Chapter 11

Formal Languages and Finite-State Machines

Time as he grows old teaches many lessons.

—AESCHYLUS

The study of finite-state machines began with the neural networks investigations of Warren S. McCulloch and Walter Pitts in 1943. Today paradigms of finite-state constructs can be seen everywhere: turnstiles, traffic signal controllers, automated teller machines, automated telephone service, garage door openers, household appliances, and coin-operated machines such as vending machines and slot machines.

Finite-state machines significantly assist the study of formal languages; for example, a machine can be designed (or a program developed) that determines if an input string over the alphabet {a,b} contains *abba* as a substring. (The string *babaabbaba* does, while *abaaba* does not.) This type of machine produces no output, but instead tells whether or not the input string has a certain property. (Example 11.26 explores this.)

Some machines, however, produce output values. For instance, adding two binary numbers requires the input of two numbers, and yields their sum as the output (see Example 11.42). Such a machine, a finite-state automaton, is described in Section 11.4.

Since all finite-state automata must recognize particular languages, formal languages and types of grammars become important. This chapter explores formal languages, and how automata and formal languages are related, as well as other interesting questions such as:

- How do we determine if a string of characters contains a certain substring?
- How do we simulate an automatic teller machine?

- Can we develop a program that accepts two binary numbers, adds them bit by bit, and outputs their sum?
- What sort of languages are accepted by finite-state automata?

11.1 Formal Languages

We now continue our study of formal languages, begun in Section 2.1. The language of sets plays an important role in the study, as we saw in Chapter 2.

You may recall that an alphabet Σ is a finite set of symbols; and a word (or string) over Σ is a finite sequence of symbols from Σ.

How do we determine whether or not two words over Σ are equal? To this end, we make the following definition.

Equality of Words

Two words $x = x_1x_2\dots x_m$ and $y = y_1y_2\dots y_n$ over Σ are **equal**, denoted by $x = y$, if $m = n$ and $x_i = y_i$ for every i. Thus two words are equal if they contain the same number of symbols and the corresponding symbols are the same.

For example, if $01z = xy0$, then $x = 0, y = 1$, and $z = 0$. Also, $011 \neq 001$.

The length of a word w is the number of symbols in it. A word of length zero is the empty word, denoted by the lowercase Greek letter λ; it contains no symbols.

Again recall that Σ^* denotes the set of words over Σ. (Σ^* can be defined recursively. See Exercise 23.) A language over Σ is a subset of Σ^*; it may be finite or infinite.

EXAMPLE 11.1 Let $\Sigma = \{x, y, z, +, -, *, /, \uparrow, (,)\}$, where $*$ denotes multiplication and $\uparrow$ denotes exponentiation. Define a language L over Σ recursively as follows:

- $x, y, z \in L$.
- If u and v are in L, then so are $(+u), (-u), (u+v), (u-v), (u*v), (u/v)$, and $(u \uparrow v)$.

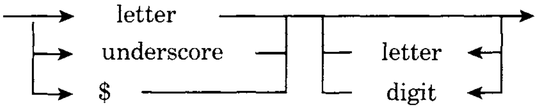
Then L consists of all fully and legally parenthesized algebraic expressions in x, y, and z. For instance, $((((x * (y \uparrow z)) - (y * z)) + x) \uparrow z)$ is a fully parenthesized and well-formed algebraic expression. Note that Σ^* contains nonsensical expressions such as $) + x\,(\,/y^* \uparrow$ also. ■

EXAMPLE 11.2 (optional) Let $\Sigma = \{\_, \$, a, \dots, z, 0, \dots, 9, \}$. Define the language L of legal identifiers in Java recursively.

SOLUTION:
An identifier in Java begins with a letter, underscore (_), or the dollar sign ($), followed by any number of alphanumeric characters (letters or digits). (See the syntax diagram in Figure 11.1.) A letter, an underscore, or $ by itself is a valid identifier. It can also be followed by a letter or a digit; that is, if $x \in L$ and $y \in \Sigma$, then $xy \in L$. Thus the language L can be defined recursively as follows:

- _ (underscore), $, and every letter of the English alphabet are in L.
- If $x \in L$ and $y \in \Sigma$, then $xy \in L$.

Figure 11.1



■

○ **EXAMPLE 11.3** (optional) The alphabet Σ for Java on a computer system that uses the ASCII character set consists of the blank character, the upper- and lowercase letters, digits, arithmetic and relational operators, special characters, and control characters. So Java is a subset of Σ^*, consisting of all words over Σ that are recognizable by a Java compiler. ■

Since both $\emptyset$ and $\{\lambda\}$ are subsets of Σ^*, both are languages by definition. The language $\emptyset$ is the **empty language**. The language $\{\lambda\}$ is denoted by the upper case Greek letter Λ. We emphasize that $\emptyset \neq \Lambda$, since $|\emptyset| = 0$, whereas $|\Lambda| = 1$. However, if $\Sigma = \emptyset$, $\Sigma^* = \Lambda$. Why?

Suppose an alphabet Σ contains at least one element a. Then $L = \{\text{a}, \text{aa}, \text{aaa}, \ldots\}$ is an infinite language over Σ. Since $L \subseteq \Sigma^*$, Σ^* is also infinite. Thus, if $\Sigma \neq \emptyset$, Σ^* contains infinitely many words, each of finite length (see Exercise 29).

Let z be the concatenation of the words x and y; that is, $z = xy$. Then x is a **prefix** of z and y a **suffix** of z. For instance, consider the word $z =$ *readability* over the English alphabet; $x =$ read is a prefix of z and $y =$ ability is a suffix of z. Since $x = \lambda x = x\lambda$, every word is a prefix and a suffix of itself; further, λ is both a prefix and a suffix of every word.

The operations of union and intersection can be applied to languages also; after all, languages are also sets. To this end, we extend the definition of concatenation of strings to languages.

Concatenation of Languages

Let A and B be any two languages over Σ. The **concatenation** of A and B, denoted by AB, is the set of all words ab with $a \in A$ and $b \in B$. That is, $AB = \{ab \mid a \in A \wedge b \in B\}$.

The next two examples illustrate this definition.

EXAMPLE 11.4 Let $\Sigma = \{0, 1\}, A = \{0, 01\}$, and $B = \{\lambda, 1, 110\}$. Find the concatenations AB and BA.

SOLUTION:

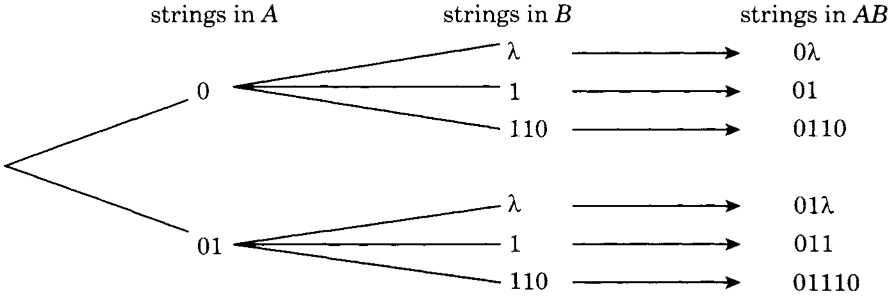
- AB consists of strings of the form ab with $a \in$ A and $b \in$ B. So

$$\begin{aligned} AB &= \{0\lambda, 01, 0110, 01\lambda, 011, 01110\} \\ &= \{0, 01, 0110, 01, 011, 01110\} \\ &= \{0, 01, 011, 0110, 01110\} \end{aligned}$$

- $$\begin{aligned} BA &= \{ba \mid b \in B \wedge a \in A\} \\ &= \{\lambda 0, \lambda 01, 10, 101, 1100, 11001\} \\ &= \{0, 01, 10, 101, 1100, 11001\} \end{aligned}$$ ■

Tree diagrams are useful in finding the various strings in the concatenation of two finite languages. Figure 11.2, for example, shows the different ways of obtaining the elements in AB for the languages in Example 11.4.

Figure 11.2



Two interesting points arise from this example:

(1) $AB \neq BA$.

(2) Further, $|AB| = 5 \leq 6 = 2 \cdot 3 = |A| \cdot |B|$; whereas $|BA| = 6 = 3 \cdot 2 = |B| \cdot |A|$. This is so since the word 01 in AB can be obtained in two ways. Therefore, all we can say in general is, if A and B are finite languages, then $|AB| \leq |A| \cdot |B|$.

○ **EXAMPLE 11.5** (optional) In the programming language QUICKBASIC, a numeric variable name must begin with a letter followed by either a period or an alphanumeric character. (QUICKBASIC does not distinguish between upper and lowercases in variable names.) Let $A = \{a, b, \dots, z\}$ and $B = \{a, \dots, z, 0, \dots, 9, .\}$. The concatenation AB gives all numeric variable names containing exactly two characters, namely:

$$\begin{gathered} aa,\ ab,\ \dots,\ a0,\ \dots,\ a9,\ a. \\ ba,\ bb,\ \dots,\ b0,\ \dots,\ b9,\ b. \\ \vdots \\ za,\ zb,\ \dots,\ z0,\ \dots,\ z9,\ z. \end{gathered}$$ ■

EXAMPLE 11.6 Let A be a language over Σ. Identify the languages $A\emptyset$ and $A\Lambda$.

SOLUTION:

- $A\emptyset = \{ab \mid a \in A \wedge b \in \emptyset\}$. Since $\emptyset$ contains no elements, no concatenations ab can be performed; therefore, $A\emptyset = \emptyset$. (Similarly, $\emptyset A = \emptyset$.)
- $A\Lambda = A\{\lambda\} = \{a\lambda \mid a \in A\} = \{a \mid a \in A\} = A$. (Similarly, $\Lambda A = A$.) ■

We are now ready to study some properties of the concatenation operation on languages.

THEOREM 11.1 Let A, B, C, and D be any languages over an alphabet Σ. Then:

(1) $A\emptyset = \emptyset = \emptyset A$ (2) $A\Lambda = A = \Lambda A$ (3) $A(BC) = (AB)C$

(4) $A(B \cup C) = AB \cup AC$ (5) $(B \cup C)A = BA \cup CA$

(6) $A(B \cap C) \subseteq AB \cap AC$ (7) $(B \cap C)A \subseteq BA \cap CA$

(8) If $A \subseteq B$ and $C \subseteq D$, then AC $\subseteq BD$.

PROOF:
We already proved parts 1 and 2 in Example 11.6. We now shall prove parts 4 and 6, and leave the other parts as exercises.

(4) *To prove that* $A(B \cup C) = AB \cup AC$:

We need to show that (a) $A(B \cup C) \subseteq AB \cup AC$ and (b) $AB \cup AC \subseteq A(B \cup C)$.

- *To prove that* $A(B \cup C) \subseteq AB \cup AC$:
  Let $x \in A(B \cup C)$. Then x is of the form yz, where $y \in A$ and $z \in B \cup C$. If $z \in B$, then $yz \in AB$ and hence $yz \in AB \cup AC$. If $z \in C$, then $yz \in AC$ and therefore $yz \in AB \cup AC$. Thus, in both cases $x = yz \in AB \cup AC$. Consequently, $A(B \cup C) \subseteq AB \cup AC$.

- *To prove that* $AB \cup AC \subseteq A(B \cup C)$:
  Let $x \in AB \cup AC$. Suppose $x \in AB$. Then $x = ab$ for some $a \in A$ and $b \in B$. Since $b \in B$, b also belongs to $B \cup C$. So $x = ab \in A(B \cup C)$. Similarly, if $x \in AC$, then also $x \in A(B \cup C)$. Thus in both cases $x \in A(B \cup C)$. Consequently, $AB \cup AC \subseteq A(B \cup C)$.

Therefore, by parts (a) and (b), $A(B \cup C) = AB \cup AC$.

(6) *To prove that* $A(B \cap C) \subseteq AB \cap AC$:
Let $x \in A(B \cap C)$. Then $x = yz$ for some element $y \in A$ and $z \in B \cap C$. Since $z \in B \cap C$, $z \in B$ and $z \in C$. So yz belongs to both AB and AC, and hence $yz \in AB \cap AC$; in other words, $x \in AB \cap AC$. Thus $A(B \cap C) \subseteq AB \cap AC$. ■

The next example verifies parts (4) and (6) of this theorem.

EXAMPLE 11.7 Let $\Sigma = \{a,b,c\}$, $A = \{a,ab\}$, $B = \{b,ab\}$, and $C = \{\lambda,bc\}$. Verify that (1) $A(B \cup C) = AB \cup AC$ and (2) $A(B \cap C) \subseteq AB \cap AC$.

SOLUTION:

$$\begin{aligned} AB &= \{ab, aab, abb, abab\} \\ AC &= \{a\lambda, abc, ab\lambda, abbc\} = \{a, ab, abc, abbc\} \\ AB \cup AC &= \{a, ab, aab, abb, abc, abab, abbc\} \\ AB \cap AC &= \{ab, abb\} \\ \text{(1)} \qquad B \cup C &= \{\lambda, b, ab, bc\} \\ \text{Then} \qquad A(B \cup C) &= \{a\lambda, ab, aab, abc, ab\lambda, abb, abab, abbc\} \\ &= \{a, ab, aab, abb, abc, abab, abbc\} \\ &= AB \cup AC \end{aligned}$$

(2) Since $B \cap C = \emptyset$, $A(B \cap C) = \emptyset$ and hence $A(B \cap C) \subseteq AB \cap AC$. ■

If the languages A and B are the same, then AB is often denoted by A^2. Thus A^2 consists of words obtained by concatenating each word in A with every word in A: $A^2 = \{xy \mid x,y \in A\}$. More generally, let $n \in \mathbb{N}$. Then A^n consists of all words obtained by $n - 1$ concatenations of words in A. In particular, Σ^n denotes the set of words obtained by $n - 1$ concatenations of symbols in Σ, that is, words of length n.

EXAMPLE 11.8 Let $\Sigma = \{a,b,c\}$, $A = \{a,ab,bc\}$, and $B = \{a,bc\}$. Find Σ^2, A^2, and B^3.

SOLUTION:

- $\Sigma^2 = \{xy \mid x,y \in \Sigma\} = \{aa, ab, ac, ba, bb, bc, ca, cb, cc\}$
- $A^2 = \{xy \mid x,y \in A\} = \{aa, aab, abc, aba, abab, abbc, bca, bcab, bcbc\}$
- $B^2 = \{aa, abc, bca, bcbc\}$

So $B^3 = \{aaa, aabc, abca, abcbc, bcaa, bcabc, bcbca, bcbcbc\}$ ■

***Stephen Cole Kleene** (1909–1994) was born in Hartford, Connecticut. His father was an economics professor and his mother, a poet. In 1930, he graduated from Amherst College and 4 years later received his Ph.D. in mathematics from Princeton.*

After teaching for 6 years at the University of Wisconsin, Madison, he joined the faculty at Amherst College for a year. For the next 4 years he served in the U.S. Naval Reserve. In 1946, he returned to the Madison campus and in 1964 became the Cyrus C. MacDuffee Professor of Mathematics and Computer Science. He served as Chairman of the Department of Mathematics, Acting Director of the Mathematics Research Center, and Dean of the College of Letters and Science.

Kleene was awarded an honorary Doctor of Science by Amherst College in 1970, the Steele Prize by the American Mathematical Society in 1983, and the National Medal of Science in 1990.

Kleene contributed significantly to the theory of recursive functions and the theory of automata.

Note: It follows by part 8 of Theorem 11.1 that if $A \subseteq B$, then $A^2 \subseteq B^2$ (Why?). More generally, it can be shown by induction that if $A \subseteq B$, then $A^n \subseteq B^n$ for every $n \in \mathbb{N}$.

Finally, from any language A over Σ, we can construct a new language A^* using the various powers of A. First we define $A^0 = \Lambda$.

Kleene Closure

Let A be a language over an alphabet Σ. Then $A^* = \bigcup_{n=0}^{\infty} A^n$ is the **Kleene closure** of A, in honor of the American logician Stephen Kleene. A^* consists of strings obtained by an arbitrary number of concatenations of words from A. * is the **Kleene operator**.

The following example illustrates this definition.

EXAMPLE 11.9 Let $A = \{0\}$, $B = \{11\}$, $C = \{000\}$, and $\Sigma = \{0, 1\}$. Find their Kleene closures.

SOLUTION:

- Since $A = \{0\}$, $A^n = \{0^n\}$. So $A^* = \bigcup_{n=0}^{\infty} A^n = \{0^n \mid n \geq 0\}$. In other words, A^* consists of strings of zero or more 0's.
- Since $B = \{11\} = \{1^2\}$, $B^2 = BB = \{1111\} = \{1^4\}$. So $B^3 = BB^2 = \{1^2\}\{1^4\} = \{1^6\}$. Thus, in general, $B^n = \{1^{2n}\}$. Thus $B^* = \bigcup_{n=0}^{\infty} B^n = \{1^{2n} \mid n \geq 0\}$. It consists of words of 1's of even length.
- Since $C = \{0^3\}$, $C^n = \{0^{3n}\}$. So $C^* = \{0^{3n} \mid n \geq 0\}$, the set of strings of 0's whose lengths are divisible by 3.

- The Kleene closure is the set of all possible words over Σ, namely, Σ^*. (This explains why we denoted it by Σ^* from the beginning of the section.) ■

We now turn to a few properties satisfied by the Kleene operator. We shall prove one of them and leave the others as exercises. Property 6 is a bit hard to prove, so we omit it; properties 4 and 5 require induction.

THEOREM 11.2 Let A and B be any languages over an alphabet Σ. Then:

(1) $\Lambda \subseteq A^*$

(2) $A \subseteq A^*$

(3) $A^*A^* = A^*$

(4) If $A \subseteq B$, then $A^* \subseteq B^*$.

(5) $(A^*)^* = A^*$

(6) $(A \cup B)^* = (A^* \cup B^*)^* = (A^*B^*)^*$

PROOF:

(3) **To prove that $\mathbf{A^*A^* = A^*}$:**

- *To prove that* $A^* \subseteq A^*A^*$: Since $\Lambda \subseteq A^*$, $A^*\Lambda \subseteq A^*A^*$ by Theorem 11.1. But $A^*\Lambda = A^*$ by Theorem 11.1. So

$$A^* \subseteq A^*A^* \tag{11.1}$$

- *To prove that* $A^*A^* \subseteq A^*$: Let $x \in A^*A^*$. Then $x = yz$ with $y,\ z \in A^*$. Since $y, z \in A^*$, $y \in A^m$ and $z \in A^n$ where $m,\ n \in W$. So $yz \in A^mA^n = A^{m+n}$. But $A^{m+n} \subseteq A^*$, so $x = yz \in A^*$. Thus

$$A^*A^* \subseteq A^* \tag{11.2}$$

Thus, by set inclusions (11.1) and (11.2), $A^*A^* = A^*$. ■

An interesting observation: For any language A, $A \subseteq A^*$. That is, when we apply the Kleene operator * on A, the resulting language A^* contains A. However, if we apply * to A^*, we find that $(A^*)^* = A^*$; so we do not get a new language. This explains why A^* is called the Kleene closure of A.

We conclude this section with an example involving both concatenation and the Kleene operators.

EXAMPLE 11.10 Identify each language over $\Sigma = \{\mathrm{a,b}\}$.

(1) $\{\mathrm{a,b}\}^*\{\mathrm{b}\}$ (2) $\{\mathrm{a}\}\{\mathrm{a,b}\}^*$ (3) $\{\mathrm{a}\}\{\mathrm{a,b}\}^*\{\mathrm{b}\}$ (4) $\{\mathrm{a,b}\}^*\{\mathrm{b}\}^*$

SOLUTION:

(1) $\{\mathrm{a,b}\}^*$ consists of all possible words over Σ including λ, whereas $\{\mathrm{b}\}$ contains just one word, namely, b. Therefore, the language $\{\mathrm{a,b}\}^*\{\mathrm{b}\}$ consists of words over Σ that have b as a suffix.

(2) Similarly, $\{\mathrm{a}\}\{\mathrm{a,b}\}^*$ consists of words that have a as a prefix.

(3) $\{a\}\{a,b\}^*\{b\}$ consists of words that begin with a and end in b.

(4) Every element in $\{b\}^*$ consists of a finite number of b's. Therefore, $\{a,b\}^*\{b\}^*$ consists of strings followed by a finite number of b's. Notice that this is different from $\{a,b\}^*\{b\}$ (Why?). ■

Exercises 11.1

In Exercises 1–4, a language L over $\Sigma = \{a,b\}$ is given. Find five words in each language.

1. $L = \{x \in \Sigma^* \mid x$ begins with and ends in $b.\}$

2. $L = \{x \in \Sigma^* \mid x$ contains exactly one $b.\}$

3. L is defined recursively as follows: (i) $\lambda \in L$ (ii) $x \in L \rightarrow xbb \in L$

4. L is defined recursively as follows: (i) $\lambda \in L$ (ii) $x \in L \rightarrow axb \in L$

Define each language L over the given alphabet recursively.

5. The language L of all palindromes over $\Sigma = \{a,b\}$. (A **palindrome** over Σ is a word that reads the same both forwards and backwards. For instance, *abba* is a palindrome.)

