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) \[ \text{string} ::= L \rightarrow D \rightarrow L + \\
    L ::= a | b | c \\
    D ::= 0 | 1 \]
- b) \[ \text{string} ::= \text{sign} D + | D + \\
    \text{sign} ::= + | -
    D ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 \]
- c) \[ \text{string} ::= L x (D + y) \rightarrow L + \\
    L ::= x | y \\
    D ::= 0 | 1 \]

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.

\[
\begin{align*}
\text{(expression)} & ::= (\text{term}) | (\text{term})(\text{term})(\text{addOperator}) \\
\text{(addOperator)} & ::= + | - \\
\text{(term)} & ::= (\text{factor}) | (\text{factor})(\text{factor})(\text{mulOperator}) \\
\text{(mulOperator)} & ::= * | / \\
\text{(factor)} & ::= (\text{identifier})(\text{expression}) \\
\text{(identifier)} & ::= a | b | \cdots | z
\end{align*}
\]

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) \[ a b c + \]
- b) \[ x y + + \]
- c) \[ x y - z * \]
- d) \[ w x y z \rightarrow x / e \]
- e) \[ a d e - w \]

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) \[ m \times (n + p) \]
- d) \[ + m - n + p - q \]
- e) \[ (m + n) \times (p - q) \]

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 \( R \)?
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.

\[(\text{expression}) ::= (\text{term}) \mid (\text{term})(\text{addOperator})\]
\[(\text{addOperator}) ::= + \mid -\]
\[(\text{term}) ::= (\text{factor}) \mid (\text{factor})(\text{mulOperator})\]
\[(\text{mulOperator}) ::= \ast \mid /\]
\[(\text{factor}) ::= (\text{identifier}) \mid (\text{expression})\]
\[(\text{identifier}) ::= a \mid b \mid \ldots \mid x\]

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) \(abc+4\)

b) \(xy+x\)

c) \(xy-x\)

d) \(wxyz+/4\)

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 parenthesis 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) \((m + n) \ast (p - q)\)

42. Let \(G\) be a grammar and let \(R\) be the relation containing the ordered pair \((w_1, w_2)\) if and only if \(w_1\) is directly derivable from \(w_2\) in \(G\). What is the reflexive transitive closure of \(R\)?

---

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 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 \(i\) 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 \((\lambda)\), 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

---

### TABLE 1 State Table for a Vending Machine.

<table>
<thead>
<tr>
<th>State</th>
<th>Next State</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s_0 )</td>
<td>(s_1 )</td>
<td>(s_2 )</td>
<td>(s_3 )</td>
</tr>
<tr>
<td>(s_1 )</td>
<td>(s_2 )</td>
<td>(s_3 )</td>
<td>(s_4 )</td>
</tr>
<tr>
<td>(s_2 )</td>
<td>(s_3 )</td>
<td>(s_4 )</td>
<td>(s_5 )</td>
</tr>
<tr>
<td>(s_3 )</td>
<td>(s_4 )</td>
<td>(s_5 )</td>
<td>(s_6 )</td>
</tr>
<tr>
<td>(s_4 )</td>
<td>(s_5 )</td>
<td>(s_6 )</td>
<td>(O )</td>
</tr>
<tr>
<td>(s_5 )</td>
<td>(s_6 )</td>
<td>(O )</td>
<td>(R )</td>
</tr>
<tr>
<td>(s_6 )</td>
<td>(O )</td>
<td>(R )</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>Next State</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s_0 )</td>
<td>(s_25 )</td>
<td>(O )</td>
<td>(A )</td>
</tr>
<tr>
<td>(s_1 )</td>
<td>(s_25 )</td>
<td>(O )</td>
<td>(A )</td>
</tr>
<tr>
<td>(s_2 )</td>
<td>(s_25 )</td>
<td>(O )</td>
<td>(A )</td>
</tr>
<tr>
<td>(s_3 )</td>
<td>(s_25 )</td>
<td>(O )</td>
<td>(A )</td>
</tr>
<tr>
<td>(s_4 )</td>
<td>(s_25 )</td>
<td>(O )</td>
<td>(A )</td>
</tr>
<tr>
<td>(s_5 )</td>
<td>(s_25 )</td>
<td>(O )</td>
<td>(A )</td>
</tr>
<tr>
<td>(s_6 )</td>
<td>(s_25 )</td>
<td>(O )</td>
<td>(A )</td>
</tr>
</tbody>
</table>

---

### 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 $s_1$, and gives 5 cents as output. The next input is the orange button, which changes the state from $s_1$ 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 $x = x_1 x_2 \ldots x_n$. 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.
ith Outputs

...a finite-state machine with output.

O, f, g, s0) consists of a finite set S of states, a finite input
of a transition function f that assigns to each state and
a state function g that assigns to each state and input pair an output,
the-state machine. We can use a state table to represent the
and the output function g for all pairs of states and input
ble for the vending machine discussed in the introduction
a finite-state machine with S = {s0, s1, s2, s3}, I =
of the transition function f are displayed in the first two
function g are displayed in the last two columns.

-a finite-state machine is to use a state diagram, which is a directed
gram, each state is represented by a circle. Arrows labeled
own for each transition.

finite-state machine with the state table shown in Table 2.

machine is shown in Figure 2.

e-state machine with the state diagram shown in Figure 3.

machine is shown in Table 3.

state through a sequence of states, as determined by the
putting symbol by symbol (from left to right), each input
state to another. Because each transition produces an output,
put string.
x = x11...xk. Then, reading this input takes the machine
f(x0, x1), then to state s2, where s2 = f(x1, x2), and so on,
with xj = f(xj-1, xj) for j = 1, 2, ..., k, ending at state s2 = f(s1, x2). This sequence of
transitions produces an output string y1y2...yk, where y1 = g(s0, x1) is the output corresponding
to the transition from s0 to s1, y2 = g(s1, x2) is the output corresponding to the transition
from s1 to s2, and so on. In general, yj = g(sj-1, xj) for j = 1, 2, ..., k. Hence, we can extend
the definition of the output function g to input strings so that g(s) = y, where y is the output
corresponding to the input string s. 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
110111.

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 0|0|1|2|3|4|5|6|7 given the input bit string 1|2|3|4|5|6|7?

