roduce an affirmative or negative response. What sort of description encompasses
of each of these seemingly different types of mechanical computation?
le newspaper vending machine, similar to those found on many street corners, is
strate the components of a finite-state machine. The input to the machine consists
dimes, and quarters. When 30 cents is inserted, the cover of the machine may
and a paper removed. If the total of the coins exceeds 30 cents, the machine
accepts the overpayment and does not give change.
wspaper machine on the street corner has no memory, at least not as we usually
f memory in a computing machine. However, the machine "knows" that an
5 cents will unlatch the cover when 25 cents has previously been inserted. This
is acquired by the machine's altering its internal state whenever input is received
sed.
hine state represents the status of an ongoing computation. The internal operation
ing machine can be described by the interactions of the following seven states.
of the states, given in italics, indicate the progress made toward opening the

30 cents: The state of the machine before any coins are inserted

25 cents: The state after a nickel has been input

20 cents: The state after two nickels or a dime have been input

15 cents: The state after three nickels or a dime and a nickel have been input

10 cents: The state after four nickels, a dime and two nickels, or two dimes have
nput

5 cents: The state after a quarter, five nickels, two dimes and a nickel, or one
nd three nickels have been input

0 cents: The state that represents having at least 30 cents input

sertion of a coin causes the machine to alter its state. When 30 cents or more
e state *needs 0 cents* is entered and the latch is opened. Such a state is called
since it indicates the correctness of the input.
sign of the machine must represent each of the components symbolically. Rather
ence of coins, the input to the abstract machine is a string of symbols. A labeled
aph known as a **state diagram** is often used to represent the transformations of
l state of the machine. The nodes of the state diagram are the states described
needs m cents node is represented simply by m in the state diagram. The state
nine at the beginning of a computation is designated ⨂. The initial state for the
vending machine is the node *30*.
cs are labeled n, d, or q, representing the input of a nickel, dime, or quarter. An
ode x to node y labeled v indicates that processing input v when the machine
t causes the machine to enter state y. Figure 5.1 gives the state diagram for the
vending machine. The arc labeled d from node *15* to *5* represents the change of
machine when 15 cents has previously been processed and a dime is input. The

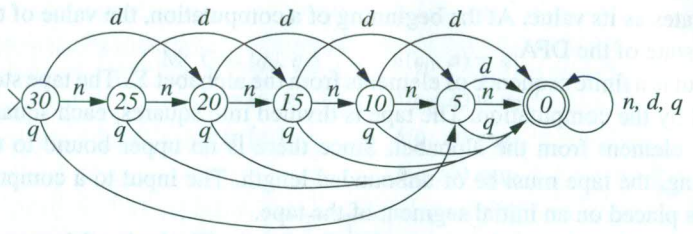


FIGURE 5.1 State diagram of newspaper vending machine.

cycles of length one from node *0* to itself indicate that any input that increases the total past 30 cents leaves the latch unlocked.

Input to the machine consists of strings from $\{n, d, q\}^*$. The sequence of states entered during the processing of an input string can be traced by following the arcs in the state diagram. The machine is in its initial state at the beginning of a computation. The arc labeled by the first input symbol is traversed, specifying the subsequent machine state. The next symbol of the input string is processed by traversing the appropriate arc from the current node, the node reached by traversal of the previous arc. This procedure is repeated until the entire input string has been processed. The string is accepted if the computation terminates in the accepting state. The string *dndn* is accepted by the vending machine, while the string *nndn* is not accepted since the computation terminates in state *5*.

5.2 Deterministic Finite Automata

The analysis of the vending machine required separating the fundamentals of the design from the implementational details. The implementation-independent description is often referred to as an *abstract machine*. We now introduce a class of abstract machines whose computations can be used to determine the acceptability of input strings.

Definition 5.2.1

A **deterministic finite automaton (DFA)** is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ a finite set called the *alphabet*, $q_0 \in Q$ a distinguished state known as the *start state*, F a subset of Q called the *final* or *accepting states*, and δ a total function from $Q \times \Sigma$ to Q known as the *transition function*.

We have referred to a deterministic finite automaton as an abstract machine. To reveal its mechanical nature, the operation of a DFA is described in terms of components that are present in many familiar computing machines. An automaton can be thought of as a machine consisting of five components: a single internal register, a set of values for the register, a tape, a tape reader, and an instruction set.

The states of a DFA represent the internal status of the machine and are often denoted $q_0, q_1, q_2, \ldots, q_n$. The register of the machine, also called the finite control, contains

CHAPTER 5

Finite Automata

In this chapter we introduce the family of abstract computing devices known as finite-state machines. The computations of a finite-state machine determine whether a string satisfies a set of conditions or matches a prescribed pattern. Finite-state machines share properties common to many mechanical devices; they process input and generate output. A vending machine takes coins as input and returns food or beverages as output. A combination lock expects a sequence of numbers and opens the lock if the input sequence is correct. The input to a finite-state machine is a string and the result of a computation indicates acceptability of the string. The set of strings that are accepted makes up the language of the machine.

The preceding examples of machines exhibit a property that we take for granted in mechanical computation, determinism. When the appropriate amount of money is inserted into a vending machine, we are upset if nothing is forthcoming. Similarly, we expect the combination to open the lock and all other sequences to fail. Initially, we require finite-state machines to be deterministic. This condition will be relaxed to examine the effects of nondeterminism on the capabilities of finite-state computation.

5.1 A Finite-State Machine

A formal definition of a machine is not concerned with the hardware involved in the operation of the machine, but rather with a description of the internal operations as the machine processes the input. A vending machine may be built with levers, a combination lock with tumblers, and an electronic entry system is controlled by a microchip, but all accept

produce an affirmative or negative response. What sort of description encompasses
res of each of these seemingly different types of mechanical computation?
mple newspaper vending machine, similar to those found on many street corners, is
lustrate the components of a finite-state machine. The input to the machine consists
s, dimes, and quarters. When 30 cents is inserted, the cover of the machine may
ed and a paper removed. If the total of the coins exceeds 30 cents, the machine
ly accepts the overpayment and does not give change.
newspaper machine on the street corner has no memory, at least not as we usually
of memory in a computing machine. However, the machine "knows" that an
al 5 cents will unlatch the cover when 25 cents has previously been inserted. This
lge is acquired by the machine's altering its internal state whenever input is received
cessed.
achine state represents the status of an ongoing computation. The internal operation
ending machine can be described by the interactions of the following seven states.
nes of the states, given in italics, indicate the progress made toward opening the

eds 30 cents: The state of the machine before any coins are inserted

eds 25 cents: The state after a nickel has been input

eds 20 cents: The state after two nickels or a dime have been input

eds 15 cents: The state after three nickels or a dime and a nickel have been input

eds 10 cents: The state after four nickels, a dime and two nickels, or two dimes have
en input

eds 5 cents: The state after a quarter, five nickels, two dimes and a nickel, or one
ne and three nickels have been input

eds 0 cents: The state that represents having at least 30 cents input

e insertion of a coin causes the machine to alter its state. When 30 cents or more
t, the state *needs 0 cents* is entered and the latch is opened. Such a state is called
ing since it indicates the correctness of the input.
e design of the machine must represent each of the components symbolically. Rather
sequence of coins, the input to the abstract machine is a string of symbols. A labeled
d graph known as a **state diagram** is often used to represent the transformations of
ernal state of the machine. The nodes of the state diagram are the states described
The *needs m cents* node is represented simply by m in the state diagram. The state
machine at the beginning of a computation is designated ⟩◯. The initial state for the
aper vending machine is the node *30*.
he arcs are labeled n, d, or q, representing the input of a nickel, dime, or quarter. An
m node x to node y labeled v indicates that processing input v when the machine
tate x causes the machine to enter state y. Figure 5.1 gives the state diagram for the
aper vending machine. The arc labeled d from node *15* to *5* represents the change of
f the machine when 15 cents has previously been processed and a dime is input. The

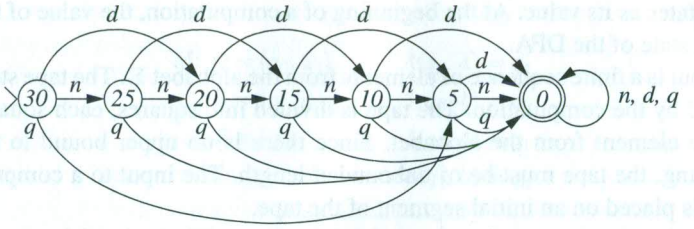


FIGURE 5.1 State diagram of newspaper vending machine.

cycles of length one from node *0* to itself indicate that any input that increases the total past 30 cents leaves the latch unlocked.

Input to the machine consists of strings from $\{n, d, q\}^*$. The sequence of states entered during the processing of an input string can be traced by following the arcs in the state diagram. The machine is in its initial state at the beginning of a computation. The arc labeled by the first input symbol is traversed, specifying the subsequent machine state. The next symbol of the input string is processed by traversing the appropriate arc from the current node, the node reached by traversal of the previous arc. This procedure is repeated until the entire input string has been processed. The string is accepted if the computation terminates in the accepting state. The string *dndn* is accepted by the vending machine, while the string *nndn* is not accepted since the computation terminates in state *5*.

5.2 Deterministic Finite Automata

The analysis of the vending machine required separating the fundamentals of the design from the implementational details. The implementation-independent description is often referred to as an *abstract machine.* We now introduce a class of abstract machines whose computations can be used to determine the acceptability of input strings.

Definition 5.2.1

A **deterministic finite automaton (DFA)** is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ a finite set called the *alphabet*, $q_0 \in Q$ a distinguished state known as the *start state*, F a subset of Q called the *final* or *accepting states*, and δ a total function from $Q \times \Sigma$ to Q known as the *transition function*.

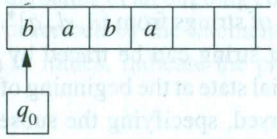
We have referred to a deterministic finite automaton as an abstract machine. To reveal its mechanical nature, the operation of a DFA is described in terms of components that are present in many familiar computing machines. An automaton can be thought of as a machine consisting of five components: a single internal register, a set of values for the register, a tape, a tape reader, and an instruction set.

The states of a DFA represent the internal status of the machine and are often denoted $q_0, q_1, q_2, \ldots, q_n$. The register of the machine, also called the finite control, contains

e states as its value. At the beginning of a computation, the value of the register is art state of the DFA.

input is a finite sequence of elements from the alphabet Σ. The tape stores the input ded by the computation. The tape is divided into squares, each square capable of one element from the alphabet. Since there is no upper bound to the length of string, the tape must be of unbounded length. The input to a computation of the on is placed on an initial segment of the tape.

tape head reads a single square of the input tape. The body of the machine consists pe head and the register. The position of the tape head is indicated by placing the the machine under the tape square being scanned. The current state of the automaton ted by the value on the register. The initial configuration of a computation with input depicted



omputation of an automaton consists of the execution of a sequence of instructions. cution of an instruction alters the state of the machine and moves the tape head one to the right. The instruction set is obtained from the transition function of the DFA. achine state and the symbol scanned determine the instruction to be executed. The of a machine in state q_i scanning an a is to reset the state to $\delta(q_i, a)$. Since δ is a nction, there is exactly one instruction specified for every combination of state and ymbol, hence the *deterministic* in deterministic finite automaton.

e objective of a computation of an automaton is to determine the acceptability of ut string. A computation begins with the tape head scanning the leftmost square of e and the register containing the state q_0. The state and symbol are used to select the tion. The machine then alters its state as prescribed by the instruction, and the tape oves to the right. The transformation of a machine by the execution of an instruction s exhibited in Figure 5.2. The instruction cycle is repeated until the tape head scans a square, at which time the computation terminates. An input string is **accepted** if the tation terminates in an accepting state; otherwise it is rejected. The computation in 5.2 exhibits the acceptance of the string aba.

tion 5.2.2

$= (Q, \Sigma, \delta, q_0, F)$ be a DFA. The **language** of M, denoted L(M), is the set of s in Σ^* accepted by M.

DFA can be considered to be a language acceptor; the language of the machine is y the set of strings accepted by its computations. The language of the machine in e 5.2 is the set of all strings over $\{a, b\}$ that end in a.

DFA is a read-only machine that processes the input in a left-to-right manner; once ut symbol has been read, it has no further effect on the computation. At any point g the computation, the result depends only on the current state and the unprocessed

M: $Q = \{q_0, q_1\}$
$\Sigma = \{a, b\}$
$F = \{q_1\}$

$\delta(q_0, a) = q_1$
$\delta(q_0, b) = q_0$
$\delta(q_1, a) = q_1$
$\delta(q_1, b) = q_0$

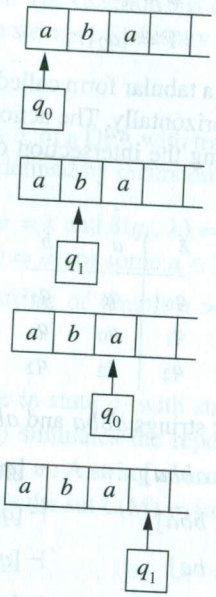


FIGURE 5.2 Computation in a DFA.

input. This combination is called a **machine configuration** and is represented by the ordered pair $[q_i, w]$, where q_i is the current state and $w \in \Sigma^*$ is the unprocessed input. The instruction cycle of a DFA transforms one machine configuration to another. The notation $[q_i, aw] \vdash_M [q_j, w]$ indicates that configuration $[q_j, w]$ is obtained from $[q_i, aw]$ by the execution of one instruction cycle of the machine M. The symbol $\vdash_M$, read "yields," defines a function from $Q \times \Sigma^+$ to $Q \times \Sigma^*$ that can be used to trace computations of the DFA. The M is omitted when there is no possible ambiguity.

Definition 5.2.3

The function $\vdash_M$ on $Q \times \Sigma^+$ is defined by

$$[q_i, aw] \vdash_M [\delta(q_i, a), w]$$

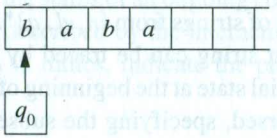
for $a \in \Sigma$ and $w \in \Sigma^*$, where δ is the transition function of the DFA M.

The notation $[q_i, u] \vdash^* [q_j, v]$ is used to indicate that configuration $[q_j, v]$ can be obtained from $[q_i, u]$ by zero or more transitions.

one of the states as its value. At the beginning of a computation, the value of the register is q_0, the start state of the DFA.

The input is a finite sequence of elements from the alphabet Σ. The tape stores the input until needed by the computation. The tape is divided into squares, each square capable of holding one element from the alphabet. Since there is no upper bound to the length of an input string, the tape must be of unbounded length. The input to a computation of the automaton is placed on an initial segment of the tape.

The tape head reads a single square of the input tape. The body of the machine consists of the tape head and the register. The position of the tape head is indicated by placing the body of the machine under the tape square being scanned. The current state of the automaton is indicated by the value on the register. The initial configuration of a computation with input *baba* is depicted



A computation of an automaton consists of the execution of a sequence of instructions. The execution of an instruction alters the state of the machine and moves the tape head one square to the right. The instruction set is obtained from the transition function of the DFA. The machine state and the symbol scanned determine the instruction to be executed. The action of a machine in state q_i scanning an a is to reset the state to $\delta(q_i, a)$. Since δ is a total function, there is exactly one instruction specified for every combination of state and input symbol, hence the *deterministic* in deterministic finite automaton.

