2

Finite Automata

2.1 Deterministic Finite Automata

In this section, we introduce a simple machine model, called deterministic finite automata, that will be shown later to recognize exactly the class of regular languages. Intuitively, a deterministic finite automaton (DFA) is a simple machine which reads an input string one letter at a time and then, after the input is completely read, decides to accept or to reject the input. A DFA consists of three parts: a tape, a tape head (or, simply, head), and a finite control.

The tape is used to store the input data. It is divided into a finite number of cells. Each cell holds a symbol from a given alphabet $\Sigma$. The tape head scans the tape, reads symbols from the tape, and passes the information to the finite control. At each move of the DFA, the head scans one cell of the tape and reads the symbol in the cell, and then moves to the next cell to the right.

The finite control has a finite number of states which form the state set $Q$. At the beginning of a move, the control is in one of the states. Then, it determines, from the current state and the symbol read by the tape head, how the state is changed to a new state. More precisely, the change of state at each move is governed by a state transition function (or, simply, transition function) $\delta: Q \times \Sigma \to Q$. At each move, if the finite control is currently at state $q \in Q$ and the symbol read by the tape head is $a \in \Sigma$, then the finite control changes to the new state $p = \delta(q, a)$.

To complete the description of a DFA, we need to specify two special kinds
of states: an initial state $q_0$ at which the DFA begins to work, and a set $F \subseteq Q$ of final states at which the DFA certifies that the input is in the language. Thus, a DFA can be represented as a quintuple $M = (Q, \Sigma, \delta, q_0, F)$.

How does a DFA $M$ work exactly? Initially, the DFA $M$ is in the initial state, the tape holds the input string, and the tape head scans the leftmost cell of the tape. Then, the DFA $M$ operates, move by move, according to the transition function $\delta$. The DFA $M$ halts after it reads the symbol in the rightmost cell of the tape and its tape head moves off the tape. When the DFA halts, it accepts the input string if it halts in one of the final states. Otherwise, the input string is rejected. The set of all strings accepted by a DFA $M$ is denoted by $L(M)$. We also say that the language $L(M)$ is accepted by $M$.

The transition diagram of a DFA is an alternative way to represent the DFA. For $M = (Q, \Sigma, \delta, q_0, F)$, the transition diagram of $M$ is a labeled digraph $(V, E)$ satisfying

$$V = Q,$$

$$E = \{q \xrightarrow{a} p \mid p = \delta(q, a)\},$$

where $q \xrightarrow{a} p$ denotes an edge $(q, p)$ with label $a$. In addition, the initial state of the DFA is pointed to by an arrow without a starting vertex, and every final state is denoted by double circles.

**Example 2.1** Consider a DFA $M = (Q, \Sigma, \delta, q_0, F)$ where $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{0, 1\}$, $F = \{q_1, q_2\}$, and $\delta$ is the function defined by the following table:

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$q_1$</td>
<td>$q_3$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$q_2$</td>
<td>$q_3$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$q_2$</td>
<td>$q_2$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$q_3$</td>
<td>$q_3$</td>
</tr>
</tbody>
</table>

Then, the transition diagram of $M$ is as shown in Figure 2.2. (A number $i$ in a circle or in double circles denotes the state $q_i$.)
2.1 Deterministic Finite Automata

![Transition Diagram]

Figure 2.2: The transition diagram of the DFA of Example 2.1.

Since the transition function $\delta$ is well-defined on $Q \times \Sigma$, the transition diagram of the DFA has the property that for every vertex (state) $q$ and every symbol $a$, there exists exactly one edge with label $a$ leaving $q$. This implies that for each string $x$, there exists exactly one path starting from $q_0$ whose labels form the string $x$. This path is called the computation path of the DFA on $x$. We note that a string $x$ is accepted by $M$ if and only if its computation path ends at one of the final states.

To formally define the notion of a DFA accepting an input string, we extend the transition function $\delta$ from the domain $Q \times \Sigma$ to the domain $Q \times \Sigma^*$ by the following inductive definition:

1. $\delta(q, \varepsilon) = q$.
2. For any $x \in \Sigma^*$ and $a \in \Sigma$, $\delta(q, xa) = \delta(\delta(q, x), a)$.

With this extension, for any string $x$, $\delta(q_0, x)$ is the last state in the computation path of the string $x$. Therefore, we can formally define that $x \in L(M)$ if and only if $\delta(q_0, x) \in F$; that is,

$$L(M) = \{ x \in \Sigma^* \mid \delta(q_0, x) \in F \}.$$

Example 2.2 Consider the DFA $M$ given by Figure 2.2. Determine whether $M$ accepts strings 000 and 010.

Solution. The computation path of the string 000 is the sequence $q_0, \delta(q_0, 0) = q_1, \delta(q_1, 0) = q_2$, and $\delta(q_2, 0) = q_3$. Thus, $\delta(q_0, 000) = q_3 \in F$ and $M$ accepts 000. Similarly, we can find that $\delta(q_0, 010) = q_3 \notin F$, and $M$ rejects 010. □

Example 2.3 What is the language $L(M)$ accepted by the DFA $M$ of Figure 2.2?

Solution. We observe that once the DFA $M$ enters state $q_3$, it will never be able to leave this state, and will reject the input string regardless what the remaining part of the input string is. (We call such a state a failure state.) On the other hand, once $M$ enters state $q_2$, it will never leave this state, and
will accept the input string. (We call such a state a success state.) From these
observations, we see that \( L(M) \) consists of the string 0 (which ends at state
\( q_1 \)) and all strings starting with 00. In the regular expression notation,
\[
L(M) = 0 + 00(0 + 1)^*.
\]
\( \square \)

**Example 2.4** What is the language \( L(M) \) accepted by the DFA \( M \) of Figure
2.3?

**Solution.** Similar to Example 2.3, \( q_0 \) is a failure state and \( q_3 \) is a success state.
So, we only need to figure out how we can reach \( q_3 \) from \( q_0 \). There are three
kinds of paths from \( q_0 \) to \( q_3 \):

\[
(q_0, q_1, q_2, \ldots, q_2, q_3),
(q_0, q_1, q_5, \ldots, q_5, q_3),
(q_0, q_4, q_5, \ldots, q_5, q_3).
\]

They correspond with strings beginning with 011*0, 000*1 and 100*1, respectively. So,
\[
L(M) = (011^*0 + 000^*1 + 100^*1)(0 + 1)^*.
\]
\( \square \)

**Example 2.5** What is the language \( L(M) \) accepted by the DFA \( M \) of Figure
2.4?

**Solution.** A string \( x \) accepted by \( M \) is either in \( 0^* \) (so, \( M \) never leaves state
\( q_0 \)), or is associated with a computation cycle from \( q_0 \) to \( q_0 \), passing through
states \( q_1, q_2, q_3, q_4 \) for a finite number of times. Note that the DFA \( M \) changes
from a state \( q_i, 0 \leq i \leq 3 \), to a new state \( q_{i+1} \) (or from \( q_4 \) to \( q_0 \)) if and only if
it reads a symbol 1. Thus, each time \( M \) goes from state \( q_0 \) to states \( q_1, \ldots, q_4 \)
then comes back to \( q_0 \), it must have read five 1's, with an arbitrary number of 0's in between any two 1's. This means that a string \( x \) is accepted by \( M \) if and only if the number of occurrences of symbol 1 in \( x \) is a multiple of 5. In the form of a regular expression,

\[
L(M) = 0^* (10^* 10^* 10^* 10^*)^*.
\]

\[\Box\]

**Exercise 2.1**

1. Consider the DFA \( M \) with the transition diagram of Figure 2.5.

    (a) Which state is the initial state of \( M \), and which states are the final states of \( M \)?

    (b) For each of the strings 0001, 010101, and 00110101101, find the computation path of \( M \) on the string and determine whether \( M \) accepts it.

    (c) Among all strings in \((01)^*\), which ones are in \( L(M) \)?

2. For integers \( n, d \geq 1 \), consider a DFA \( M_{n,d} = (Q, \Sigma, \delta, q_0, F) \), where \( Q = \{ q_0, q_1, q_2, \cdots, q_{n-1} \} \), \( \Sigma = \{ a_0, a_1, \cdots, a_{d-1} \} \), \( \delta(q_i, a_k) = q_{(d \cdot i + k) \mod n} \), and \( F = \{ q_1 \} \).

    (a) Draw the transition diagram of \( M_{n,d} \) with \( n = 7 \) and \( d = 2 \).
(b) Suppose $n = 7$, $d = 2$, $a_0 = 0$ and $a_1 = 1$. Find $\delta(q_3,0101)$ and $\delta(q_1,11010)$.

(c) Suppose $n = 7$, $d = 2$, $a_0 = 0$ and $a_1 = 1$. Find binary strings $x$ and $y$ such that $\delta(q_3,x) = q_5$ and $\delta(q_3,y) = q_6$.

*(d)* Show that for any state $q_j \in Q$, there exists a string $x \in \Sigma^*$ such that $\delta(q_3,x) = q_j$.

3. For each of the DFA’s $M_1$, $M_2$ and $M_3$, as shown in Figure 2.6(a), (b) and (c), respectively, describe in English the language accepted by it.

### 2.2 Examples of Deterministic Finite Automata

In this section, we will develop some techniques to construct DFA’s through examples. In each example, a language is given and we show how to construct a DFA accepting the language.

**Example 2.6** $(0 + 1)^*$.

*Solution.* The language $(0 + 1)^*$ contains all binary strings. So, we simply let the initial state be the final success state. The transition diagram of this DFA is actually the same as the labeled digraph representation for the regular expression $(0 + 1)^*$. We show it in Figure 2.7(a). In a transition diagram, if two edges have the same starting and ending vertices, we may merge them into
one edge and put both labels on the new edge. For instance, the transition
diagram of Figure 2.7(a) can be simplified to that of Figure 2.7(b). □

Example 2.7 The set of all binary strings beginning with prefix 01.

Solution. It is easy to find a regular expression $01(0 + 1)^*$ for this language. From the regular expression, we can immediately find its labeled digraph representation as shown in Figure 2.8(a). However, it is not the transition diagram of a DFA. Recall that in the transition diagram of a DFA $M = (Q, \Sigma, \delta, q_0, F)$, there is exactly one out-edge from state $q$ with label $a$ for each pair of $(q, a) \in Q \times \Sigma$. To satisfy this condition, we add a failure state $q_3$ and edges $q_0 \overset{0}{\to} q_3$ and $q_1 \overset{0}{\to} q_3$. The complete transition diagram is shown in Figure 2.8(b). □

Example 2.8 The set of all binary strings having a substring 00.

Solution. First, let us note that the regular expression $(0 + 1)^*00(0 + 1)^*$ for this set does not indicate where the first pair 00 occurs in the string. On the other hand, the DFA must recognize the substring 00 at its first occurrence. Thus, the above regular expression is not helpful to this problem. Instead, we need to analyze the problem more carefully.
Suppose that the string $x_1 x_2 \cdots x_n$ is stored on the tape, where each $x_i$ denotes one bit in $\{0, 1\}$. Then, we may check each of the substrings $x_1 x_2$, $x_2 x_3$, $\ldots$, $x_{n-1} x_n$ in turn, to see whether it is equal to 00, and accept the input string as soon as a substring 00 is found. This suggests the following way to construct the required DFA.

**Step 1.** Build a checker as shown in Figure 2.9(a), with state $q_2$ being a success state, meaning that if two consecutive 0's are found then the input string is to be accepted. In particular, if $x_1 x_2 = 00$, the the string $x$ is accepted.

**Step 2.** If $x_1 = 1$, then we give up on substring $x_1 x_2$ and continue to check $x_2 x_3$. So, we need to go back to state $q_0$; that is, $\delta(q_0, 1) = q_0$. This action is shown in Figure 2.9(b).

**Step 3.** If $x_1 = 0$ and $x_2 = 1$, then neither $x_1 x_2$ nor $x_2 x_3$ is 00. So, we also need to go back to restart at $q_0$ to check $x_3 x_4$. That is, we let $\delta(q_1, 1) = q_0$.

Figure 2.9(c) shows the complete DFA. \( \square \)

In general, suppose that $x_1 x_2 \cdots x_n$ is the string on the tape and we want to check $x_1 \cdots x_k$, $x_2 \cdots x_{k+1}$, $\ldots$, $x_{n-k} \cdots x_n$ in turn to match a substring $a_1 a_2 \cdots a_k$. Then, we set up $k + 1$ states $q_0, q_1, \ldots, q_k$, with $q_i$ standing for “found $a_1 a_2 \cdots a_i$.” At state $q_i$, if $b = a_{i+1}$, then we define $\delta(q_i, b) = q_j$. If $b \neq a_{i+1}$, then we define $\delta(q_i, b) = q_j$, where $j$ is the maximum index $j$ such that

$$a_{i-j+3} a_{i-j+2} a_{i-j+1} a_{i-j} b = a_1 a_2 \cdots a_j.$$

That is, we find the longest suffix $y$ of $a_1 \cdots a_k b$ which is a prefix of $a_1 \cdots a_k$ and go to $q_{b\delta}$. The following example explains this idea more clearly.

**Example 2.9** The set of all binary strings having the substring 00101.

**Solution.** Following the above idea, we first construct a checker as shown in Figure 2.10(a).
2.2 Examples of DFA’s

![DFA Diagrams]

**Figure 2.10:** Solution to Example 2.9.

Intuitively, for each \( i = 0, 1, \ldots, 5 \), state \( q_i \) means “the past \( i \) input symbols just read are \( a_1 a_2 \cdots a_i \),” where \( a_1 a_2 \cdots a_5 \) is the target substring 00101. Thus, at state \( q_0 \), if we read a new symbol 1, the new string \( a_1 \cdots a_{i+1} = 1 \) (here, \( i = 0 \)) is not a prefix of 00101, and so we need to go back to state \( q_0 \). Similarly, if a symbol 1 is read at state \( q_1 \), neither the string \( a_1 1 = 11 \) nor the string 1 is a prefix of 00101, and so we let \( \delta(q_1, 1) = q_0 \). We upgrade the DFA as shown in Figure 2.10(b).

Now, consider \( \delta(q_2, 0) \). The string \( a_1 a_2 0 = 000 \) is not a prefix of 00101, but the last two symbols \( a_2 0 = 00 \) is a prefix of 00101. That is, if the next three input symbols are 1, 0 and 1, then we should accept the input. So, we define \( \delta(q_2, 0) = q_2 \) to indicate this partial success. This action is shown in Figure 2.10(c).

Based on the same idea, we define \( \delta(q_3, 1) = q_0 \) (neither \( a_1 a_2 a_3 1 = 0011 \) nor any of its suffixes is a prefix of 00101), and \( \delta(q_4, 0) = q_2 \) (the last two bits of \( a_1 a_2 a_3 a_4 0 \) are 00 and form a prefix of 00101). The complete DFA is shown in Figure 2.10(d).

**Example 2.10** The set \( A \) of all binary strings ending with 01.

**Solution.** Initially, we build a checker as shown in Figure 2.11(a). Again, states \( q_0 \) and \( q_1 \) indicate “found no prefix of 01” and “found prefix 0 of 01,” respectively. Note, however, that although state \( q_2 \) is a final state, it is not a success state since a string must end at state \( q_2 \) to be accepted.

Now, following the idea of the last example, it is easy to see that we need to define \( \delta(q_0, 1) = q_0 \) and \( \delta(q_1, 0) = q_1 \).
At state $q_2$, if we read more input symbols, then we need to follow the same idea to set $\delta(q_2, 0) = q_1$ and $\delta(q_2, 1) = q_0$. The complete DFA is shown in Figure 2.11(b).

Example 2.11 The set of all binary expansions of positive integers which are congruent to zero modulo 5.

Solution. The idea of this DFA is similar to that of the last two examples. We need to set up five states $q_0, q_1, \ldots, q_4$, with each state $q_i$ meaning “the prefix $y$ of the input string read so far has the property of $y \equiv i \pmod{5}.”$ That is, we need to define $\delta(q_i, \ldots x_k) = q_i$ if $x_1 x_2 \cdots x_k \equiv i \pmod{5}$.

How do we determine the edges between these five states from this idea? Recall that the transition function $\delta$ of a DFA has to satisfy

$$\delta(\delta(q_0, x), a) = \delta(q_0, xa),$$

for any $x \in \{0, 1\}^*$ and any $a \in \{0, 1\}$. Suppose $\delta(q_0, x) = q_i$ and $\delta(q_0, xa) = q_j$. Then, we must have $x \equiv i \pmod{5}$ and $xa \equiv j \pmod{5}$. Thus,

$$j \equiv xa \pmod{5}$$

$$\equiv 2 \cdot x + a \pmod{5}$$

$$\equiv 2 \cdot i + a \pmod{5}.$$

Therefore, we need to define $\delta(q_i, a) = q_j$, where $j \equiv 2i + a \pmod{5}$. For instance, $\delta(q_2, 0) = q_4$ and $\delta(q_2, 1) = q_0$. See Figure 2.12 for the other edges. In addition, state $q_0$ is the unique final state, since $\delta(q_0, x) = q_0$ means $x \equiv 0 \pmod{5}$.

Finally, we note that a binary expansion of a positive integer always begins with the symbol 1. So, we need to add a new initial state and a failure state, as shown in Figure 2.12.

Note that the set $A \cup \{1\}$, where $A$ is the set defined in Example 2.10, may be regarded as the set of binary expansions of integers congruent to 1 modulo
4, with leading zeros allowed. Using the idea of the above example, we get a new DFA for set \( A \cup \{1\} \) as shown in Figure 2.13. Note that the states \( q_0 \) and \( q_2 \) in this DFA can be merged into one, and the simplified DFA is just the one shown in Figure 2.11(b), except that the initial state has been changed (to state \( q_1 \) of Figure 2.11(b)).

Example 2.12 The set of all binary strings having a substring 00 or ending with 01.

Solution. This language is the union of two languages \((0 + 1)^*00(0 + 1)^*\) and \((0 + 1)^*01\). In Examples 2.8 and 2.10, we have already constructed two DFA’s for these two languages. To check whether an input string \( x \) belongs to the union of these two languages, we can run these two DFA’s in parallel. For example, suppose \( x = 0101\). In the first DFA \( M_1 \) shown in Figure 2.9(c), the computation path of \( x \) is \((q_3, q_1, q_4, q_1, q_0)\), and in the second DFA \( M_2 \) in Figure 2.11(b), the computation path is \((q_0, q_1, q_2, q_1, q_2)\). Since the second path ends at a final state, \( x \) is accepted in this parallel simulation.

One idea on how to build a DFA for the union of these two languages is, then, to combine the two DFA’s into one such that, at each step, the new DFA would keep track of the computation paths of both DFA’s. This suggests us to consider a product automaton \( M = M_1 \times M_2 \) as follows: Assume that the first DFA is \( M_1 = (Q_1, \Sigma, \delta_1, q_0, F_1) \) and the second DFA is \( M_2 = (Q_2, \Sigma, \delta_2, q_0, F_2) \). (Note that \( Q_1 \) and \( Q_2 \) may have states of the same name but playing different roles in two DFA’s.) Then, the state set \( Q \) of \( M \) is the cross product of the state sets of \( M_1 \) and \( M_2 \); that is, \( Q = Q_1 \times Q_2 \). We denote each member of \( Q \) as \([q_i, q_j]\), where \( q_i \in Q_1 \) and \( q_j \in Q_2 \). The initial state of \( M \) is \([q_0, q_0]\). At each state \([q_i, q_j]\) in \( Q \), we simulate both computations of \( M_1 \) and \( M_2 \) in parallel by

\[
\delta([q_i, q_j], a) = [\delta_1(q_i, a), \delta_2(q_j, a)].
\]
For instance, the computation path of $x = 0101$ in this product automaton is $[q_0, q_0]$, $[q_1, q_1]$, $[q_0, q_0]$, $[q_1, q_1]$, $[q_0, q_2]$. Furthermore, if $q_i \in F_1$ or $q_j \in F_2$, then we let $[q_i, q_j] \in \tilde{F}$. That is, $\tilde{F} = (F_1 \times Q_2) \cup (Q_1 \times F_2)$.

