an identifier starts with a letter or an underscore (_) that is followed by one or more lowercase letters, uppercase letters, underscores, and digits.

Several extensions to Backus–Naur form are commonly used to define phrase-structure grammars. In one such extension, a question mark (?) indicates that the symbol, or group of symbols inside parentheses, to its left can appear zero or once (that is, it is optional), an asterisk (*) indicates that the symbol to its left can appear zero or more times, and a plus (+) indicates that the symbol to its left can appear one or more times. These extensions are part of **extended Backus–Naur form (EBNF),** and the symbols ?, *, and + are called **metacharacters.** In EBNF the brackets used to denote nonterminals are usually not shown.

34. Describe the set of strings defined by each of these sets of productions in EBNF.

a) $string ::= L{+}D?L{+}$
$L ::= a \mid b \mid c$
$D ::= 0 \mid 1$

b) $string ::= sign\ D{+} \mid D{+}$
$sign ::= + \mid -$
$D ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

c) $string ::= L{*}(D{+})?L{*}$
$L ::= x \mid y$
$D ::= 0 \mid 1$

35. Give production rules in extended Backus–Naur form that generate all decimal numerals consisting of an optional sign, a nonnegative integer, and a decimal fraction that is either the empty string or a decimal point followed by an optional positive integer optionally preceded by some number of zeros.

36. Give production rules in extended Backus–Naur form that generate a sandwich if a sandwich consists of a lower slice of bread; mustard or mayonnaise; optional lettuce; an optional slice of tomato; one or more slices of either turkey, chicken, or roast beef (in any combination); optionally some number of slices of cheese; and a top slice of bread.

37. Give production rules in extended Backus–Naur form for identifiers in the C programming language (see Exercise 33).

38. Describe how productions for a grammar in extended Backus–Naur form can be translated into a set of productions for the grammar in Backus–Naur form.

This is the Backus–Naur form that describes the syntax of expressions in postfix (or reverse Polish) notation.

$\langle expression\rangle ::= \langle term\rangle \mid \langle term\rangle\langle term\rangle\langle addOperator\rangle$
$\langle addOperator\rangle ::= + \mid -$
$\langle term\rangle ::= \langle factor\rangle \mid \langle factor\rangle\langle factor\rangle\langle mulOperator\rangle$
$\langle mulOperator\rangle ::= * \mid /$
$\langle factor\rangle ::= \langle identifier\rangle \mid \langle expression\rangle$
$\langle identifier\rangle ::= a \mid b \mid \cdots \mid z$

39. For each of these strings, determine whether it is generated by the grammar given for postfix notation. If it is, find the steps used to generate the string

a) $abc{*}{+}$ **b)** $xy{+}{+}$ **c)** $xy{-}z{*}$
d) $wxyz{-}{*}/$ **e)** $ade{-}{*}$

40. Use Backus–Naur form to describe the syntax of expressions in infix notation, where the set of operators and identifiers is the same as in the BNF for postfix expressions given in the preamble to Exercise 39, but parentheses must surround expressions being used as factors.

41. For each of these strings, determine whether it is generated by the grammar for infix expressions from Exercise 40. If it is, find the steps used to generate the string.

a) $x + y + z$ **b)** $a/b + c/d$
c) $m * (n + p)$ **d)** $+m - n + p - q$
e) $(m + n) * (p - q)$

42. Let G be a grammar and let R be the relation containing the ordered pair (w_0, w_1) if and only if w_1 is directly derivable from w_0 in G. What is the reflexive transitive closure of R?

12.2 Finite-State Machines with Output

Introduction

Links

Many kinds of machines, including components in computers, can be modeled using a structure called a finite-state machine. Several types of finite-state machines are commonly used in models. All these versions of finite-state machines include a finite set of states, with a designated starting state, an input alphabet, and a transition function that assigns a next state to every state and input pair. Finite-state machines are used extensively in applications in computer science and data networking. For example, finite-state machines are the basis for programs for spell checking, grammar checking, indexing or searching large bodies of text, recognizing speech, transforming text using markup languages such as XML and HTML, and network protocols that specify how computers communicate.

