CHAPTER

16 Finite-state recognizers

In this chapter we consider the characterization of finite-state machines and the sets of sequences that they accept. We investigate a number of generalized forms of finite-state machines and prove that these forms are equivalent, with respect to the sets of sequences that they accept, to the basic deterministic finite-state model. In Sections 16.2 and 16.3 we study the properties of nondeterministic state diagrams, called transition graphs, which will prove to be a useful tool in the study of regular expressions. Procedures are developed whereby any transition graph can be converted into a deterministic state diagram.

Section 16.4 presents the language of regular expressions, which provides a precise characterization of the sets of sequences accepted by finite-state machines. In the following two sections we prove that any finite-state machine can be characterized by a regular expression and that every regular expression can be realized by a finite-state machine. Finally, in Section 16.7 we will be concerned with a generalized form of finite-state machines known as two-way machines.

16.1 Deterministic recognizers

So far, we have regarded a finite-state machine as a *transducer* that *transforms* input sequences into output sequences. In this chapter we shall view a machine as a *recognizer* that *classifies* input strings into two classes, those that it accepts and those that it rejects. The set consisting of all the strings that a given machine accepts is said to be *recognized* by that machine.

The finite-state model that we shall use is shown in Fig. 16.1, where a *finite-state control* is coupled through a *head* to a finite linear sequence of squares, each containing a single symbol of the alphabet. Such a sequence of squares is called an (*input*) *tape*. Initially, the finite-state control is in the starting state, and the head scans the leftmost symbol of the string that appears on the tape. The head then scans the tape from left to right. In what is termed

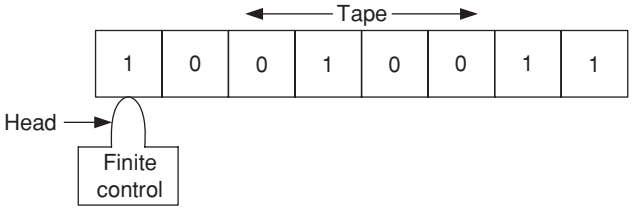
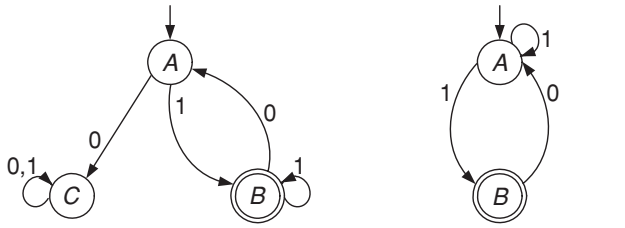


Fig. 16.1 A finite-state recognizer.



(*a*) Deterministic state diagram. (*b*) Transition graph.

Fig. 16.2 Two ways of describing a string.

a *cycle of computation*, the machine starts in some state S_i, reads the symbol currently scanned by the head, shifts one square to the right, and then enters the state S_j.

Clearly, the concept of a head reading from left to right the symbols contained in a linear tape is equivalent to a string of input symbols entering the machine at successive times. In fact, the finite-state control is a Moore finite-state machine.[1] States whose assigned output symbol is 1 are referred to as *accepting* (or *terminal*) *states* while states whose assigned output symbol is 0 are called *rejecting* (or *nonterminal*) *states*. A string (or a tape) is *accepted* by a machine if and only if the state that the machine enters after having read the rightmost tape symbol is an accepting state. Otherwise the string is rejected. The set of strings recognized by a machine thus consists of all the input strings that take the machine from its starting state to an accepting state.

The machine of Fig. 16.1 can be described by a state diagram in which the starting state is marked by an incoming short arrow and the accepting states are indicated by double circles. For example, the state diagram of Fig. 16.2*a* describes a machine that accepts a string if and only if the string begins and ends with a 1 and every 0 in the string is preceded and followed by at least a single 1. The machine consists of three states, of which A is the starting state and B is an accepting state. Note that in general a starting state may also be an accepting state. In such a case, the machine is said to accept the null string.

[1] By allowing the head to write on the tape, while restricting its motion to left-to-right, we can generalize the model to include Mealy machines.

16.2 Transition graphs

Because a state diagram describes a *deterministic* machine, the next-state transition must be determined *uniquely* by the present state and the currently scanned input symbol. No alternative behavior is allowed. Moreover, in a deterministic state diagram a transition must be specified for each input symbol. Consequently, a state diagram consists of a vertex for every state and a directed arc labeled α emanating from each vertex for every input symbol α. However, if our prime objective is to study and classify sets of sequences, some of these restrictions may be removed and different diagrams, called transition graphs, may prove more convenient.

Nondeterministic recognizers

A *transition graph* (or *transition system*) is a directed graph. It consists of a set of vertices labeled A, B, C, etc. and various directed arcs connecting them. At least one vertex is specified as a *starting vertex* and at least one is specified as an *accepting* (or *terminal*) *vertex*. The arcs are labeled with symbols from the (*input*) *alphabet* of the graph. If the graph contains an arc labeled α leading from vertex V_i to vertex V_j then V_j is said to be the α-*successor* of V_i. For a given input symbol α, a vertex may have one or more α-successors or none. Thus, for example, in the transition graph of Fig. 16.2*b*, vertex A has two 1-successors, namely A and B, but no 0-successor. A set of vertices S is said to be the α-successor of a set R if and only if every element of S is an α-successor of some element of R.

A sequence of directed arcs in a graph is referred to as a *path*. Every path is said to *describe* the string that consists of the symbols assigned to the arcs in the path. A string is accepted by a transition graph if it is described by at least one path that emanates from a starting vertex and terminates at an accepting vertex. Thus, for example, the string 1110 is accepted by the graph of Fig. 16.3, since it is described by a path that emanates from vertex A, passes through vertices B, D, and C, and terminates at vertex A. In the same manner, we find that the string 11011 is accepted by the graph, since it is described by a path that emanates from a starting vertex B, passes through D, C, B, D, and

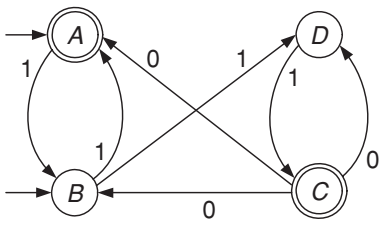


Fig. 16.3 A transition graph.

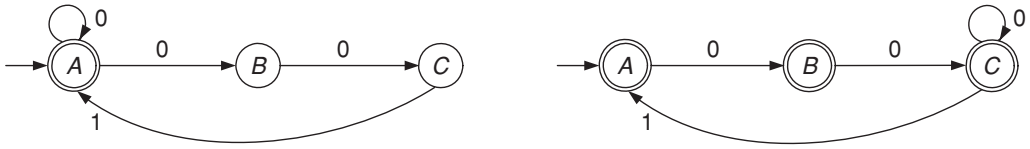


Fig. 16.4 Two equivalent transition graphs.

terminates at an accepting vertex C. However, the string 100, for example, is rejected since there is no path in the graph which describes it.

As in the case of state diagrams, the set of strings that are accepted by a transition graph is said to be *recognized* by the graph. For example, the transition graph of Fig. 16.2*b* recognizes the same set of strings as is recognized by the state diagram of Fig. 16.2*a*. If two or more graphs recognize the same set of strings then they are said to be *equivalent graphs*. Thus, the graphs in Fig. 16.4 are equivalent since each graph accepts a string if and only if each 1 in the string is preceded by at least two 0's.

Clearly, a state diagram is a special case of a transition graph and is, therefore, referred to as a *deterministic* (*transition*) *graph*. Other transition graphs are referred to as *nondeterministic* (*transition*) *graphs*. The two graphs in Fig. 16.2, for example, are equivalent although one is deterministic and the other is not. Because deterministic graphs describe the behavior of deterministic finite-state machines, we often regard nondeterministic graphs as describing the behavior of nondeterministic finite-state machines. It must, however, be emphasized that the notion of nondeterministic recognizers is useful for classifying sets of strings but should not be confused with the notion of realizable machines.

Graphs containing λ-transitions

Nondeterministic transition graphs can be generalized further by allowing transitions that are associated with a *null symbol* λ. Such transitions are referred to as λ-*transitions*, and they can occur when no input symbol is applied. When determining the string described by a path that contains arcs labeled λ, the λ-symbols are disregarded and deleted from the string.

The use of λ-transitions may sometimes simplify the transition graph by reducing the number of labeled arcs, as for the graph of Fig. 16.5*a*. This graph recognizes the set of strings that start with an even number of 1's, followed by an even number of 0's, and end up with substring 101. (Note that zero is considered as an even number.) Thus, for example, the strings 101, 11101, 110000101, and 00101 are accepted by the graph, while 110011101 and 0011101 are rejected.

It is a simple matter to convert a transition graph containing λ-transitions into an equivalent graph that contains no such transitions. A λ-transition from vertex V_1 to vertex V_2 of a given graph can always be replaced by a set of arcs emanating from V_1 and duplicating the transitions that emanate from V_2. In addition, if V_1 is a starting vertex then V_2 must also be made a starting vertex. If V_2 is an accepting vertex then V_1 must also be made an accepting

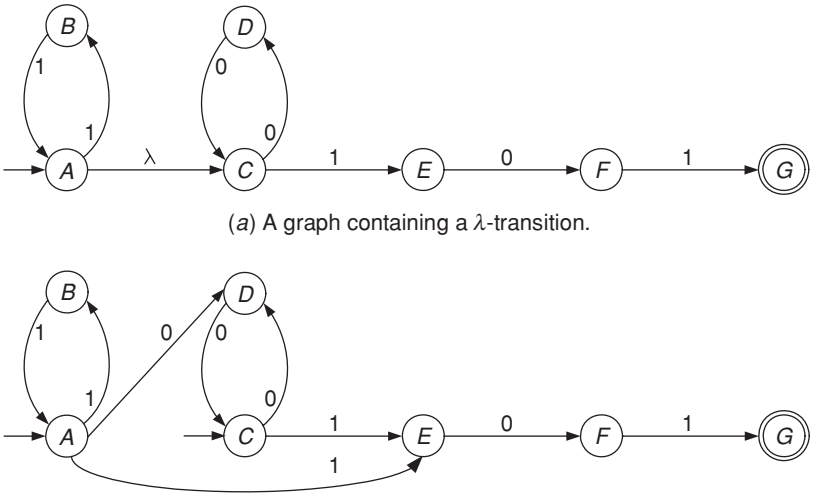


Fig. 16.5 Elimination of λ-transition.

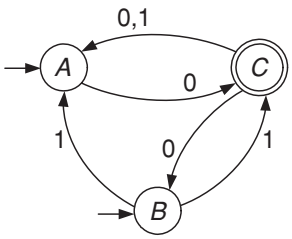
vertex. To remove the λ-transition from the graph of Fig. 16.5*a* it is necessary to duplicate the transitions from vertex C to vertices D and E by directing arcs, correspondingly labeled, from vertex A to vertices D and E. The equivalent graph that contains no λ-transition is shown in Fig. 16.5*b*.

16.3 Converting nondeterministic into deterministic graphs

A natural question, which now arises, is whether a nondeterministic graph can recognize sets of strings that cannot be recognized by a deterministic graph. At first, one might suspect that the added flexibility of nondeterministic graphs increases their computational capabilities. However, as we shall now show, *there exists an effective procedure for converting a nondeterministic transition graph into an equivalent deterministic transition graph.* This leads to the conclusion that nondeterministic graphs and deterministic graphs have identical computational capabilities.

Introductory example

Consider the nondeterministic transition graph of Fig. 16.6*a*. A tabular description of the graph, called a *transition table*, is shown in Fig. 16.6*b*, where the starting vertices are indicated by the small arrows next to rows A and B, and the accepting vertex is indicated by a circle around the row heading C. The table entry in row V_i, column α, consists of the α-successors of vertex V_i.



(*a*) Transition graph.

	0	1
→ *A*	*C*	—
→ *B*	—	*AC*
Ⓒ	*AB*	*A*

(*b*) Transition table.

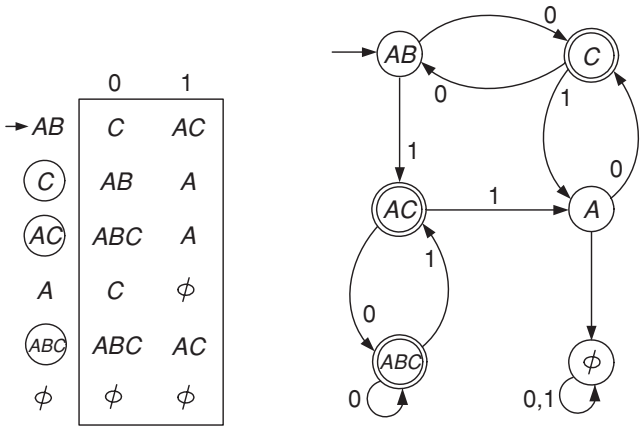
Fig. 16.6 A nondeterministic graph to be converted to a deterministic one.

Suppose now that we wish to determine whether a given string $w = a_1 a_2 \cdots a_k$ is accepted by the graph of Fig. 16.6*a*; that is, whether the graph contains a path that emanates from a starting vertex, terminates at an accepting vertex, and describes the string w. Since A and B are the starting vertices, any such path must include as its first arc an arc emanating from either A or B. Specifically, if the first symbol in w is a_1 then the first arc in the path can reach any vertex in the subset that consists of the a_1-successors of $\{A, B\}$. Using similar reasoning, we find that the ith arc in a path that describes w must lead to a vertex contained in the subset which consists of the $a_1 a_2 \cdots a_i$-successors of $\{A, B\}$. If the final subset of vertices reached by the path contains an accepting vertex then the string w is accepted; otherwise, it is rejected.

