Nondeterministic Finite Automata

Shekh. Md. Saifur Rahman

Lecturer, CSE Department

UIU

Nondeterminism

Nondeterminism is a useful concept that has had great impact on the theory of computation.

Nondeterminism is an *inessential feature* of finite automata.

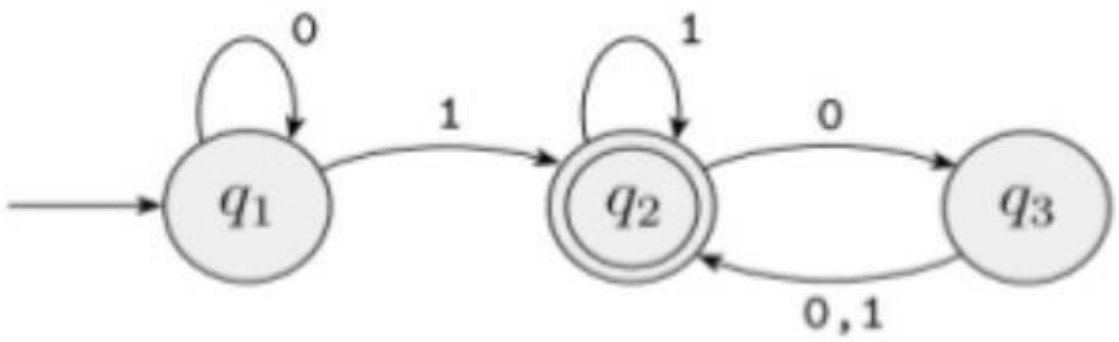
Every nondeterministic finite automaton is equivalent to a deterministic finite automaton.

Thus we shall profit from the powerful notation of nondeterministic finite automata.

But we always know that, if we must, we can always go back and redo everything in terms of the lower-level language of ordinary, down-to-earth deterministic automata.

DFA vs NFA

Every state of a DFA always has exactly one exiting transition arrow for each symbol in the alphabet.



In a DFA, labels on the transition arrows are symbols from the alphabet set Σ

DFA vs NFA

In a ***nondeterministic*** machine, several choices may exist for the next state at any point. In an NFA, a state may have zero, one or more exiting arrows for each alphabet symbol.

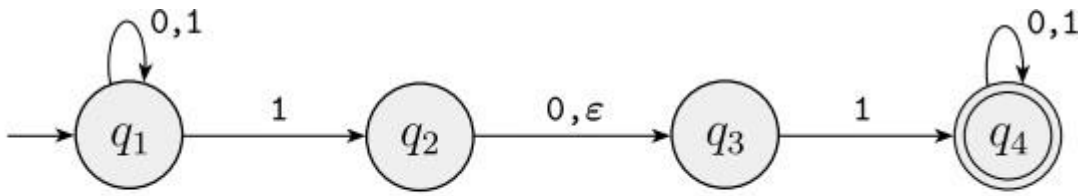
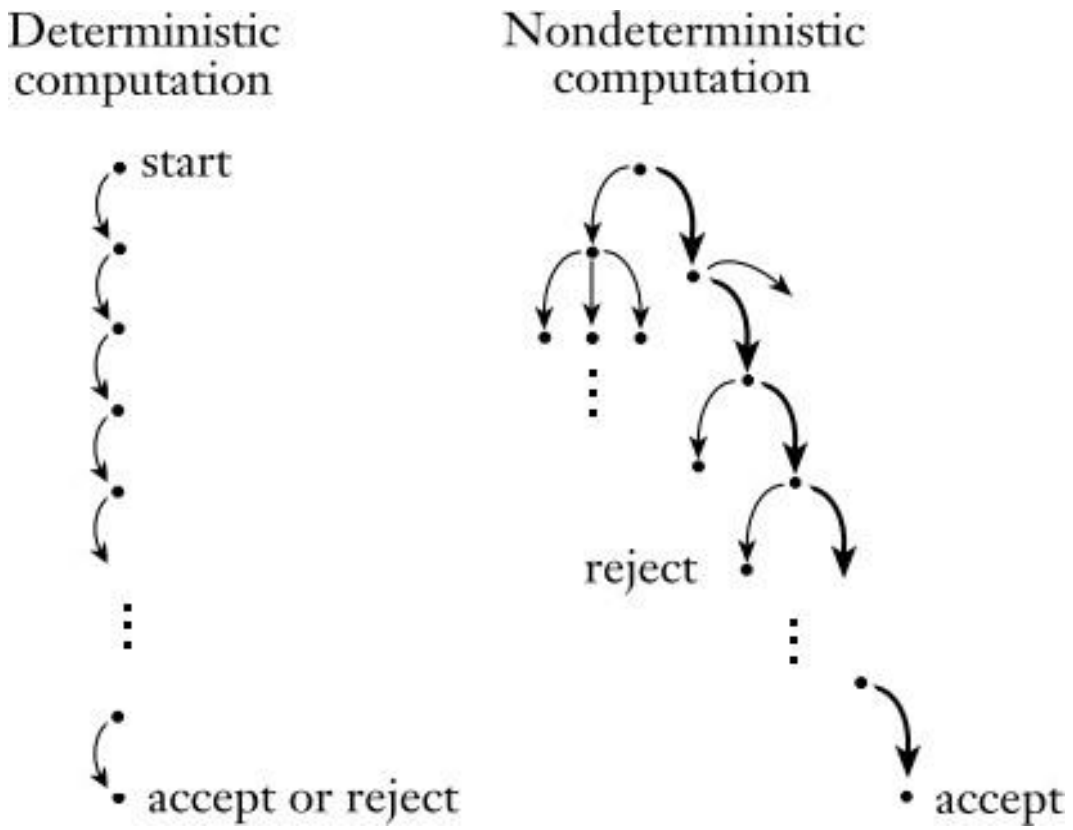


FIGURE **1.27**
The nondeterministic finite automaton N_1

In NFA, labels on the transition arrows are symbols from the alphabet or ε. Zero, one or many arrows may exit from each state with the label ε.

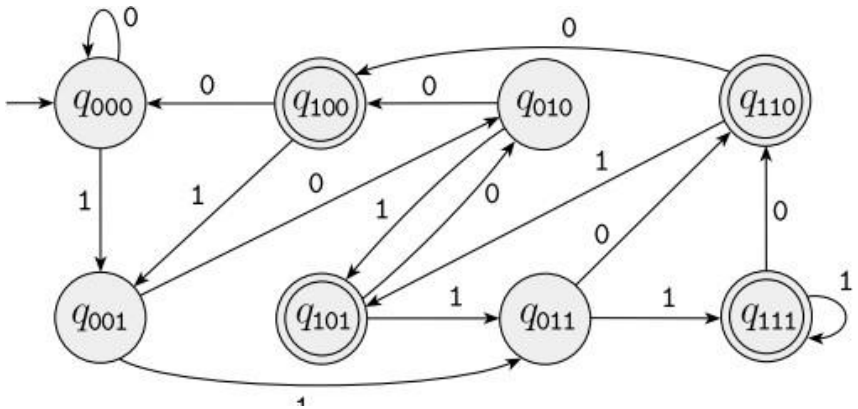
DFA vs NFA


Deterministic computation
start
accept or reject
Nondeterministic computation
reject
accept

Why NFA?