The objective of a computation of an automaton is to determine the acceptability of the input string. A computation begins with the tape head scanning the leftmost square of the tape and the register containing the state q_0. The state and symbol are used to select the instruction. The machine then alters its state as prescribed by the instruction, and the tape head moves to the right. The transformation of a machine by the execution of an instruction cycle is exhibited in Figure 5.2. The instruction cycle is repeated until the tape head scans a blank square, at which time the computation terminates. An input string is **accepted** if the computation terminates in an accepting state; otherwise it is rejected. The computation in Figure 5.2 exhibits the acceptance of the string *aba*.

Definition 5.2.2

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. The **language** of M, denoted L(M), is the set of strings in Σ^* accepted by M.

A DFA can be considered to be a language acceptor; the language of the machine is simply the set of strings accepted by its computations. The language of the machine in Figure 5.2 is the set of all strings over $\{a, b\}$ that end in a.

A DFA is a read-only machine that processes the input in a left-to-right manner; once an input symbol has been read, it has no further effect on the computation. At any point during the computation, the result depends only on the current state and the unprocessed

M: $Q = \{q_0, q_1\}$ $\quad \delta(q_0, a) = q_1$
$\Sigma = \{a, b\}$ $\quad \delta(q_0, b) = q_0$
$F = \{q_1\}$ $\quad \delta(q_1, a) = q_1$
$\quad \delta(q_1, b) = q_0$

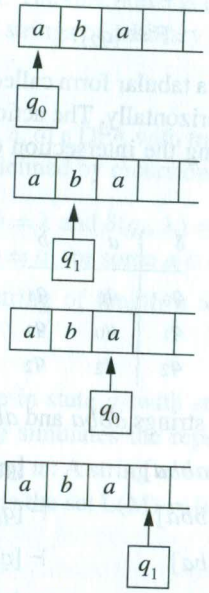


FIGURE 5.2 Computation in a DFA.

input. This combination is called a **machine configuration** and is represented by the ordered pair $[q_i, w]$, where q_i is the current state and $w \in \Sigma^*$ is the unprocessed input. The instruction cycle of a DFA transforms one machine configuration to another. The notation $[q_i, aw] \vdash_M [q_j, w]$ indicates that configuration $[q_j, w]$ is obtained from $[q_i, aw]$ by the execution of one instruction cycle of the machine M. The symbol $\vdash_M$, read "yields," defines a function from $Q \times \Sigma^+$ to $Q \times \Sigma^*$ that can be used to trace computations of the DFA. The M is omitted when there is no possible ambiguity.

Definition 5.2.3

The function $\vdash_M$ on $Q \times \Sigma^+$ is defined by

$$[q_i, aw] \vdash_M [\delta(q_i, a), w]$$

for $a \in \Sigma$ and $w \in \Sigma^*$, where δ is the transition function of the DFA M.

The notation $[q_i, u] \vdash^* [q_j, v]$ is used to indicate that configuration $[q_j, v]$ can be obtained from $[q_i, u]$ by zero or more transitions.

le 5.2.1

FA M defined below accepts the set of strings over $\{a, b\}$ that contain the substrin
at is, $L(M) = (a \cup b)^*bb(a \cup b)^*$. The states and alphabet of M are

$$M: Q = \{q_0, q_1, q_2\}$$
$$\Sigma = \{a, b\}$$
$$F = \{q_2\}.$$

ansition function δ is given in a tabular form called the *transition table*. The states are
vertically and the alphabet horizontally. The action of the automaton in state q_i with
a can be determined by finding the intersection of the row corresponding to q_i and
olumn corresponding to a.

δ	a	b
q_0	q_0	q_1
q_1	q_0	q_2
q_2	q_2	q_2

computations of M with input strings *abba* and *abab* are traced using the function $\vdash$.

$[q_0, abba]$	$[q_0, abab]$
$\vdash [q_0, bba]$	$\vdash [q_0, bab]$
$\vdash [q_1, ba]$	$\vdash [q_1, ab]$
$\vdash [q_2, a]$	$\vdash [q_0, b]$
$\vdash [q_2, \lambda]$	$\vdash [q_1, \lambda]$
accepts	rejects

e string *abba* is accepted since the computation halts in state q_2. □

ample 5.2.2

ıe newspaper vending machine from the previous section can be represented by a DFA
ith the following states, alphabet, and transition function. The start state is the state *30*.

$Q = \{0, 5, 10, 15, 20, 25, 30\}$
$\Sigma = \{n, d, q\}$
$F = \{30\}$

δ	n	d	q
0	0	0	0
5	0	0	0
10	5	0	0
15	10	5	0
20	15	10	0
25	20	15	0
30	25	20	5

e language of the vending machine consists of all strings that represent a sum of 30 cents
more. Can you construct a regular expression that defines the language of this machine? □

The transition function specifies the action of the machine for a given state and element
m the alphabet. This function can be extended to a function $\hat{\delta}$ whose input consists of a
te and a string over the alphabet. The function $\hat{\delta}$ is constructed by recursively extending
e domain from elements of Σ to strings of arbitrary length.

efinition 5.2.4 (Read)

e **extended transition function**, $\hat{\delta}$, of a DFA with transition function δ is a function from
$\times \Sigma^*$ to Q. The values of $\hat{\delta}$ are defined by recursion on the length of the input string.

i) Basis: $length(w) = 0$. Then $w = \lambda$ and $\hat{\delta}(q_i, \lambda) = q_i$.
$length(w) = 1$. Then $w = a$, for some $a \in \Sigma$, and $\hat{\delta}(q_i, a) = \delta(q_i, a)$.

ii) Recursive step: Let w be a string of length $n > 1$. Then $w = ua$ and $\hat{\delta}(q_i, ua) = \delta(\hat{\delta}(q_i, u), a)$.

The computation of a machine in state q_i with string w halts in state $\hat{\delta}(q_i, w)$. The
valuation of the function $\hat{\delta}(q_0, w)$ simulates the repeated applications of the transition
unction required to process the string w. A string w is accepted if $\hat{\delta}(q_0, w) \in F$. Using this
otation, the language of a DFA M is the set $L(M) = \{w \mid \hat{\delta}(q_0, w) \in F\}$.

5.3 State Diagrams and Examples

The state diagram of a DFA is a labeled directed graph in which the nodes represent the states of the machine and the arcs are obtained from the transition function. The graph in Figure 5.1 is the state diagram for the newspaper vending machine DFA. Because of the intuitive nature of the graphic representation, we will often present the state diagram rather than the sets and transition function that constitute the formal definition of a DFA.

Definition 5.3.1

The **state diagram** of a DFA $M = (Q, \Sigma, \delta, q_0, F)$ is a labeled directed graph G defined by the following conditions:

i) The nodes of G are the elements of Q.
ii) The labels on the arcs of G are elements of Σ.
iii) q_0 is the start node, which is depicted ⟩◯.
iv) F is the set of accepting nodes; each accepting node is depicted ◎.
v) There is an arc from node q_i to q_j labeled a, if $\delta(q_i, a) = q_j$.
vi) For every node q_i and symbol $a \in \Sigma$, there is exactly one arc labeled a leaving q_i.

Example 5.2.1

The DFA M defined below accepts the set of strings over $\{a, b\}$ that contain the substring bb. That is, $L(M) = (a \cup b)^*bb(a \cup b)^*$. The states and alphabet of M are

$$M: Q = \{q_0, q_1, q_2\}$$
$$\Sigma = \{a, b\}$$
$$F = \{q_2\}.$$

The transition function δ is given in a tabular form called the *transition table*. The states are listed vertically and the alphabet horizontally. The action of the automaton in state q_i with input a can be determined by finding the intersection of the row corresponding to q_i and the column corresponding to a.

δ	a	b
q_0	q_0	q_1
q_1	q_0	q_2
q_2	q_2	q_2

The computations of M with input strings $abba$ and $abab$ are traced using the function $\vdash$.

$[q_0, abba]$	$[q_0, abab]$
$\vdash [q_0, bba]$	$\vdash [q_0, bab]$
$\vdash [q_1, ba]$	$\vdash [q_1, ab]$
$\vdash [q_2, a]$	$\vdash [q_0, b]$
$\vdash [q_2, \lambda]$	$\vdash [q_1, \lambda]$
accepts	rejects

The string $abba$ is accepted since the computation halts in state q_2. □

Example 5.2.2

The newspaper vending machine from the previous section can be represented by a DFA with the following states, alphabet, and transition function. The start state is the state *30*.

$Q = \{0, 5, 10, 15, 20, 25, 30\}$

$\Sigma = \{n, d, q\}$

$F = \{0\}$

δ	n	d	q
0	*0*	*0*	*0*
5	*0*	*0*	*0*
10	*5*	*0*	*0*
15	*10*	*5*	*0*
20	*15*	*10*	*0*
25	*20*	*15*	*0*
30	*25*	*20*	*5*

The language of the vending machine consists of all strings that represent a sum of 30 cents or more. Can you construct a regular expression that defines the language of this machine? □

The transition function specifies the action of the machine for a given state and element from the alphabet. This function can be extended to a function $\hat{\delta}$ whose input consists of a state and a string over the alphabet. The function $\hat{\delta}$ is constructed by recursively extending the domain from elements of Σ to strings of arbitrary length.

Definition 5.2.4

The **extended transition function**, $\hat{\delta}$, of a DFA with transition function δ is a function from $Q \times \Sigma^*$ to Q. The values of $\hat{\delta}$ are defined by recursion on the length of the input string.

i) Basis: $length(w) = 0$. Then $w = \lambda$ and $\hat{\delta}(q_i, \lambda) = q_i$.
$length(w) = 1$. Then $w = a$, for some $a \in \Sigma$, and $\hat{\delta}(q_i, a) = \delta(q_i, a)$.

ii) Recursive step: Let w be a string of length $n > 1$. Then $w = ua$ and $\hat{\delta}(q_i, ua) = \delta(\hat{\delta}(q_i, u), a)$.

The computation of a machine in state q_i with string w halts in state $\hat{\delta}(q_i, w)$. The evaluation of the function $\hat{\delta}(q_0, w)$ simulates the repeated applications of the transition function required to process the string w. A string w is accepted if $\hat{\delta}(q_0, w) \in F$. Using this notation, the language of a DFA M is the set $L(M) = \{w \mid \hat{\delta}(q_0, w) \in F\}$.

5.3 State Diagrams and Examples

The state diagram of a DFA is a labeled directed graph in which the nodes represent the states of the machine and the arcs are obtained from the transition function. The graph in Figure 5.1 is the state diagram for the newspaper vending machine DFA. Because of the intuitive nature of the graphic representation, we will often present the state diagram rather than the sets and transition function that constitute the formal definition of a DFA.

Definition 5.3.1

The **state diagram** of a DFA $M = (Q, \Sigma, \delta, q_0, F)$ is a labeled directed graph G defined by the following conditions:

i) The nodes of G are the elements of Q.
ii) The labels on the arcs of G are elements of Σ.
iii) q_0 is the start node, which is depicted ⟩○.
iv) F is the set of accepting nodes; each accepting node is depicted ◎.
v) There is an arc from node q_i to q_j labeled a, if $\delta(q_i, a) = q_j$.
vi) For every node q_i and symbol $a \in \Sigma$, there is exactly one arc labeled a leaving q_i.

A transition of a DFA is represented by an arc in the state diagram. Tracing the computation of a DFA in the corresponding state diagram constructs a path that begins at node q_0 and "spells" the input string. Let $\mathbf{p}_w$ be a path beginning at q_0 that spells w, and let q_w be the terminal node of $\mathbf{p}_w$. Theorem 5.3.2 proves that there is only one such path for every string $w \in \Sigma^*$. Moreover, q_w is the state of the DFA upon completion of the processing of w.

Theorem 5.3.2

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA and let $w \in \Sigma^*$. Then w determines a unique path $\mathbf{p}_w$ in the state diagram of M and $\hat{\delta}(q_0, w) = q_w$.

Proof. The proof is by induction on the length of the string. If the length of w is zero, then $\hat{\delta}(q_0, \lambda) = q_0$. The corresponding path is the null path that begins and terminates with q_0.

Assume that the result holds for all strings of length n or less. Let $w = ua$ be a string of length $n + 1$. By the inductive hypothesis, there is a unique path $\mathbf{p}_u$ that spells u and $\hat{\delta}(q_0, u) = q_u$. The path $\mathbf{p}_w$ is constructed by following the arc labeled a from q_u. This is the only path from q_0 that spells w since $\mathbf{p}_u$ is the unique path that spells u and there is only one arc leaving q_u labeled a. The terminal state of the path $\mathbf{p}_w$ is determined by the transition $\delta(q_u, a)$. From the definition of the extended transition function, $\hat{\delta}(q_0, w) = \delta(\hat{\delta}(q_0, u), a)$. Since $\hat{\delta}(q_0, u) = q_u$, $q_w = \delta(q_u, a) = \delta(\hat{\delta}(q_0, u), a) = \hat{\delta}(q_0, w)$ as desired. ■

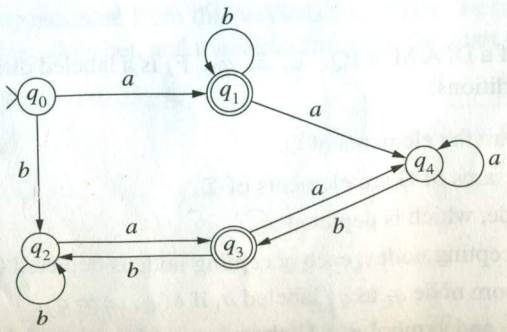
The equivalence of computations of a DFA and paths in the state diagram gives us a heuristic method for determining the language of the DFA. The strings accepted in a state q_i are precisely those spelled by paths from q_0 to q_i. We can separate the determination of these paths into two parts:

i) First, find regular expressions $u_1, \ldots, u_n$ for strings on all paths from q_0 that reach q_i the first time.

ii) Find regular expressions $v_1, \ldots, v_m$ for all ways to leave q_i and return to q_i.

The strings accepted by q_i are $(u_1 \cup \cdots \cup u_n)(v_1 \cup \cdots \cup v_m)^*$.

Consider the DFA



The language of M consists of all strings spelled by paths from q_0 to either q_1 or q_3. Using the heuristic described previously, the strings on the paths to each of the accepting states are

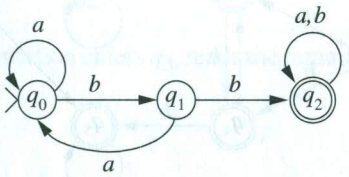
State	Paths to q_i	Simple Cycles from q_i to q_i	Accepted Strings
q_1	a	b	ab^*
q_3	ab^*aa^*b, bb^*a	bb^*a, aa^*b	$(ab^*aa \cup ba)(ab \cup ba)^*$

Consequently, $L(M) = ab^* \cup (ab^*aa^*b \cup bb^*a)(aa^*b \cup bb^*a)^*$. After we have established additional properties of finite-state computation, we will present an algorithm that automatically produces a regular expression for the language of a finite automaton.

