e. How many pairwise nonisomorphic two-state FAs over {0, 1} are there, in which both states are reachable from the initial state and at least one state is accepting?
f. How many distinct languages are accepted by the FAs in the previous part?
g. Show that the FAs described by these two transition tables are isomorphic. The states are 1–6 in the first, A–F in the second; the initial states are 1 and A, respectively; the accepting states in the first FA are 5 and 6, and D and E in the second.

q	$\delta_1(q,0)$	$\delta_1(q,1)$
1	3	5
2	4	2
3	1	6
4	4	3
5	2	4
6	3	4

q	$\delta_2(q,0)$	$\delta_2(q,1)$
A	B	E
B	A	D
C	C	B
D	B	C
E	F	C
F	C	F

h. Specify a reasonable algorithm for determining whether or not two given FAs are isomorphic.

CHAPTER 4

Nondeterminism and Kleene's Theorem

4.1 | NONDETERMINISTIC FINITE AUTOMATA

One of our goals in this chapter is to prove that a language is regular if and only if it can be accepted by a finite automaton (Theorem 3.1). However, examples like Example 3.16 suggest that finding a finite automaton (FA) corresponding to a given regular expression can be tedious and unintuitive if we rely only on the techniques we have developed so far, which involve deciding at each step how much information about the input string it is necessary to remember. An alternative approach is to consider a formal device called a nondeterministic finite automaton (NFA), similar to an FA but with the rules relaxed somewhat. Constructing one of these devices to correspond to a given regular expression is often much simpler. Furthermore, it will turn out that NFAs accept exactly the same languages as FAs, and that there is a straightforward procedure for converting an NFA to an equivalent FA. As a result, not only will it be easier in many examples to construct an FA by starting with an NFA, but introducing these more general devices will also help when we get to our proof.

A Simpler Approach to Accepting {11, 110}*{0} EXAMPLE 4.1

Figure 4.1*a* shows the FA constructed in Example 3.16, which recognizes the language L corresponding to the regular expression $(11 + 110)^*0$. Now look at the diagram in Figure 4.1*b*. If we concentrate on the resemblance between this device and an FA, and ignore for the moment the ways in which it fails to satisfy the normal FA rules, we can see that it reflects much more clearly the structure of the regular expression.

For any string x in L, a path through the diagram corresponding to x can be described as follows. It starts at q_0, and for each occurrence of one of the strings 11 or 110 in the portion of x corresponding to $(11 + 110)^*$, it takes the appropriate loop that returns to q_0; when the final 0 is encountered, the path moves to the accepting state q_4. In the other direction, we can also

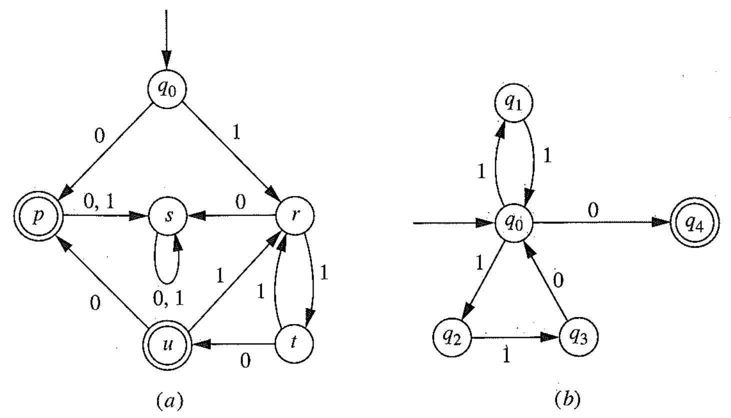

Figure 4.1 |
A simpler approach to accepting $\{11, 110\}^*\{0\}$.

see that any path starting at q_0 and ending at q_4 corresponds to a string matching the regular expression.

Let us now consider the ways in which the diagram in Figure 4.1*b* differs from that of an FA, and the way in which it should be interpreted if we are to view it as a language-accepting device. There are two apparently different problems. From some states there are not transitions for both input symbols (from state q_4 there are no transitions at all); and from one state, q_0, there is more than one arrow corresponding to the same input.

The way to interpret the first of these features is easy. The absence of an a-transition from state q means that from q there is no input string beginning with a that can result in the device's being in an accepting state. We could create a transition by introducing a "dead" state to which the device could go from q on input a, having the property that once the device gets to that state it can never leave it. However, leaving this transition out makes the picture simpler, and because it would never be executed during any sequence of moves leading to an accepting state, leaving it out does not hurt anything except that it violates the rules.

The second violation of the rules for FAs seems to be more serious. The diagram indicates two transitions from q_0 on input 1. It does not specify an unambiguous action for that state-input combination, and therefore apparently no longer represents a recognition algorithm or a language-recognizing machine.

As we will see shortly, there is an FA that operates in such a way that it simulates the diagram correctly and accepts the right language. However, even as it stands, we can salvage much of the intuitive idea of a machine from the diagram, if we are willing to give the machine an element of *nondeterminism*. An ordinary finite automaton is deterministic: The moves it makes while processing an input string are completely determined by the input symbols and the state it starts in. To the extent that a device is nondeterministic, its behavior is unpredictable. There may be situations (i.e., state-input combinations) in which it has a choice of possible moves, and in these cases it selects a move in some unspecified way. One way to describe this is to say that it *guesses* a move.

For an ordinary (deterministic) FA M, saying what it means for a string to be accepted by M is easy: x is accepted if the sequence of moves determined by the input symbols of x causes M to end up in an accepting state. If M is allowed to guess, we must think more carefully about what acceptance means. (Should we say that x is accepted if *some* choice of moves corresponding to the string x leads to an accepting state, or should we require that *every* choice does?) Returning to our example, it is helpful to think about the regular expression $(11 + 110)^*0$ corresponding to the language we are trying to accept, and the problem of trying to decide directly whether a string x (for example, 11110110) matches this regular expression. As we read the input symbols of x, there are several ways we might try to match them to the regular expression. Some approaches will not work, such as matching the prefix 1111 with $(11 + 110)^*$. If there is at least one way that does work, we conclude that x corresponds to the regular expression. (If we make a wrong guess during this matching process, we might fail to accept a string that is actually in the language; however, wrong guesses will never cause us to accept a string that is not in the language.)

This analogy suggests the appropriate way of making sense of Figure 4.1*b*. Instead of asking whether *the* path corresponding to a string leads to an accepting state (as with an ordinary FA), we ask whether *some* path corresponding to the string does. In following the diagram, when we are in state q_0 and we receive input symbol 1 we guess which arrow labeled 1 is the appropriate one to follow. Guessing wrong might result in a "no" answer for a string that is actually in the language. This does not invalidate the approach, because for a string in the language there will be at least one sequence of guesses that leads to acceptance, and for a string not in the language no sequence of guesses will cause the string to be accepted.

We are almost ready for a formal definition of the abstract device illustrated by Figure 4.1*b*. The only change we will need to make to the definition of an FA involves the transition function. As we saw in Example 4.1, for a particular combination of state and input symbol, there may be no states specified to which the device should go, or there may be several. All we have to do in order to accommodate both these cases is to let the value of the transition function be a *set* of states—possibly the empty set, possibly one with several elements. In other words, our transition function δ will still be defined on $Q \times \Sigma$ but will now have values in 2^Q (the set of subsets of Q). Our interpretation will be that $\delta(q, a)$ represents the set of states that the device can legally be in, as a result of being in state q at the previous step and then processing input symbol a.

Definition 4.1 A Nondeterministic Finite Automaton

A nondeterministic finite automaton, abbreviated NFA, is a 5-tuple $M = (Q, \Sigma, q_0, A, \delta)$, where Q and Σ are nonempty finite sets, $q_0 \in Q$, $A \subseteq Q$, and

$$\delta : Q \times \Sigma \to 2^Q$$

Q is the set of states, Σ is the alphabet, q_0 is the initial state, and A is the set of accepting states.

Just as in the case of FAs, it is useful to extend the transition function δ from $Q \times \Sigma$ to the larger set $Q \times \Sigma^*$. For an NFA M, δ^* should be defined so that $\delta^*(p, x)$ is the set of states M can legally be in as a result of starting in state p and processing the symbols in the string x. The recursive formula for FAs was

$$\delta^*(p, xa) = \delta(\delta^*(p, x), a)$$

where p is any state, x is any string in Σ^*, and a is any single alphabet symbol. We will give a recursive definition here as well, but just as before, let us start gradually, with what may still seem like a more straightforward approach. As before, we want to say that the only state M can get to from p as a result of processing Λ is p; the only difference now is that we must write $\delta^*(p, \Lambda) = \{p\}$, rather than $\delta^*(p, \Lambda) = p$.

If $x = a_1a_2 \cdots a_n$, then saying M can be in state q after processing x means that there are states $p_0, p_1, p_2, \ldots, p_n$ so that $p_0 = p$, $p_n = q$, and

M can get from p_0 (or p) to p_1 by processing a_1;
M can get from p_1 to p_2 by processing a_2;
...
M can get from p_{n-2} to p_{n-1} by processing a_{n-1};
M can get from p_{n-1} to p_n (or q) by processing a_n.

A simpler way to say "M can get from p_{i-1} to p_i by processing a_i" (or "p_i is one of the states to which M can get from p_{i-1} by processing a_i") is

$$p_i \in \delta(p_{i-1}, a_i)$$

We may therefore define the function δ^* as follows.

Definition 4.2a Nonrecursive Definition of δ^* for an NFA

For an NFA $M = (Q, \Sigma, q_0, A, \delta)$, and any $p \in Q$, $\delta^*(p, \Lambda) = \{p\}$. For any $p \in Q$ and any $x = a_1a_2 \cdots a_n \in \Sigma^*$ (with $n \geq 1$), $\delta^*(p, x)$ is the set of all states q for which there is a sequence of states $p = p_0, p_1, \ldots, p_{n-1}, p_n = q$ satisfying

$$p_i \in \delta(p_{i-1}, a_i) \text{ for each } i \text{ with } 1 \leq i \leq n$$

Although the sequence of statements above was intended specifically to say that $p_n \in \delta^*(p_0, a_1a_2 \cdots a_n)$, there is nothing to stop us from looking at intermediate points along the way and observing that for each $i \geq 1$,

$$p_i \in \delta^*(p_0, a_1a_2 \cdots a_i)$$

In order to obtain a recursive description of δ^*, it is helpful in particular to consider $i = n - 1$ and to let y denote the string $a_1a_2 \cdots a_{n-1}$. Then we have

$$p_{n-1} \in \delta^*(p, y)$$
$$p_n \in \delta(p_{n-1}, a_n)$$

In other words, if $q \in \delta^*(p, ya_n)$, then there is a state r (namely, $r = p_{n-1}$) in the set $\delta^*(p, y)$ so that $q \in \delta(r, a_n)$. It is also clear that this argument can be reversed: If $q \in \delta(r, a_n)$ for some $r \in \delta^*(p, y)$, then we may conclude from the definition that $q \in \delta^*(p, ya_n)$. The conclusion is the recursive formula we need:

$$\delta^*(p, ya_n) = \{q \mid q \in \delta(r, a_n) \text{ for some } r \in \delta^*(p, y)\}$$

or more concisely,

$$\delta^*(p, ya_n) = \bigcup_{r \in \delta^*(p,y)} \delta(r, a_n)$$

Definition 4.2b Recursive Definition of δ^* for an NFA

Let $M = (Q, \Sigma, q_0, A, \delta)$ be an NFA. The function $\delta^* : Q \times \Sigma^* \to 2^Q$ is defined as follows.

1. For any $q \in Q$, $\delta^*(q, \Lambda) = \{q\}$.
2. For any $q \in Q$, $y \in \Sigma^*$, and $a \in \Sigma$,

$$\delta^*(q, ya) = \bigcup_{r \in \delta^*(q,y)} \delta(r, a)$$

We want the statement that a string x is accepted by an NFA M to mean that there is a sequence of moves M can make, starting in its initial state and processing the symbols of x, that will lead to an accepting state. In other words, M accepts x if the set of states in which M can end up as a result of processing x contains at least one accepting state.

Definition 4.3 Acceptance by an NFA

Let $M = (Q, \Sigma, q_0, A, \delta)$ be an NFA. The string $x \in \Sigma^*$ is accepted by M if $\delta^*(q_0, x) \cap A \neq \emptyset$. The language recognized, or accepted, by M is the set $L(M)$ of all strings accepted by M. For any language $L \subseteq \Sigma^*$, L is recognized by M if $L = L(M)$.

Using the Recursive Definition of δ^* in an NFA — EXAMPLE 4.2

Let $M = (Q, \Sigma, q_0, A, \delta)$, where $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{0, 1\}$, $A = \{q_3\}$, and δ is given by the following table.

q	$\delta(q, 0)$	$\delta(q, 1)$
q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\{q_2\}$	$\{q_2\}$
q_2	$\{q_3\}$	$\{q_3\}$
q_3	$\emptyset$	$\emptyset$

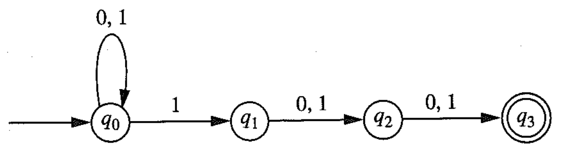

Figure 4.2 |

Then M can be represented by the transition diagram in Figure 4.2. Let us try to determine $L(M)$ by calculating $\delta^*(q_0, x)$ for a few strings x of increasing length. First observe that from the nonrecursive definition of δ^* it is almost obvious that δ and δ^* agree for strings of length 1 (see also Exercise 4.3). We see from the table that $\delta^*(q_0, 0) = \{q_0\}$ and $\delta^*(q_0, 1) = \{q_0, q_1\}$;

$$\begin{aligned}
\delta^*(q_0, 11) &= \bigcup_{r\in\delta^*(q_0,1)} \delta(r, 1) \quad \text{(by definition of } \delta^*) \\
&= \bigcup_{r\in\{q_0,q_1\}} \delta(r, 1) \\
&= \delta(q_0, 1) \cup \delta(q_1, 1) \\
&= \{q_0, q_1\} \cup \{q_2\} \\
&= \{q_0, q_1, q_2\} \\
\delta^*(q_0, 01) &= \bigcup_{r\in\delta^*(q_0,0)} \delta(r, 1) \\
&= \bigcup_{r\in\{q_0\}} \delta(r, 1) \\
&= \delta(q_0, 1) \\
&= \{q_0, q_1\} \\
\delta^*(q_0, 111) &= \bigcup_{r\in\delta^*(q_0,11)} \delta(r, 1) \\
&= \delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1) \\
&= \{q_0, q_1, q_2, q_3\} \\
\delta^*(q_0, 011) &= \bigcup_{r\in\delta^*(q_0,01)} \delta(r, 1) \\
&= \delta(q_0, 1) \cup \delta(q_1, 1) \\
&= \{q_0, q_1, q_2\}
\end{aligned}$$

We observe that 111 is accepted by M and 011 is not. You can see if you study the diagram in Figure 4.2 that $\delta^*(q_0, x)$ contains q_1 if and only if x ends with 1; that for any y with $|y| = 2$, $\delta^*(q_0, xy)$ contains q_3 if and only if x ends with 1; and, therefore, that the language recognized by M is

$$\{0, 1\}^*\{1\}\{0, 1\}^2$$

This is the language we called L_3 in Example 3.17, the set of strings with length at least 3 having a 1 in the third position from the end. By taking the diagram in Figure 4.2 as a model, you can easily construct for any $n \geq 1$ an NFA with $n + 1$ states that recognizes L_n. Since we showed in Example 3.17 that any ordinary FA accepting L_n needs at least 2^n states, it is now clear that an NFA recognizing a language may have considerably fewer states than the simplest FA recognizing the language.

