinality* if there is a bijection $h : A \to B$. In this case we write $\text{card}(A) = \text{card}(B)$. If there exists a one-to-one function h from A to B, then $\text{card}(A) \leq \text{card}(B)$. For finite sets $A = \{a_1, \ldots, a_k\}$, $k \geq 1$, $\text{card}(A) = k$. A set A is *countable* if $\text{card}(A) = \text{card}(N)$ or A is finite. A set A is *countably infinite* if $\text{card}(A) = \text{card}(N)$.

A set is *enumerable* if it is the empty set or there is a function $f : N \to_{\text{onto}} A$. In this case A can be written as a sequence: Writing a_i for $f(i)$, we have

$$A = range(f) = \{a_0, a_1, a_2, \ldots\} = \{a_i \mid i \geq 0\}.$$

To call a set enumerable is to say that its elements can be counted. Observe that an enumeration need not be one-to-one. Since $a_i = a_j$, for $i \neq j$, is possible, it is possible that some elements of A are counted more than once.

Theorem 1.2. *A set is enumerable if and only if it is countable.*

Homework 1.4 *Prove Theorem 1.2.*

The cardinality of N is denoted $\aleph_0$.

Theorem 1.3. *A set A is countable if and only if* $\text{card}(A) \leq \aleph_0$.

That is, $\aleph_0$ is the smallest nonfinite cardinality. (Of course, at the moment we have no reason to expect that there is any other nonfinite cardinality.)

Proof. Suppose $\text{card}(A) \leq \aleph_0$. Then there is a one-to-one function f from A to N. Suppose $f[A]$ has a largest element k. Then A is a finite set. Suppose $f[A]$ does not have a largest member. Let a_0 be the unique member of A such that $f(a_0)$ is the smallest member of $f[A]$. Let a_{n+1} be the unique member of A such that $f(a_{n+1})$ is the smallest member of $f[A] - \{f(a_0), \ldots, f(a_n)\}$. It follows that A is enumerable.

The reverse direction is straightforward. □

Homework 1.5 *If* $\text{card}(A) \leq \text{card}(B)$ *and* $\text{card}(B) \leq \text{card}(C)$, *then* $\text{card}(A) \leq \text{card}(C)$.

Homework 1.6 *(This is a hard problem, known as the Cantor–Bernstein Theorem.) If* $\text{card}(A) \leq \text{card}(B)$ *and* $\text{card}(B) \leq \text{card}(A)$, *then* $\text{card}(A) = \text{card}(B)$.

Example 1.8. $\{< x, y > \mid x, y \in N\}$ is countable. An enumeration is

$$< 0,0 >, < 0,1 >, < 1,0 >, < 0,2 >, \ldots.$$

Example 1.9. The set of rational numbers is countable.

Example 1.10. The set of programs in any programming language is countably infinite. For each language there is a finite alphabet Σ such that each program is in Σ^*. Because programs may be arbitrarily long, there are infinitely many of them. There are $\aleph_0$ many programs.

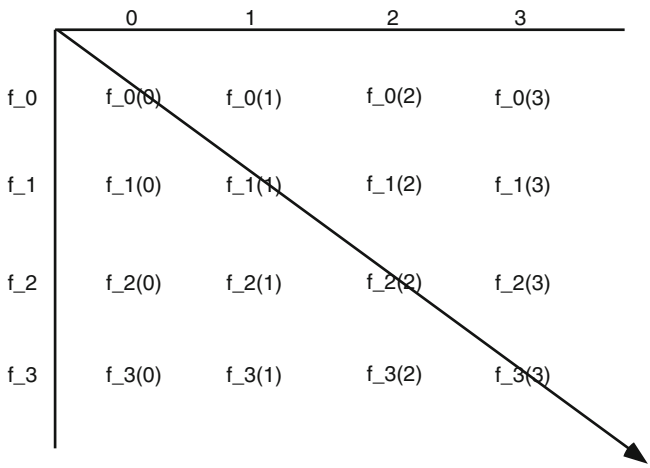

Fig. 1.3 The diagonalization technique

The proof of the following theorem employs the technique of *diagonalization*. Diagonalization was invented by the mathematician George Cantor (1845–1918), who created the theory of sets that we now take for granted. This is an important technique in theoretical computer science, so it would be wise to master this easy application first.