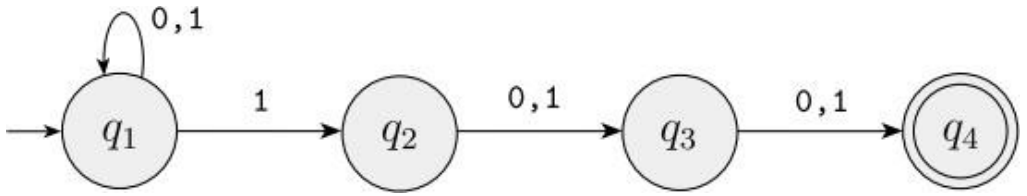
DFA :

1. Faster : It follows only one path.
2. Complex : More no of states and transitions.



NFA :

1. Slower : It chooses between many paths
2. Simple : Easy to express and join multiple machines.



Language that accepts all strings over {0,1} that contain a 1 either at the third position from the end

Formal Definition of NFA

DEFINITION 1.37

A ***nondeterministic finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta: Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

Comparison with DFA

DEFINITION 1.5

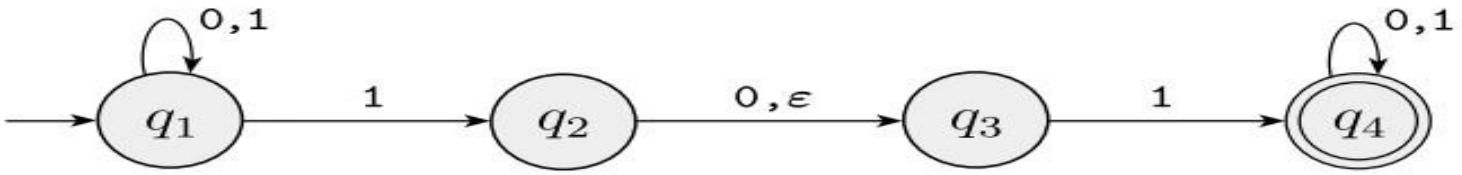
A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta: Q \times \Sigma \longrightarrow Q$ is the ***transition function***,[1]
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.[2]

DEFINITION 1.37

A ***nondeterministic finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta: Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.



The formal description of N_1 is $(Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3, q_4\}$,
2. $\Sigma = \{0,1\}$,
3. δ is given as

	0	1	ε
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$,

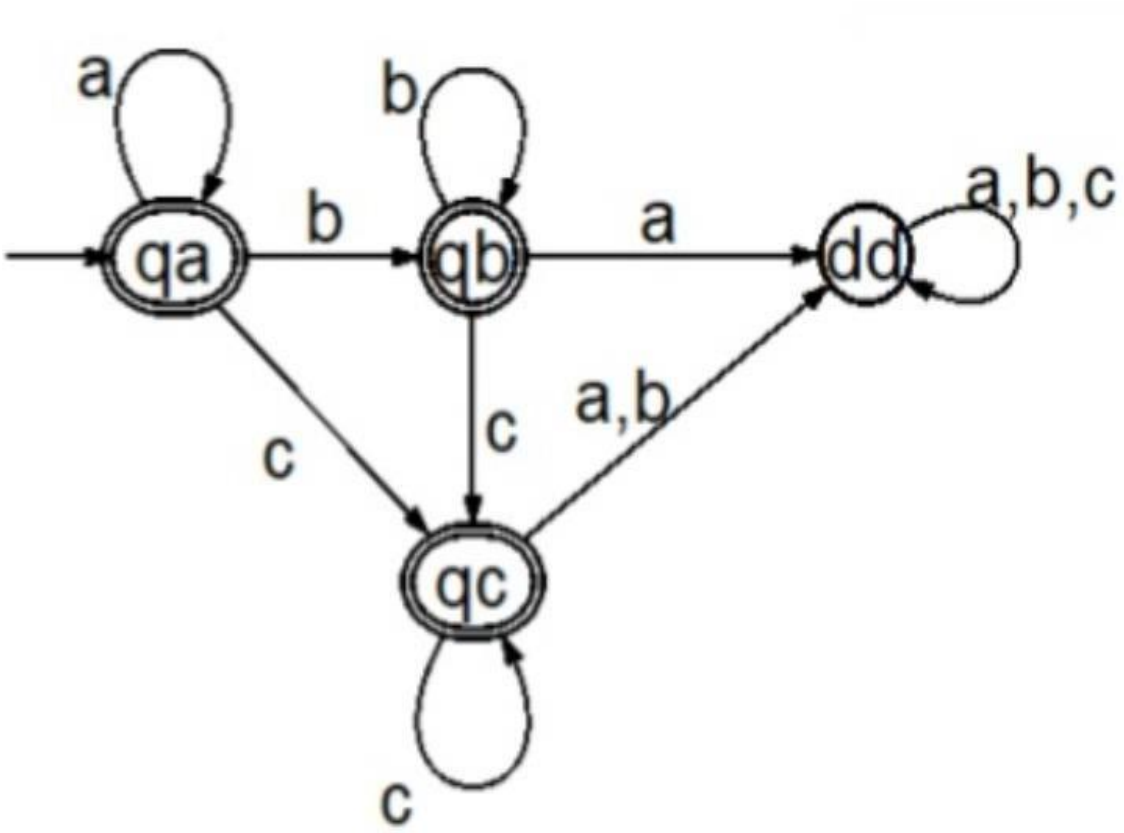
4. q_1 is the start state, and
5. $F = \{q_4\}$.

How does a DFA compute?

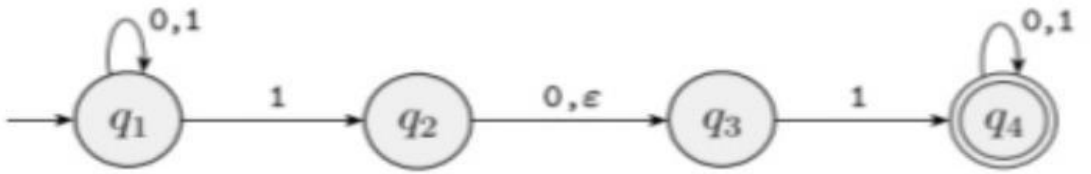
DFA will start from the start state, q_a

It will take input one symbols from the input string from left to right consecutively and will traverse to the next states accordingly.

After reaching the last state when no other input symbols left, if the last state is an accept state then this machine will accept that string otherwise it can't accept/reject that string.



How does an NFA compute?



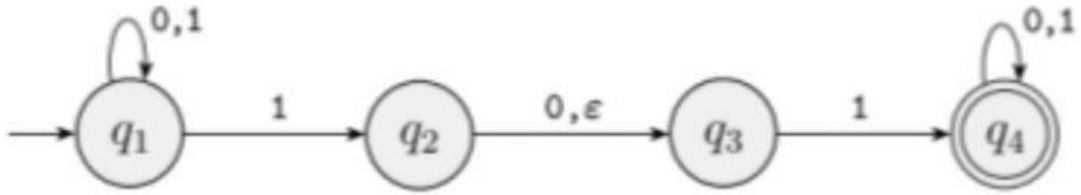
Suppose we are in state $q_1$in NFA and that the next input symbol is a 1. After reading that symbol, the machine splits into multiple copies of itself and follows all the possibilities in parallel.

Each copy of the machine takes one of the possible ways to proceed and continues as before. If there are subsequent choices, the machine splits again.

If the next input symbol doesn't appear on any of the arrows exiting the state occupied by a copy of the machine, that copy of the machine dies, along with the branch of computation associated with it.

Finally, if any one of these copies of the machine is in an accept state at the end of the input, the NFA accepts the input string.

How does an NFA compute?