From the above description, it is clear that $M$ accepts the union of the DFA's $M_1$ and $M_2$. We show this product DFA $M$ in Figure 2.14, where each vertex with the label $(i, j)$ denotes the state $[q_i, q_j]$.

Two facts about this DFA are worth mentioning: First, since states $[q_0, q_1]$, $[q_1, q_0]$ and $[q_1, q_2]$ are unreachable from the initial state $[q_0, q_0]$, we omitted them in Figure 2.14. Second, states $[q_2, q_1]$, $[q_2, q_2]$ and $[q_3, q_2]$ can be merged into a single success state, since all three states are final states and there is no way to leave these three states once we get there. There are some general techniques of simplifying DFA's. We will discuss these techniques in Section 2.7. \hfill $\Box$

The above method can also be applied to the problems of finding the intersections or the differences of two languages which are accepted by DFA's. All we need is to construct the product DFA of the two given DFA's and choose different sets of final states. This method can also be generalized to construct the product DFA of three, four, or any finite number of DFA's.

**Example 2.13** The set of all binary strings having a substring 00 and ending with 01.

**Solution** 1. This language is the intersection of two languages $(0 + 1)^*00(0 + 1)^*$ and $(0 + 1)^*01$. In Example 2.12, we constructed a product DFA to accept the union of these two languages. Here, we can use the same product DFA, except that we will define the set $F$ of final states to consist of states in which both components are final states; that is, $F = F_1 \times F_2$. The transition diagram of the resulting DFA is just like that of Figure 2.14, except that the final set consists of only one state $[q_2, q_2]$. \hfill $\Box$
2.2 Examples of DFA's

Solution 2. We can also use the checker method of Example 2.8 to construct a DFA for this set. First, we note that if a string $x$ has a substring 00 and a suffix 01, then the occurrence of 00 must come before that of 01. So, we set up the two checkers as shown in Figure 2.15(a) and 2.15(b). Intuitively, states $q_0, q_1$ and $q_2$ are like those in Figure 2.9. State $q_3$ means that “substring 00 has been found and no prefix of 01 is found,” and state $q_4$ means that “substring 00 has been found and prefix 0 of 01 has also been found.” Note that the out-edges from state $q_2$ are not determined yet, since finding the substring 00 does not imply the string being accepted.

Now we add additional edges to the checkers as in Examples 2.8 and 2.10. It is easy to see that we need to define $\delta(q_0, 1) = \delta(q_1, 1) = q_3$, since we are still at the first stage of checking substring 00.

Next, we have $\delta(q_2, 1) = q_5$, which is a final state, since, at state $q_2$, we have just seen a substring 00 and the second 0 plus the new symbol 1 form the required suffix 01. We also let $\delta(q_3, 0) = q_4$, since $q_4$ is the state at which substring 00 has been found and a symbol 0 has just been read.

The actions at states $q_3, q_4$ and $q_5$ are just like those in Figure 2.11. That is, we add

\[
\delta(q_3, 1) = q_3, \quad \delta(q_4, 0) = q_4, \\
\delta(q_5, 0) = q_4, \quad \delta(q_5, 1) = q_3.
\]

The complete DFA is shown in Figure 2.15(c).

Note that, in Solution 1, the number of states in the product DFA $M$ is, in general, the product of the number $n_1$ of states in $M_1$ and the number $n_2$ of states in $M_2$. In Solution 2 here, the number of states is, in general, only...
the sum of \( n_1 \) and \( n_2 \). So, this method saves some states. It, however, needs some heuristics to determine the edges from state \( q_2 \) to the second part of the DFA.

**Example 2.14** The set of all binary strings having a substring 00 but not ending with 01.

*Solution.* This language is the difference of language \((0 + 1)^*00(0 + 1)^*\) minus language \((0 + 1)^*01\). Thus, we can use the same product DFA as we did in Examples 2.12 and 2.13, except that we need to choose the set of final states to consist of every state in which the first component is a final state of the first DFA and the second component is a nonfinal state of the second DFA; that is, our new final set is \( F = F_1 \times (Q_2 - F_2) \). Its transition diagram is like that of Figure 2.14, with the final states \( \{[q_2, q_0], [q_2, q_1]\} \).

A special case of subtraction is complementation: \( \overline{A} = \Sigma^* - A \). There is a simpler construction in this case. Note that in DFA \( M = (Q, \Sigma, \delta, q_0, F) \), for any input string \( x \), \( x \in L(M) \) if and only if \( \delta(q_0, x) \in F \). Equivalently, \( x \notin L(M) \) if and only if \( \delta(q_0, x) \notin F \). It follows that the DFA \( (Q, \Sigma, \delta, q_0, Q - F) \) accepts the complement of \( L(M) \).

**Example 2.15** The set \( L \) of all binary strings in which every block of four consecutive symbols contains a substring 01.

*Solution.* The condition “every block of four consecutive symbols contains a substring 01” is a global condition, which appears difficult to verify. By considering the complement \( \overline{L} \), we turn this condition into a simpler local condition: \( \overline{L} \) contains binary strings with a substring 0000, 0001, 0100, 0101, or 1111. We first construct a DFA accepting \( \overline{L} \) and then change all final states into nonfinal states and all nonfinal states into final states. A solution is shown in Figure 2.16.

The above four examples established the following properties of the class of languages accepted by DFA’s.

**Theorem 2.16** The class of languages accepted by DFA’s is closed under union, intersection, subtraction, and complementation.

**Exercise 2.2**

1. For each of the following languages, construct a DFA that accepts the language:

   (a) The set of binary strings beginning with 010.
   (b) The set of binary strings ending with 101.
   (c) The set of binary strings beginning with 10 and ending with 01.
2.2 Examples of DFA’s

(d) The set of binary strings having a substring 010 or 101.
(e) The set of binary strings in which the last five symbols contain at most three 0’s.
(f) The set of binary strings \( w \) in which \( \#_1(w) + 2\#_0(w) \) is divisible by 5, where \( \#_a(w) \) is the number of occurrences of the symbol \( a \) in string \( w \).
(g) The set of strings over the alphabet \( \{1, 2, 3\} \) in which the sum of all symbols is divisible by 5.
(h) The set of strings over the alphabet \( \{0, 1, 2\} \) which are the ternary expansions (base-3 representations) of positive integers which are congruent to 2 modulo 7.
(i) The set of binary strings in which every block of four symbols contains at least two 0’s.
(j) The set of binary strings in which every substring 010 is followed immediately by substring 111.

2. For each of the following languages, use the product automaton method to construct a DFA that accepts the language:

(a) The set of binary strings beginning with 010 or ending with 101.
(b) The set of binary strings having a substring 010 but not having a substring 101.
(c) The set of binary strings beginning with 010, ending with 101 and having a substring 0000.

3. For each of the following languages, use the checker method to construct a DFA that accepts the language:

(a) The set of Example 2.12.
(b) The set of Example 2.14.
(c) The set of Exercise 2(a) above.
(d) The set of Exercise 2(c) above.

2.3 Nondeterministic Finite Automata

A nondeterministic finite automaton (NFA) \( M = (Q, \Sigma, \delta, q_0, F) \) is defined in the same way as a DFA except that multiple-state transitions and \( \varepsilon \)-transitions are allowed.

What is a multiple-state transition? It means that at each move, there could be more than one next state. That is, for any state \( q \) and any input symbol \( a \), the value of \( \delta(q, a) \) is a subset of \( Q \), where \( \delta(q, a) = \{ p_1, p_2, \ldots, p_k \} \) means that the next state from state \( q \), after reading \( a \), can be any one of \( p_1 \), \( p_2 \), \ldots, or \( p_k \). A special case is that \( \delta(q, a) \) could be the empty set \( \emptyset \). This means that the machine has no next state and hangs, and the input is rejected regardless of what remaining input symbols are. It is the equivalent of going to a failure state in a DFA.

In addition to multiple-state transitions, we also allow \( \varepsilon \)-transitions (or, \( \varepsilon \)-moves) in an NFA. An \( \varepsilon \)-transition is a move in which the tape head does not do anything (it neither reads nor moves), but the state can be changed. In other words, we allow a transition like \( \delta(q, \varepsilon) = \{ p_1, \ldots, p_k \} \), which means that state \( q \) can be changed to any one of \( p_1, p_2, \ldots, \) or \( p_k \), without reading a symbol from the input.

Therefore, the transition function \( \delta \) is, formally, a function of the form

\[
\delta : Q \times (\Sigma \cup \{ \varepsilon \}) \to 2^Q,
\]

where \( 2^Q \) denotes the collection of all subsets of \( Q \). The following is an example of a transition function of an NFA \( M_1 \):

\[
\begin{array}{c|ccc}
\delta(q, a) & 0 & 1 & \varepsilon \\
q_0 & \emptyset & \{ q_0, q_1 \} & \{ q_1 \} \\
q_1 & \{ q_2 \} & \{ q_1, q_2 \} & \emptyset \\
q_2 & \{ q_2 \} & \emptyset & \{ q_1 \}
\end{array}
\]

NFA's, like DFA's, can also be represented by transition diagrams. In the transition diagram, we still use a vertex to represent a state and a labeled edge to represent a move, except that we allow multiple edges from one vertex to other vertices with the same label. That is, if \( \delta(q, a) = \{ p_1, p_2, \ldots, p_k \} \), then we draw \( k \) edges from state \( q \) to each of \( p_1, p_2, \ldots, p_k \), and each with a label \( a \). For instance, the transition diagram of the transition function of \( M_1 \) above is as shown in Figure 2.17. (We also made \( F = \{ q_2 \} \).

On an input \( x \), an NFA may have more than one computation path. For instance, for NFA \( M_1 \) shown in Figure 2.17, the input 01 has the following
three computation paths:

\[ q_0 \xrightarrow{\varepsilon} q_1 \xrightarrow{0} q_2 \xrightarrow{\varepsilon} q_1 \xrightarrow{1} q_1, \]
\[ q_0 \xrightarrow{\varepsilon} q_1 \xrightarrow{0} q_2 \xrightarrow{\varepsilon} q_1 \xrightarrow{1} q_2, \]
\[ q_0 \xrightarrow{\varepsilon} q_1 \xrightarrow{0} q_2 \xrightarrow{\varepsilon} q_1 \xrightarrow{1} q_2 \xrightarrow{\varepsilon} q_1. \]

Note that in the last two paths, after the NFA reads all input symbols 01, it can choose to halt in state \( q_2 \) or to use the \( \varepsilon \)-move to move to state \( q_1 \).

In general, the computation paths for an input \( x \) form a computation tree since they all start with the same state \( q_0 \) and then branch out to different states. Figure 2.18 shows the computation tree of the NFA \( M_1 \) on input 01. (In Figure 2.18, we added an edge \( q_2 \xrightarrow{\varepsilon} q_2 \) to indicate that the second path ends at \( q_2 \).)

\[ q_0 \xrightarrow{\varepsilon} q_1 \xrightarrow{0} q_2 \xrightarrow{\varepsilon} q_1 \xrightarrow{1} q_2 \xrightarrow{\varepsilon} q_1. \]

**Figure 2.18:** The computation tree of input 01.

Some of these computation paths lead to final states and some do not. How do we define the notion of an NFA accepting an input in such a situation? The answer is that the NFA accepts an input \( x \) if at least one computation path on \( x \) leads to a final state. For instance, in the above example, since the second computation path ends at state \( q_2 \in F \) after the NFA reads the input 01, the string 01 is accepted by the NFA \( M_1 \).

To formally define the notion of an NFA \( M \) accepting an input \( x \), we need to define the notion of \( \varepsilon \)-closure. The \( \varepsilon \)-closure of a subset \( A \subseteq Q \) is the set of states that can be reached from a state \( q \) in \( A \) by \( \varepsilon \)-moves (including the move from \( q \) to \( q \)). That is,

\[ \varepsilon \text{-closure}(A) = \{ p \in Q \mid \exists q_0, q_1, \ldots, q_m \in Q: q_0 = p, \quad q_m = p, \quad \text{and} \quad q_{i+1} \in \delta(q_i, \varepsilon), \quad i = 0, \ldots, m - 1 \}. \]
For instance, in the above example,
\[
\varepsilon\text{-closure}\left(\{q_0\}\right) = \{q_0, q_1\},
\varepsilon\text{-closure}\left(\{q_2\}\right) = \{q_1, q_2\}.
\]

Now, we extend the transition function \(\delta\) to the domain \(2^Q \times (\Sigma \cup \{\varepsilon\})\) by
\[
\delta(A, a) = \varepsilon\text{-closure}\left(\bigcup_{q \in \varepsilon\text{-closure}(A)} \delta(q, a)\right),
\]
and then further extend it to the domain \(2^Q \times \Sigma^*\) as follows:
\[
\delta(A, \varepsilon) = \varepsilon\text{-closure}(A),
\delta(A, xa) = \delta(\delta(A, x), a), \text{ if } x \in \Sigma^* \text{ and } a \in \Sigma.
\]

For instance, in the above example, we have \(\delta(\{q_0\}, 0) = \{q_1, q_2\}\), and \(\delta(\{q_1, q_2\}, 1) = \{q_1, q_2\}\). Therefore, \(\delta(\{q_0\}, 01) = \{q_1, q_2\}\). Note that \(\delta(\{q_0\}, x)\) is the set of all leaves in all computation paths of \(x\).

We can now formally define that an NFA \(M = (Q, \Sigma, \delta, q_0, F)\) accepts input \(x\) if \(\delta(\{q_0\}, x) \cap F \neq \emptyset\). For an NFA \(M\), we let \(L(M)\) denote the set of all strings accepted by \(M\); that is
\[
L(M) = \{x \in \Sigma^* \mid \delta(\{q_0\}, x) \cap F \neq \emptyset\}.
\]

It is easier to design NFA’s than DFA’s. The following are some examples.

**Example 2.17** Find an NFA that accepts the set of binary strings having a substring 010.

**Solution.** We show the NFA \(M\) in Figure 2.19. Note that it is just the checker for substring 010 plus two loops from state \(q_0\) to \(q_0\) with labels 0 and 1. These two loops allow the machine to wait until it successfully checks the substring 010. With these two waiting loops, we do not need to define \(\delta(q_1, 0) = q_1\) and \(\delta(q_2, 1) = q_5\), as we did in a DFA. Instead, we simply let \(\delta(q_1, 0) = \delta(q_2, 1) = \emptyset\).

We also show, in Figure 2.20, the computation tree of \(M\) on input \(x = 1001010\). Note that there are two occurrences of 010 in \(x\) and so there are two different accepting paths for \(x\). 

![Figure 2.19: NFA for Example 2.17.](image)
Example 2.18 Find an NFA that accepts the set of binary strings beginning with 010 or ending with 110.

Solution. The set of binary strings beginning with 010 is accepted by the NFA of Figure 2.21(a). The set of binary strings ending with 110 is accepted by the NFA of Figure 2.21(b). Note that the final state $q_3$ has no outlet; that is, $\delta(q_3,0) = \delta(q_3,1) = \emptyset$. So, if a computation path of a string $x$ reaches state $q_3$ before $x$ is completely read, it is a rejecting path (but that does not imply that $x$ is rejected).

Now, we combine these two NFA’s by adding an initial state and two $\varepsilon$-edges to the two old initial states to form the final NFA, as shown in Figure 2.21(c). \hfill \Box

Example 2.19 Find an NFA that accepts the set of binary strings with at least two occurrences of substring 01 and ends with 11.
Solution. We use the \( \varepsilon \)-moves to connect three simple NFA’s into one, as shown in Figure 2.22. Note that there is a special case where the second occurrence of the substring 01 runs into the suffix 11. It presents no problem for the design of the NFA; we simply add one extra \( \varepsilon \)-edge.

\[\Box\]

**Example 2.20** Find an NFA that accepts the set \( \{ 0^n 1^m \mid n, m \geq 0, n \equiv m \pmod{5} \} \).

Solution We apply the idea of product automata to construct the required NFA. First, we construct a simple NFA \( M_1 \) to separate the input strings into five groups according to the value of \( n \pmod{5} \) (see Figure 2.23(a)). Next, for each group, we construct a copy of \( M_1 \) to find the value \( m \pmod{5} \). Then, we assign final states to each copy of \( M_1 \) accordingly. Figure 2.23(b) shows the complete DFA.

\[\Box\]

In Section 2.2, we have seen how to construct a DFA to accept the union or the intersection of the languages accepted by two given DFA’s. Can we do this for NFA’s? It is actually easier, as demonstrated by Example 2.18, to construct an NFA to accept the union of the languages accepted by two given NFA’s. In addition, the following examples show that for given NFA’s \( M_1 \) and \( M_2 \), it is easy to construct NFA’s to accept \( L(M_1) \cdot L(M_2) \) and \( L(M_1)^{*} \). On the other hand, there is no simple way to construct an NFA for the intersection or the difference of \( L(M_1) \) and \( L(M_2) \) from two given NFA’s \( M_1 \) and \( M_2 \).

**Example 2.21** Let \( M_1 \) and \( M_2 \) be two NFA’s. Construct an NFA \( M \) such that \( L(M) = L(M_1) \cdot L(M_2) \).

Solution. Let \( M_1 = (Q_1, \Sigma, \delta_1, q_0^1, F_1) \) and \( M_2 = (Q_2, \Sigma, \delta_2, q_0^2, F_2) \) be two NFA’s. We construct NFA \( M \) as follows: we make a copy of each of \( M_1 \) and \( M_2 \). Then, we let the initial state \( q_0^1 \) of \( M_1 \) be the initial state of \( M \) and let
2.3 Nondeterministic Finite Automata

![Figure 2.23](image-url) Solution to Example 2.20.

*Figure 2.23:* Solution to Example 2.20.

![Figure 2.24](image-url) Concatenation of two NFA's.

*Figure 2.24:* Concatenation of two NFA's.

the set \( F \) of the final states of \( M \) be equal to \( F_2 \). We also add an \( \varepsilon \)-move from each state \( q \) in \( F_1 \) to the initial state \( q_0^M \) of \( M_2 \). We show the construction in Figure 2.24. For convenience, we only show one final state for each of \( M_1 \) and \( M_2 \). □

**Example 2.22** Let \( M_1 \) be an NFA. Construct an NFA \( M \) such that \( L(M) = L(M_1)^* \).

*Solution.* Let \( M_1 = (Q_1, \Sigma, \delta_1, q_0, F_1) \) be an NFA. We construct \( M \) by adding a new initial state \( s \) and a unique final state \( f \). Then, we add an \( \varepsilon \)-move from \( s \) to the initial state \( q_0 \) of \( M_1 \) and an \( \varepsilon \)-move from each \( q_i \in F_1 \) to the new final state \( f \). We also add, from each state \( q_i \in F_1 \), an \( \varepsilon \)-move to the initial state \( q_0 \) of \( M_1 \). Finally, we add an \( \varepsilon \)-move from the initial state \( s \) to the new
final state $f$ (so that the empty string $\varepsilon$ is accepted). This NFA $M$ is shown in Figure 2.25. Note that the new initial and final states are necessary. See Exercise 4 of this section for counterexamples.

$\square$

**Exercise 2.3**

1. Consider the NFA $M$ of Figure 2.26.

![Figure 2.26: The NFA of Exercise 1.](image)

(a) What are $\varepsilon$-closure{$\{q_0\}$} and $\varepsilon$-closure{$\{q_1, q_2, q_3\}$}?
(b) What are $\delta(\{q_0\}, 0)$ and $\delta(\{q_2, q_3\}, 1)$?
(c) Draw the computation trees of $M$ on strings $x = 011$ and $y = 101$. Does $M$ accept or reject $x$ and $y$?