Theorem 1.4. *The set of all functions from N to N is not countable.*

Proof. Let $A = \{f \mid f : N \to N\}$. Suppose A is countable. Then there is an enumeration $f_0, f_1, \ldots$ of A. (Think of all the values of each f_i laid out on an infinite matrix: The idea is to define a function that cannot be on this matrix because it differs from all of the values on the diagonal. This is illustrated in Fig. 1.3.) Define a function g by $g(x) = f_x(x) + 1$, for all $x \in N$. Then, g is a function on N, but observe that g cannot be in the enumeration of all functions. That is, if $g \in A$, then for some natural number k, $g = f_k$. But g cannot equal f_k because $g(k) \neq f_k(k)$. Thus, we have contradicted the assumption that the set of all functions can be enumerated. Thus, A is not countable. □

Consider your favorite programming language. As there are countably many programs but there are uncountably many functions defined on N, there are functions that your favorite programming language cannot compute. All reasonable general-purpose programming systems compute the exact same set of functions, so it follows that there are functions defined on N that are not computable by any program in any programming system.

For any set A, $\mathscr{P}(A) = \{S \mid S \subseteq A\}$ denotes the power set of A.

Theorem 1.5. $\mathscr{P}(N)$ *has cardinality greater than* $\aleph_0$.

Proof. Let $A = \mathscr{P}(N)$. Clearly, A is infinite. Suppose A can be enumerated, and let $S_0, S_1, \ldots$ be an enumeration of A. Then, define $T = \{k \mid k \notin S_k\}$. By definition, T belongs to A. However, for every k, $T \neq S_k$ because $k \in T \Leftrightarrow k \notin S_k$. Thus, T is a set that is not in the enumeration. So we have a contradiction. Thus, A cannot be enumerated. □

1.6.1 Ordered Sets

It is useful to consider relations on the elements of a set that reflect the intuitive notion of ordering these elements.

Definition 1.6. A binary relation ρ on a set X is a *partial order* if it is:

1. *Reflexive* ($a\rho a$, for all $a \in X$),
2. *Antisymmetric* ($a\rho b$ and $b\rho a$ implies $a = b$, for all a and b in X), and
3. *Transitive* ($a\rho b$ and $b\rho c$ implies $a\rho c$, for all a, b, and c in X).

A partial order is a *linear order* on X if, in addition, for all a and b in X, $a\rho b$ or $b\rho a$. A *partially ordered set* (*linearly ordered set*) is a pair $\langle X, \rho \rangle$, where ρ is a partial order (linear order, respectively) on X. Let Z denote the set of integers.

Example 1.11. 1. $\langle Z, \leq \rangle$ and $\langle Z, \geq \rangle$ are linearly ordered sets, where $\leq$ and $\geq$ denote the customary well-known orderings on Z.
2. For any set A, $\langle \mathscr{P}(A), \subseteq \rangle$ is a partially ordered set.
3. $\langle Z, \{(a,b) \mid a,b \in Z \text{ and } a \text{ is an integral multiple of } b\} \rangle$ is a partially ordered set.
4. Let $\mathscr{C}$ be any collection of sets; then

$$\langle \{card(X) \mid X \in \mathscr{C}\}, \leq \rangle$$

is a linear order.

1.7 Elementary Algebra

Here we present some useful algebra and number theory. In the next several pages we will barely scratch the surface of these exceedingly rich subjects. This material is not needed for the main body of the course, but is useful for understanding several examples in later chapters.

1.7.1 Rings and Fields

Definition 1.7. A *ring* is a system $\langle R, +, \cdot, 0 \rangle$ that satisfies the following axioms, where R is a nonempty set, 0 is an element of R, and $+$ and $\cdot$ are operations on R. For arbitrary members a, b, and c of R:

(A1) $a+b=b+a$ (commutative law of addition);
(A2) $(a+b)+c=a+(b+c)$ (associative law of addition);
(A3) $a+0=a$ (zero element);
(A4) for every $a \in R$, there exists $x \in R$ such that $a+x=0$ (existence of additive inverses);
(A5) $(ab)c=a(bc)$[1] (associative law of multiplication);
(A6) $a(b+c)=ab+ac$;
(A7) $(b+c)a=ba+bc$ (distributive laws).

A ring is *commutative* if in addition it satisfies the following axiom:

(A8) $ab=ba$ (commutative law of multiplication).

A ring is *ring with unity* if there is an element 1 belonging to R such that

(A9) $1a=a1=a$.

Definition 1.8. A *field* is a commutative ring with unity $\langle R,+,\cdot,0,1\rangle$ such that

(A10) for every $a \neq 0$, $a \in R$, there exists $x \in R$ such that $ax=1$ (existence of multiplicative inverses).

Note that 0 and 1 do not denote numbers – they are elements of R that obey the appropriate axioms.