12-13

TABLE 1 State Table for a Vending Machine.

	Next State					**Output**				
	Input					***Input***				
State	**5**	**10**	**25**	***O***	***R***	**5**	**10**	**25**	***O***	***R***
s_0	s_1	s_2	s_5	s_0	s_0	n	n	n	n	n
s_1	s_2	s_3	s_6	s_1	s_1	n	n	n	n	n
s_2	s_3	s_4	s_6	s_2	s_2	n	n	5	n	n
s_3	s_4	s_5	s_6	s_3	s_3	n	n	10	n	n
s_4	s_5	s_6	s_6	s_4	s_4	n	n	15	n	n
s_5	s_6	s_6	s_6	s_5	s_5	n	5	20	n	n
s_6	s_6	s_6	s_6	s_0	s_0	5	10	25	OJ	AJ

In this section, we will study those finite-state machines that produce output. We will show how finite-state machines can be used to model a vending machine, a machine that delays input, a machine that adds integers, and a machine that determines whether a bit string contains a specified pattern.

Before giving formal definitions, we will show how a vending machine can be modeled. A vending machine accepts nickels (5 cents), dimes (10 cents), and quarters (25 cents). When a total of 30 cents or more has been deposited, the machine immediately returns the amount in excess of 30 cents. When 30 cents has been deposited and any excess refunded, the customer can push an orange button and receive an orange juice or push a red button and receive an apple juice. We can describe how the machine works by specifying its states, how it changes states when input is received, and the output that is produced for every combination of input and current state.

The machine can be in any of seven different states s_i, $i = 0, 1, 2, \ldots, 6$, where s_i is the state where the machine has collected $5i$ cents. The machine starts in state s_0, with 0 cents received. The possible inputs are 5 cents, 10 cents, 25 cents, the orange button (O), and the red button (R). The possible outputs are nothing (n), 5 cents, 10 cents, 15 cents, 20 cents, 25 cents, an orange juice, and an apple juice.

We illustrate how this model of the machine works with this example. Suppose that a student puts in a dime followed by a quarter, receives 5 cents back, and then pushes the orange button for an orange juice. The machine starts in state s_0. The first input is 10 cents, which changes

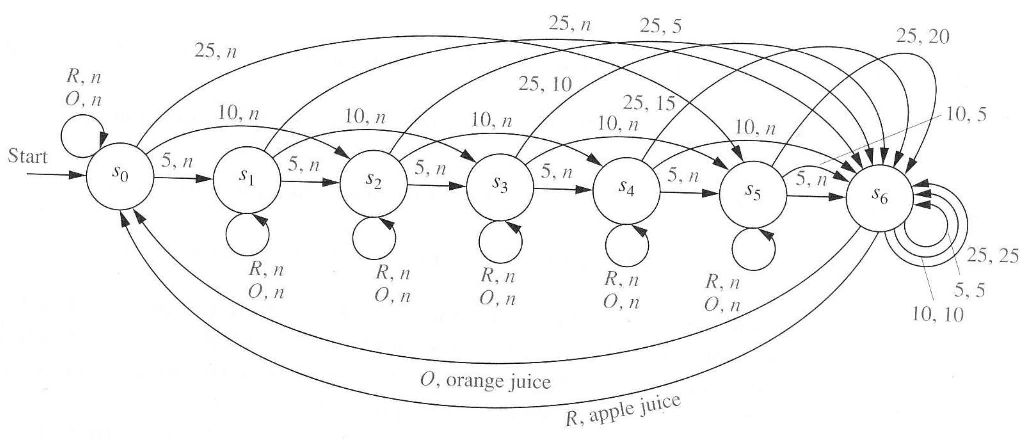

FIGURE 1 A Vending Machine.

the state of the machine to s_2 and gives no output. The second input is 25 cents. This changes the state from s_2 to s_6, and gives 5 cents as output. The next input is the orange button, which changes the state from s_6 back to s_0 (because the machine returns to the start state) and gives an orange juice as its output.