TABLE 2

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>s1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>s2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>s3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

TABLE 3

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>s1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>s2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>s3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>s4</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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

FIGURE 3 A Finite-State Machine.
TABLE 4

<table>
<thead>
<tr>
<th>Input</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>$s_0$</td>
<td>$s_1$</td>
<td>$s_2$</td>
<td>$s_3$</td>
<td>$s_4$</td>
<td>$s_5$</td>
</tr>
<tr>
<td>Output</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**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 $x_1 \ldots x_k$ is the string that begins with 0, followed by $x_1$, followed by $x_2$, ..., 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 $(x_n \ldots x_0)$ and $(y_n \ldots y_0)$ 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 $s_0$ 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). The state diagram for this machine is shown in Figure 5.

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

**FIGURE 5** A Finite-State Machine for Addition.
FIGURE 4 A Unit-Delay Machine.

that has two possible inputs, namely, 0 and 1. The machine has to remember whether the previous input was 0. An output of 0 is produced for the 1 and 1 gives an output of 1, and each transition from one state to the next state is determined by the input. The state diagram for this machine is shown in Figure 4.

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, s0, remembers that the previous input value, if it exists, was not a 1. The state, s1, remembers that the previous input was a 1, but the input before the previous input, if it exists, was not a 1. The state, s2, remembers that all the previous two inputs were 1s.

An input of 0 takes s0 to s1, because now a 1, and not two consecutive 1s, has been read; it takes s1 to s2, because now two consecutive 1s have been read; and it takes s2 to itself, because at least two consecutive 1s have been read. An input of 1 takes every state to s0, because this breaks up any string of consecutive 1s. The output for the transition from s2 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 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.
Exercises

1. Draw the state diagrams for the finite-state machines with these state tables.

   a)
   
<table>
<thead>
<tr>
<th>State</th>
<th>( f )</th>
<th>( g )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_0 )</td>
<td>( s_1 )</td>
<td>( s_0 )</td>
</tr>
<tr>
<td>( s_1 )</td>
<td>( s_2 )</td>
<td>( s_0 )</td>
</tr>
<tr>
<td>( s_2 )</td>
<td>( s_3 )</td>
<td>( s_1 )</td>
</tr>
</tbody>
</table>

   b)
   
<table>
<thead>
<tr>
<th>State</th>
<th>( f )</th>
<th>( g )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_0 )</td>
<td>( s_1 )</td>
<td>( s_4 )</td>
</tr>
<tr>
<td>( s_1 )</td>
<td>( s_2 )</td>
<td>( s_0 )</td>
</tr>
<tr>
<td>( s_2 )</td>
<td>( s_3 )</td>
<td>( s_2 )</td>
</tr>
<tr>
<td>( s_3 )</td>
<td>( s_4 )</td>
<td>( s_3 )</td>
</tr>
</tbody>
</table>

2. Give the state tables for the finite-state machines with these state diagrams.

   a)
   
<table>
<thead>
<tr>
<th>State</th>
<th>( f )</th>
<th>( g )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_0 )</td>
<td>( s_1 )</td>
<td>( s_0 )</td>
</tr>
<tr>
<td>( s_1 )</td>
<td>( s_2 )</td>
<td>( s_3 )</td>
</tr>
<tr>
<td>( s_2 )</td>
<td>( s_0 )</td>
<td>( s_0 )</td>
</tr>
<tr>
<td>( s_3 )</td>
<td>( s_1 )</td>
<td>( s_1 )</td>
</tr>
</tbody>
</table>

3. Find the output generated from the input string 01110 for the finite-state machine with the state table in Exercise 1(a).
   a) Exercise 1(a).
   b) Exercise 1(b).
   c) Exercise 1(c).

4. Find the output generated from the input string 10001 for the finite-state machine with the state diagram in a) Exercise 2(a).
   b) Exercise 2(b).
   c) Exercise 2(c).

5. Find the output for each of these input strings when given as input to the finite-state machine in Example 2.
   a) 0111
   b) 1101111
   c) 0101010010

6. Find the output for each of these input strings when given as input to the finite-state machine in Example 3.
   a) 0000
   b) 10101
   c) 110111000010

7. Construct a finite-state machine that models an old-fashioned soda machine that accepts nickels, dimes, and quarters. The soda machine accepts change until 35 cents has been put in. It gives change back for any amount greater than 35 cents. Then the customer can push buttons to receive either a cola, a root beer, or a ginger ale.

8. Construct a finite-state machine that models a newspaper vending machine that has a door that can be opened only after either three dimes (and any number of nickels) or a quarter and a nickel (and any number of coins) have been inserted. Once the door can be opened, the customer opens it and takes a paper, closing the door. No change is ever returned, no matter how much money has been inserted. The next customer starts with no credit.
9. Construct a finite-state machine that delays an input string two bits, giving 00 as the first two bits of output.

10. Construct a finite-state machine that changes every other bit, starting with the second bit, of an input string, and leaves the other bits unchanged.

11. Construct a finite-state machine for the log-on procedure for a computer, where the user logs on by entering a user identification number, which is considered to be a single input, and then a password, which is considered to be a single input. If the password is incorrect, the user is asked for the user identification number again.

12. Construct a finite-state machine for a combination lock that contains numbers 1 through 40 and that opens only when the correct combination, 10, right, 8, second left, 37 right, is entered. Each input is a triple consisting of a number, the direction of the turn, and the number of times the lock is turned in that direction.

13. Construct a finite-state machine for a toll machine that opens a gate after 25 cents, in nickels, dimes, or quarters, has been deposited. No change is given for overpayment, and no credit is given to the next driver when more than 25 cents has been deposited.

14. Construct a finite-state machine for entering a security code into an automatic teller machine (ATM) that implements these rules: A user enters a string of four digits, one digit at a time. If the user enters the correct four digits of the password, the ATM displays a welcome screen. When the user enters an incorrect string of four digits, the ATM displays a screen that informs the user that an incorrect password was entered. If a user enters the incorrect password three times, the account is locked.

15. Construct a finite-state machine for a restricted telephone switching system that implements these rules. Only calls to the telephone numbers 0, 911, and the digit 1 followed by 10-digit telephone numbers that begin with 212, 300, 686, 877, and 883 are sent to the network. All other strings of digits are blocked by the system and the user hears an error message.

16. Construct a finite-state machine that gives an output of 1 if the number of input symbols read so far is divisible by 3 and an output of 0 otherwise.

17. Construct a finite-state machine that determines whether the input string has a 1 in the last position and a 0 in the third to the last position read so far.