The formulas above may help to convince you that the recursive definition of δ^* is a workable tool for investigating the NFA, and in particular for testing a string for membership in the language $L(M)$. Another approach that provides an effective way of visualizing the behavior of the NFA as it processes a string is to draw a *computation tree* for that string. This is just a tree diagram tracing the choices the machine has at each step, and the paths through the states corresponding to the possible sequences of moves. Figure 4.3 shows the tree for the NFA above and the input string 101101.

Starting in q_0, M can go to either q_0 or q_1 using the first symbol of the string. In either case, there is only one choice at the next step, with the symbol 0. This tree shows several types of paths: those that end prematurely (with fewer than six moves), because M arrives in a state from which there is no move corresponding to the next input symbol; those that contain six

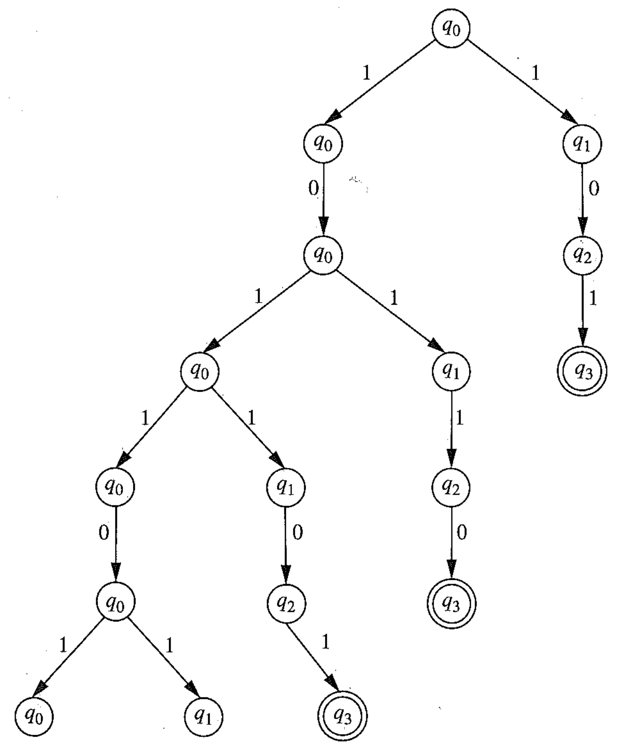

Figure 4.3 |
A computation tree for the NFA in Figure 4.2, as it processes 101101.

moves and end in a nonaccepting state; and one path of length 6 that ends in the accepting state and therefore shows the string to be accepted.

It is easy to read off from the diagram the sets $\delta^*(q_0, y)$ for prefixes y of x. For example, $\delta^*(q_0, 101) = \{q_0, q_1, q_3\}$, because these three states are the ones that appear at that level of the tree. Deciding whether x is accepted is simply a matter of checking whether any accepting states appear in the tree at level $|x|$ (assuming that the root of the tree, the initial state, is at level 0).

We now want to show that although it may be easier to construct an NFA accepting a given language than to construct an FA, nondeterministic finite automata as a group are no more powerful than FAs: Any language that can be accepted by an NFA can also be recognized by a (possibly more complicated) FA.

We have discussed the fact that a (deterministic) finite automaton can be interpreted as an algorithm for recognizing a language. Although an NFA might not represent an algorithm directly, there are certainly algorithms that can determine for any string x whether or not there is a sequence of moves that corresponds to the symbols of x and leads to an accepting state. Looking at the tree diagram in Figure 4.3, for example, suggests some sort of *tree traversal* algorithm, of which there are several standard ones. A *depth-first* traversal corresponds to checking the possible paths sequentially, following each path until it stops, and then seeing if it stops in an accepting state after the correct number of steps. A *breadth-first*, or level-by-level, traversal of the tree corresponds in some sense to testing the paths in parallel.

The question we are interested in, however, is not whether there is an algorithm for recognizing the language, but whether there is one that corresponds to a (deterministic) finite automaton. We will show that there is by looking carefully at the definition of an NFA, which contains a potential mechanism for eliminating the nondeterminism directly.

The definition of a finite automaton, either deterministic or nondeterministic, involves the idea of a *state*. The nondeterminism present in an NFA appears whenever there are state-input combinations for which there is not exactly one resulting state. In a sense, however, the nondeterminism in an NFA is only apparent, and arises from the notion of state that we start with. We will be able to transform the NFA into an FA by redefining *state* so that for each combination of state and input symbol, exactly one state results. The way to do this is already suggested by the definition of the transition function of an NFA. Corresponding to a particular state-input pair, we needed the function to have a single value, and the way we accomplished this was to make the value a set. All we have to do now is carry this idea a little further and consider a state in our FA to be a subset of Q, rather than a single element of Q. (There is a partial precedent for this in the proof of Theorem 3.4, where we considered states that were pairs of elements of Q.) Then corresponding to the "state" S and the input symbol a (i.e., to the set of all the pairs (p, a) for which $p \in S$), there is exactly one "state" that results: the union of all the sets $\delta(p, a)$ for $p \in S$. All of a sudden, the nondeterminism has disappeared! Furthermore, the resulting machine simulates the original device in an obvious way, provided that we define the initial and final states correctly.

This technique is important enough to have acquired a name, the *subset construction*: States in the FA are subsets of the state set of the NFA.

Theorem 4.1

For any NFA $M = (Q, \Sigma, q_0, A, \delta)$ accepting a language $L \subseteq \Sigma^*$, there is an FA $M_1 = (Q_1, \Sigma, q_1, A_1, \delta_1)$ that also accepts L.

Proof

M_1 is defined as follows:

$$Q_1 = 2^Q \qquad q_1 = \{q_0\} \qquad \text{for } q \in Q_1 \text{ and } a \in \Sigma,\ \delta_1(q, a) = \bigcup_{r \in q} \delta(r, a)$$

$$A_1 = \{q \in Q_1 \mid q \cap A \neq \emptyset\}$$

The last definition is the right one because a string x should be accepted by M_1 if, starting in q_0, the set of states in which M might end up as a result of processing x contains at least one element of A.

The fact that M_1 accepts the same language as M follows from the fact that for any $x \in \Sigma^*$,

$$\delta_1^*(q_1, x) = \delta^*(q_0, x)$$

which we now prove using structural induction on x. Note that the functions δ_1^* and δ^* are defined in different ways: δ^* in Definition 4.2b, since M is an NFA, and δ_1^* in Definition 3.3, since M_1 is an FA.

If $x = \Lambda$,

$$\begin{aligned}
\delta_1^*(q_1, x) &= \delta_1^*(q_1, \Lambda) \\
&= q_1 \quad \text{(by definition of } \delta_1^*) \\
&= \{q_0\} \quad \text{(by definition of } q_1) \\
&= \delta^*(q_0, \Lambda) \quad \text{(by definition of } \delta^*) \\
&= \delta^*(q_0, x)
\end{aligned}$$

The induction hypothesis is that x is a string satisfying $\delta_1^*(q_1, x) = \delta^*(q_0, x)$, and we wish to prove that for any $a \in \Sigma$, $\delta_1^*(q_1, xa) = \delta^*(q_0, xa)$:

$$\begin{aligned}
\delta_1^*(q_1, xa) &= \delta_1(\delta_1^*(q_1, x), a) \quad \text{(by definition of } \delta_1^*) \\
&= \delta_1(\delta^*(q_0, x), a) \quad \text{(by the induction hypothesis)} \\
&= \bigcup_{r \in \delta^*(q_0, x)} \delta(r, a) \quad \text{(by definition of } \delta_1) \\
&= \delta^*(q_0, xa) \quad \text{(by definition of } \delta^*)
\end{aligned}$$

That M and M_1 recognize the same language is now easy to see. A string x is accepted by M_1 if $\delta_1^*(q_1, x) \in A_1$; we can now say that this is true if and only if $\delta^*(q_0, x) \in A_1$; and using the definition of A_1, we conclude that this is true if and only if $\delta^*(q_0, x) \cap A \neq \emptyset$. In other words, x is accepted by M_1 if and only if x is accepted by M.

It is important to realize that the proof of the theorem provides an algorithm (the subset construction) for removing the nondeterminism from an NFA. Let us illustrate the algorithm by returning to the NFA in Example 4.2.

EXAMPLE 4.3 Applying the Subset Construction to Convert an NFA to an FA

The table in Example 4.2 shows the transition function of the NFA shown in Figure 4.2. The subset construction could produce an FA with as many as 16 states, since Q has 16 subsets. (In general, if M has n states, M_1 may have as many as 2^n, the number of subsets of a set with n elements.) However, we can get by with fewer if we follow the same approach as in Example 3.18 and use only those states that are reachable from the initial state. The transition function in our FA will be δ_1, and we proceed to calculate some of its values. Each time a new state (i.e., a new subset) S appears in our calculation, we must subsequently include the calculation of $\delta_1(S, 0)$ and $\delta_1(S, 1)$:

$$\begin{aligned}
\delta_1(\{q_0\}, 0) &= \{q_0\}\\
\delta_1(\{q_0\}, 1) &= \{q_0, q_1\}\\
\delta_1(\{q_0, q_1\}, 0) &= \delta(\{q_0\}, 0) \cup \delta(\{q_1\}, 0) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}\\
\delta_1(\{q_0, q_1\}, 1) &= \delta(\{q_0\}, 1) \cup \delta(\{q_1\}, 1) = \{q_0, q_1\} \cup \{q_2\} = \{q_0, q_1, q_2\}\\
\delta_1(\{q_0, q_2\}, 0) &= \delta(\{q_0\}, 0) \cup \delta(\{q_2\}, 0) = \{q_0\} \cup \{q_3\} = \{q_0, q_3\}\\
\delta_1(\{q_0, q_2\}, 1) &= \{q_0, q_1\} \cup \{q_3\} = \{q_0, q_1, q_3\}
\end{aligned}$$

It turns out that in the course of the calculation, eight distinct states (i.e., sets) arise. We knew already from the discussion in Example 3.17 that at least this many would be necessary.

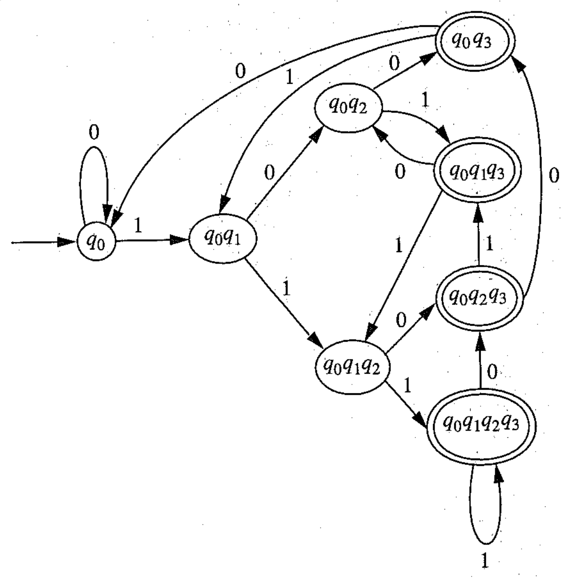

Figure 4.4 | The subset construction applied to the NFA in Figure 4.2.

Therefore, although we should not expect this to happen in general, the calculation in this case produces the FA with the fewest possible states recognizing the desired language. It is shown in Figure 4.4.

Another Example Illustrating the Subset Construction EXAMPLE 4.4

We close this section by returning to the first NFA we looked at, the one shown in Figure 4.1*b*. In fact, the FA produced by our algorithm is the one in Figure 4.1*a*; let us see how we obtain it.

If we carry out the calculations analogous to those in the previous example, the distinct states of the FA that are necessary, in the order they appear, turn out to be $\{q_0\}$, $\{q_4\}$, $\{q_1, q_2\}$, $\emptyset$, $\{q_0, q_3\}$, and $\{q_0, q_4\}$. The resulting transition table follows:

q	$\delta_1(q, 0)$	$\delta_1(q, 1)$
$\{q_0\}$	$\{q_4\}$	$\{q_1, q_2\}$
$\{q_4\}$	$\emptyset$	$\emptyset$
$\{q_1, q_2\}$	$\emptyset$	$\{q_0, q_3\}$
$\emptyset$	$\emptyset$	$\emptyset$
$\{q_0, q_3\}$	$\{q_0, q_4\}$	$\{q_1, q_2\}$
$\{q_0, q_4\}$	$\{q_4\}$	$\{q_1, q_2\}$

You can recognize this as the FA shown in Figure 4.1*a* by substituting p for $\{q_4\}$, r for $\{q_1, q_2\}$, s for $\emptyset$, t for $\{q_0, q_3\}$, and u for $\{q_0, q_4\}$. (Strictly speaking, you should also substitute q_0 for $\{q_0\}$.)

4.2 | NONDETERMINISTIC FINITE AUTOMATA WITH Λ-TRANSITIONS

One further modification of finite automata will be helpful in our upcoming proof that regular languages are the same as those accepted by FAs. We introduce the new devices by an example that suggests some of the ways they will be used in the proof.

How a Device More General Than an NFA Can Be Useful EXAMPLE 4.5

Figures 4.5*a* and 4.5*b* show simple NFAs M_1 and M_2 accepting the two languages $L_1 = \{0\}\{1\}^*$ and $L_2 = \{0\}^*\{1\}$, respectively, over the alphabet $\{0, 1\}$.

We consider two ways of combining these languages, using the operations of concatenation and union (two of the three operations involved in the definition of regular languages), and in each case we will try to incorporate the existing NFAs in a composite device M accepting the language we are interested in. Ideally, the structure of M_1 and M_2 will be preserved more or less intact in the resulting machine, and the way we combine the two NFAs will depend only on the operation being used to combine the two languages.

First we try the language L_1L_2 corresponding to the regular expression 01^*0^*1. In this case, M should in some sense start out the way M_1 does and finish up the way M_2 does. The

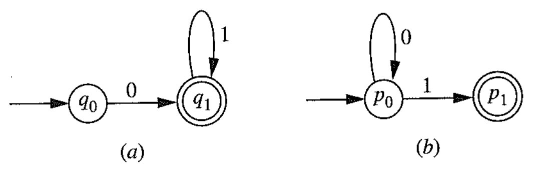

Figure 4.5 |
(*a*) An NFA accepting {0}{1}*; (*b*) An NFA accepting {0}*{1}.

question is just how to connect the two. The string 0 takes M_1 from q_0 to q_1; since 0 is not itself an element of L_1L_2, we do not expect the state q_1 to be an accepting state in M. We might consider the possible ways to interpret an input symbol once we have reached the state q_1. At that point, a 0 (if it is part of a string that will be accepted) can only be part of the 0* term from the second language. This might suggest connecting q_1 to p_0 by an arrow labeled 0; this cannot be the only connection, because a string in L does not require a 0 at this point. The symbol 1 in state q_1 could be either part of the 1* term from the first language, which suggests a connecting arrow from q_1 to p_0 labeled 1, or the 1 corresponding to the second language, which suggests an arrow from q_1 to p_1 labeled 1. These three connecting arrows turn out to be enough, and the resulting NFA is shown in Figure 4.6*a*.

We introduced a procedure in Chapter 3 to take care of the union of two languages provided we have an FA for each one, but it is not hard to see that the method is not always satisfactory if we have NFAs instead (Exercise 4.12). One possible NFA to handle $L_1 \cup L_2$ is shown in Figure 4.6*b*. This time it may not be obvious whether making q_0 the initial state, rather than p_0, is a natural choice or simply an arbitrary one. The label 1 on the connecting arrow from q_0 to p_1 is the 1 in the regular expression 0*1, and the 0 on the arrow from q_0 to p_0 is part of the 0* in the same regular expression. The NFA accepts the language corresponding to the regular expression $01^* + 1 + 00^*1$, which can be simplified to $01^* + 0^*1$.

For both L_1L_2 and $L_1 \cup L_2$, we have found relatively simple composite NFAs, and it is possible in both cases to see how the two original diagrams are incorporated into the result. However, in both cases the structure of the original diagrams has been obscured somewhat in the process of combining them, because the extra arrows used to connect the two depend on the regular expressions being combined, not just on the combining operation. We do not yet seem to have a general method that will work for two arbitrary NFAs.