For example, any path that describes string 0010 must start with the arc leading from vertex A to vertex C. Also, since the 0-successors of C are A and B, one of these vertices must be encountered next in the path describing the given string. In the same manner, since $\{AC\}$ is the 1-successor of $\{AB\}$, we find that the third arc in the path leads to either of the vertices A or C. The fourth symbol might lead to one of the vertices A, B, or C and, since vertex C is an accepting vertex, the string is accepted. A similar argument shows, for example, that the string 1100 is rejected, since it might lead to either vertex A or vertex B and neither vertex is an accepting vertex.

The foregoing example suggests a procedure for determining whether a specified string is accepted by a given graph. The procedure involves tracing the various paths that describe the given string and determining the sets of vertices that can be reached from the starting vertices by applying the symbols of the string. The procedure can be facilitated and applied to arbitrary strings by the use of a *successor table*, which lists all the subsets of vertices that are reachable from the starting vertices. The successor table for the graph of Fig. 16.6 is shown in Fig. 16.7*a*. Its column headings are symbols of the alphabet. The first row heading is the set of starting vertices, while the remaining row headings are subsets of vertices reachable from starting vertices. The entry in row Q, column α, is determined from the transition table and consists of the α-successor of $\{Q\}$.

The first row heading in Fig. 16.7*a* is AB, since A and B are the starting vertices. The entries in row AB are the 0- and 1-successors of $\{AB\}$, namely



(*a*) Successor table. (*b*) State diagram of an equivalent deterministic machine.

Fig. 16.7 Deterministic form of the graph of Fig. 16.6.

$\{C\}$ and $\{AC\}$, respectively. The entries C and AC are now made row headings, their successors found, and so on. Since vertex A has no 1-successor, the 1-successor of row A must correspond to the set that contains no vertex of the transition graph. Such a set is referred to as the *empty*, or *null*, set and is denoted ϕ. Finally, the row headings of the rows C, AC, and ABC are circled to indicate that each of the sets $\{C\}$, $\{AC\}$, and $\{ABC\}$ contains the accepting vertex C of the original transition graph.

Proof of the conversion procedure

The graph in Fig. 16.7*b* is derived directly from the successor table. It is clearly a deterministic graph, since only one transition is allowed for each input symbol in its construction. To verify that this graph indeed accepts a given string if and only if that string is accepted by the corresponding nondeterministic graph, note that the last vertex of the deterministic graph reached by the string corresponds to the subset of vertices that can be reached by the same string in the nondeterministic graph. The string is accepted by the deterministic graph if and only if there is at least one path in the nondeterministic graph that results in the string being accepted, that is, if one vertex reachable by the string is an accepting vertex. The foregoing procedure, which is also known as *subset construction*, can be applied to any nondeterministic graph. Thus, we arrive at the following theorem.

Theorem 16.1 *Let S be a set of strings that can be recognized by a nondeterministic transition graph* G_n. *Then S can also be recognized by an equivalent deterministic graph* G_d. *Moreover, if* G_n *has p vertices then* G_d *will have at most* 2^p *vertices.*

Proof The existence of a deterministic graph G_d that is equivalent to the given nondeterministic graph G_n is guaranteed by the subset construction procedure developed above. If we denote the p vertices of G_n by $V_1, V_2, \ldots, V_p$, then, by subset construction, the equivalent deterministic graph may have at most 2^p vertices labeled as follows: ϕ, $V_1, V_2, \ldots, V_p$; V_1V_2, $V_1V_3, \ldots, V_2V_3, \ldots, V_{p-1}V_p$; $V_1V_2V_3, \ldots, V_{p-2}V_{p-1}V_p; \ldots;$ $V_1V_2 \ldots V_p$. ◇

Theorem 16.1 permits us to describe deterministic finite-state machines by means of nondeterministic transition graphs. Such descriptions will prove very convenient in the following discussion of regular expressions.

16.4 Regular expressions

In this chapter we are mainly concerned with the characterization of sets of strings recognized by finite automata. It is therefore appropriate to develop a compact language for describing such sets of strings. The language developed in this section is known as *type-3 language* or as the language of *regular expressions*.

Describing sets of strings

We shall first consider informally some sets recognized by simple graphs, leaving the formal presentation to subsequent sections. Consider the transition graph in Fig. 16.8*a*, which recognizes a set $\{101\}$ that contains just one string. We shall describe the set $\{101\}$ by the expression **101**.[2] Similarly, for an arbitrary alphabet $\{a, b\}$, the set $\{abba\}$ is described by the expression **abba**, and so on.

The graph in Fig. 16.8*b* recognizes the set of strings $\{01, 10\}$, that consists of two strings, 01 and 10. To represent such a set we employ the set union operation $+$, and express the set $\{01, 10\}$ as $\mathbf{01} + \mathbf{10}$. In the same manner, the set $\{abb, a, b, bba\}$ can be described by the expression $\mathbf{abb} + \mathbf{a} + \mathbf{b} + \mathbf{bba}$. Clearly, since the set union operation is commutative and associative, the union operation of expressions is also commutative and associative.

Next, consider the graph in Fig. 16.8*c*, which recognizes the set $\{0111, 1011\}$. This set can be described by the expression $\mathbf{0111} + \mathbf{1011}$. However, we observe that this graph recognizes precisely those strings that are recognized by the graph in Fig. 16.8*b and* which are followed immediately by the substring 11. In other words, the graph of Fig. 16.8*c* recognizes the set whose members are those strings formed by concatenating the strings in $\{01, 10\}$ and $\{11\}$. In general, the *concatenation* of two sets $\{P\}$ and $\{Q\}$ is the set

[2] In this chapter, boldface type is used to describe expressions.

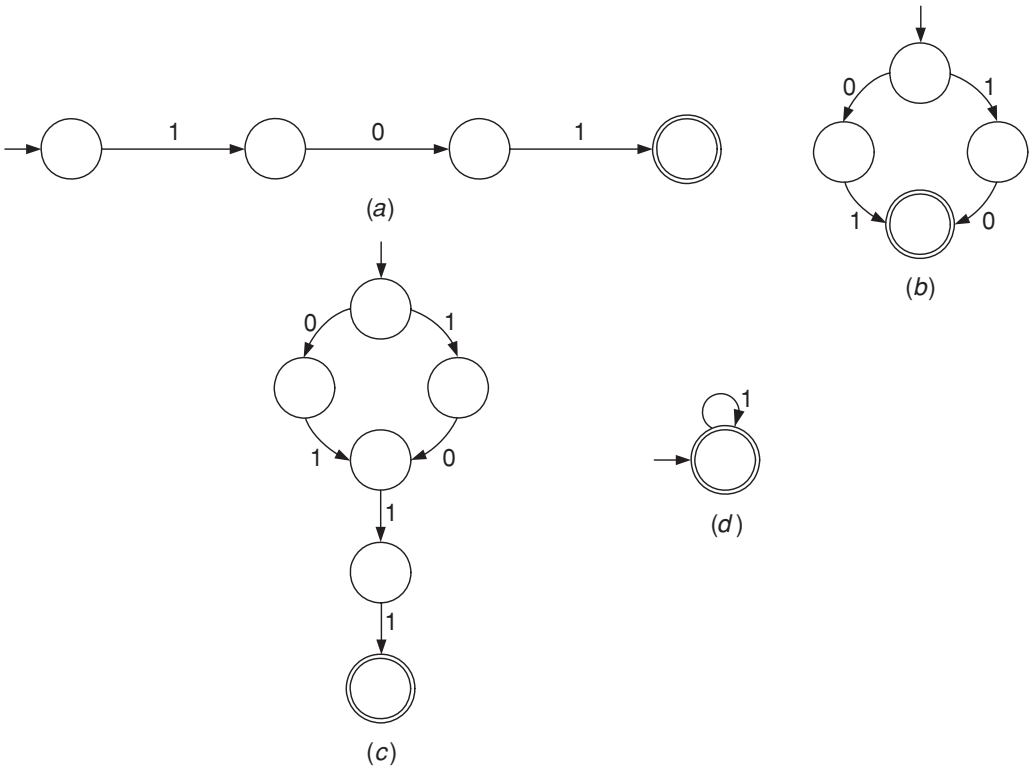


Fig. 16.8 Simple transition graphs.

consisting of strings formed by taking any string of $\{P\}$ and attaching to it any string of $\{Q\}$. The above set can thus be described by the *concatenation* of the two corresponding expressions $\mathbf{01} + \mathbf{10}$ and $\mathbf{11}$, i.e., $(\mathbf{01} + \mathbf{10})\mathbf{11}$. Clearly the concatenation operation is associative, that is, if **P**, **Q**, and **R** are expressions then $(\mathbf{PQ})\mathbf{R} = \mathbf{P}(\mathbf{QR})$, but it is not commutative, $\mathbf{PQ} \neq \mathbf{QP}$. To simplify the notation, we can omit the parentheses and write the product $(\mathbf{PQ})\mathbf{R}$ as **PQR**.

The graph in Fig. 16.8*d* recognizes the set of strings whose members consist of an arbitrary number (possibly zero) of 1's, i.e., $\{\lambda, 1, 11, 111, 1111, \ldots\}$. This set can be described by the infinite expression $\boldsymbol{\lambda} + \mathbf{1} + \mathbf{11} + \mathbf{111} + \mathbf{1111} + \cdots$ or, compactly, by $\mathbf{1}^*$, where

$$\mathbf{1}^* = \boldsymbol{\lambda} + \mathbf{1} + \mathbf{11} + \mathbf{111} + \mathbf{1111} + \cdots.$$

The symbol * is referred to as the *star* (or *closure*) *operation*. In general, $\mathbf{R}^*$ describes the set consisting of the null string λ and those strings that can be formed by concatenating a finite number of strings from $\{R\}$. For example, the expression $\mathbf{01}(\mathbf{01})^*$ describes the set consisting of those strings that can be formed by concatenating one or more 01 substrings, that is,

$$\mathbf{01}(\mathbf{01})^* = \mathbf{01} + \mathbf{0101} + \mathbf{010101} + \mathbf{01010101} + \cdots.$$

For convenience, **RR** may be abbreviated as $\mathbf{R}^2$, **RRR** as $\mathbf{R}^3$, etc. Thus,

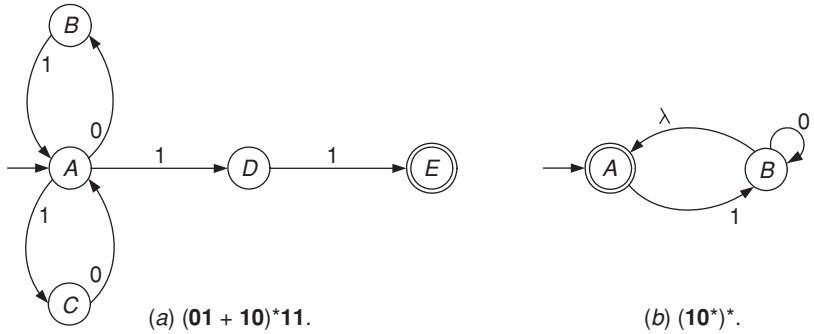
$$\mathbf{R}^* = \boldsymbol{\lambda} + \mathbf{R} + \mathbf{R}^2 + \mathbf{R}^3 + \cdots.$$

We are now able to describe some sets of strings on a given alphabet by means of the operations + , ·, *. For example, the set of strings on {0, 1} beginning with a 0 and followed only by 1's can be described by $\mathbf{01}^*$ while the set of strings containing exactly two 1's can be described by $\mathbf{0}^*\mathbf{10}^*\mathbf{10}^*$. An important expression is $(\mathbf{0} + \mathbf{1})^*$, which describes the set containing all the strings that can be formed on the binary alphabet; that is,

$$(\mathbf{0} + \mathbf{1})^* = \boldsymbol{\lambda} + \mathbf{0} + \mathbf{1} + \mathbf{00} + \mathbf{01} + \mathbf{11} + \mathbf{10} + \mathbf{000} + \cdots.$$

Thus, for example, the set of strings that begin with the substring 11 is described by the expression $\mathbf{11}(\mathbf{0} + \mathbf{1})^*$.

Example The transition graph of Fig. 16.9*a* accepts those strings that can be formed by concatenating a finite number of 01 and 10 substrings followed by a 11. Accordingly, it can be described by the expression $(\mathbf{01} + \mathbf{10})^*\mathbf{11}$. In a similar manner, the reader can verify that the set of strings recognized by the graph of Fig. 16.9*b* can be described by $(\mathbf{10}^*)^*$.



(*a*) (**01** + **10**)***11**. (*b*) (**10***)*.

Fig. 16.9 Transition graphs and the sets of strings that they recognize.

We have thus shown that some sets of strings may be described by expressions formed of symbols from the alphabets of these sets and the operations union, concatenation, and star. We now formalize these ideas.

Definition and basic properties

Let $A = \{\alpha_1, \alpha_2, \ldots, \alpha_p\}$ be a finite *alphabet*; then the class of *regular expressions over alphabet A* is defined recursively as follows.

1. Any single *symbol* $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots, \boldsymbol{\alpha}_p$ is a regular expression, as are the *null string* $\boldsymbol{\lambda}$ and the *empty set* $\boldsymbol{\phi}$.

Fig. 16.10 Recognizers for λ and ϕ.