2. For each NFA $M$ shown in Figure 2.27, determine what $L(M)$ is.

3. For each of the following languages, construct an NFA that accepts the language:

   (a) The set of binary strings that contain at least three occurrences of substring 010.
   (b) The set of binary strings that contain both substrings 010 and 101. [Hint: This is equivalent to the set of binary strings that contain either a substring 0101 or a substring 1010 or a substring 010 followed by 101 or a substring 101 followed by 010.]
2.4 Converting an NFA to a DFA

(c) The set of binary strings that contain either a substring 010 or a substring 101, and end with 111 or 000.

(d) The set of binary strings of which the (3n)th symbol is 0 for each $n \geq 1$.

(e) The set of binary strings $x$ of length $3n$ for some $n \geq 1$, such that, for each $1 \leq k \leq n$, at least one of the $(3k - 2)$nd, $(3k - 1)$st and $(3k)$th symbols of $x$ is 0.

(f) The set $\{0^{q}10^n10^q \mid q \equiv nm \pmod{5}\}$.

4. Prove that the new initial state $s$ and the final state $f$ in the construction of Example 2.22 are necessary. That is, find NFA’s $M_1$ and $M_2$ such that the NFA’s $M'_1$ and $M'_2$ of Figure 2.28 have the property $L(M'_1) \neq L(M_1)^*$ and $L(M'_2) \neq L(M_2)^*$.

2.4 Converting an NFA to a DFA

Although it is easier to construct, a nondeterministic machine is just an idealized machine which cannot be efficiently implemented in practice, since a real machine can only follow one computation path at a time. For the finite state automata, it is fortunate that there is a simple procedure to convert an NFA to an equivalent DFA which accepts the same language. Thus, the notion of NFA’s can be turned into practical use.
To see how to do this conversion, let us consider an NFA $M = (Q, \Sigma, \delta, q_0, F)$. For any $x \in \Sigma^*$, let $Q_x$ be the set of states which are the end states of the computation paths of $x$ in $M$; that is, $Q_x = \delta(\{q_0\}, x)$, where $\delta$ is the extended transition function defined on $2^Q \times \Sigma^*$. In particular, $Q_x = \varepsilon$-closure($\{q_0\}$). Then, $x$ is accepted by $M$ if and only if $Q_x \cap F \neq \emptyset$. Thus, it suggests that we can use these subsets $Q_x \subseteq Q$ as the states for the equivalent DFA. In other words, we would like to construct a DFA $M' = (Q', \Sigma, \delta', Q_s, F')$ with the following properties:

$$
Q' = \{Q_x \mid x \in \Sigma^*\},
F' = \{Q_x \mid Q_x \cap F \neq \emptyset\},
\delta'(Q_x, a) = Q_{xa}, \quad \text{for } x \in \Sigma^*, a \in \text{Sigma}.
$$

Note that each $Q_x$ is a subset of $Q$, and so $Q'$ is a finite set. Indeed, $Q' \subseteq 2^Q$ and so $|Q'| \leq 2^{|Q|}$. Furthermore, if $Q_x = Q_y$, then $Q_{xa} = Q_{ya}$ for all $a \in \Sigma$. It follows that the above definition of $\delta'$ is well defined. In fact, using the extended transition function $\delta$ of $M$, we know that for any $x \in \Sigma^*$ and $a \in \Sigma$, $\delta'(Q_x, a) = \delta(Q_x, a)$. The above analysis shows that this construction is feasible. We call it the subset construction for DFA's. The following are some examples.

**Example 2.23** An NFA $M = (Q, \{0, 1\}, \delta, q_0, F)$ is given by $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$, $F = \{q_3, q_4\}$, and

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>0</th>
<th>1</th>
<th>$\varepsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>${q_0}$</td>
<td>${q_0, q_2}$</td>
<td>${q_1}$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>${q_5}$</td>
<td>${q_2}$</td>
<td>$-$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>${q_3}$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$-$</td>
<td>$-$</td>
<td>${q_4}$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>${q_3}$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>$q_5$</td>
<td>$-$</td>
<td>${q_4}$</td>
<td>$-$</td>
</tr>
</tbody>
</table>

(See Figure 2.29(a) for the transition diagram of $M$.) Find a DFA which is equivalent to the NFA $M$. 

![Diagram](image-url)
2.4 Converting an NFA to a DFA

Solution. We construct the DFA $M'$ as follows:

Step 1. We let $Q_\varepsilon = \varepsilon$-closure($\{q_0\}$) as the initial state, and let $F' = \emptyset$ be the set of final states. Let $Q' = \{Q_\varepsilon\}$. If $Q_\varepsilon \cap F \neq \emptyset$, then add $Q_\varepsilon$ to $F'$.

Step 2. Repeat the following until $\delta'(Q_\varepsilon, a)$ is defined for all $Q_\varepsilon \in Q'$ and all $a \in \{0, 1\}$:

(1) Select $Q_\varepsilon \in Q'$ and $a \in \{0, 1\}$ such that $\delta(Q_\varepsilon, a)$ is not yet defined.

(2) Let $Q_{xa} = \delta(Q_\varepsilon, a)$.

(3) If $Q_{xa} \notin Q'$, then add $Q_{xa}$ to $Q'$, and also add it to $F'$ if $Q_{xa} \cap F \neq \emptyset$.

The whole process for this example is shown in the following table.

<table>
<thead>
<tr>
<th>$\delta'$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_\varepsilon = {q_0, q_1}$</td>
<td>${q_0, q_1, q_2}$</td>
<td>${q_0, q_1, q_2}$</td>
</tr>
<tr>
<td>$Q_0 = {q_0, q_1, q_2}$</td>
<td>${q_0, q_1, q_3} = Q_0$</td>
<td>${q_0, q_1, q_2, q_4}$</td>
</tr>
<tr>
<td>$Q_1 = {q_0, q_1, q_2}$</td>
<td>${q_0, q_1, q_3, q_4, q_5}$</td>
<td>${q_0, q_1, q_2} = Q_1$</td>
</tr>
<tr>
<td>$Q_{01} = {q_0, q_2, q_4}$</td>
<td>${q_0, q_1, q_3, q_4, q_5}$</td>
<td>${q_0, q_1, q_2} = Q_1$</td>
</tr>
<tr>
<td>$Q_{10} = {q_0, q_1, q_3, q_4, q_5}$</td>
<td>${q_0, q_1, q_3, q_4, q_5} = Q_{10}$</td>
<td>${q_0, q_1, q_2, q_4} = Q_{01}$</td>
</tr>
</tbody>
</table>

Note that, in Step 2, we did not have to consider states $Q_{00}, Q_{00}, Q_{00},$ and so on, since $Q_{00} = Q_0$ and so $Q_{00} = Q_{00} = Q_0$ and $Q_{00} = Q_{01}$. Similarly, since $Q_{11} = Q_1$, we do not need to consider $Q_{11w}$ for any $w \in \{0, 1\}^*$.

The transition diagram of the DFA $M'$ is shown in Figure 2.29(b). (We write $(i_1, i_2, \ldots, i_m)$ inside a vertex to denote the state $\{q_{i_1}, q_{i_2}, \ldots, q_{i_m}\}$.)

Example 2.24 Construct a DFA which is equivalent to the NFA $M = (\{p, q, r\}, \{0, 1\}, \delta, p, \{q, r\})$, where

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>${p, q}$</td>
<td>${p}$</td>
</tr>
<tr>
<td>$q$</td>
<td>$-$</td>
<td>${r}$</td>
</tr>
<tr>
<td>$r$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
</tbody>
</table>

Figure 2.29: Converting an NFA to a DFA.
Solution. Following the above procedure, we obtain the following table for the transition function $\delta'$ of the DFA:

<table>
<thead>
<tr>
<th>$\delta'$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_\varepsilon = {p}$</td>
<td>${p, q}$</td>
<td>${p} = Q_\varepsilon$</td>
</tr>
<tr>
<td>$Q_0 = {p, q}$</td>
<td>${p, q} = Q_0$</td>
<td>${p} = Q_0$</td>
</tr>
<tr>
<td>$Q_{01} = {p, r}$</td>
<td>${p, q} = Q_0$</td>
<td>${p} = Q_\varepsilon$</td>
</tr>
</tbody>
</table>

The DFA is thus

$$M' = (\{\{p\}, \{p, q\}, \{p, r\}\}, \{0, 1\}, \delta', \{\{p\}, \{p, q\}, \{p, r\}\}) .$$

\[ \square \]

From the above construction, we see that for each NFA there exists a DFA accepting the same language accepted by the NFA. Moreover, a DFA is also an NFA. Therefore, we have the following theorem.

**Theorem 2.25** A language is accepted by an NFA if and only if it is accepted by a DFA.

Theorem 2.25 provides us with an easy tool to construct a DFA for a given language $L$: we can first construct an NFA for $L$ and then convert it to a DFA. The following are some examples.

**Example 2.26** Let $M = (\{p, q, r, s\}, \{0, 1\}, \delta, p, \{q, s\})$ be an NFA given by

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>${q, s}$</td>
<td>${q}$</td>
</tr>
<tr>
<td>$q$</td>
<td>${r}$</td>
<td>${q, r}$</td>
</tr>
<tr>
<td>$r$</td>
<td>${s}$</td>
<td>${p}$</td>
</tr>
<tr>
<td>$s$</td>
<td>$-$</td>
<td>${p}$</td>
</tr>
</tbody>
</table>

Construct an NFA that accepts $\overline{L(M)}$.

Solution. It is hard to figure out the construction of such an NFA $M'$ directly from $M$. In particular, the naive approach of changing nonfinal states of $M$ to final states and changing final states of $M$ to nonfinal states does not work. This is due to the nature of nondeterminism of the NFA’s: It is possible that, for some string $x$, $Q_r \cap F \neq \emptyset$ and $Q_r \cap (Q - F) \neq \emptyset$; then, $x$ would be accepted by both $M$ and the new NFA $M'$. (For instance, here we have $Q_{01} = \{p, q, r\}$ and $F = \{q, s\}$, and so the string 01 would be accepted by both $M$ and $M'$.) Therefore, $M'$ does not accept $\overline{L(M)}$.

Instead, we can apply the subset construction to get the required NFA as follows: We first find a DFA $M_D$ equivalent to $M$ and, then, change final states of $M_D$ to nonfinal states and nonfinal states of $M_D$ to final states to get a new automaton $M'$. The new automaton we get is actually a DFA.
2.4 Converting an NFA to a DFA

\[ \begin{array}{c|cc}
\delta_D & 0 & 1 \\
\hline
Q_0 = \{p\} & \{q,s\} & \{q\} \\
Q_0 = \{q,s\} & \{r\} & \{p,q,r\} \\
Q_1 = \{q\} & \{r\} & \{q,r\} \\
Q_0 = \{r\} & \{s\} & \{p\} \\
Q_11 = \{p,q,r\} & \{q,r,s\} & \{p,q,r\} \\
Q_{111} = \{q,r\} & \{r,s\} & \{p,q,r\} \\
Q_{1111} = \{s\} & \emptyset & \{p\} \\
Q_{11111} = \emptyset & \emptyset & \emptyset \\
\end{array} \]

Figure 2.30: The transition function of \( M_D \) in Example 2.26.

For the first step, we obtain \( M_D = (Q_D, \{0,1\}, \delta_D, \{p\}, F_D) \), where the set \( Q_D \) and the transition function \( \delta_D \) are as shown in Figure 2.30, and the set of final states is

\[
F_D = \{ A \subseteq \{p,q,r,s\} \mid A \cap \{q,s\} \neq \emptyset \} \\
= \{\{q\}, \{s\}, \{q,s\}, \{q,r\}, \{r,s\}, \{p,q,r\}, \{q,r,s\}\}.
\]

Now, the DFA \( M' = (Q_D, \{0,1\}, \delta_D, \{p\}, Q_D - F_D) \) accepts \( L(M) \), where

\[
Q_D - F_D = \{\{p\}, \{r\}, \emptyset\}. \quad \square
\]

**Example 2.27** Construct a DFA accepting the set of all binary strings in which the fifth symbol from right is 0.

**Solution.** It is easy to construct an NFA for this language: Just draw a checker as shown in Figure 2.31(a). This checker recognizes all strings of length 5 which begins with 0.

Now, we add two loops \( q_0 \xrightarrow{0} q_0 \) and \( q_0 \xrightarrow{1} q_0 \) to the checker and this is the required NFA (see Figure 2.31(b)). More formally, this NFA can be expressed as

\( M = (\{q_0, q_1, \ldots, q_5\}, \{0,1\}, \delta, q_0, \{q_5\}) \), with

\[
\delta(q_0, 0) = \{q_0, q_1\}, \quad \delta(q_0, 1) = \{q_0\}, \\
\delta(q_i, 0) = \delta(q_i, 1) = q_{i+1}, \text{ for } 1 \leq i \leq 4, \\
\delta(q_5, 0) = \delta(q_5, 1) = 0.
\]

Next, we convert this NFA \( M \) to an equivalent DFA \( M' = (Q', \{0,1\}, \delta', \{q_0\}, F') \), where

\[
Q' = \{q' \mid q' \subseteq \{q_0, q_1, \ldots, q_5\}, q_0 \in q'\}, \\
F' = \{q' \in Q' \mid q_5 \in q'\},
\]
\[ \delta'(q', 0) = \{ q_0, q_1 \} \cup \{ q_{i+1} | q_i \in q' \text{ and } 1 \leq i \leq 4 \} \]
\[ \delta'(q', 1) = \{ q_0 \} \cup \{ q_{i+1} | q_i \in q' \text{ and } 1 \leq i \leq 4 \}. \]

This DFA \( M' \) has totally 32 states. We leave the transition diagram of \( \delta' \) to the reader. \( \square \)

**Example 2.28** Consider the following multiplication table on \( \{a, b, c\} \):

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>c</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>c</td>
<td>a</td>
</tr>
<tr>
<td>c</td>
<td>c</td>
<td>b</td>
<td>c</td>
</tr>
</tbody>
</table>

For any string \( w \) in \( \{a, b, c\}^+ \), denote by \( \text{value}(w) \) the value obtained by multiplying symbols in \( w \) from left to right. For instance, let \( w = abc \). Then, we get

\[
\text{value}(w) = ((a \times b) \times c) \times b = (a \times c) \times b = b \times b = c, \text{ and}
\]
\[
\text{value}(w^R) = ((b \times c) \times b) \times a = (a \times b) \times a = a \times a = c.
\]

Construct an NFA for the set \( L \) of all strings \( w \) over \( \{a, b, c\} \) such that \( \text{value}(w) = \text{value}(w^R) \). (E.g., \( abcb \in L \) and \( abbc \notin L \).)

**Solution.** Define \( Q = \{ q_x, q_a, q_b, q_c \} \) and \( \Sigma = \{a, b, c\} \). First, we define a transition function \( \delta : Q \times \Sigma \to Q \) by \( \delta(q_x, x) = q_x \) for \( x \in \Sigma \), and \( \delta(q_x, y) = q_z \) if \( x \times y = z \), for \( x, y, z \in \Sigma \); that is,

\[
\delta
\begin{array}{c|ccc}
\delta & a & b & c \\
\hline
q_x & q_a & q_b & q_c \\
q_a & q_c & q_a & q_b \\
q_b & q_b & q_c & q_a \\
q_c & q_c & q_b & q_c \\
\end{array}
\]

Clearly, for each \( x \in \Sigma \), the DFA \( M_x = (Q, \Sigma, \delta, q_x, \{ q_x \}) \) accepts the set of all strings \( w \) over \( \Sigma \) having \( \text{value}(w) = x \). We show \( M_a \) in Figure 2.32(a).
Next, we define $Q' = \{q_0, q_b, q_c, q_f\}$ and a transition function $\delta' : Q' \times \Sigma \rightarrow 2^Q$ by letting $q_f \in \delta(q_x, x)$ for all $x \in \Sigma$, and $q_x \in \delta(q_z, y)$ if $z = x \times y$, for $x, y, z \in \Sigma$. That is, we change state $q_x$ in Figure 2.32(a) to state $q_f$, and reverse all arrows in Figure 2.32(a). Then, for each $x \in \Sigma$, the DFA $M'_x = (Q', \Sigma, \delta', q_x, \{q_f\})$ accepts the set of all strings $w$ over $\Sigma$ having value(${w^M}$) $= x$. Figure 2.32(b) shows the NFA $M'_c$.

Now, the language $L$ can be described as

$$ L = (L(M_a) \cap L(M'_a)) \cup (L(M_b) \cap L(M'_b)) \cup (L(M_c) \cap L(M'_c)) $$

In Section 2.3, we have seen that it is easy to construct an NFA to accept the union or the concatenation of two languages accepted by two given NFA's. For the intersection of two languages, however, there is no simple way to do it. Here, we are going to use the product automaton method introduced in Section 2.2 to do it.

Let $Q'' = Q \times Q'$. To simplify the notation, we will write $q_{ux}$ to denote the state $[q_u, q_v]$ in $Q''$. Define $\delta'' : Q'' \times \Sigma \rightarrow Q''$ by

$$ \delta''(q_{ux}, x) = \{ q_{wx} \mid q_w = \delta(q_u, x) \text{ and } q_z \in \delta'(q_v, x) \} $$

We show the complete table for $\delta''$ in Figure 2.33. (Note that $\delta''(q_{xf}, y) = \emptyset$ for all $x, y \in \Sigma$, and so we do not show them in the table.)

Then, for each $x \in \{a, b, c\}$, the NFA $M''_x = (Q'', \Sigma, \delta'', q_x, \{q_f\})$ accepts the language $L(M_x) \cap L(M'_x)$.

Now, to get an NFA accepting $L(M''_a) \cup L(M''_b) \cup L(M''_c)$, we can simply use three copies of $M''_a$, $M''_b$, $M''_c$ and add an initial state $q_s$ with three $\varepsilon$-transitions from $q_s$ to the three initial states of the machines $M''_a$, $M''_b$, $M''_c$. More pre-
\[ \delta'' \]

<table>
<thead>
<tr>
<th>( \delta'' )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_{sa} )</td>
<td>( {q_{af}} )</td>
<td>( {q_{ba}} )</td>
<td>( {q_{cb}} )</td>
</tr>
<tr>
<td>( q_{ab} )</td>
<td>( {q_{ab}} )</td>
<td>( {q_{bf}, q_{bc}} )</td>
<td>( {q_{ca}} )</td>
</tr>
<tr>
<td>( q_{ac} )</td>
<td>( {q_{ac}, q_{ax}} )</td>
<td>( {q_{ab}} )</td>
<td>( {q_{cf}, q_{cc}} )</td>
</tr>
<tr>
<td>( q_{aa} )</td>
<td>( {q_{af}} )</td>
<td>( {q_{ba}} )</td>
<td>( {q_{cb}} )</td>
</tr>
<tr>
<td>( q_{ab} )</td>
<td>( {q_{af}} )</td>
<td>( {q_{ax}} )</td>
<td>( {q_{ca}} )</td>
</tr>
<tr>
<td>( q_{ac} )</td>
<td>( {q_{ac}, q_{ax}} )</td>
<td>( {q_{ab}} )</td>
<td>( {q_{bf}, q_{bc}} )</td>
</tr>
<tr>
<td>( q_{ba} )</td>
<td>( {q_{ba}} )</td>
<td>( {q_{ab}} )</td>
<td></td>
</tr>
<tr>
<td>( q_{bc} )</td>
<td>( {q_{ba}, q_{bc}} )</td>
<td>( {q_{cb}} )</td>
<td>( {q_{cf}, q_{cc}} )</td>
</tr>
<tr>
<td>( q_{ca} )</td>
<td>( {q_{cf}} )</td>
<td>( {q_{bc}} )</td>
<td>( {q_{cb}} )</td>
</tr>
<tr>
<td>( q_{cb} )</td>
<td>( {q_{cb}} )</td>
<td>( {q_{bc}} )</td>
<td>( {q_{cf}, q_{cc}} )</td>
</tr>
<tr>
<td>( q_{cc} )</td>
<td>( {q_{cc}} )</td>
<td>( {q_{cf}, q_{cc}} )</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2.33:** The transition function \( \delta'' \).

cisely, the required NFA can be described as \( \widetilde{M} = (\widetilde{Q}, \Sigma, \delta, \widetilde{q}_s, \{q_{af}^a, q_{bf}^b, q_{cf}^c\}) \), where

\[ \widetilde{Q} = \{q_{uv}^t | t \in \{a, b, c\}, q_{uv} \in Q'' \} \cup \{\widetilde{q}_s\}, \]

and

\[ \delta(q_s, \varepsilon) = \{q_{sa}^a, q_{ab}^b, q_{ac}^c\}, \]

\[ \delta(q_{uv}^t, x) = \{q_{uz}^t | q_{uz} \in \delta''(q_{uv}, x)\}, \text{ for } q_{uv} \in Q'', t, x \in \{a, b, c\}. \]

**Exercise 2.4**

1. Convert each of the following NFA’s into an equivalent DFA:
   
   (a) The NFA \( M = (\{p, q, r\}, \{0, 1\}, \delta, p, \{q, r\}) \), with

   \[
   \begin{array}{c|cc}
   \delta & 0 & 1 \\
   \hline
   p & p & \{p, q\} \\
   q & \{r\} & - \\
   r & - & - \\
   \end{array}
   \]

   (b) The NFA \( M \) of Figure 2.27(a).
   