Remember that Z denotes the set of integers and let Q denote the set of all rational numbers. Then, using ordinary integer addition and multiplication and integers 0 and 1, Z forms a ring but not a field. The rational numbers Q, with its ordinary operations, forms a field.

Theorem 1.6. *Each nonzero element in a field has a unique multiplicative inverse.*

Proof. Suppose that s and t are two multiplicative inverses of a. Then

$$s=s1=s(at)=(sa)t=(as)t=1t=t.$$

□

The unique multiplicative inverse of an element a in a field is denoted a^{-1}.

Definition 1.9. Let m be an integer greater than 1 and let a and b be integers. Then a is *congruent to b modulo m* if m divides $a-b$.

We indicate that m divides $a-b$ by writing $m|a-b$, and we write $a \equiv b \pmod{m}$ to denote that a is congruent to b modulo m. Let $\mathrm{rm}(a,m)$ denote the remainder when dividing a by m (eg., $\mathrm{rm}(5,2)=1$).

Theorem 1.7. *The following are elementary facts about congruence modulo m.*

1. $a \equiv b \pmod{m}$ *if and only if* $\mathrm{rm}(a,m)=\mathrm{rm}(b,m)$.

[1] As is customary, we write ab instead of writing $a \cdot b$.

2.

$$a \equiv b \pmod{m} \Rightarrow \textit{for all integers } x,$$
$$a + x \equiv b + x \pmod{m},$$
$$ax \equiv bx \pmod{m}, \textit{ and}$$
$$-a \equiv -b \pmod{m}.$$

3. Congruence modulo m is an equivalence relation.

Homework 1.7 *Prove Theorem 1.7.*

Given $m > 1$, the equivalence class containing the integer a is the set

$$[a] = \{x \mid x \equiv a \pmod{m}\}.$$

We call $[a]$ an *equivalence class modulo m*. Let Z_m denote the set of equivalence classes modulo m, and let $r = \text{rm}(a,m)$, so for some integer q, $a = qm + r$ and $0 \leq r < m$. Then, $a - r = qm$, so $a \equiv r \pmod{m}$ and hence $[a] = [r]$. That is, every integer is congruent modulo m to one of the m integers $0, 1, \ldots, m-1$. Finally, no two of these are congruent modulo m. Thus, there are exactly m equivalence classes modulo m, and they are the sets $[0], [1], \ldots, [m-1]$. We have learned that $Z_m = \{[0], [1], \ldots, [m-1]\}$.

Now we will show that Z_m forms a useful number system. We define operations on Z_m as follows: $[a] + [b] = [a+b]$ and $[a][b] = [ab]$. The definition is well founded, i.e., independent of choice of representative member of each equivalence class, because

$$\text{rm}(a+b,m) \equiv \text{rm}(a,m) + \text{rm}(b,m) \pmod{m}$$

and

$$\text{rm}(ab,m) \equiv \text{rm}(a,m) \cdot \text{rm}(b,m) \pmod{m}.$$

We state the following theorem.

Theorem 1.8. *For each positive integer m, $\langle Z_m, +, \cdot, [0], [1] \rangle$ is a ring with unity.*

Homework 1.8 *Prove the theorem by verifying each of the properties A1 to A8.*

Definition 1.10. A commutative ring with unity $\langle R, +, \cdot, 0, 1 \rangle$ is an *integral domain* if for all $a, b \in R$,

(A11) $ab = 0$ implies $a = 0$ or $b = 0$ (absence of nontrivial zero divisors).

The integers with their usual operations form an integral domain.

Theorem 1.9. *Every field is an integral domain.*

Proof. Suppose that $ab = 0$. We show that if $a \neq 0$, then $b = 0$, so suppose that $a \neq 0$. Then the following holds:

$$b = 1b = \left(a^{-1}a\right)b = a^{-1}(ab) = a^{-1}0 = 0.$$

This completes the proof. □

Our goal is to show that if p is a prime number, then Z_p is a field. First, though, let us observe that if m is not a prime number, then Z_m is not even an integral domain: Namely, there exist positive integers a and b, $0 < a < b < m$, such that $m = ab$. Hence, in Z_m, $[a][b] = 0$; yet $[a] \neq 0$ and $[b] \neq 0$.

To show that Z_p is a field when p is prime, we need to show that the equation

$$ax \equiv 1 \pmod{p}$$

is solvable[2] for each integer $a \in \{1, \ldots, p-1\}$. For this purpose, we introduce the following notation.

Definition 1.11. For nonzero integers $a, b \in Z$, define

$$(a,b) = \{ax + by \mid x, y \in Z\}$$

to be the set of all *linear combinations* of a and b, and define

$$(a) = \{ax \mid x \in Z\}.$$

Definition 1.12. For nonzero integers $a, b \in Z$, the positive integer d is a *greatest common divisor* of a and b if

(i) d is a divisor of a and b, and
(ii) every divisor of both a and b is a divisor of d.