(*a*) A graph accepting $\boldsymbol{\lambda}$. (*b*) A graph accepting $\boldsymbol{\phi}$.

2. If **P** and **Q** are regular expressions then so is their *concatenation* **PQ** and their *union* $\mathbf{P} + \mathbf{Q}$. If **P** is a regular expression then so is its *closure* $\mathbf{P}^*$.
3. No other expressions are regular unless they can be generated in a *finite* number of applications of the above rules.

By convention, the precedence of the operations in decreasing order is *, ·, +.

At this point, it is appropriate to consider the significance of the expressions $\boldsymbol{\lambda}$ and $\boldsymbol{\phi}$. The expression $\boldsymbol{\lambda}$ describes the set that consists of just the null string. It can be recognized, for example, by the graph of Fig. 16.10*a*. Expression $\boldsymbol{\phi}$, however, describes the set that has no strings at all. In other words, $\boldsymbol{\phi}$ describes the set recognized by a graph that accepts no strings, such as the graph shown in Fig. 16.10*b*. The reader may verify that each of the following identities, which involve the expressions $\boldsymbol{\phi}$ and $\boldsymbol{\lambda}$, exhibits different ways of describing the *same* sets of strings:

$$\boldsymbol{\phi} + \mathbf{R} = \mathbf{R}, \tag{16.1}$$

$$\boldsymbol{\phi}\mathbf{R} = \mathbf{R}\boldsymbol{\phi} = \boldsymbol{\phi}, \tag{16.2}$$

$$\mathbf{R}\boldsymbol{\lambda} = \boldsymbol{\lambda}\mathbf{R} = \mathbf{R}, \tag{16.3}$$

$$\boldsymbol{\lambda}^* = \boldsymbol{\lambda}, \tag{16.4}$$

$$\boldsymbol{\phi}^* = \boldsymbol{\lambda}. \tag{16.5}$$

A set of strings that can be described by a regular expression is called a *regular set*. Not every set of strings is regular. For example, the set over the alphabet $\{0, 1\}$ that consists of k 0's (for all k), followed by a 1, followed in turn by k 0's is not regular, as will be proved later. This set can be described by the expression $\mathbf{010} + \mathbf{00100} + \mathbf{0001000} + \cdots + \mathbf{0}^k\mathbf{10}^k + \cdots$. However, such a description involves an infinite number of applications of the union operation. Consequently, it is not a regular expression. There are, however, certain infinite sums that are regular. For example, the set that consists of alternating 0's and 1's, starting and ending with a 1, i.e., {1, 101, 10101, 1010101, ...}, can be described by the expression $\mathbf{1} + \mathbf{101} + \mathbf{10101} + \cdots$, or $\mathbf{1}(\mathbf{01})^*$, which is clearly regular.

Manipulating regular expressions

A regular set may be described by more than one regular expression. For example, the above set of alternating 0's and 1's can be described by the expression $\mathbf{1}(\mathbf{01})^*$, as well as by $(\mathbf{10})^*\mathbf{1}$. Two expressions that describe the same set of strings are said to be *equivalent*. Unfortunately, no straightforward methods are

available to determine whether two given expressions are equivalent. In certain cases, however, a regular expression can be converted into another equivalent expression by the use of simple identities. Some of these identities (whose proofs are left to the reader as an exercise) are listed as follows.

Let **P**, **Q**, and **R** be regular expressions; then

$$\mathbf{R} + \mathbf{R} = \mathbf{R}, \tag{16.6}$$

$$\mathbf{PQ} + \mathbf{PR} = \mathbf{P}(\mathbf{Q} + \mathbf{R}), \quad \mathbf{PQ} + \mathbf{RQ} = (\mathbf{P} + \mathbf{R})\mathbf{Q}, \tag{16.7}$$

$$\mathbf{R}^*\mathbf{R}^* = \mathbf{R}^*, \tag{16.8}$$

$$\mathbf{RR}^* = \mathbf{R}^*\mathbf{R}, \tag{16.9}$$

$$(\mathbf{R}^*)^* = \mathbf{R}^*, \tag{16.10}$$

$$\boldsymbol{\lambda} + \mathbf{RR}^* = \mathbf{R}^*, \tag{16.11}$$

$$(\mathbf{PQ})^*\mathbf{P} = \mathbf{P}(\mathbf{QP})^*. \tag{16.12}$$

To prove the last identity, note that each of the expressions $(\mathbf{PQ})^*\mathbf{P}$ and $\mathbf{P}(\mathbf{QP})^*$ can be written in the form $\mathbf{P} + \mathbf{PQP} + \mathbf{PQPQP} + \cdots$.

The set described by the expression $(\mathbf{P} + \mathbf{Q})^*$ consists of all the strings that can be formed by concatenating **P**'s and **Q**'s, including the null string $\boldsymbol{\lambda}$. It is easy to verify that the expression $(\mathbf{P}^* + \mathbf{Q}^*)^*$ describes the same set of strings, as does the expression $(\mathbf{P}^*\mathbf{Q}^*)^*$. Thus, we find that

$$(\mathbf{P} + \mathbf{Q})^* = (\mathbf{P}^*\mathbf{Q}^*)^* = (\mathbf{P}^* + \mathbf{Q}^*)^*. \tag{16.13}$$

However, note that $(\mathbf{P} + \mathbf{Q})^* \neq \mathbf{P}^* + \mathbf{Q}^*$.

The following identity will be proved in Section 16.5:

$$(\mathbf{P} + \mathbf{Q})^* = \mathbf{P}^*(\mathbf{QP}^*)^* = (\mathbf{P}^*\mathbf{Q})^*\mathbf{P}^*. \tag{16.14}$$

This identity leads in turn to

$$\boldsymbol{\lambda} + (\mathbf{P} + \mathbf{Q})^*\mathbf{Q} = (\mathbf{P}^*\mathbf{Q})^*. \tag{16.15}$$

Indeed, by Eqs. (16.11) and (16.14),

$$\begin{aligned}(\mathbf{P}^*\mathbf{Q})^* &= \boldsymbol{\lambda} + (\mathbf{P}^*\mathbf{Q})^*\mathbf{P}^*\mathbf{Q} \\ &= \boldsymbol{\lambda} + (\mathbf{P} + \mathbf{Q})^*\mathbf{Q}.\end{aligned}$$

The preceding identities can sometimes be used to simplify regular expressions or demonstrate their equivalence, as illustrated in the following examples.

Example Prove that the set of strings in which every 0 is immediately followed by at least two 1's can be described by both $\mathbf{R}_1$ and $\mathbf{R}_2$, where

$$\mathbf{R}_1 = \boldsymbol{\lambda} + \mathbf{1}^*(\mathbf{011})^*(\mathbf{1}^*(\mathbf{011})^*)^*,$$
$$\mathbf{R}_2 = (\mathbf{1} + \mathbf{011})^*.$$

We proceed as follows.

$$\begin{aligned}\mathbf{R}_1 &= \boldsymbol{\lambda} + \mathbf{1}^*(\mathbf{011})^*(\mathbf{1}^*(\mathbf{011})^*)^* \qquad \text{(by (16.11))}\\ &= (\mathbf{1}^*(\mathbf{011})^*)^* \qquad \text{(by (16.13))}\\ &= (\mathbf{1}+\mathbf{011})^* = \mathbf{R}_2.\end{aligned}$$

The reader can verify that R_2 indeed describes the set in question.

Example Prove the identity

$$(\mathbf{1}+\mathbf{00}^*\mathbf{1})+(\mathbf{1}+\mathbf{00}^*\mathbf{1})(\mathbf{0}+\mathbf{10}^*\mathbf{1})^*(\mathbf{0}+\mathbf{10}^*\mathbf{1}) = \mathbf{0}^*\mathbf{1}(\mathbf{0}+\mathbf{10}^*\mathbf{1})^*.$$

Consider the left-hand side:

$$\begin{aligned}&(\mathbf{1}+\mathbf{00}^*\mathbf{1})+(\mathbf{1}+\mathbf{00}^*\mathbf{1})(\mathbf{0}+\mathbf{10}^*\mathbf{1})^*(\mathbf{0}+\mathbf{10}^*\mathbf{1})\\ &\quad= (\mathbf{1}+\mathbf{00}^*\mathbf{1})[\boldsymbol{\lambda}+(\mathbf{0}+\mathbf{10}^*\mathbf{1})^*(\mathbf{0}+\mathbf{10}^*\mathbf{1})]\\ &\quad= [(\boldsymbol{\lambda}+\mathbf{00}^*)\mathbf{1}][\boldsymbol{\lambda}+(\mathbf{0}+\mathbf{10}^*\mathbf{1})^*(\mathbf{0}+\mathbf{10}^*\mathbf{1})] \quad \text{(by (16.11))}\\ &\quad= \mathbf{0}^*\mathbf{1}(\mathbf{0}+\mathbf{10}^*\mathbf{1})^*.\end{aligned}$$

In many situations, however, algebraic manipulations of regular expressions are extremely involved and thus are not a suitable tool for determining the equivalence of two regular expressions. As we shall see in the next section, perhaps the best approach is to convert the expressions in question into their equivalent state diagrams and to test the diagrams for equivalence by the techniques of Chapter 10. Other procedures for establishing the equivalence of regular expressions can be found in [3].

16.5 Transition graphs recognizing regular sets

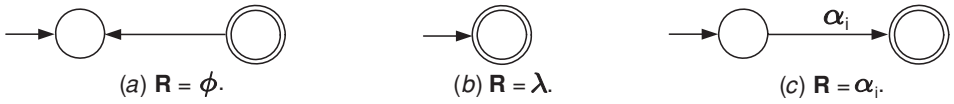
We have already seen in several examples that transition graphs are capable of recognizing regular sets. We wish to show now that to every regular set there corresponds a transition graph (and hence a deterministic finite-state machine) that recognizes that set of strings.

Constructing the transition graphs

We now prove the following theorem.

Theorem 16.2 *Every regular expression* **R** *can be recognized by a transition graph.*

Proof We shall prove the theorem by constructing the required transition graph. The construction procedure is inductive on the total number of characters in **R**, where by a *character* we refer to an appearance of any of the expressions



(*a*) $\mathbf{R} = \phi$. (*b*) $\mathbf{R} = \lambda$. (*c*) $\mathbf{R} = \alpha_i$.

Fig. 16.11 Transition graphs recognizing elementary regular sets.

$\alpha_1, \alpha_2, \ldots, \alpha_p, \lambda, \phi$ or the star operation * in $\mathbf{R}$. For example, the number of characters in $\mathbf{R} = \lambda + (\mathbf{1}^*\mathbf{0})^*\mathbf{1}^*$ is seven.

Basis Let the number of characters in $\mathbf{R}$ be one. Then $\mathbf{R}$ must be either ϕ, λ, or a symbol, say α_i, from the alphabet. The graphs in Fig. 16.11 recognize these regular sets.[3]

Induction step Assume the theorem is true for expressions with n or fewer characters. We now show that it must also be true for any expression $\mathbf{R}$ having $n + 1$ characters. The expression $\mathbf{R}$ must be in one of the following three forms:

1. $\mathbf{R} = \mathbf{P} + \mathbf{Q}$,
2. $\mathbf{R} = \mathbf{PQ}$,
3. $\mathbf{R} = \mathbf{P}^*$,

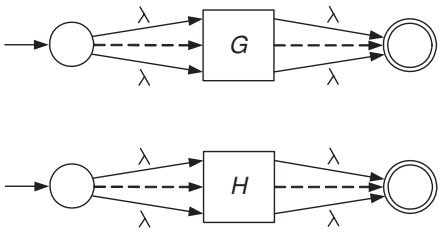
where $\mathbf{P}$ and $\mathbf{Q}$ are each expressions having n or fewer characters. According to the induction hypothesis, the sets $\mathbf{P}$ and $\mathbf{Q}$ can be recognized by transition graphs, which we shall denote G and H, respectively, as shown in Fig. 16.12*a*. (Note that each graph in Fig. 16.12 contains just one starting and one accepting vertex.)

The set described by $\mathbf{P} + \mathbf{Q}$ can be recognized by a transition graph composed of G and H, as shown in Fig. 16.12*b*. The set described by $\mathbf{PQ}$ can be recognized by a transition graph constructed in the following manner. Coalesce the accepting vertex of G with the starting vertex of H and regard the combined vertex as one that is neither starting nor accepting. The resulting graph is shown in Fig. 16.12*c*. The starting vertices of this graph are the starting vertices of G, while the accepting vertices are those of H. Clearly, this graph will accept a string if and only if that string belongs to $\mathbf{R} = \mathbf{PQ}$. Finally, to recognize the set $\mathbf{P}^*$, construct the graph of Fig. 16.12*d*. The graphs in Fig. 16.12, which are composed of G and H, are referred to as *composite graphs*.

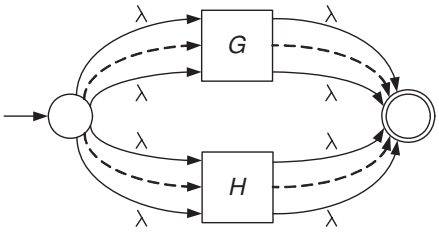
Since every regular set can be described by an expression obtained by a finite number of applications of operations $+$, $\cdot$, * on an alphabet $\{\alpha_1, \alpha_2, \ldots, \alpha_p\}$, ϕ and λ, the theorem is proved. ◇

The foregoing proof makes it possible to state an upper bound on the number of vertices in a graph that recognizes a given regular expression $\mathbf{R}$. Every graph clearly contains one starting and one accepting vertex. Subexpressions connected by the $+$ operation yield a composite graph that has as many vertices as the sum of vertices in the graphs that recognize individual subexpressions.