We can display all the state changes and output of this machine in a table. To do this we need to specify for each combination of state and input the next state and the output obtained. Table 1 shows the transitions and outputs for each pair of a state and an input.

Another way to show the actions of a machine is to use a directed graph with labeled edges, where each state is represented by a circle, edges represent the transitions, and edges are labeled with the input and the output for that transition. Figure 1 shows such a directed graph for the vending machine.

Finite-State Machines with Outputs

We will now give the formal definition of a finite-state machine with output.

DEFINITION 1 A *finite-state machine* $M = (S, I, O, f, g, s_0)$ consists of a finite set S of *states,* a finite *input alphabet* I, a finite *output alphabet* O, a *transition function* f that assigns to each state and input pair a new state, an *output function* g that assigns to each state and input pair an output, and an initial state s_0.

Let $M = (S, I, O, f, g, s_0)$ be a finite-state machine. We can use a **state table** to represent the values of the transition function f and the output function g for all pairs of states and input. We previously constructed a state table for the vending machine discussed in the introduction to this section.

EXAMPLE 1 The state table shown in Table 2 describes a finite-state machine with $S = \{s_0, s_1, s_2, s_3\}$, $I = \{0, 1\}$, and $O = \{0, 1\}$. The values of the transition function f are displayed in the first two columns, and the values of the output function g are displayed in the last two columns. ◀

Another way to represent a finite-state machine is to use a **state diagram,** which is a directed graph with labeled edges. In this diagram, each state is represented by a circle. Arrows labeled with the input and output pair are shown for each transition.

EXAMPLE 2 Construct the state diagram for the finite-state machine with the state table shown in Table 2.

Solution: The state diagram for this machine is shown in Figure 2. ◀

EXAMPLE 3 Construct the state table for the finite-state machine with the state diagram shown in Figure 3.

Solution: The state table for this machine is shown in Table 3. ◀

An input string takes the starting state through a sequence of states, as determined by the transition function. As we read the input string symbol by symbol (from left to right), each input symbol takes the machine from one state to another. Because each transition produces an output, an input string also produces an output string.

Suppose that the input string is $x = x_1x_2 \ldots x_k$. Then, reading this input takes the machine from state s_0 to state s_1, where $s_1 = f(s_0, x_1)$, then to state s_2, where $s_2 = f(s_1, x_2)$, and so on,

TABLE 2

	f		g	
	Input		*Input*	
State	**0**	**1**	**0**	**1**
s_0	s_1	s_0	1	0
s_1	s_3	s_0	1	1
s_2	s_1	s_2	0	1
s_3	s_2	s_1	0	0

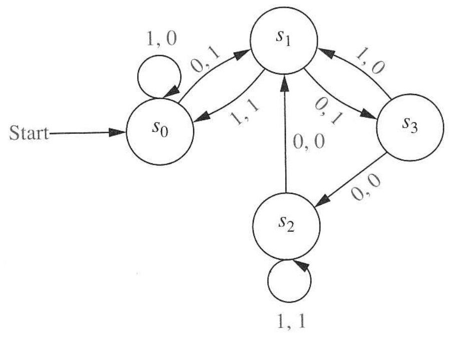

FIGURE 2 **The State Diagram for the Finite-State Machine Shown in Table 2.**

with $s_j = f(s_{j-1}, x_j)$ for $j = 1, 2, \ldots, k$, ending at state $s_k = f(s_{k-1}, x_k)$. This sequence of transitions produces an output string $y_1 y_2 \ldots y_k$, where $y_1 = g(s_0, x_1)$ is the output corresponding to the transition from s_0 to s_1, $y_2 = g(s_1, x_2)$ is the output corresponding to the transition from s_1 to s_2, and so on. In general, $y_j = g(s_{j-1}, x_j)$ for $j = 1, 2, \ldots, k$. Hence, we can extend the definition of the output function g to input strings so that $g(x) = y$, where y is the output corresponding to the input string x. This notation is useful in many applications.

EXAMPLE 4 Find the output string generated by the finite-state machine in Figure 3 if the input string is 101011.