One way to solve this problem, it turns out, is to combine the two NFAs so as to create a nondeterministic device with even a little more freedom in guessing. In the case of concatenation, for example, we will allow the new device to guess while in state q_1 that it will receive no more inputs that are to be matched with 1*. In making this guess it commits itself to proceeding to the second part of the regular expression 01*0*1; it will be able to make this guess, however, without any reference to the actual structure of the second part—in particular, without receiving any input (or, what amounts to the same thing, receiving only the null string Λ as input). The resulting diagram is shown in Figure 4.7*a*.

In the second case, we provide the NFA with an initial state that has nothing to do with either M_1 or M_2, and we allow it to guess before it has received any input whether it will be looking for an input string in L_1 or one in L_2. Making this guess requires only Λ as input and allows the machine to begin operating exactly like M_1 or M_2, whichever is appropriate. The result is shown in Figure 4.7*b*.

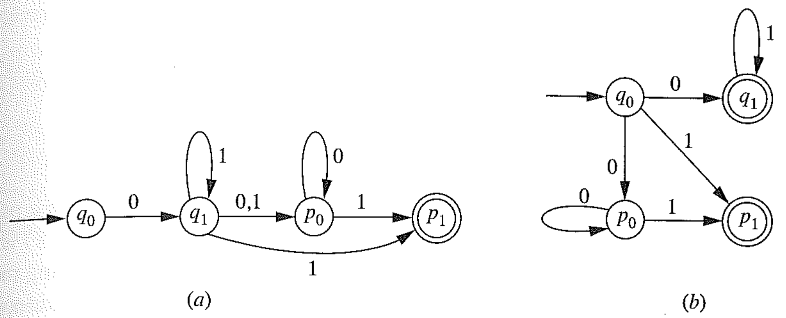

Figure 4.6 |
(*a*) An NFA accepting {0}{1}*{0}*{1}; (*b*) An NFA accepting {0}{1}* ∪ {0}*{1}.

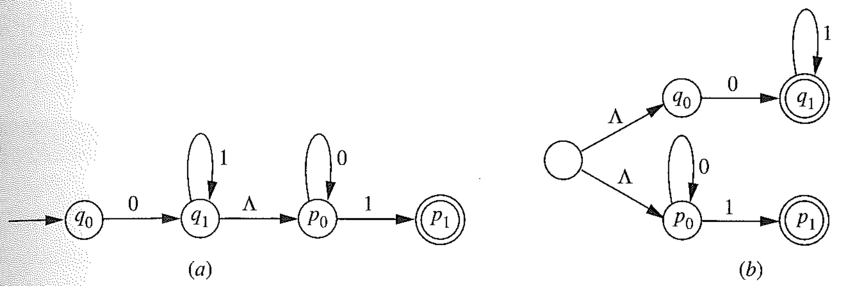

Figure 4.7 |
(*a*) An NFA-Λ accepting {0}{1}*{0}*{1}; (*b*) An NFA-Λ accepting {0}{1}* ∪ {0}*{1}.

To summarize, the devices in Figure 4.7 are more general than NFAs in that they are allowed to make transitions, not only on input symbols from the alphabet, but also on null inputs.

Definition 4.4 A Nondeterministic Finite Automaton with Λ-Transitions

A nondeterministic finite automaton with Λ-transitions (abbreviated NFA-Λ) is a 5-tuple $(Q, \Sigma, q_0, A, \delta)$, where Q and Σ are finite sets, $q_0 \in Q$, $A \subseteq Q$, and

$$\delta : Q \times (\Sigma \cup \{\Lambda\}) \to 2^Q$$

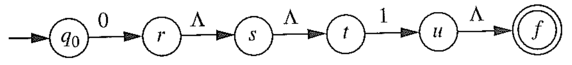

Figure 4.8 |

As before, we need to define an extended function δ^* in order to give a precise definition of acceptance of a string by an NFA-Λ. The idea is still that $\delta^*(q, x)$ will be the set of all states in which the NFA-Λ can legally end up as a result of starting in state q and processing the symbols in x. However, there is now a further complication, since "processing the symbols in x" allows for the possibility of Λ-transitions interspersed among ordinary transitions. Figure 4.8 illustrates this. We want to say that the string 01 is accepted, since $0\Lambda\Lambda 1\Lambda = 01$ and the path corresponding to these five inputs leads from q_0 to an accepting state.

Again we start with a nonrecursive definition, which is a straightforward if not especially elegant adaptation of Definition 4.2a.

Definition 4.5a Nonrecursive Definition of δ^* for an NFA-Λ

For an NFA-Λ $M = (Q, \Sigma, q_0, A, \delta)$, states $p, q \in Q$, and a string $x = a_1a_2\cdots a_n \in \Sigma^*$, we will say M moves from p to q by a sequence of transitions corresponding to x if there exist an integer $m \geq n$, a sequence $b_1, b_2, \ldots, b_m \in \Sigma \cup \{\Lambda\}$ satisfying $b_1b_2\cdots b_m = x$, and a sequence of states $p = p_0, p_1, \ldots, p_m = q$ so that for each i with $1 \leq i \leq m$, $p_i \in \delta(p_{i-1}, b_i)$.

For $x \in \Sigma^*$ and $p \in Q$, $\delta^*(p, x)$ is the set of all states $q \in Q$ such that there is a sequence of transitions corresponding to x by which M moves from p to q.

For example, in Figure 4.8 there is a sequence of transitions corresponding to 01 by which the device moves from q_0 to f; there is a sequence of transitions corresponding to the string Λ by which it moves from r to t; and there is also a sequence of transitions corresponding to Λ by which it moves from any state to itself (namely, the empty sequence).

Coming up with a recursive definition of δ^* is not as easy this time. The recursive part of the definition will still involve defining δ^* for a string with one extra alphabet symbol a. However, if we denote by S the set of states that the device may be in before the a is processed, we obtain the new set by allowing all possible transitions from elements of S on the input a, as well as all subsequent Λ-transitions. This suggests that we also want to modify the basis part of the definition, so that the set $\delta^*(q, \Lambda)$ contains not only q but any other states that the NFA-Λ can reach from q by using Λ-transitions. Both these modifications can be described in terms of the Λ-*closure* of a set S of states. This set, which we define recursively in Definition 4.6, is to be the set of all states that can be reached from elements of S by using Λ-transitions.

Definition 4.6 Λ-Closure of a Set of States

Let $M = (Q, \Sigma, q_0, A, \delta)$ be an NFA-Λ, and let S be any subset of Q. The Λ-closure of S is the set $\Lambda(S)$ defined as follows.

1. Every element of S is an element of $\Lambda(S)$;
2. For any $q \in \Lambda(S)$, every element of $\delta(q, \Lambda)$ is in $\Lambda(S)$;
3. No other elements of Q are in $\Lambda(S)$.

We know in advance that the set $\Lambda(S)$ is finite. As a result, we can translate the recursive definition into an algorithm for calculating $\Lambda(S)$ (Exercise 2.70).

Algorithm to Calculate $\Lambda(S)$ Start with $T = S$. Make a sequence of passes, in each pass considering every $q \in T$ and adding to T all elements of $\delta(q, \Lambda)$ that are not already elements of T. Stop after any pass in which T does not change. The set $\Lambda(S)$ is the final value of T. ■

The Λ-closure of a set is the extra ingredient we need to define the function δ^* recursively. If $\delta^*(q, y)$ is the set of all the states that can be reached from q using the symbols of y as well as Λ-transitions, then

$$\bigcup_{r \in \delta^*(q,y)} \delta(r, a)$$

is the set of states we can reach in one more step by using the symbol a, and the Λ-closure of this set includes any additional states that we can reach with subsequent Λ-transitions.

Definition 4.5b Recursive Definition of δ^* for an NFA-Λ

Let $M = (Q, \Sigma, q_0, A, \delta)$ be an NFA-Λ. The extended transition function $\delta^* : Q \times \Sigma^* \to 2^Q$ is defined as follows.

1. For any $q \in Q$, $\delta^*(q, \Lambda) = \Lambda(\{q\})$.
2. For any $q \in Q$, $y \in \Sigma^*$, and $a \in \Sigma$,

$$\delta^*(q, ya) = \Lambda\left(\bigcup_{r \in \delta^*(q,y)} \delta(r, a)\right)$$

A string x is accepted by M if $\delta^*(q_0, x) \cap A \neq \emptyset$. The language recognized by M is the set $L(M)$ of all strings accepted by M.

EXAMPLE 4.6 Applying the Definitions of $\Lambda(S)$ and δ^*

We consider the NFA-Λ shown in Figure 4.9, first as an illustration of how to apply the algorithm for computing $\Lambda(S)$, and then to demonstrate that Definition 4.5b really does make it possible to calculate $\delta^*(q, x)$.

Suppose we consider $S = \{s\}$. After one iteration, T is $\{s, w\}$, after two iterations it is $\{s, w, q_0\}$, after three iterations it is $\{s, w, q_0, p, t\}$, and in the next iteration it is unchanged. $\Lambda(\{s\})$ is therefore $\{s, w, q_0, p, t\}$.

Let us calculate $\delta^*(q_0, 010)$ using the recursive definition. This set is defined in terms of $\delta^*(q_0, 01)$, which is defined in terms of $\delta^*(q_0, 0)$, and so on. We therefore approach the calculation from the bottom up, calculating $\delta^*(q_0, \Lambda)$, then $\delta^*(q_0, 0)$, $\delta^*(q_0, 01)$, and finally $\delta^*(q_0, 010)$:

$$\begin{aligned}
\delta^*(q_0, \Lambda) &= \Lambda(\{q_0\}) \\
&= \{q_0, p, t\}
\end{aligned}$$

$$\begin{aligned}
\delta^*(q_0, 0) &= \Lambda\left(\bigcup_{p \in \delta^*(q_0, \Lambda)} \delta(p, 0)\right) \\
&= \Lambda(\delta(q_0, 0) \cup \delta(p, 0) \cup \delta(t, 0)) \\
&= \Lambda(\emptyset \cup \{p\} \cup \{u\}) \\
&= \Lambda(\{p, u\}) \\
&= \{p, u\}
\end{aligned}$$

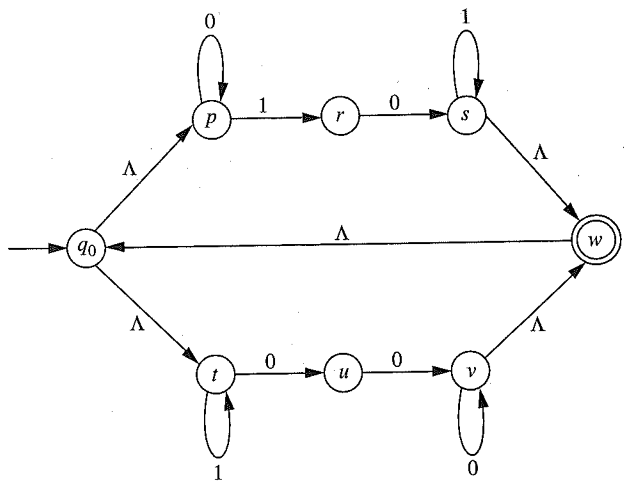

Figure 4.9 |
The NFA-Λ for Example 4.6.

$$\begin{aligned}
\delta^*(q_0, 01) &= \Lambda\left(\bigcup_{p \in \delta^*(q_0, 0)} \delta(p, 1)\right) \\
&= \Lambda(\delta(p, 1) \cup \delta(u, 1)) \\
&= \Lambda(\{r\}) \\
&= \{r\}
\end{aligned}$$

$$\begin{aligned}
\delta^*(q_0, 010) &= \Lambda\left(\bigcup_{p \in \delta^*(q_0, 01)} \delta(p, 0)\right) \\
&= \Lambda(\delta(r, 0)) \\
&= \Lambda(\{s\}) \\
&= \{s, w, q_0, p, t\}
\end{aligned}$$

(The last equality is the observation we made near the beginning of this example.) Because $\delta^*(q_0, 010)$ contains w, and because w is an element of A, 010 is accepted.

Looking at the figure, you might argue along this line instead: The string $\Lambda 010 \Lambda$ is the same as 010; the picture shows the sequence

$$q_0 \xrightarrow{\Lambda} p \xrightarrow{0} p \xrightarrow{1} r \xrightarrow{0} s \xrightarrow{\Lambda} w$$

of transitions; therefore 010 is accepted. In an example as simple as this one, going through the detailed calculations is not necessary in order to decide whether a string is accepted. The point, however, is that with the recursive definitions of $\Lambda(S)$ and δ^*, we can proceed on a solid algorithmic basis and be confident that the calculations (which are indeed feasible) will produce the correct answer.

In Section 4.1, we showed (Theorem 4.1) that NFAs are no more powerful than FAs with regard to the languages they can accept. In order to establish the same result for NFA-Λs, it is sufficient to show that any NFA-Λ can be replaced by an equivalent NFA; the notation we have developed now allows us to prove this.

Theorem 4.2
If $L \subseteq \Sigma^*$ is a language that is accepted by the NFA-Λ $M = (Q, \Sigma, q_0, A, \delta)$, then there is an NFA $M_1 = (Q_1, \Sigma, q_1, A_1, \delta_1)$ that also accepts L.

Proof
In the proof of Theorem 4.1, we started with an NFA and removed all traces of nondeterminism by changing our notion of *state*. A Λ-transition is a type of nondeterminism; for example, if the transitions in M include

$$p \xrightarrow{0} q \xrightarrow{\Lambda} r$$

then from state p, the input symbol 0 allows us to go to either q or r. Since M_1 is still allowed to be nondeterministic, we can eliminate the Λ-transition without changing the states, by simply adding the transition from p to r on input 0.

This approach will work in general. We can use the same state set Q and the same initial state in M_1. What we must do is to define the transition function δ_1 so that if M allows us to move from p to q using certain symbols together with Λ-transitions, then M_1 will allow us to move from p to q using those symbols without the Λ-transitions.

Essentially, we have the solution already, as a result of the definition of δ^* for the NFA-Λ M (Definition 4.5b). For a state q and an input symbol a,

$$\begin{aligned}\delta^*(q, a) &= \delta^*(q, \Lambda a)\\ &= \Lambda\left(\bigcup_{r \in \delta^*(q,\Lambda)} \delta(r, a)\right)\\ &= \Lambda\left(\bigcup_{r \in \Lambda(\{q\})} \delta(r, a)\right)\end{aligned}$$

Remember what this means: $\cup_{r\in\Lambda(\{q\})}\delta(r, a)$ is the set of states that can be reached from q, using the input symbol a but possibly also Λ-transitions beforehand. The Λ-closure of this set is the set of states that can be reached from q, using the input symbol a but allowing Λ-transitions both before and after. This is exactly what we want $\delta_1(q, a)$ to be, in order to incorporate into our NFA transition function the nondeterminism that arises from the possibility of Λ-transitions.

Finally, there is a slight change that may be needed in specifying the accepting states of M_1. If the initial state q_0 of M is not an accepting state, but it is possible to get from q_0 to an accepting state using only Λ-transitions, then q_0 must be made an accepting state in M_1—otherwise $\delta_1^*(q_0, \Lambda)$, which is $\{q_0\}$ because M_1 is an NFA, will not intersect A.

To summarize, we have decided that M_1 will be the NFA $(Q, \Sigma, q_0, A_1, \delta_1)$, where for any $q \in Q$ and $a \in \Sigma$,

$$\delta_1(q, a) = \delta^*(q, a)$$

and

$$A_1 = \begin{cases} A \cup \{q_0\} & \text{if } \Lambda(\{q_0\}) \cap A \neq \emptyset \text{ in } M \\ A & \text{otherwise} \end{cases}$$

The point of our definition of δ_1 is that we want $\delta_1^*(q, x)$ to be the set of states that M can reach from q, using the symbols of the string x together with Λ-transitions. In other words, we want

$$\delta_1^*(q, x) = \delta^*(q, x)$$

This formula may not be correct when $x = \Lambda$ (because M_1 is an NFA, $\delta_1^*(q, \Lambda) = \{q\}$, whereas $\delta^*(q, \Lambda)$ may contain additional states), and that is the reason for our definition of A_1 above. We now show, however, that for any nonnull string the formula is true.