6. $L = \{a^n b^n \mid n \in N\}$, $\Sigma = \{a,b\}$

7. $L = \{0, 00, 10, 100, 110, 0000, 1010, \ldots\}$, $\Sigma = \{0,1\}$

8. L = set of binary representations of positive integers, $\Sigma = \{0,1\}$

9. $L = \{1, 11, 111, 1111, 11111, \ldots\}$, $\Sigma = \{0,1\}$

10. $L = \{x \in \Sigma^* \mid x = b^n a b^n,\ n \geq 0\}, \Sigma = \{a,b\}$

11. L = set of words over $\Sigma = \{0,1\}$ with prefix 00

12. L = set of words over $\Sigma = \{0,1\}$ with suffix 11

Mark each as true or false.

13. Every language over an alphabet is infinite.

14. If $\Sigma = \emptyset$, then $\Sigma^* = \emptyset$.

○ **15.** C++ is a finite language.

16. Every language is a set.

Using Example 11.1, determine if each is a well-formed and fully parenthesized arithmetic expression.

17. $(((x+y)/(((x-y)*z) \uparrow z))$

18. $(x \uparrow ((y-x) \uparrow (-z)))$

19. $(y + (z \uparrow (+x))/(-x))$

20. $((x - (y \uparrow z)) * (x + (y \uparrow (+z))))$

21. Define the set of words S over an alphabet Σ recursively. (*Hint*: Use concatenation.)

22. Define the language L of all binary representations of nonnegative integers recursively.

23. Let Σ be an alphabet. Define Σ^* recursively.
(*Hint:* Use concatenation.)

○ **24.** Define recursively the set S of integers acceptable in Java.

Arrange the binary words of each length in increasing order.

25. Length two. **26.** Length three.

A **ternary word** is a word over the alphabet $\{0, 1, 2\}$. Arrange the ternary words of each length in increasing order.

27. Length one **28.** Length two

***29.** Let Σ be a nonempty alphabet. Prove that Σ^* is infinite.
(*Hint*: Assume Σ^* is finite. Since $\Sigma \neq \emptyset$, it contains an element a. Let $x \in \Sigma^*$ with largest length. Now consider xa.)

Let $A = \{a, bc\}$ and $B = \{\lambda, ab, bc\}$. Find each concatenation.

30. AB **31.** BA **32.** A^2 **33.** A^3

Let $A = \{a, ab\}$, $B = \{a, b, ab\}$, $C = \{c\}$, *and* $D = \{c, bc\}$. Verify each.

34. $A\Lambda = A$ **35.** $\Lambda A = A$ **36.** $A(B \cup C) = AB \cup AC$

37. $(B \cup C)A = BA \cup CA$ **38.** $A(B \cap C) = AB \cap AC$

39. $(B \cap C)A = BA \cap CA$

40. If $A \subseteq B$ and $C \subseteq D$, then $AC \subseteq BD$.

Mark each as true or false, where A and B are arbitrary finite languages.

41. $\Lambda = \emptyset$ **42.** $A\emptyset = \emptyset$ **43.** $A\emptyset = \emptyset \mathrm{A}$ **44.** $A\Lambda = \Lambda$

45. $A\Lambda = \Lambda A$ **46.** $|A \times B| = |B \times A|$ **47.** $|AB| = |BA|$

Find three words belonging to each language over $\sigma = \{0, 1\}$.

48. $\{0\}^*$ **49.** $\{0\}\{1\}^*$ **50.** $\{0\}^*\{1\}$ **51.** $\{0\}\{11\}^*\{1\}$

52. $\{0\}^*\{1\}^*$ **53.** $\{01\}^*$ **54.** $\{0\}\{0,1\}^*\{1\}$ **55.** $\{0\}^*\{1\}^*\{0\}^*$

Prove each, where A, B, and C are arbitrary languages over Σ and $x \in \Sigma$.

56. $\|x^n\| = n\|x\|$ for every $n \geq 0$.

57. If $A \subseteq B$, then $A^n \subseteq B^n$ for every $n \geq 0$.

58. If $A \subseteq B$, then $A^* \subseteq B^*$. **59.** $(A^*)^* = A^*$

60. $\emptyset A = \emptyset$ **61.** $\Lambda A = A$ **62.** $\Lambda \subseteq A^*$

63. $A \subseteq A^*$

64. $A(B \cap C) \subseteq AB \cap AC$

65. $(B \cup C)A = BA \cup CA$

66. $(B \cap C)A \subseteq BA \cap CA$

67. $(A^*B^*)^* = (B^*A^*)^*$

68. $(A^* \cup B^*)^* = (A \cup B)^*$

11.2 Grammars

Words in a natural language such as English or French can be combined in several ways. Some combinations form valid sentences, while others do not. The **grammar** of a language is a set of rules that determines whether or not a sentence is considered valid. For instance, *The milk drinks child quickly*, although meaningless, is a perfectly legal sentence.

The sentences in a language may be nonsensical, but must obey the grammar. Our discussion deals with only the **syntax** of sentences (the way words are combined), and not with the **semantics** of sentences (meaning). Although listing the rules that govern a natural language such as English is extremely complex, specifying the rules for subsets of English is certainly possible.

The next example introduces such a language.

EXAMPLE 11.11 The sentence *The child drinks milk quickly*, has two parts: a subject, *The child*, and a predicate *drinks milk quickly*. The subject consists of the definite article *The* and the noun *child*. The predicate, on the other hand, consists of the verb *drinks* and the object phrase *milk quickly*; the object phrase in turn has the object *milk* and the adverb *quickly*. This structure of the sentence can appear as a sequence of trees (Figures 11.3–11.7), with the **derivation tree** of the sentence in Figure 11.7.

Figure 11.3

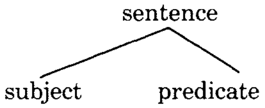
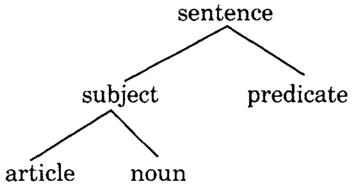


Figure 11.4



The derivation tree exhibits certain characteristics:

- Each leaf represents a word, a **terminal symbol**. The set of terminal symbols is $T = \{\text{the, child, drinks, milk, quickly}\}$.

Figure 11.5

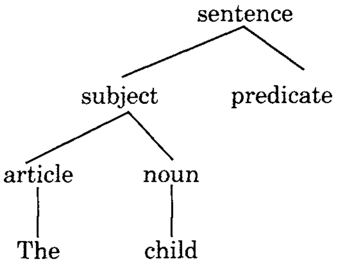


Figure 11.6

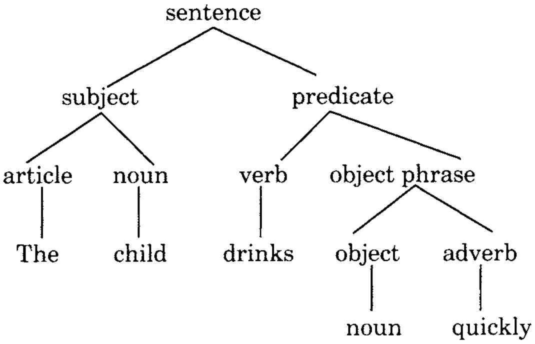
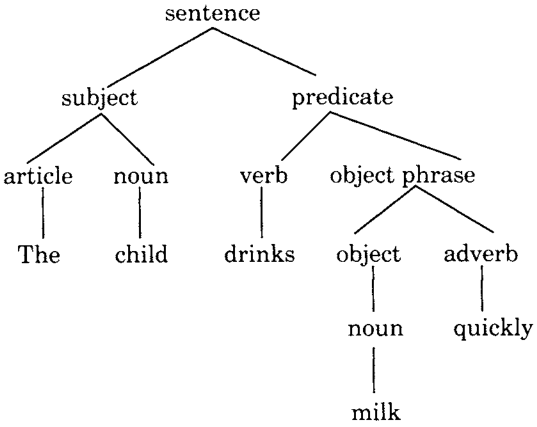


Figure 11.7



- Each internal vertex represents a grammatical class, a **nonterminal**. The set of nonterminals is $N = \{\text{sentence, subject, predicate, article, noun, object phrase, object, verb, adverb}\}$. A nonterminal symbol is enclosed within **angle brackets**, $\langle$ and $\rangle$. For instance, the nonterminal "subject" is denoted by $\langle\text{subject}\rangle$.

- The root of the tree represents the nonterminal symbol $\langle\text{sentence}\rangle$ called the **start symbol**, denoted by σ.

Certain rules can generate the above sentence. Every rule, called a **production rule** or a **substitution rule**, is of the form $w \to w'$ where $w \in N$

and w' may be a terminal symbol, a nonterminal symbol, or a combination of both.

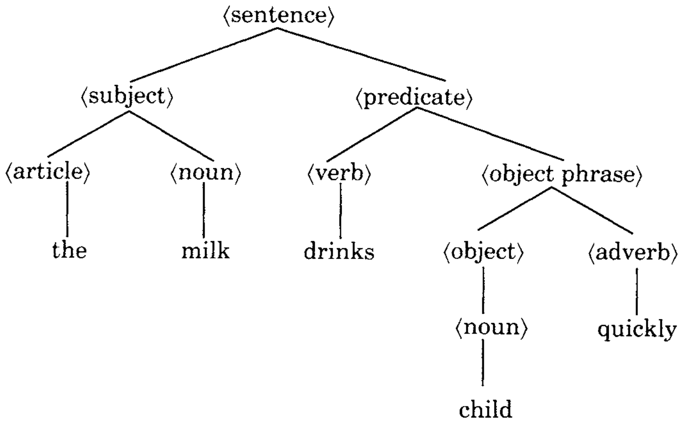
The production rules of the above sentence are:

$$\begin{aligned}
\langle\text{sentence}\rangle &\rightarrow \langle\text{subject}\rangle\langle\text{predicate}\rangle\\
\langle\text{subject}\rangle &\rightarrow \langle\text{article}\rangle\langle\text{noun}\rangle\\
\langle\text{article}\rangle &\rightarrow \textit{the}\\
\langle\text{noun}\rangle &\rightarrow \textit{child}\\
\langle\text{noun}\rangle &\rightarrow \textit{milk}\\
\langle\text{predicate}\rangle &\rightarrow \langle\text{verb}\rangle\langle\text{object phrase}\rangle\\
\langle\text{verb}\rangle &\rightarrow \textit{drinks}\\
\langle\text{object phrase}\rangle &\rightarrow \langle\text{object}\rangle\langle\text{adverb}\rangle\\
\langle\text{object}\rangle &\rightarrow \langle\text{noun}\rangle\\
\langle\text{adverb}\rangle &\rightarrow \textit{quickly}
\end{aligned}$$

■

The production rules specify the arrangement of words in a sentence: the **syntax** of the language. They produce syntactically correct sentences (which can be meaningless). For instance, the sentence, *The milk drinks child quickly* makes no sense but is syntactically valid. Figure 11.8 shows the derivation tree of this sentence.

Figure 11.8



Determining whether a program is syntactically correct is of the utmost importance in computer science. Before executing a program, the compiler checks the syntax of each sentence (or expression) by constructing derivation trees. (This process is **parsing**, and the corresponding derivation tree is a **parse tree**.)

We now turn to present the definition of a phrase-structure grammar.

Phrase-Structure Grammar

A **phrase-structure grammar** (or simply a **grammar**) G bears four features:

- A finite set N of **nonterminal symbols**;
- A finite set T of **terminal symbols**, where $N \cap T = \emptyset$;
- A finite subset P of $[(N \cup T)^* - T^*] \times (N \cup T)^*$; each element of P is called a **production**;
- A **start symbol** σ belonging to N;

The grammar G is denoted by $G = (N, T, P, \sigma)$.

These features meet certain requirements:

- The start symbol σ is nonterminal.
- No symbol can be both terminal and nonterminal.
- Every production has at least one nonterminal symbol on its LHS, because $P \subseteq [(N \cup T)^* - T^*] \times (N \cup T)^*$. Also, P is a binary relation from $(N \cup T)^* - T^*$ to $(N \cup T)^*$.
- If $(w, w') \in P$, we then write $w \to w'$; since $w \in (N \cup T)^* - T^*$, w contains at least one nonterminal symbol; but $w' \in (N \cup T)^*$; so it may contain terminal symbols, nonterminals, or both.

Grammars not only produce natural languages, but also formal ones, as the next two examples demonstrate.

EXAMPLE 11.12 Let $N = \{A, B, \sigma\}$, $T = \{a, b\}$, and $P = \{\sigma \to aA,\ A \to bA,\ A \to a\}$. Then $G = (N, T, P, \sigma)$ is a grammar. Notice that the production $A \to bA$ is recursive. ■

EXAMPLE 11.13 Let $N = \{A, \sigma\}$, $T = \{a, b\}$, and $P = \{\sigma \to a\sigma,\ \sigma \to Aa,\ A \to b\}$. Then $G = (N, T, P, \sigma)$ is a grammar. Again notice that the production $\sigma \to a\sigma$ is recursive. ■

Next we define the language generated by a grammar.

Derivation and Language

Let $G = (N, T, P, \sigma)$ be a grammar. If $w = x\alpha y$ and $w' = x\beta y$ are any two words in $(N \cup T)^*$, and if there exists a production $\alpha \to \beta$, then the word w' is said to be **directly derivable** from w; we then write $w \Longrightarrow w'$. If there is a finite sequence of words $w_0, w_1, \ldots, w_n$ in $(N \cup T)^*$ such that

$w_0 \Longrightarrow w_1, w_1 \Longrightarrow w_2, \ldots, w_{n-1} \Longrightarrow w_n$, then w_n is **derivable** from w_0. The finite sequence of steps, $w_0 \Longrightarrow w_1 \Longrightarrow \cdots \Longrightarrow w_n$, is a **derivation** of w_n from w_0.

The set of words in T^* derivable from σ by G is the **language generated** by G, denoted by *L(G).*

The next two examples illustrate these definitions.

EXAMPLE 11.14 Identify the language $L(G)$ generated by the grammar in Example 11.12.

SOLUTION:
Since the grammar contains exactly one production involving σ, namely, $\sigma \to aA$, start with it to find every word in the language. Now select the next production: $A \to bA$ or $A \to a$. The production $A \to a$ produces exactly one word, a. $A \to bA$ chosen n times, produces $\sigma \Longrightarrow aA \Longrightarrow abA \Longrightarrow ab^2A \Longrightarrow \cdots \Longrightarrow ab^nA$. Now $A \to a$ yields the word ab^na and, when $n = 0$, this yields $a\lambda a = aa$. (*Note*: $b^0 = \lambda$, the null word.) Every word derivable from σ fits the form ab^na, where $n \geq 0$. In other words, $L(G) = \{ab^na \mid n \geq 0\}$. ■

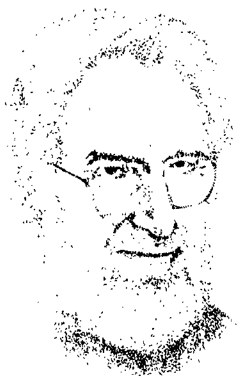
Example 11.14 illustrates that a grammar G can determine if it generates a string in the language $L(G)$. With some difficulty, the language could be described. Again with some difficulty, and a lot of patience and practice, a grammar G that generates a given language can be found, as Example 11.15 demonstrates.

EXAMPLE 11.15 Define a grammar $G = (N, T, P, \sigma)$ that generates the language $L = \{a^nb^n \mid n \geq 1\}$.

SOLUTION:
Since every word in L must contain the same number of a's and b's, G must contain a production of the form $\sigma \to$ aAb. Consequently, to produce a new word from aAb containing the same number of a's and b's requires another production A $\to$ aAb. From these two productions, we can derive all strings of the form $\mathrm{a}^n\mathrm{Ab}^n$ (Verify this.). All that remains to be done to define the grammar is the production A $\to \lambda$ to terminate the recursive procedure. Thus, $N = \{\sigma, \mathrm{A}\}$, $T = \{\mathrm{a}, \mathrm{b}, \lambda\}$, and $P = \{\sigma \to \mathrm{aAb}, \mathrm{A} \to \mathrm{aAb}, \mathrm{A} \to \lambda\}$. ■

The first two production rules in this example look quite similar, except for the start symbol, and can be combined into a single production, $\sigma \to \mathrm{a}\sigma\mathrm{b}$. The production rules $\sigma \to$ aAb and A $\to \lambda$ can yield the word ab, so the third production is $\sigma \to$ ab. Thus $P' = \{\sigma \to \mathrm{a}\sigma\mathrm{b}, \sigma \to \mathrm{ab}\}$ is an additional production set that yields the same language. In other words, the grammars $G = (N, T, P, \sigma)$ and $G' = \{N', T', P', \sigma)$ generate the same language L, where $N' = \{\sigma\}$ and $T' = \{\mathrm{a}, \mathrm{b}\}$. Thus $L(G) = L(G')$, so the grammars G and G' are **equivalent**. Our conclusion: *The grammar that generates a language need not be unique.*

John W. Backus *(1924–) was born in Philadelphia. He received his B.S. and M.S. in mathematics from Columbia University. After joining IBM in 1950, he became instrumental in the development of FORTRAN and ALGOL (**ALGO**rithmic **L**anguage). He received the W. W. McDowell Award from The* Institute of Electrical and Electronics Engineers *(IEEE) in 1967, the National Medal of Science in 1975, the A. M. Turing Award from the* Association for Computing Machinery *in 1977, the Harold Pender Award from the University of Pennsylvania in 1983, and an honorary doctorate from York University, England, in 1985.*



Peter Naur *(1928–), a computer scientist and prolific writer, was born in Frederiksberg, Denmark. After receiving his M.A. in astronomy from Copenhagen University in 1949, he spent the next two years at Cambridge University, England, where he used the EDSAC, one of the earliest computers, to pursue astronomy. He received his Ph.D. in astronomy from Copenhagen in 1957.*

From 1953 to 1959, he consulted for the design of the first Danish computer, the DASK. Beginning around 1964, he became increasingly involved in datalogy *(a word he coined), the study of data and data processes. In 1963, Naur was given the Hagemanns Gold Medal and three years later the Rosenhjaer Prize.*

Backus-Normal Form

The most widely used notation for describing the syntax of programming languages is the **Backus-Normal Form** (**BNF**), developed by John Backus, who described ALGOL 60 with it. Peter Naur edited the ALGOL 60 report, which appeared in 1963, so the BNF notation is also called the **Backus–Naur Form**.

In BNF, the production symbol → is denoted by ::=; thus the production $w \to w'$ is written as $w ::= w'$. Production rules with the same LHS are combined by separating their RHS with vertical bars. For instance, the productions $w \to w_1, w \to w_2, \dots, w \to w_n$ become $w ::= w_1 \mid w_2 \mid \dots \mid w_n$. (You may read the vertical bar as *or*.) Nonterminal symbols have angle brackets around them.

EXAMPLE 11.16 Study the following production rules:

$$\begin{aligned}
\langle\text{sentence}\rangle &\to \langle\text{subject}\rangle\langle\text{predicate}\rangle\\
\langle\text{subject}\rangle &\to \langle\text{article}\rangle\langle\text{noun}\rangle\\
\langle\text{predicate}\rangle &\to \langle\text{verb}\rangle\langle\text{object}\rangle\\
\langle\text{object}\rangle &\to \langle\text{article}\rangle\langle\text{noun}\rangle\\
\langle\text{article}\rangle &\to a\\
\langle\text{article}\rangle &\to \textit{the}\\
\langle\text{noun}\rangle &\to \textit{hare}\\
\langle\text{noun}\rangle &\to \textit{tortoise}\\
\langle\text{noun}\rangle &\to \textit{race}\\
\langle\text{verb}\rangle &\to \textit{beats}\\
\langle\text{verb}\rangle &\to \textit{wins}
\end{aligned}$$

BNF shortens these rules:

$$\begin{aligned}
\langle\text{sentence}\rangle &::= \langle\text{subject}\rangle\langle\text{predicate}\rangle\\
\langle\text{subject}\rangle &::= \langle\text{article}\rangle\langle\text{noun}\rangle\\
\langle\text{predicate}\rangle &::= \langle\text{verb}\rangle\langle\text{object}\rangle\\
\langle\text{object}\rangle &::= \langle\text{article}\rangle\langle\text{noun}\rangle\\
\langle\text{article}\rangle &::= a \mid \textit{the}\\
\langle\text{noun}\rangle &::= \textit{hare} \mid \textit{tortoise} \mid \textit{race}\\
\langle\text{verb}\rangle &::= \textit{beats} \mid \textit{wins}
\end{aligned}$$

■

EXAMPLE 11.17 The grammar for the language of correctly nested parentheses contains one production:

$$\langle\text{nested parentheses}\rangle ::= \lambda \mid (\langle\text{nested parentheses}\rangle)$$

where λ denotes the null string. [Using this definition, you may verify that (()) and ((())) are valid nested parentheses, whereas (() and (())) are not.] ■

EXAMPLE 11.18 (optional) An integer is a string of digits preceded by an optional sign, + or −. Using BNF, it can be defined as follows:

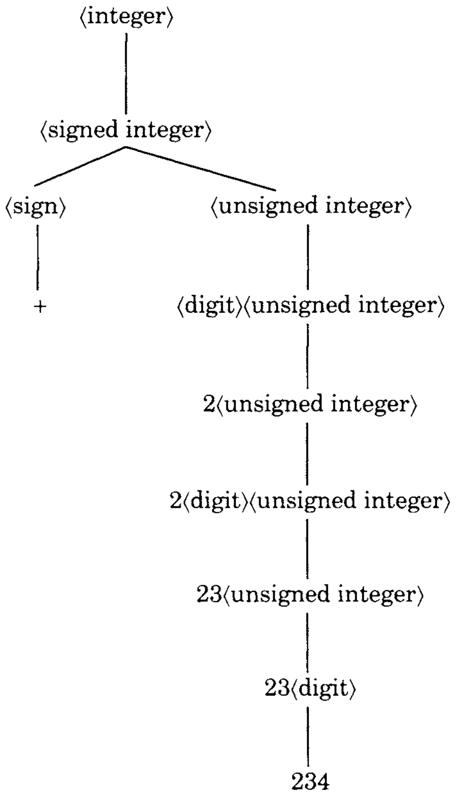
$$\begin{aligned}
\langle\text{integer}\rangle &:: = \langle\text{signed integer}\rangle \mid \langle\text{unsigned integer}\rangle\\
\langle\text{signed integer}\rangle &:: = \langle\text{sign}\rangle \mid \langle\text{unsigned integer}\rangle\\
\langle\text{sign}\rangle &:: = + \mid -
\end{aligned}$$

$$\langle\text{unsigned integer}\rangle ::= \langle\text{digit}\rangle \mid \langle\text{digit}\rangle\langle\text{unsigned integer}\rangle$$
$$\langle\text{digit}\rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

For instance, 234, +234, and −234 are valid integers. Figure 11.9 shows the derivation tree for the integer +234.