18. Construct a finite-state machine that determines whether the input string read so far ends in at least five consecutive 1's.

19. Construct a finite-state machine that determines whether the word computer has been read as the last eight characters in the input read so far, where the input can be any string of English letters.

A Moore machine \( M = (S, I, O, f, g, s_0) \) consists of a finite set of states, an input alphabet \( I \), an output alphabet \( O \), a transition function \( f \) that assigns a next state to every pair of a state and an input, an output function \( g \) that assigns an output to every state, and a starting state \( s_0 \). A Moore machine can be represented either by a table listing the transitions for each pair of state and input and the outputs for each state, or by a state diagram that displays the states, the transitions between states, and the output for each state. In the diagram, transitions are indicated with arrows labeled with the input, and the outputs are shown next to the states.

20. Construct the state diagram for the Moore machine with this state table.

21. Construct the state table for the Moore machine with the state diagram shown here. Each input string to a Moore machine \( M \) produces an output string. In particular, the output corresponding to the input string \( s_1s_2 \ldots s_n \) is the string \( g(s_1)g(s_2) \ldots g(s_n) \), where \( f(s_i, a) = f(s_{i-1}, a) \) for \( i = 1, 2, \ldots, k \).

22. Find the output string generated by the Moore machine in Exercise 20 with each of these input strings.

23. Find the output string generated by the Moore machine in Exercise 21 with each of the input strings in Exercise 22.

24. Construct a Moore machine that gives an output of 1 whenever the number of symbols in the input string read so far is divisible by 4.

25. Construct a Moore machine that determines whether an input string contains an even or odd number of a's. The machine should give 1 as output if an even number of a's are in the string and 0 as output if an odd number of a's are in the string.
12.3 Finite-State Machines with No Output

Introduction

One of the most important applications of finite-state machines is in language recognition. This application plays a fundamental role in the design and construction of compilers for programming languages. In Section 12.2 we showed that a finite-state machine with output can be used to recognize a language, by giving an output of 1 when a string from the language has been read and 0 otherwise. However, there are other types of finite-state machines that are specially designed for recognizing languages. Instead of producing output, these machines have final states. A string is recognized if and only if it takes the starting state to one of these final states.

Set of Strings

Before discussing finite-state machines with no output, we will introduce some important background material on sets of strings. The operations that will be defined here will be used extensively in our discussion of language recognition by finite-state machines.

DEFINITION 1 Suppose that \( A \) and \( B \) are subsets of \( V^* \), where \( V \) is a vocabulary. The concatenation of \( A \) and \( B \), denoted by \( AB \), is the set of all strings of the form \( xy \), where \( x \) is a string in \( A \) and \( y \) is a string in \( B \).

EXAMPLE 1 Let \( A = \{0, 1\} \) and \( B = \{1, 10, 11\} \). Find \( AB \) and \( BA \).

Solution: The set \( AB \) contains every concatenation of a string in \( A \) and a string in \( B \). Hence, \( AB = \{01, 010, 011, 110, 111\} \). The set \( BA \) contains every concatenation of a string in \( B \) and a string in \( A \). Hence, \( BA = \{10, 111, 100, 1011, 1100, 1011\} \).

Note that it is not necessarily the case that \( AB = BA \) when \( A \) and \( B \) are subsets of \( V^* \), where \( V \) is an alphabet, as Example 1 illustrates.

From the definition of the concatenation of two sets of strings, we can define \( A^n \), for \( n = 0, 1, 2, \ldots \). This is done recursively by specifying that

\[
A^0 = \{\}\,
\]

\[
A^{n+1} = A^n A \quad \text{for } n = 0, 1, 2, \ldots
\]

EXAMPLE 2 Let \( A = \{1, 00\} \). Find \( A^n \) for \( n = 0, 1, 2, \) and 3.

Solution: We have \( A^0 = \{\}\) and \( A^1 = A^0 A = \{\}\ A = \{1, 00\} \). To find \( A^2 \) we take concatenations of pairs of elements of \( A \). This gives \( A^2 = \{11, 100, 001, 0000\} \). To find \( A^3 \) we take concatenations of elements in \( A^2 \) and \( A \); this gives \( A^3 = \{111, 1100, 1001, 10000, 0011, 00100, 00001, 000000\} \).

DEFINITION 2 Suppose that \( A \) is a subset of \( V^* \). Then the Kleene closure of \( A \), denoted by \( A^* \), is the set consisting of concatenations of arbitrarily many strings from \( A \). That is, \( A^* = \bigcup_{n=0}^{\infty} A^n \).
EXAMPLE 3

What are the Kleene closures of the sets $A = \{0\}$, $B = \{0, 1\}$, and $C = \{1\}$?

Solution: The Kleene closure of $A$ is the concatenation of the string 0 with itself an arbitrary finite number of times. Hence, $A^* = \{0^n \mid n = 0, 1, 2, \ldots \}$. The Kleene closure of $B$ is the concatenation of an arbitrary number of strings, where each string is either 0 or 1. This is the set of all strings over the alphabet $V = \{0, 1\}$. That is, $B^* = V^*$. Finally, the Kleene closure of $C$ is the concatenation of the string 11 with itself an arbitrary number of times. Hence, $C^* = \{11^n \mid n = 0, 1, 2, \ldots \}$. □

Finite-State Automata

We will now give a definition of a finite-state machine with no output. Such machines are also called finite-state automata, and that is the terminology we will use for them here. (Note: The singular of automata is automaton.) These machines differ from the finite-state machines studied in Section 12.2 in that they do not produce output, but do have a set of final states. As we will see, they recognize strings that take the starting state to a final state.

DEFINITION 3

A finite-state automaton $M = (S, I, f, s_0, F)$ consists of a finite set $S$ of states, a finite input alphabet $I$, a transition function $f$ that assigns a next state to every pair of state and input (so that $f: S \times I \rightarrow S$), an initial or start state $s_0$, and a subset $F$ of $S$ consisting of final (or accepting) states.

We can represent finite-state automata using either state tables or state diagrams. Final states are indicated in state diagrams by double circles.

EXAMPLE 4

Construct the state diagram for the finite-state automaton $M = (S, I, f, s_0, F)$, where $S = \{s_0, s_1, s_2, s_3\}$, $I = \{0, 1\}$, $F = \{s_0, s_3\}$, and the transition function $f$ is given in Table 1.

