You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

3

ALGEBRAIC STRUCTURES

INTRODUCTION

In this chapter we shall first explain what is meant by an algebraic system and then give several examples of familiar algebraic systems and discuss some of their properties. These examples show that different algebraic systems may have several properties in common. This observation provides a motivation for the study of abstract algebraic systems. For such algebraic systems, certain properties are taken as axioms of the system. Any result that is valid for an abstract system holds for all those algebraic systems for which the axioms are true.

Throughout the chapter we shall introduce certain important and useful concepts associated with algebraic systems. For example, the concept of isomorphism shows that two algebraic systems which are isomorphic to one another are structurally indistinguishable and that the results of operations in one system can be obtained from those of the other by simply relabeling the names of the elements and symbols for operations. This concept has useful applications in the sense that the results of one system permit an identical interpretation in the other system. Another important concept is that of a congruence relation which has a useful property known as substitution.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

with any one system will also be true for the other system after the labels are changed. We shall now formalize these ideas for any two algebraic systems.

Definition 3-1.1 Let $\langle X, \circ \rangle$ and $\langle Y, * \rangle$ be two algebraic systems of the same type in the sense that both $\circ$ and $*$ are binary (n-ary) operations. A mapping $g: X \to Y$ is called a *homomorphism*, or simply morphism, from $\langle X, \circ \rangle$ to $\langle Y, * \rangle$ if for any $x_1, x_2 \in X$

$$g(x_1 \circ x_2) = g(x_1) * g(x_2) \tag{2}$$

If such a function g exists, then it is customary to call $\langle Y, * \rangle$ a homomorphic image of $\langle X, \circ \rangle$, although we must note that $g(X) \subseteq Y$.

The concept of homomorphism is not restricted to algebraic systems with one binary operation. One can extend this definition to any two algebraic systems of the same type. Since in a homomorphism the operations are preserved, we shall see that several properties of the operations are also preserved.

For the algebraic systems $\langle F, \circ \rangle$ and $\langle \mathbf{Z}_4, +_4 \rangle$, the mapping $\psi: F \to \mathbf{Z}_4$ given by Eq. (1) is a homomorphism. Any mapping which satisfies the condition given by Eq. (2) is a homomorphism. In the example of the algebraic systems $\langle F, \circ \rangle$ and $\langle \mathbf{Z}_4, +_4 \rangle$, the mapping is bijective, which is a special case of homomorphism as can be seen from Definition 3-1.2 which follows. It is possible to have more than one homomorphic mapping from one algebraic system to another.

Definition 3-1.2 Let g be a homomorphism from $\langle X, \circ \rangle$ to $\langle Y, * \rangle$. If $g: X \to Y$ is onto, then g is called an *epimorphism*. If $g: X \to Y$ is one-to-one, then g is called a *monomorphism*. If $g: X \to Y$ is one-to-one onto, then g is called an *isomorphism*.

Definition 3-1.3 Let $\langle X, \circ \rangle$ and $\langle Y, * \rangle$ be two algebraic systems of the same type. If there exists an isomorphic mapping $g: X \to Y$, then $\langle X, \circ \rangle$ and $\langle Y, * \rangle$ are said to be *isomorphic*.

In the case when $\langle X, \circ \rangle$ and $\langle Y, * \rangle$ are isomorphic, then the two algebraic systems are structurally indistinguishable in the sense that they differ only in the labels used to denote the elements of the sets and the operations involved. It is easy to see that the inverse of an isomorphism is also an isomorphism. Also all the properties of the operations are preserved in an isomorphism.

Definition 3-1.4 Let $\langle X, \circ \rangle$ and $\langle Y, * \rangle$ be two algebraic systems such that $Y \subseteq X$. A homomorphism g from $\langle X, \circ \rangle$ to $\langle Y, * \rangle$ in such a case is called an *endomorphism*. If $Y = X$, then an isomorphism from $\langle X, \circ \rangle$ to $\langle Y, * \rangle$ is called an *automorphism*.

EXAMPLE 5 Show that the algebraic systems of $\langle F, \circ \rangle$ and $\langle \mathbf{Z}_4, +_4 \rangle$ given in Examples 3 and 4 are isomorphic.

SOLUTION The mapping $\psi: F \to \mathbf{Z}_4$ defined by Eq. (1) is one-to-one onto and is a homomorphism; hence ψ is an isomorphism. ////

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

3-2.2 Homomorphism of Semigroups and Monoids

The concept of homomorphism for algebraic systems was introduced in Sec. 3-1.2. Now we apply this concept to semigroups and monoids. Homomorphisms of semigroups and monoids have useful applications in the economical design of sequential machines and in formal languages.

Definition 3-2.3 Let $\langle S, * \rangle$ and $\langle T, \Delta \rangle$ be any two semigroups. A mapping $g: S \to T$ such that for any two elements $a, b \in S$,

$$g(a * b) = g(a) \Delta g(b) \qquad (1)$$

is called a *semigroup homomorphism*.

As before, a semigroup homomorphism is called a semigroup monomorphism, epimorphism, or isomorphism depending on whether the mapping is one-to-one, onto, or one-to-one onto respectively. Two semigroups $\langle S, * \rangle$ and $\langle T, \Delta \rangle$ are said to be isomorphic if there exists a semigroup isomorphic mapping from S to T.

We have already seen in Example 8, Sec. 3-1.2, that there exists a semigroup homomorphism g from $\langle \mathbf{N}, + \rangle$ to $\langle \mathbf{Z}_m, +_m \rangle$ in which the identity of $\langle \mathbf{N}, + \rangle$ is mapped into the identity $[0]$ of $\langle \mathbf{Z}_m, +_m \rangle$.

Let us now examine some of the implications of Eq. (1). For this purpose, let us assume that $\langle S, * \rangle$ is a semigroup and $\langle T, \Delta \rangle$ is an algebraic structure of the same type, that is, Δ is a binary operation on T but it is not necessarily associative. If there exists an onto mapping $g: S \to T$ such that for any $a, b \in S$ Eq. (1) is satisfied, then we can show that Δ must be associative and hence $\langle T, \Delta \rangle$ must be a semigroup. In order to see this, let $a, b, c \in S$

$$\begin{aligned} g((a * b) * c) &= g(a * b) \Delta g(c) \\ &= (g(a) \Delta g(b)) \Delta (g(c)) \end{aligned}$$

On the other hand,

$$g(a * (b * c)) = g((a * b) * c)$$

but $g(a * (b * c))$ can be shown to be equal to $g(a) \Delta (g(b) \Delta g(c))$ by a similar argument. Hence Δ is associative, and $\langle T, \Delta \rangle$ must be a semigroup. This result shows that Eq. (1) preserves the semigroup character because it preserves associativity.

Next, note that if g is a semigroup homomorphism from $\langle S, * \rangle$ to $\langle T, \Delta \rangle$, then for any element $a \in S$ which is idempotent, we must have $g(a)$ idempotent, because

$$g(a * a) = g(a) = g(a) \Delta g(a)$$

The property of idempotency is preserved under the semigroup homomorphism. In a similar manner commutativity is also preserved.

If $\langle S, * \rangle$ is a semigroup with an identity e, that is, $\langle S, *, e \rangle$ is a monoid and g is a homomorphism from $\langle S, * \rangle$ to a semigroup $\langle T, \Delta \rangle$, then for any $a \in S$,

$$g(a * e) = g(e * a) = g(a) \Delta g(e) = g(e) \Delta g(a) = g(a)$$

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

cording to Theorem 3-5.1. Equation (4) can be written as

$$f(a * b) = f(a) \diamond f(b)$$

showing that f is an isomorphism. ////

Example 2 of Sec. 3-2.2 is an illustration of the representation theorem. This representation theorem is also known as Cayley's representation theorem. It was proposed by Arthur Cayley in 1854. This theorem shows that the structure of a group is determined solely by its composition table.

EXERCISES 3-5.2

1 Find all the subgroups of (*a*) $\langle \mathbf{Z}_{12}, +_{12} \rangle$; (*b*) $\langle \mathbf{Z}_5, +_5 \rangle$; (*c*) $\langle \mathbf{Z}_7^*, \times_7 \rangle$; and (*d*) $\langle \mathbf{Z}_{11}^*, \times_{11} \rangle$.

2 Find the group of rigid rotations of a rectangle which is not a square. Show that this is a subgroup of $\langle D_4, \diamond \rangle$ given in Table 3-5.7.

3 Find all the subgroups of S_4 generated by the permutations

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 3 & 2 & 4 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 3 & 4 & 2 \end{pmatrix}$$

4 Show that the set of all elements a of a group $\langle G, * \rangle$ such that $a * x = x * a$ for every $x \in G$ is a subgroup of G.

5 Show that if $\langle G, * \rangle$ is a cyclic group, then every subgroup of $\langle G, * \rangle$ must be cyclic.

6 Show that $\langle \{1, 4, 13, 16\}, \times_{17} \rangle$ is a subgroup of $\langle \mathbf{Z}_{17}^*, \times_{17} \rangle$.

7 Let $\langle G, * \rangle$ be a group and $a \in G$. Let $f: G \to G$ be given by $f(x) = a * x * a^{-1}$ for every $x \in G$. Prove that f is an isomorphism of G onto G.

8 Show that the groups $\langle G, * \rangle$ and $\langle S, \Delta \rangle$ given by the following table are isomorphic.

$*$	p_1	p_2	p_3	p_4
p_1	p_1	p_2	p_3	p_4
p_2	p_2	p_1	p_4	p_3
p_3	p_3	p_4	p_1	p_2
p_4	p_4	p_3	p_2	p_1

Δ	q_1	q_2	q_3	q_4
q_1	q_3	q_4	q_1	q_2
q_2	q_4	q_3	q_2	q_1
q_3	q_1	q_2	q_3	q_4
q_4	q_2	q_1	q_4	q_3

3-5.3 Cosets and Lagrange's Theorem

From the definition of a subgroup it is clear that not every subset of a group is a subgroup. The problem that we try to solve here is to find those subsets which can qualify to become subgroups. An important relationship exists between the subgroups and the group itself. This relationship is explained by a theorem known as Lagrange's theorem, which is proved in this section. This theorem has important applications in the development of efficient group codes required in the transmission of information. Such group codes are discussed in Sec. 3-8. Another application of subgroups is in the construction of computer modules which perform group operations. Such modules are constructed by joining various subgroup modules that do operations in subgroups. The application of these to the design of fast adders is discussed in Sec. 3-7.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

ing identities:

(C-1) $a * a' = 0$ **(C-1)'** $a \oplus a' = 1$

(C-2) $0' = 1$ **(C-2)'** $1' = 0$

(C-3) $(a * b)' = a' \oplus b'$ **(C-3)'** $(a \oplus b)' = a' * b'$

(See Definition 4-1.9 and Prob. 5, Exercises 4-1.5.)