   (c) The NFA \( M \) of Figure 2.27(b).
   
   (d) The NFA \( M \) of Figure 2.27(c).

2. For each of the following languages, construct a DFA that accepts the language:

   (a) The set of binary strings that contain both substring 010 and substring 101.
2.5 Finite Automata and Regular Expressions

(b) The set of all binary strings ending with 00 or 01 or 10.
(c) The set of binary strings that contain both 001 and 110 as substrings or contain neither 001 nor 110 as a substring.
(d) The set of binary strings in which both the fourth symbol from the right and the fourth from the left symbols are 0. [Note: Both strings 0110 and 10101 belong to this set.]

*3. Consider the following multiplication table on \{a, b, c\}:

\[
\begin{array}{ccc}
\times & a & b & c \\
a & a & c & b \\
b & a & b & c \\
c & c & c & a \\
\end{array}
\]

Construct NFA’s for the following languages:
(a) \{x \mid \text{value}(x) \neq \text{value}(x^R)\}.
(b) \{xy \mid \text{value}(x) = \text{value}(y^R)\}.
(c) \{xy \mid \text{value}(x) = \text{value}(y)\}.

2.5 Finite Automata and Regular Expressions

In this section, we are going to show that DFA’s accept exactly the class of regular languages. This result is to be established in three steps:

1. If \(L\) is a regular language, then it is accepted by an NFA.
2. If \(L\) is accepted by an NFA, then it is accepted by a DFA.
3. If \(L\) is accepted by a DFA, then it is a regular language.

In Section 1.3, we showed that a regular expression \(r\) has a labeled digraph representation \(G(r)\) such that for each string \(x, x \in L(r)\) if and only if there is a path in \(G(r)\) from the initial vertex to the final vertex whose associated labels are exactly the string \(x\). Furthermore, each edge of \(G(r)\) is labeled by exactly one symbol from \(\{\varepsilon\} \cup \Sigma\). Thus, it is clear that \(G(r)\) is the transition diagram of an NFA, with the initial vertex denoting the initial state and the final vertex denoting the unique final state. That is, part (1) above has been proven in Section 1.3.

Next, we note that part (2) has already been done in Section 2.4. Therefore, by combining parts (1) and (2), we can construct a DFA from any given regular expression. Before we prove part (3), let us see some examples of constructing an NFA or a DFA from a given regular expression.

**Example 2.29** Find a DFA accepting the language \(10 \oplus (0 + 11)^* 1\).

**Solution.** First, we find an NFA to accept this language by using the method of Section 1.3. Then we transform this NFA to a DFA. The result is shown in Figure 2.34. \(\square\)
Example 2.30  Construct an NFA accepting the set $L$ of binary strings of an odd length which contain the substring 00.

Solution. A string $x$ having a substring 00 can be written as $x = y00z$, for some $y, z \in \{0, 1\}^*$. This string $x$ is of an odd length either if $|y|$ is even and $|z|$ is odd, or if $|y|$ is odd and $|z|$ is even. Thus, $L$ can be represented as

$$\begin{align*}
((0+1)(0+1))^*00((0+1)(0+1))^*(0+1) \\
+(0+1)((0+1)(0+1))^*00((0+1)(0+1))^*.
\end{align*}$$

By using the method of Section 1.3, we obtain an NFA as shown in Figure 2.35.

Next, we prove part (3) by showing how to construct a regular expression from a given DFA. Our method actually works even if the given automaton is
an NFA. In fact, we will prove this result on the labelled digraphs introduced in Section 1.3.

Recall that a labelled digraph $G$ is a digraph with two special vertices, the initial vertex $v_1$ and the final vertex $v_f$, in which each edge is labelled by a regular expression. For each path $\pi$ from $v_1$ to $v_f$ in such a digraph $G$, we let $r(\pi)$ be the regular expression associated with $\pi$; that is, if $\pi$ is

$$v_1 \xrightarrow{r_1} v_i \xrightarrow{r_2} v_{i_2} \xrightarrow{r_3} \cdots \xrightarrow{r_k} v_f,$$

then $r(\pi) = r_1r_2 \cdots r_k$. Then, for each labelled digraph $G$, we let

$$L(G) = \bigcup \{ L(r(\pi)) \mid \pi \text{ is a path from } v_1 \text{ to } v_f \}.$$ 

We now show, by induction on the number of vertices in $G$ other than $v_1$ and $v_f$, that for each labelled digraph $G$, $L(G)$ is a regular set.

**Basis Step** For $n = 0$, the digraph $G$ has only two vertices $v_1$ and $v_f$, or only one vertex $v_1 = v_f$. We first eliminate all multiple edges in $G$ by combining multiple edges with labels $r_1, r_2, \ldots, r_m$ from a vertex $u$ to a vertex $v$ into a single edge from $u$ to $v$ with the label $r_1 + r_2 + \cdots + r_m$ (see Figure 2.36). Then, the resulting digraph are of two types, as shown in Figure 2.37, in which $a, b, c, d$ denote four regular expressions.

It is clear that a type (1) digraph $G$ has $L(G) = a^*$. For a type (2) digraph $G$, consider a path $\pi$ from $v_1$ to $v_f$. Assume that $\pi$ passes through $v_f$ for $k \geq 1$ times. Then, we can write $\pi$ as $\pi_1\pi_2\cdots\pi_k$, where $\pi_1$ is a path from $v_1$ to $v_f$, and $\pi_2, \ldots, \pi_k$ are paths from $v_f$ to $v_f$, with $v_f$ not occurring as an intermediate vertex in any path $\pi_j$, $1 \leq j \leq k$. Then, it is clear that
Figure 2.38: Induction step.

\[ r(\pi) = a_1^i b \] for some \( i_1 \geq 0 \) and, for \( j = 2, \ldots, k \), \( r(\pi_j) \) is either \( c \) or \( da_2^j b \) for some \( i_j \geq 0 \). Or, equivalently, \( r(\pi) \in a^* b (c + da^* b)^* \). Conversely, we can see that, for any regular expression \( r \) in \( a^* b (c + da^* b)^* \), there is a path \( \pi \) in \( G \) from \( v_1 \) to \( v_f \), with \( r(\pi) = r \). Thus, it follows that

\[ L(G) = a^* b (c + da^* b)^* , \]

and the basis step is proven.

**Induction Step.** For \( n \geq 1 \), we can choose a vertex \( v \) other than \( v_1 \) and \( v_f \) and remove it by the following transformation: First, by the method used in the basis step, we may eliminate multiple edges and assume that there is at most one edge from any vertex \( u \) to any vertex \( w \). Now, assume that \( v \) has a self-loop with label \( c \); that is, \( G \) has an edge \( v \xrightarrow{c} v \). We remove \( v \) along with this edge. For any pair \((u, w)\) of vertices in \( G \), with edges \( u \xrightarrow{a} v \), \( v \xrightarrow{b} w \) and \( u \xrightarrow{d} w \), we remove the edges \( u \rightarrow v \) and \( v \rightarrow w \) and replace the label \( d \) of the edge \( u \rightarrow w \) with a new label \( d + ac^* b \). (If there was no edge \( u \rightarrow w \) in \( G \), we treat it as having an edge \( u \rightarrow w \) with label \( \emptyset \).) We show this transformation in Figure 2.38. (To make the figure readable, we omit some edges from vertices on the left to vertices on the right. We assume that the edge from the \( i \)th vertex on the left to the \( j \)th vertex on the right has the label \( d_{ij} \).) Note that this transformation does not change the associated language. Thus, by the induction hypothesis, the language accepted by \( G \) is regular.

The above completes the proof that the languages associated with labelled digraphs are all regular. Finally, we show that for any NFA \( M \), \( L(M) \) is regular. For any NFA \( M \), we first convert it to an equivalent NFA \( M' \) with a single final state by changing all final states in \( M \) into nonfinal states, and adding a new final state and a new \( \epsilon \)-move from each old final state to it (see Figure 2.39). Now the transition diagram of this NFA \( M' \) is exactly a labelled digraph \( G \) with the property that each edge is labelled by a single symbol from \( \{ \epsilon \} \cup \Sigma \). It is also clear that \( L(G) = L(M') \). Thus, by the above proof, \( L(M') \) is a regular language.

We have just proved the following theorem:
Theorem 2.31  Let \( L \) be a language. The following are equivalent:
(a) \( L \) is a regular language.
(b) There is a DFA \( M \) such that \( L(M) = L \).
(c) There is an NFA \( M \) such that \( L(M) = L \).

Example 2.32  Find a regular expression for the language accepted by NFA in Figure 2.29(a).

Solution. By using the above method, we can compute a regular expression for the language accepted by NFA in Figure 2.29(a) as shown in Figure 2.40. First, we create a new unique final state and then eliminate state \( q_5 \). Then, we eliminate states \( q_1, q_3, q_3 \) and \( q_4 \), one at a time. Finally, we use the basis step to get the regular expression \( (0 + 1)^*(01 + 10)0^* \) from the last digraph.

Exercise 2.5

1. For each of the following regular expressions \( r \), construct a DFA that accepts \( I(r) \):
   (a) \( (0 + 1)^*(1 + 01)^* \).
   (b) \( (0 + 1)^*0(0 + 1)(0 + 1)0(0 + 1) \).
   (c) \( 0(0 + 1)^*0 + 1(0 + 1)^*1 \).

2. For each of the following languages, find an NFA that accepts it:
   (a) \( \{ x \# y \mid x, y \in (0 + 1)^*, |x| \equiv |y| \pmod{2} \} \).
   (b) \( \{ x \# y \mid x, y \in (0 + 1)^*, |x| + |y| \geq 5 \} \).
   (c) \( \{ x \# y \mid x, y \in (0 + 1)^*, |x| \cdot |y| \text{ is divisible by } 5 \} \).

3. Figure 2.41 shows an NFA accepting \( 0^* \), constructed based on the method of Example 2.22. The four \( \varepsilon \)-moves cannot be eliminated by the rule of Theorem 1.25. Apply the method in the proof of Theorem 2.31 to reduce some of its \( \varepsilon \)-moves. Can you find, from this example, a more general rule (than Theorem 1.25) to eliminate redundant \( \varepsilon \)-transitions?
Figure 2.40: Finding a regular expression from an NFA.

\[(01+10)0^* (\varepsilon+0)+10 = (01+10)0^*\]

Figure 2.41: An NFA accepting \(0^*\).

4. For each of the languages accepted by NFA's of Figure 2.42, find a regular expression for it.

2.6 Closure Properties of Regular Languages

In Theorem 2.31, we have shown that a regular language has three types of representations: regular expressions, DFA's and NFA's. This equivalence
2.6 Closure Properties of Regular Languages

result provides us with useful tools to manipulate regular expressions or finite automata to get new regular expressions or finite automata. In this section, we apply these tools to prove that regular languages are closed under many language operations $\Phi$, in the sense that if a given language $L$ is regular then the language $\Phi(L)$ is also a regular language. Furthermore, most results are proved by constructive methods.

We start with the most common language operations introduced in Section 1.1.

**Theorem 2.33** The class of regular languages is closed under union, intersection, subtraction, complementation, concatenation, Kleene closure and reversal.

Proof. We showed in Section 2.2 how to construct, from two given DFA’s $M_1$ and $M_2$, new DFA’s to accept the union, the intersection and the difference of languages $L(M_1)$ and $L(M_2)$, as well as the complement of $L(M_1)$. Furthermore, we showed in Examples 2.21 and 2.22 how to construct, from two given NFA’s $M_3$ and $M_4$, NFA’s to accept the concatenation and the Kleene closure of $L(M_3)$ and $L(M_4)$.

Finally, we can prove by induction that if $L$ is regular then $L^R$ is regular (cf. Example 1.21). First, for the basis step, we check that $\emptyset^R = \emptyset$, $\{\varepsilon\}^R = \{\varepsilon\}$ and $\{a\}^R = \{a\}$. Next, for the induction step, we recall from Example 1.8 that $(AB)^R = B^RA^R$ and $(A \cup B)^R = A^R \cup B^R$. Furthermore, use a similar proof of Example 1.8, we can prove that $(A^*)^R = (A^R)^*$. Thus, the induction step is complete. \qed

The following example presents a more constructive method for proving that regular languages are closed under reversal.

**Example 2.34** Let $M$ be an NFA. Construct an NFA $M'$ such that $L(M') = L(M)^R$.

**Solution.** From the last example, we can first find a regular expression $r$ such that $L(r) = L(M)$. Next, we get a regular expression $s$ such that $L(s) =
Finally, we use the method of Section 2.5 to construct an NFA $M'$ such that $L(M') = L(s)$.

This process, however, is too complicated, with three nontrivial constructions. Here, we present a different approach, with a simple idea that has many applications in other construction problems involving NFA's.

Assume that $M = (Q, \Sigma, \delta, q_0, F)$ is an NFA. Our construction of $M'$ is based on the following idea: On input $x$, we simulate $M$ on $x$, starting from a final state and going backward. That is, we reverse the arrows in the transition diagram of $M$, and search for a path from a final state $q_f$ of $M$ to the initial state $q_0$, with the labels $x$. Note that if such a path is found, then its reversed path is an accepting path for $x^R$. Also, if there is a computation path of $M$ on $x^R$, then its reversal is a such a path for $x$. Thus, $M$ accepts $x^R$ if and only if the reversed machine $M'$ accepts $x$.

Formally, assume that $Q = \{q_0, q_1, \ldots, q_n\}$. We add a new starting state $s$ and let $M' = (Q \cup \{s\}, \Sigma, \delta', s, \{q_0\})$, where

\[
\delta'(s, \varepsilon) = F,
\]

\[
\delta'(q_i, a) = \{q_j \in Q \mid q_i \in \delta(q_j, a)\}, \text{ for } q_i \in Q \text{ and } a \in \Sigma \cup \{\varepsilon\}.
\]

Finally, we remark that the machine $M'$ obtained from this reversal construction is, in general, not a DFA even if $M$ itself is a DFA: It is possible that, for some state $q_i$ and some symbol $a$, there are more than one states $q_j$ such that $\delta(q_j, a) = q_i$. In other words, the reversal simulation requires the ability of nondeterministic search for the right path from a final state to the initial state. \hfill \square

Next, we consider a new language operation called substitution. Let $f$ be a mapping which maps each symbol $a \in \Sigma$ to a language $L_a$ over an alphabet $\Gamma$. We extend function $f$ to the domain of $\Sigma^*$ by $f(\varepsilon) = \{\varepsilon\}$ and $f(a_1a_2\cdots a_k) = f(a_1)f(a_2)\cdots f(a_k)$, for $a_1, \ldots, a_k \in \Sigma$. Then, for any language $L \subseteq \Sigma^*$, we obtain a new language by applying the substitution function $f$ to $L$:

\[
f(L) = \bigcup_{x \in L} f(x).
\]

For example, suppose that $L = \{01, 10\}$ and $f(0) = 0(0+1)^*$, $f(1) = (0+1)^*1$. Then,

\[
f(L) = 0(0+1)^*1 \cup 1(0+1)^*0.
\]

A substitution $f$ is called a homomorphism if for any $a \in \Sigma$, $f(a)$ is a language with a single string.

**Example 2.35** Let $f$ be a substitution over $\Sigma$. Assume that $L \subseteq \Sigma^*$ is a regular language, and that $f(a)$ is a regular language for each $a \in \Sigma$. Then, $f(L)$ is also a regular language.
2.6 Closure Properties of Regular Languages

Proof. Let \( r \) be a regular expression for language \( L \) and, for each \( a \in \Sigma \), \( r_a \) be the regular expression for language \( f(a) \). Replace each occurrence of symbol \( a \) in \( r \) by \( f(a) \). Then, we obtain a new regular expression \( r' \).

Now, we observe the following simple facts:

(a) For any two sets \( A \subseteq \Sigma^* \) and \( B \subseteq \Sigma^* \), we have \( f(A \cup B) = f(A) \cup f(B) \), \( f(AB) = f(A)f(B) \), and \( f(A^*) = f(A)^* \).

(b) For any two regular expressions \( r \) and \( s \), \( (r + s)' = r' + s' \), \( (rs)' = r's' \), and \( (r^*)' = (r')^* \).

Using these facts, we can prove by a simple induction that \( L(r') = f(L) \) (cf. Examples 1.21 and 1.22). Thus, \( f(L) \) is regular. \( \square \)

Define the quotient of two languages \( L_1 \) and \( L_2 \) as

\[
L_1 / L_2 = \{ x \mid (\exists y \in L_2) [xy \in L_1] \}.
\]

For instance, if \( L_1 = \{ w \in \{0,1\}^* \mid w \) has an even number of occurrences of symbol 0\} \), \( L_2 = \{ 0 \} \) and \( L_3 = \{ 0, 00 \} \), then \( L_1 / L_2 = \{ w \in \{0,1\}^* \mid w \) has an odd number of occurrences of symbol 0\} \), and \( L_1 / L_3 = \{ 0, 1 \}^* \).

**Example 2.36** Show that for any language \( L_2 \), if \( L_1 \) is regular then \( L_1 / L_2 \) is also regular.

Proof. Suppose that \( L_1 \) is accepted by a DFA \( M = (Q, \Sigma, \delta, q_0, F) \). For any string \( x \in L_1 / L_2 \), there exists a string \( y \in L_2 \) such that \( \delta(q_0, xy) = \delta(q_0, x), y \in F \). In other words, suppose that we run \( M \) on \( x \) and reach a state \( q_i = \delta(q_0, x) \). If there exists \( y \in L_2 \) such that \( \delta(q_i, y) \in F \), then \( x \) should be accepted; that is, \( q_i \) should be considered as a final state for \( L_1 / L_2 \). Therefore, we can simply define

\[
F' = \{ q \in Q \mid (\exists y \in L_2) [\delta(q, y) \in F] \},
\]

and let \( M' = (Q, \Sigma, \delta, q_0, F') \). It is clear that \( M' \) accepts \( L_1 / L_2 \). \( \square \)

**Example 2.37** Let \( L \) be a regular language over \( \Sigma \), \( k \) a positive integer and \( \phi \) a mapping from \( \Sigma^k \) to \( \Sigma \). Prove that

\[
L_1 = \{ \phi(a_1 a_2 \cdots a_k) \cdots \phi(a_{(n-1)k+1} a_{(n-1)k+2} \cdots a_{nk}) \mid a_1 a_2 \cdots a_{nk} \in L \}
\]

is regular.