Solution: The state diagram is shown in Figure 1. Note that because both the inputs 0 and 1 take $s_2$ to $s_3$, we write 0, 1 over the edge from $s_2$ to $s_3$.

EXTENDING THE TRANSITION FUNCTION

The transition function $f$ of a finite-state machine $M = (S, I, f, s_0, F)$ can be extended so that it is defined for all pairs of states and strings; that is, $f$ can be extended to a function $f: S \times I^* \rightarrow S$. Let $x = x_1x_2 \ldots x_n$ be a string in $I^*$. Then $f(s, x)$ is the state obtained by using each successive symbol of $x$, from left to right, as the input to $f$. □
Table 1: The State Diagram for a Finite-State Automaton.

<table>
<thead>
<tr>
<th>State</th>
<th>Input 0</th>
<th>Input 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0</td>
<td>s0</td>
<td>s1</td>
</tr>
<tr>
<td>s1</td>
<td>s0</td>
<td>s2</td>
</tr>
<tr>
<td>s2</td>
<td>s0</td>
<td>s0</td>
</tr>
<tr>
<td>s3</td>
<td>s2</td>
<td>s1</td>
</tr>
</tbody>
</table>

If input, starting with state s1, we go on to state s2 = f(s1, x1), then to state s3 = f(s2, x2), and so on, with f(s1, x) = f(s0, x). Formally, we can define this extended transition function f recursively for the deterministic finite-state machine M = (S, I, f, s0, F) by

1. f(s, λ) = s for every state s ∈ S; and
2. f(s, xa) = f(f(s, x), a) for all s ∈ S, x ∈ I*, and a ∈ I.

We can use structural induction and this recursive definition to prove properties of this extended transition function. For example, in Exercise 15 we ask you to prove that

f(s, xy) = f(f(s, x), y)

for every state s ∈ S and strings x ∈ I* and y ∈ I*.

Language Recognition by Finite-State Machines

Next, we define some terms that are used when studying the recognition by finite-state automata of certain sets of strings.

Definition 4: A string x is said to be recognized or accepted by the machine M = (S, I, f, s0, F) if it takes the initial state s0 to a final state, that is, f(s0, x) is a state in F. The language recognized or accepted by the machine M, denoted by L(M), is the set of all strings that are recognized by M. Two finite-state automata are called equivalent if they recognize the same language.

In Example 5 we will find the languages recognized by several finite-state automata.

Example 5: Determine the languages recognized by the finite-state automata M1, M2, and M3 in Figure 2.

Solution: The only final state of M1 is s0. The strings that take s0 to itself are those consisting of zero or more consecutive 1s. Hence, L(M1) = \{1^n | n = 0, 1, 2, \ldots\}.

The only final state of M2 is s2. The only strings that take s0 to s2 are 1 and 01. Hence L(M2) = \{1, 01\}.

The final states of M3 are s1 and s3. The only strings that take s0 to itself are λ, 0, 00, 000, that is, any string of zero or more consecutive 0s. The only strings that take s0 to s3 are 01. Therefore L(M3) = \{λ, 0, 00, 000, 01\}.
Finite-State Machines

When studying the recognition by finite-state automata, we define the behavior of the machine \( M = (S, I, f, s_0, F) \) if it takes an input string to a state in \( F \). The language recognized or "digested" by \( M \) is the set of all strings that are recognized by \( M \); that is, those strings that take \( s_0 \) to some state in \( F \).

EXAMPLE 6

Construct deterministic finite-state automata that recognize each of these languages.

(a) the set of bit strings that begin with two 0s
(b) the set of bit strings that contain two consecutive 0s
(c) the set of bit strings that do not contain two consecutive 0s
(d) the set of bit strings that end with two 0s
(e) the set of bit strings that contain at least two 0s

Solution: (a) Our goal is to construct a deterministic finite-state automaton that recognizes the set of bit strings that begin with two 0s. Besides the start state \( s_0 \), we include a nonfinal state \( s_1 \); we move to \( s_1 \) from \( s_0 \) if the first bit is 0. Next, we add a final state \( s_2 \), which we move to from \( s_1 \) if the second bit is 0. When we have reached \( s_2 \) we know that the first two input bits...
are both 0s, so we stay in the state $s_2$ no matter what the succeeding bits (if any) are. We move to a nonfinal state $s_1$ from $s_0$ if the first bit is a 1 and from $s_1$ if the second bit is a 1. The reader should verify that the finite-state automaton in Figure 3(a) recognizes the set of bit strings that begin with two 0s.

(b) Our goal is to construct a deterministic finite-state automaton that recognizes the set of bit strings that contain two consecutive 0s. Besides the start state $s_0$, we include a nonfinal state $s_1$, which tells us that the last input bit seen is a 0, but either the bit before it was a 1, or this bit was the initial bit of the string. We include a final state $s_2$ that we move to from $s_1$ when the next input bit after a 0 is also a 0. If a 1 follows a 0 in the string (before we encounter two consecutive 0s), we return to $s_0$ and begin looking for consecutive 0s all over again. The reader should verify that the finite-state automaton in Figure 3(b) recognizes the set of bit strings that contain two consecutive 0s.

(c) Our goal is to construct a deterministic finite-state automaton that recognizes the set of bit strings that do not contain two consecutive 0s. Besides the start state $s_0$, which should be a final state, we include a final state $s_1$, which we move to from $s_0$ when 0 in the first input bit. When an input bit is a 1, we return to, or stay in, state $s_0$. We add a state $s_2$, which we move to from $s_1$ when the input bit is a 0. Reaching $s_2$ tells us that we have seen two consecutive 0s as input bits. We stay in state $s_2$ once we have reached it; this state is not final. The reader should verify that the finite-state automaton in Figure 3(c) recognizes the set of bit strings that do not contain two consecutive 0s. [The astute reader will notice the relationship between the finite-state automaton constructed here and the one constructed in part (b). See Exercise 39.]