We write $d = \gcd(a,b)$.

Lemma 1.1. *$(a,b) = (d)$, where $d = \gcd(a,b)$.*

Proof. Since a and b are nonzero integers, there is a positive integer in (a,b). Let d be the least positive integer in (a,b). Clearly, $(d) \subseteq (a,b)$.

We show $(a,b) \subseteq (d)$: Suppose $c \in (a,b)$. Then, there exist integers q and r such that $c = qd + r$, with $0 \leq r < d$. Since c and d are in (a,b), it follows that $r = c - qd$ is in (a,b) also. However, since $0 \leq r < d$ and d is the least positive integer in (a,b), it must be the case that $r = 0$. Hence, $c = qd$, which belongs to (d). Thus, $(d) = (a,b)$.

All that remains is to show that $d = \gcd(a,b)$. Since a and b belong to (d), d is a common divisor of (a) and (b). If c is any other common divisor of (a) and (b), then c divides every number of the form $ax + by$. Thus, $c|d$, which proves that $d = \gcd(a,b)$. □

Definition 1.13. Two integers a and b are *relatively prime* if $\gcd(a,b) = 1$.

Theorem 1.10. *If p is a prime number, then $\langle Z_p, +, \cdot, [0], [1] \rangle$ is a field.*

[2] That is, we show for each integer $a \in \{1, \ldots, p-1\}$ that there exists an integer x such that $ax \equiv 1 \pmod{p}$.

Proof. Let $[a]$ be a nonzero member of Z_p. Then, $[a] \neq [0]$, so $a \not\equiv 0 \pmod{p}$. That is, p does not divide a. Thus, since p is prime, a and p are relatively prime. Now, let us apply Lemma 1.1: There exist integers x and y such that $1 = ax + py$. We can rewrite this as $1 - ax = py$ to see that $ax \equiv 1 \pmod{p}$. Hence, $[a][x] = [1]$, which is what we wanted to prove. □

Lemma 1.1 proves that the greatest common divisor of two integers always exists, but does not give a method of finding it. Next we present the *Euclidean Algorithm*, which computes $\gcd(x,y)$ for integers x and y. Later in the course we will analyze this algorithm to show that it is efficient.

If $d = \gcd(x,y)$, then d is the greatest common divisor of $-x$ and y, of x and $-y$, and of $-x$ and $-y$, as well. Thus, in the following algorithm we assume that x and y are positive integers.

EUCLIDEAN ALGORITHM
input positive integers x and y in binary notation;
repeat
    $x := \text{rm}(x,y)$;
    exchange x and y
until $y = 0$;
output x.

Let us understand the algorithm and see that it is correct. Let $r_1 = \text{rm}(x,y)$, so for some quotient q_1, $x = q_1 y + r_1$. Since $r_1 = x - q_1 y$, every number that divides x and y also divides r_1. Thus, d divides r_1, where $d = gcd(x,y)$. Now we will show that $\gcd(x,y) = \gcd(y,r_1)$. We know already that d is a common divisor of y and r_1. If there were a common divisor $d_1 > d$ that divides y and r_1, then this value d_1 would also divide x. Thus, d would not be the greatest common divisor of x and y.

The Euclidean Algorithm reduces the problem of finding $\gcd(x,y)$ to that of finding $\gcd(y,r_1)$, where $r_1 < y$. Since the remainder is always nonnegative and keeps getting smaller, it must eventually be zero. Suppose this occurs after n iterations. Then we have the following system of equations:

$$\begin{aligned}
x &= q_1 y + r_1, 0 < r_1 < y,\\
y &= q_2 r_1 + r_2, 0 < r_2 < r_1,\\
r_1 &= q_3 r_2 + r_3, 0 < r_3 < r_2,\\
r_2 &= q_4 r_3 + r_4, 0 < r_4 < r_3,\\
&\vdots\\
r_{n-2} &= q_n r_{n-1}.
\end{aligned}$$

Finally,

$$d = \gcd(x,y) = \gcd(y,r_1) = \gcd(r_1,r_2) = \cdots = \gcd(r_{n-2},r_{n-1}) = r_{n-1},$$

and r_{n-1} is the final value of x, which completes the argument that the Euclidean Algorithm computes $d = \gcd(x, y)$.