5 There exists a partial ordering relation $\leq$ on B such that

(P-1) $a * b = \text{GLB}\ \{a, b\}$ **(P-1)'** $a \oplus b = \text{LUB}\ \{a, b\}$

(P-2) $a \leq b \Leftrightarrow a * b = a \Leftrightarrow a \oplus b = b$

(P-3) $a \leq b \Leftrightarrow a * b' = 0 \Leftrightarrow b' \leq a' \Leftrightarrow a' \oplus b = 1$

(See Theorem 4-1.1 and Prob. 6, Exercises 4-1.5.)

As pointed out earlier, not all the identities given here are independent of one another. These identities arose by looking at a Boolean algebra as a special lattice. It is possible to define a Boolean algebra as an abstract algebraic system satisfying certain properties which are independent of each other. In fact, even the two binary operations $*$ and $\oplus$, the unary operation $'$, and the two distinguished elements are not all independent. One can define a Boolean algebra in terms of the operations $*$ and $'$ and a set of independent properties satisfied by these operations. We shall not, however, concern ourselves with this approach.

EXAMPLE 1 Let $B = \{0, 1\}$ be a set. The operations $*$, $\oplus$, and $'$ on B are given by Table 4-2.1. The algebra $\langle B, *, \oplus, ', 0, 1\rangle$ satisfies all the properties listed here and is one of the simplest examples of a two-element Boolean algebra. A two-element Boolean algebra is the only Boolean algebra whose diagram is a chain.

EXAMPLE 2 Let S be a nonempty set and $\rho(S)$ be its power set. The set algebra $\langle \rho(S), \cap, \cup, \sim, \varnothing, S\rangle$ is a Boolean algebra in which the complement of any subset $A \subseteq S$ is $\sim A = S - A$, the relative complement of the set A. If S has n elements, then $\rho(S)$ has 2^n elements and the diagram of the Boolean algebra is an n cube. The partial ordering relation on $\rho(S)$ corresponding to the operations $\cap$ and $\cup$ is the subset relation $\subseteq$. The diagrams for the Boolean algebra $\langle \rho(S), \cap, \cup\rangle$ when S has 1, 2, and 3 elements are given in Fig. 4-2.1. If S is an empty set, then $\rho(S)$ has only one element, viz., $\varnothing$, so that $\varnothing = 0 = 1$, and the corresponding Boolean algebra is a degenerate Boolean algebra. We shall consider nondegenerate Boolean algebras only.

Table 4-2.1

$*$	0	1	$\oplus$	0	1	x	x'
0	0	0	0	0	1	0	1
1	0	1	1	1	1	1	0

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

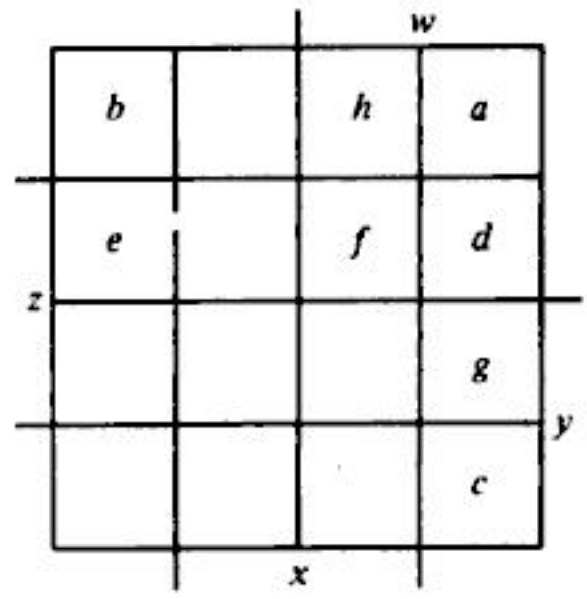


FIGURE 4-4.7

on its inner surface in the same orientation. Similar remarks hold for higher-order maps.

To this point we have discussed Karnaugh map structures, indicating what form they take for functions of different numbers of variables and describing how they are to be interpreted. To represent a Boolean function by means of a Karnaugh map, we select the Karnaugh map structure appropriate for that function and place 0s and 1s in the cells according to whether the functional value is 0 or 1 for the input combination associated with that cell. (Because one half of the map is associated with a 1 input value for a variable and the other half of the map with a 0 input value for that same variable, the input combinations are mapped one-to-one onto the cells of the Karnaugh map structure. The input combination serves as a sort of "address" for a cell within a Karnaugh map structure.) The Karnaugh map for the Boolean function $f = x_1 \cdot [x_2 + (x_3 \cdot \bar{x}_4)]$ is given in Fig. 4-4.8 (see also Fig. 4-4.5). Observe that only the functional values of 1 have been written in the appropriate cells; a blank cell is presumed to have a 0 in it. Also, for convenience, the map of the function f has had each of its cells labeled with the input combination to which it corresponds.

The last method for representing Boolean functions is one that was previously encountered in Sec. 1-2.15, namely, *circuit diagrams*. This representation does seem appropriate since Boolean functions can express the functioning of circuits. Because a circuit diagram actually shows which circuits are to be connected to which other circuits, it is occasionally possible to make use of a circuit diagram to eliminate unnecessary connectives and thus yield a simpler circuit.

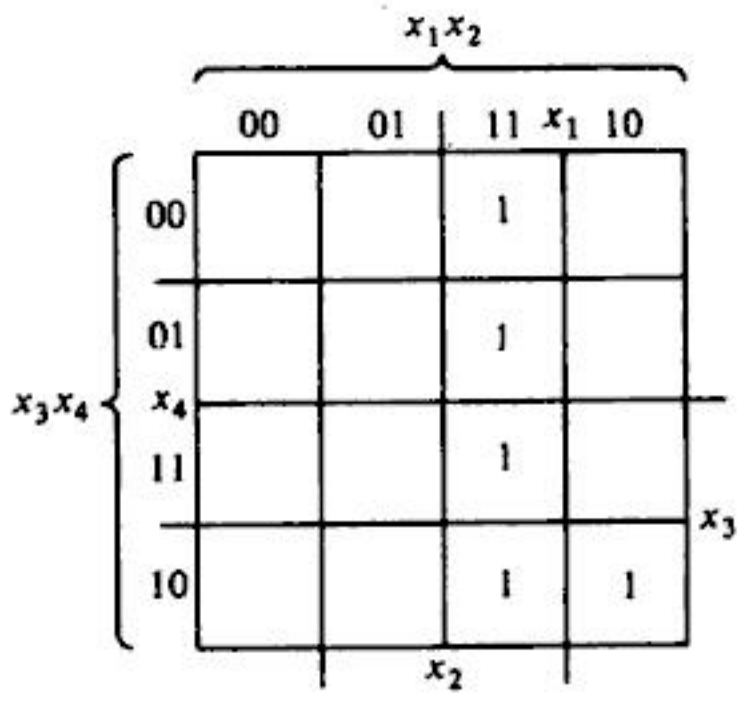


FIGURE 4-4.8

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

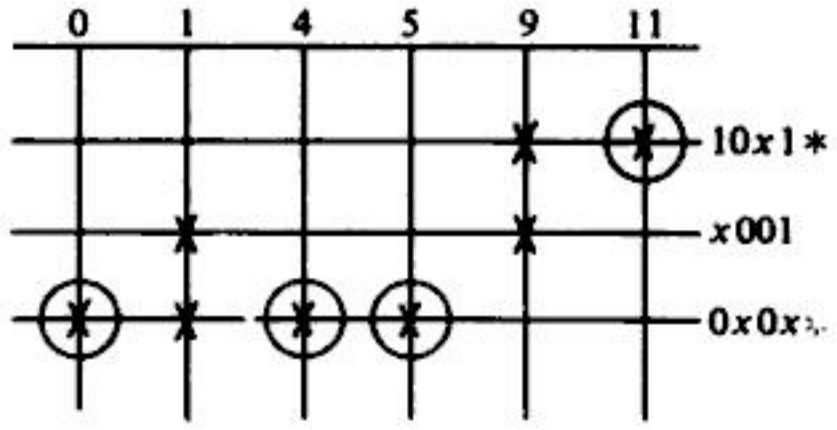


FIGURE 4-4.13 Prime implicant chart for $f = \sum(0, 1, 4, 5, 9, 11)$.

An alternate method for selecting a suitable minimum sum from the group of prime implicants is to construct a prime-implicant chart which for this problem is given in Fig. 4-4.13. The chart contains a number of columns, each of which has a number at the top that corresponds to a minterm in the sum-of-products form of the function. Each row corresponds to one of the prime implicants, as identified by $10x0$, $x001$, and $0x0x$ at the right. In each row we mark a cross under each minterm contained in the prime implicant represented by that row. Thus, in the example the first prime implicant $10x1$ contains or covers minterms 11 and 9. The remainder of the chart is completed in a similar manner.

The first step in using the prime-implicant chart is to examine the columns to see whether any column has exactly one cross in it. This is true for columns 0, 4, 5, and 11. We place circles around each of the crosses which stand alone in a column, and then we rule a line through all the crosses in each row containing a circled cross. The significance of this maneuver is that the particular prime implicant which has been marked is the only one which can cover the required encircled minterm. However, since it also covers all other minterms designated by a cross in the same row, no other prime implicants need be chosen to cover these minterms. A single asterisk is placed at the end of each prime implicant thus required. Such rows are called *primary basis rows*. In addition to covering the minterms under which the crosses are circled, each primary basis row covers other minterms where other crosses lie in that row. Thus row $0x0x$ in Fig. 4-4.13 covers not only minterms 0, 4, and 5 but also minterm 1.

We continue this process for the columns containing crosses which are in the other primary basis row, and at the end we have covered all minterms. Hence we have determined the minimum sum $\bar{x}_1\bar{x}_3 + x_1\bar{x}_2x_4$.

A general method that can be used to generate automatically all the prime implicants that cover a given set of minterms is the *Quine-McCluskey algorithm*. First, the 0 cubes are used to generate all the possible 1 cubes; then these 1 cubes are used to generate all possible 2 cubes; this process continues until the r cubes are generated for some r such that there are no $(r + 1)$ cubes. The cubes that remain after the elimination of all cubes covered by higher-order cubes are the prime implicants.