Proof. Assume that \( L \) is accepted by a DFA \( M = (Q, \Sigma, \delta, s, F) \). Then, the required NFA can be defined as \( M' = (Q, \Sigma, \delta', s, F) \), where for each \( q \in Q \) and each \( a \in \Sigma \),

\[
\delta'(q, a) = \{ \delta(q, a_1 a_2 \cdots a_k) \mid \phi(a_1, a_2, \cdots, a_k) = a \}.
\]

\( \square \)

For any language \( L \), let

\[
\text{MIN}(L) = \{ x \in L \mid \text{no proper prefix of } x \text{ belongs to } L \}.
\]
Example 2.38 Prove that if $L$ is regular, so is $\text{MIN}(L)$.

Proof. Assume that $M = (Q, \Sigma, \delta, q_0, F)$ is a DFA which accepts $L$. Then, $\text{MIN}(L)$ is accepted by the NFA $M'$ obtained from $M$ by deleting all out-edges from final states. □

For any two bits $a, b \in \{0, 1\}$, $a \lor b$ denotes the disjunction of $a$ and $b$; that is, $0 \lor 0 = 0$ and $0 \lor 1 = 1 \lor 0 = 1 \lor 1 = 1$. For any two binary strings $x$ and $y$ with $|x| = |y|$, $x \lor y$ denotes the bitwise disjunction of $x$ and $y$. For example, if $x = 0011$ and $y = 0101$, then $x \lor y = 0111$.

Example 2.39 Show that if $A$ and $B$ are regular languages over $\{0, 1\}$, then

$$A \lor B = \{ x \lor y \mid x \in A, y \in B, |x| = |y| \}$$

is also regular.

Proof. Assume that DFA’s $M_A = (Q_A, \{0, 1\}, \delta_A, s_A, F_A)$ and $M_B = (Q_B, \{0, 1\}, \delta_B, s_B, F_B)$ accept sets $A$ and $B$, respectively. To accept language $A \lor B$, we build a product DFA $M'$ to simulate $M_A$ and $M_B$. At each move, if the input symbol is 0, then we simulate them in a normal way. If the input symbol is 1, then this symbol may come from $0 \lor 1, 1 \lor 0$ or $1 \lor 1$; we simulate all three possible computation paths in $M_A$ and $M_B$. More precisely, we let $M' = (Q_A \times Q_B, \{0, 1\}, \delta', s_A \times s_B, F_A \times F_B)$, where

$$\delta'([p, q], 0) = \{ [\delta_A(p, 0), \delta_B(q, 0)] \}$$

$$\delta'([p, q], 1) = \{ [\delta_A(p, 0), \delta_B(q, 1), [\delta_A(p, 1), \delta_B(q, 0)], [\delta_A(p, 1), \delta_B(q, 1)] \}. □$$

*Example 2.40 Show that if $L$ is a regular language, so is

$$\{ xy \mid yx \in L \}.$$

Proof. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA accepting $L$. Assume that $Q = \{ q_0, q_1, \ldots, q_n \}$. Intuitively, we can simulate $M$ to decide whether an input $w$ is in $\{ xy \mid yx \in L \}$ or not as follows:

1. We nondeterministically divide $w$ into two parts $w = xy$.
2. Then, we nondeterministically jump to a state $q_i$, and simulate $M$ on $x$ starting from state $q_i$.
3. Suppose that $\delta(q_i, x) = q_j$ is not in $F$, then this simulation fails. Otherwise, we simulate $M$ on $y$ starting from state $q_0$, and accept $w$ if $\delta(q_0, y) = q_i$.

How do we implement the nondeterministic guess of state $q_i$ in step (2)? We create many copies of $M$ and each copy implements one choice. The following is a more formal construction:
2.6 Closure Properties of Regular Languages

Figure 2.43: Solution to Example 2.40.

For each state $q_i$ of $M$, we make two copies of $M$: $\hat{M}_i$ and $\check{M}_i$. We connect all final states of $\hat{M}_i$ to the initial state of $\check{M}_i$ with $\varepsilon$-edges. For a state $q_j$ in $M$, we denote the corresponding states in $\hat{M}_i$ and $\check{M}_i$ by $\hat{q}_j^i$ and $\check{q}_j^i$, respectively. Now, we create a new initial state $s$ and a new final state $f$, and connect $s$ to state $\hat{q}_i^i$ for every $\hat{M}_i$ and connect state $\check{q}_i^i$ for every $\check{M}_i$ to $f$ with $\varepsilon$-edges. Also, all final states in $\hat{M}_i$ and $\check{M}_i$ are changed to nonfinal states. (See Figure 2.43.)

We let $M'$ be the resulting NFA, and claim that $L(M') = \{ xy \mid yx \in L \}$. First, if $yx \in L$, then there exist states $q_i \in Q$ and $q_j \in F$ such that $\delta(q_i, y) = q_i$ and $\delta(q_i, x) = q_j$. Therefore, the following path in $M'$ accepts $xy$:

$$s \xrightarrow{\varepsilon} \hat{q}_i^i \xrightarrow{x} \check{q}_j^i \xrightarrow{\varepsilon} \hat{q}_j^i \xrightarrow{y} \check{q}_i^i \xrightarrow{\varepsilon} f.$$

Conversely, we can see that any accepting path in $M'$ must be of the above form, with $q_j \in F$. Thus, the original DFA accepts $yx$ by the following path:

$$q_0 \xrightarrow{y} q_i \xrightarrow{x} q_j.$$

That means $M'$ only accepts strings of the form $xy$ with $yx \in L$. \hfill \Box

* Example 2.41 Show that if $L$ is a regular language, so is $L_\downarrow = \{ x \mid \exists y \mid |x| = |y|, xy \in L \}$.

Proof. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA accepting $L$, with $Q = \{ q_0, q_1, \ldots, q_n \}$. We will construct an NFA accepting $L_\downarrow$. Intuitively, we can simulate $M$ on $x$ to decide whether $x \in L_\downarrow$ as follows:

1. We nondeterministically guess a final state $q_j \in F$ and a string $y$ with $|y| = |x|$.
2. We simulate $M$, in parallel, on $x$ from $q_0$ forward, and on $y$ from $q_j$ backward. (As demonstrated in Example 2.54, we reverse the direction of the arrows in the transition diagram of $M$ to simulate it backward.)
(3) We accept \( x \) if the two parallel simulations of step (2) end at a same state \( q_k \) (which is not required to be in \( F \)).

To implement this idea of parallel simulation, we need two tracks in the states of the new machine, one track simulates \( M \) on \( x \) and the other simulates \( M \) on \( y \). That is, our new NFA \( M' \), like the product automaton \( M \times M \), uses states in \( Q \times Q \). More precisely, we define NFA \( M' = (Q', \Sigma, \delta', s, F') \) by 
\[
Q' = (Q \times Q) \cup \{ s \}, \quad F' = \{ \{ q_i, q_i \} \mid q_i \in Q \}
\]
\[
\delta'(s, \varepsilon) = \{ \{ q_0, q_k \} \mid q_k \in F \},
\]
\[
\delta'([q_i, q_j], a) = \{ [q_u, q_s] \mid q_u = \delta(q_i, a), \delta(q_s, b) = q_j \text{ for some } b \in \Sigma \}.
\]

Note that, in an NFA, we actually cannot guess a string \( y \) and verify that \( |x| = |y| \), but we can guess the symbols of \( y \) one at a time. This guessing technique is implemented above in the definition of \( \delta'([q_i, q_j], a) \). From the above discussion, it follows that \( M' \) accepts \( L'_L \).

Proof 2. Let \( M = (Q, \Sigma, \delta, q_0, F) \) be a DFA accepting \( L \), with \( Q = \{ q_0, q_1, \ldots, q_n \} \). The idea of this construction is a combination of the product automaton method and the simulation idea of Example 2.40. That is, we need to nondeterministically guess a state \( q_i \) and a string \( y \), then we simulate \( M_i \) in parallel, on \( x \) from state \( q_0 \) and on \( y \) from state \( q_i \). We accept \( x \) if the first simulation ends at state \( q_i \) and the second simulation ends at a state in \( F \). Since we simulate them in parallel, the guessed string \( y \) must have \( |y| = |x| \).

To implement this idea, we define, for each state \( q_i \in Q \), an NFA \( M_i = (Q, \Sigma, \widehat{\delta}, q_i, F) \) by
\[
\widehat{\delta}(q_j, a) = \{ \delta(q_j, b) \mid b \in \Sigma \}.
\]
(All NFA’s \( M_i \) have the same transition function \( \widehat{\delta} \)). The NFA \( M_i \) then simulates \( M \) on all strings \( y \), starting from state \( q_i \).

Then, we combine machine \( M \) and machines \( M_i \) into a new NFA \( M' \) with \(|Q| + 1 \) tracks, with the first track simulating \( M \) on \( x \) and the \((i + 1)\)st track simulating \( M_i \) on \( y \). Formally, \( M' = (Q', \Sigma, \delta', s, F') \), where \( Q' = Q \times Q \), \( s = [q_0, q_1, q_1, \ldots, q_n] \), \( F' = \{ [q_i, q_j, q_j, \ldots, q_k] \mid 0 \leq i \leq n, q_j \in F \} \), and 
\[
\delta'([q_i, q_j, q_j, \ldots, q_k], a)
= \{ [\delta(q_i, a), q_k, q_k, \ldots, q_k] \mid q_k \in \widehat{\delta}(q_j, a), \text{ for } t = 0, \ldots, n \}.
\]

The second proof above, though creating an NFA with more states than that in the first proof, is a more general proof technique with many applications. The following is another example.

* Example 2.42* Show that if \( L \) is a regular set, so is
\[
L'_L = \{ z \mid (\exists x, y) \mid |x| = |y| = |z|, xy \in L \}.
\]
2.6 Closure Properties of Regular Languages

Proof. We use the second proof technique of the last example to solve this problem.

Let \( M = (Q, \Sigma, \delta, q_0, F) \) be a DFA accepting \( L \), with \( Q = \{ q_1, q_2, \ldots, q_n \} \). For every \( q_i \in Q \), define a DFA \( \widehat{M}_i = (Q, \Sigma, \widehat{\delta}, q_i, F) \) and an NFA \( \widehat{M} = (Q, \Sigma, \widehat{\delta}, q, F) \) by

\[
\widehat{\delta}(q, a) = \{ \delta(q, b) \mid b \in \Sigma \}.
\]

Also let \( \underbar{M} = \widehat{M}_0 \).

Now, we construct an NFA \( M' \) that contains \( 2n + 3 \) tracks simulating, in parallel, \( \underbar{M} \) and \( \widehat{M}_i \), \( i = 0, 1, \ldots, n \). Intuitively, we use \( \underbar{M} \) to simulate \( M \) on \( x \), use \( \widehat{M}_0, \ldots, \widehat{M}_n \) to simulate \( M \) on \( y \), and use \( \widehat{M}_0, \ldots, \widehat{M}_n \) to simulate \( M \) on \( z \). Therefore, after the machine \( M' \) reads input \( z \), if \( \underbar{M} \) is in state \( q_i \), \( \widehat{M}_i \) is in state \( q_j \), and \( \widehat{M}_j \) is in a final state, then \( M' \) accepts input \( z \). That is, \( M' = (Q', \Sigma, \delta', s, F') \), where \( Q' = 2^{n+3} \cdot s = [q_0, \ldots, q_0, q_1, q_2, \ldots, q_n] \),

\[
\delta'([q_{j_0}, \ldots, q_{j_n}, q_0, q_{k_0}, \ldots, q_{k_n}], a) = [\delta(q_{j_0}, a), \ldots, \delta(q_{j_n}, a), t, t_0, \ldots, t_n, t] \in \delta(q_i, a), t_\ell \in \delta(q_{k_\ell}, a) \mid 0, \ldots, n, \ell = 0, \ldots, n, \]

and

\[
F' = \{ [q_{j_0}, \ldots, q_{j_n}, q_1, q_{k_1}, \ldots, q_{k_n}] \mid q_{j_i} \in F \}.
\]

(In the above implementation, the first \( n+1 \) tracks simulate \( \widehat{M}_0 \), \( \widehat{M}_1 \), \ldots, \( \widehat{M}_n \), the \( (n+2) \)nd track simulates \( \underbar{M} \), and the last \( n+1 \) tracks simulate \( \widehat{M}_0, \widehat{M}_1, \ldots, \widehat{M}_n \). So, \( q_{j_\ell} \) denotes the state of DFA \( \widehat{M}_\ell \), \( q_i \) denotes the state of \( \underbar{M} \), and \( q_{k_\ell} \) denotes the state of NFA \( \widehat{M}_\ell \). The condition for the final states \( F' \) reads as follows: we accept the input \( z \) if \( \underbar{M} \) ends at state \( q_i \), \( \widehat{M}_\ell \) ends at some state \( q_\ell \), and \( \widehat{M}_\ell \) ends at a state \( q_{j_\ell} \in F. \)

Using the above proof techniques, we can show that for any rational number \( 0 < r < 1 \), if \( L \) is regular then the language

\[
L_r = \{ x \mid (\exists y) [ |x| = r \cdot |y|, xy \in L] \}
\]

is also regular. Can we prove this type of closure properties for \( L_r \) when \( r \) is not a constant but is a nonlinear function of the length of \( |y| \)? The answer is yes for some simple functions \( r \). However, the proof for such a result depends on certain structural properties of the DFA \( M \) that accepts \( L \), and is not a direct construction from \( M \). In the following, we show a simple example, and leave the general cases as an exercise (see Exercise 10 of this section).

* Example 2.43 Let \( A \) and \( B \) be two regular languages. Show that the language defined by

\[
C(A, B) = \{ x \in A \mid (\exists y) [ |y| = |x|^2, y \in B] \}
\]

is also regular.
Figure 2.44: A DFA over a singleton alphabet has exactly one cycle.

Proof. We prove this result in four steps. First, we consider a simple case where $A = 0^*$ and $B \subseteq 0^*$. Assume that $B$ is accepted by a DFA $M_B$. Since $M_B$ operates on only one symbol $0$, each state in the transition diagram of $M_B$ has exactly one out-edge. It follows that the transition diagram of $M_B$ contains exactly one cycle, as shown in Figure 2.44. Suppose that this cycle contains $c$ edges and that the path from the initial state $s$ to the first state $p$ in the cycle contains $a$ edges, where $c \geq 1$ and $a \geq 0$. We claim that for all integers $m$, with $m^2 \geq a$, $0^m \in C(0^*, B)$ if and only if $0^{m+c} \in C(0^*, B)$.

To prove our claim, we first note that $0^m \in C(0^*, B)$ if and only if $0^{m^2} \in B$, and $0^{m+c} \in C(0^*, B)$ if and only if $0^{(m+c)^2} \in B$. Since $m^2 \geq a$, the computation path of $M_B$ on input $0^{m^2}$ starts from the initial state $s$ and ends at a state $q$ in the cycle. It is clear that, for any $k \geq 0$, the computation path of $M_B$ on input $0^{m^2+kc}$ also ends at state $q$. That is, $0^m \in B$ if and only if $0^{m^2+kc} \in B$ for all $k \geq 0$. It follows that

$$0^m \in C(0^*, B) \iff 0^{m^2} \in B \iff 0^{m^2+(m+c)c} \in B \iff 0^{m+c} \in C(0^*, B).$$

From this claim, we can construct a DFA $M_C$ accepting $C(0^*, B)$ as follows: $M_C$ is a single-cycle DFA as shown in Figure 2.44, with $\lfloor \sqrt{a} \rfloor$ edges from state $s$ to state $p$, and with $c$ edges in the cycle. A state $q$ is a final state if and only if $0^j \in B$ where $j$ is the number of edges in the shortest path from $s$ to $q$. This completes the proof of the first case.

Next, we consider the case where $A = \Sigma^*$ for some alphabet $\Sigma$ and $B \subseteq 0^*$. Using the same argument as above, we can see that $x \in C(\Sigma^*, B)$ if and only if $xz \in C(\Sigma^*, B)$ for all $z$ of length $|z| = c$. Furthermore, $x \in C(\Sigma^*, B)$ if and only if $w \in C(\Sigma^*, B)$ for all strings $w$ of length $|w| = |x|$. Therefore, we can construct $M_C$ for set $C(\Sigma^*, B)$ as in the first case above, except that we replace each edge $\xrightarrow{a}$ by a family of edges $\xrightarrow{a}$ over all symbols $a \in \Sigma$.

In the third step, we assume that $A = \Sigma^*$ and $B \subseteq \Gamma^*$ for some alphabets $\Sigma$ and $\Gamma$. Let $B_0 = \{0^{|w|} \mid x \in B\}$. We note that $B_0$ can be obtained from $B$ by replacing each symbol $a \in \Sigma$ by symbol $0$, and so, by Example 2.35, $B_0$ is regular. In addition, it is easy to verify that $C(\Sigma^*, B_0) = C(\Sigma^*, B)$. It follows from the second case that $C(\Sigma^*, B)$ is regular.

Finally, we consider the general case of $A \subseteq \Sigma^*$ and $B \subseteq \Gamma^*$ for some
2.6 Closure Properties of Regular Languages

alphabets $\Sigma$ and $\Gamma$. We observe that $C(A, B) = A \cap C(\Sigma^*, B)$. From the third case, we know that $C(\Sigma^*, B)$ is regular. Since $A$ is also regular, we conclude that $C(A, B)$ is regular. \hfill \Box

* Example 2.44 Show that if $L$ is regular, so is

$$\text{SQRT}(L) = \{ x \mid (\exists y) \ |y| = |x|^2, xy \in L \}.$$ 

Proof. Since $L$ is regular, there exists a DFA $M = (Q, \Sigma, \delta, s, F)$ accepting $L$. For each state $q \in Q$, let $A_q$ be the set of strings $x$ whose computation path in $M$ starts from $s$ and ends at $q$, and $B_q$ the set of strings $x$ whose computation path in $M$, when started at state $q$, ends at a final state. That is,

$$A_q = \{ x \in \Sigma^* \ | \delta(s, x) = q \},$$

$$B_q = \{ x \in \Sigma^* \ | \delta(q, x) \in F \}.$$ 

Then, we can see that

$$\text{SQRT}(L) = \bigcup_{q \in Q} \{ x \in A_q \mid \exists y \ |y| = |x|^2, y \in B_q \}.$$ 

It is clear that all sets $A_q$ and $B_q$ are regular. Therefore, by Example 2.43, the set $\{ x \in A_q \mid \exists y \ |y| = |x|^2, y \in B_q \}$ is regular for all $q \in Q$. We conclude that $\text{SQRT}(L)$ is also regular. \hfill \Box

Exercise 2.6

1. Prove the following identity:
   (a) $\text{MAX}(L) = L \setminus (L/\Sigma^*)$, where $\text{MAX}(L) = \{ x \in L \mid x$ is not a proper prefix of any string in $L \}.$
   (b) $\text{MIN}(L) = L \setminus (L\Sigma^*).$

2. For any two bits $a, b \in \{0, 1\}$, $a \oplus b$ denotes the exclusive-or of $a$ and $b$; that is, $0 \oplus 0 = 1 \oplus 1 = 0$ and $0 \oplus 1 = 1 \oplus 0 = 1$. For any two binary strings $x$ and $y$ with $|x| = |y|$, $x \oplus y$ denotes the bitwise exclusive-or of $x$ and $y$. For example, if $x = 0011$ and $y = 0101$, then $x \oplus y = 0110$. Let $A = 001(0 + 1)^*$ and $B = (0 + 1)^*100$. Find a regular expression for each of the following languages:
   (a) $A \lor B$.
   (b) $A \oplus B = \{ x \oplus y \mid x \in A, y \in B, |x| = |y| \}$.
   (c) $\{ a_1 b_1 a_2 b_2 \cdots a_n b_n \mid a_1 a_2 \cdots a_n \in A, b_1 b_2 \cdots b_n \in B \}$.