In the remainder of this section we examine a number of DFAs to help develop the ability to design automata to check for patterns in strings. The types of conditions that we will consider include the number of occurrences and the relative positions of specified substrings. In addition, we establish the relationship between a DFA that accepts a language L and one that accepts the complement of L.

Example 5.3.1

The state diagram of the DFA in Example 5.2.1 is



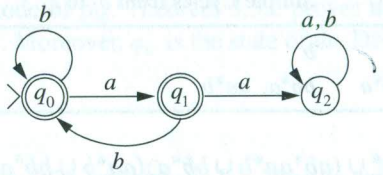
The states are used to record the number of consecutive b's processed. The state q_2 is entered when a substring bb is encountered. Once the machine enters q_2, the remainder of the input is processed, leaving the state unchanged. The computation of the DFA with input $ababb$ and the corresponding path in the state diagram are

Computation	Path
$[q_0, ababb]$	q_0,
$\vdash [q_0, babb]$	q_0,
$\vdash [q_1, abb]$	q_1,
$\vdash [q_0, bb]$	q_0,
$\vdash [q_1, b]$	q_1,
$\vdash [q_2, \lambda]$	q_2

The string $ababb$ is accepted since the halting state of the computation, which is also the terminal state of the path that spells $ababb$, is the accepting state q_2. □

Example 5.3.2

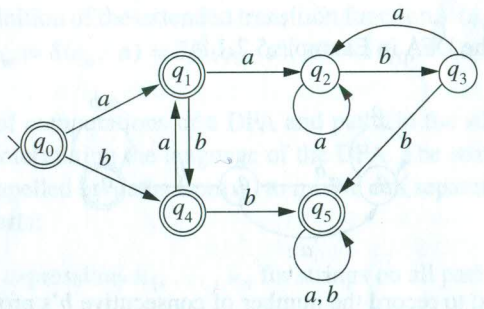
The DFA



accepts $(b \cup ab)^*(a \cup \lambda)$, the set of strings over $\{a, b\}$ that do not contain the substring aa. □

Example 5.3.3

Strings over $\{a, b\}$ that contain the substring bb or do not contain the substring aa are accepted by the DFA depicted below. This language is the union of the languages of the previous examples.



□

The state diagrams for machines that accept the strings with substring bb or without substring aa seem simple compared with the machine that accepts the union of those two languages. There does not appear to be an intuitive way to combine the state diagrams of the constituent DFAs to create the desired composite machine.

The next several examples provide a heuristic for designing DFAs. The first step is to produce an interpretation for the states of the DFA. The interpretation of a state describes properties of the string that has been processed when the machine is in the state. The pertinent properties are determined by the conditions required for a string to be accepted.

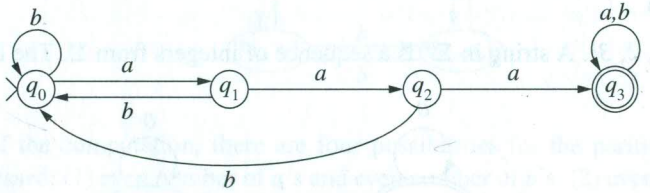
Example 5.3.4

A successful computation of a DFA that accepts the strings over $\{a, b\}$ containing the substring aaa must process three a's in a row. Four states are required to record the status of a computation checking for aaa. The interpretation of the states, along with state names, are

State	Interpretation
q_0:	No progress toward aaa
q_1:	Last symbol processed was an a
q_2:	Last two symbols processed were aa
q_3:	aaa has been found in the string

Prior to reading the first symbol, no progress has been made toward finding aaa. Consequently, this condition represents the start state.

Once the states are identified, it is frequently easy to determine the proper transitions. When computation in state q_1 processes an a, the last two symbols read are aa and q_2 is entered. On the other hand, if a b is read in q_1, the resulting string represents no progress toward aaa and the computation enters q_0. Following a similar strategy, the transitions can be determined for all states producing the DFA



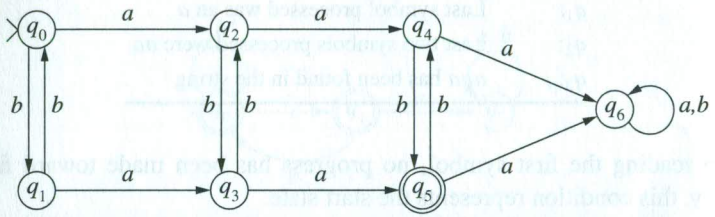
On processing aaa, the computation enters q_3, reads the remainder of the string, and accepts the input. □

Example 5.3.5

Building a machine that accepts strings with exactly two a's and an odd number of b's requires checking two conditions: the number of a's and the parity of the b's. Seven states are required to store the information needed about the string. The interpretation of the states describes the number of a's read and the parity of the string processed when the computation is in the state.

State	Interpretation
q_0:	No a's, even number of b's
q_1:	No a's, odd number of b's
q_2:	One a, even number of b's
q_3:	One a, odd number of b's
q_4:	Two a's, even number of b's
q_5:	Two a's, odd number of b's
q_6:	More than two a's

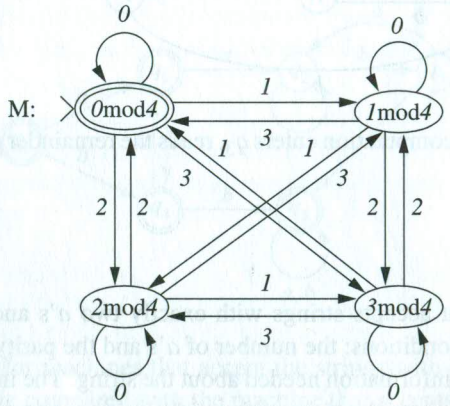
At the beginning of a computation, no a's and no b's have been processed and this becomes the condition of the start state. A DFA accepting this language is



The horizontal arcs count the number of a's in the input string and the vertical pairs of arcs record the parity of the b's. The accepting state is q_5, since it represents the condition required of a string in the language. □

Example 5.3.6

Let $\Sigma = \{0, 1, 2, 3\}$. A string in Σ^* is a sequence of integers from Σ. The DFA



determines whether the sum of integers in an input string is divisible by four. For example, the strings *1 2 3 0 2* and *0 1 3 0* are accepted and *0 1 1 1* rejected by M. The states represent the value of the sum of the processed input modulo 4. □

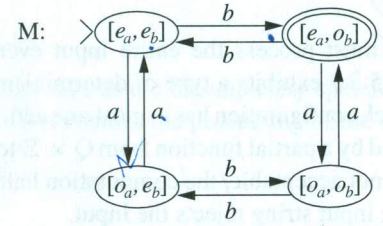
Our definition of DFA allowed only two possible outputs, accept or reject. The definition of output can be extended to have a value associated with each state. The result of a computation is the value associated with the state in which the computation terminates. A machine of this type is called a *Moore machine* after E. F. Moore, who introduced this type of finite-state computation. Associating the value i with the state $imod4$, the machine in Example 5.3.6 acts as a modulo 4 adder.

The state diagrams for machines in Examples 5.3.1, 5.3.2, and 5.3.3 showed that there is no simple method to obtain a DFA that accepts the union of two languages from DFAs

that accept each of the languages. The next two examples show that this is not the case for machines that accept complementary sets of strings. The state diagram for a DFA can easily be transformed into the state diagram for another machine that accepts all, and only, the strings rejected by the original DFA.

Example 5.3.7

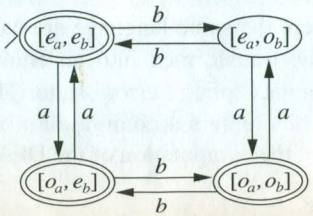
The DFA M accepts the language consisting of all strings over $\{a, b\}$ that contain an even number of a's and an odd number of b's.



At any step of the computation, there are four possibilities for the parities of the input symbols processed: (1) even number of a's and even number of b's, (2) even number of a's and odd number of b's, (3) odd number of a's and even number of b's, (4) odd number of a's and odd number of b's. These four states are represented by ordered pairs in which the first component indicates the parity of the a's and the second component, the parity of the b's that have been processed. Processing a symbol changes one of the parities, designating the appropriate transition. □

Example 5.3.8

Let M be the DFA constructed in Example 5.3.7. A DFA M′ is constructed that accepts all strings over $\{a, b\}$ that do not contain an even number of a's and an odd number of b's. In other words, $L(M') = \{a, b\}^* - L(M)$. Any string rejected by M is accepted by M′ and vice versa. A state diagram for the machine M′ can be obtained from that of M by interchanging the accepting and nonaccepting states.



□

The preceding example shows the relationship between DFAs that accept complementary sets of strings. This relationship is formalized by the following result.

Theorem 5.3.3

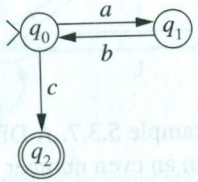
Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. Then $M' = (Q, \Sigma, \delta, q_0, Q - F)$ is a DFA with $L(M') = \Sigma^* - L(M)$.

Proof. Let $w \in \Sigma^*$ and $\hat{\delta}$ be the extended transition function constructed from δ. For each $w \in L(M)$, $\hat{\delta}(q_0, w) \in F$. Hence, $w \notin L(M')$. Conversely, if $w \notin L(M)$, then $\hat{\delta}(q_0, w) \in Q - F$ and $w \in L(M')$. ■

By definition, a DFA must process the entire input even if the result has already been established. Example 5.3.9 exhibits a type of determinism, sometimes referred to as *incomplete determinism*; each configuration has at most one action specified. The transitions of such a machine are defined by a partial function from $Q \times \Sigma$ to Q. As soon as it is possible to determine that a string is not acceptable, the computation halts. A computation that halts before processing the entire input string rejects the input.

Example 5.3.9

The state diagram below defines an incompletely specified DFA that accepts $(ab)^*c$. A computation terminates unsuccessfully as soon as the input varies from the desired pattern.

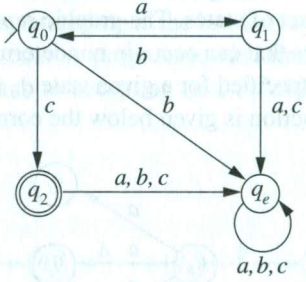


The computation with input *abcc* is rejected since the machine is unable to process the final *c* from state q_2. □

Two machines that accept the same language are called *equivalent*. An incompletely specified DFA can easily be transformed into an equivalent DFA. The transformation requires the addition of a nonaccepting "error" state. This state is entered whenever the incompletely specified machine enters a configuration for which no action is indicated. Upon entering the error state, the computation of the DFA reads the remainder of the string and halts.

Example 5.3.10

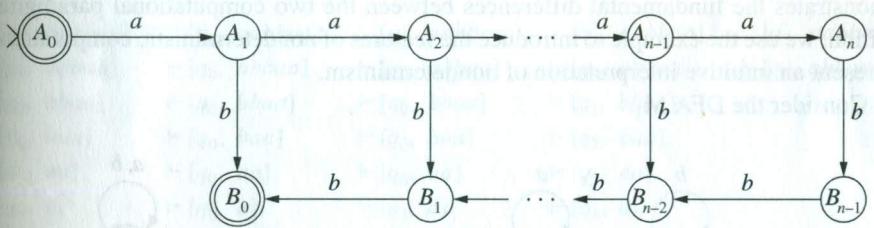
The DFA



accepts the same language as the incompletely specified DFA in Example 5.3.9. The state q_e is the error state that ensures the processing of the entire string. □

Example 5.3.11

The incompletely specified DFA defined by the state diagram

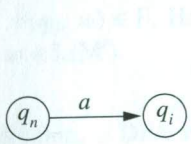


accepts the language $\{a^i b^i \mid i \leq n\}$, for a fixed integer n. The states labeled A_k count the number of a's, and then the B_k's ensure an equal number of b's. This technique cannot be extended to accept $\{a^i b^i \mid i \geq 0\}$ since an infinite number of states would be needed. In the next chapter we will show that the language $\{a^i b^i \mid i \geq 0\}$ is not accepted by any finite automaton. □

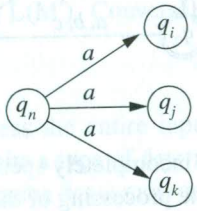
5.4 Nondeterministic Finite Automata

We now alter our definition of machine to allow nondeterministic computations. In a nondeterministic automaton there may be several instructions that can be executed from a given machine configuration. Although this property may seem unnatural for computing machines, the flexibility of nondeterminism often facilitates the design of language acceptors.

A transition in a *nondeterministic finite automaton* (*NFA*) has the same effect as one in a DFA: to change the state of the machine based upon the current state and the symbol being scanned. The transition function must specify all possible states that the machine may enter from a given machine configuration. This is accomplished by having the value of the transition function be a set of states. The graphic representation of state diagrams is used to illustrate the alternatives that can occur in nondeterministic computation. Any finite number of transitions may be specified for a given state q_n and symbol a. The value of the nondeterministic transition function is given below the corresponding diagram.



$\delta(q_n, a) = \{q_i\}$

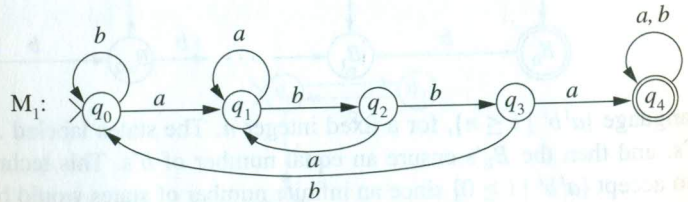


$\delta(q_n, a) = \{q_i, q_j, q_k\}$

$\delta(q_n, a) = \emptyset$

Because nondeterministic computation differs significantly from its deterministic counterpart, we begin the presentation of nondeterministic machines with an example that demonstrates the fundamental differences between the two computational paradigms. In addition, we use the example to introduce the features of nondeterministic computation and to present an intuitive interpretation of nondeterminism.

Consider the DFA M_1



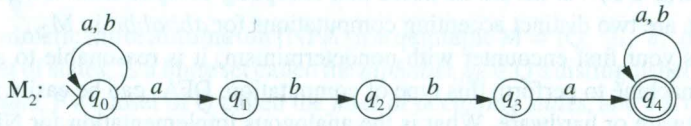
that accepts $(a \cup b)^*abba(a \cup b)^*$, the strings over $\{a, b\}$ that contain the substring *abba*. The states q_0, q_1, q_2, q_3 record the progress toward obtaining the substring *abba*. The states of the machine are

State	Interpretation
q_0:	When there is no progress toward *abba*
q_1:	When the last symbol processed was an *a*
q_2:	When the last two symbols processed were *ab*
q_3:	When the last three symbols processed were *abb*

Upon processing *abba*, state q_4 is entered, the remainder of the string is read, and the input is accepted.

The deterministic computation must "back up" in the sequence q_0, q_1, q_2, q_3 when the current substring is discovered not to have the desired form. If a *b* is scanned when the machine is in state q_3, then q_0 is entered since the last four symbols processed are *abbb* and the current configuration represents no progress toward finding *abba*.