Consider the example of $g = x_1 \cdot (x_2 + x_3\bar{x}_4)$ which can be represented by the standard sum $\sum$ (10, 12, 13, 14, 15). Writing the minterms in cube form, grouping them in order of increasing number of 1s in each 0 cube, and finally applying the Quine-McCluskey algorithm to this set of ordered cubes gives the result shown in Fig. 4-4.14. The arrows in the diagram have been drawn and labeled to indicate which cubes generated the higher-order cubes. Observe that

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

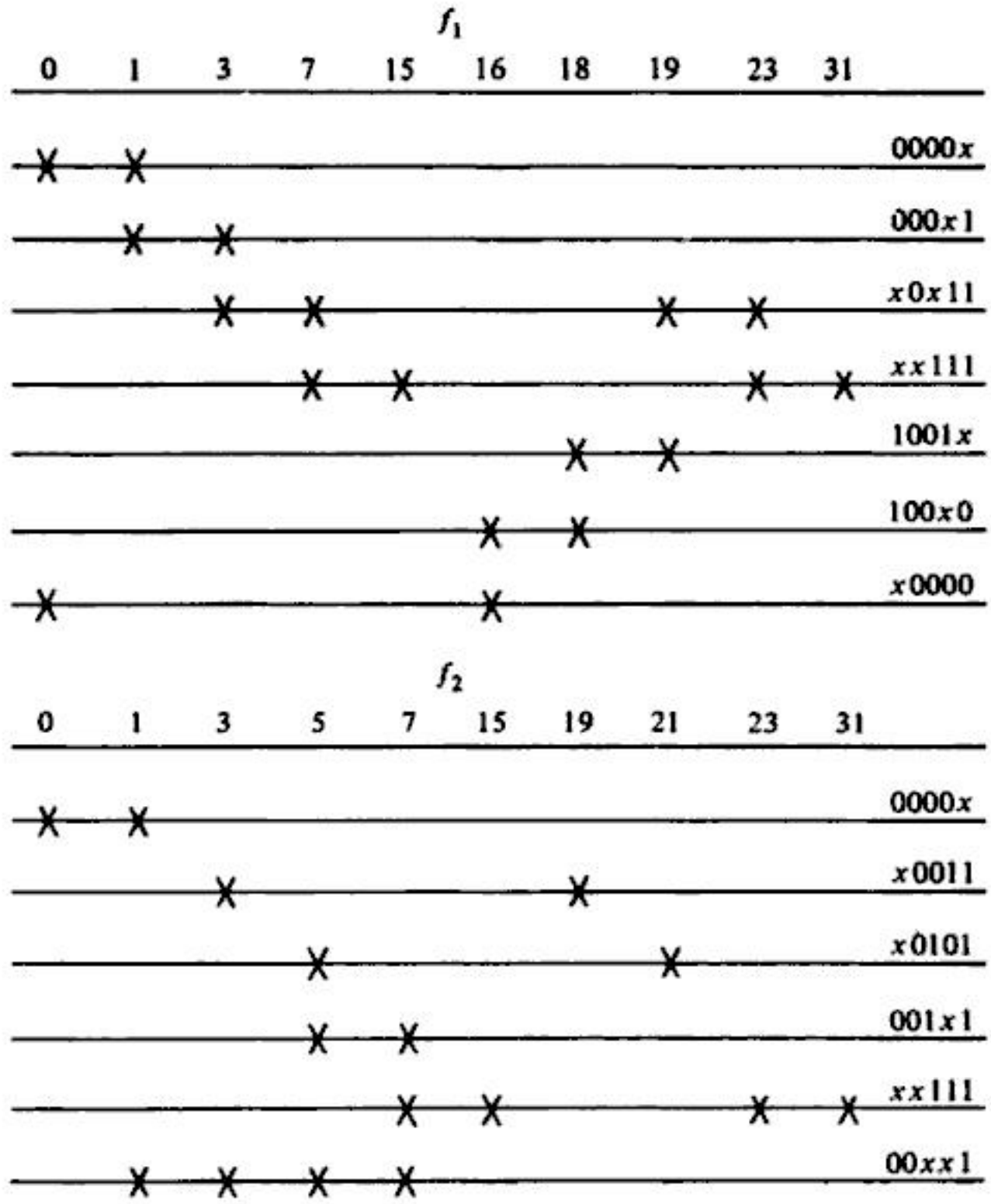


FIGURE 4-4.20

will find all possible minimal covers of a set of 0 cubes, where minimal means that the cover does not contain redundant cubes.

The first step that should be taken in your algorithm is to determine the essential prime implicants. Then the problem reduces to finding the minimal covers of nonessential prime implicants for cubes not yet covered. A brute-force solution would be to test all combinations of nonessential prime implicants. There are other solutions to this problem however.

4-5 DESIGN EXAMPLES USING BOOLEAN ALGEBRA

In this section we illustrate how Boolean algebra is used in the design of some simple switching circuits that perform various arithmetic operations on numbers. We are concerned with fixed-length binary numbers, since in general most digital computers manipulate binary numbers of fixed length.

Initially, we give a brief discussion of arithmetic operations that can be performed on integral binary numbers. Let $a = \langle a_1, a_2, \ldots, a_n \rangle$ and $b = \langle b_1, b_2, \ldots, b_n \rangle$ denote two binary numbers. The binary addition table given in Table 4-5.1 is simple. Note, however, that in the case of two 1s, a 2-digit sequence is

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

5	00101	00101	−9
+ −9	+ −01001	+ 10110	+ −5
−4	Change to	11011	−14

10110
+ 11010
110000 → 10001
↑
end-around carry,
no overflow

7	00111	−7	11000	−7	11000		11000
+ +9	+ 01001	+ −9	+ 10110	− −9	− 10110	+	01001
16	10000		101110	2	Change to		100001 ⇒ 00010
	↑		↑				↑
	overflow		end-around carry *but* also overflow				end-around carry

Another form by which signed binary numbers can be represented is the 2s complement representation. Again, numbers represented in this fashion have all the bits (including the sign bit) taking part in the arithmetic operation, and, as in the case of 1s complement, subtraction is unnecessary. Addition of the second operand in 2s complement form yields the same result as the subtraction of the second operand from the first operand. There are no end-around carries. The sign bit always comes out correct, except in the case of an overflow, for which no correction is possible. Except for one case, overflows are detected by checking the signs of the operands. If, when actually adding, the two operands have the same sign and the result has the opposite sign, then an overflow has occurred. The one exceptional case that cannot be detected by this method is the following: if a and b are n-bit words representing negative binary numbers (that is, 1 sign bit and $n - 1$ magnitude bits), then if $|a| + |b| = 2^{n-1}$, an unused word is generated ($100\cdots0$) with a carry of 1 out of the sign bit. To detect this overflow, we must check for the magnitude bits being all 0 and for the carry-out bit value of 1.

To form the 2s complement representation of a number, first form the 1s complement representation of it (including the sign bit in the scope of the 1s complementation operation) and then add 1 to the low-order bit position. Another method is to copy bits from the original number, starting at the low-order end until the first 1 bit is encountered. After copying this first 1 bit, then the complements of each of the remaining bits are copied. Thus to form the 2s complement representation of +9, we have 01001 ⇒ 10110 + 00001 = 10111. Using the alternate method, we would have

01001 ⇒	1	(copy until first 1 is copied)
	10111	(copy complements of remaining bits)

Again, in the 2s complement representation, positive numbers are identical to positive numbers in sign-magnitude form or in 1s complement form. Negative numbers are represented as the result of performing a 2s complementation of the corresponding positive number of the same magnitude. Some examples of 2s com-

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

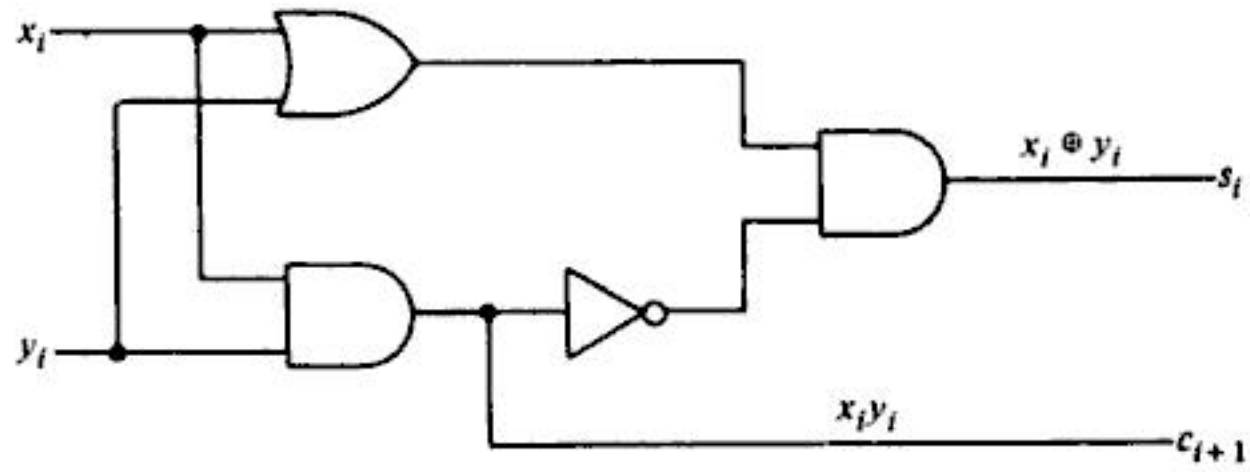


FIGURE 4-5.5 Circuit diagram for a half-adder module.

Again, we can use the circuit developed for the 1s complementer. In fact, that circuit with one extra connection is the required half-adder circuit. This circuit is given in Fig. 4-5.5.

Half-adders are not of much use by themselves in actually performing arithmetic operations because they cannot handle carry bits, and, in general, carry bits must be processed when an addition is performed. However, half-adders are very useful as components from which we can construct larger units that do perform arithmetic operations, e.g., a full-adder.