(d) Our goal is to construct a deterministic finite-state automaton that recognizes the set of bit strings that end with two 0s. Besides the start state $s_0$, we include a nonfinal state $s_1$, which we move to if the first bit is 0. We include a final state $s_2$, which we move to from $s_1$ if the next input bit after a 0 is also a 0. If an input of 0 follows a previous 0, we stay in state $s_1$ because the last two input bits are still 0s. Once we are in state $s_2$, an input bit of 1 sends us back to $s_0$, and we begin looking for consecutive 0s all over again. We also return to $s_0$ if the next input bit is a 1.
Languages in Example 6.

what the succeeding bits (if any) are. We move
and from $s_1$ if the second bit is a 1. The reader
figure 3(a) recognizes the set of bit strings that

finite-state automaton that recognizes the set of
less the start state $s_0$, we include a nonfinal state
0, but either the bit before it was a 1, or this
a final state $s_2$ that we move to from $s_1$ when
ows a 0 in the string (before we encounter two
for consecutive 0s all over again. The reader
figure 3(b) recognizes the set of bit strings that

finite-state automaton that recognizes the set of
ve 0s. Besides the start state $s_0$, which should
b we move to from $s_0$ when 0 in the first input
stay in, state $s_0$. We add a state $s_2$, which we
aching $s_2$ tells us that we have seen two con-
we have reached it; this state is not final.
automaton in Figure 3(c) recognizes the set of
3s. (The astute reader will notice the relationship
ted here and the one constructed in part (b).

finite-state automaton that recognizes the set of
state $s_0$, we include a nonfinal state $s_1$, which
state $s_2$, which we move to from $s_1$ if the next
as a previous 0, we stay in state $s_2$ because the
0, an input bit of 1 sends us back to $s_0$, and
in. We also return to $s_0$ if the next input is a 1
when we are in state $s_1$. The reader should verify that the finite-state automaton in Figure 3(d)
recognizes the set of bit strings that end with two 0s.

(c) Our goal is to construct a deterministic finite-state automaton that recognizes the
set of bit strings that contain two 0s. Besides the start state, we include a state $s_3$, which
is not final; we stay in $s_0$ until an input bit is a 0 and we move to $s_1$ when we encounter
the first 0 bit in the input. We add a final state $s_2$, which we move to from $s_1$ once we en-
counter a second 0 bit. Whenever we encounter a 1 as input, we stay in the current state.
Once we have reached $s_2$, we remain there. Here, $s_1$ and $s_2$ are used to tell us that we have
already seen one or two 0s in the input string so far, respectively. The reader should ver-
ify that the finite-state automaton in Figure 3(e) recognizes the set of bit strings that contain
two 0s.

EXAMPLE 7 Construct a deterministic finite-state automaton that recognizes the set of bit strings that contain
an odd number of 1s and that end with at least two consecutive 0s.

Solution: We can build a deterministic finite-state automaton that recognizes the specified set
by including states that keep track of both the parity of the number of 1 bits and whether we
have seen no, one, or at least two 0s at the end of the input string.

The start state $s_0$ can be used to tell us that the input read so far contains an even number
of 1s and ends with no 0s (that is, is empty or ends with a 1). Besides the start state, we include
five more states. We move to states $s_1$, $s_2$, $s_3$, $s_4$, and $s_5$, respectively, when the input string read
so far contains an even number of 1s and ends with one 0; when it contains an even number
of 1s and ends with at least two 0s; when it contains an odd number of 1s and ends with no 0s;
when it contains an odd number of 1s and ends with one 0; and when it contains an odd number
of 1s and ends with two 0s. The state $s_2$ is a final state.

The reader should verify that the finite-state automaton in Figure 4 recognizes the set of bit
strings that contain an odd number of 1s and end with at least two consecutive 0s.

EQUIVALENT FINITE-STATE AUTOMATA. In Definition 4 we specified that two finite-
state automata are equivalent if they recognize the same language. Example 8 provides an
example of two equivalent deterministic finite-state machines.
**EXAMPLE 8** Show that the two finite-state automata $M_0$ and $M_1$ shown in Figure 5 are equivalent.

**Solution:** For a string $x$ to be recognized by $M_0$, $x$ must take us from $s_0$ to the final state $s_1$, or the final state $s_4$. The only string that takes us from $s_0$ to $s_1$ is the string 1. The strings that take us from $s_0$ to $s_4$ are those strings that begin with a 0, which takes us from $s_0$ to $s_2$, followed by zero or more additional 0's, which keep the machine in state $s_1$, followed by a 1, which takes us from state $s_2$ to the final state $s_4$. All other strings take us from $s_0$ to a state that is not final. (We leave it to the reader to fill in the details.) We conclude that $L(M_0)$ is the set of strings of zero or more 0 bits followed by a final 1.

For a string $x$ to be recognized by $M_1$, $x$ must take us from $s_0$ to the final state $s_1$, $s_2$, or $s_3$. So, for $x$ to be recognized, it must begin with some number of 0's, which leave us in state $s_0$, followed by a 1, which takes us to the final state $s_1$. A string of all zeros is not recognized because it leaves us in state $s_0$, which is not final. All strings that contain a 0 after 1 are not recognized because they take us to state $s_1$, which is not final. It follows that $L(M_1)$ is the same as $L(M_0)$. We conclude that $M_0$ and $M_1$ are equivalent.

Note that the finite-state machine $M_1$ only has three states. No finite state machine with fewer than three states can be used to recognize the set of all strings of zero or more 0 bits followed by a 1 (see Exercise 37).

As Example 8 shows, a finite-state automaton may have more states than one equivalent to it. In fact, algorithms used to construct finite-state automata to recognize certain languages may have many more states than necessary. Using unnecessarily large finite-state machines to recognize languages can make both hardware and software applications inefficient and costly. This problem arises when finite-state automata are used in compilers, which translate computer programs to a language a computer can understand (object code).

Exercises 39–61 develop a procedure that constructs a finite-state automaton with the fewest states possible among all finite-state automata equivalent to a given finite-state automaton. This procedure is known as **machine minimization**. The minimization procedure described in these exercises reduces the number of states by replacing states with equivalence classes of states with respect to an equivalence relation in which two states are equivalent if every input string that sends both states to a final state or sends both to a state that is not final. Before the minimization...
procedure begins, all states that cannot be reached from the start state using any input string are first removed; removing these does not change the language recognized.

**Nondeterministic Finite-State Automata**

The finite-state automata discussed so far are deterministic, because for each pair of state and input value there is a unique next state given by the transition function. There is another important type of finite-state automation in which there may be several possible next states for each pair of input value and state. Such machines are called nondeterministic. Nondeterministic finite-state automata are important in determining which languages can be recognized by a finite-state automation.