If a state with an ε symbol on an exiting arrow is encountered, something similar happens. Without reading any input, the machine splits into multiple copies, one following each of the exiting ε –labeled arrows and one staying at the current state.

Then the machine proceeds nondeterministically as before.

NFA Simulation Example

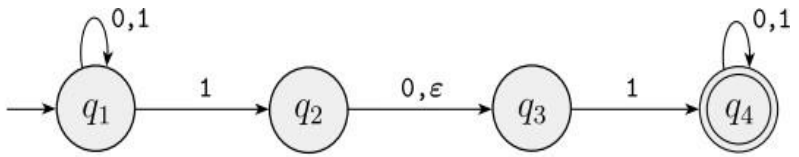


FIGURE 1.27
The nondeterministic finite automaton N_1

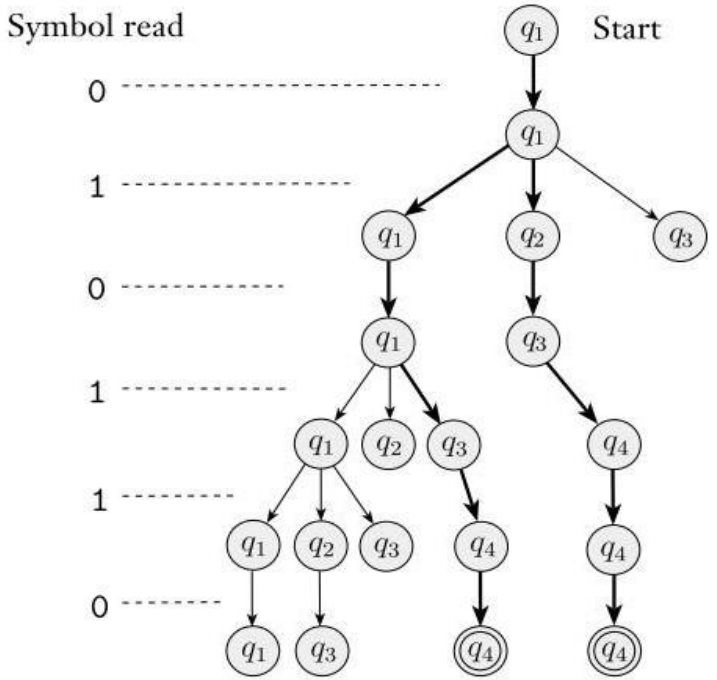


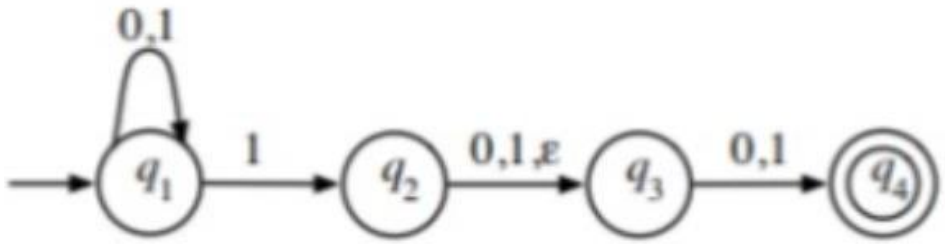
FIGURE 1.29
The computation of N_1 on input 010110

Computing NFA

Let’s compute the following NFA machine for input string 010110

Computing NFA

Let’s compute the following NFA machine for input string 010110

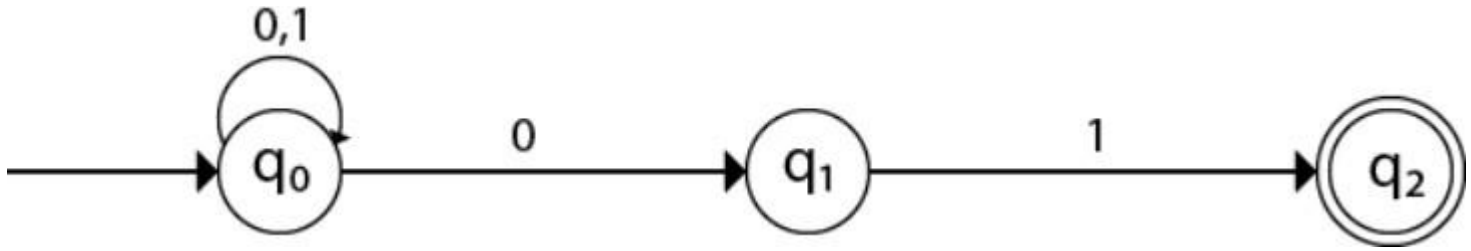


Example-2

Design an NFA with ∑ = {0, 1} accepts all string ending with 01.

Example-2

Design an NFA with ∑ = {0, 1} accepts all string ending with 01.

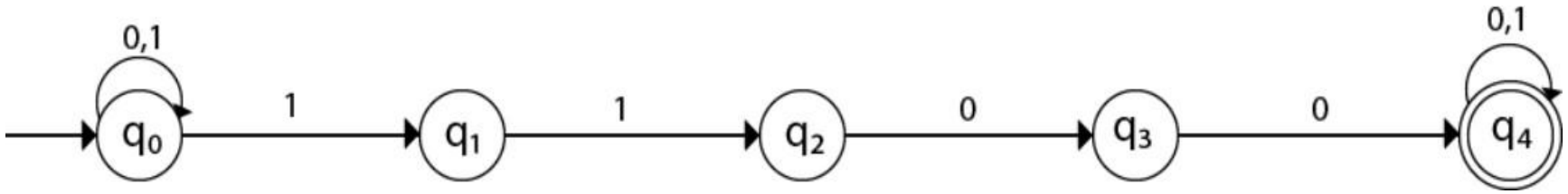


Example-3

Design an NFA with ∑ = {0, 1} in which double '1' is followed by double '0'.

Example-3

Design an NFA with ∑ = {0, 1} in which double '1' is followed by double '0'.