A full-adder module accepts two input data bits and a carry bit from the preceding full-adder module and generates the proper sum and carry bits. Figure 4-5.6 illustrates such a module with its interface to other modules in the iterative network. In tabular form, the requirements for a full-adder module are given in Table 4-5.6. From this table, we can construct equations as follows:

$$s_i = \bar{x}_i\bar{y}_i c_i + \bar{x}_i y_i \bar{c}_i + x_i \bar{y}_i \bar{c}_i + x_i y_i c_i$$

$$c_{i+1} = \bar{x}_i y_i c_i + x_i \bar{y}_i c_i + x_i y_i \bar{c}_i + x_i y_i c_i$$

These equations can be simplified or modified in such a manner that previous circuits, such as half-adders, can be used to construct new circuits. Formally,

$$\begin{aligned} s_i &= \bar{x}_i\bar{y}_i c_i + \bar{x}_i y_i \bar{c}_i + x_i \bar{y}_i \bar{c}_i + x_i y_i c_i \\ &= \bar{x}_i(\bar{y}_i c_i + y_i \bar{c}_i) + x_i(\bar{y}_i \bar{c}_i + y_i c_i) && \text{(using the distributive law)} \\ &= \bar{x}_i(y_i \oplus c_i) + x_i(\bar{y}_i \bar{c}_i + y_i c_i) && \text{(from the definition of } \oplus) \\ &= x_i \oplus (y_i \oplus c_i) && \text{[using Eq. (2)]} \end{aligned}$$

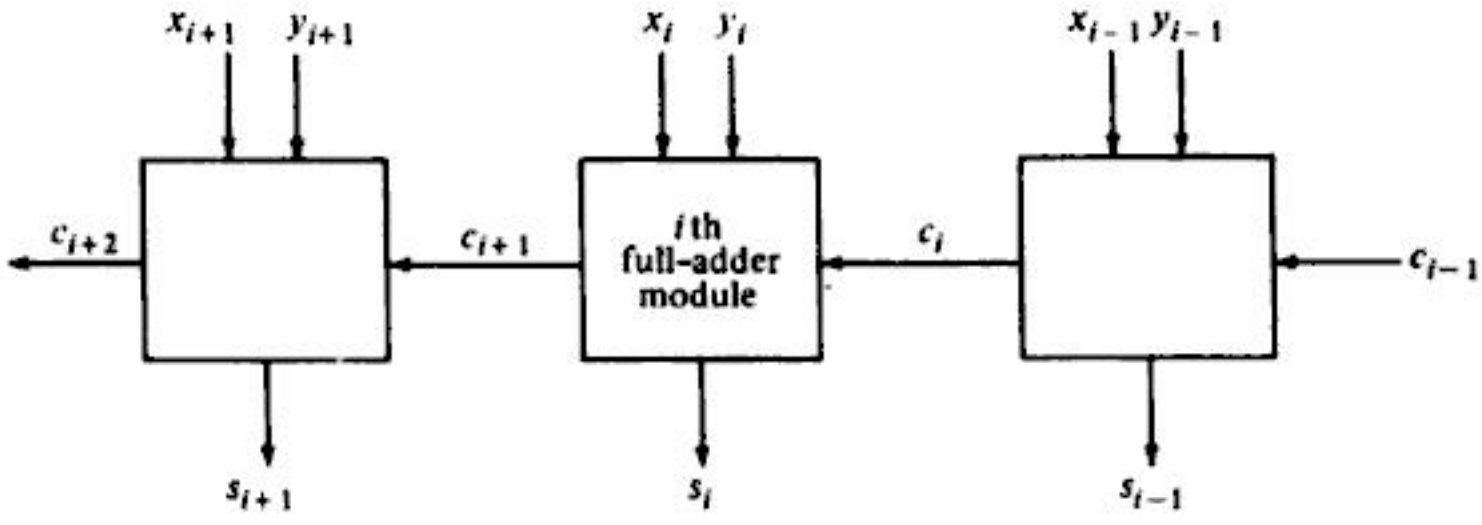


FIGURE 4-5.6 Typical module interface in a full-adder.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

(1) $x - y$
(2) $x + y$
(3) $-x + y$

(*b*) Repeat part (*a*), performing subtraction by complementing and adding.
(*c*) Repeat part (*a*), using 2s complement representation of signed numbers.

3 Design a gate network that will have as input two 2-bit numbers and present the 4-bit product on four output terminals.

4 One form of an old puzzle requires a farmer (f) to transport a wolf (w), a goose (g), and a sack of corn (c) across a river, subject to the condition that if left unattended, the wolf will eat the goose, or the goose will eat the corn. Let $f = 0$ indicate the presence of the farmer on the west bank, $f = 1$ indicate his presence on the east bank, and similarly for w, g, and c.

(*a*) Write the table of combinations describing a switching circuit which has a transmission of 1 if and only if the farmer is in danger of losing the goose or corn. Assume that if an object is not on one side of the river, it must be on the other side.
(*b*) Design a two-level minimal circuit having the required transmission, and show the circuit.

5 Design a circuit which determines whether four signals on four input lines represent a valid BCD (Binary Coded Decimal) code word. The BCD code is given in Table 4-5.9.

Table 4-5.9

	Inputs			
Decimal number	***a***	***b***	***c***	***d***
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

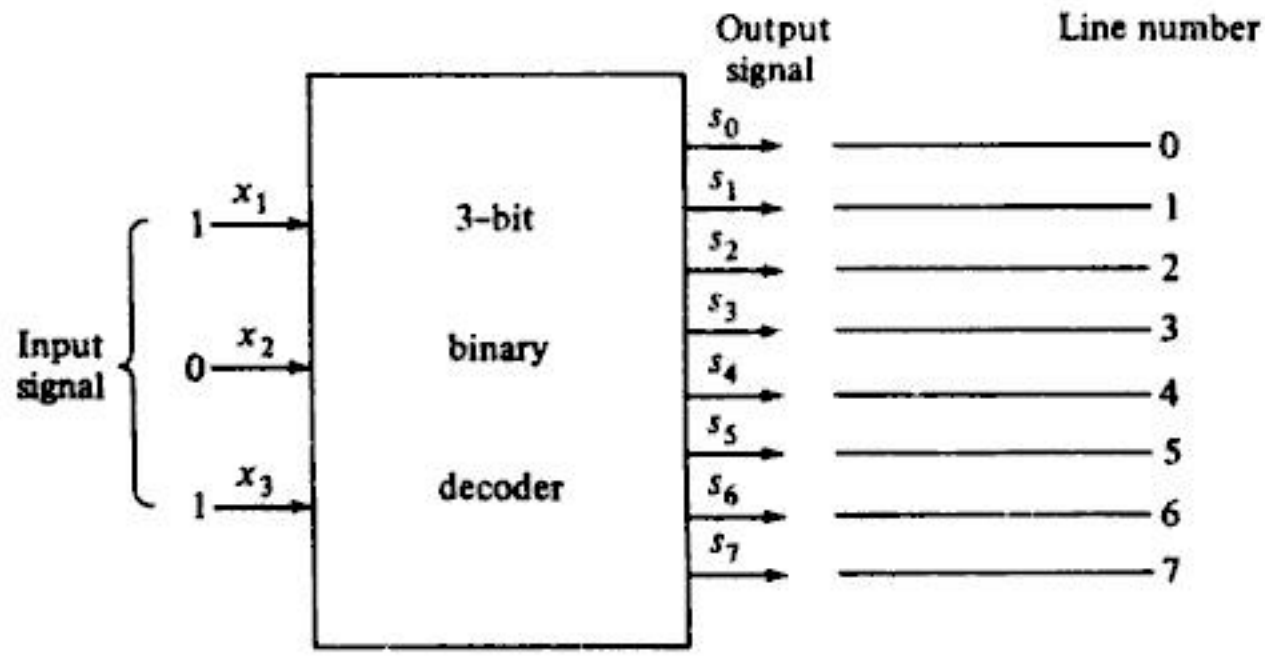


FIGURE 4-5.13 A 3-bit binary decoder.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

delay element will be represented by c, and it will assume a value of 0 or 1, corresponding to the absence or presence of a carry bit. Since the present value at the input of the unit delay at time t_i is equal to its output at time t_{i+1}, this input is called the next state of the delay element. A block-diagram representation of the adder is given in Fig. 4-6.2, where an additional input line denoting the clock signal is also included. This additional input indicates that the serial adder is a synchronous circuit and that all events occur at discrete points in time.

We have given a very brief introduction to sequential circuits. In the next subsection we formalize these ideas and proceed with the development of an algorithm for determining a minimal equivalent sequential circuit for a given sequential circuit.

4-6.2 Equivalence of Finite-state Machines

In the previous subsection the basic notions of sequential circuits were introduced. We now wish to formalize these concepts by defining a finite-state machine. Furthermore, the important question of equivalent machines is discussed. In particular, we shall define what is meant by equivalent machines and show that for any given machine there exists a minimal equivalent machine. This reduced machine is homomorphic to the given machine. An algorithm for obtaining a minimal machine is developed. Finite-state machines have interesting algebraic properties based on the theory of semigroups, but we will not be concerned with such properties in this subsection. Finite-state machines can do many things, but there are certain operations such as multiplication which are beyond their range. We now proceed to the definition of a finite-state machine.

Definition 4-6.1 A *sequential machine*, or *finite-state machine*, is a system $N = \langle I, S, O, \delta, \lambda \rangle$, where the finite sets I, S, and O are alphabets that represent the input, state, and output symbols of the machine respectively. The alphabets I and O are not necessarily disjoint, but $I \cap S = O \cap S = \varnothing$. We shall denote the alphabets by

$$I = \{a_0, a_1, \ldots, a_n\} \qquad S = \{s_0, s_1, \ldots, s_m\} \qquad O = \{o_0, o_1, \ldots, o_r\}$$

δ is a mapping of $S \times I \rightarrow S$ which denotes the next-state function, and λ is a mapping $S \times I \rightarrow O$ which denotes the output function. We assume that the machine is in an initial state s_0.

Formally, a finite-state machine therefore consists of three not necessarily distinct alphabets and two functions. An abstract representation of a finite-state machine is given in Fig. 4-6.3. The machine reads a sequence of input symbols that are stored on an *input tape* and stores a sequence of output symbols on an *output tape*. Let the machine be in some state s_i and reading the input symbol a_p under its reading head. The mapping λ is then applied to s_i and a_p, thus causing the writing head to record a symbol o_k on the output tape. The function δ then causes the machine to go into state s_j. The machine proceeds to read the next input symbol and continues its operation until all symbols on the input tape are processed. Observe that the tapes are allowed to move only in one direction. In Fig. 4-6.3 the input symbols are processed from left to right. It should be

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Definition 4-6.7 A finite-state machine $M = \langle I, S, O, \delta, \lambda \rangle$ is said to be *reduced* if and only if $s_i \equiv s_j$ implies that $s_i = s_j$ for all states $s_i, s_j \in S$.

In other words, a reduced finite-state machine is one in which each state is equivalent to itself and to no other. The partition of S in such a machine has all its equivalence classes consisting of a single element.

We shall now show how to construct a reduced finite-state machine M' which is equivalent to some given machine M. Let S in M be partitioned in a set of equivalence classes $[s]$ such that $P = \cup [s]$. Let the function ϕ be defined on the partition P such that $\phi([s]) = s'$, where s' is an arbitrary fixed element of $[s]$, called a representative. It is clear that $s' \equiv s$ in M. Let S' in M' be defined as

$$S' = \{s' \mid (\exists s)[s \in S \text{ and } \phi([s]) = s']\}$$

and let $I' = I$ and $O' = O$; that is, both machines will have the same input and output alphabets. The functions δ' and λ' are defined as follows:

$$\delta'(s', a) = \phi([\delta(s', a)])$$

and

$$\lambda'(s', a) = \lambda(s', a)$$

where s' is both in S and S'. Therefore the reduced machine is $M' = \langle I, S', O, \delta', \lambda' \rangle$.