The proof is by structural induction on x. For the basis step we consider $x = a \in \Sigma$. In this case, $\delta^*(q, x) = \delta_1(q, a)$ by definition of δ_1, and $\delta_1(q, a) = \delta_1^*(q, a)$ simply because M_1 is an NFA (see Example 4.2 and Exercise 4.3).

In the induction step, we assume that $|y| \geq 1$ and that $\delta_1^*(q, y) = \delta^*(q, y)$ for any $q \in Q$. We want to show that for $a \in \Sigma$, $\delta_1^*(q, ya) = \delta^*(q, ya)$. We have

$$\begin{aligned}\delta_1^*(q, ya) &= \bigcup_{r\in\delta_1^*(q,y)} \delta_1(r, a) \quad \text{(by Definition 4.2b)}\\ &= \bigcup_{r\in\delta^*(q,y)} \delta_1(r, a) \quad \text{(by the induction hypothesis)}\\ &= \bigcup_{r\in\delta^*(q,y)} \delta^*(r, a) \quad \text{(by definition of } \delta_1)\\ &= \bigcup_{r\in\delta^*(q,y)} \Lambda\left(\bigcup_{\rho\in\Lambda\{r\}} \delta(\rho, a)\right) \quad \text{(by Definition 4.5b)}\end{aligned}$$

The Λ-closure operator has the property that the union of Λ-closures is the Λ-closure of the union (Exercise 4.40). We may therefore write the last set as

$$\Lambda\left(\bigcup_{r\in\delta^*(q,y)}\left(\bigcup_{\rho\in\Lambda\{r\}} \delta(\rho, a)\right)\right)$$

However, it follows from the definition of Λ-closure (and the fact that $\delta^*(q, y)$ is a Λ-closure) that for every $r \in \delta^*(q, y)$, $\Lambda\{r\} \subseteq \delta^*(q, y)$; therefore, the two separate unions are unnecessary, and the formula reduces to

$$\Lambda\left(\bigcup_{r\in\delta^*(q,y)} \delta(r, a)\right)$$

which is the definition of $\delta^*(q, ya)$. The induction is complete.

Now it is not hard to show that M_1 recognizes L, the language recognized by M. First we consider the case when $\Lambda(\{q_0\}) \cap A = \emptyset$ in M. Then the null string is accepted by neither M nor M_1. For any other string x, we have shown that $\delta_1^*(q_0, x) = \delta^*(q_0, x)$, and we know that the accepting states of M and M_1 are the same. Therefore, x is accepted by M_1 if and only if it is accepted by M.

In the other case, when $\Lambda(\{q_0\}) \cap A \neq \emptyset$, we have defined A_1 to be $A \cup \{q_0\}$. This time Λ is accepted by both M and M_1. For any other x, again we have $\delta_1^*(q_0, x) = \delta^*(q_0, x)$. If this set contains a state in A, then both M and M_1 accept x. If not, the only way x could be accepted by one of the two machines but not the other would be for $\delta^*(q_0, x)$ to contain q_0. (It would then intersect A_1 but not A.) However, this cannot happen either. By definition, $\delta^*(q_0, x)$ is the Λ-closure of a set. If $\delta^*(q_0, x)$ contained q_0, it would have to contain every element of $\Lambda(\{q_0\})$, and therefore, since $\Lambda(\{q_0\}) \cap A \neq \emptyset$, it would have to contain an element of A.

In either case, we may conclude that M and M_1 accept exactly the same strings, and the proof is complete.

We think of NFAs as generalizations of FAs, and of NFA-Λs as generalizations of NFAs. It is technically not *quite* correct to say that every FA is an NFA, since the values of the transition function are states in one case and sets of states in the other; similarly, the domain of the transition function in an NFA is $Q \times \Sigma$, and in an NFA-Λ it is $Q \times (\Sigma \cup \{\Lambda\})$. Practically speaking, we can ignore these technicalities. The following theorem formalizes this assertion and ties together the results of Theorems 4.1 and 4.2.

Theorem 4.3
For any alphabet Σ, and any language $L \subset \Sigma^*$, these three statements are equivalent:

1. L can be recognized by an FA.
2. L can be recognized by an NFA.
3. L can be recognized by an NFA-Λ.

Proof
According to Theorems 4.1 and 4.2, statement 3 implies statement 2, and statement 2 implies statement 1. It is therefore sufficient to show that statement 1 implies statement 3.

Suppose that L is accepted by the FA $M = (Q, \Sigma, q_0, A, \delta)$. We construct an NFA-$\Lambda$ $M_1 = (Q, \Sigma, q_0, A, \delta_1)$ accepting L as follows. $\delta_1 : Q \times (\Sigma \cup \{\Lambda\}) \to 2^Q$ is defined by the formulas

$$\delta_1(q, \Lambda) = \emptyset \qquad \delta_1(q, a) = \{\delta(q, a)\}$$

(for any $q \in Q$ and any $a \in \Sigma$). The first formula just says that there are no Λ-transitions in M_1, and the second says that the transition functions δ and δ_1 are identical (so that there is really no nondeterminism at all), except for the technical difference that the value is a state in one case and a set containing only that state in the other. It can be verified almost immediately that M_1 accepts L (Exercise 4.21).

Just as in the case of Theorem 4.1, the proof of Theorem 4.2 provides us with an algorithm for eliminating Λ-transitions from an NFA-Λ. We illustrate the algorithm in two examples, in which we can also practice the algorithm for eliminating nondeterminism.

Converting an NFA-Λ to an NFA — EXAMPLE 4.7

Let M be the NFA-Λ pictured in Figure 4.10*a*, which accepts the language $\{0\}^*\{01\}^*\{0\}^*$. We show in tabular form the values of the transition function, as well as the values $\delta^*(q, 0)$ and $\delta^*(q, 1)$ that give us the values of the transition function in the resulting NFA.

q	$\delta(q,\Lambda)$	$\delta(q,0)$	$\delta(q,1)$	$\delta^*(q,0)$	$\delta^*(q,1)$
A	$\{B\}$	$\{A\}$	$\emptyset$	$\{A,B,C,D\}$	$\emptyset$
B	$\{D\}$	$\{C\}$	$\emptyset$	$\{C,D\}$	$\emptyset$
C	$\emptyset$	$\emptyset$	$\{B\}$	$\emptyset$	$\{B,D\}$
D	$\emptyset$	$\{D\}$	$\emptyset$	$\{D\}$	$\emptyset$

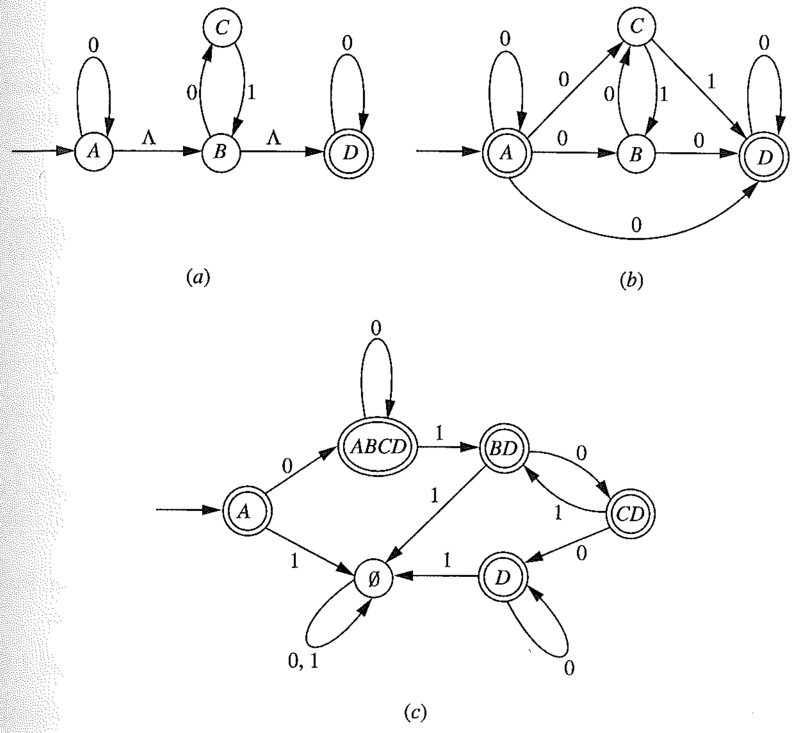

Figure 4.10 |
An NFA-Λ, an NFA, and an FA for $\{0\}^*\{01\}^*\{0\}^*$.

For example, $\delta^*(A, 0)$ is calculated using the formula

$$\delta^*(A,0) = \Lambda\left(\bigcup_{r\in\Lambda(\{A\})} \delta(r,0)\right)$$

In a more involved example we might feel more comfortable carrying out each step literally: calculating $\Lambda(\{A\})$, finding $\delta(r, 0)$ for each r in this set, forming the union, and calculating the Λ-closure of the result. In this simple example, we can see that from A with input 0, M can stay in A, move to B (using a 0 followed by a Λ-transition), move to C (using a Λ-transition and then a 0), or move to D (using the 0 either immediately preceded or immediately followed by two Λ-transitions). The other entries in the last two columns of the table can be obtained similarly. Since M can move from the initial state to D using only Λ-transitions, A must be an accepting state in M_1, which is shown in Figure 4.10*b*.

Having the values $\delta_1(q, 0)$ and $\delta_1(q, 1)$ in tabular form is useful in arriving at an FA. For example (if we denote the transition function of the FA by δ_2), in order to compute $\delta_2(\{C, D\}, 0)$ we simply form the union of the sets in the third and fourth rows of the $\delta^*(q, 0)$ column of the table; the result is $\{D\}$. It turns out in this example that the sets of states that came up as we filled in the table for the NFA were all we needed, and the resulting FA is shown in Figure 4.10*c*.

EXAMPLE 4.8 Another Example of Converting an NFA-Λ to an NFA

For our last example we consider the NFA-Λ in Figure 4.11*a*, recognizing the language $\{0\}^*(\{01\}^*\{1\} \cup \{1\}^*\{0\})$. Again we show the transition function in tabular form, as well as the transition function for the resulting NFA.

q	$\delta(q, \Lambda)$	$\delta(q, 0)$	$\delta(q, 1)$	$\delta^*(q, 0)$	$\delta^*(q, 1)$
A	$\{B, D\}$	$\{A\}$	$\emptyset$	$\{A, B, C, D, E\}$	$\{D, E\}$
B	$\emptyset$	$\{C\}$	$\{E\}$	$\{C\}$	$\{E\}$
C	$\emptyset$	$\emptyset$	$\{B\}$	$\emptyset$	$\{B\}$
D	$\emptyset$	$\{E\}$	$\{D\}$	$\{E\}$	$\{D\}$
E	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

The NFA we end up with is shown in Figure 4.11*b*. Note that the initial state A is not an accepting state, since in the original device an accepting state cannot be reached with only Λ-transitions.

Unlike the previous example, when we start to eliminate the nondeterminism from our NFA, new sets of states are introduced in addition to the ones shown. For example,

$$\delta_2(\{A, B, C, D, E\}, 1) = \{D, E\} \cup \{E\} \cup \{B\} \cup \{D\} \cup \emptyset = \{B, D, E\}$$
$$\delta_2(\{B, D, E\}, 0) = \{C\} \cup \{E\} \cup \emptyset = \{C, E\}$$

The calculations are straightforward, and the result is shown in Figure 4.11*c*.

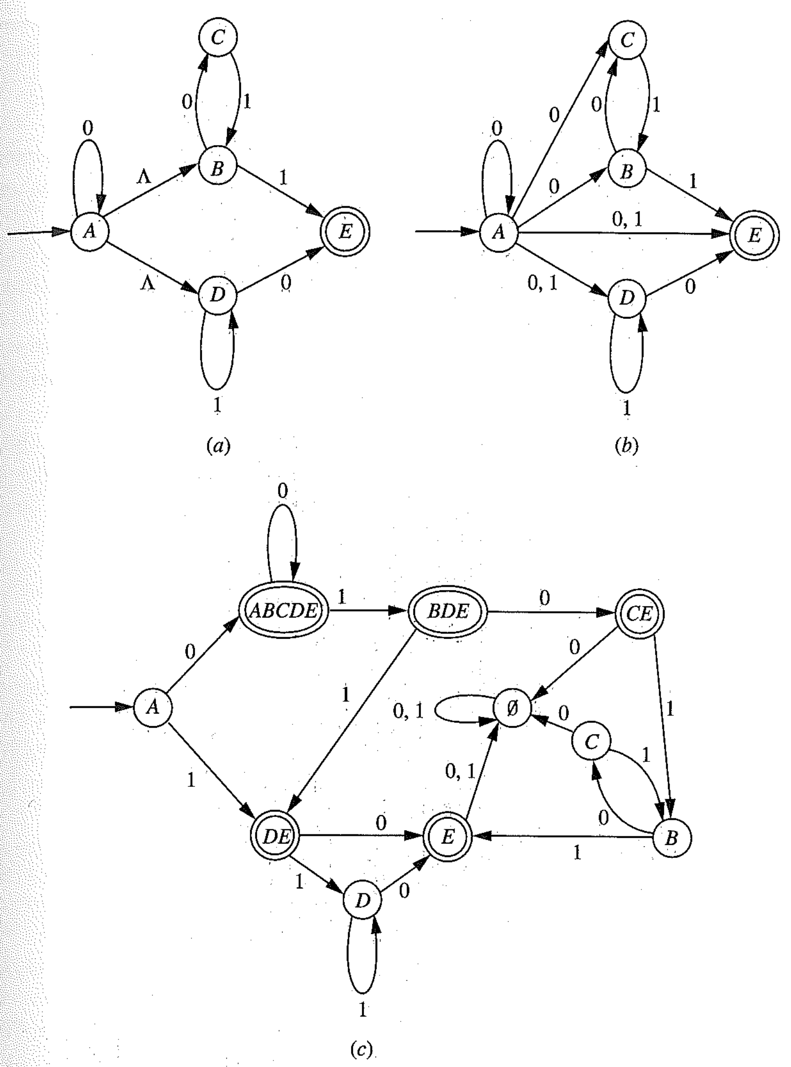

Figure 4.11 | An NFA-Λ, an NFA, and an FA for $\{0\}^*\{01\}^*\{1\} \cup \{1\}^*\{0\}$.

4.3 | KLEENE'S THEOREM

Sections 4.1 and 4.2 of this chapter have provided the tools we need to prove Theorem 3.1. For convenience we have stated the two parts, the "if" and the "only if," as separate results.

Theorem 4.4 Kleene's Theorem, Part 1
Any regular language can be accepted by a finite automaton.

Proof
Because of Theorem 4.3, it is sufficient to show that every regular language can be accepted by an NFA-Λ. The set of regular languages over the alphabet Σ is defined in Definition 3.1 to contain the basic languages $\emptyset$, $\{\Lambda\}$, and $\{a\}$ ($a \in \Sigma$), to be closed under the operations of union, concatenation, and Kleene *, and to be the smallest set of languages that has those two properties. This allows us to prove using structural induction that every regular language over Σ can be accepted by an NFA-Λ. The basis step of the proof is to show that the three basic languages can be accepted by NFA-Λs. The induction hypothesis is that L_1 and L_2 are languages that can be accepted by NFA-Λs, and the induction step is to show that $L_1 \cup L_2$, L_2L_2, and L_1^* can also be. NFA-Λs for the three basic languages are shown in Figure 4.12.