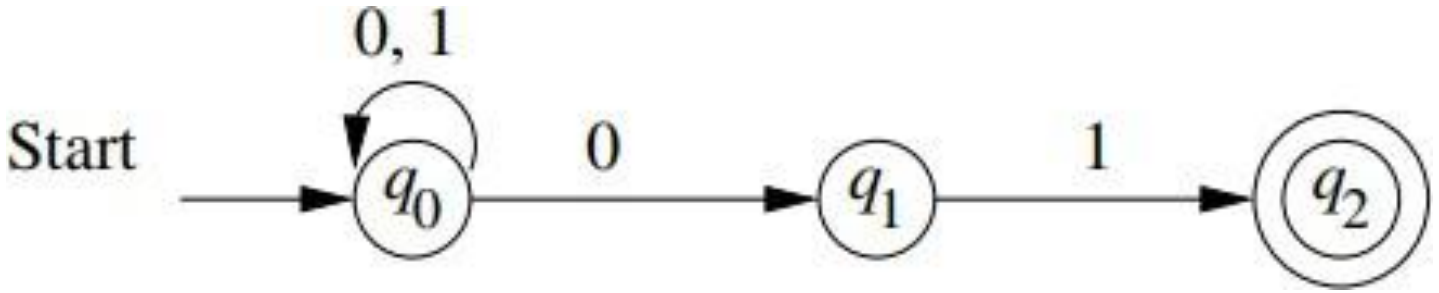


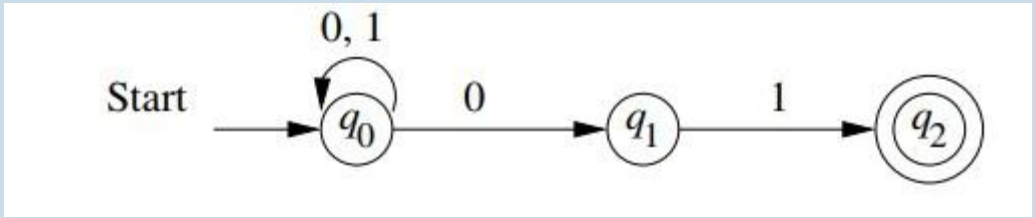
Example-4

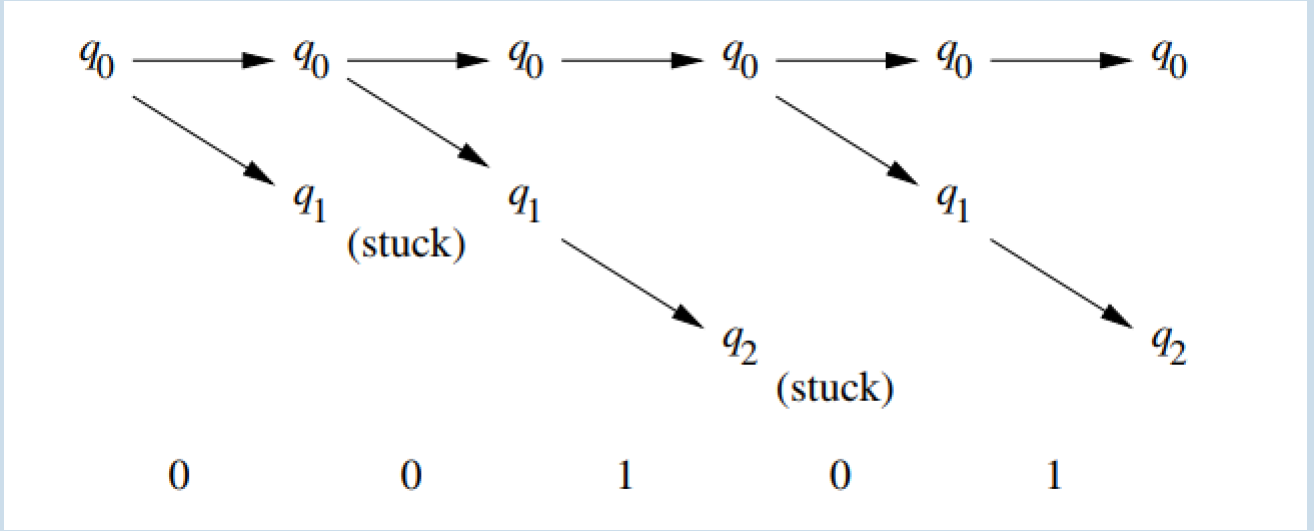
Job of this automaton is to accept all and only the strings of 0's and 1's that end in 01.

Example-4

Job of this automaton is to accept all and only the strings of 0's and 1's that end in 01.







Simulating 00101 in the automata

Example-5

Design a €-NFA that accepts all strings of the form 0^k where k is a multiple of 2 or

3. accepts the strings , 00, 000, 0000, and 000000, but
not 0 or 00000

Example-5

Design a €-NFA that accepts all strings of the form 0^k where *k* is a multiple of 2 or

3. accepts the strings , 00, 000, 0000, and 000000, but
not 0 or 00000

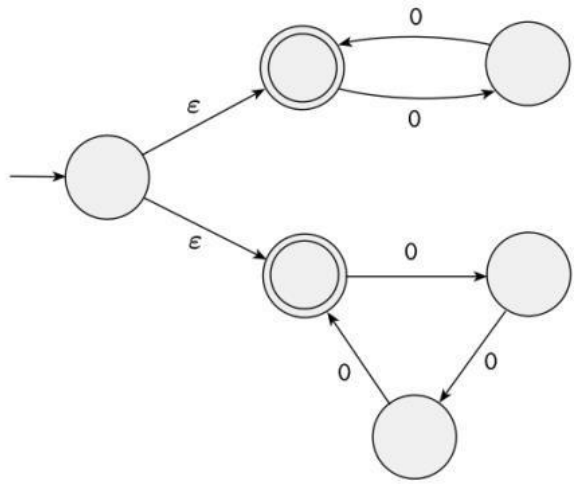


FIGURE **1.34**
The NFA N_3

Exercise

NFA – Design Draw the state diagram of NFA machines that can recognize the following languages:

L(M) = { w | w begins with 101 } over ∑={0,1}

L(M) = { w | w begins with abb } over ∑={a,b}

L(M) = { w | w ends with 101 } over ∑={0,1}

L(M) = { w | w ends with aa } over ∑={a,b}

L(M) = { w | w contains 110 as substring } over ∑={0,1}

L(M) = { w | w contains abb as substring } over ∑={a,b}

L(M) = { w | w is exactly 101 }

L(M) = { w | w contains a 1 in the 3rd position from the end } over ∑={0,1}

Regular Operations on NFA

UNION:

The class of regular languages is closed under the union operation.

That means, if A_1 and A_2 are regular languages, then A_1U A_2 is also a regular language. Here, A_1U A_2= { x | x ϵ A_1 or x ϵ A2 }

Example: A_1= { good, bad }

and A_2 = { boy, girl }

Then, A_1 U A_2= { good, bad, boy, girl }.

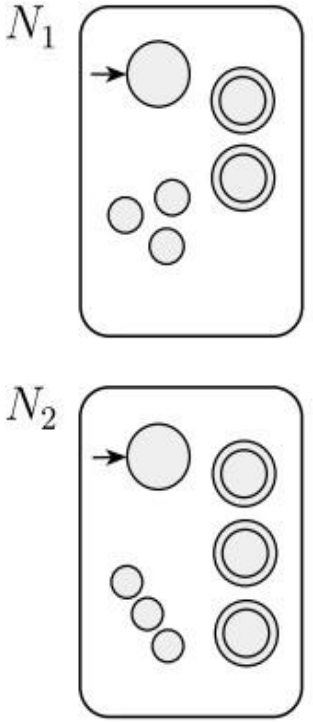
In this case, if N_1 and N_2 represent the NFAs to recognize A_1 and A_2 then we need to build a machine N from N_1 and N_2 so that N can also recognize A_1 U A_2.

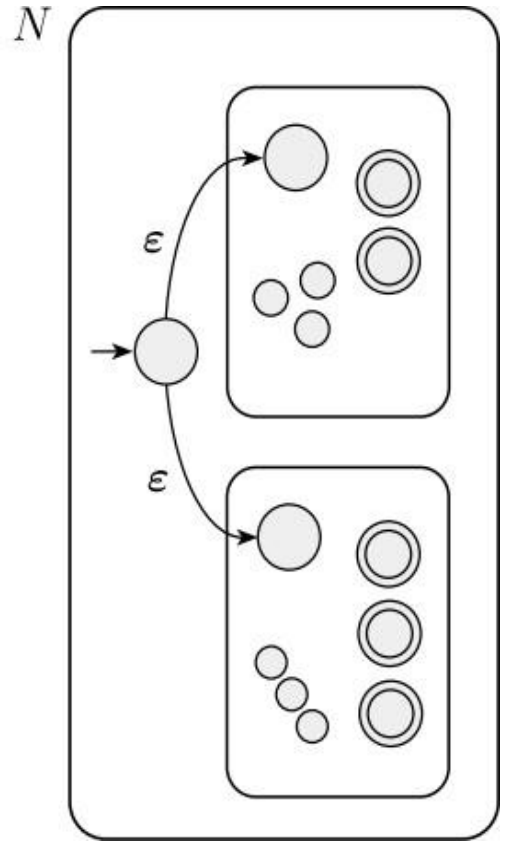
Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1, and
$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2.

Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \cup A_2$.

1. $Q = \{q_0\} \cup Q_1 \cup Q_2$.
   The states of N are all the states of N_1 and N_2, with the addition of a new start state q_0.
2. The state q_0 is the start state of N.
3. The set of accept states $F = F_1 \cup F_2$.
   The accept states of N are all the accept states of N_1 and N_2. That way, N accepts if either N_1 accepts or N_2 accepts.
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \\ \delta_2(q,a) & q \in Q_2 \\ \{q_1, q_2\} & q = q_0 \text{ and } a = \varepsilon \\ \emptyset & q = q_0 \text{ and } a \neq \varepsilon. \end{cases}$$





Regular Operations on NFA

Concatenation:

The class of regular languages is closed under the concatenation operation.

That means, if A_1 and A_2 are regular languages, then A_1o A_2 is also a regular language. Here, A_1o A_2= { xy | x ϵ A_1 and y ϵ A_2}

Example: A_1= { good, bad }

and A_2 = { boy, girl }

Then, A_1o A_2= { goodboy, goodgirl, badboy, badgirl }.

In this case, if N_1 and N_2 represent the NFAs to recognize A_1 and A_2 then we need to build a machine N from N_1 and N_2 so that N can also recognize A_1o A_2

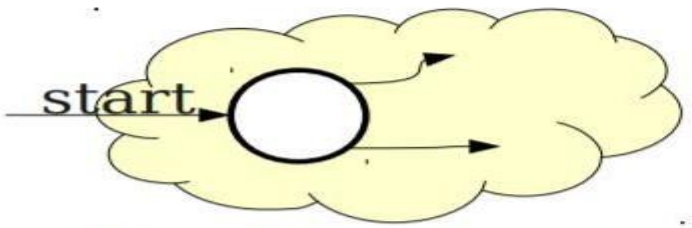
Concatenation Example

- Let Σ = { a, b, …, z, A, B, …, Z } and consider these languages over Σ:
  - ***Noun*** = { Puppy, Rainbow, Whale, … }
  - ***Verb*** = { Hugs, Juggles, Loves, … }
  - ***The*** = { The }
- The language ***TheNounVerbTheNoun*** is

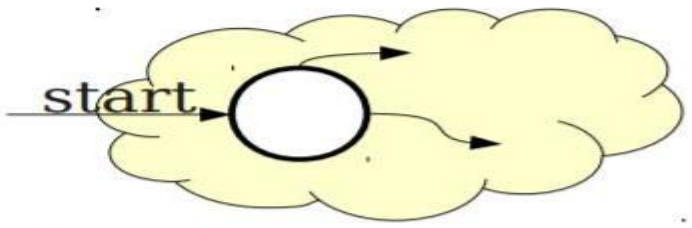
  { ThePuppyHugsTheWhale,
  TheWhaleLovesTheRainbow,
  TheRainbowJugglesTheRainbow, … }

Concatenating Regular Languages

- If L_1 and L_2 are regular languages, is L_1L_2?
- Intuition – can we split a string w into two strings xy such that $x \in L_1$ and $y \in L_2$?



Machine for L_1



Machine for L_2

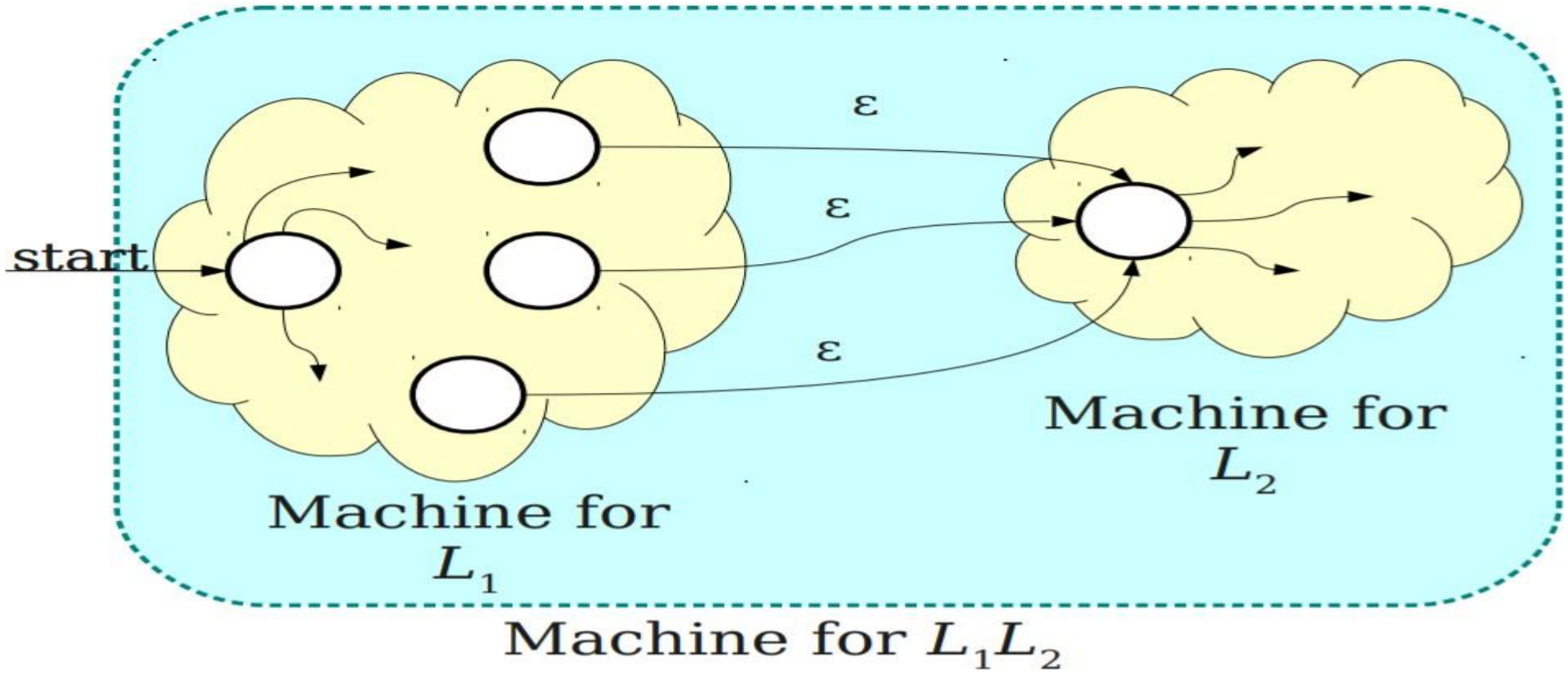
b	o	o	k

k	e	e	p	e	r

Concatenating Regular Languages

- If L_1 and L_2 are regular languages, is L_1L_2?
- Intuition – can we split a string w into two strings xy such that $x \in L_1$ and $y \in L_2$?
- **Idea**: Run the automaton for L_1 on w, and whenever L_1 reaches an accepting state, optionally hand the rest off w to L_2.
  - If L_2 accepts the remainder, then L_1 accepted the first part and the string is in L_1L_2.
  - If L_2 rejects the remainder, then the split was incorrect.