Figure 11.9

Derivation tree for the integer +234.



■

The grammar defined in this example is $G = (N, T, P, \sigma)$, where:

- $N = \{\langle\text{integer}\rangle, \langle\text{signed integer}\rangle, \langle\text{unsigned integer}\rangle, \langle\text{sign}\rangle, \langle\text{digit}\rangle\}$,
- $T = \{+, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$,
- The production rules are:

$$\begin{aligned}
\langle\text{integer}\rangle &\to \langle\text{signed integer}\rangle \mid \langle\text{unsigned integer}\rangle\\
\langle\text{signed integer}\rangle &\to \langle\text{sign}\rangle \mid \langle\text{unsigned integer}\rangle\\
\langle\text{sign}\rangle &\to + \mid -\\
\langle\text{unsigned integer}\rangle &\to \langle\text{digit}\rangle \mid \langle\text{digit}\rangle\langle\text{unsigned integer}\rangle\\
\langle\text{digit}\rangle &\to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{aligned}$$

- The start symbol σ is $\langle$integer$\rangle$.

Grammars are categorized by the productions that define them.

Context-Sensitive, Context-Free, and Regular Grammars

Let $G = (N, T, P, \sigma)$ be a grammar. Let $A, B \in N$ and α, α', $\beta \in (N \cup T)^*$. Notice that α, α', and β could be the null word.

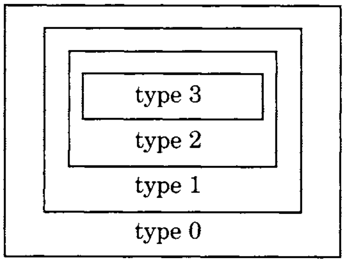
- Any phrase-structure grammar is **type 0**.
- G is **context-sensitive** (or **type 1**) if every production is of the form $\alpha \mathrm{A} \alpha' \to \alpha \beta \alpha'$.
- G is **context-free** (or **type 2**) if every production is of the form $\mathrm{A} \to \alpha$.
- G is **regular** (or **type 3**) if every production is of the form $\mathrm{A} \to t$ or $\mathrm{A} \to t\mathrm{B}$, where $t \in T$.

In a context-sensitive grammar, β can replace A in the word $\alpha \mathrm{A} \alpha'$ only when A lies between α and α'. In a context-free grammar, the LHS of every production is a single nonterminal symbol A, which α can replace. In a regular grammar, the LHS of every production consists of a single nonterminal symbol A and the RHS consists of a terminal symbol t or a terminal symbol t followed by a nonterminal symbol B; t or tB can always replace A. (In tB, the nonterminal must be on the RHS of the terminal symbol t.)

A regular grammar is also context-free and a context-free grammar is also context-sensitive. The Venn diagram in Figure 11.10 shows the **Chomsky hierarchy** of the various grammars, named in honor of Noam Chomsky, who developed the theory of formal languages.

Figure 11.10

Chomsky hierarchy of grammars.



Context-Sensitive, Context-Free, and Regular Languages

A language $L(G)$ is **context-sensitive, context-free,** or **regular** if the grammar G is context-sensitive, context-free, and regular, respectively.

The next five examples clarify these definitions.

***(Avram) Noam Chomsky** (1928–), a linguist, writer, and political activist, was born in Philadelphia, as the son of a Hebrew scholar. At 10 he proofread the manuscript of his father's edition of a 13th century Hebrew grammar. "This backdoor introduction to 'historical linguistics' had considerable impact on his future" (The New York Times Magazine). The young Chomsky, however, was more passionate about politics than about grammar.*

On graduating from Central High School in Philadelphia in 1945, Chomsky entered the University of Pennsylvania and received his B.A. in 1949 and M.A. 2 years later.

Chomsky received his Ph.D. in linguistics from the University of Pennsylvania in 1955 and joined the faculty at the Massachusetts Institute of Technology.

His first book, Syntactic Structures *(1957), developed from his notes for an introductory course in linguistics, triggered the Chomskyan revolution in linguistics "by disputing traditional ideas about language development." Chomsky is considered the father of the theory of formal languages.*

In 1966, Chomsky became the Ferrari P. Ward Professor of Modern Languages and Linguistics. He had been a visiting professor at Columbia, Princeton, and the University of California at Los Angeles and at Berkeley.

A recipient of numerous awards and honorary degrees, including the Kyoto prize in Basic Sciences in 1988, Chomsky was named one of the thousand "makers of the twentieth century" by the London Times.

EXAMPLE 11.19 Every production of the grammar G in Example 11.12 is A → t or A → tB, so G is a regular grammar. Consequently, $L(G) = \{\mathrm{ab}^n\mathrm{a} \mid n \geq 0\}$ is a regular language. (See also Example 11.14.) ■

EXAMPLE 11.20 In Example 11.13, the RHS of the production $\sigma \rightarrow \mathrm{A}a$ contains the terminal symbol a on the right of the nonterminal symbol A, so G is not regular. However, since every production appears as $w \rightarrow \alpha$ where $w \in N$ and $\alpha \in (N \cup T)^*$, G is context-free; thus $L(G)$ is a context-free language. ■

EXAMPLE 11.21 (optional) Not every production of the grammar G in Example 11.18 is of the form A → t or A → tB. For instance, the production ⟨unsigned integer⟩ ::= ⟨digit⟩⟨unsigned integer⟩ is not of either form.

The production rules, however, can be rewritten as follows:

⟨integer⟩ ::= +⟨unsigned integer⟩
⟨integer⟩ ::= −⟨unsigned integer⟩
⟨unsigned integer⟩ ::= 0⟨unsigned integer⟩| . . . | 9⟨unsigned integer⟩
⟨unsigned integer⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Clearly, the form A $\rightarrow t$ or A $\rightarrow t$B always results. So this grammar G for the set L(G) of integers is regular. Thus the set of integers is a regular language and hence context-free. ■

EXAMPLE 11.22 Consider the grammar $G = (N, T, P, \sigma)$, where $N = \{A, B, \sigma\}$, $T = \{\mathrm{a}, \mathrm{b}\}$, and $P = \{\sigma \rightarrow \mathrm{a}\sigma\mathrm{b}, \sigma \rightarrow \mathrm{aAb}, \mathrm{aAb} \rightarrow \mathrm{aBb}, \mathrm{A} \rightarrow \mathrm{a}, \mathrm{B} \rightarrow \mathrm{b}, \mathrm{A} \rightarrow \lambda, \mathrm{B} \rightarrow \lambda\}$. In the production aAb → aBb, A can be replaced with B only if A is surrounded by a and b. Notice that $L(G) = \{\mathrm{a}^m\mathrm{b}^m, \mathrm{a}^m\mathrm{b}^{m+1}, \mathrm{a}^{m+1}\mathrm{b}^m \mid m \geq 1\}$. ■

EXAMPLE 11.23 The grammar $G = (N, T, P, \sigma)$ in Example 11.15 is context-free, so $L(G) = \{\mathrm{a}^n\mathrm{b}^n \mid \mathrm{n} \geq 1\}$ is a context-free language. Example 11.53 will demonstrate that G is not regular. ■

A language $L(G)$ may contain words derivable from σ in more than one way. Accordingly, we make the following definition.

Ambiguous Grammar

A grammar G is **ambiguous** if a string in $L(G)$ has more than one derivation tree.

The next two examples present ambiguous grammars.

EXAMPLE 11.24 The following grammar G defines the syntax of simple algebraic expressions:

$$\begin{aligned}\langle\text{expression}\rangle &::= \langle\text{expression}\rangle\langle\text{sign}\rangle\langle\text{expression}\rangle \mid \langle\text{letter}\rangle\\ \langle\text{sign}\rangle &::= + \mid -\\ \langle\text{letter}\rangle &::= \mathrm{a} \mid \mathrm{b} \mid \mathrm{c} \mid \ldots \mid \mathrm{z}\end{aligned}$$

This grammar can produce the expression a − b + c two ways, as the derivation trees in Figure 11.11 show. As a result, G is an ambiguous grammar. ■

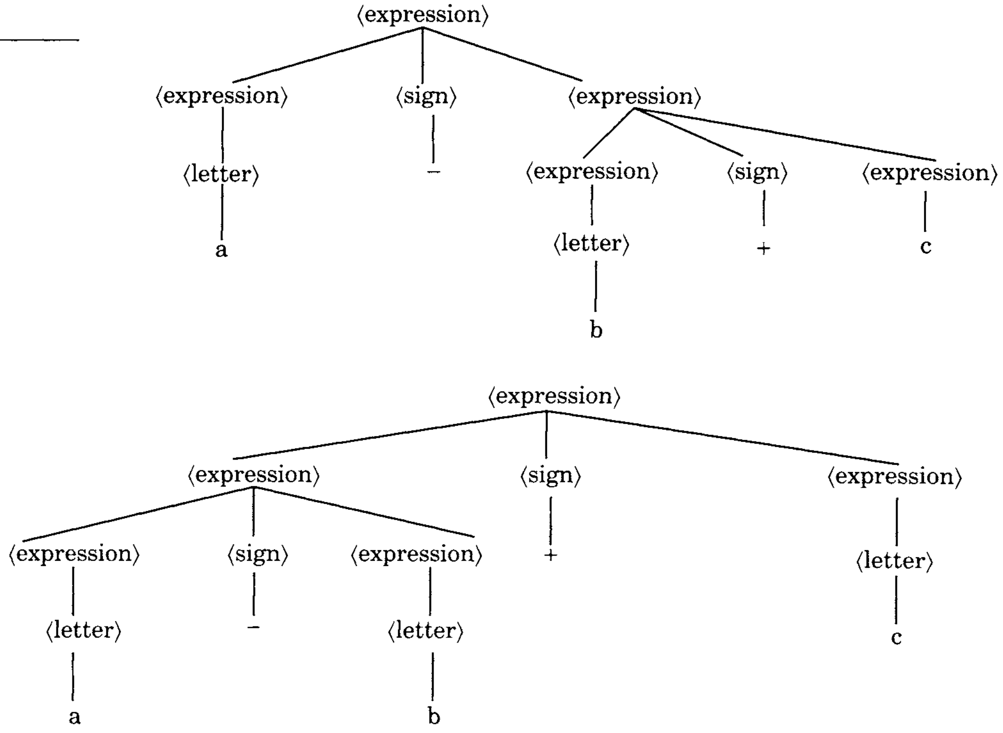
○ **EXAMPLE 11.25** (optional) The following are simplified production rules for an *if–then statement S*:

$$\begin{aligned}S &::= \text{if } \langle\text{expression}\rangle \text{ then } \langle\text{statement}\rangle \mid\\ &\qquad \text{if } \langle\text{expression}\rangle \text{ then } \langle\text{statement}\rangle \text{ else } \langle\text{statement}\rangle\\ \langle\text{expression}\rangle &::= E_1 \mid E_2\\ \langle\text{statement}\rangle &::= S_1 \mid S_2 \mid \text{ if } \langle\text{expression}\rangle \text{ then } \langle\text{statement}\rangle\end{aligned}$$

To see that these rules produce an ambiguous grammar, notice that the if–then statement

$$\textit{If } E_1 \textit{ then if } E_2 \textit{ then } S_1 \textit{ else } S_2 \qquad (11.3)$$

Figure 11.11



can be interpreted in two ways:

(i) If E_1 then (if E_2 then S_1 else S_2), or
(ii) If E_1 then (if E_2 then S_1) else S_2.

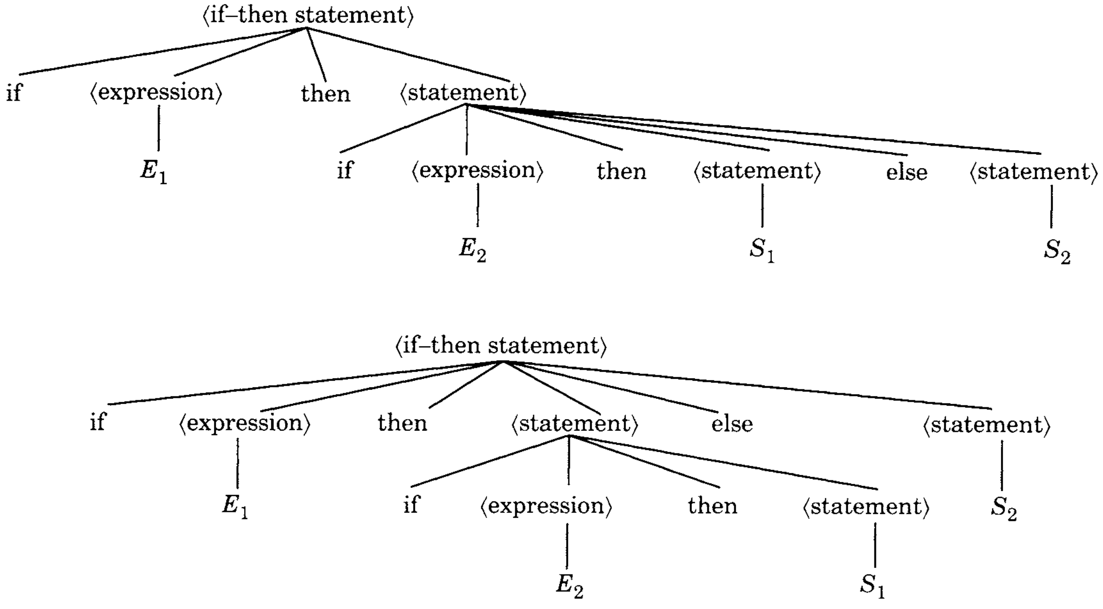
Using indentation, these possibilities can be displayed as follows:

```
(i)  if $E_1$ then                    (ii)  if $E_1$ then
        if $E_2$ then                          if $E_2$ then
           $S_1$                                  $S_1$
        else                                else
           $S_2$                                  $S_2$
```

Accordingly, statement (11.3) can be generated by two distinct derivation trees (see Figure 11.12).

To avoid this confusion, each **else** is paired with the nearest **if**. Consequently, statement (i) is the correct interpretation of statement (11.3). If you would like statement (11.3) to mean statement (ii), you have

Figure 11.12



two options:

```
if E1 then
   if E2 then
      S1
   else
else
   S2
```

```
if E1 then
          begin
            if E2 then
              S1
          end
else
    S2
```

■

The way a grammar produces its language of terminal and nonterminal symbols determines whether it is regular, context-free, or context-sensitive. The BNF notation facilitates such a differentiation.

Exercises 11.2

In the grammar $G = (N, T, P, \sigma)$, $N = \{\langle\text{sentence}\rangle, \langle\text{noun phrase}\rangle, \langle\text{verb}\rangle, \langle\text{object phrase}\rangle, \langle\text{article}\rangle, \langle\text{noun}\rangle\}$, $T = \{\text{a, the, cat, dog, chicken, milk, drinks, eats}\}$, $\sigma = \langle\text{sentence}\rangle$ and the production rules are:

$$\begin{aligned}\langle\text{sentence}\rangle &\to \langle\text{noun phrase}\rangle\langle\text{verb}\rangle\langle\text{object phrase}\rangle\\ \langle\text{noun phrase}\rangle &\to \langle\text{article}\rangle\langle\text{noun}\rangle\\ \langle\text{article}\rangle &\to a \mid \textit{the}\end{aligned}$$

$$\langle\text{noun}\rangle \to \textit{cat} \mid \textit{dog} \mid \textit{chicken} \mid \textit{milk}$$
$$\langle\text{verb}\rangle \to \textit{drinks} \mid \textit{eats}$$
$$\langle\text{object phrase}\rangle \to \langle\text{article}\rangle\langle\text{noun}\rangle$$

Determine if each is a valid sentence in L(G).

1. The cat drinks the milk. **2.** A chicken eats the dog.

3. The dog swallows the cat. **4.** The chicken drinks a rabbit.

Construct a derivation tree for each sentence in $L(G)$.

5. The cat eats the chicken. **6.** A dog drinks the milk.

With the grammar in Example 11.12, construct a derivation tree for each word in $L(G)$.

7. aa **8.** aba **9.** ab^2a **10.** ab^3a

Determine if each word belongs to the language generated by the grammar in Example 11.13.

11. aba **12.** abba **13.** a^3ba **14.** $a^2b^3a^4$

Use the grammar $G = (N, T, P, \sigma)$, where $N = \{A, \sigma\}$, $T = \{a, b\}$, and $P = \{\sigma \to a\sigma, \sigma \to aA, A \to b\}$, to answer Exercises 15–23.
Draw a derivation tree for each word in $L(G)$.

15. ab **16.** a^2b **17.** a^3b **18.** a^4b

Do the following words belong to $L(G)$?

19. aba **20.** abba **21.** a^3b **22.** a^5b

23. Identify the language $L(G)$.

Consider the grammar $G = (N, T, P, \sigma)$, where $N = \{\sigma\}$, $T = \{a, b\}$, and $P = \{\sigma \to a\sigma b, \sigma \to ab\}$. Determine if each word belongs to $L(G)$.

24. abba **25.** abab **26.** a^2b^2 **27.** a^3b^3

28. Identify the language $L(G)$.

Find the language generated by each grammar $G = (N, T, P, \sigma)$ where:

29. $N = \{\sigma, A, B\}$, $T = \{a, b\}$, $P = \{\sigma \to aA, A \to Bb, A \to a, B \to b\}$

30. $N = \{\sigma, A, B\}$, $T = \{a, b\}$, $P = \{\sigma \to aAa, A \to bBb, \sigma \to \lambda, A \to a, B \to a, B \to b\}$

Develop a grammar that generates each language over $\{0, 1\}$.

31. $\{1, 11, 1111, 11111111, \ldots\}$

32. $\{0, 00, 10, 100, 110, 0000, 1010, \ldots\}$

33. The set of words with prefix 00.

34. The set of words with suffix 11.

35. The set of binary representations of positive integers.

Create a grammar to produce each language over {a,b}.

36. $\{\mathrm{b}^n\mathrm{ab}^n \mid n \geq 0\}$ **37.** $\{\mathrm{a}^n\mathrm{b} \mid n \geq 1\}$ **38.** $\{\mathrm{a}^n\mathrm{ba} \mid n \geq 1\}$

39. $\{\mathrm{a}^m\mathrm{b}^n \mid m, n \geq 1\}$ **40.** The set of palindromes.

○ Using Example 11.18, draw the derivation tree for each integer.

41. 234 **42.** −234

○ **43.** An identifier in Java is a letter, underscore, or $, followed by any number of alphanumeric characters. With BNF, define the grammar for a Java identifier.

Use the grammar in Exercise 43 to see if each string is a valid Java identifier.

○ **44.** catch 22 **45.** 20/20 **46.** algorist **47.** three roots

Construct a derivation tree for each identifier.

○ **48.** result2 **49.** value **50.** R2D2 **51.** math

The production rules of a grammar for simple arithmetic expressions are:

$$\begin{aligned}\langle\text{expression}\rangle &::= \langle\text{digit}\rangle \mid (\langle\text{expression}\rangle) \mid +(\langle\text{expression}\rangle) \mid \\ &\quad -(\langle\text{expression}\rangle) \mid \langle\text{expression}\rangle\langle\text{operator}\rangle\langle\text{expression}\rangle \\ \langle\text{digit}\rangle &::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\ \langle\text{operator}\rangle &::= + \mid - \mid * \mid / \mid \uparrow\end{aligned}$$

Use this grammar to answer Exercises 52–59.
Determine if each is a valid arithmetic expression.

52. $2 * 3 + 4$ **53.** $-(3 * 4 \uparrow 5)$ **54.** $3 + \uparrow 7$ **55.** $6 + 5/8*$

Construct a derivation tree for each expression.