Applying this procedure to the machine given in Table 4-6.3 gives the equivalent reduced machine in Table 4-6.4.

We shall now state a theorem without proof which shows the existence of a reduced equivalent machine.

Theorem 4-6.5 Let $M = \langle I, S, O, \delta, \lambda \rangle$ be a finite-state machine. Then there exists an equivalent machine M' with a set of states S' such that $S' \subseteq S$ and M' is reduced.

The idea of one finite-state machine simulating another is very important in a number of applications. This notion is formalized in the next definition.

Definition 4-6.8 Let $M = \langle I, S, O, \delta, \lambda \rangle$ and $M' = \langle I, S', O, \delta', \lambda' \rangle$ be two finite-state machines. Let function ϕ be a mapping from S into S'. A *finite-state homomorphism* is defined as

$$\left.\begin{aligned} \phi(\delta(s, a)) &= \delta'(\phi(s), a) \\ \lambda(s, a) &= \lambda'(\phi(s), a) \end{aligned}\right\} \quad \text{for all } a \in I$$

If ϕ is a one-one and onto function, then M is *isomorphic* to M'.

Finite-state machines are often used in compilers where they usually perform the task of a scanner (Sec. 3-3). The machine in such a case does lower-level syntax analysis such as identifying variable names, operators, constants, etc. A machine which performs this scanning task is called an *acceptor*. In Chap. 6 we show that the set of languages that can be recognized by an acceptor is exactly the set of those languages that can be generated by a regular grammar.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

5-1 BASIC CONCEPTS OF GRAPH THEORY

The terminology used in graph theory is not standard. It is not uncommon to find several different terms being used as synonyms. This situation, however, becomes more complicated when we find that a particular term is used by different authors to describe different concepts. This situation is natural because of the diversity of the fields in which graph theory is applied. Wherever possible, we shall indicate the alternative terms which are frequently used. We shall generally select alternatives that are often used in the literature in computer science.

In this section we shall define a graph as an abstract mathematical system. However, in order to provide some motivation for the terminology used and also to develop some intuitive feelings, we shall represent graphs diagrammatically. Any such diagram will also be called a graph. Our definitions and terms are not restricted to those graphs which can be represented by means of diagrams, even though this may appear to be the case because these terms have strong associations with such a representation. We shall see later on that a diagrammatic representation is only suitable in some very simple cases. Alternative methods of representing graphs will also be discussed. After introducing the terminology, we shall also discuss some of the basic results and theorems of graph theory.

5-1.1 Basic Definitions

Recall that in Chap. 2 a binary relation in a set V was defined as a subset of $V \times V$. It was shown that such a relation could be represented at least in some cases by a diagram which was called the graph of the relation. An alternative method of representing a relation was given by means of a relation or an incidence matrix. In this section we shall extend these ideas and in some ways generalize them.

We first consider several graphs which are represented by means of diagrams. Some of these graphs may be considered as graphs of certain relations, but there are others which cannot be interpreted in this manner.

Consider the diagrams shown in Fig. 5-1.1. For our purpose here, these diagrams represent graphs. Notice that every diagram consists of a set of points which are shown by dots or circles and are sometimes labeled $v_1, v_2, \ldots$, or $1, 2, \ldots$. Also in every diagram certain pairs of such points are connected by lines or arcs. The other details, such as the geometry of the arcs, their lengths, the position of the points etc., are of no importance at present. Notice that every arc starts at one point and ends at another point. A definition of the graph which is essentially an abstract mathematical system will now be given. Such a mathematical system is an abstraction of the graphs given in Fig. 5-1.1.

Definition 5-1.1 A *graph* $G = \langle V, E, \phi \rangle$ consists of a nonempty set V called the set of *nodes* (*points, vertices*) of the graph, E is said to be the set of *edges* of the graph, and ϕ is a mapping from the set of edges E to a set of ordered or unordered pairs of elements of V

We shall assume throughout that both the sets V and E of a graph are finite. It would be convenient to write a graph G as $\langle V, E \rangle$, or simply as G. Notice

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

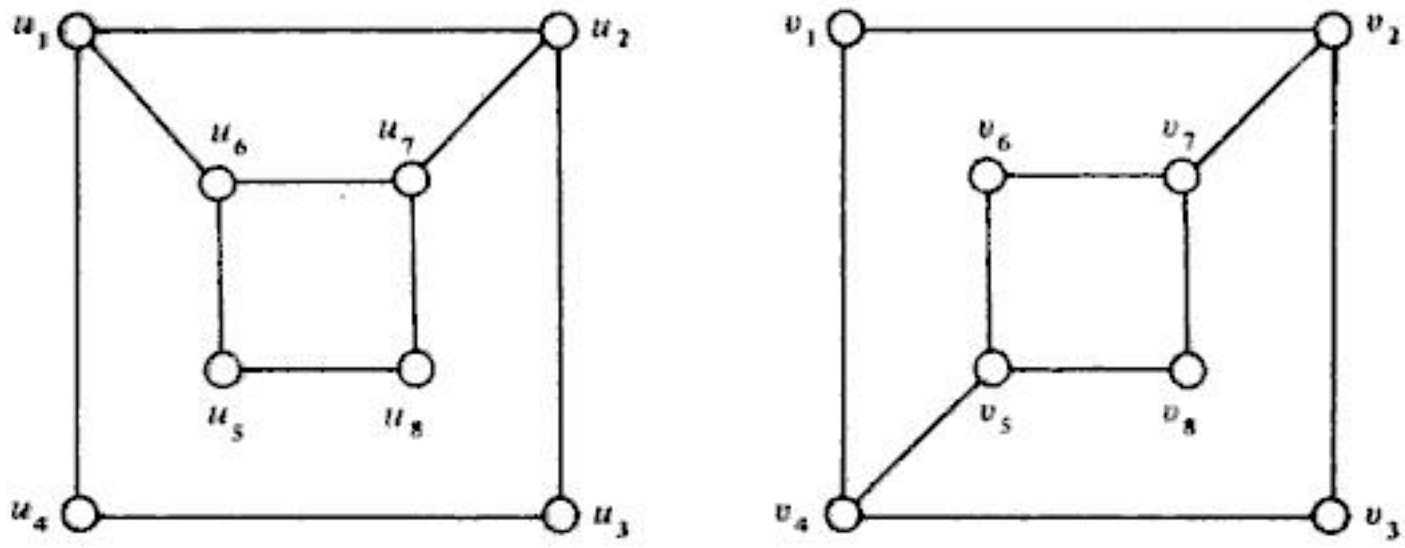


FIGURE 5-1.7

State the properties of these digraphs such as symmetry, transitivity, antisymmetry, etc.

3 Show that the digraphs given in Fig. 5-1.5*a* and *b* are isomorphic.

4 Show that the digraphs given in Fig. 5-1.6*a* and *b* are isomorphic.

5 Show that the digraphs in Fig. 5-1.7 are not isomorphic.

6 A simple digraph $G = \langle V, E \rangle$ is said to be *complete* if every node is adjacent to all other nodes of the graph. Show that a complete digraph with n nodes has the maximum number of edges viz., $n(n-1)$ edges, assuming that there are no loops.

7 The *complement* of a simple digraph $G = \langle V, E \rangle$ is the digraph $\bar{G} = \langle V, \bar{E} \rangle$ where $\bar{E} = V \times V - E$. Find the complements of the graphs given in Prob. 2.

5-1.2 Paths, Reachability, and Connectedness

In this section we introduce some additional terminology associated with a simple digraph. During the course of our discussion we shall also indicate how the same terminology and concepts can be extended to simple undirected graphs as well as to multigraphs.

Let $G = \langle V, E \rangle$ be a simple digraph. Consider a sequence of edges of G such that the terminal node of any edge in the sequence is the initial node of the next edge, if any, in the sequence. An example of such a sequence is

$$(\langle v_{i_1}, v_{i_2} \rangle, \langle v_{i_2}, v_{i_3} \rangle, \ldots, \langle v_{i_{k-2}}, v_{i_{k-1}} \rangle, \langle v_{i_{k-1}}, v_{i_k} \rangle)$$

where it is assumed that all the nodes and edges appearing in the sequence are in V and E respectively. It is customary to write such a sequence as

$$(v_{i_1}, v_{i_2}, \ldots, v_{i_{k-1}}, v_{i_k})$$

Note that not all edges and nodes appearing in a sequence need be distinct. Also, for a given graph any arbitrary set of nodes written in any order do not give a sequence as required. In fact each node appearing in the sequence must be adjacent to the nodes appearing just before and after it in the sequence, except in the case of the first and last nodes.

Definition 5-1.5 Any sequence of edges of a digraph such that the terminal node of any edge in the sequence is the initial node of the edge, if any, appearing next in the sequence defines a *path* of the graph.

A path is said to *traverse* through the nodes appearing in the sequence, *orig-*

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

3 Given a simple digraph $G = \langle V, E \rangle$, under what condition is the equation

$$d(v_1, v_2) + d(v_2, v_3) = d(v_1, v_3)$$

satisfied for v_1, v_2, and $v_3 \in V$?

4 Find the reachable sets of $\{v_1, v_4\}$, $\{v_4, v_5\}$, and $\{v_3\}$ for the digraph given in Fig. 5-1.14.

5 Find a node base for each of the digraphs given in Figs. 5-1.13 and 5-1.14.

6 Explain why no node in a node base is reachable from another node in the node base.

7 Prove that in an acyclic simple digraph a node base consists of only those nodes whose indegree is zero.

8 For the digraphs given in Figs. 5-1.13 and 5-1.14, determine whether they are strongly, weakly, or unilaterally connected.

9 Show that a simple digraph is strongly connected iff there is a cycle in G which includes each node at least once and no isolated node.

10 The *diameter* of a simple digraph $G = \langle V, E \rangle$ is given by δ, where

$$\delta = \max_{u,\, v \in V} d(u, v)$$

Find the diameter of the digraphs given in Figs. 5-1.13 and 5-1.14.

11 Find the strong components of the digraph given in Fig. 5-1.14. Also find its unilateral and weak components.

12 Show that every node and edge of a graph are contained in exactly one weak component.

5-1.3 Matrix Representation of Graphs

A diagrammatic representation of a graph has limited usefulness. Furthermore, such a representation is only possible when the number of nodes and edges is reasonably small. In this subsection we shall present an alternative method of representing graphs using matrices. Such a method of representation has several advantages. It is easy to store and manipulate matrices and hence the graphs represented by them in a computer. Well-known operations of matrix algebra can be used to calculate paths, cycles, and other characteristics of a graph.

Given a simple digraph $G = \langle V, E \rangle$, it is necessary to assume some kind of ordering of the nodes of the graph in the sense that a particular node is called a first node, another a second node, and so on. Our matrix representation of G depends upon the ordering of the nodes.