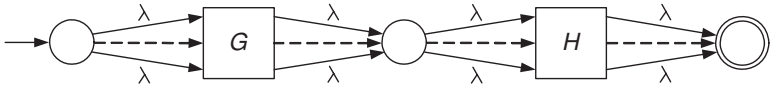
[3] Although there is a distinction between regular expressions and the sets that they describe, it is customary to speak of the regular set $\mathbf{R}$ as the set that can be described by the expression $\mathbf{R}$.



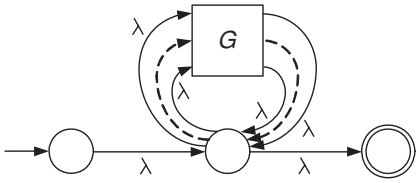
(*a*) Graphs recognizing **P** and **Q**.



(*b*) A graph recognizing **P** + **Q**.



(*c*) A graph recognizing **PQ**.



(*d*) A graph recognizing **P***.

Fig. 16.12 Construction of composite graphs.

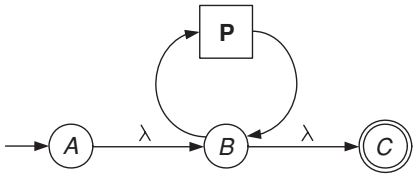
Two subexpressions connected by the concatenation operation add a new vertex to the composite graph, and similarly for the closure operation *. By induction on the number of vertices, we find that the number of vertices v in a graph that recognizes the given expression **R** need not exceed

$$v = 2 + \text{number of concatenations} + \text{number of stars}.$$

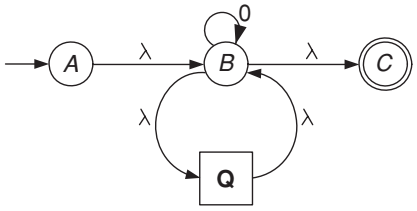
Theorem 16.2 provides us with a procedure for constructing a transition graph that recognizes a given regular expression **R**. Converting the graph to a deterministic form yields a state diagram of a finite-state machine that recognizes the set **R**.

Example Consider the regular expression $\mathbf{R} = (\mathbf{0} + \mathbf{1}(\mathbf{01})^*)^*$. Since it is of the form $\mathbf{P}^*$, where $\mathbf{P} = \mathbf{0} + \mathbf{1}(\mathbf{01})^*$, it is recognized by the graph of Fig. 16.13*a*. We now observe that $\mathbf{P} = \mathbf{0} + \mathbf{Q}$, where $\mathbf{Q} = \mathbf{1}(\mathbf{01})^*$, and the resulting graph is shown in Fig. 16.13*b*. The subexpression **Q** can be decomposed into $\mathbf{Q} = \mathbf{ST}$, where $\mathbf{S} = \mathbf{1}$ and $\mathbf{T} = (\mathbf{01})^*$. This yields the graph of Fig. 16.13*c*. The process is continued in a similar manner until each subexpression consists of only a single symbol. The final transition graph that

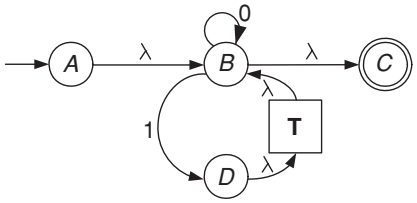
recognizes **R** is shown in Fig. 16.13*d*. Note that the number of vertices in the graph is six, in agreement with the value of v derived above.



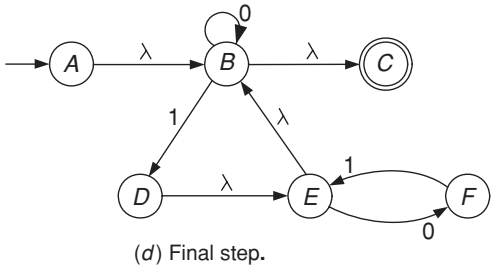
(*a*) **R** = **P***; **P** = **0** + **1**(**01**)*.



(*b*) **P** = **0** + **Q**; **Q** = **1**(**01**)*.



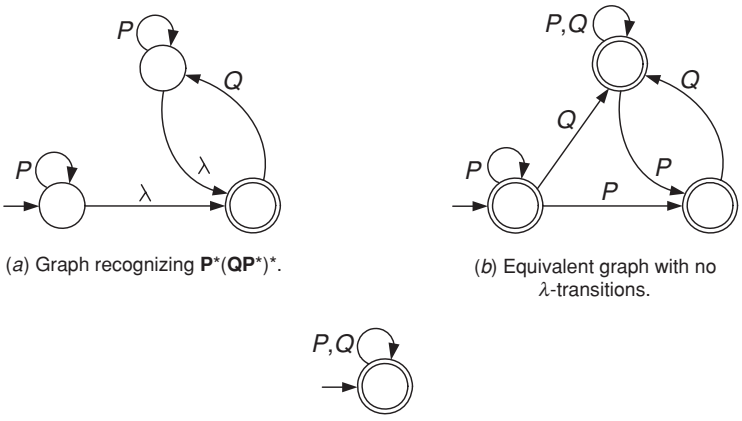
(*c*) **Q** = **1T**; **T** = (**01**)*.



(*d*) Final step.

Fig. 16.13 Construction of a transition graph recognizing $\mathbf{R} = (\mathbf{0} + \mathbf{1}(\mathbf{01})^*)^*$.

We can now prove the first identity in Eq. (16.14) by demonstrating that the expressions $(\mathbf{P} + \mathbf{Q})^*$ and $\mathbf{P}^*(\mathbf{QP}^*)^*$ can be recognized by equivalent transition graphs. The graph in Fig. 16.14*a* recognizes the set described by $\mathbf{P}^*(\mathbf{QP}^*)^*$. Removal of the λ-transitions results in the graph of Fig. 16.14*b*, which can be converted to the deterministic graph of Fig. 16.14*c*. Clearly this graph recognizes set $(\mathbf{P} + \mathbf{Q})^*$, and thus the two expressions are equivalent. By Eq. (16.12), we obtain $\mathbf{P}^*(\mathbf{QP}^*)^* = (\mathbf{P}^*\mathbf{Q})^*\mathbf{P}^*$, which proves the second identity.



(*a*) Graph recognizing $\mathbf{P}^*(\mathbf{QP}^*)^*$.

(*b*) Equivalent graph with no λ-transitions.

(*c*) Equivalent deterministic graph recognizing $(\mathbf{P}+\mathbf{Q})^*$.

Fig. 16.14 Illustration of the proof that $\mathbf{P}^*(\mathbf{QP}^*)^* = (\mathbf{P}+\mathbf{Q})^*$.

Informal techniques

In practice, in many cases it is possible to construct transition graphs from their corresponding regular expressions in a straightforward manner, without resorting to the above induction procedure.

Example Construct a graph that recognizes the regular set

$$\mathbf{P} = (\mathbf{01} + (\mathbf{11} + \mathbf{0})\mathbf{1}^*\mathbf{0})^*\mathbf{11}.$$

As an introduction, we shall construct a graph that recognizes the subexpression $\mathbf{Q} = (\mathbf{11} + \mathbf{0})\mathbf{1}^*\mathbf{0}$. Every string in $\mathbf{Q}$ starts with one of the substrings 11 and 0, followed by an arbitrary number of 1's, and ends with a 0. The graph of Fig. 16.15 clearly recognizes just this set of strings. The subexpressions **11** and **0** are represented by parallel paths between the vertices A and C, while 1^* corresponds to a self-loop around vertex C. To ensure that a string is accepted only if it ends with a 0, an arc labeled 0 leads from vertex C to accepting vertex D.

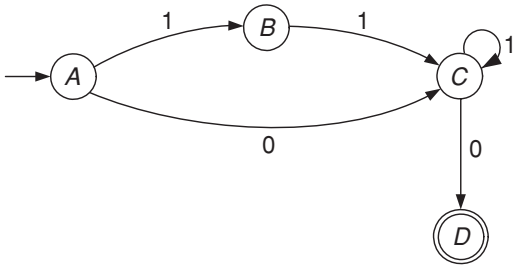


Fig. 16.15 A graph recognizing $\mathbf{Q} = (\mathbf{11} + \mathbf{0})\mathbf{1}^*\mathbf{0}$.

Now consider expression **P**. The graph that recognizes **P** is constructed in such a way that paths are provided for strings from the sets **01** and $(\mathbf{11} + \mathbf{0})\mathbf{1}^*\mathbf{0}$, followed by a string from the set **11**. One such possible graph is shown in Fig. 16.16.

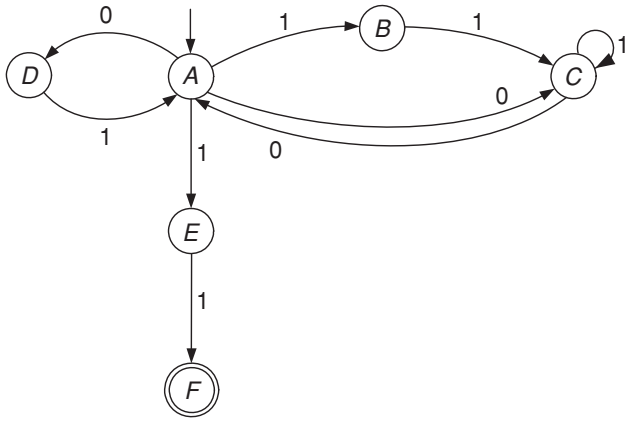


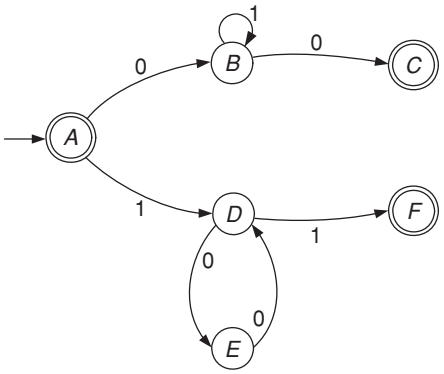
Fig. 16.16 A graph recognizing $\mathbf{P} = (\mathbf{01} + (\mathbf{11} + \mathbf{0})\mathbf{1}^*\mathbf{0})^*\mathbf{11}$.

In a number of cases it is convenient to use λ-transitions to preserve the order in which substrings appear. As an example, consider the expression $\mathbf{R} = (\mathbf{11})^*(\mathbf{00})^*\mathbf{101}$. In this expression, substrings from $(\mathbf{00})^*$ must follow substrings from $(\mathbf{11})^*$. One way of ensuring that this order is preserved is by using a λ-transition, as shown in Fig. 16.5*a*. This graph accepts only those strings that start with a substring from $(\mathbf{11})^*$, continue with a substring from $(\mathbf{00})^*$, and end with the substring 101.

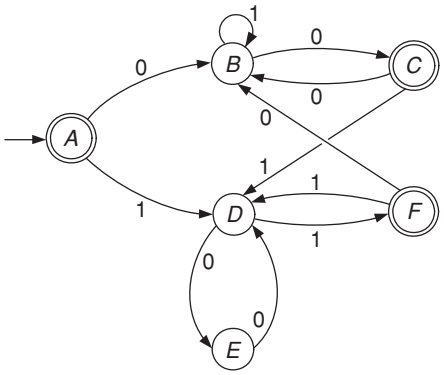
Example Construct a transition graph that recognizes the set

$$\mathbf{R} = (\mathbf{1}(\mathbf{00})^*\mathbf{1} + \mathbf{01}^*\mathbf{0})^*.$$

We begin by setting up paths for the subexpressions $\mathbf{1}(\mathbf{00})^*\mathbf{1}$ and $\mathbf{01}^*\mathbf{0}$, as shown in Fig. 16.17*a*. Vertex *A* is the starting vertex, while *A*, *C*, and *F* are accepting vertices. To complete the graph, an arc labeled α_i is drawn from vertex V_j to vertex V_k if and only if a sequence leading from the starting vertex to V_j that is followed by α_i and then by a sequence that emanates from V_k to an accepting vertex is an acceptable sequence. Accordingly, for example, an arc labeled 0 is drawn from *F* to *B* since 1100 is an acceptable sequence. The graph is completed in a similar manner, as shown in Fig. 16.17*b*.



(*a*) Partial graph.



(*b*) Complete graph.

Fig. 16.17 Transition graph recognizing $\mathbf{R} = (\mathbf{1}(\mathbf{00})^*\mathbf{1} + \mathbf{01}^*\mathbf{0})^*$.

In conclusion, we have established that every regular set can be recognized by a finite-state machine. Moreover, there is a routine procedure for determining the machine that recognizes a given regular set. This procedure involves the use of nondeterministic transition graphs, which can later be converted into the equivalent deterministic graphs. Other methods, however, are available [6] that provide a state-diagram description of the machine directly, without the need to resort to nondeterministic graphs.

16.6 Regular sets corresponding to transition graphs

We now consider the problem of deriving regular expressions that describe specified transition graphs. Specifically, we shall show that the set of strings that can be recognized by a transition graph (and hence a finite-state machine) is a regular set.

Proof of uniqueness

Before proceeding with our main topic, we shall establish the following theorem.