Now suppose that L_1 and L_2 are recognized by the NFA-Λs M_1 and M_2, respectively, where for both $i = 1$ and $i = 2$,

$$M_i = (Q_i, \Sigma, q_i, A_i, \delta_i)$$

By renaming states if necessary, we may assume that $Q_1 \cap Q_2 = \emptyset$. We will construct NFA-Λs M_u, M_c, and M_k, recognizing the languages $L_1 \cup L_2$, L_1L_2, and L_1^*, respectively. Pictures will be helpful here also; schematic diagrams presenting the general idea in each case are shown in Figure 4.13. (You will observe that the first two cases have already been illustrated in Example 4.5.) In the three parts of the figure, both M_1 and M_2 are shown as having two accepting states.

Construction of $M_u = (Q_u, \Sigma, q_u, A_u, \delta_u)$. Let q_u be a new state, not in either Q_1 or Q_2, and let

$$Q_u = Q_1 \cup Q_2 \cup \{q_u\} \qquad A_u = A_1 \cup A_2$$

Now we define δ_u so that M_u can move from its initial state to either q_1 or q_2 by a Λ-transition, and then make exactly the same moves that the respective M_i would. Formally, we define

$$\delta_u(q_u, \Lambda) = \{q_1, q_2\} \qquad \delta_u(q_u, a) = \emptyset \text{ for every } a \in \Sigma$$

and for each $q \in Q_1 \cup Q_2$ and $a \in \Sigma \cup \{\Lambda\}$,

$$\delta_u(q, a) = \begin{cases} \delta_1(q, a) & \text{if } q \in Q_1 \\ \delta_2(q, a) & \text{if } q \in Q_2 \end{cases}$$

For either value of i, if $x \in L_i$, then M_u can process x by moving to q_i on a Λ-transition and then executing the moves that cause M_i to accept x. On the other hand, if x is accepted by M_u, there is a sequence of transitions corresponding to x, starting at q_u and ending at an element of A_1 or A_2. The first of these transitions must be a Λ-transition from q_u to either q_1 or q_2, since there are no other transitions from q_u. Thereafter, since $Q_1 \cap Q_2 = \emptyset$, either

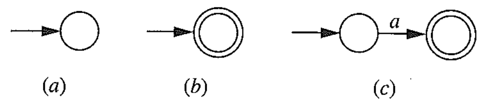

Figure 4.12 |
NFA-Λs for the three basic regular languages.

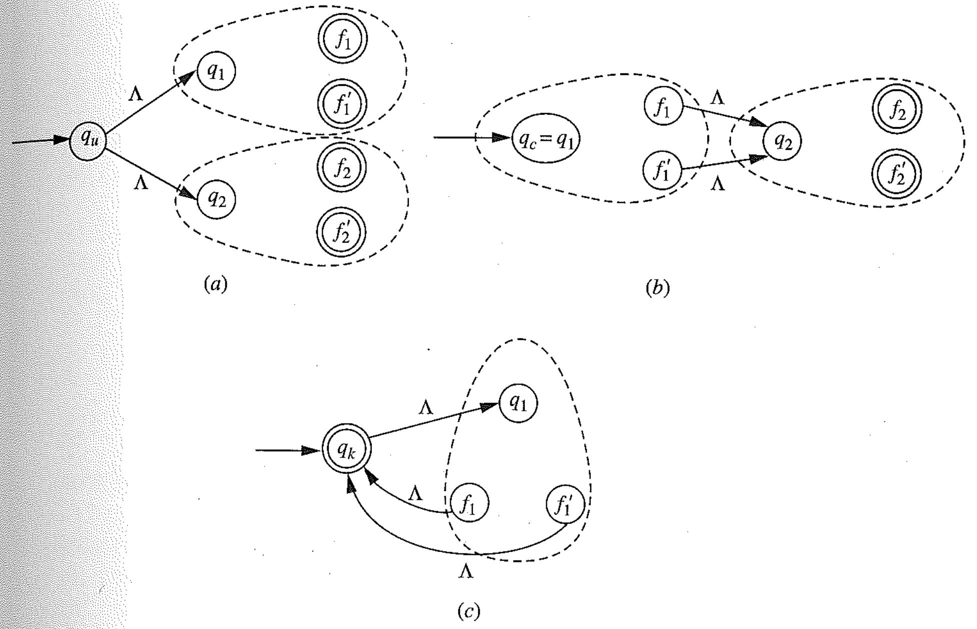

Figure 4.13 |
NFA-Λs for union, concatenation, and Kleene *.

all the transitions are between elements of Q_1 or all are between elements of Q_2. It follows that x must be accepted by either M_1 or M_2.

Construction of $M_c = (Q_c, \Sigma, q_c, A_c, \delta_c)$. In this case we do not need any new states. Let $Q_c = Q_1 \cup Q_2$, $q_c = q_1$, and $A_c = A_2$. (In Figure 4.13*b*, we illustrate this by making f_2 and f_2' accepting states in M_c, but not f_1 or f_1'.) The transitions will include all those of M_1 and M_2, as well as a Λ-transition from each state in A_1 to q_2. In other words, for any q not in A_1, and $\alpha \in \Sigma \cup \{\Lambda\}$, $\delta_c(q, \alpha)$ is defined to be either $\delta_1(q, \alpha)$ or $\delta_2(q, \alpha)$, depending on whether q is in Q_1 or Q_2. For $q \in A_1$,

$$\delta_c(q, a) = \delta_1(q, a) \text{ for every } a \in \Sigma, \text{ and } \delta_c(q, \Lambda) = \delta_1(q, \Lambda) \cup \{q_2\}$$

On an input string x_1x_2, where $x_i \in L_i$ for both values of i, M_c can process x_1, arriving at a state in A_1; jump from this state to q_2 by a Λ-transition; and then process x_2 the way M_2 would, so that x_1x_2 is accepted. Conversely, if x is accepted by M_c, there is a sequence of transitions corresponding to x that begins at q_1 and ends at an element of A_2. One of them must therefore be from an element of Q_1 to an element of Q_2, and according to the definition of δ_c this can only be a Λ-transition from an element of A_1 to q_2. Because $Q_1 \cap Q_2 = \emptyset$, all the previous transitions are between elements of Q_1 and all the subsequent ones are between elements of Q_2. It follows that $x = x_1\Lambda x_2 = x_1x_2$, where x_1 is accepted by M_1 and x_2 is accepted by M_2; in other words, $x \in L_1L_2$.

Construction of $M_k = (Q_k, \Sigma, q_k, A_k, \delta_k)$. Let q_k be a new state not in Q_1, and let $Q_k = Q_1 \cup \{q_k\}$ and $A_k = \{q_k\}$. Once again all the transitions of M_1 will be allowed in M_k, but in addition there is a Λ-transition from q_k to q_1, and there is a Λ-transition from each element of A_1 to q_k. More precisely,

$\delta_k(q_k, \Lambda) = \{q_1\}$ and $\delta_k(q_k, a) = \emptyset$ for $a \in \Sigma$.

For $q \in Q_1$ and $\alpha \in \Sigma \cup \{\Lambda\}$, $\delta_k(q, \alpha) = \delta_1(q, \alpha)$ unless $q \in A_1$ and $\alpha = \Lambda$.

For $q \in A_1$, $\delta_k(q, \Lambda) = \delta_1(q, \Lambda) \cup \{q_k\}$.

Suppose $x \in L_1^*$. If $x = \Lambda$, then clearly x is accepted by M_k. Otherwise, for some $m \geq 1$, $x = x_1x_2 \cdots x_m$, where $x_i \in L_1$ for each i. M_k can move from q_k to q_1 by a Λ-transition; for each i, M_k moves from q_1 to an element f_i of A_1 by a sequence of transitions corresponding to x_i; and for each i, M_k then moves from f_i back to q_k by a Λ-transition. It follows that $(\Lambda x_1\Lambda)(\Lambda x_2\Lambda)\cdots(\Lambda x_m\Lambda) = x$ is accepted by M_k. On the other hand, if x is accepted by M_k, there is a sequence of transitions corresponding to x that begins and ends at q_k. Since the only transition from q_k is a Λ-transition to q_1, and the only transitions to q_k are Λ-transitions from elements of A_1, x can be decomposed in the form

$$x = (\Lambda x_1\Lambda)(\Lambda x_2\Lambda)\cdots(\Lambda x_m\Lambda)$$

where, for each i, there is a sequence of transitions corresponding to x_i from q_1 to an element of A_1. Therefore, $x \in L_1^*$.

Since we have constructed an NFA-Λ recognizing L in each of the three cases, the proof is complete.

The constructions in the proof of Theorem 4.4 provide an algorithm for constructing an NFA-Λ corresponding to a given regular expression. The next example illustrates its application, as well as the fact that there may be simplifications possible along the way.

Applying the Algorithm in the Proof of Theorem 4.4 EXAMPLE 4.9

Let r be the regular expression $(00 + 1)^*(10)^*$. We illustrate first the literal application of the algorithm, ignoring possible shortcuts. The primitive (zero-operation) regular expressions appearing in r are shown in Figure 4.14*a*. The NFA-Λs corresponding to 00 and 10 are now constructed using concatenation and are shown in Figure 4.14*b*. Next, we form the NFA-Λ corresponding to $(00 + 1)$, as in Figure 4.14*c*. Figures 4.14*d* and 4.14*e* illustrate the NFA-Λs corresponding to $(00 + 1)^*$ and $(10)^*$, respectively. Finally, the resulting NFA-Λ formed by concatenation is shown in Figure 4.14*f*.

It is probably clear in several places that there are states and Λ-transitions called for in the general construction that are not necessary in this example. The six parts of Figure 4.15

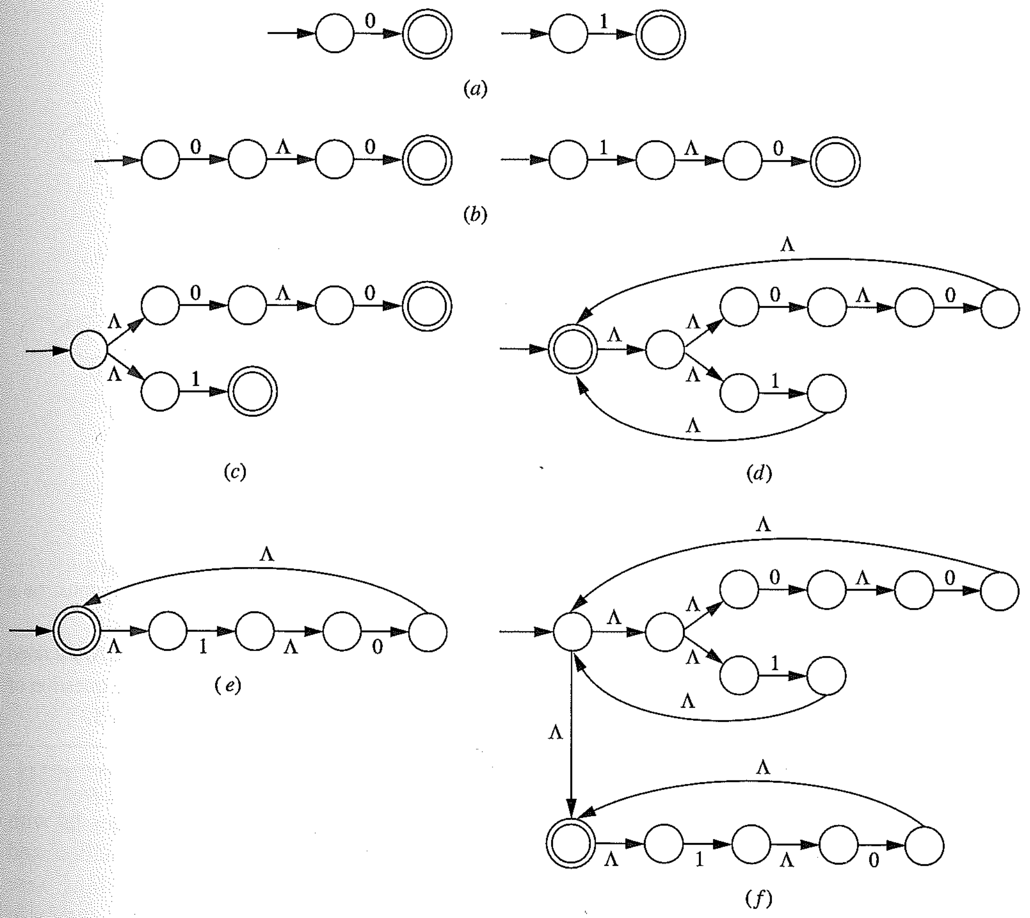

Figure 4.14 |
Constructing an NFA-Λ for $(00 + 1)^*(10)^*$.

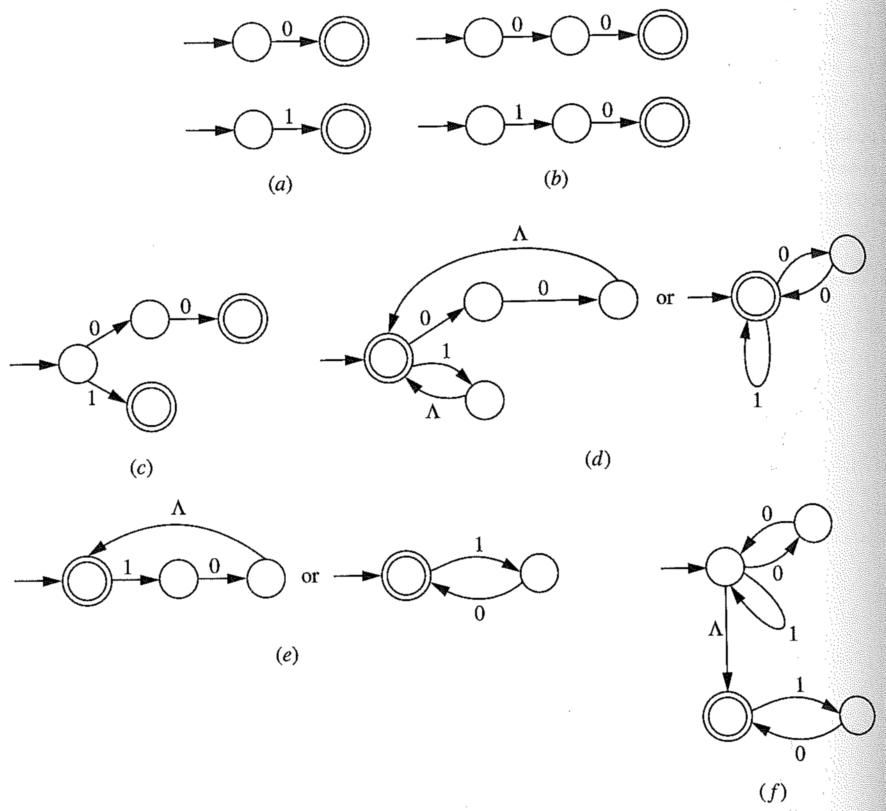

Figure 4.15 |
A simplified NFA-Λ for $(00 + 1)^*(10)^*$.

parallel those of Figure 4.14 and incorporate some obvious simplifications. One must be a little careful with simplifications such as these, as Exercises 4.32 to 4.34 illustrate.

We know from Sections 4.1 and 4.2 of this chapter how to convert an NFA-Λ obtained from Theorem 4.4 into an FA. Although we have not yet officially considered the question of simplifying a given FA as much as possible, we will see how to do this in Chapter 5.