Solution: The output obtained is 001000. The successive states and outputs are shown in Table 4. ◀

We can now look at some examples of useful finite-state machines. Examples 5, 6, and 7 illustrate that the states of a finite-state machine give it limited memory capabilities. The states can be used to remember the properties of the symbols that have been read by the machine. However, because there are only finitely many different states, finite-state machines cannot be used for some important purposes. This will be illustrated in Section 12.4.

EXAMPLE 5 An important element in many electronic devices is a *unit-delay machine,* which produces as output the input string delayed by a specified amount of time. How can a finite-state machine be constructed that delays an input string by one unit of time, that is, produces as output the bit string $0x_1x_2 \ldots x_{k-1}$ given the input bit string $x_1x_2 \ldots x_k$?

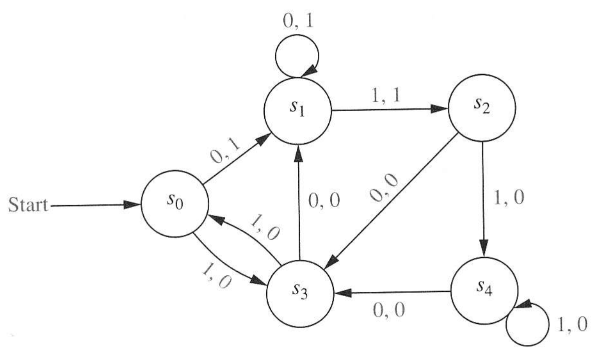

FIGURE 3 **A Finite-State Machine.**

TABLE 3

	f		g	
	Input		*Input*	
State	**0**	**1**	**0**	**1**
s_0	s_1	s_3	1	0
s_1	s_1	s_2	1	1
s_2	s_3	s_4	0	0
s_3	s_1	s_0	0	0
s_4	s_3	s_4	0	0

TABLE 4

Input	1	0	1	0	1	1	—
State	s_0	s_3	s_1	s_2	s_3	s_0	s_3
Output	0	0	1	0	0	0	—

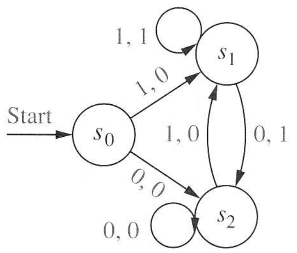

FIGURE 4 **A Unit-Delay Machine.**

Solution: A delay machine can be constructed that has two possible inputs, namely, 0 and 1. The machine must have a start state s_0. Because the machine has to remember whether the previous input was a 0 or a 1, two other states s_1 and s_2 are needed, where the machine is in state s_1 if the previous input was 1 and in state s_2 if the previous input was 0. An output of 0 is produced for the initial transition from s_0. Each transition from s_1 gives an output of 1, and each transition from s_2 gives an output of 0. The output corresponding to the input of a string $x_1 \dots x_k$ is the string that begins with 0, followed by x_1, followed by $x_2, \dots,$ ending with x_{k-1}. The state diagram for this machine is shown in Figure 4. ◀

EXAMPLE 6 Produce a finite-state machine that adds two integers using their binary expansions.

Solution: When $(x_n \dots x_1x_0)_2$ and $(y_n \dots y_1y_0)_2$ are added, the following procedure (as described in Section 3.6) is followed. First, the bits x_0 and y_0 are added, producing a sum bit z_0 and a carry bit c_0. This carry bit is either 0 or 1. Then, the bits x_1 and y_1 are added, together with the carry c_0. This gives a sum bit z_1 and a carry bit c_1. This procedure is continued until the nth stage, where x_n, y_n, and the previous carry c_{n-1} are added to produce the sum bit z_n and the carry bit c_n, which is equal to the sum bit z_{n-1}.

A finite-state machine to carry out this addition can be constructed using just two states. For simplicity we assume that both the initial bits x_n and y_n are 0 (otherwise we have to make special arrangements concerning the sum bit z_{n+1}). The start state s_0 is used to remember that the previous carry is 0 (or for the addition of the rightmost bits). The other state, s_1, is used to remember that the previous carry is 1.