56. $3 + 5 * 6$ **57.** $5 + (4 \uparrow 3)$ **58.** $(5 + 3) - 7/4$ **59.** $-(3 \uparrow (5 + 2))$

A number in ALGOL (excluding the exponential form) is defined as follows:

$$\begin{aligned}\langle\text{number}\rangle &::= \langle\text{decimal number}\rangle \mid \langle\text{sign}\rangle\langle\text{decimal number}\rangle \\ \langle\text{decimal number}\rangle &::= \langle\text{unsigned integer}\rangle \mid .\langle\text{unsigned integer}\rangle \mid \\ &\quad \langle\text{unsigned integer}\rangle.\langle\text{unsigned integer}\rangle\end{aligned}$$

⟨unsigned integer⟩ ::= ⟨digit⟩ | ⟨unsigned integer⟩⟨digit⟩

⟨digit⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

⟨sign⟩ ::= + | −

Use this grammar to answer Exercises 60–67.

○ Determine if each is a valid ALGOL number.

60. 234 **61.** 2.34 **62.** 234. **63.** .234

Draw a derivation tree for each ALGOL number.

○ **64.** −3.76 **65.** +376 **66.** .376 **67.** 0.23

For Exercises 68–73, use the following definition of a simple algebraic expression:

⟨expression⟩ ::= ⟨term⟩ | ⟨sign⟩⟨term⟩ |
⟨expression⟩⟨adding operator⟩⟨term⟩

⟨sign⟩ ::= + | −

⟨adding operator⟩ ::= + | −

⟨term⟩ ::= ⟨factor⟩ |
⟨term⟩⟨multiplying operator⟩⟨factor⟩

⟨multiplying operator⟩ ::= * | /

⟨factor⟩ ::= ⟨letter⟩ | (⟨expression⟩)|⟨expression⟩

⟨letter⟩ ::= a | b | c | . . . | z

○ Determine if each is a legal expression.

68. a + b * (c/d) **69.** a + b + c **70.** −a * b/c + d **71.** ((a − b) + c)

○ Construct a derivation tree for each expression.

72. (a * b) + c/d **73.** a * (b + c/d)

74. Use BNF to define a grammar for the language of well-formed parentheses (wfp).

Use the grammar in Exercise 74 to see if each is a valid sequence of parentheses.

75. (()) **76.** ()(()) **77.** (()()) **78.** ()()()

79. Figures 11.13 and 11.14 diagram the syntax for an unsigned integer and an unsigned number, respectively. Define the grammar for an unsigned number in BNF.

Figure 11.13

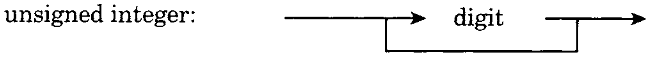
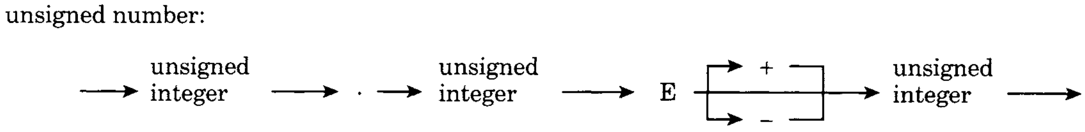


Figure 11.14



○ Using the grammar in Exercise 79, check if each is a valid unsigned number.

80. 177.76 **81.** .1776 **82.** 1776. **83.** 17.76E-2

11.3 Finite-State Automata

This section presents an abstract model of a machine that accepts input values, but produces no output values.

Often the question arises whether or not a word over an alphabet is acceptable. For example, is 2R2D an acceptable identifier or is 17.06 a valid real number in C++? Finite-state automata can model the steps in determining if a given word exists in a language. Accordingly, finite-state automata, also known as **language recognizers**, play a central role in the development of compilers.

Before we study the definition, we present a simple example of a language recognizer.

EXAMPLE 11.26 Determining if an input string over the alphabet {a, b} contains *abba* as a substring involves the following five steps:

Step 0 If the first symbol in the string is *a*, move to step 1 and look for the character *b*. Otherwise, no progress has been made.

Step 1 If the next character is *b*, the substring *ab* has occurred, so go to step 2 and look for another *b*. Otherwise, the symbol *b* is still missing, so stay in step 1.

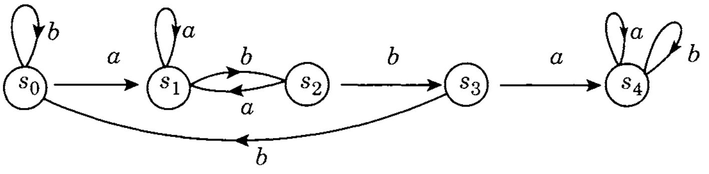
Step 2 If the next symbol is *b*, the substring *abb* exists; go to step 3; if *a*, return to step 1.

Step 3 If the next symbol is *a*, the given input string contains the substring *abba*; otherwise, return to step 0 and start all over again.

Step 4 Once the substring *abba* has occurred in the input string, any sequence of *a*'s and *b*'s may follow.

These steps can be represented by a digraph (see Figure 11.15), each vertex representing a step. Exactly two edges, labeled *a* or *b*, leave each vertex.

Figure 11.15



To determine the action required from a given step, simply follow the directed edges from the corresponding vertex. For example, at vertex s_3 (step 3) if the next input symbol is a, move to vertex s_4 (step 4); otherwise, return to vertex s_0 (step 0). The other (labeled) edges are interpreted similarly.

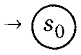
The digraph indicates a string contains *abba* as a substring if and only if the directed path the string determines terminates at vertex s_4. The string *abab* determines the path s_0-s_1-s_2-s_1-s_2, which does not end at s_4; consequently, *abab* is not acceptable. On the other hand, the string *ababbab* determines the path s_0-s_1-s_2-s_1-s_2-s_3-s_4-s_4, which terminates at s_4; so the string does have the desired property. ■

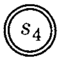
The digraph in Figure 11.15 displays a **finite-state automaton**. (*Automaton* is the singular form of *automata*.) Its five vertices, s_0 through s_4, are the **states** of the automaton. Since the whole process begins at s_0 (step 0), s_0 is the **initial state**. A string is acceptable, that is, contains *abba* as a substring, if and only if its path ends at s_4; accordingly, s_4 is an **accepting state**.

The digraph shows the transition of the machine between states. For example, if the automaton is at state s_2 and the input symbol is a, the automaton switches its state to s_1. The digraph is the **transition diagram** of the finite-state automaton.

The initial state is customarily identified by an arrow pointing to it and an accepting state by two concentric circles, as Figure 11.16 shows. The transition diagram appears in Figure 11.17.

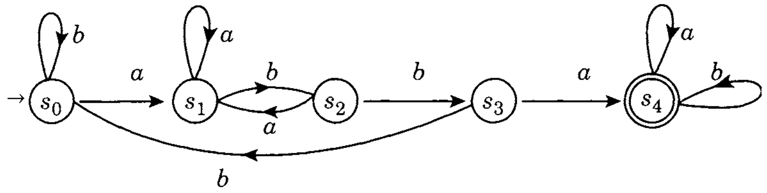
Figure 11.16





The initial state An accepting state

Figure 11.17



Each state s_i and an input symbol determine a unique state s_j. So we can define a function $f : S \times I \to S$ as follows, where $S = \{s_0, s_1, s_2, s_3, s_4\}$, the set of states, and $I = \{a, b\}$, the input alphabet:

$f(s_0, \mathrm{a}) = s_1$ $f(s_0, \mathrm{b}) = s_0$ $f(s_1, \mathrm{a}) = s_1$ $f(s_1, \mathrm{b}) = s_2$
$f(s_2, \mathrm{a}) = s_1$ $f(s_2, \mathrm{b}) = s_3$ $f(s_3, \mathrm{a}) = s_4$ $f(s_3, \mathrm{b}) = s_0$
$f(s_4, \mathrm{a}) = s_4$ $f(s_4, \mathrm{b}) = s_4$

The function f is the **transition function** of the finite-state automaton. It can also be defined by the **transition table** in Table 11.1.

Table 11.1

State	**Input symbol**	
	a	**b**
s_0	s_1	s_0
s_1	s_1	s_2
s_2	s_1	s_3
s_3	s_4	s_0
s_4	s_4	s_4

We are now ready to define a finite-state automaton.

Finite-State Automaton

A **finite-state automaton** (FSA), M, manifests five characteristics:

- A finite set, S, of **states** of the automaton.
- A specially designated state, s_0, called the **initial state**.
- A subset A of S, consisting of the **accepting states** (or **final states**) of the automaton.
- A finite set, I, of **input symbols**.
- A function $f : S \times I \to S$, called the **transition function** or the **next-state function**.

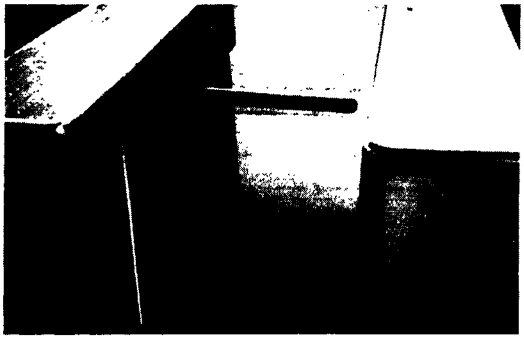
In symbols, $M = (S, A, I, f, s_0)$.

For instance, for the FSA in Example 11.26, $S = \{s_0, s_1, s_2, s_3, s_4\}$, $A = \{s_4\}$, $I = \{a, b\}$, and the transition function f is defined by Table 11.1.

New York City subway commuters use an FSA everyday, as the next example shows.

EXAMPLE 11.27 A turnstile in the subway entrance contains four arms at waist level (Figure 11.18). Initially, it is locked so that the arms cannot be moved. Depositing a token into the slot, however, unlocks it and allows the arms to rotate through one quarter of a complete turn, so the commuter passes through the turnstile.

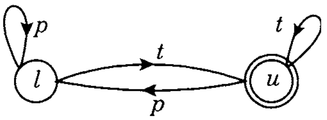
Figure 11.18



The turnstile has two states: locked (l) and unlocked (u). Depositing a token (t) shifts the turnstile from the locked state to the unlocked state and no matter how many times the commuter inputs t, the turnstile remains in the same state. Pushing (p), the arms, takes the turnstile back to the locked state. Once it is in the locked state, it remains there regardless of how many times the commuter pushes the arms; that is, regardless of the number of times he inputs p into the device.

The turnstile exemplifies an FSA. Figure 11.19 shows its transition diagram.

Figure 11.19



■

The next two examples draw transition diagrams of FSAs from their algebraic definitions.

EXAMPLE 11.28 Draw the transition diagram of the FSA $M = (S, A, I, f, s_0)$, where $S = \{s_0, s_1, s_2\}$, $A = \{s_2\}$, $I = \{a, b\}$, and the transition function f is defined by

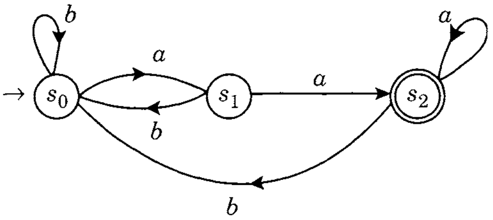
$$f(s_0, a) = s_1, \quad f(s_0, b) = s_0, \quad f(s_1, a) = s_2,$$
$$f(s_1, b) = s_0, \quad f(s_2, a) = s_2, \quad f(s_2, b) = s_0.$$

*Based on B. Hayes, "On the Finite-State Machine, A Minimal Model of Mousetraps, Ribosomes, and the Human Soul," *Scientific American*, Vol. 249 (Dec. 1983), pp. 20–28, 178.

SOLUTION:

The FSA contains three states — s_0, s_1, and s_2 — with s_2 the only accepting state. Since there are two input symbols, exactly two edges leave each vertex. Draw a directed edge from state s_i to state s_j if there is an input symbol x such that $f(s_i,x) = s_j$; then label the edge x. For example, since $f(s_1,b) = s_0$, a directed edge runs from s_1 to s_0 labeled b. Figure 11.20 shows the resulting transition diagram.

Figure 11.20



■

EXAMPLE 11.29

Draw the transition diagram of the FSA $M = (S, A, I, f, s_0)$, where $S = \{s_0, s_1, s_2, s_3, s_4\}$, $A = \{s_2\}$, $I = \{\text{a, b, c}\}$, and f is defined by Table 11.2.

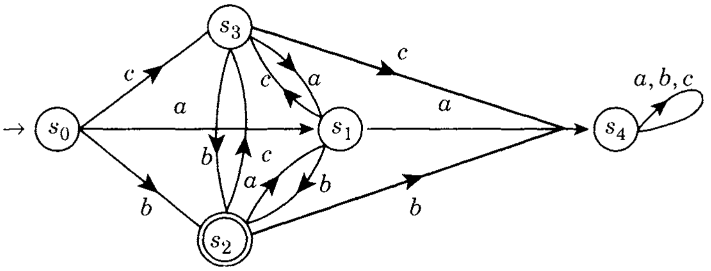
Table 11.2

S \ I	a	b	c
s_0	s_1	s_2	s_3
s_1	s_4	s_2	s_3
s_2	s_1	s_4	s_3
s_3	s_1	s_2	s_4
s_4	s_4	s_4	s_4

SOLUTION:

The automaton contains five states, with s_2 the only accepting one. Since there are three input symbols, three edges originate from every state. Draw a directed edge from state s_i to state s_j if there exists an input symbol x such that $f(s_i,x) = s_j$. For instance, $f(s_1,c) = s_3$, so a directed edge labeled c runs from state s_1 to state s_3. Figure 11.21 displays the resulting transition diagram, where, for convenience, the three loops at s_4 appear as a single loop with labels a, b, and c.

Figure 11.21



■

Suppose a string is input into an FSA. If the path it determines ends at an accepting state, the string is **accepted** (or **recognized**) by the automaton; otherwise, it is **rejected** by the automaton.

EXAMPLE 11.30 Determine if the strings a^3b^2ab and ab^3a are accepted by the FSA in Figure 11.17.

SOLUTION:
First, find the path determined by the string and check if it terminates at s_4, the accepting state. (Recall that a^3b^2ab = aaabbab.) Begin at the initial state, s_0. When a is input, move to state s_1. Every time a is input, remain there, so the path defined by aaa is s_0-s_1-s_1-s_1. When b is input, transfer to state s_2. The path obtained thus far is s_0-s_1-s_1-s_1-s_2. Now b moves to s_3 and a to s_4, yielding the path s_0-s_1-s_1-s_1-s_2-s_3-s_4. Once in s_4, remain there no matter what the input is. Thus the path determined by the given string is s_0-s_1-s_1-s_1-s_2-s_3-s_4-s_4. Since it terminates at s_4, the FSA accepts the given word.

Notice that the path determined by the string ab^3a is s_0-s_1-s_2-s_3-s_0-s_1, and it does not end at the accepting state s_4; consequently, the automaton rejects the string. ■

Two different FSAs may accept the same language over an alphabet. This occurrence requires that we make a new definition.

Equivalent Finite-State Automata

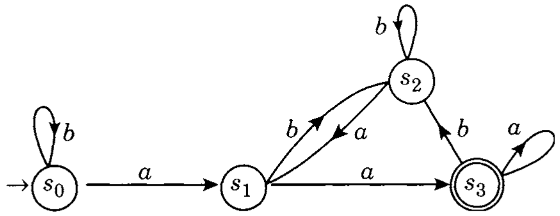
The set of words accepted by an FSA, M, is the **language accepted** (or **recognized**) by M and is denoted by $L(M)$. Two finite-state automata, M and M', are **equivalent** if they recognize the same language: $L(M) = L(M')$.

EXAMPLE 11.31 Identify the language $L(M)$ accepted by the automaton M in Figure 11.20.

SOLUTION:
Look for paths beginning at s_0 and terminating at s_2. $L(M)$ consists of all words over {a, b} that end in aa. ■

EXAMPLE 11.32 By Example 11.31, the automaton in Figure 11.20 accepts the language of words over {a, b} ending in aa. You may verify that the FSA in Figure 11.22 accepts the same language. Consequently, the automata in Figures 11.20 and 11.22 are equivalent.

Figure 11.22



■

The next four examples build FSAs with desired properties, as Example 11.26 did.

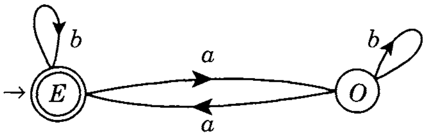
EXAMPLE 11.33

Design an FSA that accepts words over $I = \{a, b\}$ containing an even number of *a*'s.

SOLUTION:

Every word over I contains either an even number of *a*'s (E) or an odd number of *a*'s (O), so the automaton has two states, E and O, E being the accepting state. Initially, the number of *a*'s in the word is zero, an even integer; E is the initial state of the automaton. If the automaton is at E and an *a* is input, it moves to state O. If it is at O and an *a* is input, it moves to state E. Figure 11.23 shows the transition diagram of the FSA.

Figure 11.23



A word over I has **even parity** if it contains an even number of *a*'s and **odd parity** if an odd number. Since the automaton in Example 11.33 determines whether a word has even or odd parity, it is called a **parity-check machine**. ■

EXAMPLE 11.34

Design an FSA accepting words over $\{a, b\}$ that begin with *aa* and end in *bb*.

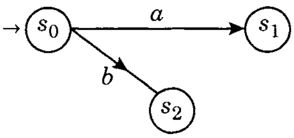
SOLUTION:

We build the automaton step by step:

Step 0 Initially, the automaton is at the initial state s_0.

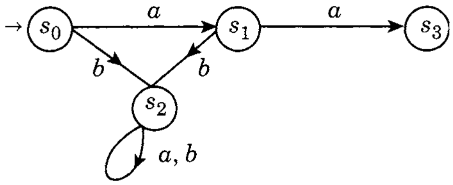
Step 1 If the first symbol is *a*, move to state s_1 from s_0 and wait for the next symbol. But if the first symbol is *b*, the word is not acceptable (state s_2). See Figure 11.24.

Figure 11.24



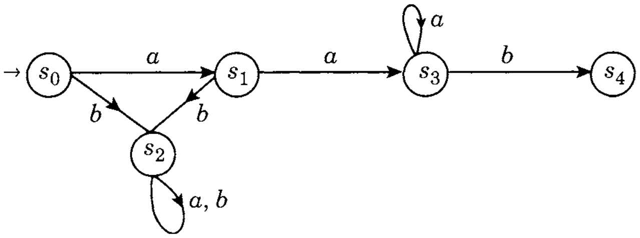
Step 2 If the input symbol at s_1 is *a*, move to state s_3 and determine whether the string ends with *bb*. On the other hand, if the input symbol at s_1 is *b*, move to s_2 to trap such unacceptable words. Once at s_2, remain there no matter what the input symbol is. See Figure 11.25.

Figure 11.25



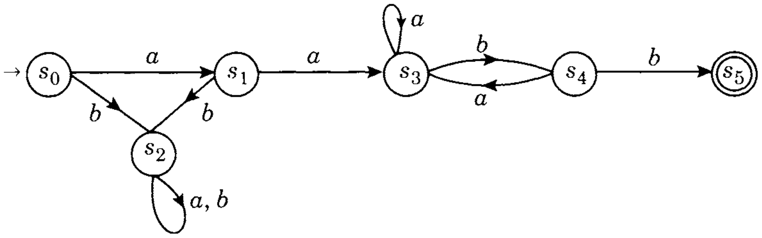
Step 3 Every word that triggers a move from s_0 to s_3 begins with aa. Any number of a's can follow it (see the loop at s_3 in Figure 11.26). However, if b follows the word, move to a new state s_4, as in Figure 11.26.

Figure 11.26



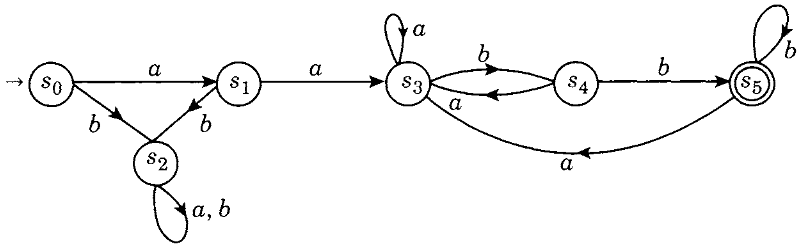
Step 4 If the input symbol at s_4 is a, return to s_3 and look for the pair bb. But if it is b, move to a new state s_5. See Figure 11.27.

Figure 11.27



Step 5 Once at s_5, any number of b's may occur. However, if the input symbol at s_5 is a, return to s_3 to look for bb. Since words ending in bb are acceptable, s_5 is the accepting state. These six steps create the FSA in Figure 11.28.

Figure 11.28



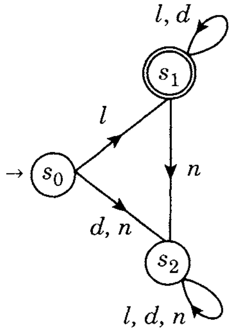
■

EXAMPLE 11.35 (optional) An identifier in a programming language consists of a letter followed by any number of alphanumeric characters (Section 11.1). Design an FSA that recognizes such legal identifiers.

SOLUTION:

Let I denote the set of all characters in the alphabet recognizable by a compiler. Let l denote a letter, d a digit, and n any nonalphanumeric character. The automaton will have three states: s_0, s_1, and s_2. State s_2 traps all invalid strings. (Accordingly, it is called a **trap state** or a **dump state**.) The resulting automaton appears in Figure 11.29.