Theorem 16.3 *Let* $\mathbf{Q}$, $\mathbf{P}$, *and* $\mathbf{R}$ *be regular expressions on a finite alphabet. Then, if* $\mathbf{P}$ *does not contain* $\boldsymbol{\lambda}$, *the equation*

$$\mathbf{R} = \mathbf{Q} + \mathbf{RP} \tag{16.16}$$

has a unique solution given by

$$\mathbf{R} = \mathbf{QP}^*. \tag{16.17}$$

Proof Clearly, $\mathbf{R} = \mathbf{QP}^*$ is a solution to the equation $\mathbf{R} = \mathbf{Q} + \mathbf{RP}$, since (by substitution and Eq. (16.11))

$$\mathbf{R} = \mathbf{Q} + \mathbf{RP} = \mathbf{Q} + \mathbf{QP}^*\mathbf{P} = \mathbf{Q}(\boldsymbol{\lambda} + \mathbf{P}^*\mathbf{P}) = \mathbf{QP}^*.$$

To prove uniqueness, make the expansion

$$\begin{aligned}
\mathbf{R} &= \mathbf{Q} + \mathbf{RP} \\
&= \mathbf{Q} + (\mathbf{Q} + \mathbf{RP})\mathbf{P} = \mathbf{Q} + \mathbf{QP} + \mathbf{RP}^2 \\
&= \mathbf{Q} + \mathbf{QP} + (\mathbf{Q} + \mathbf{RP})\mathbf{P}^2 = \mathbf{Q} + \mathbf{QP} + \mathbf{QP}^2 + \mathbf{RP}^3 \\
&\;\;\vdots \\
&= \mathbf{Q}(\boldsymbol{\lambda} + \mathbf{P} + \mathbf{P}^2 + \cdots + \mathbf{P}^{i-1} + \mathbf{P}^i) + \mathbf{RP}^{i+1},
\end{aligned} \tag{16.18}$$

where i is any arbitrary integer. Choose some string w in $\mathbf{R}$, suppose that the length of w is k, and then substitute $i = k$ into Eq. (16.18):

$$\mathbf{R} = \mathbf{Q}(\boldsymbol{\lambda} + \mathbf{P} + \mathbf{P}^2 + \cdots + \mathbf{P}^k) + \mathbf{RP}^{k+1}.$$

Since $\mathbf{P}$ does not contain $\boldsymbol{\lambda}$, the length of the shortest string in the set $\mathbf{RP}^{k+1}$ is at least $k + 1$. Consequently, w is not contained in $\mathbf{RP}^{k+1}$, but is contained in $\mathbf{Q}(\boldsymbol{\lambda} + \mathbf{P} + \mathbf{P}^2 + \cdots + \mathbf{P}^k)$. However, since $\mathbf{Q}(\boldsymbol{\lambda} + \mathbf{P} + \mathbf{P}^2 + \cdots + \mathbf{P}^k)$ is contained in $\mathbf{QP}^*$, w is contained in $\mathbf{QP}^*$.

To prove the converse, suppose that w is a string in $\mathbf{QP}^*$. Then there exists some integer k such that w is in $\mathbf{QP}^k$. This, in turn, implies that w is contained in $\mathbf{Q}(\boldsymbol{\lambda} + \mathbf{P} + \mathbf{P}^2 + \cdots + \mathbf{P}^k)$ and hence in $\mathbf{R} = \mathbf{Q} + \mathbf{RP}$. ◇

In an analogous manner, we can show that if $\mathbf{P}$ does not contain $\boldsymbol{\lambda}$ then $\mathbf{R} = \mathbf{P}^*\mathbf{Q}$ is the unique solution to the equation $\mathbf{R} = \mathbf{Q} + \mathbf{PR}$. Note that if $\mathbf{P}$ contains $\boldsymbol{\lambda}$, the solution of Eq. (16.16) is not unique. If $\mathbf{P} = \boldsymbol{\phi}$ then $\mathbf{R} = \mathbf{Q}$.

*Systems of equations

Consider the transition graph of Fig. 16.18, whose starting vertex is A and accepting vertex C. The set of strings recognized by this graph consists of all the strings that can be described by paths emanating from vertex A and

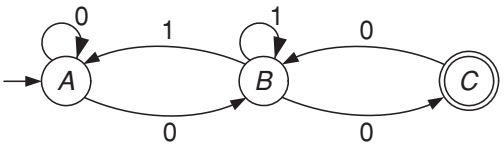


Fig. 16.18 A transition graph to be analyzed.

terminating at vertex C. However, since vertex C can be reached only through vertex B, each of these strings must end with a 0 and have as prefix a string leading from A to B. Let us denote the set of strings leading from A to B by $\mathbf{B}$ and the set of strings that take the graph from A to C by $\mathbf{C}$. Set $\mathbf{C}$ can then be expressed as $\mathbf{C} = \mathbf{B0}$.

Next consider set $\mathbf{A}$, which consists of exactly those strings that take the graph from vertex A to itself. Vertex A can be reached from B with a 1, from A with a 0, and with the null string λ. Thus, $\mathbf{A}$ can be expressed as $\mathbf{A} = \boldsymbol{\lambda} + \mathbf{A0} + \mathbf{B1}$. Finally, vertex B can be reached from A with a 0, from B with a 1, and from C with a 0. As a result, we obtain the equation $\mathbf{B} = \mathbf{A0} + \mathbf{B1} + \mathbf{C0}$.

The foregoing analysis yields a system of three simultaneous equations which characterize the sets of strings that take the graph from its starting vertex to each of its vertices. In Theorem 16.4 we shall prove that each of these sets of strings is regular, i.e.,

$$\mathbf{A} = \boldsymbol{\lambda} + \mathbf{A0} + \mathbf{B1}, \tag{16.19}$$

$$\mathbf{B} = \mathbf{A0} + \mathbf{B1} + \mathbf{C0}, \tag{16.20}$$

$$\mathbf{C} = \mathbf{B0}. \tag{16.21}$$

These equations can now be solved for the variables $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$. Substituting Eq. (16.21) for $\mathbf{C}$ into Eq. (16.20) yields

$$\mathbf{B} = \mathbf{A0} + \mathbf{B1} + \mathbf{B00} = \mathbf{A0} + \mathbf{B}(\mathbf{1} + \mathbf{00}). \tag{16.22}$$

Equation (16.22) is now of the form of Eq. (16.16),

$$\mathbf{R} = \mathbf{Q} + \mathbf{RP},$$

and its solution is given by Eq. (16.17), i.e.,

$$\mathbf{R} = \mathbf{QP}^*.$$

Applying Eq. (16.17) to Eq. (16.22), we obtain

$$\mathbf{B} = \mathbf{A0}(\mathbf{1} + \mathbf{00})^*. \tag{16.23}$$

Now $\mathbf{B}$ can be substituted into Eq. (16.19) to give

$$\mathbf{A} = \boldsymbol{\lambda} + \mathbf{A0} + \mathbf{A0}(\mathbf{1} + \mathbf{00})^*\mathbf{1} = \boldsymbol{\lambda} + \mathbf{A}(\mathbf{0} + \mathbf{0}(\mathbf{1} + \mathbf{00})^*\mathbf{1}). \tag{16.24}$$

Equation (16.24) is again of the general form of Eq. (16.16) and, thus, has the solution

$$\mathbf{A} = \boldsymbol{\lambda}(\mathbf{0} + \mathbf{0}(\mathbf{1} + \mathbf{00})^*\mathbf{1})^* = (\mathbf{0} + \mathbf{0}(\mathbf{1} + \mathbf{00})^*\mathbf{1})^*. \tag{16.25}$$

Since the set recognized by the graph is given by **C**, we want to find a solution for this variable. Substituting Eq. (16.25) for **A** into Eq. (16.23), we obtain a solution for **B** that, in turn, may be substituted into Eq. (16.21) to yield the solution for **C**, i.e.,

$$\mathbf{B} = (\mathbf{0} + \mathbf{0}(\mathbf{1} + \mathbf{00})^*\mathbf{1})^*\mathbf{0}(\mathbf{1} + \mathbf{00})^*, \tag{16.26}$$

$$\mathbf{C} = (\mathbf{0} + \mathbf{0}(\mathbf{1} + \mathbf{00})^*\mathbf{1})^*\mathbf{0}(\mathbf{1} + \mathbf{00})^*\mathbf{0}. \tag{16.27}$$

The above procedure can now be applied to find a system of simultaneous equations for any transition graph that contains no λ-transitions and has a single starting vertex. (Recall that every transition graph can be converted to an equivalent graph with no λ-transitions and just one starting vertex.) Suppose that V_1 is the starting vertex in a graph containing n vertices, $V_1, V_2, \ldots, V_n$. Let $\mathbf{V}_i$ denote the set of strings that take the graph from V_1 to V_i, and let $\boldsymbol{\alpha}_{ij}$ denote the set of strings that take the graph from vertex V_i to vertex V_j without going through any other vertex; $\boldsymbol{\alpha}_{ij} = \boldsymbol{\phi}$ if no direct transition exists from V_i to V_j. Then we arrive at the following equations:

$$\begin{aligned}
\mathbf{V}_1 &= \mathbf{V}_1\boldsymbol{\alpha}_{11} + \mathbf{V}_2\boldsymbol{\alpha}_{21} + \cdots + \mathbf{V}_n\boldsymbol{\alpha}_{n1} + \boldsymbol{\lambda}, \\
\mathbf{V}_2 &= \mathbf{V}_1\boldsymbol{\alpha}_{12} + \mathbf{V}_2\boldsymbol{\alpha}_{22} + \cdots + \mathbf{V}_n\boldsymbol{\alpha}_{n2}, \\
&\vdots \\
\mathbf{V}_n &= \mathbf{V}_1\boldsymbol{\alpha}_{1n} + \mathbf{V}_2\boldsymbol{\alpha}_{2n} + \cdots + \mathbf{V}_n\boldsymbol{\alpha}_{nn}.
\end{aligned} \tag{16.28}$$

This system of equations can now be solved for $\mathbf{V}_1, \mathbf{V}_2, \ldots, \mathbf{V}_n$ by repeated substitution and successive applications of Eq. (16.17) in the following manner. Whenever an equation is of the form $\mathbf{V}_i = \mathbf{V}_j\boldsymbol{\alpha}_{ji} + \mathbf{V}_k\boldsymbol{\alpha}_{ki}$ or $\mathbf{V}_i = \mathbf{V}_j\boldsymbol{\alpha}_{ji} + \mathbf{V}_k\boldsymbol{\alpha}_{ki} + \boldsymbol{\lambda}$, where $i \neq j \neq k$, then $\mathbf{V}_i$ can be substituted into all other equations to yield a system with fewer equations and unknowns. Whenever an equation has the form $\mathbf{V}_i = \mathbf{V}_i\boldsymbol{\alpha}_{ii} + \mathbf{V}_j\boldsymbol{\alpha}_{ji}$ (plus $\boldsymbol{\lambda}$ if appropriate), then Eq. (16.17) can be applied to yield $\mathbf{V}_i = \mathbf{V}_j\boldsymbol{\alpha}_{ji}(\boldsymbol{\alpha}_{ii})^*$, which can now be substituted for $\mathbf{V}_i$ in the other equations. Note that, since the graph is assumed to contain no λ-transitions, the condition in Theorem 16.3 that $\boldsymbol{\alpha}_{ii}$ should not contain $\boldsymbol{\lambda}$ can always be met. This procedure will finally lead to a single equation in one variable. This variable can in turn be determined by another application of Eq. (16.17).

The set of strings recognized by a given graph can be described by the union of the **V**'s that correspond to accepting vertices. For example, if vertices B and C in the graph of Fig. 16.18 were accepting vertices then the set of strings recognized by the graph could be described by $\mathbf{B} + \mathbf{C} = (\mathbf{0} + \mathbf{0}(\mathbf{1} + \mathbf{00})^*\mathbf{1})^*\mathbf{0}(\mathbf{1} + \mathbf{00})^*(\boldsymbol{\lambda} + \mathbf{0})$.

Clearly, any system of equations of the form Eq. (16.28) can be uniquely solved by the procedure just outlined, provided that we prove that each of the $\mathbf{V}_i$'s and $\boldsymbol{\alpha}_{ij}$'s is a regular expression. This proof is given in the following theorem.

Theorem 16.4 *The set of strings that take a finite-state machine M from an arbitrary state S_i to another state S_j is a regular set.*

Proof Let Q be any subset of the states of M containing both S_i and S_j, and let R_{ij}^Q denote the set of strings that take the machine from state S_i to state S_j without passing through any state that is outside Q. Since Q may consist of all the states in M, the theorem will be proved if we can show that R_{ij}^Q is regular. The proof will be by induction on the number of states in Q.

Basis Suppose that Q consists of just a single state, which we shall call S_i. Then the set of strings that take S_i into itself without passing through any other state consists of only a finite number of single input symbols. Since by definition each such input symbol is regular, the above set of strings is regular. The corresponding regular expression will be denoted $\mathbf{T}_{ii}$.

Induction step Assume that R_{ij}^Q is regular for all subsets of states containing m or fewer states. Thus, R_{ij}^Q can be described by the regular expression $\mathbf{R}_{ij}^Q$. We shall now prove that the set of strings R_{ij}^P is also regular, where P is a set containing $m+1$ states, including the states S_i and S_j. Suppose now that we remove state S_i from P. The resulting subset consists of only m states and will be referred to as Q; the theorem is assumed to hold for this subset.