1.7.2 Groups

Definition 1.14. A *group* is a system $\langle G, \cdot, 1\rangle$ that satisfies the following axioms, where G is a nonempty set, 1 is an element of G, and $\cdot$ is an operation on G:

(A5) $(ab)c = a(bc)$
(A9) $1a = a1 = a$; and
(A12) for every $a \in G$ there exists $x \in G$ such that $ax = 1$.

A group is *commutative* if in addition it satisfies axiom (A8), the commutative law of multiplication.

The set of integers Z forms a commutative group $\langle Z, +, 0\rangle$ known as the *additive group of the integers*; for every positive integer m, $\langle Z_m, +, [0]\rangle$ is a commutative group. It follows from Theorem 1.10 that if p is a prime number, then $\langle Z_p - \{[0]\}, \cdot, [1]\rangle$ is a commutative group. More generally, for every field $\langle F, +, \cdot, 0, 1, \rangle$, the nonzero elements of F form a commutative group $\langle F - \{0\}, \cdot, 1\rangle$ known as the *multiplicative group of the field.*

Definition 1.15. The *order* of a group $\langle G, \cdot, 1\rangle$, written $o(G)$, is the number of elements in G if G is finite, and is infinite otherwise.

The *order* of an element a in G, $o(a)$, is the least positive m such that $a^m = 1$. If no such integer exists, then $o(a)$ is infinite.

The order of the additive group of the integers is infinite. The order of the additive group $\langle Z_m, +, [0]\rangle$ is m, while, for p a prime, the order of the multiplicative group of the nonzero elements of Z_p is $p - 1$.

Definition 1.16. Let H be a nonempty subset of G. H is a *subgroup* of G (or more precisely $\langle H, \cdot, 1\rangle$ is a *subgroup* of $\langle G, \cdot, 1\rangle$) if H contains the identity element 1 and $\langle H, \cdot, 1\rangle$ is a group.

Let $\langle G, \cdot, 1\rangle$ be a group and $a \in G$. Then the set $H = \{a^i \mid i \in Z\}$ is a subgroup of G. We claim that H contains $o(a)$ many elements. Of course, if H is infinite, then $o(a)$ is infinite. Suppose that H is finite, and let $o(a) = m$. Let $a^k \in H$. Then for some integer q and $0 \le r < m$, $a^k = a^{qm+r} = (a^m)^q a^r = 1a^r = a^r$. Thus, $H = \{1, a, \ldots, a^{m-1}\}$, so H contains at most $o(a)$ elements. If $o(H) < o(a)$, then for some i and j, $0 \le i < j < o(a)$, $a^i = a^j$. Hence, $a^{j-i} = 1$. However, $j - i < o(a)$, which is a contradiction. Thus, $o(H) = o(a)$.

Definition 1.17. If G contains an element a such that $G = \{a^i \mid i \in Z\}$, then G is a *cyclic* group and a is a *generator*.

For any group $\langle G, \cdot, 1\rangle$ and $a \in G$, $H = \{a^i \mid i \in Z\}$ is a cyclic subgroup of G and a is a generator of the subgroup.

1.7.2.1 Cosets

Now we come to a remarkable point: Every subgroup H of a group G partitions G into disjoint cosets.

Definition 1.18. Given a subgroup H of a group G and element $a \in G$, define $aH = \{ah \mid h \in H\}$. The set aH is called a *coset* of H.

The following lemma lists the basic properties of cosets.

Lemma 1.2. *Let a and b be members of a group G and let H be a subgroup of G.*

1. *$aH \cap bH \neq \emptyset$ implies $aH = bH$.*
2. *For finite subgroups H, aH contains $o(H)$ many elements.*

Proof. Suppose that aH and bH have an element $c = ah' = bh''$ $(h', h'' \in H)$ in common. Let $h \in H$. Then, $bh = bh''h''^{-1}h = a(h'h''^{-1}h)$, which belongs to aH. Thus, $aH \subseteq bH$. Similarly, bH contains every element of aH, and so $aH = bH$.

To see that aH has $o(H)$ many elements, we note that the mapping $h \mapsto ah$ (from H to aH) is one-to-one: Each element $x = ah$, $h \in H$, in the coset aH is the image of the unique element $h = a^{-1}x$. □

The element $a = a1 \in aH$. Thus, every element of G belongs to some coset, and because distinct cosets are disjoint, every element of G belongs to a unique coset. The cosets of H partition G. Thus, the proof of the next theorem follows immediately.

Theorem 1.11 (Lagrange). *Let H be a subgroup of a finite group G. Then, $o(H)|o(G)$.*

Lagrange's theorem has several important corollaries.