Concatenating Regular Languages



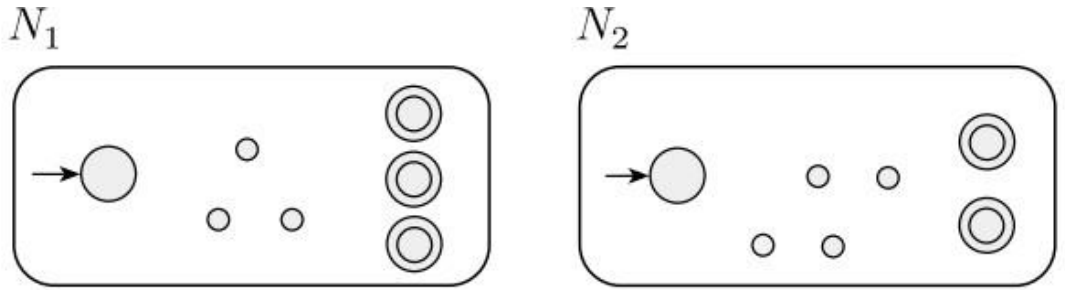
Concatenation

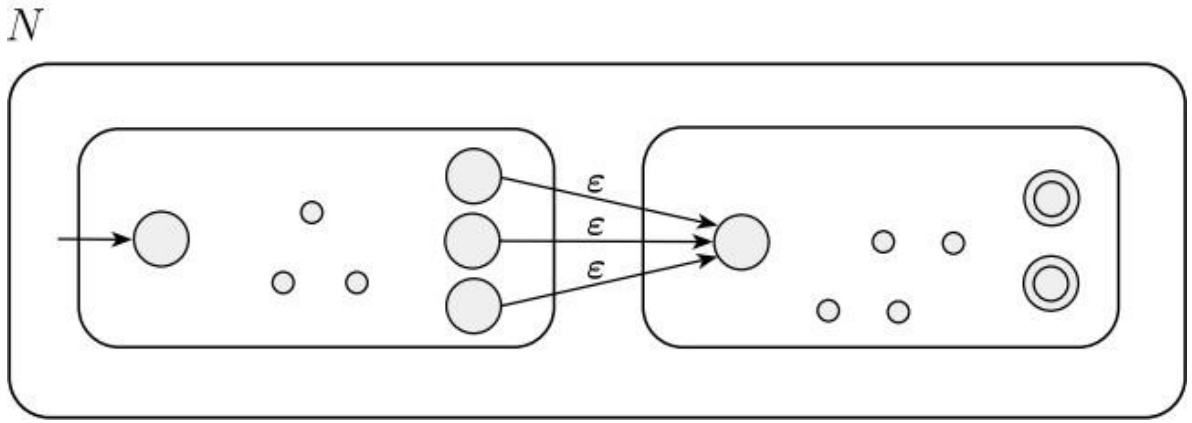
Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1, and
$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2.

Construct $N = (Q, \Sigma, \delta, q_1, F_2)$ to recognize $A_1 \circ A_2$.

1. $Q = Q_1 \cup Q_2$.
   The states of N are all the states of N_1 and N_2.
2. The state q_1 is the same as the start state of N_1.
3. The accept states F_2 are the same as the accept states of N_2.
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q,a) & q \in F_1 \text{ and } a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_2\} & q \in F_1 \text{ and } a = \varepsilon \\ \delta_2(q,a) & q \in Q_2. \end{cases}$$





Regular Operations on NFA

Star:

The class of regular languages is closed under the star operation.

That means, if A_1 is a regular language, then A_1* is also a regular language. Here, A_1* = $\{x_1x_2x_3 \dots \dots \dots x_k \mid$ k>=0 and each $x_i \in A_1 \}$

Example: A_1= { good, bad }

Then, A_1* = { ε , good, bad, goodgood, goodbad, badgood, badbad, goodgoodgood, goodgoodbad, goodbadgood, goodbadbad, … … } .

In this case, if N_1 represent the NFA to recognize A_1 then we need to build a machine N from N_1 so that N can also recognize A_1*

Kleene Star

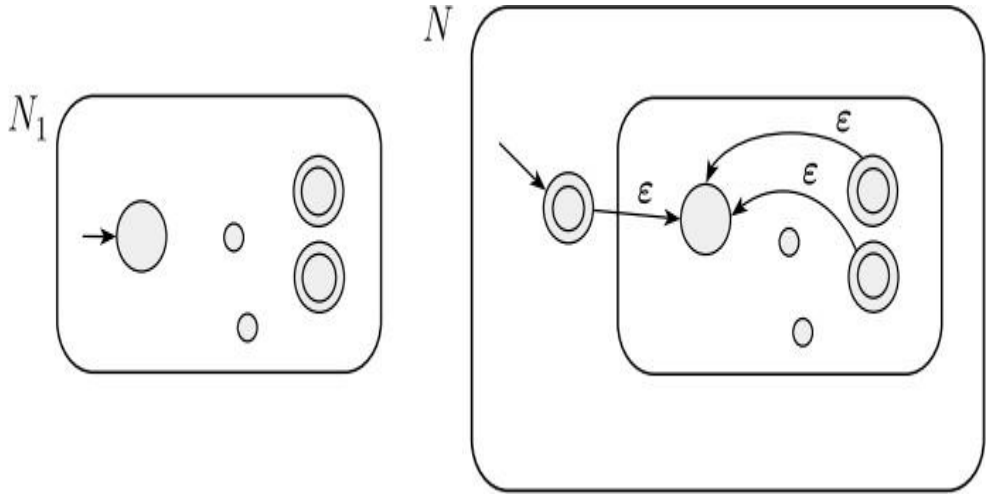
Construction of N to recognize A^*

PROOF Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1.
Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize A_1^*.

1. $Q = \{q_0\} \cup Q_1$.
   The states of N are the states of N_1 plus a new start state.
2. The state q_0 is the new start state.
3. $F = \{q_0\} \cup F_1$.
   The accept states are the old accept states plus the new start state.

4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q,a) & q \in F_1 \text{ and } a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_1\} & q \in F_1 \text{ and } a = \varepsilon \\ \{q_1\} & q = q_0 \text{ and } a = \varepsilon \\ \emptyset & q = q_0 \text{ and } a \neq \varepsilon. \end{cases}$$



NFA Design

Design Draw the state diagram of NFA machines that can recognize the following languages:

- Union: A be the language consisting of all strings of the form 0^k over {0} where k is a multiple of 2 or 3
- Union: All strings beginning with 101 or with 110
- Concatenation: All strings beginning with 101 and ending with 101
- Star: All strings consisting of 0 or more repetitions of 101
- Plus: All strings consisting of 1 or more repetitions of 101
- Complement: All strings that doesn't contain substring 101
- Concat + Complement: All strings with exactly 1 occurrence of 101

Equivalence of NFA and DFA

Deterministic and nondeterministic finite automata recognize the same class of languages.

Such equivalence is both surprising and useful.

It is surprising because NFAs appear to have more power than DFAs, so we might expect that NFAs recognize more languages.

It is useful because describing an NFA for a given language sometimes is much easier than describing a DFA for that language.

Say that two machines are equivalent if they recognize the same language.

Equivalence of NFA and DFA

THEOREM 1.39

Every nondeterministic finite automaton has an equivalent deterministic finite automaton.

Equivalence of NFA and DFA

If a language is recognized by an NFA, then we must show the existence of a DFA that also recognizes it.

The idea is to convert the NFA into an equivalent DFA that simulates the NFA.

Recall the “reader as automaton” strategy for designing finite automata.