Consider a string from R_{ij}^P. In general, it will cause the machine to go through state transitions as follows:

$$S_i, S_t, \ldots, S_u, S_i, \ldots, S_i, \ldots, S_j$$

where the ellipses correspond to transitions within set Q and therefore do not contain occurrences of S_i. The substrings that take the machine from S_i and back into S_i may consist of either single input symbols from the regular set $\mathbf{T}_{ii}$ or of sequences of symbols that take the machine from S_i through some states, say $S_t, \ldots, S_u$, and back into S_i. Such an input sequence actually consists of a single symbol, denoted T_{it}, that takes M from S_i to S_t followed by a sequence from $\mathbf{R}_{tu}^Q$ and ending with a symbol T_{ui} that returns M to S_i. Each of the symbols T_{it} and T_{ui} is clearly regular and, consequently, the set of strings that take M from S_i into S_i can be described by the regular expression

$$\mathbf{T}_{ii} + \sum_{tu} \mathbf{T}_{it}\mathbf{R}_{tu}^Q\mathbf{T}_{ui},$$

where the sum is taken over all possible pairs of states in Q. In addition, since the machine can be taken an arbitrary number of times from S_i through states in Q and back into S_i, the set of corresponding strings can be described by the regular expression

$$\left(\mathbf{T}_{ii} + \sum_{tu} \mathbf{T}_{it}\mathbf{R}_{tu}^Q\mathbf{T}_{ui}\right)^*$$

This set of strings is followed by the set of substrings that take the machine from S_i into S_j. This latter set of substrings consists of all single symbols T_{ij} that take the machine from S_i to S_j and all other strings that take the machine from S_i to S_j via certain states $S_t, \ldots, S_u$. Clearly, this set can be described by the regular expression

$$\mathbf{T}_{ij} + \sum_{tu} \mathbf{T}_{it}\mathbf{R}_{tu}^{Q}\mathbf{T}_{uj}.$$

Consequently, the set of strings R_{ij}^{P} is regular and can be described by the expression

$$\mathbf{R}_{ij}^{P} = \left(\mathbf{T}_{ii} + \sum_{tu} \mathbf{T}_{it}\mathbf{R}_{tu}^{Q}\mathbf{T}_{ui}\right)^{*} \left(\mathbf{T}_{ij} + \sum_{tu} \mathbf{T}_{it}\mathbf{R}_{tu}^{Q}\mathbf{T}_{uj}\right).$$

◇

Combining Theorems 16.2 and 16.4, we obtain the following general result, which is known as Kleene's theorem.

- A finite-state machine recognizes a set of strings if and only if it is a regular set.

Applications

The correspondence between regular sets and finite-state machines enables us to determine whether certain sets are regular. For example, let $\mathbf{R}$ denote a regular set on an alphabet A that can be recognized by a (Moore) machine M_1. Define the complement of $\mathbf{R}$, denoted $\mathbf{R}'$, as the set containing all the strings on A that are not contained in $\mathbf{R}$. The set $\mathbf{R}'$ is regular, since it can be recognized by a machine M_2 that is obtained from M_1 by complementing the output values associated with the states of M_1.

As another example, let us define the intersection of two sets, $\mathbf{P}$ and $\mathbf{Q}$, denoted $\mathbf{P}\&\mathbf{Q}$, as the set consisting of all the strings that are contained in both $\mathbf{P}$ and $\mathbf{Q}$. We can show that the set $\mathbf{P}\&\mathbf{Q}$ is regular by observing that each of the sets $\mathbf{P}'$ and $\mathbf{Q}'$ is regular and, consequently, $\mathbf{P}' + \mathbf{Q}'$ and $(\mathbf{P}' + \mathbf{Q}')'$ are regular. In addition, since $\mathbf{P}\&\mathbf{Q} = (\mathbf{P}' + \mathbf{Q}')'$, the set $\mathbf{P}\&\mathbf{Q}$ is regular. Regular expressions containing the complementation and intersection operations as well as union, concatenation, and closure are called *extended regular expressions*.

The added operations increase our versatility in describing regular sets. For example, consider the set of strings on the alphabet $\{0, 1\}$ such that no string in the set contains three consecutive 0's. This set can be described by the expression $[(\mathbf{0} + \mathbf{1})^{*}\mathbf{000}(\mathbf{0} + \mathbf{1})^{*}]'$, whereas a more complicated expression, such as $(\mathbf{1} + \mathbf{01} + \mathbf{001})^{*}(\boldsymbol{\lambda} + \mathbf{0} + \mathbf{00})$, would be required if the complementation operation were not used. However, since expressions containing the complementation and intersection operations are difficult to manipulate or transform to the corresponding graphs, their usefulness is limited.

The following example will illustrate some additional techniques that can be used to determine whether certain sets are regular.

Example Let M be a finite-state machine whose input and output alphabets are $\{0, 1\}$. Assume that the machine has a designated starting state. Let $z_1 z_2 \cdots z_n$ denote the output sequence produced by M in response to the input sequence $x_1 x_2 \cdots x_n$. Define a set S_M that consists of all the strings w such that $w = z_1 x_1 z_2 x_2 \cdots z_n x_n$, for any $x_1 x_2 \cdots x_n$ in $(\mathbf{0} + \mathbf{1})^*$. Prove that S_M is regular.

Given the state diagram of M, replace each directed arc with two directed arcs and a new state, as shown in Fig. 16.19. Retain the original starting state and designate all the original states as accepting states. The resulting nondeterministic transition graph recognizes the set S_M. Therefore, S_M must be regular.

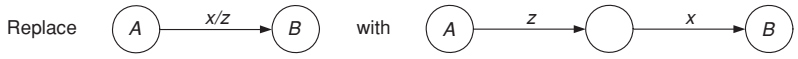


Fig. 16.19 Illustration of the procedure for designing a recognizer for S_M.

This procedure will now be applied to find a deterministic machine that recognizes the set S_N, where N is the machine described in Fig. 16.20. Replacing every arc of the machine N with two directed arcs, and following the procedure just outlined, we arrive at the transition graph in Fig. 16.21a. Converting this graph into deterministic form yields the state diagram of Fig. 16.21b.

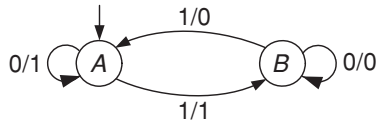
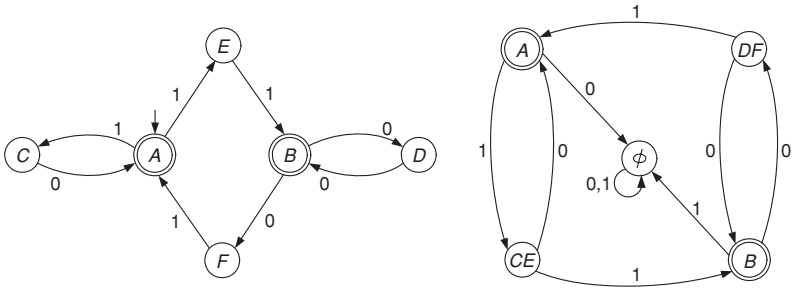


Fig. 16.20 Machine N.



(*a*) Transition graph. (*b*) Equivalent deterministic form.

Fig. 16.21 Constructing a finite-state machine that recognizes S_N.

*16.7 Two-way recognizers

In Section 16.1, we introduced the concept of a recognizer as a finite-state control coupled through a head to a linear input tape. We assumed that the recognizer could move its head in only one direction, to the right. In an attempt to generalize the model further, we will consider recognizers that are not confined to a strict forward motion but can move two ways on their input tapes, that is, to the right and left. A natural question that now arises is whether the option given to the machine to move left and reexamine the input tape increases its computational capabilities. In other words, what characterizes the sets of tapes that are recognized by this class of machines? As we shall see, machines that can move both ways but *cannot* change the tape symbols are no more (nor less) powerful than machines that can move in only one direction.

Description of the model

A *two-way recognizer*, or *two-way machine*, consists of a finite-state control coupled through a head to a tape. Initially, the finite-state control is in its designated starting state, with its head scanning the leftmost square of the tape. The machine then proceeds to read the symbols of the tape one at a time. In each cycle of computation, the machine examines the symbol currently scanned by the head, shifts the head one square to the right or left, and then enters a new (not necessarily distinct) state.

If, when operating in this manner on a given tape, the machine eventually *moves off* the tape at the right-hand end and at that time enters an accepting state, then we shall say that the tape is *accepted* by the machine. A machine can *reject* a tape either by moving off its right-hand end while entering a rejecting state or by looping within the tape. As in the case of one-way machines, the set of tapes that are accepted by a given two-way machine is said to be *recognized* by that machine. The null string λ can be represented either by the absence of an input tape or by a completely blank tape. A machine accepts λ if and only if its starting state is an accepting state.

It is convenient to supply the two-way machine with a new symbol, ¢, called a *left-end marker*, which is entered in the leftmost square of the tape and prevents the head from moving off the left-hand end of the tape. The end marker is not a symbol of the machine's alphabet and must not appear on any other square within the tape.

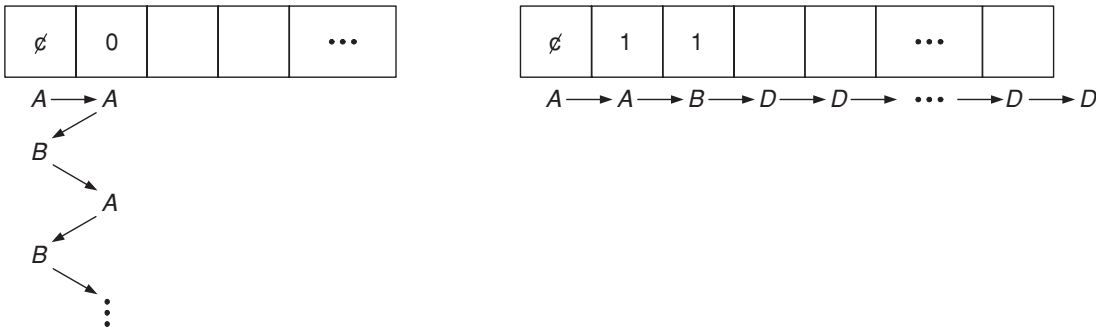
A two-way machine can be described by a state table (or diagram) that specifies, for every possible combination of present state and tape symbol being scanned, the next state that the machine should assume and the direction in which the head is to move. As directional entries, we use the letters L to denote a shift to the left and R to denote a shift to the right.

Example Table 16.1 describes a two-way machine having four states and two tape symbols, 0 and 1, plus the ¢ marker. The starting state is A and the accepting state is C. A blank tape entry indicates that the corresponding state-symbol combination cannot occur. Figure 16.22*a* illustrates the computation that the machine will perform when supplied with a tape that starts with the symbols ¢0. The computation begins with the machine in state A and with its head scanning the left-end marker. According to the state table, the machine will move one square to the right while remaining in state A. The machine will then be scanning a 0 and, consequently, will enter state B and move one square to the left. From now on, the machine will oscillate between these two squares and thus all strings beginning with a 0 will be rejected.

Table 16.1 A two-way-machine recognizing set $\mathbf{100}^*$

	¢	0	1
A	A, R	B, L	B, R
B	A, R	C, R	D, R
C		C, R	D, R
D		D, R	D, R

Next, suppose that the machine is presented with a tape that starts with ¢11. The computation is illustrated in Fig. 16.22*b*. When the third symbol is reached, the machine is in state D. Thereafter, it remains in state D regardless of the tape content until it moves off the tape. Since D is a rejecting state, all sets of tapes starting with 11 are rejected. Finally, let the tape consist of the string ¢10. Again, the machine starts by moving to the right, and it goes through a succession of states until it moves off the tape in state C. Since C is an accepting state, the tape in question is accepted. By similar reasoning, we can verify that the machine recognizes the set $\mathbf{100}^*$.



(*a*) A loop. (*b*) Rejection of a tape.

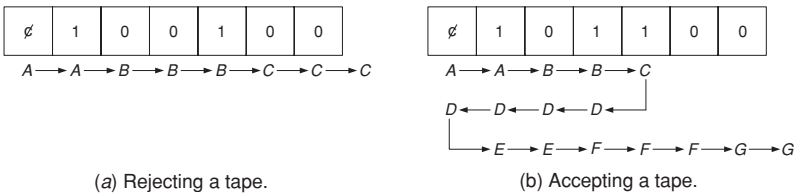
Fig. 16.22 Illustration of computations.

In the next section we shall prove that two-way machines are as powerful as one-way machines with respect to the classes of tapes that they can recognize. For some computations, however, it is convenient to use two-way recognizers since they may require fewer states than the equivalent one-way recognizers. However, for the ability of a two-way machine to reverse direction and reread its tape, we pay in terms of an increased computation time.

Example Consider the two-way machine shown in Table 16.2, which accepts a tape if and only if it contains at least three 1's and at least two 0's. The starting and accepting states are A and G, respectively. Some typical computations are shown in Fig. 16.23. The operation of the machine can be summarized as follows. Initially the machine is in state A and the head is scanning the left-end marker. The head then proceeds to the right to determine whether the tape contains at least three 1's. If the tape contains two or fewer 1's, it is rejected; if it contains three 1's then the head reverses its direction and moves left until it again reaches the left-end marker. The machine then proceeds to the right to determine whether the tape contains two or more 0's. If it does, the machine enters state G and will eventually accept the tape; otherwise the tape will be rejected.

Table 16.2 A two-way machine

	¢	0	1
A	A, R	A, R	B, R
B		B, R	C, R
C		C, R	D, L
D	E, R	D, L	D, L
E		F, R	E, R
F		G, R	F, R
G		G, R	G, R



(*a*) Rejecting a tape. (b) Accepting a tape.

Fig. 16.23 Example of computations.