If we have NFA-Λs M_1 and M_2, the proof of Theorem 4.4 provides us with algorithms for constructing new NFA-Λs to recognize the union, concatenation, and Kleene * of the corresponding languages. The first two of these algorithms were illustrated in Example 4.5. As a further example, we start with the FAs shown in Figures 4.16*a* and 4.16*b* (which were shown in Examples 3.12 and 3.13 to accept the languages $\{0, 1\}^*\{10\}$ and $\{00\}^*\{11\}^*$, respectively). We can apply the algorithms for union and Kleene *, making one simplification in the second step, to obtain the

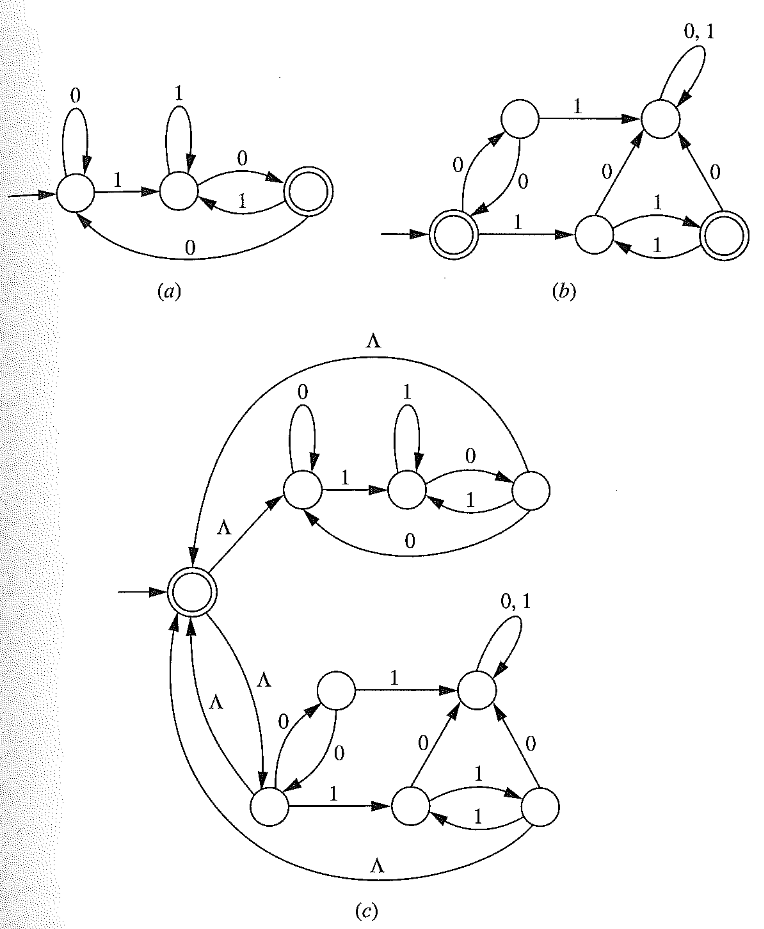

Figure 4.16 |
An NFA-Λ for $((0 + 1)^*10 + (00)^*(11)^*)^*$.

NFA-Λ in Figure 4.16*c* recognizing the language

$$(\{0, 1\}^*\{10\} \cup \{00\}^*\{11\}^*)^*$$

Theorem 4.5 Kleene's Theorem, Part 2
The language accepted by any finite automaton is regular.

Proof
Let $L \subseteq \Sigma^*$ be accepted by the FA $M = (Q, \Sigma, q_0, A, \delta)$. What this means is that $L = \{x \in \Sigma^* \mid \delta^*(q_0, x) \in A\}$. By considering the individual elements of A, we can express L as the union of a finite number of sets of the form $\{x \in \Sigma^* \mid \delta^*(q_0, x) = q\}$; because finite unions of regular

sets are regular, it will be sufficient to show that for any two states p and q, the set

$$L(p,q) = \{x \in \Sigma^* \mid \delta^*(p,x) = q\}$$

is regular.

In looking for ways to use mathematical induction, we have often formulated statements involving the length of a string. At first this approach does not seem promising, because we are trying to prove a property of a language, not a property of individual strings in the language. However, rather than looking at the number of transitions in a particular path from p to q (i.e., the length of a string in $L(p,q)$), we might look at the number of *distinct* states through which M passes in moving from p to q. It turns out to be more convenient to consider, for each k, a specific set of k states, and to consider the set of strings that cause M to go from p to q by going through only states in that set. If k is large enough, this set of strings will be all of $L(p,q)$.

To simplify notation, let us relabel the states of M using the integers 1 through n, where n is the number of states. Let us also formalize the idea of a path *going through* a state s: for a string x in Σ^*, we say x represents a path from p to q going through s if there are nonnull strings y and z so that

$$x = yz \qquad \delta^*(p,y) = s \qquad \delta^*(s,z) = q$$

Note that a path can go to a state, or from a state, without going through it. In particular, the path

$$p \xrightarrow{a} q \xrightarrow{b} r$$

goes through q, but not through p or r. Now, for $j \geq 0$, we let $L(p,q,j)$ be the set of strings corresponding to paths from p to q that go through no state numbered higher than j. No string in $L(p,q)$ can go through a state numbered higher than n, because there *are* no states numbered higher than n. In other words,

$$L(p,q,n) = L(p,q)$$

The problem, therefore, is to show that $L(p,q,n)$ is regular, and this will obviously follow if we can show that $L(p,q,j)$ is regular for every j with $0 \leq j \leq n$. (This is where the induction comes in.) In fact, there is no harm in asserting that $L(p,q,j)$ is regular for every $j \geq 0$; this is not really a stronger statement, since for any $j \geq n$, $L(p,q,j) = L(p,q,n)$, but this way it will look more like an ordinary induction proof.

For the basis step we need to show that $L(p,q,0)$ is regular. Going through no state numbered higher than 0 means going through no state at all, which means that the string can contain no more than one symbol. Therefore,

$$L(p,q,0) \subseteq \Sigma \cup \{\Lambda\}$$

and $L(p,q,0)$ is regular because it is finite. (See the discussion after the proof for an explicit formula for $L(p,q,0)$.)

The induction hypothesis is that $0 \leq k$ and that for every p and q satisfying $0 \leq p,q \leq n$, the language $L(p,q,k)$ is regular. We wish to show that for every p and q in the same range, $L(p,q,k+1)$ is regular. As we have already observed, $L(p,q,k+1) = L(p,q,k)$ if $k \geq n$, and we assume for the remainder of the proof that $k < n$.

A string x is in $L(p,q,k+1)$ if it represents a path from p to q that goes through no state numbered higher than $k+1$. There are two ways this can happen. First, the path can bypass the state $k+1$ altogether, in which case it goes through no state higher than k, and $x \in L(p,q,k)$. Second, the path can go through $k+1$ and nothing higher. In this case, it goes from p to the first occurrence of $k+1$, then loops from $k+1$ back to itself zero or more times, then goes from the last occurrence of $k+1$ to q. (See Figure 4.17.) This means that we can write x as yzw, where y corresponds to the path from p to the first occurrence of $k+1$, z to all the loops from $k+1$ back to itself, and w to the path from $k+1$ to q. The crucial observation here is that in each of the two parts y and w, and in each of the individual loops making up z, the path does not go through any state higher than k; in other words,

$$y \in L(p,k+1,k) \qquad w \in L(k+1,q,k) \qquad z \in L(k+1,k+1,k)^*.$$

It follows that in either of the two cases,

$$x \in L(p,q,k) \cup L(p,k+1,k)L(k+1,k+1,k)^*L(k+1,q,k)$$

On the other hand, it is also clear that any string in this right-hand set is an element of $L(p,q,k+1)$, since the corresponding path goes from p to q without going through any state higher than $k+1$. Therefore,

$$L(p,q,k+1) = L(p,q,k) \cup L(p,k+1,k)L(k+1,k+1,k)^*L(k+1,q,k)$$

Each of the languages appearing in the right side of the formula is regular because of the induction hypothesis, and $L(p,q,k+1)$ is obtained from them by using the operations of union, concatenation, and Kleene *. Therefore, $L(p,q,k+1)$ is regular.

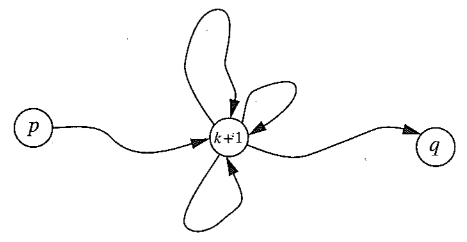

Figure 4.17

The only property of $L(p, q, 0)$ that is needed in the proof of Theorem 4.5 is that it is finite. It is simple enough, however, to find an explicit formula. This formula and the others used in the proof are given below and provide a summary of the steps in the algorithm provided by the proof for finding a regular expression corresponding to a given FA:

$$L(p,q,0) = \begin{cases} \{a \in \Sigma \mid \delta(p,a) = q\} & \text{if } p \neq q \\ \{a \in \Sigma \mid \delta(p,a) = p\} \cup \{\Lambda\} & \text{if } p = q \end{cases}$$

$$L(p,q,k+1) = L(p,q,k) \cup L(p,k+1,k)L(k+1,k+1,k)^*L(k+1,q,k)$$

$$L(p,q) = L(p,q,n)$$

$$L = \bigcup_{q \in A} L(q_0, q)$$

EXAMPLE 4.10 Applying the Algorithm in the Proof of Theorem 4.5

Let $M = (Q, \Sigma, q_0, A, \delta)$ be the FA pictured in Figure 4.18. In carrying out the algorithm above for finding a regular expression corresponding to M, we construct tables showing regular expressions $r(p, q, j)$ corresponding to the languages $L(p, q, j)$ for $0 \le j \le 2$.

p	$r(p,1,0)$	$r(p,2,0)$	$r(p,3,0)$
1	$a+\Lambda$	b	$\emptyset$
2	a	Λ	b
3	a	b	Λ

p	$r(p,1,1)$	$r(p,2,1)$	$r(p,3,1)$
1	a^*	a^*b	$\emptyset$
2	a^+	$\Lambda+a^+b$	b
3	a^+	a^*b	Λ

p	$r(p,1,2)$	$r(p,2,2)$	$r(p,3,2)$
1	$a^*(ba^+)^*$	$a^*(ba^+)^*b$	$a^*(ba^+)^*bb$
2	$a^+(ba^+)^*$	$(a^+b)^*$	$(a^+b)^*b$
3	$a^+ + a^*(ba^+)^+$	$a^*b(a^+b)^*$	$\Lambda + a^*b(a^+b)^*b$

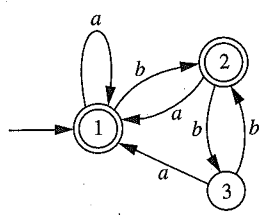

Figure 4.18 |

Although many of these table entries can be obtained by inspection, the formula can be used wherever necessary. In a number of cases, simplifications have been made in the expressions produced by the formula.

For example,

$$\begin{aligned} r(1,3,1) &= r(1,3,0) + r(1,1,0)r(1,1,0)^*r(1,3,0) \\ &= \emptyset \end{aligned}$$

because $r(1,3,0) = \emptyset$, and the concatenation of any language with $\emptyset$ is $\emptyset$.

$$\begin{aligned} r(3,2,1) &= r(3,2,0) + r(3,1,0)r(1,1,0)^*r(1,2,0) \\ &= b + a(a+\Lambda)^*b \\ &= \Lambda b + a^+b \\ &= a^*b \\ r(1,1,2) &= r(1,1,1) + r(1,2,1)r(2,2,1)^*r(2,1,1) \\ &= a^* + a^*b(a^+b)^*a^+ \\ &= a^* + a^*(ba^+)^*ba^+ \\ &= a^* + a^*(ba^+)^+ \\ &= a^*(ba^+)^* \\ r(3,2,2) &= r(3,2,1) + r(3,2,1)r(2,2,1)^*r(2,2,1) \\ &= a^*b + a^*b(a^+b)^*(\Lambda + a^+b) \\ &= a^*b + a^*b(a^+b)^* \\ &= a^*b(a^+b)^* \end{aligned}$$

The two accepting states are 1 and 2, and so the regular expression r we are looking for is $r(1,1,3) + r(1,2,3)$. We use the formula for both, making a few simplifications along the way:

$$\begin{aligned} r(1,1,3) &= r(1,1,2) + r(1,3,2)r(3,3,2)^*r(3,1,2) \\ &= a^*(ba^+)^* + a^*(ba^+)^*bb(\Lambda + a^*b(a^+b)^*b)^*(a^+ + a^*(ba^+)^+) \\ &= a^*(ba^+)^* + a^*(ba^+)^*bb(a^*(ba^+)^*bb)^*(a^+ + a^*(ba^+)^+) \\ &= a^*(ba^+)^* + (a^*(ba^+)^*bb)^+(a^+ + a^*(ba^+)^+) \\ r(1,2,3) &= r(1,2,2) + r(1,3,2)r(3,3,2)^*r(3,2,2) \\ &= a^*(ba^+)^*b + a^*(ba^+)^*bb(a^*b(a^+b)^*b)^*a^*b(a^+b)^* \\ &= a^*(ba^+)^*b + a^*(ba^+)^*bb(a^*(ba^+)^*bb)^*a^*b(a^+b)^* \\ &= a^*(ba^+)^*b + (a^*(ba^+)^*bb)^+a^*(ba^+)^*b \\ &= (a^*(ba^+)^*bb)^*a^*(ba^+)^*b \\ r &= r(1,1,3) + r(1,2,3) \end{aligned}$$

If you wish, you can almost certainly find ways to simplify this further.

EXERCISES

4.1. In the NFA pictured in Figure 4.19, calculate each of the following.

a. $\delta^*(1, bb)$

b. $\delta^*(1, bab)$

c. $\delta^*(1, aabb)$

d. $\delta^*(1, aabbab)$

e. $\delta^*(1, aba)$

4.2. An NFA with states 1–5 and input alphabet $\{a, b\}$ has the following transition table.

q	$\delta(q, a)$	$\delta(q, b)$
1	$\{1, 2\}$	$\{1\}$
2	$\{3\}$	$\{3\}$
3	$\{4\}$	$\{4\}$
4	$\{5\}$	$\emptyset$
5	$\emptyset$	$\{5\}$

a. Draw a transition diagram.

b. Calculate $\delta^*(1, ab)$.

c. Calculate $\delta^*(1, abaab)$.

4.3. Let $M = (Q, \Sigma, q_0, A, \delta)$ be an NFA. Show that for any $q \in Q$ and any $a \in \Sigma$, $\delta^*(q, a) = \delta(q, a)$.

4.4. Suppose $L \subseteq \Sigma^*$ is a regular language. If every FA accepting L has at least n states, then every NFA accepting L has at least ____ states. (Fill in the blank, and explain your answer.)

4.5. In Definition 4.2b, δ^* is defined recursively in an NFA by first defining $\delta^*(q, \Lambda)$ and then defining $\delta^*(q, ya)$, where $y \in \Sigma^*$ and $a \in \Sigma$. Give an

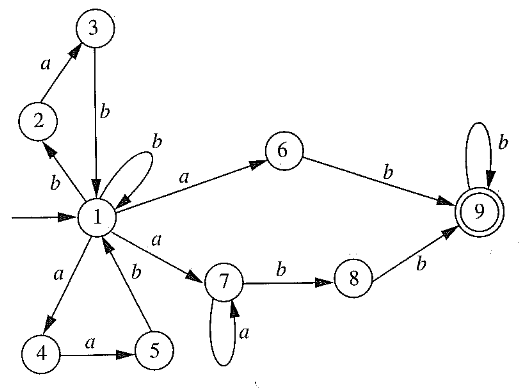

Figure 4.19 |

acceptable recursive definition in which the recursive part of the definition defines $\delta^*(ay)$ instead.

4.6. Give an example of a regular language L containing Λ that cannot be accepted by any NFA having only one accepting state, and show that your answer is correct.

4.7. Can every regular language not containing Λ be accepted by an NFA having only one accepting state? Prove your answer.

4.8. (Refer to Exercise 3.29). Let $M = (Q, \Sigma, q_0, A, \delta)$ be an FA.

a. Show that if every state other than q_0 from which no element of A can be reached is deleted, then what remains is an NFA recognizing the same language.

b. Show that if all states not reachable from q_0 are deleted and all states other than q_0 from which no element of A can be reached are deleted, what remains is an NFA recognizing the same language.