3. Show that if $A$ and $B$ are regular languages, so are the following languages:
   (a) $\{ x \mid xx^R \in A \}$.
(b) \( \{ x \mid x \in A, xx \in A \} \).
(c) \( A_x = \{ y \mid xy \in A \} \), where \( x \) is a fixed string.
(d) \( \{ a_1a_2 \cdots a_{2n} \mid a_1a_4a_3 \cdots a_{2n}a_{2n-1} \in A \} \).
(e) \( \{ a_1a_3 \cdots a_{2n-3}a_{2n-1} \mid a_1a_2 \cdots a_{2n} \in A \} \).
(f) \( \{ a_1b_1a_2b_2 \cdots a_nb_n \mid a_1a_2 \cdots a_n \in A, b_1b_2 \cdots b_n \in B \} \).
(g) \( A \oplus B \).

* 4. Give an alternative proof for Example 2.42 based on the following idea:
   We may simulate NFA \( \mathcal{M} \) on \( xy \) together with \( \mathcal{M}_1 \) on \( z \) by simulating
two moves of \( \mathcal{M} \) and one move of \( \mathcal{M}_1 \) at each step. (Thus, the new NFA
   for \( L_\mathcal{M} \) has only \( n + 2 \) tracks.)

* 5. Show that if \( A \) and \( B \) are regular languages, so are the following:
   
   (a) \( \{ xyz \mid zyx \in A \} \).
   (b) \( \{ xyz \mid xy \in A, y \in B \} \).
   (c) \( \{ x \mid (\exists y) [ |x| = |y|, xy \in A] \} \).
   (d) \( \{ x \mid (\exists y, z) [ |x| = |y| + |z|, xyz \in A] \} \).
   (e) \( \{ y \mid (\exists x) [ |x| = |y|, yxz \in A] \} \).
   (f) \( \{ yz \mid (\exists x) [ |x| = |y|, z, yxz \in A] \} \).
   (g) \( \{ xy \mid (\exists z) [ |x| = |y|, xz \in A] \} \).
   (h) \( \{ x \mid (\exists y, z) [ x = wyz \text{ and } w^{-1} \in A] \} \).

6. Consider a Boolean function \( f(a_1, a_2, \ldots, a_n) \). For any \( n \) binary strings
   \( x_1, x_2, \ldots, x_n \) of equal length, denote by \( f(x_1, x_2, \ldots, x_n) \) the bitwise
   function \( f \) on \( x_1, x_2, \ldots, x_n \). That is, if \( x_i = x_{i_1}x_{i_2} \cdots x_{i_k} \)
   for \( i = 1, 2, \ldots, n \), where each \( x_{i_j} \) is a bit in \( \{0, 1\} \), then \( f(x_1, x_2, \ldots, x_n) \)
is equal to
   \[ f(x_{11}, x_{21}, \ldots, x_{n1}) \cdot f(x_{12}, x_{22}, \ldots, x_{n2}) \cdots f(x_{1k}, x_{2k}, \ldots, x_{nk}). \]
   Show that if languages \( A_1, A_2, \ldots, A_n \) are regular, then language
   \[ \{ f(x_1, x_2, \cdots, x_n) \mid |x_1| = |x_2| = \cdots = |x_n|, \]
   \[ x_1 \in A_1, x_2 \in A_2, \cdots, x_n \in A_n \]
is also regular.

7. In Exercise 3(c) above, show that, for any regular language \( A \), the
   number of distinct \( A_x \)'s is finite. Find an upper bound for this number,
   assuming that \( A \) is accepted by a DFA with \( s \) states.

8. Are the following statements true? Prove or disprove your answer.
   
   (a) If \( A \) is regular and \( A \subseteq B \), then \( B \) is regular.
   (b) If \( A \) is regular and \( B \subseteq A \), then \( B \) is regular.
   (c) If \( A^2 \) is regular, then \( A \) is regular.
2.7 Minimum Deterministic Finite Automata

(d) If $A$ and $AB$ are regular, then $B$ is regular.
(e) If $A$ and $B$ are regular, then $\bigcup_{i=0}^{n-1} (A^i \cap B^i)$ is regular.

9. Show that every regular language in $0^*$ can be represented in the form

$$0^{a_1} + 0^{a_2} + \cdots + 0^{a_k} + (0^{b_1} + 0^{b_2} + \cdots + 0^{b_l})(0^*)^c,$$

for some integer constants $a_1, a_2, \ldots, a_k, b_1, b_2, \ldots, b_l, c$.

**10.** A subset $P$ of nonnegative integers is ultimately periodic if there exists two positive integers $b, p$ such that, for all $m \geq b$, $m \in P$ implies $m + p \in P$. Prove the following statements:

(a) For any regular language $L$, $\{|x| \mid x \in L\}$ is ultimately periodic.
(b) A language $L$ over $\{0\}$ is regular if and only if $\{|x| \mid x \in L\}$ is ultimately periodic.
(c) If $f$ is a mapping from integers to integers such that $f^{-1}(P)$ is ultimately periodic for every ultimately periodic set $P$, then the set $\{x \in A \mid (\exists y) \ |y| = f(|x|), y \in B\}$ is regular for every pair of regular sets $A$ and $B$.
(d) If $f$ is a mapping from integers to integers such that $f^{-1}(P)$ is ultimately periodic for every ultimately periodic set $P$, then the set $\{x \mid (\exists y) \ |y| = f(|x|), xy \in L\}$ is regular for every regular set $L$.

**11.** Apply Exercise 10(d) above to prove the following results:

(a) If $L$ is a regular language, then so is $\{x \mid (\exists y) \ |y| = 2|x|, xy \in L\}$. [Hint: Use Fermat’s theorem, which states that for any odd integer $m \geq 3$, there exists an integer $\varphi(m)$ such that $2^{\varphi(m)} \equiv 1 \mod m$.]
(b) If $L$ is a regular language, then so is $\{x \mid (\exists y) \ |y| = |x|, xy \in L\}$.}

2.7 Minimum Deterministic Finite Automata

In the past few sections, we have introduced a number of ways to construct a DFA for a given regular language. These methods, however, often result in DFA’s with a large number of states. For instance, if we are given a regular expression $r$, we can find a DFA for $L(r)$ by first finding an NFA $M_1$ from $r$, using the method developed in Section 1.3, and then converting $M_1$ to a DFA $M_2$ by the subset construction method of Section 2.4. What is the size of the DFA $M_2$ compared with the size of $r$ or $M_1$? In general, the first step creates an NFA of $O(n)$ states from a regular expression of $n$ symbols, and the second step creates a DFA of up to $2^n$ states from an NFA of $n$ states (cf. Example 2.27). Apparently, such a DFA is too big for a reasonably big number $n$.

As another example, consider the NFA $M'$ constructed from a given DFA $M$ in Example 2.40. Suppose the given DFA $M$ has $n$ states, then $M'$ has
$2n^2 + 2$ states. If we convert this NFA to an equivalent DFA, the new DFA would have, in the worst case, $2^{O(n^2)}$ states.

Whereas this large size of DFA’s is unavoidable in some cases, many DFA’s constructed this way can be reduced to smaller, equivalent DFA’s. In this section, we show how to find, for a given regular language, a DFA with the minimum number of states. Such a DFA is called a minimum DFA.

To begin with, we define, for any language $L \subseteq \Sigma^*$, a relation $R_L$ on $\Sigma^*$ as follows:

$$x R_L y \text{ if and only if } (\forall w) [xw \in L \iff yw \in L].$$

This relation $R_L$ is an equivalence relation. That is, it satisfies the following properties:

1. Reflexivity: $(\forall x \in \Sigma^*) [x R_L x]$;
2. Symmetry: $(\forall x, y \in \Sigma^*) [x R_L y \Rightarrow y R_L x]$;
3. Transitivity: $(\forall x, y, z \in \Sigma^*) [x R_L y, y R_L z \Rightarrow x R_L z]$.

Recall that for any equivalence relation $R$ on a set $S$ and for any $x \in S$, the class

$$[x]_R = \{y \in S \mid x Ry\}$$

is called an equivalence class containing $x$. Every equivalence relation $R$ on a set $S$ divides $S$ into disjoint equivalence classes. The number of equivalence classes is called the index of $R$, and is denoted by Index$(R)$.

**Example 2.45** Let $L = \{x \in (0 + 1)^* \mid |x| \text{ is odd}\}$. Find all equivalence classes of $R_L$.

**Solution.** Note that $x R_L y$ if and only if $|x| - |y|$ is even. Therefore, there are two equivalence classes

$$[e]_{R_L} = \{x \in (0 + 1)^* \mid |x| \text{ is even}\}$$

$$[0]_{R_L} = \{x \in (0 + 1)^* \mid |x| \text{ is odd}\}. \quad \square$$

**Example 2.46** Let $L$ be the set of nonempty binary strings starting and ending with the same symbol. Find all equivalence classes of $R_L$.

**Solution.** We first show that $x R_L y$ if and only if $x$ and $y$ start with the same symbol and end with the same symbol.

For the forward direction, we first assume that $x$ and $y$ start with different symbols; for instance, $x$ starts with 0 and $y$ starts with 1. Then, $x0 \not\in L$ and $y0 \not\in L$. This implies that $x$ and $y$ do not satisfy the relation $x R_L y$. Or, equivalently, $x R_L y$ implies that $x$ and $y$ start with the same symbol. Next, note that $x R_L y$ implies that $x \in L \iff y \in L$. Since $x$ and $y$ start with the same symbol, they must end with the same symbol.

For the backward direction, we assume that $x$ and $y$ start with the same symbol and end with the same symbol. Since $x$ and $y$ start with the same
symbol, $xw \in L \Leftrightarrow yw \in L$ for all $w \in (0+1)^+$. Moreover, since $x$ and $y$ also
end with the same symbol, $x \in L \Leftrightarrow y \in L$. Therefore, for any $w \in (0+1)^*$,
$xw \in L \Leftrightarrow yw \in L$. That is, $xR_L y$.
From the above characterization of the relation $R_L$, we can easily see that
$R_L$ has five equivalence classes:

$$
\begin{align*}
[\varepsilon]_{R_L} &= \varepsilon, \\
[0]_{R_L} &= 0 + 0(0+1)^*0, \\
[01]_{R_L} &= 0(0+1)^*1, \\
[10]_{R_L} &= 1(0+1)^*0.
\end{align*}
$$

The following lemma relates, for a regular language $L$, $\text{Index}(R_L)$ with the
number of states in a DFA $M$ that accepts $L$.

**Lemma 2.47** Assume that $L$ is accepted by a DFA $M = (Q, \Sigma, s, \delta, F)$. Then,
for any strings $x, y \in \Sigma^*$, if $\delta(s, x) = \delta(y, y)$ then $xR_L y$.

**Proof.** Suppose that $\delta(s, x) = \delta(s, y)$ in $M$. Then, for any $w \in \Sigma^*,$

$$
\delta(s, xw) = \delta(\delta(s, x), w) = \delta(\delta(s, y), w) = \delta(s, yw).
$$

Therefore, for any $w \in \Sigma^*$, $xw \in L$ if and only if $yw \in L$. $\Box$

The above lemma shows that if $L$ is accepted by a DFA $M$ of $n$ states,
then all strings with the same ending state are in the same equivalence class
of $R_L$. Therefore, $\text{Index}(R_L)$ is bounded by $n$ and is finite. Thus, if we can
find a DFA accepting language $L$ with exactly $\text{Index}(R_L)$ states, then this
DFA must be a minimum DFA. The following theorem shows that, in fact,
this property characterizes the minimum DFA’s.

**Theorem 2.48** For any regular language $L$, its minimum DFA has exactly
$\text{Index}(R_L)$ states.

**Proof.** Assume that $L$ is a language over alphabet $\Sigma$. Define a DFA $M =
(Q, \Sigma, \delta, s, F)$ as follows:

1. $Q = \{[x]_{R_L} \mid x \in \Sigma^*\}$;
2. $\delta([x]_{R_L}, a) = [xa]_{R_L}$;
3. $s = [\varepsilon]_{R_L}$;
4. $F = \{[x]_{R_L} \mid x \in L\}$.

From the discussion above, we know that if $L$ is regular then $\text{Index}(R_L) < \infty$.
It follows that $Q$ is a finite set. In addition, the function $\delta$ is well defined because

$$
[x]_{R_L} = [y]_{R_L} \Rightarrow [xa]_{R_L} = [ya]_{R_L} \Rightarrow \delta([x]_{R_L}, a) = [xa]_{R_L} = [ya]_{R_L} = \delta([y]_{R_L}, a).
$$
By a simple induction, we can extend this property to \( \delta([x]_{R_L}, x) = [x]_{R_L} \), for all \( x \in \Sigma^* \). This shows that \( L(M) = L \):

\[
x \in L \iff [x]_{R_L} \in F \iff \delta([x]_{R_L}, x) \in F \iff M \text{ accepts } x.
\]

Since \( M \) has \( \text{Index}(R_L) \) states, it is, by Lemma 2.47, a minimum DFA for \( L \).

\[ \square \]

**Corollary 2.49** A language \( L \) is regular if and only if \( \text{Index}(R_L) < \infty \).

**Proof.** Note that, in the proof of Theorem 2.48, we did not use the fact that \( L \) is regular. That is, we can construct the minimum DFA for any language \( L \) as long as \( \text{Index}(R_L) < \infty \). Thus, such languages \( L \) must be regular. \( \square \)

**Example 2.50** Show that \( L \) is regular if and only if there exists a positive integer \( k \) such that \( x \in R_L y \) if and only if for every \( z \in \Sigma^* \) with \( |z| \leq k \), \( xz \in L \iff yz \in L \).

**Proof.** First, assume that \( L \) is regular and is accepted by a DFA \( M = (Q, \Sigma, \delta, s, F) \). Let \( k = |Q|^2 - 1 \). We claim that if \( xz \in L \iff yz \in L \) for all \( z \) of length \( |z| \leq k \), then \( x \in R_L y \). To see this, let us consider the product DFA \( M^* = M \times M \). For every string \( w \) of length \( |w| > k \), the computation path of \( M^* \) on \( w \), starting from state \( q_1^* = [\delta(s, x), \delta(s, y)] \) to \( q_2^* = [\delta(s, xw), \delta(s, yw)] \), contains at least \( |Q|^2 + 1 \) states. Therefore, it contains some cycles in the transition diagram of \( M^* \). Let us eliminate these cycles and keep only a simple path from \( q_1^* \) to \( q_2^* \). This simple path corresponds to the computation path of \( M^* \) on a string \( z \) of length \( |z| \leq k \), from \( q_1^* \) to \( q_2^* \). (E.g., we show in Figure 2.45 the computation path of a string \( w = w_1w_2w_3w_4w_5w_6 \), with \( w_2, w_4 \) and \( w_5 \) associated to the two cycles. We eliminate these cycles to form a new string \( z = w_1w_3w_6 \) which is associated with the simple path.) That is, \( q_2^* = [\delta(s, xz), \delta(s, yz)] \); or, \( \delta(s, xw) = \delta(s, xz) \) and \( \delta(s, yw) = \delta(s, yz) \). Now, by the assumption, \( xz \in L \iff yz \in L \); that is, two states \( \delta(s, xz) \) and \( \delta(s, yz) \) both are in \( F \) or both are not in \( F \). Therefore, \( xw \in L \iff yw \in L \). It follows that \( x \in R_L y \).

Conversely, suppose there exists such a constant \( k \). Note that each \( z \) can divide \( \Sigma^* \) into at most two parts: \( \{ x \mid xz \in L \} \) and \( \{ x \mid xz \notin L \} \). Therefore, \( R_L \) has at most \( 2^{|\Sigma|+\cdots+|\Sigma|} \) equivalence classes. Hence, by Corollary 2.49, \( L \) is regular. \( \square \)
2.7 Minimum Deterministic Finite Automata

\[
\begin{array}{c}
\varepsilon \\
0 \quad 1
\end{array}
\longrightarrow
\begin{array}{c}
0 \quad 1 \\
\varepsilon
\end{array}
\quad \text{01}
\]

**Figure 2.46:** Solution to Example 2.51.

**Example 2.51** Find the minimum DFA for language \((0 + 1)^*01\).

**Solution.** Let us denote this language by \(L\), and study the equivalence classes of \(R_L\). First, we consider the class \([\varepsilon]_{R_L}\). Note that

\[
[\varepsilon]_{R_L} = \{ x \in \{0, 1\}^* | (\forall w) [xw \in (0 + 1)^*01 \Leftrightarrow w \in (0 + 1)^*01] \}.
\]

Therefore, if \(x \in [\varepsilon]_{R_L}\), then \(x\) must not end with 0 or 01, for otherwise \(x1\) or \(x\varepsilon\) would be in \((0 + 1)^*01\), which would imply that 1 or \(\varepsilon\) is in \((0 + 1)^*01\). Conversely, if \(x\) does not end with 0 or 01, then it is clearly in \([\varepsilon]_{R_L}\). So, we know that \([\varepsilon]_{R_L}\) contains strings \(\varepsilon, 1\) and all strings ending with 11; or,

\[
[\varepsilon]_{R_L} = \varepsilon + 1 + (0 + 1)^*11.
\]

Next, we consider

\[
[0]_{R_L} = \{ x \in \{0, 1\}^* | (\forall w) [0w \in (0 + 1)^*01 \Leftrightarrow w \in (0 + 1)^*01] \}.
\]

Since, for \(w = 1, 0w \in (0 + 1)^*01\), we know that \(x \in [0]_{R_L}\) must end with 0. Conversely, it is clear that if \(x\) ends with 0, then \(x \in [0]_{R_L}\). So, we get

\[
[0]_{R_L} = (0 + 1)^*0.
\]

Now, we note that \(1 \in [\varepsilon]_{R_L}\) and so \([1]_{R_L} = [\varepsilon]_{R_L}\); and \(00 \in [0]_{R_L}\) and so \([00]_{R_L} = [0]_{R_L}\). Therefore, we jump to

\[
[01]_{R_L} = \{ x \in \{0, 1\}^* | (\forall w) [01w \in (0 + 1)^*01 \Leftrightarrow w \in (0 + 1)^*01] \}.
\]

By the same argument as that for \([0]_{R_L}\), we can see that \([01]_{R_L}\) contains all strings \(x\) which end with 01; or,

\[
[01]_{R_L} = (0 + 1)^*01.
\]

Finally, we note that \([10]_{R_L} = [0]_{R_L}\), \([11]_{R_L} = [\varepsilon]_{R_L}\), and for all strings \(w\) of length \(|w| \geq 3\), \(wR_Lx\) for some \(x\) of length \(|x| \leq 2\). Therefore, there are only three equivalence classes: \([\varepsilon]_{R_L}, [0]_{R_L}\) and \([01]_{R_L}\), and the minimum DFA for \(L\) has only three states. We show it in Figure 2.46. \(\square\)
Example 2.52 Construct the minimum DFA for language \((0+1)^*0(0+1)^9\).

Solution. This language \(L\) contains all strings \(w\) of length at least 10 whose tenth rightmost bit is 0. We will show that \(R_L\) defines the following set of equivalence classes:

\[ Q = \{ [e]_{R_L} \} \cup \{ [0x]_{R_L} \mid x \in \{0,1\}^*, |x| \leq 9 \}. \]

First, we show that all equivalence classes in \(Q\) are distinct. Since, for any \(x\) with \(|x| \leq 9\), \(0^9|\sim| \not\in L\) and \(0x0^9|\sim| \not\in L\), the strings \(\varepsilon\) and \(0\varepsilon\) are not in the same equivalence class. Now, we look at \(0x\) and \(0y\) with \(x \neq y\). We consider three cases:

Case 1. \(|x| < |y|\). Since \(0x0^9|\sim| \not\in L\) and \(0y0^9|\sim| \not\in L\), strings \(0x\) and \(0y\) are not in the same equivalence class.