A nondeterministic approach to accepting $(a \cup b)^*abba(a \cup b)^*$ is illustrated by the machine



There are two possible transitions when M_2 processes an *a* in state q_0. One possibility is for M_2 to continue reading the string in state q_0. The second option enters the sequence of states q_1, q_2, q_3 to check if the next three symbols complete the substring *abba*.

The first thing to observe is that with a nondeterministic machine, there may be multiple computations for an input string. For example, M_2 has five different computations for string *aabbaa*. We will trace the computations using the $\vdash$ notation introduced in Section 5.2.

$[q_0, aabbaa]$	$[q_0, aabbaa]$	$[q_0, aabbaa]$	$[q_0, aabbaa]$	$[q_0, aabbaa]$
$\vdash [q_0, abbaa]$	$\vdash [q_0, abbaa]$	$\vdash [q_0, abbaa]$	$\vdash [q_0, abbaa]$	$\vdash [q_1, abbaa]$
$\vdash [q_0, bbaa]$	$\vdash [q_0, bbaa]$	$\vdash [q_0, bbaa]$	$\vdash [q_1, bbaa]$	
$\vdash [q_0, baa]$	$\vdash [q_0, baa]$	$\vdash [q_0, baa]$	$\vdash [q_2, baa]$	
$\vdash [q_0, aa]$	$\vdash [q_0, aa]$	$\vdash [q_0, aa]$	$\vdash [q_3, aa]$	
$\vdash [q_0, a]$	$\vdash [q_0, a]$	$\vdash [q_1, a]$	$\vdash [q_4, a]$	
$\vdash [q_0, \lambda]$	$\vdash [q_1, \lambda]$		$\vdash [q_4, \lambda]$	

What does it mean for a string to be accepted when there are some computations that halt in an accepting state and others that halt in a rejecting state? The answer lies in the use of the word *check* in the preceding paragraph. An NFA is designed to check whether a condition is satisfied, in this case, whether the input string has a substring *abba*. If one of the computations discovers the presence of the substring, the condition is satisfied and the string is accepted. As with incompletely specified DFAs, it is necessary to read the entire string to receive an affirmative answer. Summing up, a string is accepted by an NFA if there is at least one computation that

i) processes the entire string, and

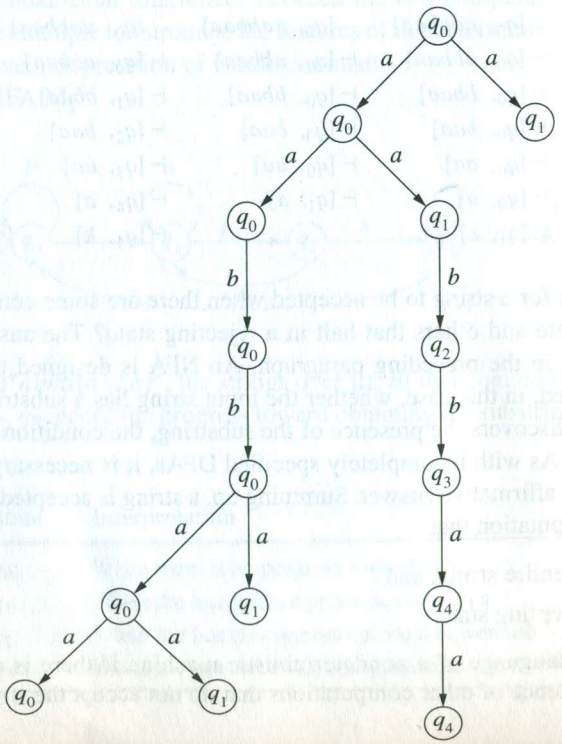
ii) halts in an accepting state.

A string is in the language of a nondeterministic machine if there is a computation that accepts it; the existence of other computations that do not accept the string is irrelevant.

Nondeterministic machines are frequently designed to employ a "guess and check" strategy. The transition from q_0 to q_1 in M_2 represents the guess that the a being read is the first symbol in the substring $abba$. After the guess, the computation continues to states q_1, q_2, and q_3 to check whether the guess is correct. If symbols following the guess are bba, the string is accepted.

If an input string has the substring $abba$, one of the guesses will cause M_2 to enter state q_1 upon reading the initial a in the substring, and this computation accepts the string. Moreover, M_2 enters q_4 only upon processing $abba$. Consequently, the language of M_2 is $(a \cup b)^*abba(a \cup b)^*$. It should be noted that accepting computations are not necessarily unique; there are two distinct accepting computations for $abbabba$ in M_2.

If this is your first encounter with nondeterminism, it is reasonable to ask about the ability of a machine to perform this type of computation. DFAs can be easily implemented in either software or hardware. What is the analogous implementation for NFAs? We can intuitively imagine nondeterministic computation as a type of multiprocessing. When the computation enters a machine configuration for which there are multiple transitions, a new process is generated for each alternative. With this interpretation, a computation produces a tree of processes running in parallel with the branching generated by the multiple choices in the NFA. The tree corresponding to the computation of $aabbaa$ is



If one of the branches reads the entire string and halts in an accepting state, the input is accepted and the entire computation terminates. The input is rejected only when all branches terminate without accepting the string.

Having introduced the properties of nondeterministic computation in the preceding example, we now present the formal definitions of nondeterministic machines, their state diagrams, and their languages. With the exception of the transition function, the components of an NFA are identical to those of a DFA.

Definition 5.4.1

A **nondeterministic finite automaton (NFA)** is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ a finite set called the *alphabet*, $q_0 \in Q$ a distinguished state known as the *start state*, F a subset of Q called the *final* or *accepting states*, and δ a total function from $Q \times \Sigma$ to $\mathcal{P}(Q)$ known as the *transition function*.

Definition 5.4.2

The **language** of an NFA M, denoted L(M), is the set of strings accepted by the M. That is, $L(M) = \{w \mid \text{there is a computation } [q_0, w] \vdash^* [q_i, \lambda] \text{ with } q_i \in F\}$.

Definition 5.4.3

The **state diagram** of an NFA $M = (Q, \Sigma, \delta, q_0, F)$ is a labeled directed graph G defined by the following conditions:

i) The nodes of G are elements of Q.

ii) The labels on the arcs of G are elements of Σ.

iii) q_0 is the start node.

iv) F is the set of accepting nodes.

v) There is an arc from node q_i to q_j labeled a, if $q_j \in \delta(q_i, a)$.

The relationship between DFAs and NFAs is clearly exhibited by comparing the properties of the corresponding state diagrams. Definition 5.4.3 is obtained from Definition 5.3.1 by omitting condition (vi), which translates the deterministic property of the DFA transition function into its graphic representation.

The relationship between DFAs and NFAs can be summarized by the seemingly paradoxical phrase, "Every deterministic finite automaton is nondeterministic." The transition function of a DFA specifies exactly one transition for each combination of state and input symbol, while an NFA allows zero, one, or more transitions. By interpreting the transition function of a DFA as a function from $Q \times \Sigma$ to singleton sets of states, the family of DFAs may be considered to be a subset of the family of NFAs.

The following example describes an NFA in terms of the components in the formal definition. We then construct the corresponding state diagram using the technique outlined in Definition 5.4.3.

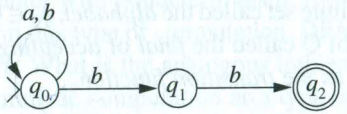
Example 5.4.1

The NFA

$M: Q = \{q_0, q_1, q_2\}$
$\Sigma = \{a, b\}$
$F = \{q_2\}$

δ	a	b
q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\emptyset$	$\{q_2\}$
q_2	$\emptyset$	$\emptyset$

with start state q_0 accepts the language $(a \cup b)^*bb$. The state diagram of M is



Pictorially, it is clear that a string is accepted if, and only if, it ends with the substring bb.

As noted previously, an NFA may have multiple computations for an input string. The three computations for the string $ababb$ are

$[q_0, ababb]$
$\vdash [q_0, babb]$
$\vdash [q_0, abb]$
$\vdash [q_0, bb]$
$\vdash [q_0, b]$
$\vdash [q_0, \lambda]$

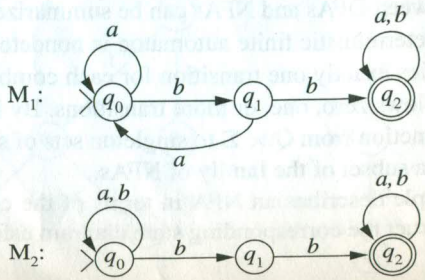
$[q_0, ababb]$
$\vdash [q_0, babb]$
$\vdash [q_1, abb]$

$[q_0, ababb]$
$\vdash [q_0, babb]$
$\vdash [q_0, abb]$
$\vdash [q_0, bb]$
$\vdash [q_1, b]$
$\vdash [q_2, \lambda]$

The second computation halts after the execution of three instructions since no action is specified when the machine is in state q_1 scanning an a. The first computation processes the entire input and halts in a rejecting state while the final computation halts in an accepting state. The third computation demonstrates that $ababb$ is in the language of machine M. □

Example 5.4.2

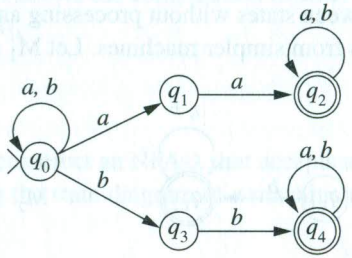
The state diagrams M_1 and M_2 define finite automata that accept $(a \cup b)^*bb(a \cup b)^*$.



M_1 is the DFA from Example 5.3.1. The path exhibiting the acceptance of strings by M_1 enters q_2 when the first substring bb is encountered. M_2 can enter the accepting state upon processing any occurrence of bb. □

Example 5.4.3

An NFA that accepts strings over $\{a, b\}$ with the substring aa or bb can be constructed by combining a machine that accepts strings with bb (Example 5.4.2) with a similar machine that accepts strings with aa.



A path exhibiting the acceptance of a string reads the input in state q_0 until an occurrence of the substring aa or bb is encountered. At this point, the path branches to either q_1 or q_3, depending upon the substring. There are three distinct paths that exhibit the acceptance of the string $abaaabb$. □

The flexibility permitted by the use of nondeterminism does not always simplify the problem of constructing a machine that accepts $L(M_1) \cup L(M_2)$ from the machines M_1 and M_2. This can be seen by attempting to construct an NFA that accepts the language of the DFA in Example 5.3.3.

5.5 λ-Transitions

The transitions from state to state in both deterministic and nondeterministic automata were initiated by processing an input symbol. The definition of NFA is now relaxed to allow state transitions without requiring input to be processed. A transition of this form is called a **λ-transition**. The class of nondeterministic machines that utilize λ-transitions is denoted NFA-λ.

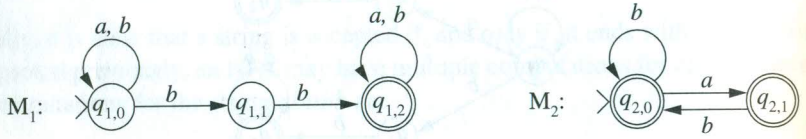
The incorporation of λ-transitions into finite state machines represents another step away from the deterministic computations of a DFA. They do, however, provide a useful tool for the design of machines to accept complex languages.

Definition 5.5.1

A **nondeterministic finite automaton with λ-transitions** is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$, where Q, δ, q_0, and F are the same as in an NFA. The transition function is a function from $Q \times (\Sigma \cup \{\lambda\})$ to $\mathcal{P}(Q)$.

The definition of halting must be extended to include the possibility that a computation may continue using λ-transitions after the input string has been completely processed. Employing the criteria used for acceptance in an NFA, the input is accepted if there is a computation that processes the entire string and halts in an accepting state. As before, the language of an NFA-λ is denoted L(M). The state diagram for an NFA-λ is constructed according to Definition 5.4.3 with λ-transitions represented by arcs labeled by λ.

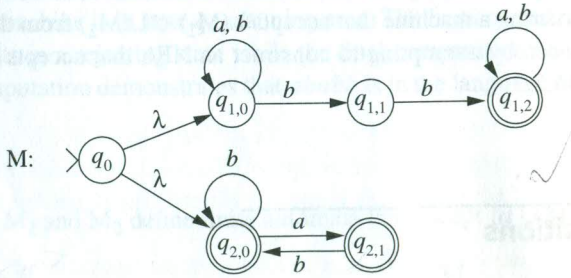
The ability to move between states without processing an input symbol can be used to construct complex machines from simpler machines. Let M_1 and M_2 be the machines



that accept $(a \cup b)^*bb(a \cup b)^*$ and $(b \cup ab)^*(a \cup \lambda)$, respectively. Composite machines are built by appropriately combining the state diagrams of M_1 and M_2.

Example 5.5.1

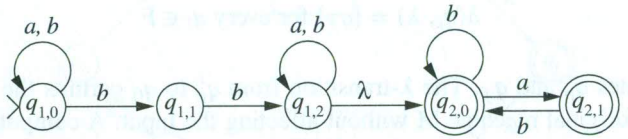
The language of the NFA-λ M is $L(M_1) \cup L(M_2)$.



A computation in the composite machine M begins by following a λ-arc to the start state of either M_1 or M_2. If the path **p** exhibits the acceptance of a string by machine M_i, then that string is accepted by the path in M consisting of the λ-arc from q_0 to $q_{i,0}$ followed by **p** in the copy of the machine M_i. Since the initial move in each computation does not process an input symbol, the language of M is $L(M_1) \cup L(M_2)$. Compare the simplicity of the machine obtained by this construction with that of the deterministic state diagram in Example 5.3.3. □

Example 5.5.2

An NFA-λ that accepts $L(M_1)L(M_2)$, the concatenation of the languages of M_1 and M_2, is constructed by joining the two machines with a λ-arc.

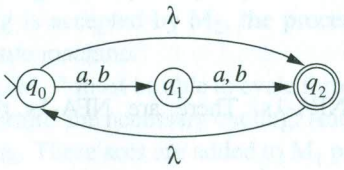


An input string is accepted only if it consists of a string from $L(M_1)$ concatenated with one from $L(M_2)$. The λ-transition allows the computation to enter M_2 whenever a prefix of the input string is accepted by M_1. □

Example 5.5.3

We will use λ-transitions to construct an NFA-λ that accepts all strings of even length over $\{a, b\}$. We begin by building the state diagram of a machine that accepts strings of length two.

To accept the null string, a λ-arc is added from q_0 to q_2. Strings of any positive, even length are accepted by following the λ-arc from q_2 to q_0 to repeat the sequence q_0, q_1, q_2.



□

The constructions presented in Examples 5.5.1, 5.5.2, and 5.5.3 can be generalized to construct machines that accept the union, concatenation, and Kleene star of languages accepted by existing finite-state machines. The first step is to transform the machines into an equivalent NFA-λ whose form is amenable to these constructions.

Lemma 5.5.2

Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA-λ. There is an equivalent NFA-λ $M' = (Q \cup \{q_0', q_f\}, \Sigma, \delta', q_0', \{q_f\})$ that satisfies the following conditions:

i) The in-degree of the start state q_0' is zero.

ii) The only accepting state of M′ is q_f.

iii) The out-degree of the accepting state q_f is zero.