4.9. Let $M = (Q, \Sigma, q_0, A, \delta)$ be an NFA, let m be the maximum size of any of the sets $\delta(q, a)$ for $q \in Q$ and $a \in \Sigma$, and let x be a string of length n over the input alphabet.

a. What is the maximum number of distinct paths that there might be in the computation tree corresponding to x?

b. In order to determine whether x is accepted by M, it is sufficient to replace the complete computation tree by one that is perhaps smaller, obtained by "pruning" the original one so that no level of the tree contains more than $|Q|$ nodes (and no level contains more nodes than there are at that level of the original tree). Explain why this is possible, and how it might be done.

4.10. In each part of Figure 4.20 is pictured an NFA. Using the subset construction, draw an FA accepting the same language. Label the final picture so as to make it clear how it was obtained from the subset construction.

4.11. After the proof of Theorem 3.4, we observed that if $M = (Q, \Sigma, q_0, A, \delta)$ is an FA accepting L, then the FA $M' = (Q, \Sigma, q_0, Q - A, \delta)$ accepts L'. Does this still work if M is an NFA? If so, prove it. If not, explain why, and find a counterexample.

4.12. As in the previous problem, we consider adapting Theorem 3.4 to the case of NFAs. For $i = 1$ and 2, let $M_i = (Q_i, \Sigma, q_i, A_i, \delta_i)$, and let $M = (Q_1 \times Q_2, \Sigma, (q_1, q_2), A, \delta)$, where A is defined as in the theorem for each of the three cases and δ still needs to be defined. If M_1 and M_2 are FAs, the appropriate definition of δ is to use the formula $\delta(p, q) = (\delta_1(p), \delta_2(q))$. If M_1 and M_2 are NFAs, let us define $\delta((p, q), a) = \delta_1(p, a) \times \delta_2(q, a)$. (This says for example that if from state p M_1 can reach either p_1 or p_2 on input a, and from state r M_2 can reach either r_1 or r_2 on input a, then M can reach any of the four states (p_1, r_1), (p_1, r_2), (p_2, r_1), (p_2, r_2) from (p, r) on input a.)

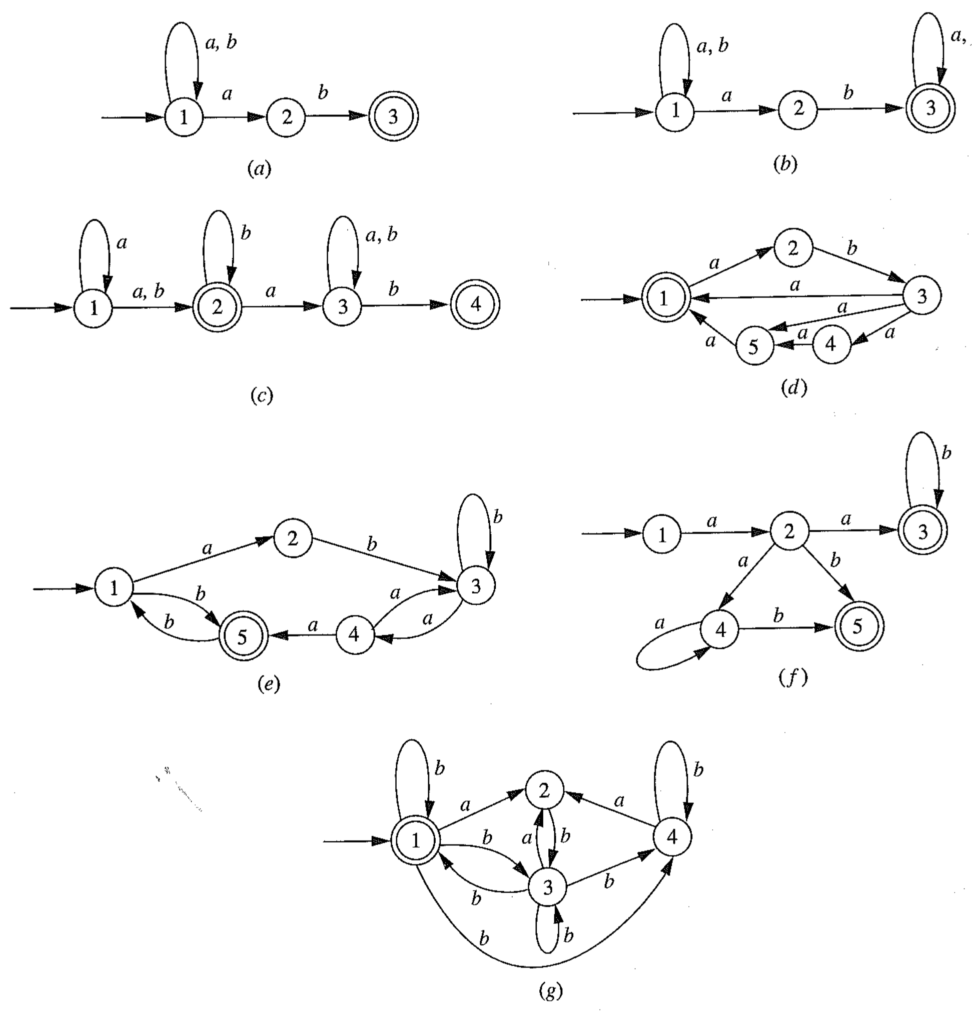

Figure 4.20 |

Do the conclusions of the theorem still hold in this more general situation? Answer in each of the three cases (union, intersection, difference), and give reasons for your answer.

4.13. In Figure 4.21 is a transition diagram of an NFA-Λ. For each string below, say whether the NFA-Λ accepts it:

a. *aba*

b. *abab*

c. *aaabbb*

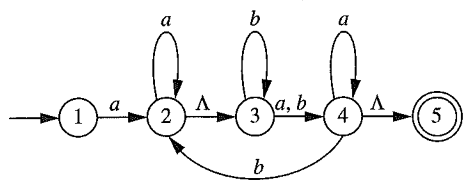

Figure 4.21 |

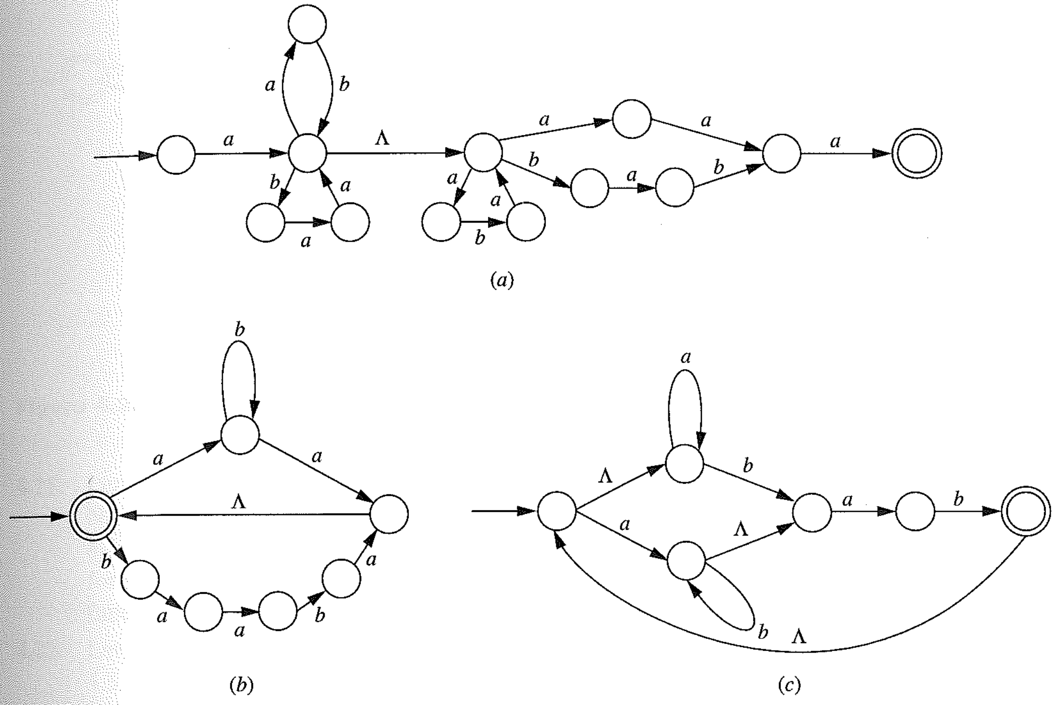

Figure 4.22 |

4.14. Find a regular expression corresponding to the language recognized by the NFA-Λ pictured in Figure 4.21. You should be able to do it without applying Kleene's theorem: First find a regular expression describing the most general way of reaching state 4 the first time, and then find a regular expression describing the most general way, starting in state 4, of moving to state 4 the next time.

4.15. For each of the NFA-Λs shown in Figure 4.22, find a regular expression corresponding to the language it recognizes.

4.16. A transition table is given for an NFA-Λ with seven states.

q	$\delta(q,a)$	$\delta(q,b)$	$\delta(q,\Lambda)$
1	$\emptyset$	$\emptyset$	$\{2\}$
2	$\{3\}$	$\emptyset$	$\{5\}$
3	$\emptyset$	$\{4\}$	$\emptyset$
4	$\{4\}$	$\emptyset$	$\{1\}$
5	$\emptyset$	$\{6,7\}$	$\emptyset$
6	$\{5\}$	$\emptyset$	$\emptyset$
7	$\emptyset$	$\emptyset$	$\{1\}$

Find:

a. $\Lambda(\{2,3\})$
b. $\Lambda(\{1\})$
c. $\Lambda(\{3,4\})$
d. $\delta^*(1, ba)$
e. $\delta^*(1, ab)$
f. $\delta^*(1, ababa)$

4.17. A transition table is given for another NFA-Λ with seven states.

q	$\delta(q,a)$	$\delta(q,b)$	$\delta(q,\Lambda)$
1	$\{5\}$	$\emptyset$	$\{4\}$
2	$\{1\}$	$\emptyset$	$\emptyset$
3	$\emptyset$	$\{2\}$	$\emptyset$
4	$\emptyset$	$\{7\}$	$\{3\}$
5	$\emptyset$	$\emptyset$	$\{1\}$
6	$\emptyset$	$\{5\}$	$\{4\}$
7	$\{6\}$	$\emptyset$	$\emptyset$

Calculate $\delta^*(1, ba)$.

4.18. For each of these regular expressions over $\{0, 1\}$, draw an NFA-Λ recognizing the corresponding language. (You should not need the construction in Kleene's theorem to do this.)

a. $(0+1)^*(011+01010)(0+1)^*$
b. $(0+1)(01)^*(011)^*$
c. $010^*+0(01+10)^*11$

4.19. Suppose M is an NFA-Λ accepting $L \subseteq \Sigma^*$. Describe how to modify M to obtain an NFA-Λ recognizing $rev(L) = \{x^r \mid x \in L\}$.

4.20. In each part of Figure 4.23, two NFA-Λs are illustrated. Decide whether the two accept the same language, and give reasons for your answer.

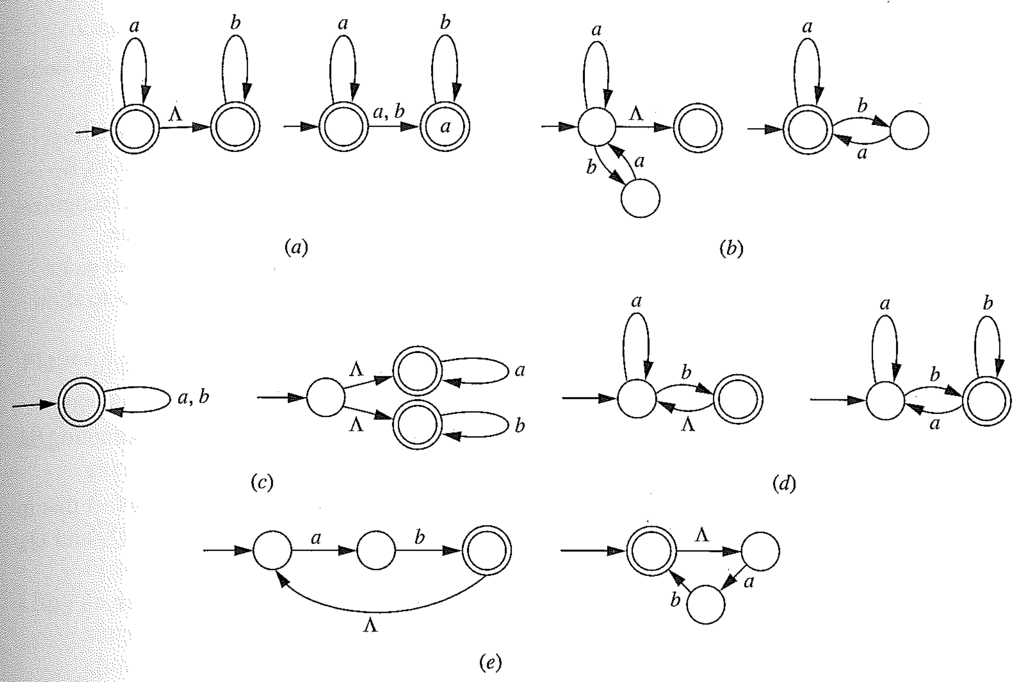

Figure 4.23 |

4.21. Let $M = (Q, \Sigma, q_0, A, \delta)$ be an FA, and let $M_1 = (Q, \Sigma, q_0, A, \delta_1)$ be the NFA-Λ defined in the proof of Theorem 4.3, in which $\delta_1(q, \Lambda) = \emptyset$ and $\delta_1(q, a) = \{\delta(q, a)\}$, for every $q \in Q$ and $a \in \Sigma$. Give a careful proof that for every $q \in Q$ and $x \in \Sigma^*$, $\delta_1^*(q, x) = \{\delta^*(q, x)\}$. Recall that the two functions δ^* and δ_1^* are defined differently.

4.22. Let M_1 be the NFA-Λ obtained from the FA M as in the proof of Theorem 4.3. The transition function δ_1 of M_1 is defined so that $\delta_1(q, \Lambda) = \emptyset$ for every state q. Would defining $\delta_1(q, \Lambda) = \{q\}$ also work? Give reasons for your answer.

4.23. Let $M = (Q, \Sigma, q_0, A, \delta)$ be an NFA-Λ. The proofs of Theorems 4.2 and 4.1 describe a two-step process for obtaining an FA $M_1 = (Q_1, \Sigma, q_1, A_1, \delta_1)$ that accepts the language $L(M)$. Do it in one step, by defining Q_1, q_1, A_1, and δ_1 directly in terms of M.

4.24. Let $M = (Q, \Sigma, q_0, A, \delta)$ be an NFA-Λ. In the proof of Theorem 4.2, the NFA M_1 might have more accepting states than M: The initial state q_0 is made an accepting state if $\Lambda(\{q_0\}) \cap A \neq \emptyset$. Explain why it is not necessary to make *all* the states q for which $\Lambda(\{q\}) \cap A \neq \emptyset$ accepting states in M_1.

4.25. Suppose $M = (Q, \Sigma, q_0, A, \delta)$ is an NFA-Λ recognizing a language L. Let M_1 be the NFA-Λ obtained from M by adding Λ-transitions from each element of A to q_0. Describe (in terms of L) the language $L(M_1)$.

4.26. Suppose $M = (Q, \Sigma, q_0, A, \delta)$ is an NFA-Λ recognizing a language L.

a. Describe how to construct an NFA-Λ M_1 with no transitions to its initial state so that M_1 also recognizes L.

b. Describe how to construct an NFA-Λ M_2 with exactly one accepting state and no transitions from that state so that M_2 also recognizes L.

4.27. Suppose M is an NFA-Λ with exactly one accepting state q_f that recognizes the language $L \subseteq \{0, 1\}^*$. In order to find NFA-Λs recognizing the languages $\{0\}^*L$ and $L\{0\}^*$, we might try adding 0-transitions from q_0 to itself and from q_f to itself, respectively. Draw transition diagrams to show that neither technique always works.