Case 2. \(|x| > |y|\). Symmetric to Case 1.

Case 3. \(|x| = |y|\). We can write \(x = x_1x_2 \cdots x_k\) and \(y = y_1y_2 \cdots y_k\) for some \(k \geq 1\), where each \(x_i\) and each \(y_j\) is a 0 or 1. Since \(x \neq y\), there exists an integer \(i\), \(1 \leq i \leq k\), such that \(x_i \neq y_i\). Without loss of generality, assume that \(x_i = 0\) and \(y_i = 1\). In this case, \(0x0^9|\sim| \not\in L\) and \(0y0^9|\sim| \not\in L\).

Thus, \(0x\) and \(0y\) are not in the same equivalence class.

Next, we show that every binary string is in one of the equivalence classes in \(Q\). For each string \(w \in (0+1)^*\), consider its suffix \(u\) of length \(\min\{|w|, 10\}\).

First, if \(u \in 1^*\), then we observe that \(wy \in L\) for some \(y \in \{0,1\}^*\) if and only if \(|y| \geq 10\) and the tenth rightmost bit of \(y\) is 0. (If \(|y| < 10\), then either \(|wy| < 10\) or the tenth rightmost bit of \(wy\) is a 0 in \(u\) and, hence, equal to 1.) It follows that for any string \(y \in \{0,1\}^*\), \(wy \in L\) if and only if \(y \in L\), and so \(w R_L \varepsilon\). Next, if \(u = 1^0x\) for some \(j \geq 0\) and \(x \in \{0,1\}^*\) then, by following the same argument as above, we can see that \(wy \in L\) if and only if \(0y \in L\). Thus, \(u R_L 0x\).

Now, we can use the above analysis to construct the minimum DFA \(M = (Q, \{0,1\}, \delta, [e]_{R_L}, F)\) for \(L\), where \(Q\) is the set defined above,

\[ F = \{ [0x]_{R_L} \in Q \mid |x| = 9 \}, \]

and the transition function \(\delta\) can be described as follows:

1. \(\delta([e]_{R_L}, 0) = [0]_{R_L}, \delta([e]_{R_L}, 1) = [e]_{R_L}.\)

2. If \(|x| < 9\), then, for \(a \in \{0,1\}, \delta([0x]_{R_L}, a) = [0xa]_{R_L}.\)

3. If \(x = 1^9\) then \(\delta([0x]_{R_L}, 0) = [0]_{R_L}, \delta([0x]_{R_L}, 1) = [e]_{R_L}.\)

4. If \(|x| = 9\) and \(x = 1^0y\) for some \(y\), with \(0 \leq i \leq 8\), then \(\delta([0x]_{R_L}, a) = [0ya]_{R_L}, \) for \(a \in \{0,1\}.\)

From the above example, we can see that the minimum DFA for language \((0+1)^*0(0+1)^{9-1}\) must contain \(2^9\) states. However, it is easy to construct an NFA for the same language with \(n + 1\) states. Thus, the exponential size increase of the subset construction is unavoidable in these cases. It also
shows that, although NFA’s and DFA’s have the same computational power for recognizing languages, they are different in term of computational complexity.

In general, finding the equivalence classes of \( R_L \) for a given language \( L \) (from its regular expression or an informal description) may not be easy. However, if we are given a DFA \( M \) for \( L \), then there is a simple method to find the equivalent minimum DFA.

The basic idea of this method is the simple observation of Lemma 2.47: For any DFA \( M \), two input strings which end at the same state of \( M \) must belong to the same equivalence class of \( R_L \). That is, for any DFA \( M = (Q, \Sigma, \delta, s, F) \) accepting \( L \), if we define, for each \( q \in Q, \quad S_q = \{ x \in \Sigma^* \mid \delta(s, x) = q \} \), then every \( S_q \) is contained in a single equivalence class of \( R_L \). Thus, we may define a new equivalence relation \( R_L^* \) on \( Q \) as follows:

\[
p R_L^* q \iff S_p \text{ and } S_q \text{ are in the same equivalence class of } R_L \iff (\forall w) [\delta(p, w) \in F \iff \delta(q, w) \in F].
\]

Each equivalence class of \( R_L \) corresponds to an equivalence class of \( R_L^* \) in the following way:

\[ [x]_{R_L} = \bigcup \{ S_q \mid q \in [\delta(s, x)]_{R_L^*} \} \]

From this relation, we can build the minimum DFA \( M^* = (Q^*, \Sigma^*, \delta^*, s^*, F^*) \) for \( L \) as follows:

1. \( Q^* = \{ [q]_{R_L^*} \mid q \in Q \} \);
2. \( \delta^*([q]_{R_L^*}, a) = [\delta(q, a)]_{R_L^*} \)
3. \( s^* = [s]_{R_L^*} \);
4. \( F^* = \{ [f]_{R_L^*} \mid f \in F \} \).

Thus, to construct the minimum DFA \( M^* \) which is equivalent to a given DFA \( M \), all we need to do is to compute the equivalence classes of \( R_L^* \). We demonstrate three different methods in the next example.

**Example 2.53** Find the minimum DFA equivalent to the DFA \( M \) of Figure 2.47.

**Solution 1.** Let \( M = (Q, \Sigma, \delta, q_0, F) \), where \( Q = \{ q_0, q_1, \ldots, q_L \} \), and let \( L = L(M) \). We note that states \( q_i \) and \( q_j \) are equivalent under \( R_L^* \) if and only if for all strings \( w \), \( \delta(q_i, w) \) and \( \delta(q_j, w) \) both are in \( F \) or both are not in \( F \). Therefore, to find the relation \( R_L^* \) between any two states, we construct a graph \( G \) as follows: Each vertex is an unordered pair \((q_i, q_j)\) of states. Let \( U \) be the set of vertices \((q_i, q_j)\) with one vertex \( q_i \in F \) and the other vertex \( q_j \notin F \). For each vertex \((q_i, q_j) \notin U \), with \( i \neq j \), we draw edges

\[
(q_i, q_j) \xrightarrow{a} (\delta(q_i, a), \delta(q_j, a))
\]

for all \( a \in \Sigma \). Then, it is clear that \( q_i R_L^* q_j \) if and only if there is no path in \( G \) from \((q_i, q_j)\) to a vertex in \( U \).
For this example, we show the corresponding graph $G$ in Figure 2.48. (We only show vertices which are reachable from vertices not in $U$.) The vertex $(q_4, q_5)$ is in $U$ and is denoted by double circles. All vertices from which there are paths going to a vertex in $U$ have been marked with X. Thus, we have $q_0 R_L^* q_1$ and $q_2 R_L^* q_3$. The resulting minimum DFA is shown in Figure 2.49.

Solution 2. In the above solution, the critical step is to determine, from a given pair $(q_i, q_j)$ of states, whether there is a path in $G$ from it to a vertex in $U$. This question can be solved using a table buildup method. We create a table of pairs $(q_i, q_j)$. Initially, we mark all pairs in $U$ with value 0. At stage $k > 0$, we mark each unmarked pair $(q_i, q_j)$ with value $k$ if there is an edge from it to a pair with mark $k - 1$; that is, we mark pair $(q_i, q_j)$ with value $k$ if
there is an $a \in \Sigma$ such that $(\delta(q_i, a), \delta(q_j, a))$ has the value $k - 1$. We repeat this process until no new pairs are marked in a stage. The pairs $(q_i, q_j)$ which remain unmarked must then satisfy $q_i R_L^* q_j$.

For this example, the above table buildup process is shown in Figure 2.50. The pairs of states which remain unmarked are $(q_0, q_1)$ and $(q_2, q_3)$.  

**Solution 3.** Instead of building a table to check the relation $R_L^*$, we can also do it directly on the transition diagram of $M$. Initially, we divide all states into two blocks $F$ and $Q - F$. Then, at each subsequent stage, we check every block $S$ of states. For every $a \in \Sigma$, we consider $\delta(q_i, a)$ for all $q_i \in S$. If they do not all belong to the same block, then we divide $S$ into smaller blocks according to their destination blocks; that is, if $\delta(q_i, a)$ and $\delta(q_j, a)$ belong to the same block but $\delta(q_k, a)$ belongs to a different block, then $q_i$ and $q_j$ remain in the same smaller block and $q_k$ belongs to a different smaller block. We repeat this process until no block of states can be divided further.

The whole process on this example is shown in Figure 2.51.

Corollary 2.49 gives us a simple characterization of regular languages. It is very useful in the analysis of the structure of regular languages. In the following two examples, we show how to apply it to prove that a given language
is not regular.

**Example 2.54** Show that the following language is not regular:

\[ L = \{ xx^R \mid x \in \{0,1\}^* \}. \]

*Proof.* For any \( x, y \in 0^*1 \) with \( x \neq y \), we know that \( xx^R \in L \) and \( yy^R \notin L \). Thus, any two different strings in \( 0^*1 \) must belong to two different equivalence classes of \( R_L \). This means that \( \text{Index}(R_L) = \infty \). Thus, by Corollary 2.49, \( L \) is not regular. \( \square \)

**Example 2.55** Show that \( L = \{ 0^m 1^n \mid \gcd(m,n) = 1 \} \) is not regular.

*Proof.* For any two different primes \( p \) and \( q \), \( 0^p 1^q \notin L \) and \( 0^q 1^p \in L \). Therefore, \( 0^p \) and \( 0^q \) are not in the same equivalence class of \( R_L \). Since there are an infinite number of primes, we have \( \text{Index}(R_L) = \infty \). Thus, \( L \) is not regular. \( \square \)

**Example 2.56** For any language \( L \), define an equivalence relation \( D_L \) by

\[ x D_L y \iff (\forall u)(\forall w) [uxw \in L \Leftrightarrow uyw \in L]. \]

Show that \( L \) is regular if and only if \( \text{Index}(D_L) < \infty \).

*Proof.* Clearly, for any strings \( x \) and \( y \), \( x D_L y \) implies \( x R_L y \). Thus, \( \text{Index}(R_L) \leq \text{Index}(D_L) \). It follows that \( \text{Index}(D_L) < \infty \) implies that \( L \) is regular.

Conversely, assume that \( L \) is regular. To show that \( \text{Index}(D_L) < \infty \), we consider a DFA \( M = (Q, \Sigma, \delta, q_0, F) \) accepting \( L \) and define an equivalence relation \( D_M \) on \( \Sigma^* \) as follows:

\[ x D_M y \iff (\forall q_i \in Q) [\delta(q_i, x) = \delta(q_i, y)]. \]

First, note that \( x D_M y \) implies \( x D_L y \). It follows that \( \text{Index}(D_L) \leq \text{Index}(D_M) \). Assume that the state set \( Q \) of \( M \) is \( \{ q_0, q_1, \ldots, q_{n-1} \} \). We claim that \( D_M \) has at most \( n^n \) equivalence classes. To see this, we observe that every equivalence class \( [x]_{D_M} \) can be represented in the following way:

\[ [x]_{D_M} = \bigcap_{i=0}^{n-1} \{ y \mid \delta(q_i, x) = \delta(q_i, y) \}. \]

That is, \( [x]_{D_M} \) is uniquely determined by the following sequence of \( n \) states:

\[ \overline{Q}_x = (\delta(q_0, x), \delta(q_1, x), \ldots, \delta(q_{n-1}, x)). \]

If two strings \( x \) and \( y \) have the same sequence \( \overline{Q}_x = \overline{Q}_y \), then \( x D_M y \). Thus, the number of equivalence classes of the relation \( D_M \) is the number of possible sequences \( \overline{Q}_x \), which is bounded by \( n^n \). Therefore, we have \( \text{Index}(D_L) \leq \text{Index}(D_M) \leq n^n < \infty \). \( \square \)
Exercise 2.7

1. Find all equivalence classes of $R_L$ for the following languages:
   (a) $(0+1)^*01(0+1)^*$.
   (b) $(00+11)(0+1)^*$.
   (c) $011(0+1)^*001$.
   (d) The set of binary strings in which each block of four symbols have at least two 0's.
   (e) $\{x \in \{0,1\}^* \mid \#a(x) = \#b(x)\}$, where $\#a(w)$ is the number of occurrences of symbol $a$ in $w$.

2. For each of the following languages $L$, show that $\text{Index}(L) = \infty$ and so $L$ is not regular.
   (a) $\{0^n1^n \mid 0 \leq m \leq n\}$.
   (b) $\{0^n1^n0^n+m \mid n, m \geq 0\}$.
   (c) $\{ww \mid w \in \{0,1\}^*\}$.
   (d) $\{xx^Rw \mid x, w \in \{0,1\}^+\}$.

3. For each of the following languages $L$, show that no two strings can be in the same equivalence class of $R_L$.
   (a) $\{0^p \mid p \text{ is a prime}\}$.
   (b) $\{0^{n^2} \mid n \geq 0\}$.

4. Construct the minimum DFA’s for languages accepted by DFA’s in Figure 2.52(a) and 2.52(b).

5. Construct a minimum DFA equivalent to the NFA of Figure 2.53.
2.8 Pumping Lemmas

Not all languages are regular. In fact, there are uncountably many languages over an alphabet $\Sigma$ (cf. Example 5.18), but only countably many of them are regular (there are at most $c^n$ possible regular expressions of length $n$ for some constant $c > 0$). Therefore, most languages are not regular.

In the last section, we have used the simple characterization of Corollary 2.49 for regular languages to prove that some languages are not regular. However, this method involves the analysis of equivalence classes of the relation $R_L$, and is sometimes difficult to apply. In this section, we introduce another necessary condition for regularity of languages which can be used to prove that a language is nonregular. In the following, we write, for any string $v$, $v^*$ to denote the set $\{v\}^*$.

**Lemma 2.57** (Pumping Lemma). If a language $L$ is accepted by a DFA $M$ with $s$ states, then every string $x$ in $L$ with $|x| \geq s$ can be written as $x = uvw$ such that $v \neq \varepsilon$ and $uw^*w \subseteq L$.

**Proof.** Consider the transition diagram of $M$. Since $x \in L$, the computation path $\pi$ of $x$ starts from the initial state $q_0$ and ends at a final state $q_f$. The concatenation of the labels over the path $\pi$ is exactly the string $x$. The path $\pi$ has exactly $|x|$ edges because each edge is labeled by a symbol. Thus, the path $\pi$ contains a vertex sequence of $|x| + 1$ elements. Since $|x| \geq s$, some state $q_i$ occurs more than once in the sequence. Break the path $\pi$ into three subpaths at the first and second occurrences of $q_i$. That is, the first subpath is from state $q_0$ to the first occurrence of $q_i$, the second subpath is a cycle from the first $q_i$ to the second $q_i$, and the third subpath is from the second $q_i$ to $q_f$ (see Figure 2.54).

Let $u$, $v$, and $w$ be the concatenations of the labels of the three subpaths, respectively. Then, $x = uvw$, and $v \neq \varepsilon$. Since $v$ is associated with a cycle, we

**Figure 2.54:** The path $(q_0, \cdots, q_i, \cdots, q_i, \cdots, q_f)$. 
## 2.8 Pumping Lemmas

also have $uw^*w \subseteq L$. (E.g., $uv^2w$ is in $L$ because $\delta(q_i, uv^2w) = \delta(q_i, vw) = \delta(q_i, w) = q_f$.)

Now, for a given language $L$, if we can prove that the necessary condition of the pumping lemma does not hold with respect to any $s > 0$, then $L$ is not regular.

### Example 2.58

$\{0^p \mid p \text{ is a prime} \}$ is not a regular language.

**Proof.** By way of contradiction, assume that $L = \{0^p \mid p \text{ is a prime number} \}$ is regular. Then, $L$ is accepted by a DFA $M$. Let $s$ be the number of states in $M$. Consider a prime number $p > s$. Note that $0^p \in L$ and $|0^p| = p > s$. Therefore, by the pumping lemma, $0^p$ can be written as $0^p = uvw$ such that $v \neq \varepsilon$ and $|uw^*w| \subseteq L$.

Let $i = |u| + |v|$ and $j = |v|$. Then, the condition $uv^*w \subseteq L$ means that, for any $k \geq 0$, $uv^kw = 0^{i+kj} \in L$. Or, equivalently, for any $k \geq 0$, $i + kj$ is a prime. In particular, when $k = 0$, it means that $i$ is a prime. So, $i \geq 2$. When $k = 1$, this means that $i(1 + j)$ is a prime. However, since $v \neq \varepsilon$, we have $j = |v| \geq 1$, and so $i(1 + j)$ is not a prime. This is a contradiction. □

Note that in the above example, the underlying language is over a singleton alphabet $\{0\}$. For languages over an alphabet with more than one symbol, the above pumping lemma is not convenient and sometimes even not sufficient. For instance, consider the language $\{0^n1^n \mid n \geq 0 \}$. When we follow the argument in the proof of the above example, we get $0^n1^n = uvw$ for some strings $u, v, w$. There are three possible cases for $v$: (1) $v$ contains only symbol 0; (2) $v$ contains only symbol 1; and (3) $v$ contains both symbols 0 and 1. A complete proof needs to produce a contradiction for each case. This makes the proof more complicated and tedious. The following stronger pumping lemma is a nice tool to avoid this problem.

### Lemma 2.59 (Pumping Lemma, Stronger Form)

If a language $L$ is accepted by a DFA $M$ with $s$ states, then for any string $a \in L$ with $|a| \geq s$ and any way of breaking $a$ into $a = xyz$ with $|y| \geq s$, $y$ can be written as $y = uvw$ such that $v \neq \varepsilon$ and $xu^*wz \subseteq L$.

**Proof.** Consider the transition diagram of $M$. Since $a = xyz \in L$, the computation path $\pi$ of $a$ goes from the initial state $q_0$ to a final state $q_f$. The path $\pi$ can be divided into three subpaths, associated with the strings $x$, $y$, and $z$, respectively. Assume that the second subpath $\pi_2$, which is associated with $y$, is from state $q_1$ to state $q_2$. Then, $\pi_2$ has exactly $|y|$ edges and, by the same argument as in the proof of Lemma 2.57, $\pi_2$ can be further divided into three subpaths, with the middle one being a cycle (see Figure 2.55). Let $u$, $v$, and $w$ be the strings associated with the three subpaths, respectively. Then, $y = uvw$ and $v \neq \varepsilon$. Since $v$ is associated with a cycle, we also have $xu^*wz \subseteq L$. □
Example 2.60 \( \{0^n1^n \mid n \geq 0 \} \) is not a regular language.

Proof. By way of contradiction, assume that \( L = \{0^n1^n \mid n \geq 0 \} \) is regular. Then, \( L \) is accepted by a DFA \( M \). Let \( s \) be the number of states in \( M \). Consider a string \( \alpha = 0^s1^s \in L \). Choose \( x = \varepsilon \), \( y = 0^s \), and \( z = 1^s \). Note that \( |y| = |0^s| = s \). By the pumping lemma, \( \alpha \) can be written as \( \alpha = xuvwz \) such that \( v \neq \varepsilon \) and \( xuv^kuz \subseteq L \). This means that for any \( k \geq 0 \), \( xuv^kuz \in L \). When \( k = 0 \), this means that \( xuvz = 0^{|v|}1^{|v|} \in L \). However, since \( v \neq \varepsilon \), we have \( s - |v| < s \), contradicting the definition of \( L \).

From the above example, we can summarize in the following how to prove, by the pumping lemma, that a given language \( L \) is not regular:

1. Assume that \( L \) is accepted by a DFA \( M \) of \( s \) states.
2. Select a string \( \alpha \in L \), with \( |\alpha| \geq s \).
3. Divide \( \alpha \) into three parts \( \alpha = xyz \) with \( |y| \geq s \).
4. For any way of dividing \( y \) into three parts \( y = uvw \) with \( v \neq \varepsilon \), argue that \( xuv^kuz \notin L \) for some \( k \geq 0 \).