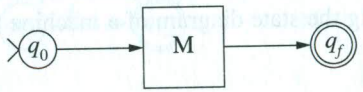
Proof. The transition function of M′ is constructed from that of M by adding the λ-transitions

$$\delta(q_0', \lambda) = \{q_0\}$$

$$\delta(q_i, \lambda) = \{q_f\} \text{ for every } q_i \in F$$

for the new states q_0' and q_f. The λ-transition from q_0' to q_0 permits the computation to proceed to the original machine M without affecting the input. A computation of M′ that accepts an input string is identical to that of M followed by a λ-transition from the accepting state of M to the accepting state q_f of M′. ■

If a machine satisfies the conditions of Lemma 5.5.2, the sole role of the start state is to initiate a computation, and the computation terminates as soon as q_f is entered. Such a machine can be pictured as



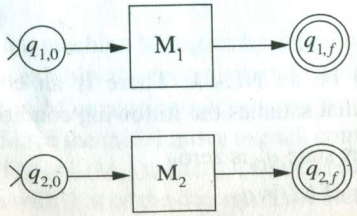
The diagram depicts a machine with three distinct parts: the initial state, the body of the machine, and the final state. This can be likened to a railroad car with couplers on either end. Indeed, the conditions on the start and final state are designed to allow them to act as couplers of finite-state machines.

Theorem 5.5.3

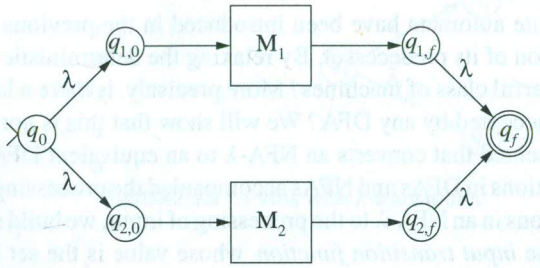
Let M_1 and M_2 be two NFA-λs. There are NFA-λs that accept $L(M_1) \cup L(M_2)$, $L(M_1)L(M_2)$, and $L(M_1)^*$.

Proof. We assume, without loss of generality, that M_1 and M_2 satisfy the conditions of Lemma 5.5.2. The machines constructed to accept the languages $L(M_1) \cup L(M_2)$, $L(M_1)L(M_2)$, and $L(M_1)^*$ will also satisfy the conditions of Lemma 5.5.2.

Because of the restrictions on the start and final states, M_1 and M_2 may be depicted

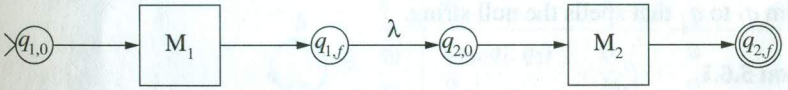


The language $L(M_1) \cup L(M_2)$ is accepted by



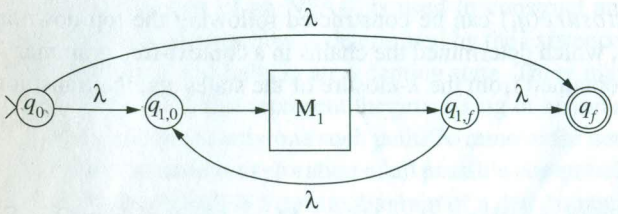
A computation begins by following a λ-arc to M_1 or M_2. If the string is accepted by either of these machines, the λ-arc can be traversed to reach the accepting state of the composite machine. This construction may be thought of as building a machine that runs M_1 and M_2 in parallel. The input is accepted if either of the machines successfully processes the string.

Concatenation can be obtained by operating the component machines sequentially. The start state of the composite machine is $q_{1,0}$ and the accepting state is $q_{2,f}$. The machines are joined by connecting the final state of M_1 to the start state of M_2.



When a prefix of the input string is accepted by M_1, the computation continues with M_2. If the remainder of the string is accepted by M_2, the processing terminates in $q_{2,f}$, the accepting state of the composite machine.

A machine that accepts $L(M_1)^*$ must be able to cycle through M_1 any number of times. The λ-arc from $q_{1,f}$ to $q_{1,0}$ permits the necessary cycling. Another λ-arc is added from $q_{1,0}$ to $q_{1,f}$ to accept the null string. These arcs are added to M_1 producing



■

The ability to repeatedly connect machines of this form will be used in Chapter 6 to establish the equivalence of languages described by regular expressions and accepted by finite-state machines.

5.6 Removing Nondeterminism

Three classes of finite automata have been introduced in the previous sections, each class being a generalization of its predecessor. By relaxing the deterministic restriction, have we created a more powerful class of machines? More precisely, is there a language accepted by an NFA that is not accepted by any DFA? We will show that this is not the case. Moreover, an algorithm is presented that converts an NFA-λ to an equivalent DFA.

The state transitions in DFAs and NFAs accompanied the processing of an input symbol. To relate the transitions in an NFA-λ to the processing of input, we build a modified transition function t, called the *input transition function,* whose value is the set of states that can be entered by processing a single input symbol from a given state. The value of $t(q_1, a)$ for the diagram in Figure 5.3 is the set $\{q_2, q_3, q_5, q_6\}$. State q_4 is omitted since the transition from state q_1 does not process an input symbol.

Intuitively, the definition of the input transition function $t(q_i, a)$ can be broken into three parts. First, the set of states that can be reached from q_i without processing a symbol is constructed. This is followed by processing an a from all the states in that set. Finally, following λ-arcs from the resulting states yields the set $t(q_i, a)$.

The function t is defined in terms of the transition function δ and the paths in the state diagram that spell the null string. A node q_j is said to be in the λ-closure of q_i if there is a path from q_i to q_j that spells the null string.

Definition 5.6.1

The **λ-closure** of a state q_i, denoted $\lambda\text{-}closure(q_i)$, is defined recursively by

i) Basis: $q_i \in \lambda\text{-}closure(q_i)$.

ii) Recursive step: Let q_j be an element of $\lambda\text{-}closure(q_i)$. If $q_k \in \delta(q_j, \lambda)$, then $q_k \in \lambda\text{-}closure(q_i)$.

iii) Closure: q_j is in $\lambda\text{-}closure(q_i)$ only if it can be obtained from q_i by a finite number of applications of the recursive step.

The set $\lambda\text{-}closure(q_i)$ can be constructed following the top-down approach used in Algorithm 4.3.1, which determined the chains in a context-free grammar. The input transition function is obtained from the λ-closure of the states and the transition function of the NFA-λ.

Definition 5.6.2

The **input transition function** t of an NFA-λ M is a function from $Q \times \Sigma$ to $\mathcal{P}(Q)$ defined by

$$t(q_i, a) = \bigcup_{q_j \in \lambda\text{-}closure(q_i)} \lambda\text{-}closure(\delta(q_j, a)),$$

where δ is the transition function of M.

Path	String Processed
q_1, q_2	a
q_1, q_2, q_3	a
q_1, q_4	λ
q_1, q_4, q_5	a
q_1, q_4, q_5, q_6	a

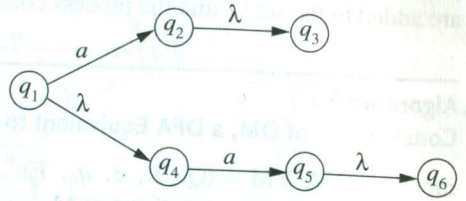
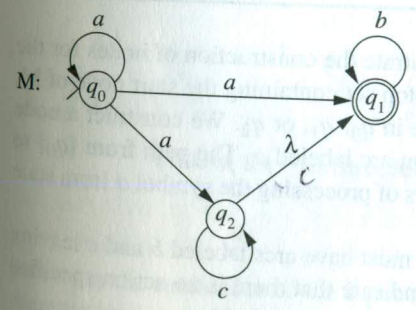


FIGURE 5.3 Paths with λ-transitions.

The input transition function has the same form as the transition function of an NFA. That is, it is a function from $Q \times \Sigma$ to sets of states. For an NFA without λ-transitions, the input transition function t is identical to the transition function δ of the automaton.

Example 5.6.1

Transition tables are given for the transition function δ and the input transition function t of the NFA-λ with state diagram M. The language of M is $a^+c^*b^*$.



δ	a	b	c	λ
q_0	$\{q_0, q_1, q_2\}$	∅	∅	∅
q_1	∅	$\{q_1\}$	∅	∅
q_2	∅	∅	$\{q_2\}$	$\{q_1\}$

t	a	b	c
q_0	$\{q_0, q_1, q_2\}$	∅	∅
q_1	∅	$\{q_1\}$	∅
q_2	∅	$\{q_1\}$	$\{q_1, q_2\}$

□

The input transition function of an NFA-λ is used to construct an equivalent DFA. Acceptance in a nondeterministic machine is determined by the existence of a computation that processes the entire string and halts in an accepting state. There may be several paths in the state diagram of an NFA-λ that represent the processing of an input string, while the state diagram of a DFA contains exactly one such path. To remove the nondeterminism, the DFA must simulate the simultaneous exploration of all possible computations in the NFA-λ.

Algorithm 5.6.3 iteratively builds the state diagram of a deterministic machine equivalent to an NFA-λ M. The nodes of the DFA, called DM for *deterministic equivalent of M*, are sets of nodes of M. The start node of DM is the λ-closure of the start node of M. The key to the algorithm is step 2.1.1, which generates the nodes of the deterministic machine. If X is a node in DM, the set Y is constructed that contains all the states that can be entered by processing the symbol a from any state in the set X. This relationship is represented in the state diagram of DM by an arc from X to Y labeled a. The node X is made deterministic by

producing an arc from it for every symbol in the alphabet. New nodes generated in step 2.1.1 are added to the set Q′ and the process continues until every node in Q′ is deterministic.

Algorithm 5.6.3
Construction of DM, a DFA Equivalent to NFA-λ M

input: an NFA-λ M = (Q, Σ, δ, q_0, F)
input transition function t of M

1. initialize Q′ to λ-*closure*(q_0)
2. **repeat**
   2.1. **if** there is a node X ∈ Q′ and a symbol $a \in \Sigma$ with no arc leaving X labeled a, **then**
      2.1.1. let $Y = \bigcup_{q_i \in X} t(q_i, a)$
      2.1.2. **if** Y ∉ Q′, **then** set Q′ := Q′ ∪ {Y}
      2.1.3. add an arc from X to Y labeled a
   else done := *true*
   until done
3. the set of accepting states of DM is F′ = {X ∈ Q′ | X contains an element $q_i \in$ F}

The NFA-λ from Example 5.6.1 is used to illustrate the construction of nodes for the equivalent DFA. The start node of DM is the singleton set containing the start node of M. A transition from q_0 processing an a can terminate in q_0, q_1, or q_2. We construct a node $\{q_0, q_1, q_2\}$ for the DFA and connect it to $\{q_0\}$ by an arc labeled a. The path from $\{q_0\}$ to $\{q_0, q_1, q_2\}$ in DM represents the three possible ways of processing the symbol a from state q_0 in M.

Since DM is to be deterministic, the node $\{q_0\}$ must have arcs labeled b and c leaving it. Arcs from q_0 to ∅ labeled b and c are added to indicate that there is no action specified by the NFA-λ when the machine is in state q_0 scanning these symbols.

The node $\{q_0\}$ has the deterministic form; there is exactly one arc leaving it for every member of the alphabet. Figure 5.4(a) shows DM at this stage of its construction. Two additional nodes, $\{q_0, q_1, q_2\}$ and ∅, have been created. Both of these must be made deterministic.

An arc leaving node $\{q_0, q_1, q_2\}$ terminates in a node consisting of all the states that can be reached by processing the input symbol from the states q_0, q_1, or q_2 in M. The input transition function $t(q_i, a)$ specifies the states reachable by processing an a from q_i. The arc from $\{q_0, q_1, q_2\}$ labeled a terminates in the set consisting of the union of the $t(q_0, a)$, $t(q_1, a)$, and $t(q_2, a)$. The set obtained from this union is again $\{q_0, q_1, q_2\}$. An arc from $\{q_0, q_1, q_2\}$ to itself is added to the diagram designating this transition.

The empty set represents an error state for DM. A computation enters ∅ on reading an a in state Y only if there is no transition for a for any $q_i \in$ Y. Once in ∅, the computation

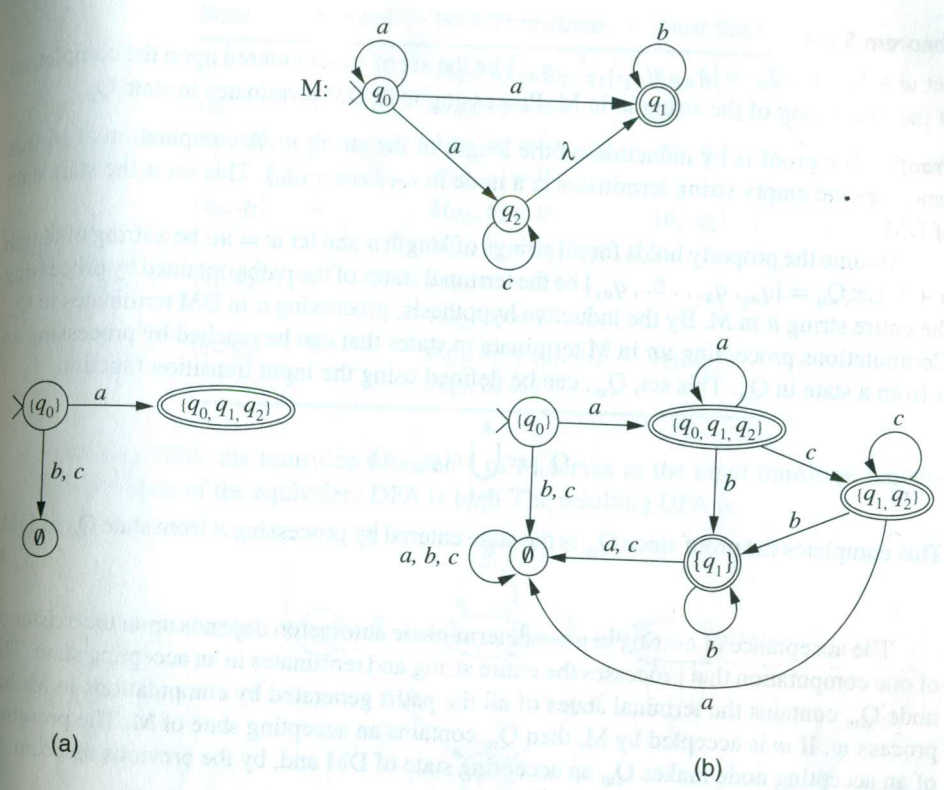


FIGURE 5.4 Construction of equivalent deterministic automaton.

processes the remainder of the input and rejects the string. This is indicated in the state diagram by the arc from ∅ to itself labeled by each alphabet symbol.

Figure 5.4(b) gives the completed deterministic equivalent of the M. Computations of the nondeterministic machine with input *aaa* can terminate in state q_0, q_1, and q_2. The acceptance of the string is exhibited by the path that terminates in q_1. Processing *aaa* in DM terminates in state $\{q_0, q_1, q_2\}$. This state is accepting in DM since it contains the accepting state q_1 of M.