4.28. In each part of Figure 4.24 is pictured an NFA-Λ. Use the algorithm illustrated in Example 4.7 to draw an NFA accepting the same language.

4.29. In each part of Figure 4.25 is pictured an NFA-Λ. Draw an FA accepting the same language.

4.30. Give an example (i.e., draw a transition diagram) to illustrate the fact that in the construction of M_u in the proof of Theorem 4.4, the two sets Q_1 and Q_2 must be disjoint.

4.31. Give an example to illustrate the fact that in the construction of M_c in the proof of Theorem 4.4, the two sets Q_1 and Q_2 must be disjoint.

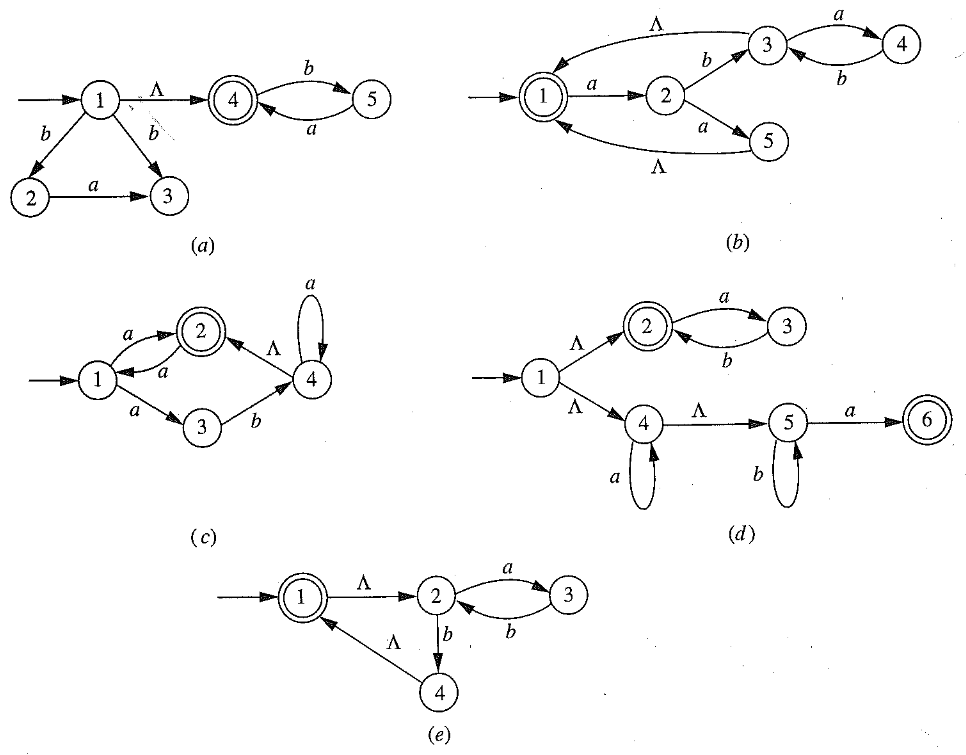

Figure 4.24 |

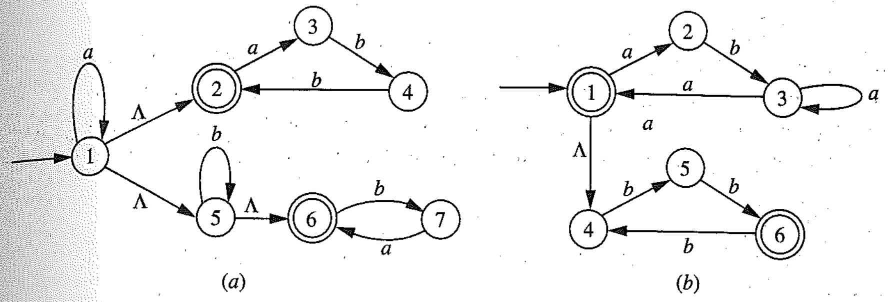

Figure 4.25 |

4.32. In the construction of M_u in the proof of Theorem 4.4, consider this alternative to the construction described: Instead of a new state q_u and Λ-transitions from it to q_1 and q_2, make q_1 the initial state of the new NFA-Λ, and create a Λ-transition from it to q_2. Either prove that this works in general, or give an example in which it fails.

4.33. In the construction of M_c in the proof of Theorem 4.4, consider the simplified case in which M_1 has only one accepting state. Suppose that we eliminate the Λ-transition from the accepting state of M_1 to q_2, and merge these two states into one. Either show that this would always work in this case, or give an example in which it fails.

4.34. In the construction of M_k in the proof of Theorem 4.4, suppose that instead of adding a new state q_k, with Λ-transitions from it to q_1 and to it from each accepting state of Q_1, we make q_1 both the initial state and the accepting state, and create Λ-transitions from each accepting state of M_1 to q_k. Either show that this works in general, or give an example in which it fails.

4.35. In each case below, find an NFA-Λ recognizing the language corresponding to the regular expression, by applying literally the algorithm in the chapter. Do not attempt to simplify the answer.

a. $((ab)^*b + ab^*)^*$

b. $aa(ba)^* + b^*aba^*$

c. $(ab + (aab)^*)(aa + a)$

4.36. In Figures 4.26*a* and 4.26*b* are pictured FAs M_1 and M_2, recognizing languages L_1 and L_2, respectively. Draw NFA-Λs recognizing each of the following languages, using the constructions in this chapter:

a. L_1L_2

b. $L_1L_1L_2$

c. $L_1 \cup L_2$

d. L_1^*

e. $L_2^* \cup L_1$

f. $L_2L_1^*$

g. $L_1L_2 \cup (L_2L_1)^*$

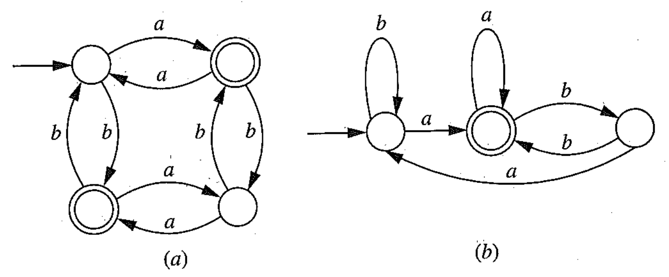

Figure 4.26 |

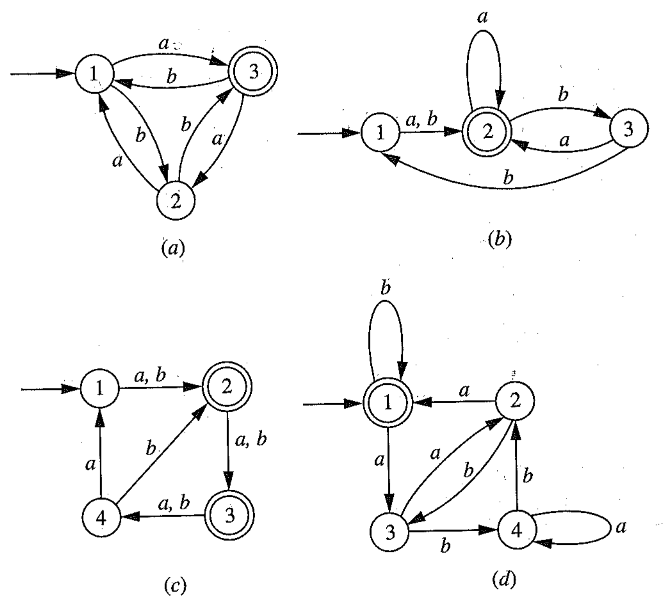

Figure 4.27 |

4.37. Draw NFAs accepting L_1L_2 and L_2L_1, where L_1 and L_2 are as in the preceding problem. Do this by connecting the two given diagrams directly, by arrows with appropriate labels.

4.38. Use the algorithm of Theorem 4.5 to find a regular expression corresponding to each of the FAs shown in Figure 4.27. In each case, if the FA has n states, construct tables showing $L(p, q, j)$ for each j with $0 \le j \le n-1$.

MORE CHALLENGING PROBLEMS

4.39. Which of the following, if any, would be a correct substitute for the second part of Definition 4.5b? Give reasons for your answer.

a. $\delta^*(q, ay) = \Lambda\left(\bigcup_{r \in \delta(q,a)} \delta^*(r, y)\right)$

b. $\delta^*(q, ay) = \bigcup_{r \in \delta(q,a)} \Lambda(\delta^*(r, y))$

c. $\delta^*(q, ay) = \bigcup_{r \in \Lambda(\delta(q,a))} \delta^*(r, y)$

d. $\delta^*(q, ay) = \bigcup_{r \in \Lambda(\delta(q,a))} \Lambda(\delta^*(r, y))$

4.40. Let $M = (Q, \Sigma, q_0, A, \delta)$ be an NFA-Λ. This exercise involves properties of the Λ-closure of a set S. Since $\Lambda(S)$ is defined recursively (Definition 4.6), structural induction can be used to show that every state in $\Lambda(S)$ satisfies some property—such as being a member of some other set.

a. Show that if S and T are subsets of Q for which $S \subseteq T$, then $\Lambda(S) \subseteq \Lambda(T)$.

b. Show that for any $S \subseteq Q$, $\Lambda(\Lambda(S)) = \Lambda(S)$.

c. Show that if $S, T \subseteq Q$, then $\Lambda(S \cup T) = \Lambda(S) \cup \Lambda(T)$.

d. Show that if $S \subseteq Q$, then $\Lambda(S) = \bigcup_{p \in S} \Lambda(\{p\})$.

e. Draw a transition diagram to illustrate the fact that $\Lambda(S \cap T)$ and $\Lambda(S) \cap \Lambda(T)$ are not always the same. Which is always a subset of the other?

f. Draw a transition diagram illustrating the fact that $\Lambda(S')$ and $\Lambda(S)'$ are not always the same. Which is always a subset of the other? Under what circumstances are they equal?

4.41. Let $M = (Q, \Sigma, q_0, A, \delta)$ be an NFA-Λ. A set $S \subseteq Q$ is called Λ-*closed* if $\Lambda(S) = S$.

a. Show that the union of two Λ-closed sets is Λ-closed.

b. Show that the intersection of two Λ-closed sets is Λ-closed.

c. Show that for any subset S of Q, $\Lambda(S)$ is the smallest Λ-closed set of which S is a subset.

4.42. a. Let $M = (Q, \Sigma, q_0, A, \delta)$ be an NFA. Show that for every $q \in Q$ and every $x, y \in \Sigma^*$,

$$\delta^*(q, xy) = \bigcup_{r \in \delta^*(q,x)} \delta^*(r, y)$$

b. Prove the same formula, this time assuming that M is an NFA-Λ.

4.43. Suppose $M = (Q, \Sigma, q_0, A, \delta)$ is an NFA. We may consider a new NFA $M_1 = (Q, \Sigma, q_0, A, \delta_1)$ obtained by letting $\delta_1(q, a) = \delta(q, a)$ if $|\delta(q, a)| \le 1$ and, for every pair (q, a) for which $|\delta(q, a)| > 1$, choosing one state $p \in \delta(q, a)$ arbitrarily and letting $\delta_1(q, a) = \{p\}$. Although M_1 is not necessarily an FA, it could easily be converted to an FA, because it never has more than one choice of moves.

If S is the set of all pairs (q, a) for which $|\delta(q, a)| > 1$, and c denotes some specific sequence of choices, one choice for each element of S, then we let M_1^c denote the specific NFA that results. What is the relationship between $L(M)$ and the collection of languages $L(M_1^c)$ obtained by considering all possible sequences c of choices? Be as precise as you can, and give reasons for your answer.

4.44. Let $M = (Q, \Sigma, q_0, A, \delta)$ be an NFA-Λ recognizing a language L. Assume that there are no transitions to q_0, that A has only one element, q_f, and that there are no transitions from q_f.

a. Let M_1 be obtained from M by adding Λ-transitions from q_0 to every state that is reachable from q_0 in M. (If p and q are states, q is reachable from p if there is a string $x \in \Sigma^*$ such that $q \in \delta^*(p, x)$.) Describe (in terms of L) the language accepted by M_1.

b. Let M_2 be obtained from M by adding Λ-transitions to q_f from every state from which q_f is reachable in M. Describe in terms of L the language accepted by M_2.

c. Let M_3 be obtained from M by adding both the Λ-transitions in (a) and those in (b). Describe the language accepted by M_3.

d. Let M_4 be obtained from M by adding Λ-transitions from p to q whenever q is reachable from p in M. Describe the language accepted by M_4.

4.45. In Example 4.5, we started with NFAs M_1 and M_2 and incorporated them into a composite NFA M accepting $L(M_1) \cup L(M_2)$ in such a way that no new states were required.

a. Show by considering the languages $\{0\}^*$ and $\{1\}^*$ that this is not always possible. (Each of the two languages can obviously be accepted by a one-state NFA; show that their union cannot be accepted by a two-state NFA.)

b. Describe a reasonably general set of circumstances in which this *is* possible. (Find a condition that might be satisfied by one or both of the NFAs that would make it possible.)

4.46. Suppose Σ_1 and Σ_2 are alphabets, and the function $f : \Sigma_1^* \to \Sigma_2^*$ is a *homomorphism*; i.e., $f(xy) = f(x)f(y)$ for every $x, y \in \Sigma_1^*$.

a. Show that $f(\Lambda) = \Lambda$.

b. Show that if $L \subseteq \Sigma_1^*$ is regular, then $f(L)$ is regular. ($f(L)$ is the set $\{y \in \Sigma_2^* \mid y = f(x) \text{ for some } x \in L\}$.)

c. Show that if $L \subseteq \Sigma_2^*$ is regular, then $f^{-1}(L)$ is regular. ($f^{-1}(L)$ is the set $\{x \in \Sigma_1^* \mid f(x) \in L\}$.)

4.47. Suppose $M = (Q, \Sigma, q_0, A, \delta)$ is an NFA-Λ. For two (not necessarily distinct) states p and q, we define the regular expression $e(p, q)$ as follows: $e(p, q) = l + r_1 + r_2 + \cdots + r_k$, where l is either Λ (if $\delta(p, \Lambda)$ contains q) or $\emptyset$, and the r_i's are all the elements a of Σ for which $\delta(p, a)$ contains q. It's possible for $e(p, q)$ to be $\emptyset$, if there are no transitions from p to q; otherwise, $e(p, q)$ represents the "most general" transition from p to q.

If we generalize this by allowing $e(p, q)$ to be an arbitrary regular expression over Σ, we get what is called an *expression graph*. If p and q are two states in an expression graph G, and $x \in \Sigma^*$, we say that x allows G to move from p to q if there are states $p_0, p_1, \ldots, p_m$, with $p_0 = p$ and $p_m = q$, so that x corresponds to the regular expression $e(p_0, p_1)e(p_1, p_2)\cdots e(p_{n-1}, p_n)$. This allows us to say how G accepts a string x (x allows G to move from the initial state to an accepting state), and therefore to talk about the language accepted by G. It is easy to see that in the special case where G is simply an NFA-Λ, the two definitions for the language accepted by G coincide. (See Definition 4.5a.) It is also not hard to convince yourself, using Theorem 4.4, that for any expression graph G, the language accepted by G can be accepted by an NFA-Λ.

We can use the idea of an expression graph to obtain an alternate proof of Theorem 4.5, as follows. Starting with an FA M accepting L, we may easily convert it to an NFA-Λ M_1 accepting L, so that M_1 has no transitions to its initial state q_0, exactly one accepting state q_f (which is different from q_0), and no transitions from q_f. The remainder of the proof is to specify a reduction technique to reduce by one the number of states other than q_0 and q_f, obtaining an equivalent expression graph at each step, until q_0 and q_f are the only states remaining. The regular expression $e(q_0, q_f)$ then describes the language accepted. If p is the state to be eliminated, the reduction step involves redefining $e(q, r)$ for every pair of states q and r other than p.

Describe in more detail how this reduction can be done. Then apply this technique to the FAs in Figure 4.27 to obtain regular expressions corresponding to their languages.