Since \( s \) is the size of the DFA \( M \) accepting \( L \), and since the DFA \( M \) is unknown to us, we do not know how large \( s \) is. Therefore, steps (2), (3) and (4) must work for all positive integers \( s \). Similarly, the breakdown of \( y \) into \( y = uvw \) depends on the unknown DFA \( M \), and so is unknown to us. Therefore, we must argue, in step (4), against all possible way of dividing \( y \) into \( uvw \), as long as \( v \neq \varepsilon \).

On the other hand, we are free to select the strings \( x, y, z \), as long as \( \alpha = xyz \in L \) and \( |y| \geq s \). Indeed, the choice of \( x = \varepsilon \), \( y = 0^s \) and \( z = 0^s \) in the above example made the proof simple. (The reader may verify this claim by trying the choice of \( \alpha = 0^21^2 \), \( x = 0^2 \), \( y = 0^21^2 \) and \( z = 1^2 \) to see how complicated the corresponding proof is.) In general, the main difficulty of using the pumping lemma to prove a language nonregular is to determine which strings \( x, y, z \) are to be used in the proof. The next two examples illustrate this point.

Example 2.61 Show that \( L = \{\beta \beta^R \mid \beta \in \{0, 1\}^+ \} \) is not a regular language.

Proof. By way of contradiction, assume that \( L \) is a regular language, accepted by a DFA \( M \) of \( s > 0 \) states. Following the idea of Example 2.60, we select a
string $a = 0^*11^0 \in L$, and let $x = \epsilon$, $y = 0^*$, and $z = 110^*$. By the pumping lemma, $y$ can be written as $y = uv^iz$ such that $v \neq \epsilon$ and $xuv^iz \subseteq L$. This means that for any $k \geq 0$, $xuv^iz \subseteq L$. When $k = 0$, this means that $xuv = 0^{s-k-1}11^0 \in L$. However, we observe that $0^{s-k-1}11^0$ is not of the form $\beta \beta^R$; Since $|v| \geq 1$, either the string $0^{s-k-1}11^0$ is of an odd length (when $|v|$ is odd) or its first half contains two 1’s and the second half has no 1. This is a contradiction. □

Example 2.62 Show that $L = \{ \beta \beta^R \gamma | \beta \in \{0,1\}^+, \gamma \in \{0,1\}^* \}$ is not regular.

Proof. By way of contradiction, assume that $L$ is a regular language, accepted by a DFA $M$ of $s$ states. We need to choose a string $a \in L$ with $|a| \geq s$ and divide it into three substrings $a = xyz$ with $|y| \geq s$. Note that if we do it like in the last example, with $a = 0^*11^0$, $x = \epsilon$, $y = 0^*$ and $z = 110^*$, then the proof does not work:

1. The string $0^s110^*10^s$ is of the form $\beta \beta^R \gamma$, with $\beta = 0^s110^*$ and $\gamma = 0^{*1}$;  
2. The string $0^{s+k+1}110^*10^s$, with $k \geq 2$, is of the form $\beta \beta^R \gamma$, with $\beta = 0$.

To fix this problem, we choose $a = 01^*0^s101^0 \in L$ ($\beta = 01^s0^1$ and $\gamma = \epsilon$), and let $x = 0^s$, $y = 0^*$, and $z = 110^*10$. Note that the only prefix of $x0^s$ that is of the form $\beta \beta^R$ is the whole string, and it holds only for $t = s$.

More precisely, we check that, by the pumping lemma, $y$ can be written as $y = uv^iz$ such that $v \neq \epsilon$ and $xuv^iz \subseteq L$. This means that for any $k \geq 0$, $xuv^iz \subseteq L$. In particular, $xuv^iz = 01^s0^s10^s10^s \in L$. Now, $xuv^iz \subseteq L$ means $01^s0^s10^s10^s = 0\beta \beta^R \gamma$ for some $\beta, \gamma \in \{0,1\}^*$. Since there are only two occurrences of the substring $010$ in this string (as the prefix and suffix), $\beta$ must contain the prefix $010$ and $\beta^R$ must contain the suffix $010$ and $\gamma$ must be the empty string. In other words, $01^s0^s10^s10^s = 0\beta \beta^R$. However, this is obvious impossible, as explained in the proof of the last example. We have reached a contradiction. □

It is interesting to note that $\{ \beta \beta^R | \beta \in \{0,1\}^+, \gamma \in \{0,1\}^* \}$ is equal to the language with the following regular expression:

$$0(0 \cup 1)^+0 \cup 1(0 \cup 1)^+1.$$ Thus, it is a regular language.

Example 2.63 Show that the language $L = \{0^n10^m10^n | n, m, p \geq 1, q \equiv nm \text{ (mod } p) \}$ is not regular.

Solution. By way of contradiction, assume that $L$ is a regular language, accepted by a DFA $M$ of $s$ states. Consider the string $a = 010^{s+1}10^{s+1}10^{s+1}$. Apply the pumping lemma to string $a$, with $x = 010^{s+1}10^{s+1}10$, $y = 0^*$ and $z = \epsilon$. Then, the suffix $y = 0^*$ can be written as $uv^iz$ such that $v \neq \epsilon$ and for
any \( k \geq 0, xuv^kw \) is in \( L \). Take \( k = 0 \). We get \( xuv = 010^{s+110^{t+110^t}} \), with 
\( 1 \leq t \leq s \). Since \( t \neq 1(s + 1) \) (mod \( s + 1 \)), we get a contradiction. \( \square \)

Note that, for any fixed integer \( p > 1 \), the language \( \{ 0^n10^m | n, m, p \geq 1, q \equiv nm \) (mod \( p \)) \} is regular (see Exercise 3(f) of Section 2.3).

* Example 2.64 Consider the following multiplication table on \( \{ a, b, c \} \):

\[
\begin{array}{c|ccc}
\times & a & b & c \\
\hline
a & a & a & c \\
b & c & a & b \\
c & b & c & a \\
\end{array}
\]

Recall, from Example 2.28, that for any string \( x \) in \( \{ a, b, c \}^{+} \), value(\( x \)) denotes the value obtained by multiplying symbols in \( x \) from left to right. Show that the set

\[
L = \{ xy | x, y \in \{ a, b, c \}^{*}, |x| = |y|, \text{ value}(x) = \text{ value}(y) \}
\]

is not regular.

Proof. Assume, by way of contradiction, that language \( L \) is a regular set and is accepted by a DFA \( M \) of \( s > 0 \) states. To find a contradiction, we select a string \( a = bc^*bc^* \in L \) and let \( x = b, y = c^* \) and \( z = bc^* \). Apparently, \( a \in L \). Now, we apply the pumping lemma to decompose \( y \) into \( y = uvw \) with \( v \neq \varepsilon \) and \( xuv^*w \subseteq L \). This means that, for any \( k \geq 0, bc^{k+1}bc^* \in L \); in particular, \( bc^{k+1}bc^* \in L \). It follows that \( bc^{k+1}bc^* = \beta \gamma \) for some \( \beta \) and \( \gamma \) in \( \{ a, b, c \}^{*} \) with \( |\beta| = |\gamma| \) and value(\( \beta \)) = value(\( \gamma \)). From \( |\beta| = |\gamma| \), we know that \( \beta = bc^{k+1} \) and \( \gamma = c^kb^c \). From the given multiplication table, we get value(\( \beta \)) = \( b \) and value(\( \gamma \)) \( \notin \{ a, c \} \). This is a contradiction. \( \square \)

* Example 2.65 Show that the set \( L \) of all strings over alphabet

\[
\Gamma = \left\{ \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix} \right\}
\]

that represent correct multiplication is not regular. For example, the relation

\[
\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}
\]

implies that the following string is in \( L \):

\[
\begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}
\]
2.8 Pumping Lemmas

Proof. Assume, by way of contradiction, that \( L \) is regular and is accepted by a DFA \( M \) of \( s > 0 \) states. Consider the following string \( \alpha \):

\[
\begin{pmatrix}
0 \\
0 \\
1
\end{pmatrix}^s \begin{pmatrix}
1 \\
1 \\
1
\end{pmatrix} \begin{pmatrix}
0 \\
1 \\
1
\end{pmatrix}^s.
\]

The string \( \alpha \) represents the multiplication of the following form:

\[
s \underbrace{0 \cdots 0} \times \underbrace{s+1} \times \underbrace{s+1} = \underbrace{1 \cdots 1} \underbrace{0 \cdots 0},
\]

and hence is in \( L \). We let

\[
x = \begin{pmatrix}
0 \\
0 \\
1
\end{pmatrix}^s \begin{pmatrix}
1 \\
1 \\
1
\end{pmatrix}, \quad y = \begin{pmatrix}
0 \\
1 \\
0
\end{pmatrix}^s, \quad z = \varepsilon.
\]

By the pumping lemma, \( y \) can be written as \( y = uvw \) such that \( v \neq \varepsilon \) and \( xuvwz \in L \). This implies the following incorrect multiplication:

\[
s-|v| \underbrace{0 \cdots 0} \times \underbrace{s-|v|+1} \times \underbrace{s+1} \times \underbrace{s-|v|} = \underbrace{1 \cdots 1} \underbrace{0 \cdots 0}.
\]

Thus, we have reached a contradiction. \( \square \)

**Example 2.66** Show that the set \( L_3 \) of the binary expansions of the integers in set \( A = \{2^n \mid n \geq 1\} \) is regular, but the set \( L_3 \) of the ternary expansions (base 3 representations) of the integers in \( A \) is not regular.

Proof. It is clear that the set \( L_3 \) consists of all strings of the form \( 10^p \) for all \( n \geq 1 \); that is, \( L_3 = 10^+ \). Thus, \( L_3 \) is regular.

Next, we assume, for the sake of contradiction, that \( L_3 \) is regular and is accepted by a DFA \( M \) of \( s > 0 \) states. For any string \( t \in \{0, 1, 2\}^* \), we let \( n_t \) be the integer whose ternary expansion is \( t \) (with possible leading zeros). We select an arbitrary \( x \) in \( L_3 \) with \( |x| \geq s \). We apply the first pumping lemma (Lemma 2.57) to the string \( x \) to get \( x = uvw \), with \( v \neq \varepsilon \) and \( uv^sw \in L_3 \).

Then, for any \( k \geq 0 \), the string \( uv^kw \in L_3 \); that is, \( n_{uv^kw} \in A \). Let \( 2^{m_k} \) be the integer whose ternary expansion is equal to \( uv^kw \). Since \( v \neq \varepsilon \) and since \( x > 0 \), we know that \( m_{k+1} > m_k \) for all \( k \geq 0 \). What is \( 2^{m_k} \) in terms of \( n_v, n_w \) and \( n_\varepsilon \)? Assume that \( |v| = p > 0 \) and \( |w| = q \). Then, for \( k \geq 1 \), we have

\[
2^{m_k} = n_u \cdot 3^{k-1} + n_v \cdot 3^{(k-2)p+q} + \cdots + \underbrace{3^q} + n_w.
\]

It follows that, for \( k \geq 2 \),

\[
2^{m_k} - 2^{m_{k-1}} = n_u \cdot 3^{k-1} + (n_v - n_u) \cdot 3^{(k-2)p+q} \\
= 3^{(k-1)p+q} (n_u (3^p - 1) + n_v).
\]
Since $3^{(k-1)p+q}$ is an odd integer, and since $2^{m_1-1}$ divides $2^{m_1} - 2^{m_1-1}$, we must have that $2^{m_1-1}$ divides the integer $n_0(3^p-1)+n_e$. However, this cannot be true for $k \geq 3$, since

$$n_0(3^p-1)+n_e \leq n_0 \cdot 3^p + n_e \cdot 3^p = 2^{m_1} < 2^{m_2}.$$ 

So, we have reached a contradiction. \(\square\)

Sometimes, using the pumping lemma directly to prove a language nonregular takes some thinking to come up with the required string $a$. In these cases, we can often combine the pumping lemma with the closure properties of Section 2.6 to produce simpler proofs. The following are some examples. Let $\#_a(w)$ denote the number of occurrences of symbol $a$ in string $w$.

**Example 2.67** Show that $L = \{ w \in \{0,1\}^* \mid \#_0(w) \neq \#_1(w) \}$ is not regular.

**Proof.** We may prove this by selecting $a = 0^*1^{(k-1)+s}$, with $x = \varepsilon$, $y = 0^t$ and $z = 1^{(k-1)+s}$, and arguing that for any way of dividing $y$ into $y = uvw$ with $v \neq \varepsilon$, $xuv^kwz = 0^{s+(k-1)+1}1^{(k-1)+s} \notin L$ when $k = (s!)/|v| + 1$. (Note: Since $|v| \leq |y| = s$, $k$ must be an integer.)

This proof, though somewhat inspiring, is not easy to find. A simpler proof is as follows:

1. $L_1 = \{0^n1^n \mid n \geq 0\}$ is not regular. (This can be proved by the pumping lemma easily as in Example 2.60.)
2. $L_2 = \{ w \in \{0,1\}^* \mid \#_0(w) = \#_1(w) \}$ is not regular, since $L_2 \cap 0^*1^* = L_1$ is not regular. (If $L_2$ were regular then, by the property that regular languages are closed under intersection, $L_1$ would be regular.)
3. $L$ is not regular since $\overline{L} = L_2$ is not regular. \(\square\)

**Example 2.68** Show that $L = \{ a^nb^mc^k \mid n, m, k \geq 0, n \neq m \text{ or } m \neq k \text{ or } k \neq n \}$ is not regular.

**Proof.** It is easy to use the pumping lemma to prove that

$$\overline{L} \cap a^*b^*c^* = \{ a^nb^mc^k \mid n \geq 0 \}$$

is not regular. Thus, by the closure properties of the regular languages, $L$ is not regular. \(\square\)

**Example 2.69** Let $L$ be a regular language. Show that

$$L' = \{ xz \mid (\exists y) \mid x = |y| = |z| \text{ and } xyz \in L \}$$

is not necessarily regular.

**Proof.** Consider the regular language $L = a^*bc^*$. For an arbitrary string $a^ib^jc^j$ in $L$ with $i + j + 1 = 3n$, let $a^ib^jc^j = xyz$ with $|x| = |y| = |z| = n > 0$. There are three cases:
2.8 Pumping Lemmas

(1) Both $i$ and $j$ are greater than or equal to $n$. Then, $x = a^n$ and $z = c^j$ and so $a^i c^j \in L'$.

(2) The integer $i$ is less than $n$. Then, $x = a^i b c^{n-i-1}$ and $z = c^j$ and so $a^i b c^{2n-i-1} \in L'$. Note that $i + (2n - i - 1) = 2n - 1$ is odd.

(3) The integer $j$ is less than $n$. Similar to case (2), we get $a^{2n-j-1} b c^j \in L'$. Thus, we can see that

$$L' = \{ a^n c^j \mid n > 0 \} \cup \{ a^m b c^j \mid m + n \text{ is odd} \}.$$ 

It follows that $L'$ is not regular, since $L' \cap a^* c^* = \{ a^n c^j \mid n > 0 \}$ is not regular (by Example 2.60). 

\[\square\]

Exercise 2.8

1. Show that the following languages are not regular.
   (a) $\{0^{n^3 + 2n^2 - 3n} \mid n \geq 0\}$.
   (b) $\{0^p 1^q 0^n 1^p \mid p + q = m + n, p, q, m, n \geq 0\}$.
   (c) $\{0^n 1^n \mid m, n \geq 0 \text{ and } m \neq 2n + 1\}$.
   (d) $\{0^n 1^n \mid 2n \leq m \leq 3n, m, n \geq 0\}$.
   (e) $\{w \in \{0, 1, 2\}^* \mid \#_a(w) + \#_1(w) = \#_2(w)\}$.
   (f) $\{0^p q \mid p \text{ and } q \text{ are primes}\}$.

2. For each of the following languages, determine whether it is regular. Present a proof for your answer.
   (a) The set of binary strings having an equal number of 0’s and 1’s.
   (b) The set of binary strings having an equal number of 01’s and 10’s.
   (c) The set of binary strings having an equal number of 010’s and 101’s.
   (d) $\{xy \mid x, y \in \{0, 1\}^*, |x| = |y|, \#_a(x) \geq \#_a(y)\}$.
   (e) $\{xyz \mid x, y, z \in \{0, 1\}^*, |x| = |z| > 0, \#_a(x) \geq \#_a(z)\}$.
   (f) $\{x\#y\#z \mid x, y, z \text{ are binary expansions of positive integers satisfying } x + y = z\}$.

* 3. Let $\Gamma$ be the alphabet of Example 2.65.
   (a) Show that the set $L$ of all strings over alphabet $\Gamma$ that represent correct division is not regular. For example,

   \[
   \begin{array}{cccc}
   \hline
   & 1 & 1 & 1 & 1 \\
   z & 0 & 0 & 1 & 1 \\
   \hline
   & 0 & 0 & 1 & 1 \\
   \end{array}
   \]

   implies that the following string is in $L$:

   \[
   \left(\begin{array}{c}
   1 \\
   0 \\
   1 \\
   0 \\
   \end{array}\right) \left(\begin{array}{c}
   1 \\
   0 \\
   1 \\
   0 \\
   \end{array}\right) \left(\begin{array}{c}
   1 \\
   0 \\
   1 \\
   1 \\
   \end{array}\right).
   \]
(b) Show that the set of all strings over $\Gamma$ that represent correct multiplication, with the second multiplier equal to 3, is regular.

4. Is it true that for any regular language $L$ over $\{0, 1\}$, the set
$$N(L) = \{0^{\#(x)}1^{\#(\bar{x})} \mid x \in L\}$$
is also regular? Prove your answer.

5. Prove the following stronger form of the pumping lemma: For any regular language $L$ and any positive integer $k$, there exists a positive integer $s$ such that any string $x$ in $L$ with $|x| > s$ can be decomposed into $x = uvw$ such that $|v| > k$ and for any $i \geq 0$, $uv^iw \in L$.

*6. Find a regular language $L$ such that
$$\hat{L} = \{xz \mid (\exists y)(|x| = |y| = |z| \text{ and } xyzy \in L)\}$$
is not regular.

7. Let $A$ and $B$ be regular sets over alphabet $\Sigma$. Which of the following languages, if any, are necessarily regular?

(a) $\{x \mid x \in A \text{ and } x^R \in B\}$.
(b) $\{x \mid x \in A \text{ and } x^R \notin B\}$.
(c) $\{x \mid x = x^R \text{ and } x \in A\}$.

* (d) $\{a_1b_1a_2b_2 \cdots a_nb_n \mid a_i, b_i \in \Sigma \text{ for } 1 \leq i \leq n, a_1a_2 \cdots a_n \in A, b_1b_2 \cdots b_n \in B\}$.

* (e) $\{a_1a_2a_3a_4 \cdots a_{2n-1}a_1 \mid a_i \in \Sigma \text{ for } 1 \leq i \leq n, a_1a_2 \cdots a_n \in A\}$.

* (f) $\{a_1a_2a_3a_4 \cdots a_{2n-1}a_2 \mid a_i \in \Sigma \text{ for } 1 \leq i \leq 2n, a_1a_2 \cdots a_n \in A\}$.

8. Consider the language
$$L = \{x0^n1^nz \mid x \in P, y \in Q, z \in R\},$$
where $P$, $Q$, and $R$ are nonempty sets over alphabet $\{0, 1\}$. Can you find regular sets $P, Q, R$ such that $L$ is not regular? Can you find regular sets $P, Q, R$ such that $L$ is regular? What if $P, Q, R$ must be infinite regular sets?

*9. (a) Is the language $\{0^{2n^2+4n} \mid m, n \geq 0\}$ regular? Prove your answer.
(b) Let $L$ be a language over alphabet $\{a\}$. Show that $L^*$ is regular. [*Hint: Prove and use the fact that if $a$ and $b$ are relatively prime natural numbers, then for any integer $n \geq ab$, there exist nonnegative integers $u$ and $v$ such that $n = ua + vb.$]