The algorithm for constructing the deterministic state diagram consists of repeatedly adding arcs to make the nodes in the diagram deterministic. As arcs are constructed, new nodes may be created and added to the diagram. The procedure terminates when all the nodes are deterministic. Since each node is a subset of Q, at most $card(\mathcal{P}(Q))$ nodes can be constructed. Algorithm 5.6.3 always terminates since $card(\mathcal{P}(Q))card(\Sigma)$ is an upper bound on the number of iterations of the repeat-until loop. Theorem 5.6.4 establishes the equivalence of M and DM.

Theorem 5.6.4

Let $w \in \Sigma^*$ and $Q_w = \{q_{w_1}, q_{w_2}, \ldots, q_{w_j}\}$ be the set of states entered upon the completion of the processing of the string w in M. Processing w in DM terminates in state Q_w.

Proof. The proof is by induction on the length of the string w. A computation of M that processes the empty string terminates at a node in $\lambda\text{-}closure(q_0)$. This set is the start state of DM.

Assume the property holds for all strings of length n and let $w = ua$ be a string of length $n + 1$. Let $Q_u = \{q_{u_1}, q_{u_2}, \ldots, q_{u_k}\}$ be the terminal states of the paths obtained by processing the entire string u in M. By the inductive hypothesis, processing u in DM terminates in Q_u. Computations processing ua in M terminate in states that can be reached by processing an a from a state in Q_u. This set, Q_w, can be defined using the input transition function:

$$Q_w = \bigcup_{i=1}^{k} t(q_{u_i}, a).$$

This completes the proof since Q_w is the state entered by processing a from state Q_u of DM. ■

The acceptance of a string in a nondeterministic automaton depends upon the existence of one computation that processes the entire string and terminates in an accepting state. The node Q_w contains the terminal states of all the paths generated by computations in M that process w. If w is accepted by M, then Q_w contains an accepting state of M. The presence of an accepting node makes Q_w an accepting state of DM and, by the previous theorem, w is accepted by DM.

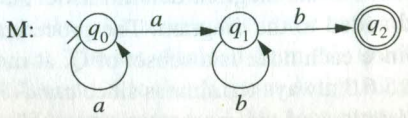
Conversely, let w be a string accepted by DM. Then Q_w contains an accepting state of M. The construction of Q_w guarantees the existence of a computation in M that processes w and terminates in that accepting state. These observations provide the justification for Corollary 5.6.5.

Corollary 5.6.5

The finite automata M and DM are equivalent.

Example 5.6.2

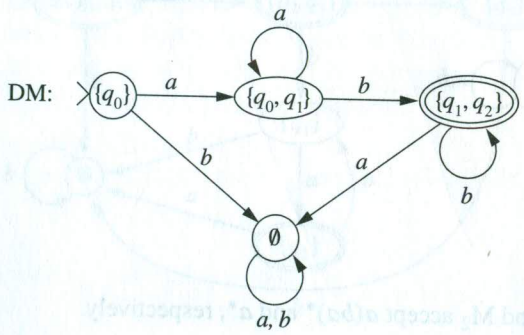
The NFA



accepts the language a^+b^+. The construction of an equivalent DFA is traced in the following table.

State	Symbol	NFA Transitions	Next State
$\{q_0\}$	a	$\delta(q_0, a) = \{q_0, q_1\}$	$\{q_0, q_1\}$
$\{q_0\}$	b	$\delta(q_0, b) = \emptyset$	$\emptyset$
$\{q_0, q_1\}$	a	$\delta(q_0, a) = \{q_0, q_1\}$ $\delta(q_1, a) = \emptyset$	$\{q_0, q_1\}$
$\{q_0, q_1\}$	b	$\delta(q_0, b) = \emptyset$ $\delta(q_1, b) = \{q_1, q_2\}$	$\{q_1, q_2\}$
$\{q_1, q_2\}$	a	$\delta(q_1, a) = \emptyset$ $\delta(q_2, a) = \emptyset$	$\emptyset$
$\{q_1, q_2\}$	b	$\delta(q_1, b) = \{q_1, q_2\}$ $\delta(q_2, b) = \emptyset$	$\{q_1, q_2\}$

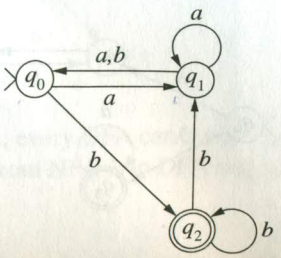
Since M is an NFA, the transition function δ of M serves as the input transition function and the start state of the equivalent DFA is $\{q_0\}$. The resulting DFA is



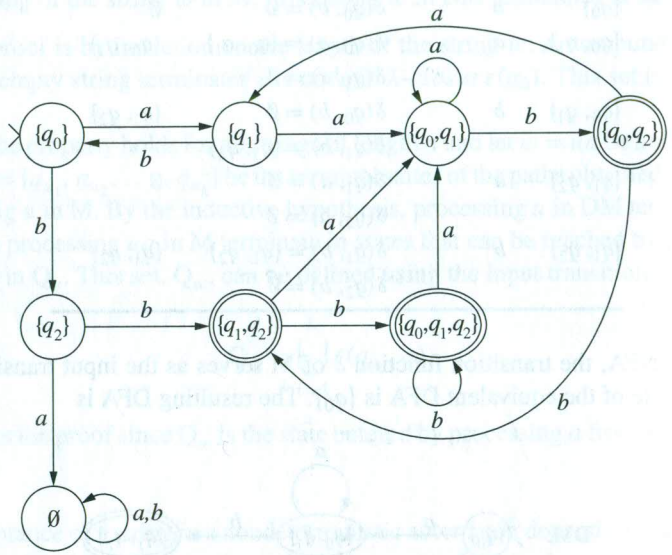
□

Example 5.6.3

As seen in the preceding examples, the states of the DFA constructed using Algorithm 5.6.3 are sets of states of the original nondeterministic machine. If the nondeterministic machine has n states, the DFA may have 2^n states. The transformation of the NFA



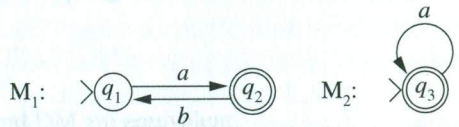
shows that the theoretical upper bound on the number of states may be attained. The start state of DM is $\{q_0\}$ since M does not have λ-transitions.



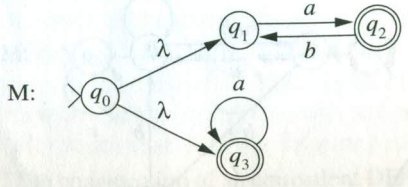
□

Example 5.6.4

The machines M_1 and M_2 accept $a(ba)^*$ and a^*, respectively.



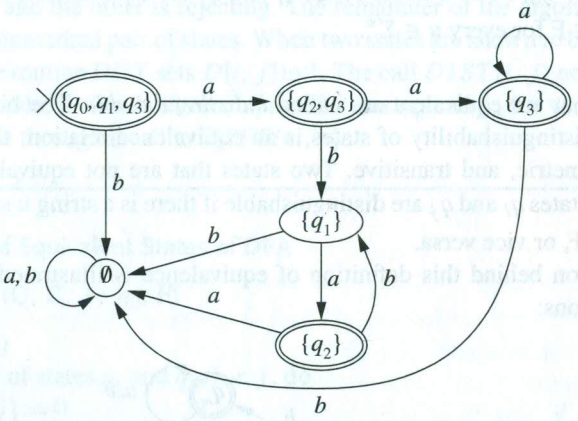
Using λ-arcs to connect a new start state to the start states of the original machines creates an NFA-λ M that accepts $a(ba)^* \cup a^*$.



The input transition function for M is

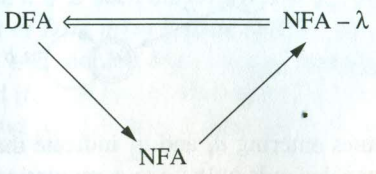
t	a	b
q_0	$\{q_2, q_3\}$	Ø
q_1	$\{q_2\}$	Ø
q_2	Ø	$\{q_1\}$
q_3	$\{q_3\}$	Ø

The equivalent DFA obtained from Algorithm 5.6.3 is



□

Algorithm 5.6.3 completes the following cycle describing the relationships between the classes of finite automata.



The arrows represent inclusion; every DFA can be reformulated as an NFA that is, in turn, an NFA-λ. The double arrow from NFA-λ to DFA indicates the existence of an equivalent deterministic machine.

5.7 DFA Minimization

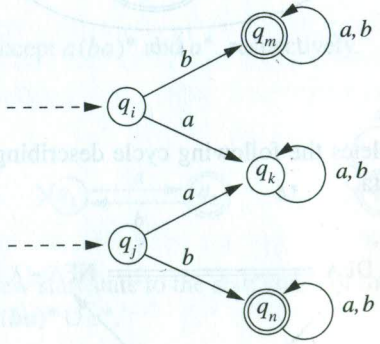
The preceding sections established that the family of languages accepted by DFAs is the same as that accepted by NFAs and NFA-λs. The flexibility of nondeterminism and λ-transitions aid in the design of machines to accept complex languages. The nondeterministic machine can then be transformed into an equivalent deterministic machine using Algorithm 5.6.3. The resulting DFA, however, may not be the minimal DFA that accepts the language. This section presents a reduction algorithm that produces the minimal state DFA accepting the language L from any DFA that accepts L. To accomplish the reduction, the notion of equivalent states in a DFA is introduced.

Definition 5.7.1

Let M = (Q, Σ, δ, q_0, F) be a DFA. States q_i and q_j are equivalent if $\hat{\delta}(q_i, u) \in F$ if, and only if, $\hat{\delta}(q_j, u) \in F$ for every $u \in \Sigma^*$.

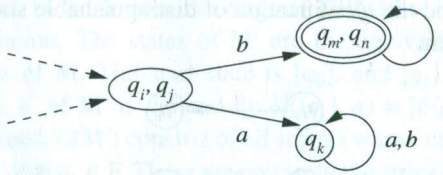
Two states that are equivalent are called *indistinguishable*. The binary relation over Q defined by indistinguishability of states is an equivalence relation; that is, the relation is reflexive, symmetric, and transitive. Two states that are not equivalent are said to be *distinquishable*. States q_i and q_j are distinguishable if there is a string u such that $\hat{\delta}(q_i, u) \in F$ and $\hat{\delta}(q_j, u) \notin F$, or vice versa.

The motivation behind this definition of equivalence is illustrated by the following states and transitions:



The unlabeled dotted lines entering q_i and q_j indicate that the method of reaching a state is irrelevant; equivalence depends only upon computations from the state. The states q_i and q_j are equivalent since the computation with any string beginning with b from either state halts in an accepting state and all other computations halt in the nonaccepting state q_k. States q_m and q_n are also equivalent; all computations beginning in these states end in an accepting state.

The intuition behind the transformation is that equivalent states may be merged. Applying this to the preceding example yields



To reduce the size of a DFA M by merging states, a procedure for identifying equivalent states must be developed. In the algorithm to accomplish this, each pair of states q_i and q_j, $i < j$, has associated with it values $D[i, j]$ and $S[i, j]$. $D[i, j]$ is set to 1 when it is determined that the states q_i and q_j are distinguishable. $S[m, n]$ contains a set of indices. Index $[i, j]$ is in the set $S[m, n]$ if the distinguishability of q_i and q_j follows from that of q_m and q_n.

The algorithm begins by marking each pair of states q_i and q_j as distinguishable if one is accepting and the other is rejecting. The remainder of the algorithm systematically examines each nonmarked pair of states. When two states are shown to be distinguishable, a call to a recursive routine *DIST* sets $D[i, j]$ to 1. The call $DIST(i, j)$ not only marks q_i and q_j as distinguishable, it also marks each pair of states q_m and q_n for which $[m, n] \in S[i, j]$ as distinguishable through a call to $DIST(m, n)$.

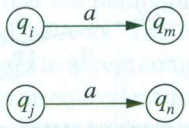
Algorithm 5.7.2
Determination of Equivalent States of DFA

input: DFA M = (Q, Σ, δ, q_0, F)

1. (Initialization)
   for every pair of states q_i and q_j, $i < j$, **do**
   1.1. $D[i, j] := 0$
   1.2. $S[i, j] := \emptyset$
   end for
2. **for** every pair $i, j, i < j$, **if** one of q_i or q_j is an accepting state and the other is not an accepting state, **then** set $D[i, j] := 1$
3. **for** every pair $i, j, i < j$, with $D[i, j] = 0$, **do**
   3.1. **if** there exists an $a \in \Sigma$ such that $\delta(q_i, a) = q_m$, $\delta(q_j, a) = q_n$ and $D[m, n] = 1$ or $D[n, m] = 1$, **then** $DIST(i, j)$
   3.2. **else** for each $a \in \Sigma$, do: Let $\delta(q_i, a) = q_m$ and $\delta(q_j, a) = q_n$
   if $m < n$ and $[i, j] \neq [m, n]$, **then** add $[i, j]$ to $S[m, n]$
   else if $m > n$ **and** $[i, j] \neq [n, m]$, **then** add $[i, j]$ to $S[n, m]$
   end for

$DIST(i, j)$;
begin
   $D[i, j] := 1$
   for all $[m, n] \in S[i, j]$, $DIST(m, n)$
end

The motivation behind the identification of distinguishable states is illustrated by the relationships in the diagram



If q_m and q_n are already marked as distinguishable when q_i and q_j are examined in step 3, then $D[i, j]$ is set to 1 to indicate the distinguishability of q_i and q_j. If the status of q_m and q_n is not known when q_i and q_j are examined, then a later determination that q_m and q_n are distinguishable also provides the answer for q_i and q_j. The role of the array S is to record this information: $[i, j] \in S[n, m]$ indicates that the distinguishability of q_m and q_n is sufficient to establish the distinguishability of q_i and q_j. These ideas are formalized in the proof of Theorem 5.7.3.

Theorem 5.7.3

States q_i and q_j are distinguishable if, and only if, $D[i, j] = 1$ at the termination of Algorithm 5.7.2.

Proof. First we show that every pair of states q_i and q_j for which $D[i, j] = 1$ is distinguishable. If $D[i, j]$ is assigned 1 in the step 2, then q_i and q_j are distinguishable by the null string. Step 3.1 marks q_i and q_j as distinguishable only if $\delta(q_i, a) = q_m$ and $\delta(q_j, a) = q_n$ for some input a when states q_m and q_n have already been determined to be distinguishable by the algorithm. Let u be a string that exhibits the distinguishability of q_m and q_n. Then au exhibits the distinguishability of q_i and q_j.

To complete the proof, it is necessary to show that every pair of distinguishable states is designated as such. The proof is by induction on the length of the shortest string that demonstrates the distinguishability of a pair of states. The basis consists of all pairs of states q_i, q_j that are distinguishable by a string of length 0. That is, the computations $\hat{\delta}(q_i, \lambda) = q_i$ and $\hat{\delta}(q_j, \lambda) = q_j$ distinguish q_i from q_j. In this case, exactly one of q_i or q_j is accepting and the position $D[i, j]$ is set to 1 in step 2.