The minimal one-way machine that is equivalent to the two-way machine in Table 16.2 has 12 states. This larger number of states is necessary because of the way in which a one-way machine operates. Any one-way machine that recognizes the above set of tapes must examine the tapes for the proper

number of 0's and 1's *simultaneously*. This can be done, for example, by the use of two separate counters, one for the 1's and the other for the 0's. The state of the machine in such a case is the composite state of the two counters. Consequently, the number of states required to perform the above computation is proportional to the *product* of the numbers of states required to test the tapes for the number of 0's and the number of 1's separately. The two-way machine in this example tests the tapes first for the appropriate number of 1's and then for the appropriate number of 0's. Thus, the number of states is proportional to the *sum* of the numbers of states required to test the tapes for the two requirements separately.

Conversion to one-way recognizers

We now turn to proving that two-way machines can recognize sets of tapes (or strings) if and only if they are regular sets. Specifically, we shall show that for every given two-way machine there is an equivalent one-way machine that recognizes the same set of tapes. Since the details of the construction procedure do not add significantly to its understanding, we shall confine our discussion to sketching the main ideas of the proof.

Since a one-way machine makes as many moves as there are symbols on the tape while a two-way machine can make moves by reversing direction, the one-way machine cannot keep track of all the moves of the two-way machine or simulate them. It is, therefore, necessary to isolate the significant information gained by a two-way machine on moving to the left from the particular sequence of moves. Consider an initial segment at the left of the input tape, and suppose that the head is scanning the rightmost square of this segment. The only way in which this segment can influence the future behavior of the two-way machine is via the state which the machine is in when (and if) it leaves this segment. Thus, when a two-way machine backs up and reexamines a segment of the tape, the state S_i in which the machine reenters the segment and the corresponding state S_i' which the machine would be in if it left the segment are the only two factors of significance in predicting the future behavior of the machine.

A two-way machine having n states can be in any of these states when it scans the rightmost square of the initial segment. Two cases must be considered. First, the machine may never leave the segment but oscillate within it. Second, the machine will ultimately leave the segment on the right in one of its n states. Thus, a reentry into a segment may have $n + 1$ outcomes, that is, leaving the segment in one of the n states or not leaving it. Consequently the effect of the segment on the computation can be determined by specifying, for each state S_i in which the machine might reenter the segment, which of the $n + 1$ outcomes would indeed result. Such a specification is accomplished by means of a *crossing function* (or *crossing table*), denoted $C(S)$.

Table 16.3 A two-way machine M

	$\not\subset$	0	1
A	A, R	B, R	C, R
B		A, R	A, L
C		B, R	D, L
D		C, L	B, R

Table 16.4 Crossing functions for M

S_i	$C(S_i)$ for $\not\subset 001$	$C(S_i)$ for $\not\subset 0011$
A	C	C
B	0	0
C	C	0
D	B	B

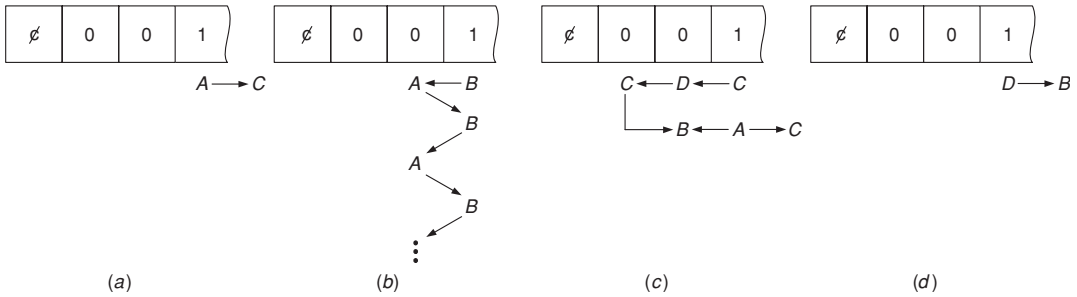


Fig. 16.24 Computations on the segment $\not\subset 001$.

The following is extracted from Shepherdson's proof [11]. It summarizes the informal arguments in support of his proof. (Note that M denotes the given two-way machine and t denotes an initial tape segment.)

If we think of the different states which M could be in when it reentered t as the different questions M could ask about t, and the corresponding states M would be in when it subsequently left t again, as the answers, then we can state the result more crudely and succinctly thus: A machine can spare itself the necessity of coming back to refer to a piece of tape t again, if, before it leaves t, it thinks of all the possible questions it might later come back and ask about t, answers these questions now and carries the table of question–answer combinations forward along the tape with it, altering the answers where necessary as it goes along.

As an example, consider the two-way machine M given in Table 16.3 and the initial tape segment $\not\subset 001$. The starting and accepting states are A and C, respectively. Figure 16.24 illustrates, for each possible initial state, the computation performed by the machine if its head is initially scanning the rightmost symbol of the given segment. If the initial state is A than the machine immediately leaves the segment in state C. If, however, the initial state is B then the machine will oscillate between states B and A and will never leave the segment. From Fig. 16.24 we can derive the crossing function associated with the segment $\not\subset 001$, as shown in the first two columns of Table 16.4. The first column, S_i, of this table lists the states of the machine while the second column, $C(S_i)$, lists the states in which the machine crosses the given segment to the right. An entry 0 indicates that the tape will be rejected.

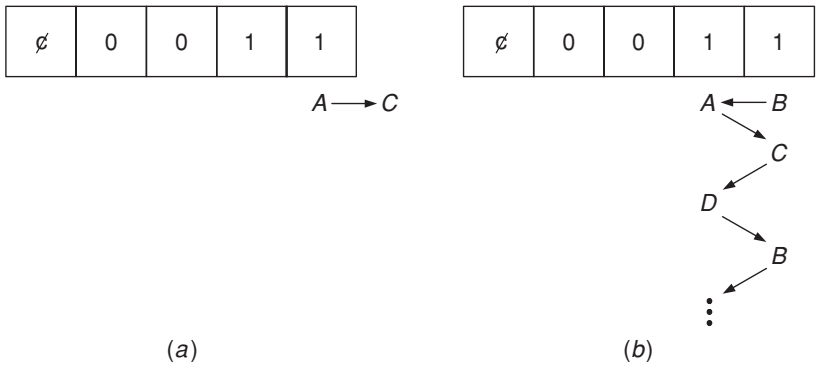


Fig. 16.25 Illustration of computations on the segment ¢0011.

An important property of crossing functions is that the crossing function of a $(k + 1)$-symbol segment can be obtained from the crossing function of a k-symbol segment. The rightmost column of Table 16.4 contains the crossing function associated with the segment ¢0011. This crossing function can be obtained from the crossing function of the segment ¢001. Suppose, for example, that the machine is in state A and is scanning the rightmost symbol of ¢0011. According to the state table in Table 16.3, the machine will move to the right and enter state C, as illustrated in Fig. 16.25*a*. Accordingly, the entry in row A in the rightmost column is C. If, however, the machine is in state B while scanning the rightmost symbol of the given segment then it will move left and enter state A. According to the crossing function associated with the segment ¢001, the machine will leave this segment in state C, as shown in Fig. 16.25*b*. Again it will scan the rightmost symbol of ¢0011 and, according to the state table, again it will move left and enter state D. According to the crossing function for ¢001, the machine will ultimately leave this segment on the right and enter state B. Evidently such a sequence of moves indicates that the computation will never halt and, consequently, a 0 is entered in row B of Table 16.4. The same line of reasoning leads to the specification of the entries in rows C and D.

The procedure followed in this example leads to the conclusion that, *given the crossing functions associated with the initial segments containing k symbols, we can readily obtain the crossing functions associated with all initial segments containing* $k + 1$ *symbols*. In fact, since the number of distinct crossing functions associated with a specific two-way machine cannot exceed $(n + 1)^n$, where n is the number of states, it is possible to construct a one-way machine that will read the tape from left to right and compute with each move the crossing function associated with the corresponding initial segment. Such a machine will have as many states as there are crossing functions. Its input alphabet is the same as that of the corresponding two-way machine. The next-state entries of the one-way machine are obtained as follows. For a given state, which corresponds to a crossing function of the two-way machine, the next-state entry under the input symbol α corresponds to the new crossing function obtained from the given one and the symbol α, as illustrated in Fig. 16.25.

Once we have a one-way machine that scans the tape from left to right and computes the crossing functions associated with successive initial segments, since the starting state of the two-way machine is specified it is a simple matter to determine, after each move of the one-way machine, the corresponding next state of the two-way machine. Consequently, we can determine the state of the two-way machine when it moves off the tape. If this state is an accepting state then the one-way machine will also accept the tape; otherwise it will reject the tape. We thus have the following result.

- The sets of strings recognized by two-way finite-state machines are the same as the sets recognized by one-way finite-state machines. Moreover, there exists an effective procedure for constructing a one-way machine that recognizes the same set of strings as a given two-way machine.

Although two-way machines are no more powerful than one-way machines with respect to the sets of strings that they can recognize, it is often more convenient to describe certain computations in terms of two-way machines. The equivalence of the two models, however, makes it generally possible to use either.

Notes and references

Nondeterministic graphs were first used by Myhill [8] and further developed by numerous investigators, in particular those working on languages. The initial concept of regular expressions and the equivalence between regular expressions and finite-state machines were presented by Kleene [5]. Simpler techniques for converting regular expressions into transition graphs, and vice versa, were subsequently developed by Copi, Elgot, and Wright [4], McNaughton and Yamada [6], and Ott and Feinstein [9]. The procedure presented in this chapter of constructing transition graphs from regular expressions is due to Ott and Feinstein [9], while the procedure used to derive regular expressions that describe transition graphs is due to Arden [1]. A survey of regular expressions is available in Brzozowski [2].

Two-way machines were first investigated by Rabin and Scott [10], who provided the first proof that two-way machines are equivalent to one-way machines. Shepherdson [11] subsequently provided a simpler proof, the one outlined in Section 16.7.

[1] Arden, D. N.: "Delay logic and finite state machines," in *Proc. Second Ann. Symp. Switching Theory and Logical Design*, pp. 133–151, October 1961.

[2] Brzozowski, J. A.: "A survey of regular expressions and their applications," *IRE Trans. Electron. Computers*, vol. EC-11, pp. 324–335, June 1962.

[3] Brzozowski, J. A.: "Derivatives of regular expressions," *J. Assoc. Computing Machinery*, vol. 11, pp. 481–494, 1964.

[4] Copi, I. M., C. C. Elgot, and J. B. Wright: "Realization of events by logical nets," *J. Assoc. Computing Machinery*, vol. 5, pp. 181–196, April 1958; reprinted in Moore [7].

[5] Kleene, S. C.: *Representation of Events in Nerve Nets and Finite Automata*, pp. 3–41, Automata Studies, Princeton University Press, 1956.
[6] McNaughton, R., and H. Yamada: "Regular expressions and state graphs for automata," *IRE Trans. Electron. Computers*, vol. EC-9, pp. 39–47, March 1960; reprinted in Moore [7].
[7] Moore, E. F. (ed.): *Sequential Machines: Selected Papers*, Addison-Wesley, Reading MA, 1964.
[8] Myhill, J.: "Finite automata and the representation of events," WADC Technical Report 57–624, pp. 112–137, 1957.
[9] Ott, G. H., and N. H. Feinstein: "Design of sequential machines from their regular expressions," *J. Assoc. Computing Machinery*, vol. 8, pp. 585–600, October 1961.
[10] Rabin, M. O., and D. Scott: "Finite automata and their decision problems," *IBM J. Res. Develop.*, vol. 3, no. 2, pp. 114–125, April 1959; reprinted in Moore [7].
[11] Shepherdson, J. C.: "The reduction of two-way automata to one-way automata," *IBM J. Res. Develop.*, vol. 3, no. 2, pp. 198–200, April 1959; reprinted in Moore [7].

Problems

Problem 16.1. For each of the sets described as follows, find a transition graph that recognizes the set.
(a) The set of strings on the alphabet $\{0, 1\}$ that start with 01 and end with 10.
(b) The set of strings on the alphabet $\{0, 1\}$ that start and end with a 1, and in which every 0 is immediately preceded by at least two 1's.
(c) The set of strings on the alphabet $\{0, 1, 2\}$ in which every 2 is immediately followed by exactly two 0's and every 1 is immediately followed by either 0 or else by 20.

Problem 16.2. Consider the class of transition graphs containing no λ-transitions.
(a) Show a procedure for converting a specified transition graph with several starting vertices into a graph with just one starting vertex. Apply your procedure to the graph in Fig. P16.2.
Hint: Add a new vertex and designate it as the starting vertex.
(b) Show a procedure for converting a given transition graph with several accepting vertices into a graph with just one accepting vertex. Apply your procedure to the graph in Fig. P16.2.
(c) Is it always possible to convert an arbitrary transition graph into a graph with just one starting vertex and just one accepting vertex? Determine the conditions under which such a conversion is possible.