Figure 11.29



■

The FSA in Figure 11.29 can be translated into an algorithm which determines if a sequence of characters is a legal identifier. See Algorithm 11.1.

```
Algorithm identifier
(* This algorithm determines whether a sequence of characters is a
   valid identifier, using the FSA in Figure 11.29. All characters
   are read from the same input line. Symbol denotes an arbitrary
   character; state denotes an arbitrary state; state0, statel,
   and state2 denote the various states of the FSA. state2 is a
   dump state. *)
 Begin (* algorithm *)
   state ← state0     (* initialize state *)
   read(symbol)
   while not at the end of the current line
   begin
   case state of
     state0:  if symbol is a letter then
                state ← statel
              else     (* invalid sequence; dump it. *)
                state ← state2
     statel:  if symbol is a letter or a digit then
                state ← statel
              else
                state ← state 2
     state2:  (* do nothing; stay there. *)
  read(symbol)
endwhile
```

```
if state = statel then
   the sequence is a valid identifier
else
   the sequence is an invalid identifier
End (* algorithm *)
```

Algorithm 11.1

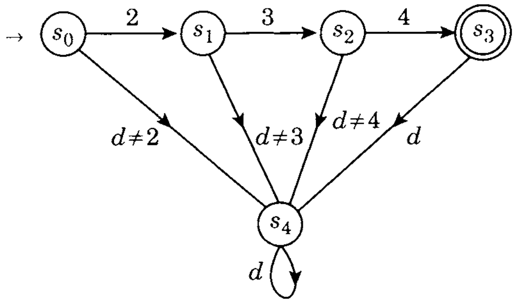
With a trap state, an FSA can simulate an automatic teller machine, or ATM, which is widely used because it allows bank customers to make transactions without human intervention, as the following example demonstrates.

EXAMPLE 11.36 After a bank customer inserts his bank card into the ATM, it requests him to input his secret identification number (ID). Suppose the ID is 234. Design an FSA that models the ATM.

SOLUTION:

The input to the automaton contains three digits d. It has five states: s_0 (the initial state, waiting for the first digit in the ID), s_1 (the first digit is correct; now waiting for the second digit), s_2 (the second digit is correct; waiting for the third digit), s_3 (the third digit is correct), and s_4 (the trap state that captures all invalid ID's). The ensuing FSA is shown in Figure 11.30.

Figure 11.30



■

The salient characteristics of an FSA have emerged through its many applications to ATMs, programming languages, parity checks, and subway turnstiles. Every FSA manifests an input set, a transition function, and a finite number of states.

Exercises 11.3

Using the FSA in Figure 11.17, identify the directed paths determined by each input string.

1. a^3b **2.** abab **3.** ab^3 **4.** a^2b^3a

With the FSA in Figure 11.21, identify the directed path determined by each word:

5. abcab **6.** caba^2 **7.** a^2bc^3 **8.** ab^2c^3

Determine if each word is acceptable by the FSA in Figure 11.17.

9. ab^3 **10.** $\text{a}^2\text{b}^2\text{a}^2$ **11.** $\text{a}^3\text{b}^2\text{a}^3$ **12.** $\text{ab}^4\text{ab}^2\text{ab}$

Determine if the FSA in Figure 11.21 recognizes each word.

13. abcabc **14.** abacbc **15.** ab^4c^3 **16.** ab^5c^6

Draw the transition diagram of the FSA, $M = (S, A, I, f, s_0)$, where $I = \{\text{a,b}\}$, and:

17. $S = \{s_0, s_1, s_2\}$, $A = \{s_2\}$

$f(s_0, \text{a}) = s_0$ $f(s_0, \text{b}) = s_1$ $f(s_1, \text{a}) = s_0$ $f(s_1, \text{b}) = s_2$
$f(s_2, \text{a}) = s_0$ $f(s_2, \text{b}) = s_2$

18. $S = \{s_0, s_1, s_2, s_3\}$, $A = \{s_3\}$

$f(s_0, \text{a}) = s_1$ $f(s_0, \text{b}) = s_0$ $f(s_1, \text{a}) = s_1$ $f(s_1, \text{b}) = s_2$
$f(s_2, \text{a}) = s_1$ $f(s_2, \text{b}) = s_3$ $f(s_3, \text{a}) = s_1$ $f(s_3, \text{b}) = s_0$

19. $S = \{s_0, s_1, s_2, s_3\}$, $A = \{s_2\}$

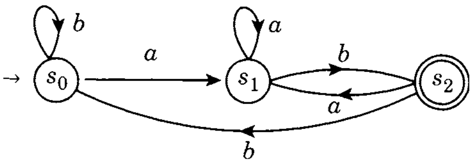
	f	
$S \backslash I$	**a**	**b**
s_0	s_0	s_1
s_1	s_1	s_2
s_2	s_2	s_3
s_3	s_3	s_3

20. $S = \{s_0, s_1, s_2, s_3, s_4\}$, $A = \{s_3\}$

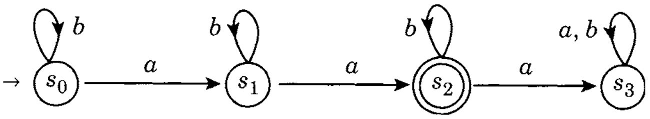
	f	
$S \backslash I$	**a**	**b**
s_0	s_1	s_4
s_1	s_4	s_2
s_2	s_3	s_4
s_3	s_3	s_3
s_4	s_4	s_4

Construct a transition table for each FSA.

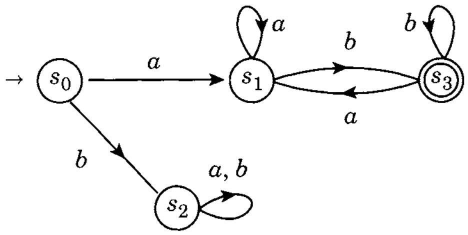
21.



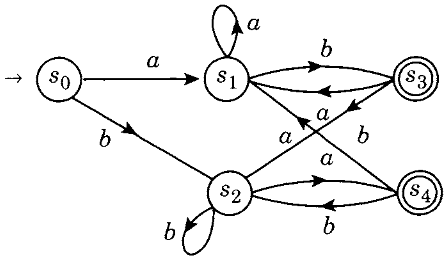
22.



23.

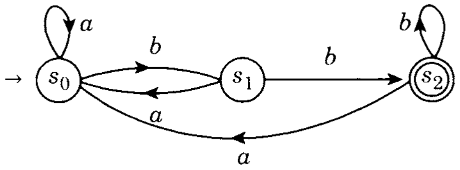


24.

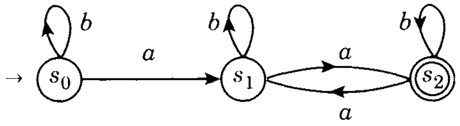


Characterize the language recognized by the FSAs in Exercises 25–35.

25.

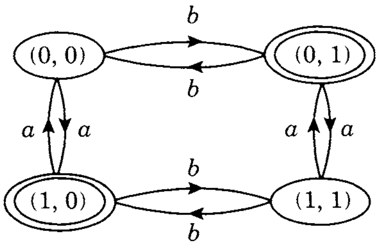


26.



27–34. The finite-state automata in Exercises 17–24.

***35.**



Let m denote the number of a's in a string. Design an FSA that accepts strings over {a, b} which:

36. Contain exactly one a.

37. Begin with aa.

38. Contain aba as a substring.

39. Contain aaa as a substring.

40. Begin with aa or bb.

41. Contain $baab$ as a substring.

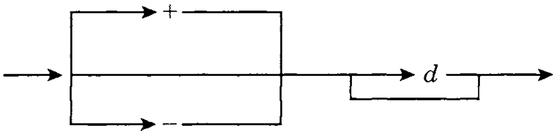
42. Have $m \equiv 0 (\text{mod } 3)$. **43.** Have $m \equiv 2 (\text{mod } 3)$.

44. Simulate an automatic teller machine by means of an FSA that accepts 1776 as a valid identification number.

45. Design an FSA to model an automatic teller machine that accepts 23 or 45 as a valid identification number.

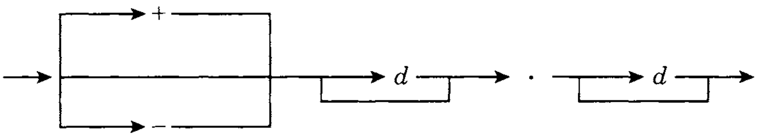
○ **46.** An integer is a nonempty string of digits, preceded by an optional sign (+ or −). See the syntax diagram in Figure 11.31. Design an FSA that recognizes integers.

Figure 11.31



○ **47.** A real number, excluding the exponential form, consists of an optional sign (+ or −) followed by one or more digits, a decimal point, and one or more digits. (See the syntax diagram in Figure 11.32.) Design an FSA that recognizes such real numbers.

Figure 11.32



48. Write an algorithm to implement an automatic teller machine as an FSA that accepts 234 as a valid identification number.

○ **49.** Write an algorithm to determine if a sequence of characters represents a valid integer.

○ **50.** Write an algorithm to determine if a sequence of characters represents a valid real number. Exclude the exponential form.

11.4 Finite-State Machines

As a generalization of FSAs, finite-state machines abstractly model computing machines. In an FSA, movements from state s_i to state s_j depend on the input at s_i, and no output emerges. But as a finite-state machine moves from state s_i to state s_j, an output does emerge. Consequently, a finite-state machine possesses two features not required of an FSA: a finite set O of output symbols and an output function $g : S \times I \to O$, where I is the input alphabet. (An accepting state cannot exist here because a word is not being checked for certain characteristics.) The output depends on two things: the current state and the input symbol.

Finite-State Machine

A **finite-state machine** (FSM), M, bears six characteristics:

- A finite set, S, of **states**;
- A finite **input alphabet,** I;
- A finite set, O, of **output symbols**;
- A **transition function,** $f : S \times I \rightarrow S$;
- An **output function,** $g : S \times I \rightarrow O$;
- An **initial state,** s_0.

In symbols, $M = (S, I, O, f, g, s_0)$.

In this definition, the output function g depends on both the state of the machine and the current input. Such FSMs are called **Mealy machines,** after George H. Mealy, who introduced them in 1955. (Another type of FSM appears in the Supplementary Exercises.)

EXAMPLE 11.37 Let $S = \{s_0, s_1, s_2\}$, $I = \{a, b\}$, and $O = \{0, 1\}$. Define functions $f : S \times I \rightarrow S$ and $g : S \times I \rightarrow O$ by means of Table 11.3. For example, $f(s_0, b) = s_1$, $f(s_2, b) = s_1$, $g(s_0, b) = 1$, and $g(s_2, b) = 1$.

Table 11.3

	f		g	
$S \backslash I$	**a**	**b**	**a**	**b**
s_0	s_0	s_1	0	1
s_1	s_1	s_2	1	0
s_2	s_2	s_1	1	1

Then $M = (S, I, O, f, g, s_0)$ is an FSM with transition function f and output function g. Table 11.3 is the **transition table** of the machine. ■

Like an FSA, an FSM can be represented by a **transition diagram,** with one main difference: every directed edge (s_j, s_k) has two labels. One indicates the input symbol i; the other the output o from entering i into state s_j. For instance, if $f(s_j, i) = s_k$ and $g(s_j, i) = 0$, the directed edge (s_j, s_k) is labeled $i/0$.

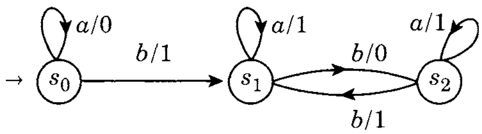
The next example illustrates how to draw transition diagrams of FSMs.

EXAMPLE 11.38

Draw the transition diagram for the FSM in Example 11.37.

SOLUTION:
The FSM has three states — s_0, s_1, and s_2; and two input symbols — a and b; two output symbols–0 and 1. Two input symbols produce exactly two outgoing edges for each state. Each directed edge $(s_j,\ s_k)$ in the diagram is labeled i/o, where $f(s_j, i) = s_k$ and $g(s_j, i) = o$. For instance, since $f(s_0, a) = s_0$ and $g(s_0, a) = 0$, a loop exists at s_0 labeled $a/0$. And because $f(s_0,\text{b}) = s_1$ and $g(s_0,\text{b}) = 1$, the edge (s_0, s_1) is labeled $b/1$. The other directed edges carry similar labels. Figure 11.33 shows the transition diagram produced by this process.

Figure 11.33



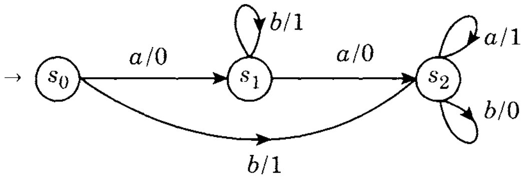
■

The transition diagram of an FSM can generate the transition table, as the following example demonstrates.

EXAMPLE 11.39

Construct the transition table of the FSM in Figure 11.34.

Figure 11.34



SOLUTION:
From the transition diagram, $f(s_0, \text{a}) = s_1$, $f(s_0, \text{b}) = s_2$, $f(s_1, \text{a}) = s_2$, $f(s_1, \text{b}) = s_1$, $f(s_2, \text{a}) = s_2$, and $f(s_2, \text{b}) = s_2$; also $g(s_0, \text{a}) = 0$, $g(s_0, \text{b}) = 1$, $g(s_1, \text{a}) = 0$, $g(s_1, \text{b}) = 1$, $g(s_2, \text{a}) = 1$, and $g(s_2, \text{b}) = 0$. These values generate the transition table in Table 11.4.

Table 11.4

	f		g	
$S \setminus I$	**a**	**b**	**a**	**b**
s_0	s_1	s_2	0	1
s_1	s_2	s_1	0	1
s_2	s_2	s_2	1	0

■

Suppose we input the string $x = x_1x_2\ldots x_n$ into an FSM. Suppose further that there exist states s_{i-1} and s_i, and an output y_i such that $f(s_{i-1}, x_i) = s_i$ and $g(s_{i-1}, x_i) = y_i$ for every i. Then $y_1y_2\ldots y_n$ is the output **produced** by the machine for the input x.

EXAMPLE 11.40 Find the output of the FSM in Figure 11.33 for the input string *abbaba*.

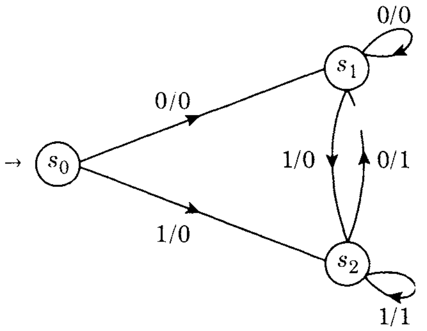
SOLUTION:
Start at state s_0. When a is input, stay at s_0 with output 0. When the next symbol b is input, move to s_1 and produce the output 1. When the third symbol b is input at s_1, move to s_2 and output 0. Continuing like this yields the output 010111. ■

The next two examples present FSMs useful in electronics. These machines have limited memory: at each state they must remember the previous input.

EXAMPLE 11.41 Let $I = O = \{0, 1\}$. A **unit delay machine**, an FSM $M = (S, I, O, f, g, s_0)$, delays an input string by unit time. When the string $x_1x_2\ldots x_n$ is input, it produces $0x_1x_2\ldots x_n$ as the output. Construct such a machine.

SOLUTION:
Since each state has two possible outputs, each has two outgoing edges. The machine must certainly have an initial state s_0. With the first output always 0, both edges leaving s_0 must yield 0. The machine must remember whether the previous input was 0 or 1; this requires two additional states, s_1 and s_2. If the previous input was 0, the machine moves to state s_1 and outputs 0; if it was 1, it moves to state s_2 and outputs 1. Figure 11.35 shows the transition diagram of this FSM.

Figure 11.35



For instance, the input 101110 yields the output 010111 (Verify this.), which has lost the trailing zero of the input. By appending a 0, however, to the input, that is, by inputing 1011100, the desired output results: 0101110. Deleting the leading 0 yields an exact copy of the input. ■

EXAMPLE 11.42 Design an FSM that adds two binary integers, x and y.

SOLUTION:
Assume, for convenience, x and y contain the same number of bits, and the leftmost bits are zeros. Thus, let $x = (x_nx_{n-1}\ldots x_1x_0)_{\text{two}}$ and $y = (y_ny_{n-1}\ldots y_1y_0)_{\text{two}}$, where $x_n = y_n = 0$. Add the corresponding bits x_i and

y_i from right to left, as usual. Adding x_i and y_i yields a sum bit z_i and a carry bit c_i:

$$z_i = (x_i + y_i) \bmod 2 \text{ and } c_i = (x_i + y_i) \operatorname{div} 2$$

For instance, adding the bits 1 and 1 gives the sum bit 0 and the carry bit 1. Tables 11.5 and 11.6 display the sum and carry bits for paired values of x_i and y_i.

Table 11.5

Sum bits.

		y_i	
		0	**1**
x_i	0	0	1
	1	1	0

Table 11.6

Carry bits.

		y_i	
		0	**1**
x_i	0	0	0
	1	0	1

Any two binary numbers can be added if the pairs 00, 01, 10, and 11 can be. When two bits x_i and y_i are added, the carry is 0 or 1. Consequently, a machine can be manufactured with two states: $c0$ (carry is 0) and $c1$ (carry is 1). Since at first the carry is 0, $c0$ is the initial state of the machine (Figure 11.36).

Figure 11.36

Since four bit-pairs exist, exactly four edges leave each state. Tables 11.5 and 11.6 can find the state following a given state and the output from a given input. For instance, if at state $c0$ and input 11, output 0 and move to state $c1$ (Figure 11.37). If at state $c1$ and input 10, output 0 and remain at state $c1$ (Figure 11.38). Continuing like this produces the transition diagram in Figure 11.39.

Figure 11.37

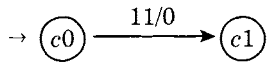


Figure 11.38

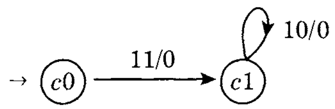
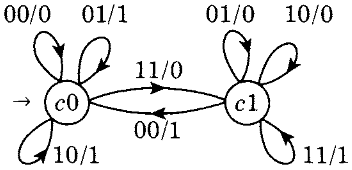


Figure 11.39



■

Finally, every FSA is a special case of an FSM. To see this, label all incoming edges to each accepting state with output 1 and all incoming edges to each nonaccepting state with output 0. Consequently, an input string is **accepted** by the FSM if and only if the last output of the machine is 1, as the following example illustrates.

EXAMPLE 11.43

Example 11.26 showed that the FSA in Figure 11.40 accepts a string over {a, b} if and only if the string contains *abba* as a substring. To convert the automaton into an FSM, add an output to every edge. Each incoming edge to the accepting state s_4 is labeled with output 1, and every incoming edge to other edges 0. The resulting FSM appears in Figure 11.41.

Figure 11.40

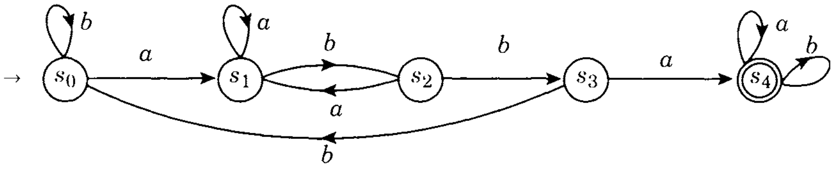
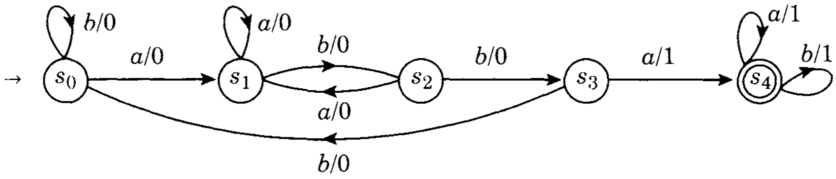


Figure 11.41



According to Example 11.30, the word a^3b^2ab is accepted by the automaton in Figure 11.40. The machine in Figure 11.41 verifies this: the substring a^3 takes the machine from s_0 to s_1 and it outputs 0 three times, b^2 takes it to s_3 and it outputs 0 twice, a takes it from s_3 to s_4 and it outputs 1; b takes the machine from s_4 to itself and it outputs 1. With the last output 1, the string is accepted by the FSM, as expected. ■

As this example indicates, FSMs like Mealy machines add output to the FSA configuration. This means that we can use them in such fields as electronics, in addition to using their transition tables and diagrams as definitional models.

Exercises 11.4

Using the FSM in Figure 11.33, evaluate each.

1. $f(s_1, \mathrm{a})$ **2.** $f(s_2, \mathrm{b})$ **3.** $f(s_0, \mathrm{b})$ **4.** $f(s_2, \mathrm{a})$

5. $g(s_1, \mathrm{b})$ **6.** $g(s_2, \mathrm{b})$ **7.** $g(s_0, \mathrm{b})$ **8.** $g(s_2, \mathrm{a})$

Draw the transition diagram of the FSM with each transition table.