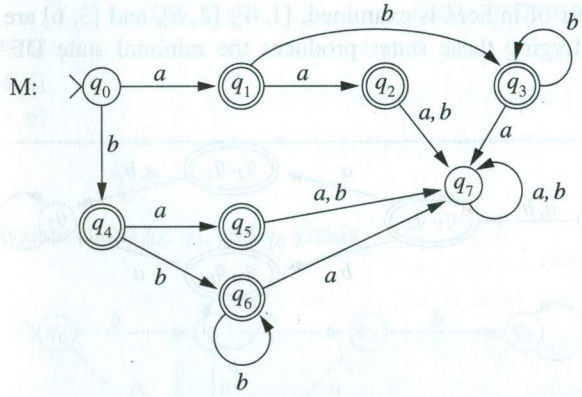
Now assume that every pair of states distinguishable by a string of length k or less is marked by the algorithm. Let q_i and q_j be states for which the shortest distinguishing string u has length $k + 1$. Then u can be written av and the computations with input u have the form $\hat{\delta}(q_i, u) = \hat{\delta}(q_i, av) = \hat{\delta}(q_m, v) = q_s$ and $\hat{\delta}(q_j, u) = \hat{\delta}(q_j, av) = \hat{\delta}(q_n, v) = q_t$. Exactly one of q_s and q_t is accepting since the preceding computations distinguish q_i from q_j. Clearly, the same computations exhibit the distinguishability of q_m from q_n by a string of length k. By induction, we know that the algorithm will set $D[m, n]$ to 1.

If $D[m, n]$ is marked before the states q_i and q_j are examined in step 3, then $D[i, j]$ is set to 1 by the call $DIST(i, j)$. If q_i and q_j are examined in the loop in step 3.1 and $D[m, n] \neq 1$ at that time, then $[i, j]$ is added to the set $S[m, n]$. By the inductive hypothesis, $D[m, n]$ will eventually be set to 1. $D[i, j]$ will also be set to 1 at this time by a recursive call from $DIST(m, n)$ since $[i, j]$ is in $S[m, n]$. ∎

A new DFA M′ can be built from the original DFA $M = (Q, \Sigma, \delta, q_0, F)$ and the indistinguishability relation. The states of M′ are the equivalence classes consisting of indistinguishable states of M. The start state is $[q_0]$, and $[q_i]$ is a final state if $q_i \in F$. The transition function δ' of M′ is defined by $\delta'([q_i], a) = [\delta(q_i, a)]$. In Exercise 44, δ' is shown to be well defined. L(M′) consists of all strings whose computations have the form $\hat{\delta}'([q_0], u) = [\hat{\delta}(q_i, \lambda)]$ with $q_i \in F$. These are precisely the strings accepted by M. If M′ has states that are unreachable by computations from $[q_0]$, these states and all associated arcs are deleted.

Example 5.7.1

The minimization process is exhibited using the DFA M



that accepts the language $(a \cup b)(a \cup b^*)$.

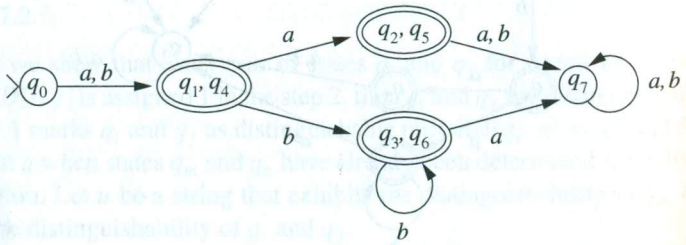
In step 2, $D[0, 1]$, $D[0, 2]$, $D[0, 3]$, $D[0, 4]$, $D[0, 5]$, $D[0, 6]$, $D[1, 7]$, $D[2, 7]$, $D[3, 7]$, $D[4, 7]$, $D[5, 7]$, and $D[6, 7]$ are set to 1. Each index not marked in step 2 is examined in step 3. The table shows the action taken for each such index.

Index	Action	Reason
[0, 7]	$D[0, 7] = 1$	Distinguished by a
[1, 2]	$D[1, 2] = 1$	Distinguished by a
[1, 3]	$D[1, 3] = 1$	Distinguished by a
[1, 4]	$S[2, 5] = \{[1, 4]\}$ $S[3, 6] = \{[1, 4]\}$	
[1, 5]	$D[1, 5] = 1$	Distinguished by a
[1, 6]	$D[1, 6] = 1$	Distinguished by a
[2, 3]	$D[2, 3] = 1$	Distinguished by b

(Continued)

Index	Action	Reason
[2, 4]	$D[2, 4] = 1$	Distinguished by a
[2, 5]		No action since $\delta(q_2, x) = \delta(q_5, x)$ for every $x \in \Sigma$
[2, 6]	$D[2, 6] = 1$	Distinguished by b
[3, 4]	$D[3, 4] = 1$	Distinguished by a
[3, 5]	$D[3, 5] = 1$	Distinguished by b
[3, 6]		
[4, 5]	$D[4, 5] = 1$	Distinguished by a
[4, 6]	$D[4, 6] = 1$	Distinguished by a
[5, 6]	$D[5, 6] = 1$	Distinguished by b

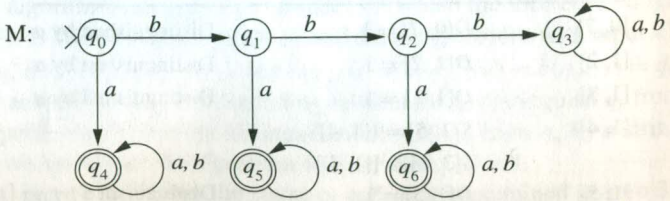
After each pair of indices is examined, [1, 4], [2, 5], and [3, 6] are left as equivalent pairs of states. Merging these states produces the minimal state DFA M′ that accepts $(a \cup b)(a \cup b^*)$.



□

Example 5.7.2

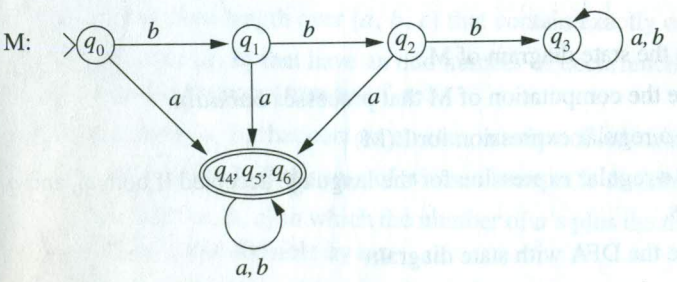
Minimizing the DFA M illustrates the recursive marking of states by the call to *DIST*. The language of M is $a(a \cup b)^* \cup ba(a \cup b)^* \cup bba(a \cup b)^*$.



The comparison of accepting states to nonaccepting states assigns 1 to $D[0, 4]$, $D[0, 5]$, $D[1, 6]$, $D[2, 4]$, $D[2, 5]$, $D[2, 6]$, $D[3, 4]$, $D[3, 5]$, and $D[3, 6]$.

Index	Action	Reason
[0, 1]	$S[4, 5] = \{[0, 1]\}$	
	$S[1, 2] = \{[0, 1]\}$	
[0, 2]	$S[4, 6] = \{[0, 2]\}$	
	$S[1, 3] = \{[0, 2]\}$	
[0, 3]	$D[0, 3] = 1$	Distinguished by a
[1, 2]	$S[5, 6] = \{[1, 2]\}$	
	$S[2, 3] = \{[1, 2]\}$	
[1, 3]	$D[1, 3] = 1$	Distinguished by a
	$D[0, 2] = 1$	Call to *DIST*(1, 3)
[2, 3]	$D[2, 3] = 1$	Distinguished by a
	$D[1, 2] = 1$	Call to *DIST*(1, 2)
	$D[0, 1] = 1$	Call to *DIST*(0, 1)
[4, 5]		
[4, 6]		
[5, 6]		

Merging equivalent states q_4, q_5, and q_6 yields



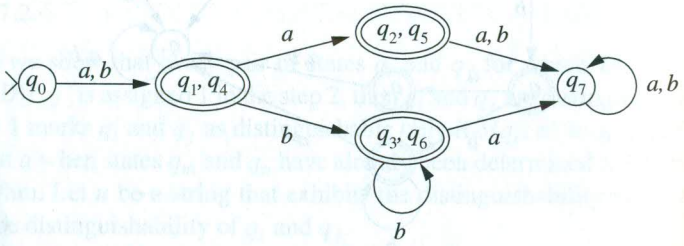
□

The minimization algorithm completes the sequence of algorithms required for the construction of optimal DFAs. Nondeterminism and λ-transitions provide tools for designing finite automata to match complicated patterns or to accept complex languages. Algorithm 5.6.3 can then be used to transform the nondeterministic machine into a DFA, which may not be minimal. Algorithm 5.7.2 completes the process by producing the minimal state DFA.

For the moment, we have presented an algorithm for DFA reduction but have not established that it produces the minimal DFA. In Section 6.7 we prove the Myhill-Nerode Theorem, which characterizes the language accepted by a finite automaton in terms of equivalence classes of strings. This characterization will then be used to prove that the machine M′ produced by Algorithm 5.7.2 is the unique minimal state DFA that accepts L.

Index	Action	Reason
[2, 4]	$D[2, 4] = 1$	Distinguished by a
[2, 5]		No action since $\delta(q_2, x) = \delta(q_5, x)$ for every $x \in \Sigma$
[2, 6]	$D[2, 6] = 1$	Distinguished by b
[3, 4]	$D[3, 4] = 1$	Distinguished by a
[3, 5]	$D[3, 5] = 1$	Distinguished by b
[3, 6]		
[4, 5]	$D[4, 5] = 1$	Distinguished by a
[4, 6]	$D[4, 6] = 1$	Distinguished by a
[5, 6]	$D[5, 6] = 1$	Distinguished by b

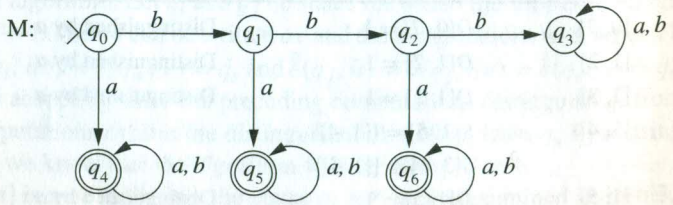
After each pair of indices is examined, [1, 4], [2, 5], and [3, 6] are left as equivalent pairs of states. Merging these states produces the minimal state DFA M′ that accepts $(a \cup b)(a \cup b^*)$.



□

Example 5.7.2

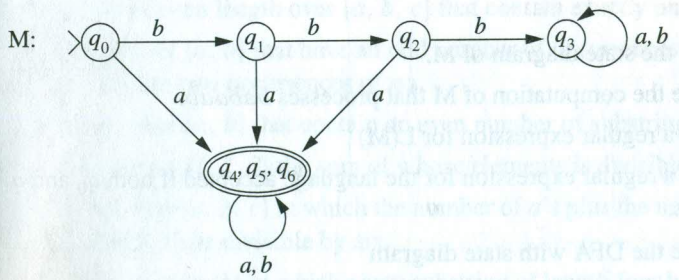
Minimizing the DFA M illustrates the recursive marking of states by the call to *DIST*. The language of M is $a(a \cup b)^* \cup ba(a \cup b)^* \cup bba(a \cup b)^*$.



The comparison of accepting states to nonaccepting states assigns 1 to $D[0, 4]$, $D[0, 5]$, $D[0, 6]$, $D[1, 4]$, $D[1, 5]$, $D[1, 6]$, $D[2, 4]$, $D[2, 5]$, $D[2, 6]$, $D[3, 4]$, $D[3, 5]$, and $D[3, 6]$. Tracing the algorithm produces

Index	Action	Reason
[0, 1]	$S[4, 5] = \{[0, 1]\}$	
	$S[1, 2] = \{[0, 1]\}$	
[0, 2]	$S[4, 6] = \{[0, 2]\}$	
	$S[1, 3] = \{[0, 2]\}$	
[0, 3]	$D[0, 3] = 1$	Distinguished by a
[1, 2]	$S[5, 6] = \{[1, 2]\}$	
	$S[2, 3] = \{[1, 2]\}$	
[1, 3]	$D[1, 3] = 1$	Distinguished by a
	$D[0, 2] = 1$	Call to *DIST*(1, 3)
[2, 3]	$D[2, 3] = 1$	Distinguished by a
	$D[1, 2] = 1$	Call to *DIST*(1, 2)
	$D[0, 1] = 1$	Call to *DIST*(0, 1)
[4, 5]		
[4, 6]		
[5, 6]		

Merging equivalent states q_4, q_5, and q_6 yields



□

The minimization algorithm completes the sequence of algorithms required for the construction of optimal DFAs. Nondeterminism and λ-transitions provide tools for designing finite automata to match complicated patterns or to accept complex languages. Algorithm 5.6.3 can then be used to transform the nondeterministic machine into a DFA, which may not be minimal. Algorithm 5.7.2 completes the process by producing the minimal state DFA.

For the moment, we have presented an algorithm for DFA reduction but have not established that it produces the minimal DFA. In Section 6.7 we prove the Myhill-Nerode Theorem, which characterizes the language accepted by a finite automaton in terms of equivalence classes of strings. This characterization will then be used to prove that the machine M′ produced by Algorithm 5.7.2 is the unique minimal state DFA that accepts L.

Exercises

1. Let M be the deterministic finite automaton defined by

$Q = \{q_0, q_1, q_2\}$

$\Sigma = \{a, b\}$

$F = \{q_2\}$

δ	a	b
q_0	q_0	q_1
q_1	q_2	q_1
q_2	q_2	q_0

a) Give the state diagram of M.

b) Trace the computations of M that process the strings *abaa*, *bbbabb*, *bababa*, and *bbbaa*.

c) Which of the strings from part (b) are accepted by M?

d) Give a regular expression for L(M).

2. Let M be the deterministic finite automaton

$Q = \{q_0, q_1, q_2\}$

$\Sigma = \{a, b\}$

$F = \{q_0\}$

δ	a	b
q_0	q_1	q_0
q_1	q_1	q_2
q_2	q_1	q_0

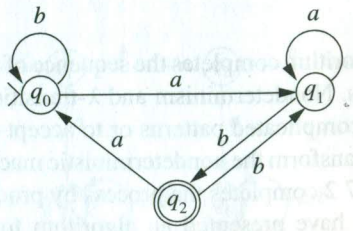
a) Give the state diagram of M.

b) Trace the computation of M that processes *babaab*.

c) Give a regular expression for L(M).

d) Give a regular expression for the language accepted if both q_0 and q_1 are accepting states.

3. Let M be the DFA with state diagram



a) Construct the transition table of M.

b) Which of the strings *baba*, *baab*, *abab*, *abaaab* are accepted by M?

c) Give a regular expression for L(M).

*4. The recursive step in the definition of the extended transition function (Definition 5.2.4) may be replaced by $\hat{\delta}'(q_i, au) = \hat{\delta}'(\delta(q_i, a), u)$, for all $u \in \Sigma^*$, $a \in \Sigma$, and $q_i \in Q$. Prove that $\hat{\delta} = \hat{\delta}'$.

For Exercises 5 through 21, build a DFA that accepts the described language.