Equivalence of NFA and DFA

- How would you simulate the NFA if you were pretending to be a DFA?
- What do you need to keep track of as the input string is processed?
- In the examples of NFA's, you kept track of the various branches of the computation by placing a finger on each state that could be active at given points in the input.
- You updated the simulation by moving, adding, and removing fingers according to the way the NFA operates. All you needed to keep track of was the set of states having fingers on them.

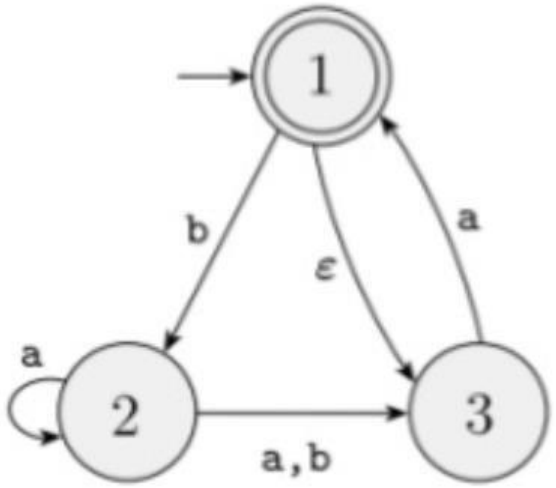
Equivalence of NFA and DFA

- If *k* is the number of states of the NFA, it has 2^k subsets of states.
- Each subset corresponds to one of the possibilities that the DFA must remember, so the DFA simulating the NFA will have 2^k states.
- Now we need to figure out which will be the start state and accept states of the DFA.
- What will be its transition function?
- We can discuss this more easily after setting up some formal notation.

Converting NFA into Equivalent DFA

Let N = (Q, Σ , δ, q_0, F) be the NFA recognizing some language A.

We need to construct a DFA machine M = (Q', Σ , δ', q_0', F') recognizing A.

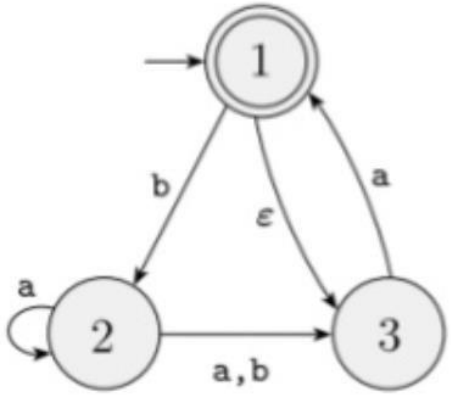


$Q' = P(Q)$

Every state of *M* is a set of states of *N*.

Recall that *P*(*Q*) is the set of subsets of *Q*.

Q' = P(Q) = set of all subsets of Q = { Ø , {1}, {2}, {3}, {1,2}, {1,3}, {2,3}, {1,2,3} }



ε - closure

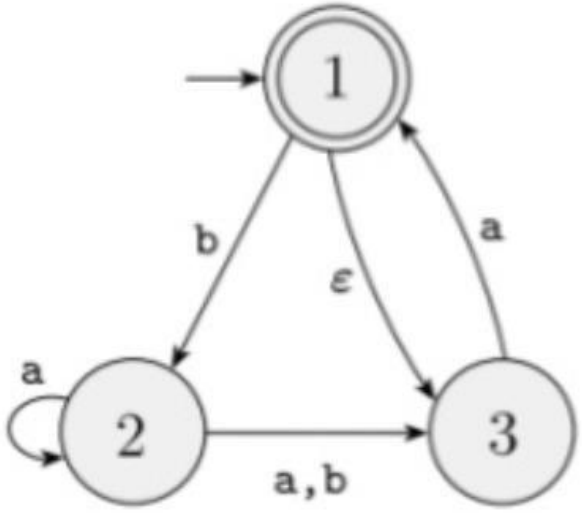
To discuss the further steps, we need to consider the ε arrows.

For any state *R* of *M*, we define *E*(*R*) to be the collection of states that can be reached from members of *R* by going only along ε arrows, including the members of *R* themselves.

For R $\subseteq$ Q, ε - closure of R = E(R) = { q | q can be reached from members of R by traveling along 0 or more ε arrows }

Start State, q_0'

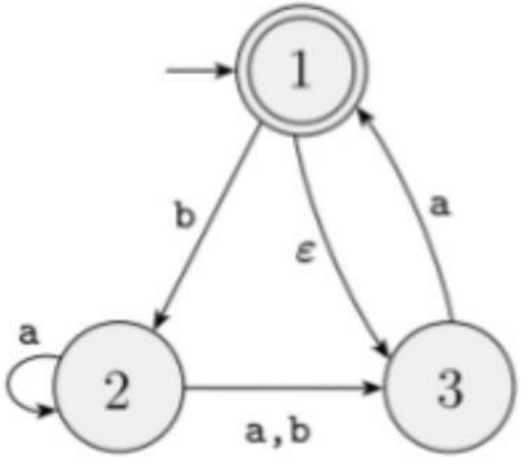
q_0' = E (q0) = E({1}) = ε - closure of {1} = { 1, 3 } is our start state.



Final State, F’

F’ = { R ∈ Q’ | R contains an accept state of N } i.e. the machine M accepts if one of the possible states that N could be in at this point is an accept state.

For this case, F’ = { {1}, {1,2}, {1,3}, {1,2,3} }

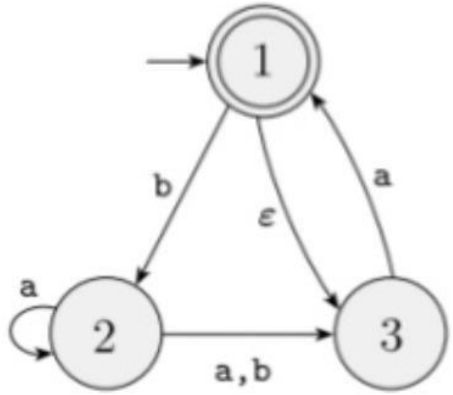


Transition Function, δ'(R, a)

For R $\subseteq$ Q, ε - closure of R = E(R) = { q | q can be reached from members of R by traveling along 0 or more ε arrows }

For input symbol a, the transition function can be defined as,

$\delta'(R; a) = \{q \in Q \mid q \in E(\delta(R, a)) \text{ for some } r \in R\} = \cup_{r \in R} \delta(R, a)$.



In our case for example,

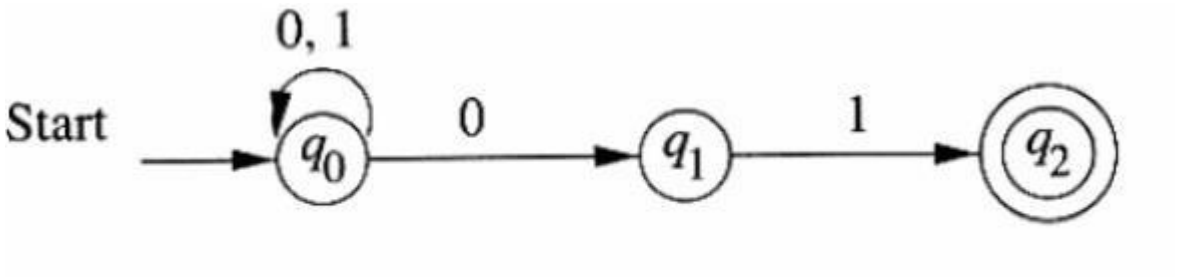
δ'(R, a) = δ'({ 2, 3 } , a) = E(δ(2 , a)) U E(δ(3 , a)) = E({2, 3}) U E({1}) = {2, 3} U { 1, 3 } = { 1, 2, 3 }