9.

	f		*g*	
S \ *I*	a	b	a	b
s_0	s_0	s_1	1	0
s_1	s_1	s_2	0	0
s_2	s_0	s_1	1	1

10.

	f		*g*	
S \ *I*	a	b	a	b
s_0	s_1	s_1	0	1
s_1	s_1	s_2	1	0
s_2	s_1	s_2	0	1

11.

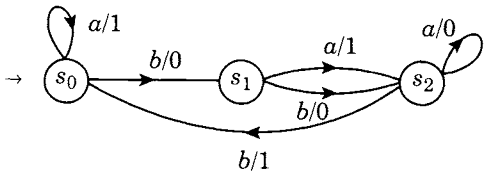
	f		*g*	
S \ *I*	a	b	a	b
s_0	s_1	s_1	0	0
s_1	s_1	s_2	0	1
s_2	s_3	s_2	0	1
s_3	s_3	s_1	1	0

12.

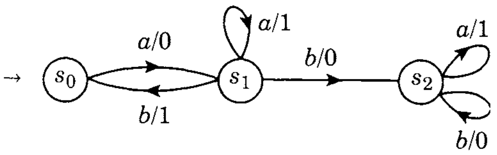
	f		*g*	
S \ *I*	a	b	a	b
s_0	s_1	s_2	1	0
s_1	s_2	s_2	0	1
s_2	s_2	s_3	0	0
s_3	s_2	s_3	1	1

Construct a transition table for each FSM.

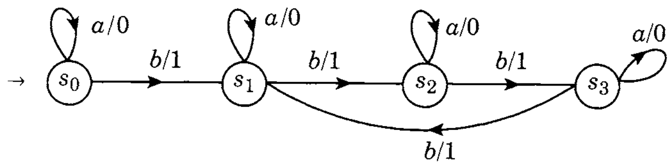
13.



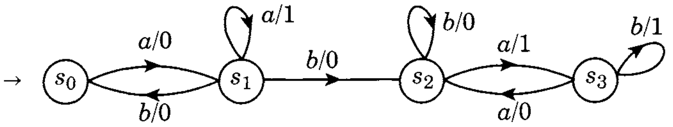
14.



15.



16.



Using the FSM in Figure 11.33, find the output from each input string.

17. *abba* **18.** *baab* **19.** a^2b^3a **20.** $a^3b^2ab^3$

Using the unit delay machine in Figure 11.35, find the output of each input string.

21. 1101 **22.** 1111 **23.** 0000 **24.** 101110

25. With a transition table, define the transition function *f* and the output function *g* of the FSM for binary addition in Figure 11.39.

Using the FSM in Figure 11.39, compute the sum of each pair of binary numbers.

26. 1001
0110

27. 00111
10010

28. 1011
0110

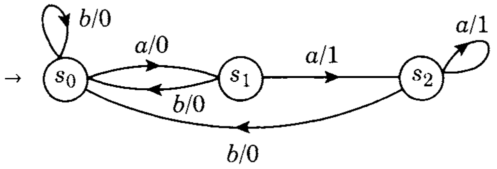
29. 11011
10101

30. Redraw Figure 11.20 as the transition diagram of an FSM.

31–34. Redraw the transition diagram of each automaton in Exercises 17–20 of Section 11.3 as that of an FSM.

Determine if the input string in Exercises 35–38 is accepted by the FSM in Figure 11.42.

Figure 11.42



35. *abba* **36.** *aabb* **37.** a^3 **38.** b^3a^4

39. Identify the language accepted by the FSM in Figure 11.42.

Design an FSM accepting strings over {a, b} that:

40. Contain *aa* as a substring. **41.** Contain exactly one *a*.

With x an input symbol and s an arbitrary state of an FSM $M = (S, I, O, f, g, s_0)$, define $g(s, x)$ in each case.

42. $f(s, x)$ is an accepting state. **43.** $f(s, x)$ is a nonaccepting state.

11.5 Deterministic Finite-State Automata and Regular Languages

Is the language accepted by an FSA context-sensitive? Or is it context-free, regular, or something else? This section provides a definitive answer to these questions.

In an FSA $M = (S, A, I, f, s_0)$, where $|I| = m$, exactly m outgoing edges leave every state s_i, each labeled with a unique element of I. Besides, since $f : S \times I \to S$, every state–input pair yields a unique state; in other words, every state–input pair uniquely determines the next state.

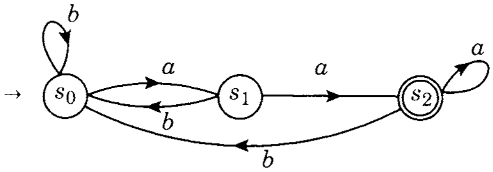
For the automaton in Figure 11.15, the pair (s_2, a) determines the state s_1, whereas the pair (s_2, b) determines the state s_3. Accordingly, the automata in Section 11.3 are called **deterministic finite-state automata** (DFSA).

This determinism suggests that the language accepted by a DFSA is indeed regular, as the next example demonstrates.

EXAMPLE 11.44

By Example 11.31 the language $L(M)$ accepted by the DFSA in Figure 11.43 consists of words over $\{\mathrm{a}, \mathrm{b}\}$ ending in aa. Employing it, a regular grammar $G = (N, T, P, \sigma)$ can be constructed. Choose $\{\mathrm{a}, \mathrm{b}\}$ as the set of terminal symbols: $T = \{\mathrm{a}, \mathrm{b}\}$. Choose the states as the nonterminal symbols: $N = \{s_0, s_1, s_2\}$. Select the initial state s_0 as the start symbol: $\sigma = s_0$.

Figure 11.43



Define the two productions rules:

- If there is an edge labeled x from state s_i to state s_j, define the production $s_i \to \mathrm{x}s_j$. The various productions obtained this way are:

$$s_0 \to as_1, \quad s_0 \to bs_0, \quad s_1 \to as_2,$$
$$s_1 \to bs_0, \quad s_2 \to as_2, \quad \text{and} \quad s_2 \to bs_0.$$

- If there is an edge labeled x from state s_i to an accepting state, induce the production $s_i \to x$. Two additional productions can be obtained by this method:

$$s_1 \to a \quad \text{and} \quad s_2 \to a$$

The grammar $G = (N, T, P, \sigma)$ where N, T, P, and σ are defined as above is clearly regular, therefore $L(G)$ is a regular language. You may verify that $L(G)$ consists of strings over T ending in aa. Thus $L(M) = L(G)$. ■

This example leads us to a fundamental result whose proof resembles that in Example 11.44.

THEOREM 11.3 The language accepted by a DFSA is regular.

PROOF:

Let $M = (S, A, I, f, s_0)$ be a DFSA and $L(M)$ denote the language accepted by the automaton. We shall construct a regular grammar G using the machine M and show that $L(G) = L(M)$.

To construct the grammar $G = (N, T, P, \sigma)$, choose $N = S$ as the set of states, $T = I$ as the input alphabet, and $\sigma = s_0$ as the initial state. Define the productions P this way:

Let s_i and s_j be any two states, and x any input symbol. If $f(s_i, x) = s_j$, define the production $s_i \to xs_j$; if $f(s_i, x) = s_j$, an accepting state, include the production $s_i \to x$. Clearly, G is a regular grammar.

To prove that $L(M) \subseteq L(G)$:

Let $x = x_1x_2 \ldots x_n$ be a string accepted by the automaton M; that is, let $x \in L(M)$. Then the transition diagram of the automaton contains a directed path s_0-s_1-s_2-$\cdots$-s_n, where s_n is an accepting state. Correspondingly, these production rules follow:

$$\begin{aligned} s_0 &\to x_1s_1 \\ s_1 &\to x_2s_2 \\ &\vdots \\ s_{i-1} &\to x_is_0i \\ &\vdots \\ s_{n-1} &\to x_n \qquad \text{(Note: } s_n \text{ is an accepting state.)} \end{aligned} \tag{11.4}$$

and the derivation of the string x:

$$\begin{aligned} &\Longrightarrow x_1s_1 \\ &\Longrightarrow x_1x_2s_2 \\ &\vdots \\ &\Longrightarrow x_1x_2 \ldots x_{n-1}s_{n-1} \\ &\Longrightarrow x_1x_2 \ldots x_{n-1}x_n \end{aligned} \tag{11.5}$$

since $s_{n-1} \to x_n$. Thus $x \in L(G)$, so $L(M) \subseteq L(G)$.

Conversely, let $x = x_1x_2 \ldots x_n \in L(G)$. Then it must have a derivation of the form (11.4). Correspondingly, the transition diagram of the automaton M must contain a directed path, s_0-s_1-s_2-$\cdots$-s_n. The string determined by this path is $x = x_1x_2 \ldots x_n$. Since the last production in the derivation

(11.4) is $s_{n-1} \to x_n$, s_n must be an accepting state, thus $x \in L(M)$ and hence $L(G) \subseteq L(M)$.

Thus $L(M) = L(G)$. In other words, the language accepted by the DFSA is regular. ■

This proof provides an elegant method for finding the regular language accepted by a DFSA. We demonstrate it again in the next example.

EXAMPLE 11.45 Find the grammar of the regular language accepted by the parity check machine in Example 11.33.

SOLUTION:
Using the transition diagram in Figure 11.23, $N = \{E, O\}$, $T = \{a, b\}$, $S = \{E\}$, and the production rules are:

$$E \to \text{a}O, \quad E \to \text{b}E, \quad O \to \text{a}E, \quad O \to \text{b}O, \quad E \to \text{b}, \quad \text{and} \quad O \to \text{a}$$

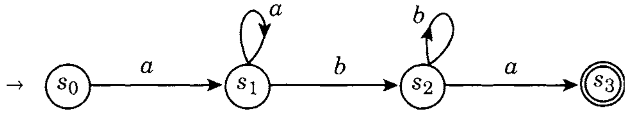
The regular grammar defined by the parity check machine M is $G = (N, T, P, S)$. [So $L(G) = L(M) =$ the set of strings over T containing an even number of a's.] ■

Finally, is the converse of Theorem 11.3 true? With G a regular grammar, does a DFSA exist such that $L(M) = L(G)$? The next two sections will give us an answer.

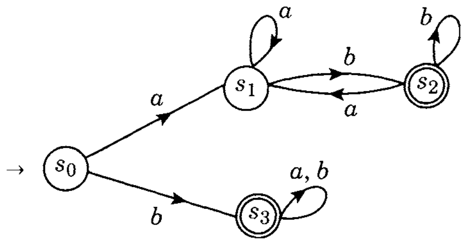
Exercises 11.5

Determine if each is a DFSA.

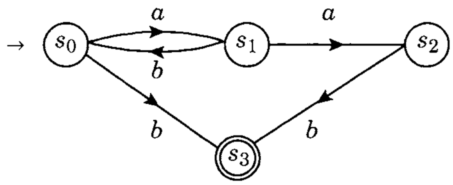
1.



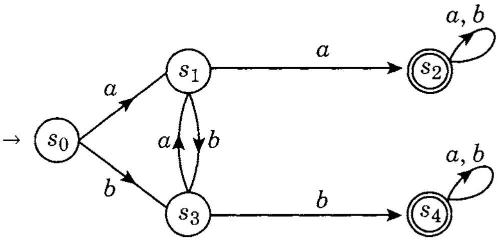
2.



3.



4.



Write the regular grammar defined by the DFSA in each figure.

5. Figure 11.17 **6.** Figure 11.28

7–14. Construct the regular grammar defined by each DFSA in Exercises 17–24 of Section 11.3.

By making a DFSA, define a regular grammar $G = (N, T, P, \sigma)$ that generates the language consisting of strings over {a, b} that:

15. Contain exactly one a. **16.** Contain at least one a.

17. Begin with aa. **18.** End with bb.

19. Contain aba as a substring. **20.** Contain aaa as a substring.

21. Begin with aa or bb. **22.** Contain $baab$ as a substring.

11.6 Nondeterministic Finite-State Automata

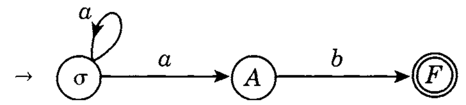
We ended the preceding section with a question: For a regular grammar G, is there a DFSA M such that $L(G) = L(M)$? The obvious temptation is to simply reverse the steps in Example 11.44 (or Theorem 11.3) to look for it. Let's see what happens if we do so.

EXAMPLE 11.46 With the regular grammar $G = (N, T, P, \sigma)$, where $N = \{A, \sigma\}$, $T = \{\text{a}, \text{b}\}$, and $P = \{\sigma \to \text{a}\sigma,\ \sigma \to \text{aA},\ \text{A} \to \text{b}\}$, let us see what happens if we reverse the steps in Theorem 11.3 in order to construct a DFSA $M = (S, A, I, f, s_0)$. Then $I = T = \{\text{a}, \text{b}\}$ and $s_0 = \sigma$. Corresponding to the productions $\sigma \to \text{a}\sigma$ and $\sigma \to \text{aA}$, there must be two states, namely, σ and A; besides, by virtue of the production A → b, an accepting state F must exist. Thus S must be $\{\sigma, \text{A}, \text{F}\}$.

Use the productions to draw the edges in the transition diagram of the automaton: If $s_i \to xs_j$, draw an edge from state s_i to s_j and label it x; if $s_i \to x$, draw an edge from s_i to the accepting state F and label it x. The diagram in Figure 11.44 results.

Unfortunately, it is not a DFSA for two reasons: (1) A state, σ, has two outgoing edges with the same label a; (2) not every state, namely A and F, has two edges with different labels. Thus reversing the steps illustrated in Example 11.44 does *not* yield a DFSA.

Figure 11.44



■

But, fortunately, we have another option. The automaton in Figure 11.44 is a nondeterministic finite-state automaton. "Nondeterministic" means that each state–input pair may determine more than one state. For instance, the pair (σ, a) determines two states, σ and A. If a is input at state σ, two choices exist for the next state: remain at σ or move to A.

We can now move to the following definition.

Nondeterministic Finite-State Automata

A **nondeterministic finite-state automaton** (NDFSA) M exhibits five characteristics:

- A finite set S of **states**;
- A specially designated state σ, called the **initial state**;
- A subset A of S consisting of the **accepting states** (or **final states**) of the automaton;
- A finite set I of **input symbols**;
- A function $f : S \times I \to P(S)$, called the **transition function** (or the **next-state function**). [*Note*: $P(S)$ denotes the power set of S.]

In symbols, $M = (S, A, I, f, \sigma)$.

In an NDFSA, each state–input pair is linked with a set of states, not necessarily a unique state; it can be the null set. A NDFSA can be represented by a transition diagram and a transition table can define a transition function, as the next two examples illustrate.

EXAMPLE 11.47 For the NDFSA in Figure 11.44, $S = \{\sigma, A, F\}$ and $A = \{F\}$. The transition table in Table 11.7 defines the transition function.

Table 11.7

$S \backslash I$	a	b
σ	$\{\sigma, A\}$	$\emptyset$
A	$\emptyset$	$\{F\}$
F	$\emptyset$	$\emptyset$

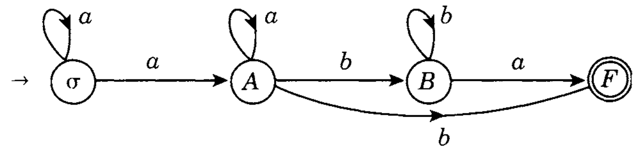
■

EXAMPLE 11.48 The NDFSA $M = (S, A, I, f, \sigma)$, where $S = \{\sigma, \mathrm{A}, \mathrm{B}, \mathrm{C}\}$, $A = \{\mathrm{F}\}$, $I = \{\mathrm{a}, \mathrm{b}\}$, and f is defined by Table 11.8. Its transition diagram is given in Figure 11.45.

Table 11.8

$S \backslash I$	a	b
σ	$\{\sigma, A\}$	$\emptyset$
A	$\{A\}$	$\{B, F\}$
B	$\{F\}$	$\{B\}$
F	$\emptyset$	$\emptyset$

Figure 11.45



■

The definition of a string accepted by an FSA can be extended to NDFSA as well.

Equivalent Nondeterministic Finite-State Automata

A string is **accepted** or **recognized** by a NDFSA $M = (S, A, I, f, s_0)$ if a directed path runs from the initial vertex s_0 to an accepting state that generates the string. The language of all strings accepted by M is $L(M)$. Two NDFSAs are **equivalent** if they accept the same language.

The next two examples illustrate the definition of $(L(M))$.

EXAMPLE 11.49 The word a^3b is accepted by the NDFSA in Figure 11.44 since the corresponding path, σ-σ-σ-A-F, ends at an accepting state F. Notice that $L(M) = \{\text{a}^n\text{b} \mid n \geq 1\}$. ■

EXAMPLE 11.50 The string $\text{a}^2\text{b}^3\text{a}$ is accepted by the NDFSA in Figure 11.45. Two paths generate it, σ-σ-A-B-B-B-F and σ-A-A-B-B-B-F. The automaton accepts strings a^mb and $\text{a}^m\text{b}^n\text{a}$, where $m, n \geq 1$. Thus $L(M) = \{\text{a}^m\text{b}, \text{a}^m\text{b}^n\text{a} \mid m, n \geq 1\}$. ■

The question we posed at the beginning of this section can be partially answered now.

THEOREM 11.4 Every regular language is accepted by an NDFSA.

PROOF:

Let $G = (N, T, P, \sigma)$ be a regular grammar. Through essentially the same steps as in Example 11.46, make a suitable NDFSA $M = (S, A, I, f, s_0)$ such that $L(G) = L(M)$. Select $I = T$, $s_0 = \{\sigma\}$, and N as the set of nonaccepting states of M. Since the grammar contains productions of the form $s_i \rightarrow x$, introduce an accepting state F; choose $S = N \cup \{F\}$ and $A = \{F\}$. Finally, since every production of G is $s_i \rightarrow xs_j$ or $s_i \rightarrow x$, the transition function $f : S \times I \rightarrow P(S)$ follows: $f(s_i, x) = \{s_j \mid s_i \rightarrow xs_j\} \cup \{F \mid s_i \rightarrow x\}$.

As in Theorem 11.3, it can be shown that $L(G) = L(M)$. (Complete the proof.) ■

Although nondeterministic finite-state automata have been defined, an explicit answer to the question posed earlier has yet to surface: Given a regular grammar G, does there exist a DFSA such that $L(G) = L(M)$? We will answer this in the next section.

Exercises 11.6

Draw the transition diagram of the NDFSA $M = (S,A,I,f,s_0)$, where:

1. $S = \{s_0,s_1,s_2)$, $A = \{s_2\}$

$S \backslash I$	a	b
s_0	$\{s_1\}$	$\{s_0\}$
s_1	$\{s_1\}$	$\{s_1,s_2\}$
s_2	$\emptyset$	$\emptyset$

2. $S = \{s_0,s_1,s_2\}$, $A = \{s_1\}$

$S \backslash I$	a	b
s_0	$\{s_1\}$	$\{s_0\}$
s_1	$\{s_2\}$	$\{s_1,s_2\}$
s_2	$\emptyset$	$\emptyset$

3. $S = \{s_0,s_1,s_2,s_3), A = \{s_2\}$

$S \backslash I$	a	b
s_0	$\{s_0,s_1\}$	$\{s_3\}$
s_1	$\{s_1,s_2\}$	$\{s_1\}$
s_2	$\{s_2\}$	$\{s_3\}$
s_3	$\{s_3\}$	$\{s_3\}$

4. $S = \{s_0,s_1,s_2,s_3\}, A = \{s_2\}$

$S \backslash I$	a	b
s_0	$\{s_0,s_1\}$	$\{s_3\}$
s_1	$\{s_1,s_2\}$	$\{s_0\}$
s_2	$\emptyset$	$\emptyset$
s_3	$\{s_1\}$	$\{s_3\}$

5. $S = \{s_0,s_1,s_2,s_3,s_4)$, $A = \{s_2,s_3\}$

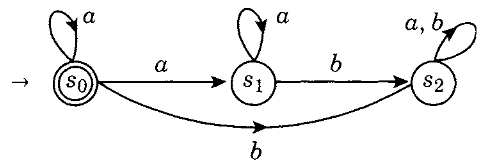
$S \backslash I$	a	b
s_0	$\{s_0,s_1\}$	$\{s_4\}$
s_1	$\{s_1,s_2\}$	$\{s_1,s_3\}$
s_2	$\emptyset$	$\emptyset$
s_3	$\emptyset$	$\emptyset$
s_4	$\{s_4\}$	$\{s_4\}$

6. $S = \{s_0,s_1,s_2,s_3,s_4,s_5)$, $A = \{s_2,s_5\}$

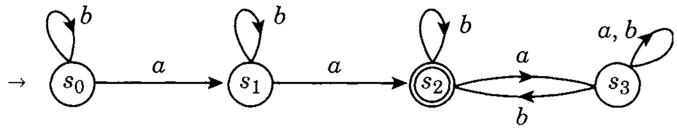
$S \backslash I$	a	b
s_0	$\{s_0,s_1\}$	$\{s_4\}$
s_1	$\{s_1,s_2\}$	$\{s_3\}$
s_2	$\{s_2\}$	$\{s_2\}$
s_3	$\{s_3\}$	$\{s_3\}$
s_4	$\{s_3\}$	$\{s_4,s_5\}$
s_5	$\{s_5\}$	$\{s_5\}$

Construct a transition table for each NDFSA.