Corollary 1.1. *If G is a finite group and $a \in G$, then $a^{o(G)} = 1$.*

Proof. For some nonzero integer n, $o(G) = n \cdot o(a)$, so

$$a^{o(G)} = a^{n \cdot o(a)} = (a^{o(a)})^n = 1^n = 1.$$ □

Corollary 1.2. *Every group of order p, where p is prime, is a cyclic group.*

As a consequence, for each prime p, the additive group $\langle Z_p, +, [0]\rangle$ is a cyclic group.

We apply Corollary 1.1 to the multiplicative group $\langle Z_p - \{[0]\}, \cdot, [1]\rangle$ to obtain the following corollary.

Corollary 1.3 (Fermat). *If a is an integer, p is prime, and p does not divide a, then* $a^{p-1} \equiv 1 \pmod{p}$.

1.7.3 Number Theory

Our goal is to show that the multiplicative group of the nonzero elements of a finite field is a cyclic group. We know from Corollary 1.3 that for each prime number p and integer a, $1 \leq a \leq p-1$, $a^{p-1} \equiv 1 \pmod{p}$. However, we do not yet know whether there is a generator g, $1 \leq g \leq p-1$, such that $p-1$ is the *least* power m such that $g^m \equiv 1 \pmod{p}$. This is the result that will conclude this section. We begin with the following result, known as the *Chinese Remainder Theorem*.

Theorem 1.12 (Chinese Remainder Theorem). *Let $m_1, \ldots, m_k$ be pairwise relatively prime positive integers; that is, for all i and j, $1 \leq i, j \leq k$, $i \neq j$, $\gcd(m_i, m_j) = 1$. Let $a_1, \ldots, a_k$ be arbitrary integers. Then there is an integer x that satisfies the following system of simultaneous congruences:*

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\vdots \\ x &\equiv a_k \pmod{m_k}. \end{aligned}$$

Furthermore, there is a unique solution in the sense that any two solutions are congruent to one another modulo the value $M = m_1 m_2 \cdots m_k$.

Proof. For every i, $1 \leq i \leq k$, define $M_i = M/m_i$. Then, clearly, $\gcd(m_i, M_i) = 1$. By Lemma 1.1, there exist c_i and d_i such that $c_i M_i + d_1 m_i = 1$, so $c_i M_i \equiv 1 \pmod{m_i}$. Take $x = \sum_i a_i c_i M_i$. For any i, consider the ith term of the sum: For each $j \neq i$, $m_i | M_j$. Thus, every term in the sum other than the ith term is divisible by m_i. Hence, $x \equiv a_i c_i M_i \equiv a_i \pmod{m_i}$, which is what we needed to prove.

Now we prove uniqueness modulo M. Suppose that x and y are two different solutions to the system of congruences. Then, for each i, $x - y \equiv 0 \pmod{m_i}$. It follows that $x - y \equiv 0 \pmod{M}$, and this completes the proof. □

The Euler *phi-function* $\phi(m)$ is defined to be the number of integers less than m that are relatively prime to m. If p is a prime, then $\phi(p) = p - 1$.

Theorem 1.13. *If m and n are relatively prime positive integers, then* $\phi(mn) = \phi(m)\phi(n)$.

Proof. We compute $\phi(mn)$. For each $1 \leq i < mn$, let r_1 be the remainder of dividing i by m and let r_2 be the remainder of dividing i by n. Then $0 \leq r_1 < m$, $0 \leq r_2 < n$, $i \equiv r_1 \pmod{m}$, and $i \equiv r_2 \pmod{n}$. Furthermore, for each such r_1 and r_2, by the

Chinese Remainder Theorem, there is exactly one value i, $1 \leq i < mn$, such that $i \equiv r_1 \pmod{m}$ and $i \equiv r_2 \pmod{n}$. Consider this one-to-one correspondence between integers $1 \leq i < mn$ and pairs of integers (r_1, r_2), $0 \leq r_1 < m$, $0 \leq r_2 < n$, such that $i \equiv r_1 \pmod{m}$ and $i \equiv r_2 \pmod{n}$: Note that i is relatively prime to mn if and only if i is relatively prime to m and i is relatively prime to n. This occurs if and only if r_1 is relatively prime to m and r_2 is relatively prime to n. The number of such i is $\phi(mn)$, while the number of such pairs (r_1, r_2) is $\phi(m)\phi(n)$. Thus, $\phi(mn) = \phi(m)\phi(n)$. □

Let the prime numbers in increasing order be

$$p_1 = 2, p_2 = 3, p_3 = 5, \ldots.$$

Every positive integer a has a unique factorization as a product of powers of primes of the form

$$a = p_0^{a_o} p_1^{a_1} \cdots p_i^{a_i},$$

where $a_i = 0$ for all but at most finitely many i.