Definition 5-1.12 Let $G = \langle V, E \rangle$ be a simple digraph in which $V = \{v_1, v_2, \ldots, v_n\}$ and the nodes are assumed to be ordered from v_1 to v_n. An $n \times n$ matrix A whose elements a_{ij} are given by

$$a_{ij} = \begin{cases} 1 & \text{if } \langle v_i, v_j \rangle \in E \\ 0 & \text{otherwise} \end{cases}$$

is called the *adjacency matrix* of the graph G.

Recall that the adjacency matrix is the same as the relation matrix or the incidence matrix of the relation E in V. Any element of the adjacency matrix is either 0 or 1. Any matrix whose elements are either 0 or 1 is called a *bit matrix*

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

length from v_i to v_j. In order to decide this, with the help of the adjacency matrix we would have to consider all possible A^r for $r = 1, 2, \ldots$. This method is neither practical nor necessary, as we shall show.

Recall that in a simple digraph with n nodes, the length of an elementary path or cycle does not exceed n (see Theorem 5-1.1). Also for a path between any two nodes one can obtain an elementary path by deleting certain parts of the path which are cycles. Similarly (for cycles), we can always obtain an elementary cycle from a given cycle. If we are interested in determining whether there exists a path from v_i to v_j, all we need to examine are the elementary paths of length less than or equal to $n - 1$. In the case of $v_i = v_j$ and the path is a cycle, we need to examine all possible elementary cycles of length less than or equal to n. Such cycles or paths are easily determined from the matrix B_n where

$$B_n = A + A^2 + A^3 + \cdots + A^n$$

The element in the ith row and jth column of B_n shows the number of paths of length n or less which exist from v_i to v_j. If this element is nonzero, then it is clear that v_j is reachable from v_i. Of course, in order to determine reachability, we need to know the existence of a path, and not the number of paths between any two nodes. In any case, the matrix B_n furnishes the required information about the reachability of any node of the graph from any other node.

Definition 5-1.13 Let $G = \langle V, E \rangle$ be a simple digraph in which $| V | = n$ and the nodes of G are assumed to be ordered. An $n \times n$ matrix P whose elements are given by

$$p_{ij} = \begin{cases} 1 & \text{if there exists a path from } v_i \text{ to } v_j \\ 0 & \text{otherwise} \end{cases}$$

is called the *path matrix* (*reachability matrix*) of the graph G.

Note that the path matrix only shows the presence or absence of at least one path between a pair of points and also the presence or absence of a cycle at any node. It does not, however, show all the paths that may exist. In this sense a path matrix does not give complete information about a graph as does the adjacency matrix. The path matrix is important in its own right.

The path matrix can be calculated from the matrix B_n by choosing $p_{ij} = 1$ if the element in the ith row and jth column of B_n is nonzero and $p_{ij} = 0$ otherwise. We shall apply this method of calculating the path matrix to our sample problem, whose graph is given in Fig. 5-1.15. The adjacency matrix $A = A_1$ and its powers A^2, A^3, A^4 have already been calculated. We thus have B_4 and the path matrix P given by

$$B_4 = \begin{pmatrix} 3 & 4 & 2 & 3 \\ 5 & 5 & 4 & 6 \\ 7 & 7 & 4 & 7 \\ 3 & 2 & 1 & 2 \end{pmatrix} \qquad P = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

matrix is given by

$$
\begin{aligned}
d_{ij} &= \infty && \text{if } \langle v_i, v_j \rangle \notin E \\
d_{ii} &= 0 && \text{for all } i = 1, 2, \ldots, n \\
d_{ij} &= k && \text{where } k \text{ is the smallest integer for which } a_{ij}^{(k)} \neq 0
\end{aligned}
$$

Determine the distance matrix of the digraph given in Fig. 5-1.17. What does $d_{ij} = 1$ mean?

7 Show that a digraph G is strongly connected if all the entries of the distance matrix except the diagonal entries are nonzero. How will you obtain the path matrix from a distance matrix? How will you modify the diagonal entries?

8 Modify algorithm *MINIMA* so that all minimal paths are computed.

5-1.4 Trees

An important class of diagraphs called directed trees will be introduced in this section along with the terminology associated with such trees. Trees are useful in describing any structure which involves hierarchy. Familiar examples of such structures are family trees, the decimal classification of books in a library, the hierarchy of positions in an organization, an algebraic expression involving operations for which certain rules of precedence are prescribed, etc. We shall describe here how trees can be represented by diagrams and other means. Representation of trees in a computer is discussed in Sec. 5-2.1. Applications of trees to grammars is given in Sec. 5-3.1.

Definition 5-1.14 A *directed tree* is an acyclic digraph which has one node called its *root* with indegree 0, while all other nodes have indegree 1.

Note that every directed tree must have at least one node. An isolated node is also a directed tree.

Definition 5-1.15 In a directed tree, any node which has outdegree 0 is called a *terminal node* or a *leaf*; all other nodes are called *branch nodes*. The *level* of any node is the length of its path from the root.

The level of the root of a directed tree is 0, while the level of any node is equal to its distance from the root. Observe that all the paths in a directed tree are elementary, and the length of a path from any node to another node, if such a path exists, is the distance between the nodes, because a directed tree is acyclic.

Figure 5-1.18 shows three different diagrams of a directed tree. Several other diagrams of the same tree can be drawn by choosing different relative positions of the nodes with respect to its root. The directed tree of our example has two nodes at level 1, five nodes at level 2, and three nodes at level 3. Figure 5-1.18*a* shows a natural way of representation, viz., the way a tree grows from its root up and ending in leaves at different levels. Figure 5-1.18*b* shows the same tree drawn upside down. This is a convenient way of drawing a directed tree and is commonly used in the literature. Figure 5-1.18*c* differs from *b* in the order in which the nodes appear at any level from left to right. According to our defini-

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

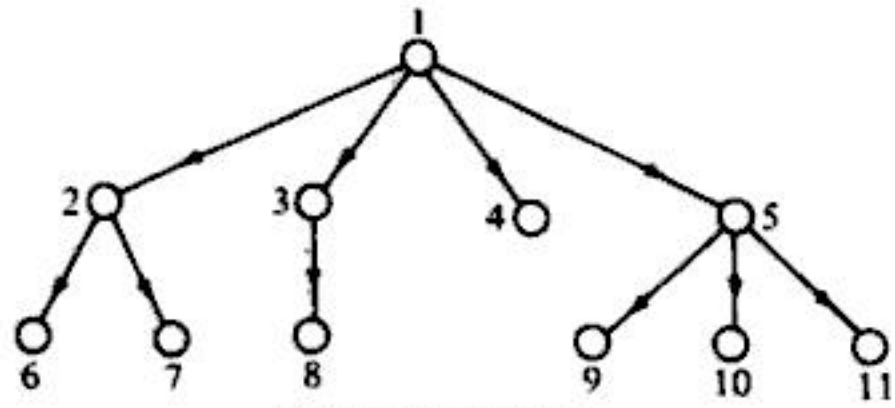
You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

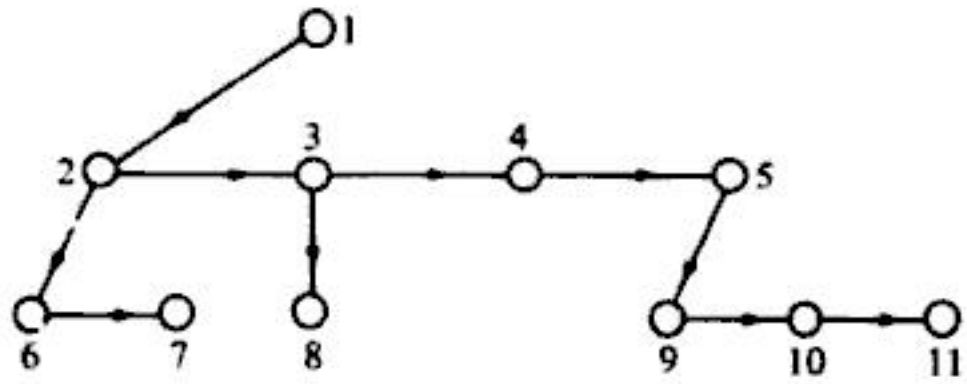
represented by a string over the alphabet {0, 1}, the root being represented by an empty string. Any son of a node u has a string which is prefixed by the string of u. The string of any terminal node is not prefixed to the string of any other node. The set of strings which correspond to terminal nodes form a *prefix* code. Thus the prefix code of the binary tree in Fig. 5-1.21*b* is {000, 001, 01, 10, 110, 111}. A similar representation of nodes of a positional m-ary tree by means of strings over an alphabet $\{0, 1, \ldots, m-1\}$ is possible.

The string representation of the nodes of a positional binary tree immediately suggests a natural method of representing a binary tree in a computer. It is sufficient for our purpose at this stage simply to recognize that such a natural representation exists.

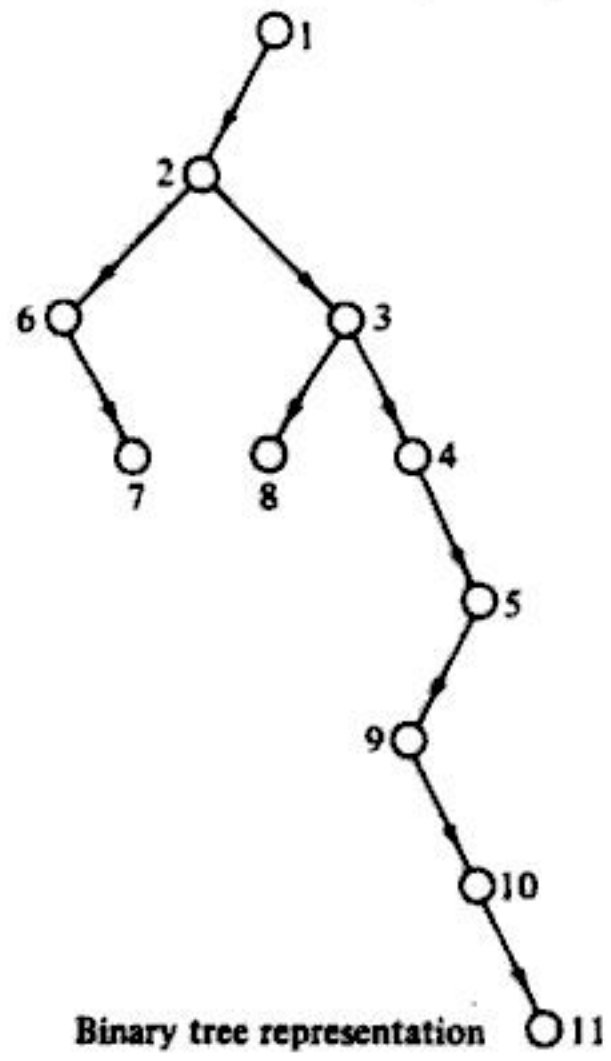
Binary trees are useful in several applications. We shall now show that every tree can be uniquely represented by a binary tree, so that for the computer representation of a tree it is possible to consider the representation of its



Given directed tree



Stage 1



Binary tree representation

FIGURE 5-1.22

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

If a particular subtree is empty (i.e., when a node has no left or right descendant), the traversal is performed by doing nothing. In other words, a null subtree is considered to be fully traversed when it is encountered.