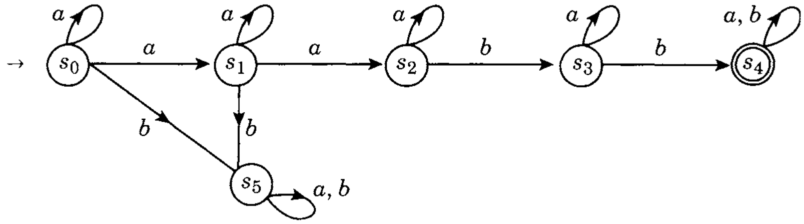
7.



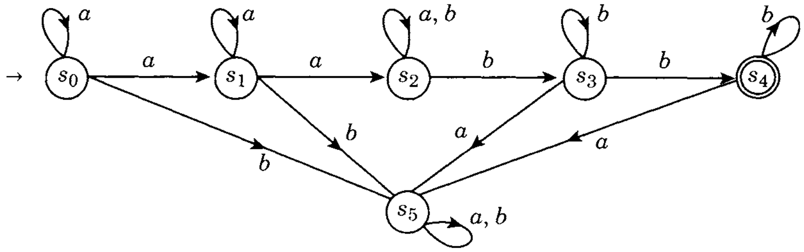
8.



9.



10.



Does the NDFSA in Figure 11.45 accept each string? Identify a path defined by any accepted string.

11. ab^2a

12. abab

13. a^3b

14. ab^2ab

Is each string accepted by the NDFSA in Exercise 1? Give a path for accepted strings.

15. a^2b

16. ab^2a

17. a^3b^3

18. $(ab)^3$

Does the NDFSA in Exercise 10 accept each string? Show a path that defines any accepted string.

19. abba

20. $(ab)^3$

21. a^2b^2

22. $a^4b^2ab^3$

Construct a NDFSA that accepts the language generated by the regular grammar $G = (N, T, P, \sigma)$, where:

23. $N = \{\sigma, A, B\}$, $T = \{a, b\}$, and $P = \{\sigma \to aA, A \to aA, A \to bB, B \to bB, A \to a\}$

24. $N = \{\sigma, A, B\}$, $T = \{a, b\}$, and $P = \{\sigma \to aA, \sigma \to bA, A \to aB, \sigma \to b, B \to b\}$

25. $N = \{\sigma, \mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}\}$, $T = \{\mathrm{a}, \mathrm{b}\}$, and $P = \{\sigma \to \mathrm{b}\sigma,\ \sigma \to \mathrm{aA},\ \mathrm{A} \to \mathrm{aA},\ \mathrm{A} \to \mathrm{bB},\ \mathrm{B} \to \mathrm{aA},\ \mathrm{B} \to \mathrm{bC},\ \mathrm{C} \to \mathrm{aD},\ \mathrm{C} \to \mathrm{b}\sigma,\ \mathrm{D} \to \mathrm{aD},\ \mathrm{D} \to \mathrm{bD},\ \mathrm{C} \to \mathrm{a}\}$

26. $N = \{\sigma, \mathrm{A}, \mathrm{B}, \mathrm{C}\}$, $T = \{\mathrm{a}, \mathrm{b}\}$, and $P = \{\sigma \to \mathrm{b}\sigma,\ \sigma \to \mathrm{aA},\ \mathrm{A} \to \mathrm{aA},\ \mathrm{A} \to \mathrm{bB},\ \mathrm{B} \to \mathrm{aA},\ \mathrm{B} \to \mathrm{bC},\ \mathrm{C} \to \mathrm{aA},\ \mathrm{C} \to \mathrm{b}\sigma,\ \mathrm{B} \to \mathrm{b}\}$

Create a NDFSA that accepts the regular language over $\{\mathrm{a}, \mathrm{b}\}$ of strings that:

27. Contain exactly one *a*.

28. Contain at least one *a*.

29. Begin with *aa*.

30. End with *bb*.

31. Contain *aba* as a substring.

32. Contain a^3 as a substring.

33. Begin with *aa* or *bb*.

34. Contain ba^2b as a substring.

***35.** Begin with *aa*, but not end in *bb*.

***36.** Begin with *aa* and end in *bb*.

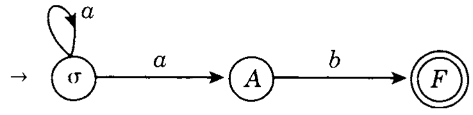
11.7 Automata and Regular Languages

The preceding two sections demonstrated that the language accepted by a DFSA is regular and that every regular language is accepted by an NDFSA. This section shows that every NDFSA is equivalent to a DFSA, which answers affirmatively our question about the existence of a possible DFSA M such that $L(G) = L(M)$. Every regular language is, in fact, accepted by a suitable DFSA.

The next two examples illustrate step by step how to construct a DFSA equivalent to a given NDFSA.

EXAMPLE 11.51 Consider the regular grammar $G = (N, T, P, \sigma)$, where $N = \{A, \sigma\}$, $T = \{\mathrm{a}, \mathrm{b}\}$, and $P = \{\sigma \to \mathrm{a}\sigma,\ \sigma \to \mathrm{aA},\ \mathrm{A} \to \mathrm{b}\}$. The NDFSA $M = (S, A, I, f, s_0)$ that accepts $L(G)$ is shown in Figure 11.46 (same as Figure 11.44). By Example 11.49, $L(M) = \{\mathrm{a}^n b \mid n \geq 1\}$. Using M, we shall construct the DFSA $M' = (S', A', I', f', s_0')$ which accepts $L(G)$:

Figure 11.46



Step 1 Choose $I' = I = \{\mathrm{a}, \mathrm{b}\}$, $s_0' = \{s_0\} = \{\sigma\}$, and $S' = P(S)$. The various states in M' are subsets of S. If there are n states in M, there can be 2^n states in M', so the states of M' are:

$$\emptyset,\ \{\sigma\},\ \{A\},\ \{F\},\ \{\sigma, A\},\ \{\sigma, F\},\ \{A, F\},\ \text{and } \{\sigma, A, F\}$$

Step 2 The accepting states of M' are those states of M' that contain an accepting state of M. They are $\{F\}$, $\{\sigma,F\}$, $\{A,F\}$, and $\{\sigma,A,F\}$.

Step 3 Let $X = \{s_1,s_2,\dots,s_m\}$ be a state in M'. An input symbol x leads from state X to state Y, where $Y = \bigcup_{i=1}^{m} f(s_i,x)$. In other words, an edge labeled x runs from state X to state Y if $Y = \bigcup_{i=1}^{m} f(s_i,x)$.

Figure 11.46 produces all possible transitions:

$$\begin{array}{llll} f(\emptyset,\mathrm{a}) = \emptyset & f(\emptyset,\mathrm{b}) = \emptyset & f(\sigma,\mathrm{a}) = \{\sigma,A\} & f(\sigma,\mathrm{b}) = \emptyset \\ f(A,\mathrm{a}) = \emptyset & f(A,\mathrm{b}) = \{F\} & f(F,\mathrm{a}) = \emptyset & f(F,\mathrm{b}) = \emptyset \end{array}$$

Since $f(\emptyset,\mathrm{a}) = \emptyset = f(\emptyset,\mathrm{b})$, edges run from $\emptyset$ to itself labeled a and b. Since $f(\sigma,\mathrm{a}) = \{\sigma, A\}$ and $f(\sigma,\mathrm{b}) = \emptyset$, an edge labeled a goes from $\{\sigma\}$ to $\{\sigma, A\}$ and an edge b from $\{\sigma\}$ to $\emptyset$. Similarly, there is an edge labeled a from $\{A\}$ to $\emptyset$, an edge b from $\{A\}$ to $\{F\}$, and two edges a and b from $\{F\}$ to $\emptyset$.

Since $f(\sigma,\mathrm{a}) \cup f(A,\mathrm{a}) = \{\sigma, A\} \cup \emptyset = \{\sigma, A\}$, an edge labeled a runs from $\{\sigma, A\}$ to $\{\sigma, A\}$. Also, $f(\sigma,\mathrm{b}) \cup f(\mathrm{A},\mathrm{b}) = \emptyset \cup \{F\} = \{F\}$, so an edge b goes from $\{\sigma,A\}$ to $\{F\}$. Similarly, there are edges labeled a and b from $\{\sigma, F\}$ to $\{\sigma, A\}$ and $\emptyset$, respectively; edges a and b from $\{A, F\}$ to $\emptyset$ and $\{F\}$, respectively; and edges a and b from $\{\sigma, A,F\}$ to $\{\sigma, A\}$ and $\{F\}$, respectively.

These results appear in the transition table in Table 11.9.

Table 11.9

S' \ I'	a	b
$\emptyset$	$\emptyset$	$\emptyset$
$\{\sigma\}$	$\{\sigma,A\}$	$\emptyset$
$\{A\}$	$\emptyset$	$\{F\}$
$\{F\}$	$\emptyset$	$\emptyset$
$\{\sigma,A\}$	$\{\sigma,A\}$	$\{F\}$
$\{\sigma,F\}$	$\{\sigma,A\}$	$\emptyset$
$\{A,F\}$	$\emptyset$	$\{F\}$
$\{\sigma,A,F\}$	$\{\sigma,A\}$	$\{F\}$

Figure 11.47 shows the resulting DFSA.

Since the states $\{A\}$, $\{\sigma, F\}$, $\{A, F\}$, and $\{\sigma, A, F\}$ cannot be reached from the initial state $\{\sigma\}$, they can be dropped out to yield the simplified DFSA M' in Figure 11.48.

From this transition diagram, $L(M') = \{\mathrm{aa}^n\mathrm{b} \mid n \geq 0\} = \{\mathrm{a}^n\mathrm{b} \mid n \geq 1\} = L(G)$. Thus the automata M and M' are equivalent, so the NDFSA is the same as the DFSA. ■

EXAMPLE 11.52 Construct a DFSA $M' = (S',A',I',f',s_0')$ equivalent to the NDFSA $M = (S,A,I,f,s_0)$ in Example 11.50. Recall that $L(M) = \{\mathrm{a}^m\mathrm{b}, \mathrm{a}^m\mathrm{b}^n\mathrm{a} \mid m,n \geq 1\}$. The key steps lie below. (Fill in the details.)

Figure 11.47

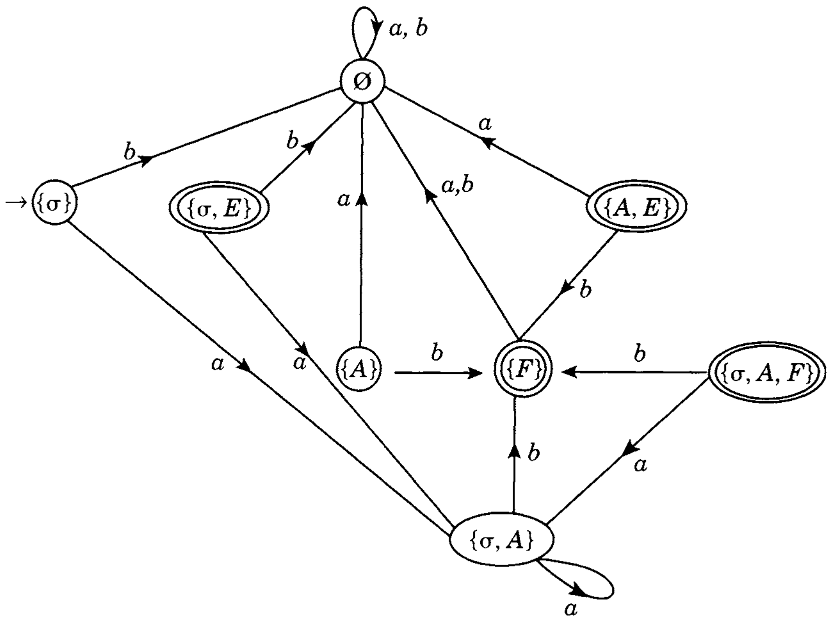
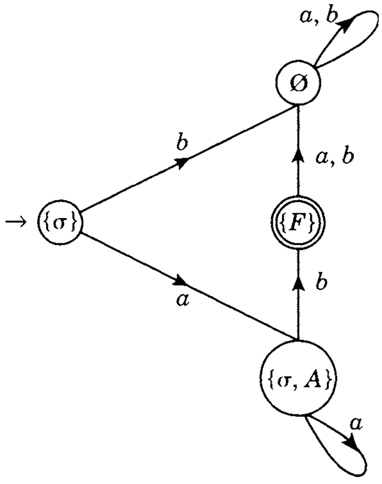


Figure 11.48



SOLUTION:

Step 1 Select $I' = I = \{a,b\}$, $s'_0 = \{s_0\} = \{\sigma\}$, and $S' = P(S)$. The states of M' are $\emptyset$, $\{\sigma\}$, $\{A\}$, $\{B\}$, $\{F\}$, $\{\sigma,A\}$, $\{\sigma,B\}$, $\{\sigma,F\}$, $\{A,B\}$, $\{A,F\}$, $\{B,F\}$, $\{\sigma,A,B\}$, $\{\sigma,A,F\}$, $\{\sigma,B,F\}$, $\{A,B,F\}$, and $\{\sigma,A,B,F\}$.

Step 2 The accepting states of M′ are $\{F\}$, $\{\sigma,F\}$, $\{A,F\}$, $\{B,F\}$, $\{\sigma,A,F\}$, $\{\sigma,B,F\}$, $\{A,B,F\}$, and $\{\sigma,A,B,F\}$.

Step 3 The transition table of the DFSA is Table 11.10.

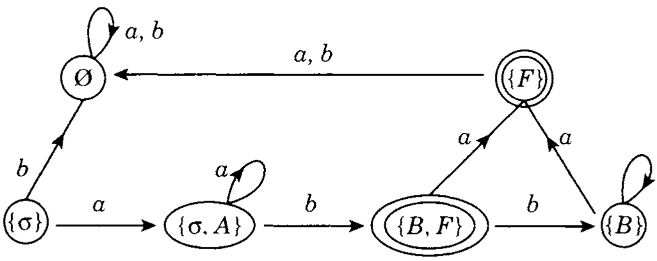
Table 11.10

S \ I	a	b
$\emptyset$	$\emptyset$	$\emptyset$
$\{\sigma\}$	$\{\sigma, A\}$	$\emptyset$
$\{A\}$	$\{A\}$	$\{B, F\}$
$\{B\}$	$\{F\}$	$\{B\}$
$\{F\}$	$\emptyset$	$\emptyset$
$\{\sigma, A\}$	$\{\sigma, A\}$	$\{B, F\}$
$\{\sigma, B\}$	$\{\sigma, A, F\}$	$\{B\}$
$\{\sigma, F\}$	$\{\sigma, A\}$	$\emptyset$
$\{A, B\}$	$\{A, F\}$	$\{B, F\}$
$\{A, F\}$	$\{A\}$	$\{B, F\}$
$\{B, F\}$	$\{F\}$	$\{B\}$
$\{\sigma, A, B\}$	$\{\sigma, A, F\}$	$\{B, F\}$
$\{\sigma, A, F\}$	$\{\sigma, A\}$	$\{B, F\}$
$\{\sigma, B, F\}$	$\{\sigma, A, F\}$	$\{B\}$
$\{A, B, F\}$	$\{A, F\}$	$\{B, F\}$
$\{\sigma, A, B, F\}$	$\{\sigma, A, F\}$	$\{B, F\}$

Step 4 The table indicates the states $\{\sigma, B\}$, $\{\sigma, F\}$, $\{A, B\}$, $\{\sigma, A, B\}$, $\{\sigma, B, F\}$, $\{A, B, F\}$, and $\{\sigma, A, B, F\}$ are not reachable from any state, so they are *not* the initial state $\{\sigma\}$. Delete the corresponding rows from the table. It is now obvious from the table that the states $\{A\}$, $\{A, F\}$, $\{\sigma, A, F\}$ also cannot be reached from $\{\sigma\}$; delete those rows also from the table.

The resulting transition diagram of the DFSA M' appears in Figure 11.49.

Figure 11.49



From the diagram, it follows that $L(M') = \{a^m b, a^m b^n a \mid m, n \geq 1\} = L(M)$. Thus M and M' are equivalent automata. As in the previous example, we have shown that the equivalency between an NDFSA and a DFSA. ■

The techniques illustrated in the two previous examples can be generalized to arrive at the following result. (The proof is a bit complicated, so we omit it.)

THEOREM 11.5 Every NDFSA is equivalent to a DFSA. ■

The next theorem follows from Theorems 11.3, 11.4, and 11.5.

THEOREM 11.6 A language is regular if and only if it is accepted by a DFSA. ■

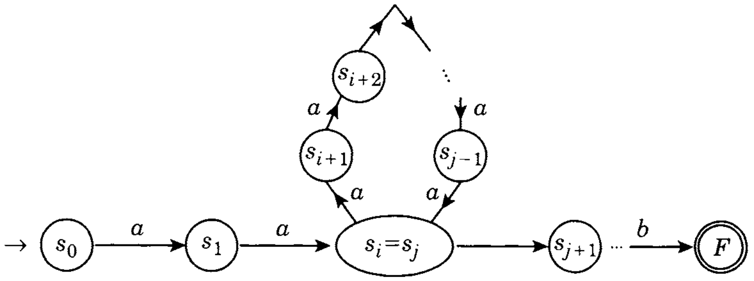
As Theorem 11.6 indicates, a DFSA can define a regular grammar and vice versa. Each is a characterization of the other.

We now look for an example of a simple-looking language that is *not* regular.

EXAMPLE 11.53 Show that the language $L = \{a^n b^n \mid n \geq 1\}$ is not regular.

PROOF (by contradiction):
Suppose L is regular. Then, by Theorem 11.6, a DFSA M exists such that $L(M) = L$. Suppose M has m states. Since the string $x = a^{m+1}b^{m+1} \in L$, x is accepted by the DFSA. Let P be the path corresponding to x; it ends at an accepting state F.

Figure 11.50



The path corresponding to the substring a^{m+1} contains $m+1$ states. But, since only m states exist, by the pigeonhole principle at least two of the $m+1$ states, say, s_i and s_j, where $i < j$, must be the same; consequently, there must be a directed cycle at s_i, each edge labeled a (see Figure 11.50). Let l be the length of the cycle. The path s_0-s_1-$\cdots$-s_i-s_{j+1}-s_{j+2}-$\cdots$-F generates the string $x' = a^{m+1-l}b^{m+1}$. Since this path ends at F (an accepting state), x' is accepted by the automaton; so $x' \in L$. This is a contradiction, since x' does not contain the same number of a's and b's. Thus L is not a regular language. ■

It follows by this example that the set of well-formed nested parentheses is also *not* a regular language. (Why?)

These discussions lead us to a powerful conclusion: Regular languages are accepted by DFSAs.

Exercises 11.7

1–6. Construct a DFSA equivalent to each NDFSA in Exercises 1–4, 7, 8 of Section 11.6. Eliminate all unreachable states.

7–8. Design DFSAs equivalent to the NDFSAs in Exercises 23 and 24 of Section 11.6. Eliminate all unreachable states.

Let L be the language recognized by an FSA and $L^R = \{x_n \dots x_1 \mid x_1 \dots x_n \in L\}$. Construct an NDFSA that accepts L^R from each FSA in Exercises 9–16. (*Hint:* Reverse the directions of the edges; switch the roles of the initial state and the accepting states.)

9. Figure 11.20

10. Exercise 17 in Section 11.3

11. Exercise 18 in Section 11.3

12. Exercise 36 in Section 11.3

13. Exercise 37 in Section 11.3

14. Exercise 38 in Section 11.3

15. Exercise 40 in Section 11.3

16. Exercise 41 in Section 11.3

17–24. Identify the language $L(M)$ accepted by the FSA in Exercises 9–16.

25–32. Construct a DFSA equivalent to the NDFSA in Exercises 9–16.

Chapter Summary

The abstract models of computing machines with limited capabilities are DFSA, FSM, and NDFSA. An automaton checks if a given input string has a desired property and produces no output values. An FSM, on the other hand, yields an output value corresponding to each input.

Formal Language

- A **formal language** over an alphabet Σ is a subset of Σ^*, the set of all possible words over Σ (page 734).
- The **concatenation** of two languages A and B over Σ consists of words ab with $a \in A$ and $b \in$ B (page 736).
- $A^n = \{a_1a_2 \dots a_n \mid a_i \in A\}$, where $A^0 = \Lambda$ (page 739).
- $A^* = \bigcup_{n=0}^{\infty} A^n$ is the **Kleene closure** of a language A (page 739).

Grammar

- A **grammar** $G = (N, T, P, \sigma)$ consists of a finite set N of **nonterminal symbols**, a finite set T of **terminal symbols**, a finite set P of **production rules**, and a **start symbol** σ (page 746).