Let us compute $\phi(p^a)$ for a prime power p^a. The numbers less than p^a that are *not* relatively prime to p^a are exactly those that are divisible by p. If $n \geq p^{a-1}$, then $n \cdot p \geq p^{a-1} \cdot p = p^a$. So, the numbers less than p^a that have p as a divisor are $p, 2 \cdot p, \ldots, (p^{a-1} - 1) \cdot p$. Hence, there are $(p^{a-1} - 1)$ integers less than p^a that are not relatively prime to p^a. It follows that there are

$$\phi(p^a) = (p^a - 1) - (p^{a-1} - 1) = (p^a - p^{a-1})$$

positive integers less than p^a that are relatively prime to p^a.

We define a function f on the positive integers by $f(n) = \sum_{d|n} \phi(d)$. We need to prove for all positive integers n, that $f(n) = n$.

Lemma 1.3. *$f(p^a) = p^a$, for any prime power p^a.*

Proof. The divisors of p^a are p^j for $0 \leq j \leq a$, so

$$f(p^a) = \sum_{j=0}^{a} \phi(p^j) = 1 + \sum_{j=1}^{a} (p^j - p^{j-1}) = p^a.$$

□

Lemma 1.4. *If m and n are relatively prime positive integers, then $f(mn) = f(m)f(n)$.*

Proof. Every divisor d of mn can be written uniquely as a product $d = d_1 d_2$, where d_1 is a divisor of m and d_2 is a divisor of n. Conversely, for every divisor d_1 of m and d_2 of n, $d = d_1 d_2$ is a divisor of mn. Note that d_1 and d_2 are relatively prime. Thus, by Theorem 1.13, $\phi(d) = \phi(d_1)\phi(d_2)$. It follows that

$$\begin{aligned} f(mn) &= \sum_{d|mn} \phi(d) \\ &= \sum_{d_1|m} \sum_{d_2|n} \phi(d_1)\phi(d_2) \\ &= \sum_{d_1|m} \phi(d_1) \sum_{d_2|n} \phi(d_2) \\ &= f(m)f(n). \end{aligned}$$

□

Theorem 1.14. *For every positive integer n,* $\sum_{d|n} \phi(d) = n$.

Proof. The integer n is a product of relatively prime terms of the form p^a, so the proof follows immediately from Lemmas 1.3 and 1.4. □

1.7.3.1 Polynomials

Let $\langle F, +, \cdot, 0, 1\rangle$ be a field and let x be a symbol. The expression

$$\sum_0^k a_k x^k,$$

where the *coefficients* a_i, $i \leq k$, belong to F, is a *polynomial*. The degree of a polynomial is the largest number k such that the coefficient $a_k \neq 0$; this coefficient is called the *leading* coefficient. One adds or multiplies polynomials according to the rules of high school algebra. With these operations the set of all polynomials over F forms a *polynomial ring* $F[x]$. We say that g *divides* f, where $f, g \in F[x]$, if there is a polynomial $h \in F[x]$ such that $f = gh$. An element $a \in F$ is a *root* of a polynomial $f(x) \in F[x]$ if $f(a) = 0$.

Homework 1.9 *Verify that* $F[x]$ *is a ring.*

Theorem 1.15. *If* $f(x), g(x) \in F[x]$, $g(x)$ *is a polynomial of degree n, the leading coefficient of* $g(x)$ *is 1, and* $f(x)$ *is of degree* $m \geq n$, *then there exist unique polynomials* $q(x)$ *and* $r(x)$ *in* $F[x]$ *such that* $f(x) = q(x)g(x) + r(x)$, *and* $r(x) = 0$ *or the degree of* $r(x)$ *is less than the degree of* $g(x)$.

The proof proceeds by applying the *division algorithm*, which we now sketch: Suppose that $f(x) = \sum_0^m a_m x^m$. We can make the leading coefficient a_m vanish by subtracting from f a multiple of g, namely, $a_m x^{m-n} g(x)$. After this subtraction, if the degree is still not less than n, then we can again remove the leading coefficient by subtracting another multiple of $g(x)$. Continuing this way, we eventually have $f(x) - q(x)g(x) = r(x)$, where the degree of $r(x)$ is of lower degree than $g(x)$ or equal to zero.