**DEFINITION 5**

A nondeterministic finite-state automation $M = (S, I, F, s_0, F)$ consists of a set $S$ of states, an input alphabet $I$, a transition function $f$ that assigns a set of states to each pair of state and input (so that $f : S \times I \rightarrow P(S)$), a starting state $s_0$, and a subset $F$ of $S$ consisting of the final states.

We can represent nondeterministic finite-state automata using state tables or state diagrams. When we use a state table, for each pair of state and input value we give a list of possible next

---

**GRACE BREWSTER MURRAY HOPPER (1906–1992)** Grace Hopper, born in New York City, displayed an intense curiosity as a child with no limits. At the age of seven, she disassembled a clock to discover their mechanisms. She inherited her love of mathematics from her mother, who received special permission to study geometry (but not algebra and trigonometry) at a time when women were actively discouraged from such study. Hopper was inspired by her father, a successful insurance broker, who had lost his legs from circular problems. He told his children they could do anything if they put their minds to it. He inspired Hopper to pursue higher education and conform to the usual roles for women. Her parents made sure that she had an excellent education; she attended private schools for girls in New York. Hopper entered Vassar College in 1924, where she majored in mathematics and physics; she graduated in 1928. She received a master's degree in mathematics from Yale University in 1930. In 1930 she also married her English instructor at the New York School of Commerce; she later divorced and did not have children. Hopper was a mathematics professor at Vassar from 1931 until 1943, earning a Ph.D. from Yale in 1934. After the attack on Pearl Harbor, Hopper, coming from a family with strong military traditions, decided to leave her academic position and join the Navy WAVES. To enlist, she needed special permission to leave her academic position as a mathematics professor, as well as a waiver for weighing too little. In December 1943, she was sworn into the Navy Reserve and trained at the Midshipman's School for Women. Hopper was assigned to work at the Naval Ordnance Laboratory at Harvard University. She wrote programs for the world's first large-scale computer, which was designed to help World War II artillery in varying weather. Hopper has been credited with coining the term "bug" to refer to a hardware glitch, but it was used at Harvard prior to her arrival there. However, it is true that Hopper and her programming team found a moth in one of the relays in the computer hardware that shut the system down. This famous moth was placed into a lab book. In the 1950s Hopper coined the term "debug" for the process of removing programming errors.

In 1946, when the Navy told her that she was too old for active service, Hopper chose to remain at Harvard as a civilian research fellow. In 1949 she left Harvard to join the Eckert-Mauchly Computer Corporation, where she helped develop the first commercial computer, UNIVAC. Hopper remained with this company when it was taken over by Remington Rand and then Remington Rand merged with the Sperry Corporation. She was a visionary for the potential power of computers; she understood that computers would become widely used if tools that were both programmer-friendly and application-friendly could be developed. In particular, she believed that computer programs could be written in English, rather than using machine instructions. To help achieve this goal, she developed the first compiler. She published the first research paper on compilers in 1952. Hopper is also known as the mother of the computer language COBOL; members of Hopper's staff helped to frame the basic language design for COBOL using their earlier work as a basis.

In 1968, Hopper retired from the Navy Reserve. However, only seven months later, the Navy recalled her from retirement to help standardize high-level naval computer languages. In 1980 she was promoted to the rank of Commodore by special Presidential appointment, and in 1985 she was elevated to the rank of Rear Admiral. Her retirement from the Navy, at the age of 80, was held on the USS Constitution.
states. In the state diagram, we include an edge from each state to all possible next states, labeling edges with the input or inputs that lead to this transition.

**EXAMPLE 9** Find the state diagram for the nondeterministic finite-state automaton with the state table shown in Table 2. The final states are $S_3$ and $S_1$.

*Solution:* The state diagram for this automaton is shown in Figure 6.

**EXAMPLE 10** Find the state table for the nondeterministic finite-state automaton with the state diagram shown in Figure 7.

*Solution:* The state table is given as Table 3.

What does it mean for a nondeterministic finite-state automaton to recognize a string $x = x_1x_2 \ldots x_n$? The first input symbol $x_1$ takes the starting state $S_0$ to a set $S_1$ of states. The next input symbol $x_2$ takes each of the states in $S_1$ to a set of states. Let $S_2$ be the union of these sets. We continue this process, including at a stage all states obtained using a state obtained at the previous stage and the current input symbol. We recognize, or accept, the string $x$ if there is a final state in the set of all states that can be obtained from $S_0$ using $x$. The language recognized by a nondeterministic finite-state automaton is the set of all strings recognized by this automaton.

![Figure 6](image)

**Figure 6** The Nondeterministic Finite-State Automaton with State Table Given in Table 2.

![Figure 7](image)

**Figure 7** A Nondeterministic Finite-State Automaton.

![Table 2](image)

**Table 2**

<table>
<thead>
<tr>
<th>Input</th>
<th>$f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>0</td>
</tr>
<tr>
<td>$S_0$</td>
<td>$S_0$, $S_1$</td>
</tr>
<tr>
<td>$S_1$</td>
<td>$S_0$</td>
</tr>
<tr>
<td>$S_2$</td>
<td>$S_0$, $S_2$</td>
</tr>
<tr>
<td>$S_3$</td>
<td>$S_0$, $S_1$, $S_2$</td>
</tr>
</tbody>
</table>

![Table 3](image)

**Table 3**

<table>
<thead>
<tr>
<th>Input</th>
<th>$f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>0</td>
</tr>
<tr>
<td>$S_0$</td>
<td>$S_0$, $S_1$, $S_2$, $S_3$</td>
</tr>
<tr>
<td>$S_1$</td>
<td>$S_3$</td>
</tr>
<tr>
<td>$S_2$</td>
<td>$S_3$</td>
</tr>
<tr>
<td>$S_3$</td>
<td>$S_1$</td>
</tr>
<tr>
<td>$S_4$</td>
<td>$S_5$</td>
</tr>
</tbody>
</table>
EXAMPLE 11 Find the language recognized by the nondeterministic finite-state automaton shown in Figure 7.

Solution: Because \( s_6 \) is a final state, and there is a transition from \( s_9 \) to itself when 0 is the input, the machine recognizes all strings consisting of zero or more consecutive 0s. Furthermore, because \( s_9 \) is a final state, any string that has \( s_9 \) in the set of states that can be reached from \( s_0 \) with this input string is recognized. The only such strings are strings consisting of zero or more consecutive 0s followed by 01 or 11. Because \( s_6 \) and \( s_9 \) are the only final states, the language recognized by the machine is \( \{0^n, 0^n1, 0^n11 \mid n \geq 0\} \).