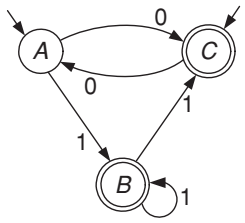
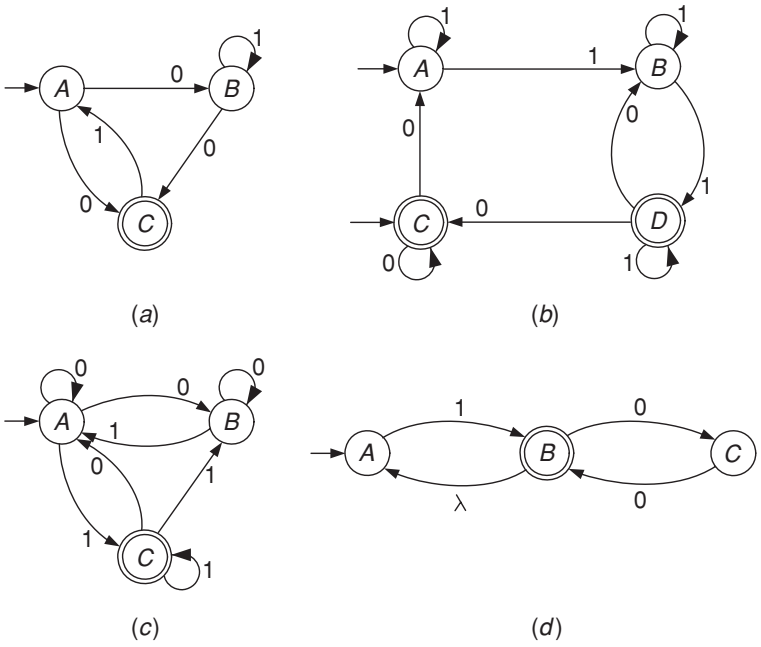


Fig. P16.2

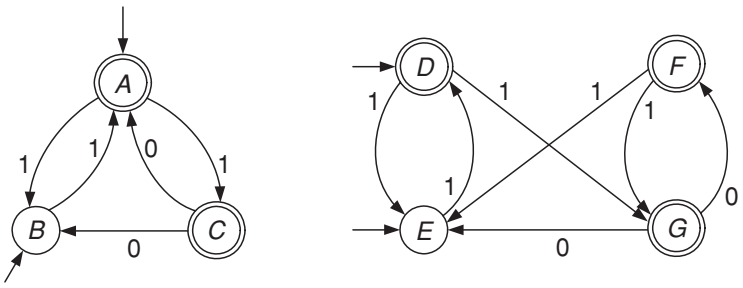
Problem 16.3. For each of the nondeterministic graphs in Fig. P16.3, find an equivalent deterministic graph (in standard form) that recognizes the same set of strings.

Fig. P16.3



Problem 16.4. Show that the two graphs in Fig. P16.4 are equivalent by converting them to deterministic forms.

Fig. P16.4



Problem 16.5. Design a finite-state machine that accepts only those input sequences that end with either 101 or 0110. First construct a nondeterministic graph that recognizes the above set of sequences and then convert this graph into an equivalent deterministic graph. Discuss the merits of this approach versus the direct approach of deriving a state diagram from a word description.

Problem 16.6. Give a word description of the sets described by the following regular expressions:

(a) $\mathbf{110}^*(\mathbf{0}+\mathbf{1})$;

(b) $\mathbf{1}(\mathbf{0}+\mathbf{1})^*\mathbf{101}$;

(c) $(\mathbf{10})^*(\mathbf{01})^*(\mathbf{00}+\mathbf{11})^*$;

(d) $(\mathbf{00}+(\mathbf{11})^*\mathbf{0})^*\mathbf{10}$.

Problem 16.7. Find a regular expression for each set described in Problem 16.1.

Problem 16.8. Use the identities in Section 16.4 to verify the identities below:
(a) $\mathbf{10} + (\mathbf{1010})^*[\boldsymbol{\lambda}^* + \boldsymbol{\lambda}(\mathbf{1010})^*] = \mathbf{10} + (\mathbf{1010})^*$;
(b) $(\mathbf{0}^*\mathbf{01} + \mathbf{10})^*\mathbf{0}^* = (\mathbf{0} + \mathbf{01} + \mathbf{10})^*$;
(c) $\boldsymbol{\lambda} + \mathbf{0}(\mathbf{0} + \mathbf{1})^* + (\mathbf{0} + \mathbf{1})^*\mathbf{00}(\mathbf{0} + \mathbf{1})^* = [(\mathbf{1}^*\mathbf{0})^*\mathbf{01}^*]^*$.

Problem 16.9.
(a) Use the induction procedure developed in Section 16.5 to find a transition graph that recognizes the set of strings described by

$$\mathbf{R} = \mathbf{0}(\mathbf{11} + \mathbf{0}(\mathbf{00} + \mathbf{1})^*)^*.$$

(b) Convert the graph found in (a) to a deterministic state diagram.

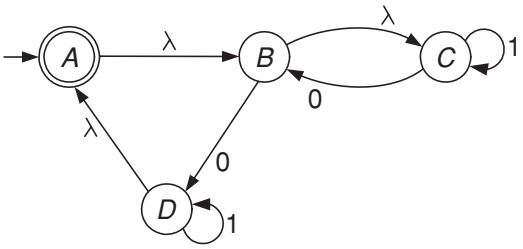
Problem 16.10. For each of the following expressions, find a transition graph that recognizes the corresponding set of strings:
(a) $(\mathbf{0} + \mathbf{1})(\mathbf{11} + \mathbf{0}^*)^*(\mathbf{0} + \mathbf{1})$;
(b) $(\mathbf{1010}^* + \mathbf{1}(\mathbf{101})^*\mathbf{0})^*\mathbf{1}$;
(c) $(\mathbf{0} + \mathbf{11})^*(\mathbf{1} + (\mathbf{00})^*)^*\mathbf{11}$.

Problem 16.11. The regular expression that corresponds to the transition graph in Fig. P16.11 is

$$\mathbf{R} = [(\mathbf{1}^*\mathbf{0})^*\mathbf{01}^*]^*.$$

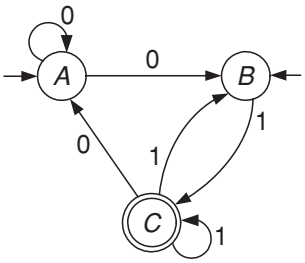
Find a finite-state machine that recognizes the same set of strings.

Fig. P16.11



Problem 16.12. The nondeterministic graph in Fig. P16.12 has A and B as starting vertices and C as an accepting vertex.
(a) Find a regular expression that describes the set of strings accepted by this graph.
(b) Derive a reduced deterministic machine equivalent to this graph.

Fig. P16.12



Problem 16.13. For each machine in Table P16.13, find a regular expression that describes the set of input strings recognized by the machine. In each case the starting state is A.

Table P16.13

	NS		
PS	$x=0$	$x=1$	z
A	A	B	0
B	B	A	1

(*a*)

	NS		
PS	$x=0$	$x=1$	z
A	B	A	1
B	B	C	0
C	A	B	1

(*b*)

	NS, z	
PS	$x=0$	$x=1$
A	$B,0$	$A,1$
B	$A,1$	$C,1$
C	$C,0$	$B,0$

(*c*)

Problem 16.14. Find a regular expression on the alphabet $\{0, 1, 2\}$ for the set of strings recognized by the graph of Fig. P16.14.

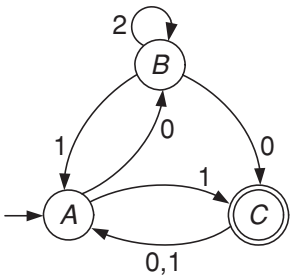


Fig. P16.14

Problem 16.15. Determine whether each of the following sets on the alphabet $\{0, 1\}$ is regular and justify your answer:

(a) the set consisting of those strings that contain, for all k, k 1's and $k+1$ 0's;

(b) the set of strings in which every 0 is immediately preceded by *at least* k 1's and is immediately followed by *exactly* k 1's, where k is a specified integer;

(c) the set of strings that contain more 1's than 0's;

(d) the set of strings consisting of a block of k^2 0's immediately followed by a single 1, where $k = 0, 1, 2, \ldots$

Problem 16.16

(a) Let M be a deterministic Mealy-type finite-state machine with a starting state A. Prove that if T is the set of strings that can be produced as output strings by M then T is a regular set. Find a procedure to design a finite-state machine that will recognize T.

Hint: Use the output successor table of M.

(b) Apply your procedure to find a finite-state machine that will recognize the set of output strings that can be produced by the machine defined by Table P16.16.

Table P16.16

	NS, z	
PS	$x = 0$	$x = 1$
A	$B, 1$	$A, 1$
B	$A, 0$	$C, 0$
C	$D, 1$	$B, 0$
D	$C, 0$	$A, 1$

Problem 16.17. The reverse $\mathbf{R}^{\mathrm{r}}$ of a set $\mathbf{R}$ is the set that consists of the reverses of the strings in $\mathbf{R}$. Thus, for example, if 0101 is in $\mathbf{R}$ then 1010 is in $\mathbf{R}^{\mathrm{r}}$.

(a) Prove that if $\mathbf{R}$ is regular then so is $\mathbf{R}^{\mathrm{r}}$.
Hint: Develop a systematic procedure to convert a given regular expression into its reverse.

(b) Apply the above procedure to find the reverse of the expression

$$\mathbf{R} = (\mathbf{00})^*(\mathbf{0} + \mathbf{10}^*)^* + \mathbf{10}^*(\mathbf{01}^*\mathbf{10}^*)^*.$$

Problem 16.18. Either prove each of the following statements or show a counter example.

(a) Every *finite* subset of a nonregular set is regular.
(b) The expressions $\mathbf{P} = (\mathbf{1}^*\mathbf{0} + \mathbf{001})^*\mathbf{01}$ and $\mathbf{Q} = (\mathbf{1}^*\mathbf{001} + \mathbf{00101})^*$ are equivalent.
(c) Let $\mathbf{R}$ denote a regular set. Then the set consisting of all the strings in $\mathbf{R}$ that are identical to their own reverses is also a regular set.
(d) Every subset of a regular set is also regular.

Problem 16.19. Consider the nondeterministic machine M^{n}, which is obtained from a strongly connected deterministic machine M by interchange of the sets of starting and accepting states and reversal of the arrows on the state diagram.

(a) If the machine M recognizes the set $\mathbf{R}$, what is the set recognized by M^{n}?
(b) Prove that the deterministic machine obtained by applying "subset construction" to M^{n} has no equivalent states.

Problem 16.20. Let $\mathbf{P}$ be a regular set consisting of strings of even length. Define a set $\mathbf{Q}$ that consists of exactly those strings that can be formed by taking the first half of each member of $\mathbf{P}$. (For example, if 10110100 is contained in $\mathbf{P}$ then 1011 will be contained in $\mathbf{Q}$.) Prove that $\mathbf{Q}$ is a regular set.

Hint: Design a machine that recognizes $\mathbf{Q}$.

Problem 16.21. Let $\mathbf{P}$ be a regular set, and let $\mathbf{Q}$ be the set formed of all the strings from $\mathbf{P}$ with even-numbered symbols deleted; that is, if $a_1a_2a_3a_4a_5\cdots$ is a string in $\mathbf{P}$, then $a_1a_3a_5\cdots$ is a string in $\mathbf{Q}$. Prove that $\mathbf{Q}$ is a regular set.

Problem 16.22. Let $\mathbf{P}$ be an arbitrary regular set. Consider those strings w in $\mathbf{P}$ such that both w and ww are in $\mathbf{P}$. Define $\mathbf{Q}$ to be the set consisting of all the above w's. Thus, for example, if 101 and 101101 are in $\mathbf{P}$ then 101 is in $\mathbf{Q}$. Prove that $\mathbf{Q}$ is a regular set.

Problem 16.23. Let $\mathbf{R}$ be a regular set on the alphabet $\{0, 1\}$. The *derivative of* $\mathbf{R}$ *with respect to* x, denoted $\mathbf{R}_x$, is defined as the set consisting of all substrings y such that xy is in $\mathbf{R}$. For example, if $\mathbf{R} = \mathbf{01}^* + \mathbf{100}^*$ then $\mathbf{R}_0 = \mathbf{1}^*$ and $\mathbf{R}_{10} = \mathbf{0}^*$.

(a) Prove that, for all x, $\mathbf{R}_x$ is a regular set.

(b) Show that there is only a finite number of distinct derivatives for any regular set (although there is an infinite number of choices for x). Find an upper bound on this number if it is known that $\mathbf{R}$ can be recognized by a transition graph with k vertices.

Problem 16.24. The *right quotient* of two sets X and Y, denoted X/Y, is defined as the set Z that consists of all strings z such that $x = zy$ is a string in X and y is a string in Y. Prove that if X is a regular set then $Z = X/Y$ is also a regular set. The set Y may or may not be regular.

Problem 16.25. Determine which of the following tapes is accepted by the two-way machine shown in Table P16.25. The starting and accepting states are A and D, respectively.

(a) ¢010101

(b) ¢010110

(c) ¢10101

Table P16.25

	¢	0	1
A	A, R	B, R	C, R
B		D, L	C, L
C		C, R	D, R
D		B, R	C, L

Problem 16.26. A two-way machine with n states is started at the left end of a tape containing p squares. What is the maximum number of moves that the machine can make before accepting the tape?

Problem 16.27. Construct a two-way machine whose tape may contain symbols from the alphabet $\{0, 1, 2\}$ plus the left-end marker and which accepts a string if and only if it starts and ends with a 2 and every 2 except the first is immediately preceded by a substring from the set $\mathbf{0}(\mathbf{01})^*$.

Problem 16.28. A given two-way machine recognizes a set of tapes A, rejects a set B, and does not accept (by never halting) a set C. Can a two-way machine be designed so that it:

(a) recognizes B, rejects A, does not accept C?

(b) recognizes A and rejects B and C?

(c) recognizes A but does not accept B and C?

(d) recognizes A and C and rejects B?

(e) recognizes C, rejects B, and does not accept A?

Hint: Determine first which of the sets A, B, and C is regular.