- A word $w' = x\beta y$ is **directly derivable** from a word $w = x\alpha y$ if a production $\alpha \to \beta$ exists; we then write $w \Rightarrow w'$. A word w_n is **derivable** from w_1 if there exists a finite sequence of **derivations**, $w_1 \Rightarrow w_2, w_2 \Rightarrow w_3, \dots, w_{n-1} \Rightarrow w_n$. The language derivable from σ is the **language generated** by G, denoted by $L(G)$ (page 746).
- In BNF, each production $w \to w'$ is written as $w ::= w'$; each nonterminal symbol w is enclosed by angle brackets, as in $\langle w \rangle$; and production rules with the same left-hand sides are combined with vertical bars into a single rule (page 748).
- A **type 0** grammar has phrase-structure (page 751).
- In a **context-sensitive** (**type 1**) grammar, every production is of the form $\alpha A \alpha' \to \alpha \beta \alpha'$. (page 751)
- In a **context-free** (**type 2**) grammar, every production is of the form $A \to \alpha$. (page 751).
- In a **regular** (**type 3**) grammar, every production is of the form A→ t or $A \to t$B (page 751).
- A language $L(G)$ is **context-sensitive**, **context-free**, or **regular** according as whether G is context-sensitive, context-free, or regular (page 751).
- An **ambiguous** language contains a word that has more than one derivation tree (page 753).
- The language accepted by a DFSA is regular (page 780).

Finite-State Automaton (FSA)

- A **FSA** $M = (S, A, I, f, s_0)$ consists of a finite set S of **states**, a finite set A of **accepting states**, a finite set I of **input symbols**, a **transition function** $f : S \times I \to S$, and an **initial state** s_0. Every state–input pair yields a unique next-state of the automaton (page 761).
- A **transition table** defines the transition function. (page 761).
- A **transition diagram** can represent a DFSA. The initial state s_0 is identified by drawing an arrow toward it; an accepting state by two concentric circles around it (page 762).
- An input string is **accepted** by an automaton M if and only if the string traces a path that ends at an accepting state. The **language** $L(M)$ **accepted** by M consists of all words recognized by it (page 764).
- Two automata M and M' are **equivalent** if $L(M) = L(M')$ (page 764).

Finite-State Machine (FSM)

- An **FSM** $M = (S, I, O, f, g, s_0)$ consists of a finite set S of states, a finite set I of **input symbols**, a finite set O of **output symbols**, a **transition function** $f : S \times I \to S$, an **output function** $g : S \times I \to O$, and an **initial state** s_0. Every state–input pair produces a next-state and an output value (page 772).
- A **transition table** can define the transition and output functions of an FSM (page 772).
- A **transition diagram** also can define an FSM (page 772).

Nondeterministic Finite-State Automaton (NDFSA)

- An **NDFSA** $M = (S, A, I, f, \sigma)$ consists of a finite set S of states, a subset A of S of accepting states, a finite set I of input symbols, a transition function $f : S \times I \to P(S)$, and an initial state σ. A state–input pair may be paired with zero, one, or more states (page 783).
- Every regular language is accepted by a NDFSA (page 784).
- Every NDFSA is equivalent to a DFSA (page 787).
- Every regular language is accepted by a DFSA (page 787).

Review Exercises

Let $A = \{\lambda, \text{a}, \text{bc}\}$ and $B = \{\text{a}, \text{ab}\}$. Find each.

1. AB **2.** BA **3.** A^3 **4.** B^3

Find three words belonging to each language over {a,b,c}.

5. $\{\text{a,b}\}\{\text{c}\}^*$

6. $\{\text{a}\}\text{b}^*\{\text{c}\}^*$

7. $\{\text{ab}\}\{\text{ab}\}^*$

8. $\{\text{b}\}\{\text{a,b,c}\}^*\{\text{b}\}$

A grammar $G = (N, T, P, \sigma)$ has $N = \{\langle\text{noun phrase}\rangle, \langle\text{verb}\rangle, \langle\text{adjective}\rangle, \langle\text{noun}\rangle, \langle\text{article}\rangle\}$, $T = \{$a,the,chicken,wolf,cabbage,eats,walks,reliable, discreet, gracious$\}$, $\sigma = \langle\text{sentence}\rangle$, and the production rules are:

$$\begin{aligned}
\langle\text{sentence}\rangle &\to \langle\text{noun phrase}\rangle\langle\text{verb}\rangle\langle\text{noun phrase}\rangle\\
\langle\text{noun phrase}\rangle &\to \langle\text{article}\rangle\langle\text{noun}\rangle \mid \langle\text{article}\rangle\langle\text{adjective}\rangle\langle\text{noun}\rangle\\
\langle\text{article}\rangle &\to \textit{the} \mid \textit{a}\\
\langle\text{noun}\rangle &\to \textit{chicken} \mid \textit{wolf} \mid \textit{cabbage}\\
\langle\text{adjective}\rangle &\to \textit{reliable} \mid \textit{discreet} \mid \textit{gracious}\\
\langle\text{verb}\rangle &\to \textit{eats} \mid \textit{walks}
\end{aligned}$$

Determine if each is a valid sentence in $L(G)$.

9. The gracious chicken walks the wolf.

10. The reliable wolf eats a chicken.

Make a derivation tree for each sentence.

11. The discreet wolf eats the cabbage.

12. The reliable cabbage walks the gracious chicken.

Using the grammar $G = (N, T, P, \sigma)$ where $N = \{\sigma, A, B\}$, $T = \{\text{a,b}\}$, and $P = \{\sigma \to \text{b}\sigma, \sigma \to \text{a}A, A \to \text{a}B, A \to \text{b}\sigma, B \to \text{a}B, B \to \text{b}B, A \to \text{a}, B \to \text{a}, B \to \text{b}\}$, determine if each string belongs to $L(G)$.

13. ab^3a **14.** $(\text{ab})^3$ **15.** aba^2b **16.** ab^2a^4

Construct a parse tree for each string.

17. ba^2b **18.** $\text{b}^2\text{a}^3\text{b}$ **19.** aba^2 **20.** $\text{a}^2\text{b}^2\text{a}^2$

Develop a grammar that generates each language over {a,b}.

21. $\{\text{b}^n \mid n \geq 1\}$ **22.** $\{\text{a}^n\text{ba}^n \mid n \geq 0\}$ **23.** $\{\text{b}^{2n+1} \mid n \geq 0\}$ **24.** $\{\text{ab}^n\text{a} \mid n \geq 0\}$

With the grammar below, construct parse trees for the simple **while statements** in Exercises 25 and 26.

⟨while statement⟩ ::= while ⟨expression⟩ do ⟨statement⟩
⟨statement⟩ ::= ⟨assignment statement⟩ | ⟨while statement⟩ | λ
⟨assignment statement⟩ ::= ⟨variable⟩ := ⟨expression⟩
⟨variable⟩ ::= *a* | *b* | *c* | . . . | *z*
⟨expression⟩ ::= ⟨variable⟩⟨sign⟩⟨variable⟩ |
⟨variable⟩⟨operator⟩⟨variable⟩
⟨operator⟩ ::= =|≠|<|≤|>|≥
⟨sign⟩ ::= + | −

○ **25.** While $x \geq y$ do $x := y + z$.

○ **26.** While $x \geq y$ do while $y < z$ do $a := b + c$.

27. Draw the transition diagram of the *DFSAM* $= (S, \text{A}, I, f, s_0)$, where $S = \{s_0, s_1, s_2, s_3, s_4\}$, $A = \{s_3\}$, $I = \{\text{a,b}\}$, and f is defined by Table 11.11.

28. Redo Exercise 27 with $A = \{s_1, s_3\}$ and f defined by Table 11.12.

Construct the transition table for each DFSA.

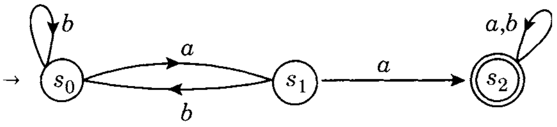
Table 11.11

S I	a	b
s_0	s_1	s_4
s_1	s_4	s_2
s_2	s_4	s_3
s_3	s_3	s_3
s_4	s_4	s_4

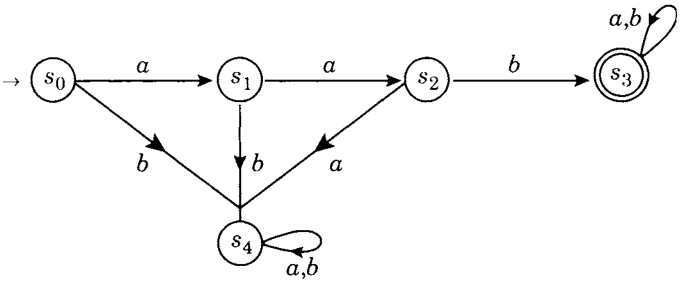
Table 11.12

S I	a	b
s_0	s_1	s_2
s_1	s_1	s_1
s_2	s_4	s_3
s_3	s_3	s_3
s_4	s_4	s_4

29.



30.



31–34. Identify the language $L(M)$ accepted by the automata in Exercises 27–30.

Design a DFSA that accepts strings over {a, b} that:

35. Begin with *aaa*.

36. Contain *abb* as a substring.

37.

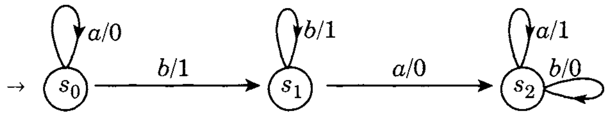
	f		g	
S I	a	b	a	b
s_0	s_0	s_1	1	0
s_1	s_2	s_2	0	0
s_2	s_0	s_3	1	0
s_3	s_3	s_2	1	0

38.

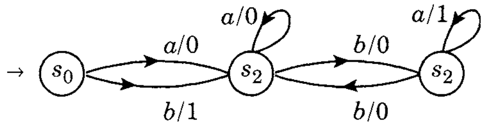
	f		g	
S I	a	b	a	b
s_0	s_0	s_1	0	0
s_1	s_2	s_3	0	1
s_2	s_3	s_2	1	1
s_3	s_3	s_2	1	0

Construct a transition table for each FSM.

39.



40.



Using the FSM in Figure 11.33 (Example 11.38), find the output from each input string.

41. a^2b^2aba **42.** aba^2ba **43.** ab^3ab **44.** $a^2b^3a^2$

45–46. Redraw the DFSAs in Exercises 27 and 29 as FSMs.

Design an FSM to accept string over {a, b} that:

47. Contain ab^2 as a substring. **48.** Begin with a or b^2.

49–50. Compose the regular grammar defined by the DFSA in Exercises 27-28.

Draw the transition diagram of the NDFSA $M = (S, A, I, f, s_0)$, where $I =$ {a,b} and:

51. $S = \{s_0, s_1, s_2\}, A = \{s_1\}$

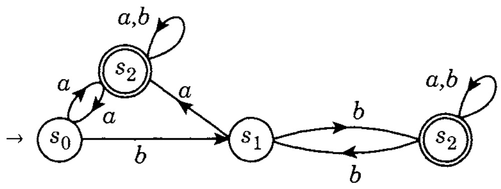
S \ I	a	b
s_0	$\{s_1\}$	$\{s_0\}$
s_1	$\{s_2\}$	$\{s_1, s_2\}$
s_2	$\{s_2\}$	$\{s_2\}$

52. $S = \{s_0, s_1, s_2\}, A = \{s_1\}$

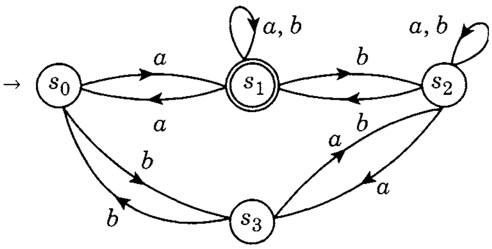
S \ I	a	b
s_0	$\{s_1\}$	$\{s_0\}$
s_1	$\{s_0, s_2\}$	$\{s_1\}$
s_2	$\{s_1\}$	$\{s_0, s_2\}$

Construct a transition table for each NDFSA.
Determine if the NDFSA in Exercise 52 accepts each input string.

53.



54.



55. a^3 **56.** ab^2ab^4 **57.** a^2b^3 **58.** a^3b^4

59–62. Determine if each input string in Exercises 55–58 is accepted by the NDFSA in Exercise 54.

Create a NDFSA that accepts the language $L(G)$ generated by the regular grammar $G = (N, T, P, \sigma)$, where:

63. $N = \{\sigma, A, B\}$, $T = \{A, B\}$, and $P = \{\sigma \to \mathrm{b}\sigma, \sigma \to \mathrm{a}A, A \to \mathrm{a}B, A \to \mathrm{b}A, A \to \mathrm{b}B, B \to \mathrm{a}B, B \to \mathrm{b}B, \sigma \to \mathrm{a}, A \to \mathrm{b}\}$

64. $N = \{\sigma, A, B\}$, $T = \{\mathrm{a,b}\}$, and $P = \{\sigma \to \mathrm{a}A, \sigma \to \mathrm{b}\sigma, A \to \mathrm{a}\sigma, A \to \mathrm{a}B, A \to \mathrm{b}A, B \to \mathrm{a}B, B \to \mathrm{b}\sigma, B \to \mathrm{b}A, \sigma \to \mathrm{a}, A \to \mathrm{b}\}$

65–66. Construct a DFSA equivalent to each NDFSA in Exercises 51 and 52.

67–68. What languages do the DFSAs in Exercises 65 and 66 accept?

Let A and B be any languages over a finite alphabet Σ. Prove each.

***69.** $(A \cup B^*)^* = (A^* \cup B)^*$ ****70.** $(A \cup B)^* = (A^* \cup B^*)^* = (A^* B^*)^*$

Supplementary Exercises

Let m denote the number of a's and n the number of b's in a string over {a,b}. Design an FSA that accepts strings with the given properties.

1. $m \equiv 1 \pmod 2$ and $n \equiv 1 \pmod 2$.

2. $m \equiv 0 \pmod 3$ and $n \equiv 1 \pmod 3$.

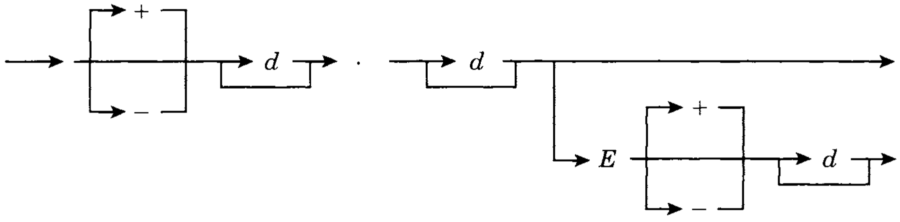
3. $m \equiv 0 \pmod 2$ and $n \equiv 1 \pmod 2$, or $m \equiv 1 \pmod 2$ and $n \equiv 0 \pmod 2$.

4. Design an FSA that accepts positive integers n divisible by 3.

○ ***5.** Using the syntax diagram in Figure 11.51 for a real number, design an FSA to recognize valid real numbers.

***6.** The Roman numerals M, D, C, L, X, V, and I have values 1000, 500, 100, 50, 10, 5, and 1, respectively. In the strict additive notation no numeral with a smaller value precedes a numeral with a larger value. For instance, 19 is written as XVIIII in lieu of the shorter

Figure 11.51



representation XIX and MMLXXVI, unlike MCMXCVI, is a well-formed sequence. Excepting M's, C, X, or I should not appear more than four times in the same sequence, and D, L, or V no more than once. This makes sense since CCCCC = D, XXXXX = L, and so on. Design an FSA to recognize the language of such well-formed sequences of additive Roman numerals.

Develop a grammar that generates each language over {a,b}.

***7.** The set of words that begin and end with *a*.

***8.** The set of words that begin with *aa* and end with *bb*.

9. Using BNF, define a grammar for the language of well-formed nested parentheses.

10. Use productions instead of BNF to define the grammar in Exercise 9.

A **Moore machine** $M = \langle S, I, O, f, g, s_0 \rangle$, named after Edward Moore who introduced it in 1956, is an FSM consisting of a finite set S of states, a finite set I of input symbols, a finite set O of output symbols, a transition function $f : S \times I \rightarrow S$, an output function $g : S \rightarrow O$, and an initial state s_0. Draw a transition diagram for the Moore machine defined by each transition table.

11.

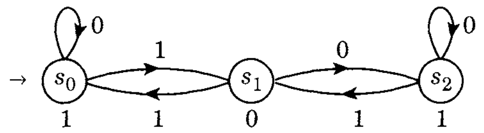
	f input		*g*
s	0	1	
s_0	s_0	s_1	1
s_1	s_3	s_2	0
s_2	s_2	s_3	1
s_3	s_0	s_1	1

12.

	f input		*g*
s	0	1	
s_0	s_1	s_3	0
s_1	s_1	s_2	1
s_2	s_3	s_2	1
s_3	s_1	s_2	0

Construct a transition table for each Moore machine.

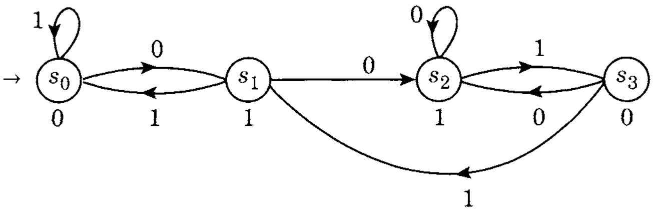
13.



***Edward Forrest Moore** (1925–) was born in Baltimore, Maryland. He graduated from Virginia Polytechnic Institute in 1947 and received his Ph.D. in mathematics from Brown 3 years later. After teaching at the University of Illinois for a year, he joined the technical staff at Bell Telephone Labs. In 1966, he joined the faculty of the University of Wisconsin, Madison, and taught there until his retirement in 1985.*

Moore has made outstanding contributions to the logical design of switching circuits, automata theory, graph theory, and database management.

14.



The output generated by the Moore machine $M = (S,I,O,f,g,s_0)$ for the input string $a_1a_2\ldots a_m$ is $g(s_0)g(s_1)\ldots g(s_m)$, where $s_i = f(s_{i-1},a_i)$ and $1 \leq i \leq m$. Find the output produced by the machine in Exercise 13 for each input.

15. 011 **16.** 1010 **17.** 10001 **18.** 1101101

19. Let L be a regular language. Prove that $L^R = \{x_n \ldots x_1 \mid x_1 \ldots x_n \in L\}$ is also regular.

****20.** An FSM $M = (S,I,O,f,g,s_0)$ is **simply minimal** if no output rows in its transition table are identical. If $|S| = n$, $|I| = m$, and $|O| = p$, how many simply minimal FSMs are possible?

Computer Exercises

Write a program to do each task, where $\Sigma = \{a,b\}$.

1. Determine if a string over Σ:

- Begins with *aa*.
- Ends with *bb*.
- Contains exactly one *a*.
- Contains at least one *a*.
- Contains *aba* as a substring.
- Has its number of *a*'s congruent to 1 mod 3.

- Has an even number of a's and b's.
- Has both its number of a's and b's congruent to 1 mod 3.

2. Let m denote the number of a's and n the number of b's in a string over Σ. Read in a word over Σ and see if it has:
   - $m \equiv 0 (\text{mod } 5)$
   - $m \equiv 0 (\text{mod } 3)$ or $m \equiv 1 (\text{mod } 3)$
   - $m \equiv 0 (\text{mod } 3)$ and $n \equiv 1 (\text{mod } 3)$
   - $m \equiv 3 (\text{mod } 5)$
   - $m \equiv 0 (\text{mod } 3)$ or $m \equiv 2 (\text{mod } 3)$
   - $m \equiv 0 (\text{mod } 3)$ and $n \equiv 2 (\text{mod } 3)$
3. For a DFSA with n (≤ 10) states, labeled 1 through n, read in its number of states n and transition table. Read in a sequence of input strings over Σ and determine if each is accepted by the DFSA.
4. Implement the unit delay machine in Example 11.41.
5. Read in two binary numbers and use the FSM in Example 11.42 to compute their sum.

○ 6. By means of the syntax diagram in Figure 11.31, determine if a string of characters represents a valid integer.

○ 7. Ascertain with the syntax diagram in Figure 11.32 whether a string of characters represents a valid real number. (Excluding the exponential form.)

8. Read in the number of states n, the transition table, and a set of input strings for an FSM with n (≤ 10) states, labeled 1 through n. Print the output produced by each input string.
9. Using a DFSA with n (≤ 10) states, labeled 1 through n, read its number of states n and transition table. Determine the corresponding regular grammar.

Exploratory Writing Projects

Using library and Internet resources, write a team report on each of the following in your own words. Provide a well-documented bibliography.

1. Discuss how BNF rules are used to define programming languages such as C++ and Java.
2. Discuss Turing machines and Church's thesis.
3. Explain how vending machines, slot machines, and garage door openers can be modeled by FSAs.
4. Write an essay on Kleene closure.
5. Write an essay on different types of FSMs and their applications.

Enrichment Readings

1. W. J. Barnier, "Finite-State Machines as Recognizers," *The UMAP Module 671* (1986), pp. 209–232.
2. B. Hayes, "On the Finite-State Machine, a Minimal Model of Mousetraps, Ribosomes, and the Human Soul," *Scientific American*, Vol. 249 (Dec. 1983), pp. 19–28, 178.
3. J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1979.
4. Z. Kohavi, *Switching and Finite Automata Theory*, 2nd ed., McGraw-Hill, New York, 1978.
5. P. Linz, *An Introduction to Formal Languages and Automata*, D. C. Heath, Lexington, MA 1990.
6. J. C. Martin, *Introduction to Languages and the Theory of Computation*, 2nd ed., McGraw-Hill, New York, 1997.
7. M. Sipser, *Introduction to the Theory of Computation*, PWS, Boston, 1997.
8. W. A. Wulf *et al.*, *Fundamental Structures of Computer Science*, Addison-Wesley, Reading, MA, 1981, pp. 1–64.