One important fact is that a language recognized by a nondeterministic finite-state automaton is also recognized by a deterministic finite-state automaton. We will take advantage of this fact in Section 12.4 when we will determine which languages are recognized by finite-state automata.

THEOREM 1 If the language \( L \) is recognized by a nondeterministic finite-state automaton \( M_0 \), then \( L \) is also recognized by a deterministic finite-state automaton \( M_1 \).

Proof: We will describe how to construct the deterministic finite-state automaton \( M_1 \) that recognizes \( L \) from \( M_0 \), the nondeterministic finite-state automaton that recognizes this language. Each state in \( M_1 \) will be made up of a set of states in \( M_0 \). The start symbol of \( M_1 \) is \( \{s_0\} \), which is the set containing the start state of \( M_0 \). The input set of \( M_1 \) is the same as the input set of \( M_0 \).

Given a state \( \{s_0, s_1, \ldots, s_n\} \) of \( M_1 \), the input symbol \( x \) takes this state to the union of the sets of next states for the elements of this set, that is, the union of the sets \( f(s_0), f(s_1), \ldots, f(s_n) \). The states of \( M_1 \) are all the subsets of \( S \), the set of states of \( M_0 \), that are obtained in this way starting with \( \{s_0\} \). (There are as many as \( 2^n \) states in the deterministic machine, where \( n \) is the number of states in the nondeterministic machine, because all subsets may occur as states, including the empty set, although usually far fewer states occur.) The final states of \( M_1 \) are those sets that contain a final state of \( M_0 \).

Suppose that an input string is recognized by \( M_0 \). Then one of the states that can be reached from \( s_0 \) using this input string is a final state (the reader should provide an inductive proof of this). This means that in \( M_1 \), this input string leads from \( \{s_0\} \) to a set of states of \( M_0 \) that contains a final state. This subset is a final state of \( M_1 \), so this string is also recognized by \( M_1 \). Also, an input string not recognized by \( M_0 \) does not lead to any final states in \( M_0 \). (The reader should provide the details that prove this statement.) Consequently, this input string does not lead from \( \{s_0\} \) to a final state in \( M_1 \).

EXAMPLE 12 Find a deterministic finite-state automaton that recognizes the same language as the nondeterministic finite-state automaton in Example 10.

Solution: The deterministic automaton shown in Figure 8 is constructed from the nondeterministic automaton in Example 10. The states of this deterministic automaton are subsets of the set of all states of the nondeterministic machine. The next state of a subset under an input symbol is the subset containing the next states in the nondeterministic machine of all elements in this subset. For instance, on input of 0, \( \{s_0\} \) goes to \( \{s_0, s_1\} \), because \( s_0 \) has transitions to itself and to \( s_2 \) in the nondeterministic machine; the set \( \{s_1, s_2\} \) goes to \( \{s_1, s_2\} \) on input of 1, because \( s_0 \) goes just to \( s_1 \) and \( s_2 \) goes just to \( s_2 \) on input of 1 in the nondeterministic machine; and the set \( \{s_1, s_3\} \) goes to \( \{s_3\} \) on input of 0, because \( s_2 \) and \( s_4 \) both go to \( s_3 \) on input of 0 in the deterministic machine. All subsets that are obtained in this way are included in the deterministic finite-state machine. Note that the empty set is one of the states of this machine, because it is the subset containing all the next states of \( \{s_1\} \) on input of 0. The start state is \( \{s_0\} \), and the set of final states are all those that include \( s_0 \) or \( s_4 \).
Exercises

1. Let $A = \{0, 1\}$ and $B = \{00, 01\}$. Find each of these sets.
   a) $AB$  b) $BA$  c) $A^2$  d) $B^2$
2. Show that if $A$ is a set of strings, then $AB = \emptyset A = \emptyset$.
3. Find all pairs of sets of strings $A$ and $B$ for which $AB = \{10, 111, 1010, 1000, 10111, 10000\}$.
4. Show that these equalities hold.
   a) $(\lambda)^* = \{\lambda\}$
   b) $(A^*)^* = A^*$ for every set of strings $A$
5. Describe the elements of the set $A^*$ for these values of $A$.
   a) $\{10\}$  b) $\{11\}$  c) $\{0, 01\}$  d) $\{1, 101\}$
6. Let $V$ be an alphabet, and let $A$ and $B$ be subsets of $V^*$. Show that $|AB| \leq |A||B|$.
7. Let $V$ be an alphabet, and let $A$ and $B$ be subsets of $V^*$ with $A \subseteq B$. Show that $A^* \subseteq B^*$.
8. Suppose that $A$ is a subset of $V^*$, where $V$ is an alphabet. Prove or disprove each of these statements.
   a) $A \subseteq A^2$  b) if $A = A^2$, then $\lambda \in A$
   c) $A(\lambda) = A$  d) $(A^*)^* = A^*$
9. Determine whether the string $1101$ is in each of these sets.
   a) $\{0, 1\}^*$  b) $\{1\}^*\{0\}^*\{1\}^*$
   c) $\{1\}^*\{0\}^*\{01\}$  d) $\{11\}^*\{0\}^*\{1\}^*$
   e) $\{11, 0\}^*\{00, 101\}$
10. Determine whether the string $01001$ is in each of these sets.
    a) $\{0, 1\}^*$  b) $\{0\}^*\{10\}^*\{1\}^*$
    c) $\{010\}^*\{0\}^*\{1\}^*$  d) $\{010, 011\}^*\{00, 01\}$
    e) $\{00\}^*\{0\}^*\{01\}^*$  f) $\{01\}^*\{01\}^*$
11. Determine whether each of these strings is recognized by the deterministic finite-state automaton in Figure 1.
    a) $111$  b) $001$  c) $1010111$  d) $01101011$
12. Determine whether each of these strings is recognized by the deterministic finite-state automaton in Figure 1.
    a) $010$  b) $1101$  c) $1111110$  d) $010101010$
13. Determine whether all the strings in each of these sets are recognized by the deterministic finite-state automaton in Figure 1.
    a) $\{0\}^*$  b) $\{0\}^*\{0\}^*$  c) $\{1\}^*\{0\}^*$
    d) $\{01\}^*$  e) $\{0\}^*\{1\}^*$  f) $\{1\}^*\{0, 1\}^*$
14. Show that if $M = (S, I, f, q_0, F)$ is a deterministic finite-state automaton and $f(s, x) = s$ for the state $s \in S$ and the input string $x \in I^*$, then $f(s, x^n) = s$ for every non-negative integer $n$. (Here $x^n$ is the concatenation of $n$ copies of the string $x$, defined recursively in Exercise 17 in Section 4.3.)
15. Given a deterministic finite-state automaton $M = (S, I, f, q_0, F)$, use structural induction and the recursive definition of the extended transition function $f$ to prove that $f(s, x y) = f(f(s, x), y)$ for all states $s \in S$ and all strings $x \in I^*$ and $y \in I^*$.