If the words "left" and "right" are interchanged in the preceding definitions, then we have three new traversal methods which are called *converse preorder*, *converse inorder*, and *converse postorder*, respectively.

The preorder, inorder, and postorder traversals of the tree given in Fig. 5-2.1 will process the nodes in the following order:

ABCDEFGH (preorder)
CBDAEGHF (inorder)
CDBHGFEA (postorder)

(The respective converse traversals would be *AEFGHBDC*, *FHGEADBC*, and *HGFEDCBA*.)

Although recursive algorithms would probably be the simplest to write for the traversals of binary trees, we will formulate algorithms which are nonrecursive. Since in traversing a tree it is required to descend and subsequently ascend parts of the tree, pointer information which will permit movement up the tree must be temporarily stored. Observe that the structural information that is already present in the tree permits downward movement from the root of the tree. Because movement up the tree must be made in a reverse manner from that taken in descending the tree, a stack is required to save pointer values as the tree is traversed. We will now give an algorithm for traversing a tree in preorder.

Algorithm *PREORDER* Given a binary tree whose root node address is given by a variable T and whose node structure is the same as previously described, this algorithm traverses the tree in preorder. An auxiliary stack S is used, and TOP is the index of the top element of S. P is a temporary variable which denotes where we are in the tree.

1 [Initialize] If $T = NULL$, then Exit (the tree has no root and therefore is not a proper binary tree); otherwise set $P \leftarrow T$ and $TOP \leftarrow 0$.

2 [Visit node, stack right branch address, and go left] Process node P. If $RLINK(P) \neq NULL$, then set $TOP \leftarrow TOP + 1$ and $S[TOP] \leftarrow RLINK(P)$. Set $P \leftarrow LLINK(P)$.

3 [End of chain?] If $P \neq NULL$, then go to step 2.

4 [Unstack a right branch address] If $TOP = 0$, then Exit; otherwise set $P \leftarrow S[TOP]$, $TOP \leftarrow TOP - 1$, and go to step 2.

In the second and third steps of the algorithm, we visit and process a node. The address of the right branch of such a node, if it exists, is stacked, and a chain of left branches is followed until this chain ends. At this point, we enter step 4 and delete from the stack the address of the root node of the most recently encountered right subtree and process it according to steps 2 and 3. A trace of the algorithm for the binary tree given in Fig. 5-2.1 appears in Table 5-2.1, where the rightmost element in the stack is considered to be its top element and the

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

in preorder, while the suffix form is generated by an inorder traversal of the binary tree (not a postorder traversal!).

As an application of binary trees, we will formulate an algorithm that will maintain a tree-structured symbol table. (See Sec. 2-4.6.) One of the criteria that a symbol table routine must meet is that table searching be performed efficiently. This requirement originates in the compilation phase where many references to the entries of a symbol table are made. The two required operations that must be performed on a symbol table are insertion and "look-up," each of which involves searching. A binary tree structure was chosen for two reasons. The first reason is because if the symbol entries as encountered are randomly distributed according to lexicographic order, then table searching becomes approximately equivalent to a binary search as long as the tree is maintained in lexicographic order. Second, a binary tree structure is easily maintained in lexicographic order (in the sense that only a few pointers need be changed).

For simplicity, we assume a relatively sophisticated system which allows variable-length character strings to be used without much effort on the part of the programmer to handle them. We further assume that the symbol table routine is used to create trees that are local to a block of program code. This implies that an attempt to insert a duplicate entry is an error. In a global context, duplicate entries would be permitted as long as they were at different block levels. In a sense, the symbol table is a set of trees, one for each block level.

A binary tree will be constructed whose typical node is of the form

LLINK	*SYMBOLS*	*INFO*	*RLINK*

where *LLINK* and *RLINK* are pointer fields, *SYMBOLS* is the field for the character string which is the identifier or variable name (note that string descriptors might well be used here to allow fixed-length nodes, but it is assumed that this use is clear to the user), and *INFO* is some set of fields containing additional information about the identifier, such as its type. A node will be created by the execution of the statement $P \leftarrow NODE$ where the address of the new node is stored in P.

Finally, it is assumed that prior to any use of the symbol table routine at a particular block level, the appropriate tree head node is created with the *SYMBOLS* field set to a value that is greater lexicographically than any valid identifier. $HEAD[n]$ will point to this node where n designates the nth block level. Hence, the existence of an appropriate main routine which administers to the creation of tree heads as a new block is entered and to the deletion of tree heads as a block is exited, is assumed.

Because both the insertion and look-up operations involve many of the same actions (e.g., searching), we will actually produce only one routine, *TABLE*, and distinguish between insertion and look-up by the value of a global logical variable, *FLAG*. On invoking algorithm *TABLE*, if *FLAG* is *true*, then the requested operation is insertion; *NAME* and *DATA* contain the identifier name and additional information respectively. If the insertion is successful, then *FLAG* retains its original value; otherwise the value of *FLAG* is negated to indicate an error (because the identifier is already present in the table at that level), and an exit from the algorithm is made. On the other hand, if the algorithm is invoked with *FLAG* set to *false*, then the requested operation is look-up. In this

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

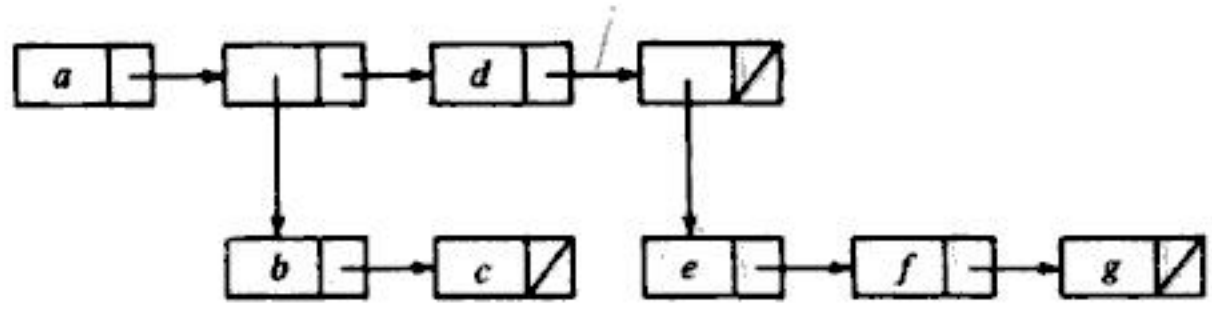


FIGURE 5-2.6 Storage representation of $(a, (b, c), d, (e, f, g))$.

has a depth of 2. The depth of a list is the number of levels it contains. The elements a and d are at level 1, and b and c are at level 2. The number of pairs of parentheses surrounding an element indicates its level. The element d in the list $(a, (b, (c, (d))))$ has a level of 4.

Order and depth are more easily understood in terms of our storage representation, where order is indicated by horizontal arrows and depth by vertical (downward-pointing) arrows. Thus, the list $(a, (b, c), d, (e, f, g))$ would be represented by the storage structure in Fig. 5-2.6. In storage, several lists may share common sublists. For example, the lists $(a, (b, c), d)$ and $(1, 5.2, (b, c))$ could be represented as in Fig. 5-2.7.

The recursive list M, where M is (a, b, M), can be represented as shown in Fig. 5-2.8. We would naturally use this storage representation where the common structure is shared rather than generating an infinite number of nodes to correspond to an infinite graph, but great care must be taken when manipulating such recursive structures in order to avoid programming oneself into an infinite loop.

A list structure occurs quite frequently in the processing of information, although it is not always evident. Consider a simple English sentence which consists of a subject, verb, and object. Any such sentence can be interpreted as a three-element list, whose elements can be atoms (single words) or lists (word phrases). The following sentences and their corresponding list representations are examples:

Man bites dog. = (Man, bites, dog)

The man bites the dog. = ((The, man), bites, (the, dog))

The big man is biting the small dog. = ((The, big, man), (is, biting), (the, small, dog))

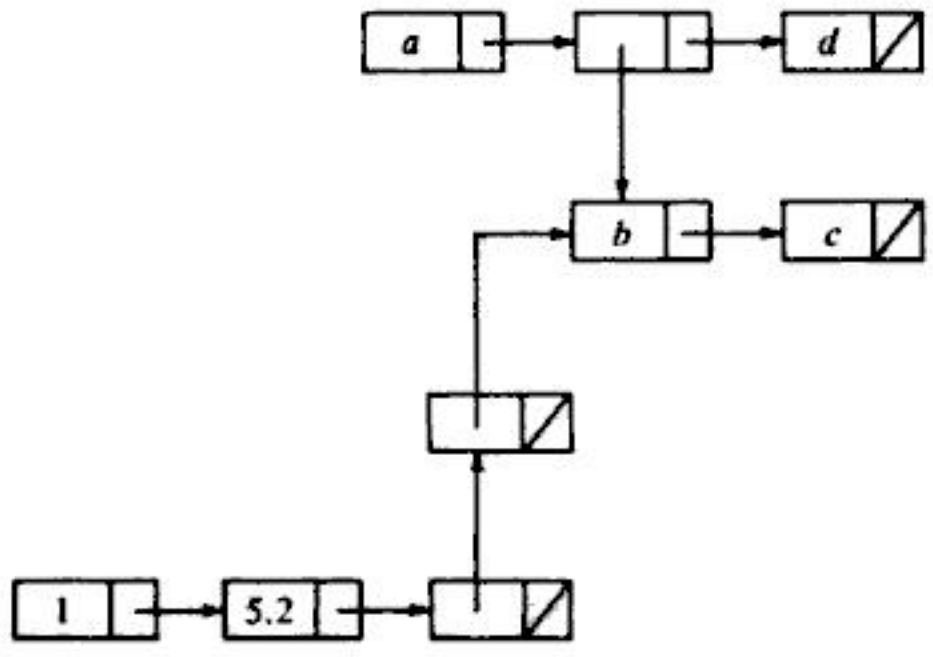


FIGURE 5-2.7 Two lists sharing a common sublist.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

where Φ is

$$\langle \text{identifier} \rangle ::= \langle \text{letter} \rangle \mid \langle \text{identifier} \rangle \langle \text{letter} \rangle \mid \langle \text{identifier} \rangle \langle \text{digit} \rangle$$

$$\langle \text{letter} \rangle ::= a \mid b \mid \cdots \mid y \mid z$$

$$\langle \text{digit} \rangle ::= 0 \mid 1 \mid \cdots \mid 8 \mid 9$$

The existence of the rule $\langle \text{identifier} \rangle ::= \langle \text{letter} \rangle$ does not imply that $\langle \text{letter} \rangle$ is a phrase since we cannot generate $\langle \text{digit} \rangle \langle \text{identifier} \rangle \langle \text{digit} \rangle$ from the starting symbol $\langle \text{identifier} \rangle$. What are the phrases of $\langle \text{letter} \rangle$ 1? A derivation for this sentential form is

$$\langle \text{identifier} \rangle \Rightarrow \langle \text{identifier} \rangle \langle \text{digit} \rangle \Rightarrow \langle \text{letter} \rangle \langle \text{digit} \rangle \Rightarrow \langle \text{letter} \rangle 1$$