Because the inputs to the machine are pairs of bits, there are four possible inputs. We represent these possibilities by 00 (when both bits are 0), 01 (when the first bit is 0 and the second is 1), 10 (when the first bit is 1 and the second is 0), and 11 (when both bits are 1). The transitions and the outputs are constructed from the sum of the two bits represented by the input and the carry represented by the state. For instance, when the machine is in state s_1 and receives 01 as input, the next state is s_1 and the output is 0, because the sum that arises is $0 + 1 + 1 = (10)_2$. The state diagram for this machine is shown in Figure 5. ◀

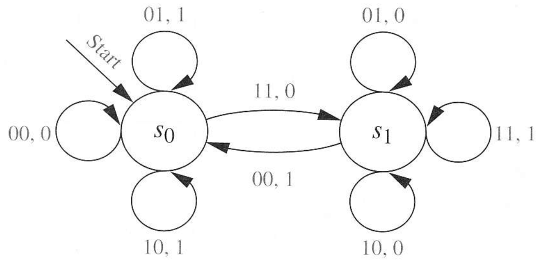

FIGURE 5 **A Finite-State Machine for Addition.**

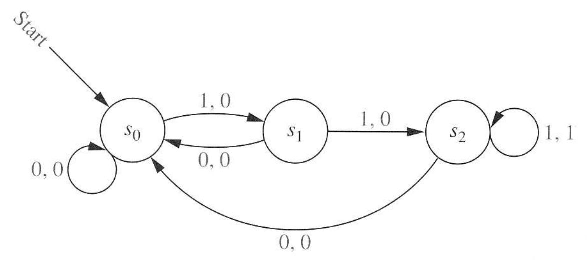

FIGURE 6 A Finite-State Machine That Gives an Output of 1 If and Only If the Input String Read So Far Ends with 111.

EXAMPLE 7 In a certain coding scheme, when three consecutive 1s appear in a message, the receiver of the message knows that there has been a transmission error. Construct a finite-state machine that gives a 1 as its current output bit if and only if the last three bits received are all 1s.

Solution: Three states are needed in this machine. The start state s_0 remembers that the previous input value, if it exists, was not a 1. The state s_1 remembers that the previous input was a 1, but the input before the previous input, if it exists, was not a 1. The state s_2 remembers that the previous two inputs were 1s.

An input of 1 takes s_0 to s_1, because now a 1, and not two consecutive 1s, has been read; it takes s_1 to s_2, because now two consecutive 1s have been read; and it takes s_2 to itself, because at least two consecutive 1s have been read. An input of 0 takes every state to s_0, because this breaks up any string of consecutive 1s. The output for the transition from s_2 to itself when a 1 is read is 1, because this combination of input and state shows that three consecutive 1s have been read. All other outputs are 0. The state diagram of this machine is shown in Figure 6. ◀

The final output bit of the finite-state machine we constructed in Example 7 is 1 if and only if the input string ends with 111. Because of this, we say that this finite-state machine **recognizes** the set of bit strings that end with 111. This leads us to Definition 2.

DEFINITION 2 Let $M = (S, I, O, f, g, s_0)$ be a finite-state machine and $L \subseteq I^*$. We say that M *recognizes* (or *accepts*) L if an input string x belongs to L if and only if the last output bit produced by M when given x as input is a 1.

TYPES OF FINITE-STATE MACHINES Many different kinds of finite-state machines have been developed to model computing machines. In this section we have given a definition of one type of finite-state machine. In the type of machine introduced in this section, outputs correspond to transitions between states. Machines of this type are known as **Mealy machines,** because they were first studied by G. H. Mealy in 1955. There is another important type of finite-state machine with output, where the output is determined only by the state. This type of finite-state machine is known as a **Moore machine,** because E. F. Moore introduced this type of machine in 1956. Moore machines are considered in a sequence of exercises at the end of this section.

In Example 7 we showed how a Mealy machine can be used for language recognition. However, another type of finite-state machine, giving no output, is usually used for this purpose. Finite-state machines with no output, also known as finite-state automata, have a set of final states and recognize a string if and only if it takes the start state to a final state. We will study this type of finite-state machine in Section 12.3.