Lemma 1.5. *If a is a root of $f(x) \in F[x]$, then $f(x)$ is divisible by $x-a$.*

Proof. Applying the division algorithm, we get $f(x) = q(x)(x-a)+r$, where $r \in F$ is a constant. Substitute a for the x to see that $0 = f(a) = q(a) \cdot 0 + r = r$. Thus, $f(x) = q(x)(x-r)$. □

Theorem 1.16. *If $a_1, \ldots, a_k$ are different roots of $f(x)$, then $f(x)$ is divisible by the product $(x-a_1)(x-a_2)\cdots(x-a_k)$.*

The proof is by mathematical induction in which Lemma 1.5 provides the base case $k = 1$.

Corollary 1.4. *A polynomial in $F[x]$ of degree n that is distinct from zero has at most n roots in F.*

This concludes our tutorial on polynomials. We turn now to show that the multiplicative group of the nonzero elements of a finite field is cyclic.

Let $\langle F, +, \cdot, 0, 1\rangle$ be a finite field with q elements. Then the multiplicative subgroup of nonzero elements of F has order $q-1$. Our goal is to show that this group has a generator g of order $q-1$. By Lagrange's theorem, Theorem 1.11, we know that the order of every nonzero element a in F is a divisor of $q-1$. Our first step is to show that for every positive integer $d|(q-1)$, there are either 0 or $\phi(d)$ nonzero elements in F of order d.

Lemma 1.6. *For every $d|(q-1)$, there are either 0 or $\phi(d)$ nonzero elements in F of order d.*

Proof. Let $d|(q-1)$, and suppose that some element a has order d. We will show that there must be $\phi(d)$ elements of order d. By definition, each of the elements $a, a^2, \ldots, a^d = 1$ is distinct. Each of these powers of a is a root of the polynomial $x^d - 1$. Thus, by Corollary 1.4, since every element of order d is a root of this polynomial, every element of order d must be among the powers of a. Next we will show that a^j, $1 \le j < d$, has order d if and only if $\gcd(j,d) = 1$. From this, it follows immediately that there are $\phi(d)$ elements of order d.

Let $\gcd(j,d) = 1$, where $1 \le j < d$, and suppose that a^j has order $c < d$. Then $(a^c)^j = (a^j)^c = 1$ and $(a^c)^d = (a^d)^c = 1$. By Lemma 1.1, since j and d are relatively prime, there are integers u and v such that $1 = uj + vd$. Clearly, one of these integers must be positive and the other negative. Assume that $u > 0$ and $v \le 0$. Then $(a^c)^{uj} = 1$ and $(a^c)^{-vd} = 1$, so dividing on both sides, we get $a^c = (a^c)^{uj+vd} = 1$. However, since $c < d$, this contradicts the fact that $o(a) = d$. Thus, our supposition that $o(a^j) < d$ is false; a^j has order d.

Conversely, suppose that $\gcd(j,d) = d' > 1$. Then d/d' and j/d' are integers, so $(a^j)^{d/d'} = (a^d)^{j/d'} = 1$. Thus, $o(a^j) \le d/d' < d$.

This completes the proof. □

Theorem 1.17. *The multiplicative group of the nonzero elements of a finite field is cyclic. If the finite field has q elements, then there exist $\phi(q-1)$ generators.*

Proof. Let $\langle F, +, \cdot, 0, 1\rangle$ be a finite field with q elements. We need to show that the multiplicative group $\langle F - \{0\}, +, \cdot, 1\rangle$ has a generator, an element a of order $q-1$. Every element a has some order d such that $d|(q-1)$, and by Lemma 1.6, for every such d, there are either 0 or $\phi(d)$ nonzero elements in F of order d. By Theorem 1.14, $\sum_{d|(q-1)} \phi(d) = q-1$, which is the number of elements in $F - \{0\}$. Hence, in order for every element to have some order d that is a divisor of $q-1$, it must be the case for every such d that there are $\phi(d)$ many elements of order d. In particular, there are $\phi(q-1)$ different elements $g \in F$ of order $q-1$. Thus, there is a generator, and the multiplicative group of nonzero elements of F is cyclic. □

Corollary 1.5. *If p is a prime number, then $\langle Z_p - \{[0]\}, \cdot, [1]\rangle$ is a cyclic group.*

For example, the number 2 is a generator of Z_{19}. Namely, the powers of 2 modulo 19 are 2, 4, 8, 16, 13, 7, 14, 9, 18, 17, 15, 11, 3, 6, 12, 5, 10, 1.

Homework 1.10 *What are the other generators for Z_{19}?*

Once considered to be the purest branch of mathematics, devoid of application, number theory is the mathematical basis for the security of electronic commerce, which is fast becoming an annual trillion-dollar industry. Modern cryptography depends on techniques for finding large prime numbers p and generators for Z_p – and on the (still unproven) hypotheses of the computational hardness of factoring integers.