5. The set of strings over $\{a, b, c\}$ in which all the a's precede the b's, which in turn precede the c's. It is possible that there are no a's, b's, or c's.

6. The set of strings over $\{a, b\}$ in which the substring aa occurs at least twice.

7. The set of strings over $\{a, b\}$ that do not begin with the substring aaa.

8. The set of strings over $\{a, b\}$ that do not contain the substring aaa.

9. The set of strings over $\{a, b, c\}$ that begin with a, contain exactly two b's, and end with cc.

10. The set of strings over $\{a, b, c\}$ in which every b is immediately followed by at least one c.

11. The set of strings over $\{a, b\}$ in which the number of a's is divisible by three.

12. The set of strings over $\{a, b\}$ in which every a is either immediately preceded or immediately followed by b, for example, *baab*, *aba*, and *b*.

13. The set of strings of odd length over $\{a, b\}$ that contain the substring bb.

14. The set of strings over $\{a, b\}$ that have odd length or end with aaa.

15. The set of strings of even length over $\{a, b, c\}$ that contain exactly one a.

16. The set of strings over $\{a, b\}$ that have an odd number of occurrences of the substring aa. Note that aaa has two occurrences of aa.

17. The set of strings over $\{a, b\}$ that contain an even number of substrings ba.

18. The set of strings over $\{1, 2, 3\}$ the sum of whose elements is divisible by six.

19. The set of strings over $\{a, b, c\}$ in which the number of a's plus the number of b's plus twice the number of c's is divisible by six.

20. The set of strings over $\{a, b\}$ in which every substring of length four has at least one b. Note that every substring with length less than four is in this language.

*21. The set of strings over $\{a, b, c\}$ in which every substring of length four has exactly one b.

22. For each of the following languages, give the state diagram of a DFA that accepts the languages.

a) $(ab)^*ba$

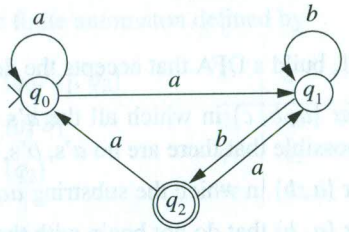
b) $(ab)^*(ba)^*$

c) $aa(a \cup b)^+bb$

d) $((aa)^+bb)^*$

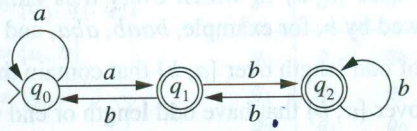
e) $(ab^*a)^*$

et M be the nondeterministic finite automaton



) Construct the transition table of M.
) Trace all computations of the string *aaabb* in M.
) Is *aaabb* in L(M)?
) Give a regular expression for L(M).

et M be the nondeterministic finite automaton



) Construct the transition table of M.
) Trace all computations of the string *aabb* in M.
) Is *aabb* in L(M)?
) Give a regular expression for L(M).
) Construct a DFA that accepts L(M).
) Give a regular expression for the language accepted if both q_0 and q_1 are accepting states.

For each of the following languages, give the state diagram of an NFA that accepts the anguages.

a) $(a \cup ab \cup aab)^*$
b) $(ab)^* \cup a^*$
c) $(abc)^*a^*$
d) $(ba \cup bb)^* \cup (ab \cup aa)^*$
e) $(ab^+a)^+$

Give a recursive definition of the extended transition function $\hat{\delta}$ of an NFA-λ. The value $\hat{\delta}(q_i, w)$ is the set of states that can be reached by computations that begin at node q_i and completely process the string w.

For Exercises 27 through 34, give the state diagram of an NFA that accepts the given language. Remember that an NFA may be deterministic, but you should use nondeterminism whenever it is appropriate.

27. The set of strings over $\{a, b\}$ that contain either *aa* and *bb* as substrings.

28. The set of strings over $\{a, b\}$ that contain both or neither *aa* and *bb* as substrings.

* **29.** The set of strings over $\{a, b\}$ whose third-to-the-last symbol is *b*.

30. The set of strings over $\{a, b\}$ whose third and third-to-last symbols are both *b*. For example, *aababaa*, *abbbbbbbb*, and *abba* are in the language.

31. The set of strings over $\{a, b\}$ in which every *a* is followed by *b* or *ab*.

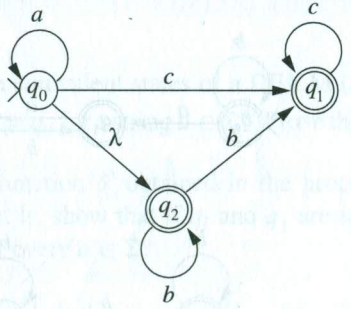
32. The set of strings over $\{a, b\}$ that have a substring of length four that begins and ends with the same symbol.

33. The set of strings over $\{a, b\}$ that contain substrings *aaa* and *bbb*.

34. The set of strings over $\{a, b, c\}$ that have a substring of length three containing each of the symbols exactly once.

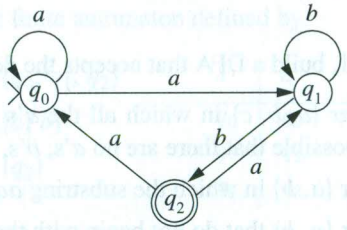
35. Construct the state diagram of a DFA that accepts the strings over $\{a, b\}$ ending with the substring *abba*. Give the state diagram of an NFA with six arcs that accepts the same language.

36. Let M be the NFA-λ



a) Compute λ-*closure*(q_i) for $i = 0, 1, 2$.
b) Give the input transition function t for M.
c) Use Algorithm 5.6.3 to construct a state diagram of a DFA that is equivalent to M.
d) Give a regular expression for L(M).

23. Let M be the nondeterministic finite automaton



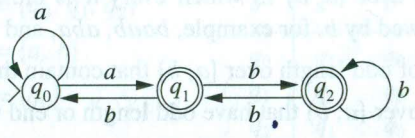
a) Construct the transition table of M.

b) Trace all computations of the string *aaabb* in M.

c) Is *aaabb* in L(M)?

d) Give a regular expression for L(M).

24. Let M be the nondeterministic finite automaton



a) Construct the transition table of M.

b) Trace all computations of the string *aabb* in M.

c) Is *aabb* in L(M)?

d) Give a regular expression for L(M).

e) Construct a DFA that accepts L(M).

f) Give a regular expression for the language accepted if both q_0 and q_1 are accepting states.

25. For each of the following languages, give the state diagram of an NFA that accepts the languages.

a) $(a \cup ab \cup aab)^*$

b) $(ab)^* \cup a^*$

c) $(abc)^*a^*$

d) $(ba \cup bb)^* \cup (ab \cup aa)^*$

e) $(ab^+a)^+$

26. Give a recursive definition of the extended transition function $\hat{\delta}$ of an NFA-λ. The value $\hat{\delta}(q_i, w)$ is the set of states that can be reached by computations that begin at node q_i and completely process the string w.

For Exercises 27 through 34, give the state diagram of an NFA that accepts the given language. Remember that an NFA may be deterministic, but you should use nondeterminism whenever it is appropriate.

27. The set of strings over $\{a, b\}$ that contain either *aa* and *bb* as substrings.

28. The set of strings over $\{a, b\}$ that contain both or neither *aa* and *bb* as substrings.

*29. The set of strings over $\{a, b\}$ whose third-to-the-last symbol is *b*.

30. The set of strings over $\{a, b\}$ whose third and third-to-last symbols are both *b*. For example, *aababaa*, *abbbbbbbb*, and *abba* are in the language.

31. The set of strings over $\{a, b\}$ in which every *a* is followed by *b* or *ab*.

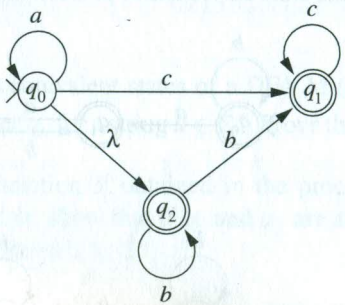
32. The set of strings over $\{a, b\}$ that have a substring of length four that begins and ends with the same symbol.

33. The set of strings over $\{a, b\}$ that contain substrings *aaa* and *bbb*.

34. The set of strings over $\{a, b, c\}$ that have a substring of length three containing each of the symbols exactly once.

35. Construct the state diagram of a DFA that accepts the strings over $\{a, b\}$ ending with the substring *abba*. Give the state diagram of an NFA with six arcs that accepts the same language.

36. Let M be the NFA-λ



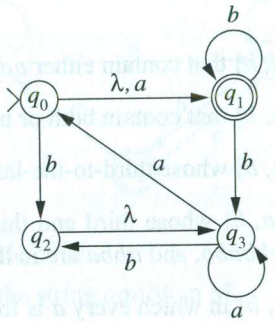
a) Compute λ-$closure(q_i)$ for $i = 0, 1, 2$.

b) Give the input transition function t for M.

c) Use Algorithm 5.6.3 to construct a state diagram of a DFA that is equivalent to M.

d) Give a regular expression for L(M).

37. Let M be the NFA-λ



a) Compute λ-$closure(q_i)$ for $i = 0, 1, 2, 3$.

b) Give the input transition function t for M.

c) Use Algorithm 5.6.3 to construct a state diagram of a DFA that is equivalent to M.

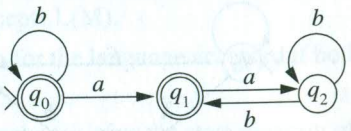
d) Give a regular expression for L(M).

38. Use Algorithm 5.6.3 to construct the state diagram of a DFA equivalent to the NFA in Example 5.5.2.

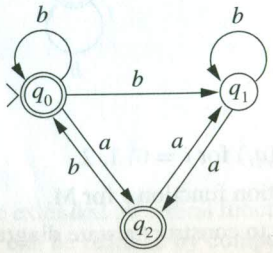
39. Use Algorithm 5.6.3 to construct the state diagram of a DFA equivalent to the NFA in Exercise 17.

40. For each of the following NFAs, use Algorithm 5.6.3 to construct the state diagram of an equivalent DFA.

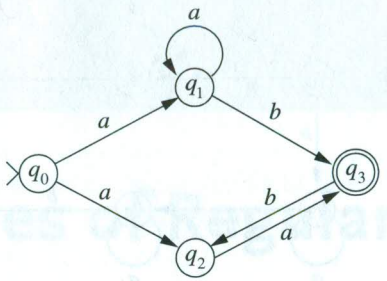
a)



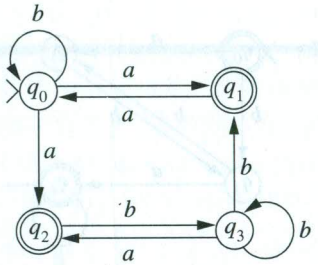
b)



c)



d)



41. Build an NFA M_1 that accepts $(ab)^*$ and an NFA M_2 that accepts $(ba)^*$. Use λ-transitions to obtain a machine M that accepts $(ab)^*(ba)^*$. Give the input transition function of M. Use Algorithm 5.6.3 to construct the state diagram of a DFA that accepts L(M).

42. Build an NFA M_1 that accepts $(aba)^+$ and an NFA M_2 that accepts $(ab)^*$. Use λ-transitions to obtain a machine M that accepts $(aba)^+ \cup (ab)^*$. Give the input transition function of M. Use Algorithm 5.6.3 to construct the state diagram of a DFA that accepts L(M).

43. Assume that q_i and q_j are equivalent states of a DFA M (as in Definition 5.7.1) and $\hat{\delta}(q_i, u) = q_m$ and $\hat{\delta}(q_j, u) = q_n$ for a string $u \in \Sigma^*$. Prove that q_m and q_n are equivalent.

44. Show that the transition function δ' obtained in the process of merging equivalent states is well defined. That is, show that if q_i and q_j are states with $[q_i] = [q_j]$, then $\delta'([q_i], a) = \delta'([q_j], a)$ for every $a \in \Sigma$.

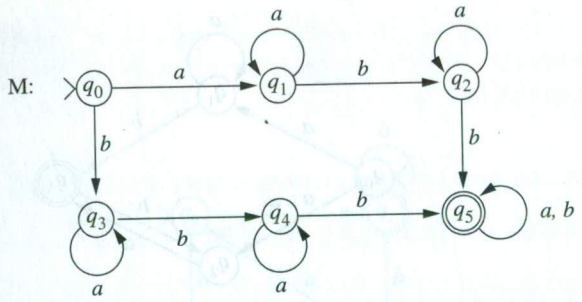
45. For each DFA:

i) Trace the actions of Algorithm 5.7.2 to determine the equivalent states of M. Give the values of $D[i, j]$ and $S[i, j]$ computed by the algorithm.

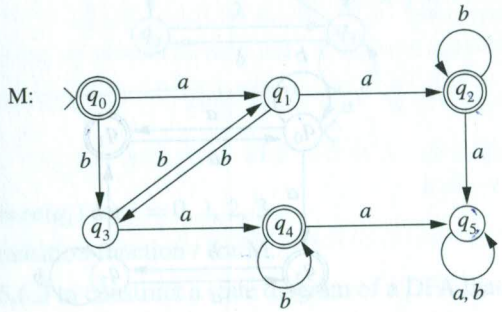
ii) Give the equivalence classes of states.

iii) Give the state diagram of the minimal state DFA that accepts L(M).

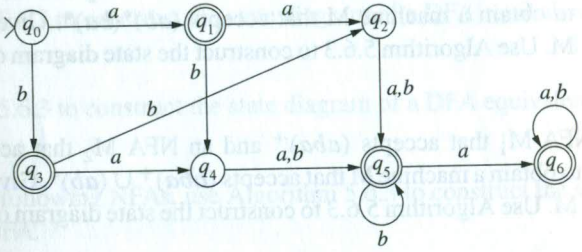
a)



b)



c)



Bibliographic Notes

Alternative interpretations of the result of finite-state computations were studied in Mealy [1955] and Moore [1956]. Transitions in Mealy machines are accompanied by the generation of output. A two-way automaton allows the tape head to move in both directions. A proof that two-way and one-way automata accept the same languages can be found in Rabin and Scott [1959] and Sheperdson [1959]. Nondeterministic finite automata were introduced by Rabin and Scott [1959]. The algorithm for minimizing the number of states in a DFA was presented in Nerode [1958]. The algorithm of Hopcroft [1971] increases the efficiency of the minimization technique.

The theory and applications of finite automata are developed in greater depth in the books by Minsky [1967]; Salomaa [1973]; Denning, Dennis, and Qualitz [1978]; and Bavel [1983].

CHAPTER 6

Properties of Regular Languages

Grammars were introduced as language generators, finite automata as language acceptors, and regular expressions as pattern descriptors. This chapter develops the relationship between these three approaches to language definition and explores the limitations of finite automata as language acceptors.

6.1 Finite-State Acceptance of Regular Languages

In this section we show that an NFA-λ can be constructed to accept any regular language. Regular sets are built recursively from Ø, {λ}, and singleton sets containing elements from the alphabet by applications of union, concatenation, and the Kleene star operation (Definition 2.3.2). The construction of an NFA-λ that accepts a regular set can be obtained following the steps of its recursive generation, but using state diagrams as the building blocks rather than sets.

State diagrams for machines that accept Ø, {λ}, and singleton sets {a} are