Therefore

$$\langle \text{identifier} \rangle \overset{*}{\Rightarrow} \langle \text{letter} \rangle \langle \text{digit} \rangle \qquad \text{and} \qquad \langle \text{digit} \rangle \overset{+}{\Rightarrow} 1$$

Consequently, 1 is a simple phrase. Another derivation for the given sentential form is

$$\langle \text{identifier} \rangle \Rightarrow \langle \text{identifier} \rangle \langle \text{digit} \rangle \Rightarrow \langle \text{identifier} \rangle 1 \Rightarrow \langle \text{letter} \rangle 1$$

where

$$\langle \text{identifier} \rangle \overset{*}{\Rightarrow} \langle \text{identifier} \rangle 1 \qquad \text{and} \qquad \langle \text{identifier} \rangle \overset{+}{\Rightarrow} \langle \text{letter} \rangle$$

Again, the only phrase which is also a simple phrase is $\langle \text{letter} \rangle$.

In the subsequent discussion the leftmost simple phrase of a sentential form will be required. We therefore formulate the following definition.

Definition 5-3.2 The *handle* of a sentential form is its leftmost simple phrase.

In the current example, we have two possible simple phrases, namely, $\langle \text{letter} \rangle$ and 1, and since $\langle \text{letter} \rangle$ is the leftmost phrase, it is also the handle.

As previously discussed in Sec. 3-3.3, syntax trees are an important aid to understanding the syntax of a sentence. A syntax tree for a sentence of some language has a distinguished node called its *root* which is labeled by the starting symbol of the grammar. The leaf nodes of the syntax tree represent the terminal symbols in the sentence being diagramed. All nonleaf nodes correspond to nonterminal symbols. Each nonterminal node has a number of branches emanating downward, each of which represents a symbol in the right side of the production being applied at that point in the syntax tree.

The syntax tree corresponding to the following derivation of the sentence $c1$ in grammar G_1 is given in Fig. 5-3.1.

$$\langle \text{identifier} \rangle \Rightarrow \langle \text{identifier} \rangle \langle \text{digit} \rangle \Rightarrow \langle \text{letter} \rangle \langle \text{digit} \rangle \Rightarrow c \langle \text{digit} \rangle \Rightarrow c1$$

Note that another possible derivation for the same sentence is

$$\langle \text{identifier} \rangle \Rightarrow \langle \text{identifier} \rangle \langle \text{digit} \rangle \Rightarrow \langle \text{identifier} \rangle 1 \Rightarrow \langle \text{letter} \rangle 1 \Rightarrow c1$$

and that this derivation has the same syntax tree as that given in Fig. 5-3.1. Therefore for each syntax tree there exists at least one derivation.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

If $A \to \beta$ is a rule of the grammar and $\phi_2 \in V_T^*$. Again the relation is easily extended to $\overset{+}{\underset{R}{\Rightarrow}}$ and $\overset{*}{\underset{R}{\Rightarrow}}$ where the relation $\psi \overset{+}{\underset{R}{\Rightarrow}} \sigma$ may be read as "ψ right-produces σ." The relation $\overset{+}{\underset{R}{\Rightarrow}}$ specifies that there is exactly one $\underset{R}{\Rightarrow}$ derivation. For example, the leftmost and rightmost derivations for $i + i * i$ in G_4 are

$$\begin{aligned}
\langle \text{expression} \rangle &\underset{L}{\Rightarrow} \langle \text{expression} \rangle + \langle \text{expression} \rangle \underset{L}{\Rightarrow} \langle \text{term} \rangle + \langle \text{expression} \rangle \\
&\underset{L}{\Rightarrow} \langle \text{factor} \rangle + \langle \text{expression} \rangle \underset{L}{\Rightarrow} i + \langle \text{expression} \rangle \\
&\underset{L}{\Rightarrow} i + \langle \text{term} \rangle \underset{L}{\Rightarrow} i + \langle \text{term} \rangle * \langle \text{factor} \rangle \\
&\underset{L}{\Rightarrow} i + \langle \text{factor} \rangle * \langle \text{factor} \rangle \underset{L}{\Rightarrow} i + i * \langle \text{factor} \rangle \\
&\underset{L}{\Rightarrow} i + i * i
\end{aligned}$$

and

$$\begin{aligned}
\langle \text{expression} \rangle &\underset{R}{\Rightarrow} \langle \text{expression} \rangle + \langle \text{expression} \rangle \underset{R}{\Rightarrow} \langle \text{expression} \rangle + \langle \text{term} \rangle \\
&\underset{R}{\Rightarrow} \langle \text{expression} \rangle + \langle \text{term} \rangle * \langle \text{factor} \rangle \\
&\underset{R}{\Rightarrow} \langle \text{expression} \rangle + \langle \text{term} \rangle * i \\
&\underset{R}{\Rightarrow} \langle \text{expression} \rangle + \langle \text{factor} \rangle * i \\
&\underset{R}{\Rightarrow} \langle \text{expression} \rangle + i * i \underset{R}{\Rightarrow} \langle \text{term} \rangle + i * i \\
&\underset{R}{\Rightarrow} \langle \text{factor} \rangle + i * i \underset{R}{\Rightarrow} i + i * i
\end{aligned}$$

respectively. The leftmost and the rightmost derivations correspond to a left-to-right and right-to-left top-down parse, respectively.

The general problem of parsing is to start with a string σ of terminal symbols, for example, $i + i * i$, and to find a sequence of productions such that $\langle \text{expression} \rangle \overset{*}{\Rightarrow} \sigma$. The bottom-up method (proceeding from left to right) attacks the problem by first "reducing" the above string to

$$\langle \text{factor} \rangle + i * i$$

then reducing this string to

$$\langle \text{term} \rangle + i * i$$

and then

$$\langle \text{expression} \rangle + i * i$$

and

$$\langle \text{expression} \rangle + \langle \text{factor} \rangle * i$$

etc., where a reduction is the opposite of a production. This process continues until everything is reduced to $\langle \text{expression} \rangle$ or until it is shown that it cannot be done. Note that the sequence of reductions is the reverse of the right canonical derivation. In general, bottom-up parsing from left to right proceeds by right reductions.

In a left-to-right bottom-up parse, the handle is to be reduced at each step in the parse. The questions which arise are, How do we find the handle of a sentential form and to what do we reduce it?

One obvious approach is merely to select one of the possible alternatives. If a mistake is subsequently detected, then we must retrace our steps to the location of the error and try some other alternative. This process is called "back-up," and it can be very time-consuming.

A more efficient solution involves looking at the context around the sub-

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

$A \rightarrow x$

$B \rightarrow Ax)$

Using only syntax trees, obtain as many of the precedence relations as possible.

6 Using the formal definition of precedence relations, obtain the precedence matrix for Prob. 5.

7 Prove Theorem 5-3.2.

8 Prove Theorem 5-3.3.

9 Prove Theorem 5-3.4.

10 Prove that $\cdot > = (R^+)^T \circ (\doteq) \circ (L^*)$.

11 Using the parsing algorithm of the text and the grammar of Prob. 5, give a tree of the parse for the strings $y(xx)y$, $(((xx)x)x)y$, $y((xx))x)y$.

12 Formulate an algorithm and write a program that will obtain a precedence matrix.

13 Write a program to use the precedence matrix obtained in Prob. 12 and parse a string according to the parsing algorithm of the text.

14 Can you obtain the precedence functions for the grammar of Prob. 5?

15 Is the following grammar a simple precedence grammar? If not, why not?

$E \rightarrow a$ $\qquad$ $B \rightarrow R$

$E \rightarrow b$ $\qquad$ $B \rightarrow (B)$

$E \rightarrow (E + E)$ $\qquad$ $R \rightarrow E = E$

5-4 FAULT DETECTION IN COMBINATIONAL SWITCHING CIRCUITS

The synthesis and analysis of combinational switching circuits were discussed in the previous chapter. In this section, we discuss a different but equally important problem, fault detection and fault diagnosis. Fault detection is concerned with determining whether a fault exists in a circuit. In fault diagnosis, one tries to locate a specific fault in a system.

The diagnosis of faults in modern computing systems is becoming more important as the complexity of such systems increases. Rapid and effective diagnosis of faults in computer systems is desirable since once these diagnoses are in service, down time can be kept at a minimum.

The detection and diagnosis of faults in early computers were performed by technicians. Although these computers were physically very large, the number of components in them was rather small as compared to present computers, and a skillful engineer could locate a fault in a reasonable period of time. With the advent of transistors and integrated circuits, the number of components in a modern computer has greatly increased and fault detection and diagnosis have become increasingly difficult tasks.

The logical conclusion is that if a computer is at all operational, it should diagnose itself, or at worst another computer should be used to perform the diagnosis.

In this section we shall consider a combinational circuit diagram to be a directed graph whose nodes are the gates and whose edges are the connections between gates. From this representation an algorithm will be given to generate a fault table. The purpose of such a table is to be able to detect and more generally diagnose faults. The algorithm will use certain notions of relations such as their

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

ALSO FROM TATA McGRAW-HILL

ANTIA :	**NUMERICAL METHODS FOR SCIENTISTS AND ENGINEERS**
DAS :	**FUNDAMENTALS OF MATHEMATICAL ANALYSIS**
CODDINGTON :	**THEORY OF ORDINARY DIFFERENTIAL EQUATIONS**
RAO :	**LINEAR ALGEBRA**

The McGraw-Hill Companies

Tata McGraw-Hill
Publishing Company Limited
7 West Patel Nagar, New Delhi 110 008

Visit our website at : www.tatamcgrawhill.com

ISBN-13: 978-0-07-463113-3
ISBN-10: 0-07-463113-6

9 780074 631133