In Exercises 16–22 find the language recognized by the given deterministic finite-state automaton.

16.

17.

18. Construct a deterministic finite-state automaton that recognizes the set of all bit strings that contain exactly three 0s.
19. Construct a deterministic finite-state automaton that recognizes the set of all bit strings that contain at least three 0s.
20. Construct a deterministic finite-state automaton that recognizes the set of all bit strings that contain three consecutive 1s.
21. Construct a deterministic finite-state automaton that recognizes the set of all bit strings that begin with 0 or with 11.
22. Construct a deterministic finite-state automaton that recognizes the set of all bit strings that begin and end with 11.
23. Construct a deterministic finite-state automaton that recognizes the set of all bit strings that contain an even number of 1s.
24. Construct a deterministic finite-state automaton that recognizes the set of all bit strings that contain an even number of 0s and an odd number of 1s.
25. Construct a finite-state automaton that recognizes the set of bit strings consisting of 0 followed by a string with an even number of 1s.
26. Construct a finite-state automaton with four states that recognizes the set of bit strings containing an even number of 1s and an odd number of 0s.
27. Construct a deterministic finite-state automaton constructed in Example 6 to find deterministic finite-state automata that recognize each of these sets.
   a) the set of bit strings that do not begin with two 0s
   b) the set of bit strings that do not end with two 0s
   c) the set of bit strings that contain at most one 0 (that is, that do not contain at least two 0s)
28. Use the procedure you described in Exercise 39 and the finite-state automata you constructed in Exercise 25 to find a deterministic finite-state automaton that recognizes the set of all bit strings that do not contain the string 101.
29. Use the procedure you described in Exercise 39 and the finite-state automaton you constructed in Exercise 29 to find a deterministic finite-state automaton that recognizes the set of all bit strings that do not contain the string 101.
In Exercises 43–49 find the language recognized by the given nondeterministic finite-state automaton.

43. 

```
Start
```

```
S1  0, 1
```

```
S2  1
```

```
End
```

44. 

```
Start
```

```
S1  0, 1
```

```
S2  1
```

```
End
```

45. 

```
Start
```

```
S1  0
```

```
S2  1, 0
```

```
S3  1
```

```
End
```

46. 

```
Start
```

```
S1  0
```

```
S2  1, 0
```

```
S3  1
```

```
End
```

47. 

```
Start
```

```
S1  0
```

```
S2  1
```

```
S3  0, 1
```

```
S4  1
```

```
End
```

48. 

```
Start
```

```
S1  0
```

```
S2  1
```

```
S3  0, 1
```

```
S4  1
```

```
End
```

49. 

```
Start
```

```
S1  0
```

```
S2  1
```

```
S3  0, 1
```

```
S4  1
```

```
End
```

50. Find a deterministic finite-state automaton that recognizes the same language as the nondeterministic finite-state automaton in Exercise 43.

51. Find a deterministic finite-state automaton that recognizes the same language as the nondeterministic finite-state automaton in Exercise 44.

52. Find a deterministic finite-state automaton that recognizes the same language as the nondeterministic finite-state automaton in Exercise 45.

53. Find a deterministic finite-state automaton that recognizes the same language as the nondeterministic finite-state automaton in Exercise 46.

54. Find a deterministic finite-state automaton that recognizes the same language as the nondeterministic finite-state automaton in Exercise 47.

55. Find a deterministic finite-state automaton that recognizes each of these sets.
   a) \{0\}    b) \{1, 00\}    c) \{1^n | n = 2, 3, 4, \ldots\}

56. Find a nondeterministic finite-state automaton that recognizes each of the languages in Exercise 27, and has four states, if possible, than the deterministic automaton you found in that exercise.

57. Show that there is no finite-state automaton that recognizes the set of bit strings containing an equal number of 0s and 1s.

In Exercises 58–62 we introduce a technique for constructing a deterministic finite-state machine equivalent to a given deterministic finite-state machine with the least number of states possible. Suppose that \(M = (S, I, f, q_0, F)\) is a finite state automaton and that \(k\) is a nonnegative integer. Let \(R_k\) be the relation on the set \(S\) of states of \(M\) such that \(s R_k t\) if and only if for every input string \(x\) with \(|x| \leq k\) (where \(|x|\) is the length of \(x\), as usual), \(f(s, x)\) and \(f(t, x)\) are both final states or both not final states. Furthermore, let \(R_0\) be the relation on the set of states of \(M\) such that \(s R_0 t\) if and only if for every input string \(x\), regardless of length, \(f(s, x)\) and \(f(t, x)\) are both final states or both not final states.

58. a) Show that for every nonnegative integer \(k\), \(R_k\) is an equivalence relation on \(S\). We say that two states \(s\) and \(t\) are \(k\)-equivalent if \(s R_k t\).
   b) Show that \(R_k\) is an equivalence relation on \(S\). We say that two states \(s\) and \(t\) are \(<\)-equivalent if \(s R_k t\).
   c) Show that if \(s\) and \(t\) are two \(k\)-equivalent states of \(M\) where \(k\) is a positive integer, then \(s\) and \(t\) are also \((k - 1)\)-equivalent.
   d) Show that the equivalence classes of \(R_k\) are a refinement of the equivalence classes of \(R_{k+1}\) if \(k\) is a positive integer. (The refinement of a partition of a set \(S\) is defined in the preface to Exercise 49 in Section 6.3.)
   e) Show that if \(s\) and \(t\) are \(k\)-equivalent for every nonnegative integer \(k\), then they are \(<\)-equivalent.
   f) Show that all states in a given \(R_k\)-equivalence class are final states or all are not final states.
   g) Show that if \(s\) and \(t\) are \(<\)-equivalent, then \(f(s, a)\) and \(f(t, a)\) are also \(R_k\)-equivalent for all \(a \in I\).