δ'(R, a) = δ'({ 1, 2 } , a) = E(δ(1 , a)) U E(δ(2 , a)) = E({ }) U E({2, 3}) = { } U { 2, 3 } = { 2, 3 }

δ'(R, a) = δ'({ 3 } , a) = E(δ(3 , a)) = E({ 1 }) = {1, 3}

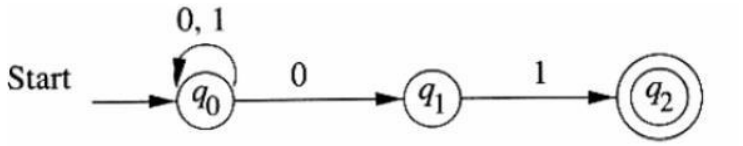
δ'(R, a) = δ'({ 3 } , b) = E(δ(3 , b)) = E({ }) = { }

Example



An NFA accepting all strings that end in 01

Example - Continued

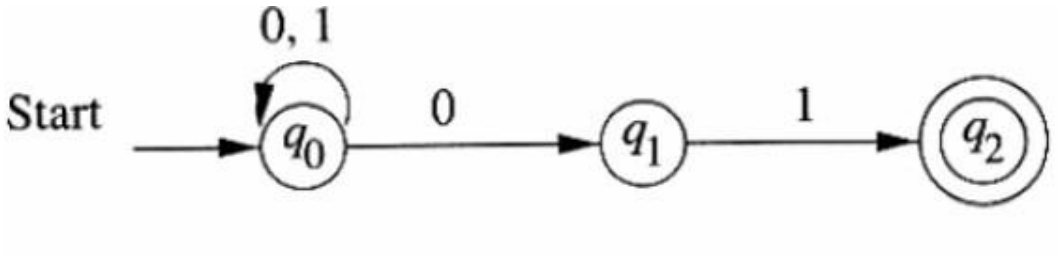


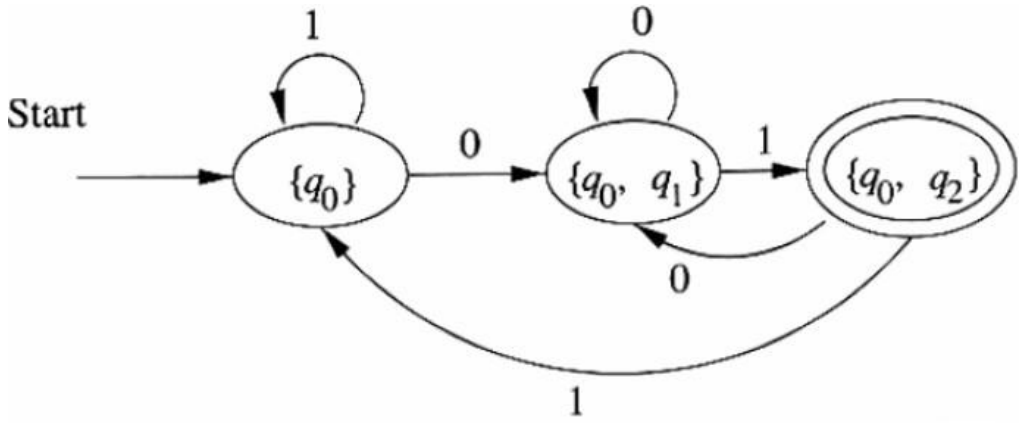
An NFA accepting all strings that end in 01

	0	1
$\emptyset$	$\emptyset$	$\emptyset$
$\rightarrow \{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1\}$	$\emptyset$	$\{q_2\}$
$*\{q_2\}$	$\emptyset$	$\emptyset$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$*\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
$*\{q_1, q_2\}$	$\emptyset$	$\{q_2\}$
$*\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

Complete subset construction and transition table

Example - Continued





The DFA constructed from the NFA

Another Example

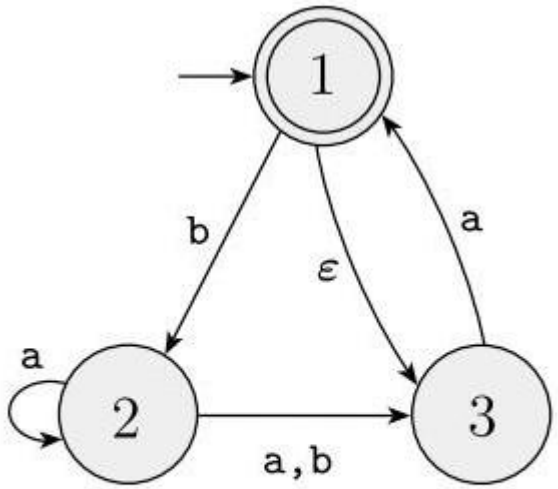


FIGURE 1.42
The NFA N_4

Solved?

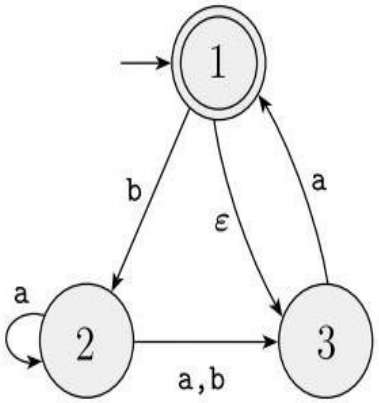


FIGURE **1.42**
The NFA N_4

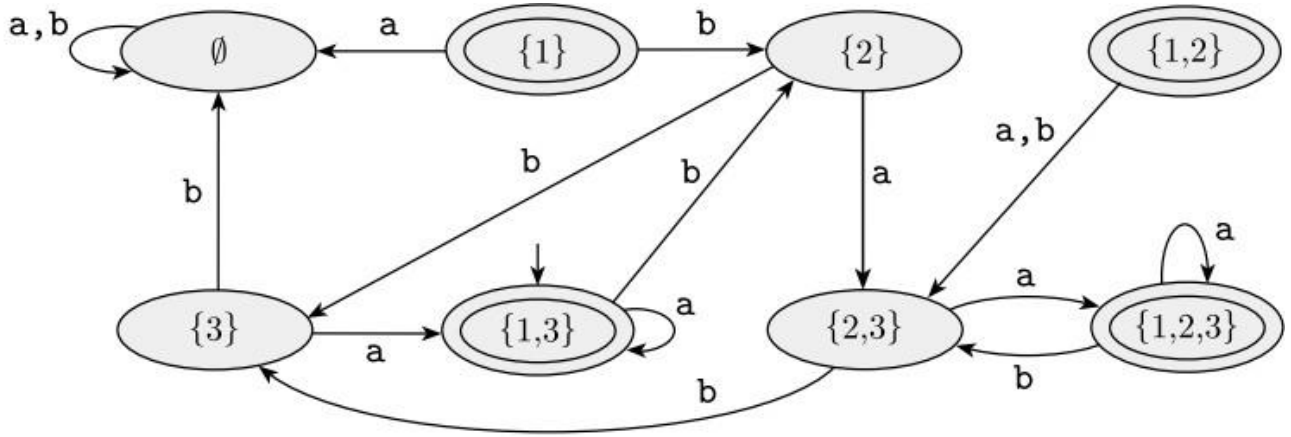


FIGURE **1.43**
A DFA D that is equivalent to the NFA N_4

Removing Unnecessary States

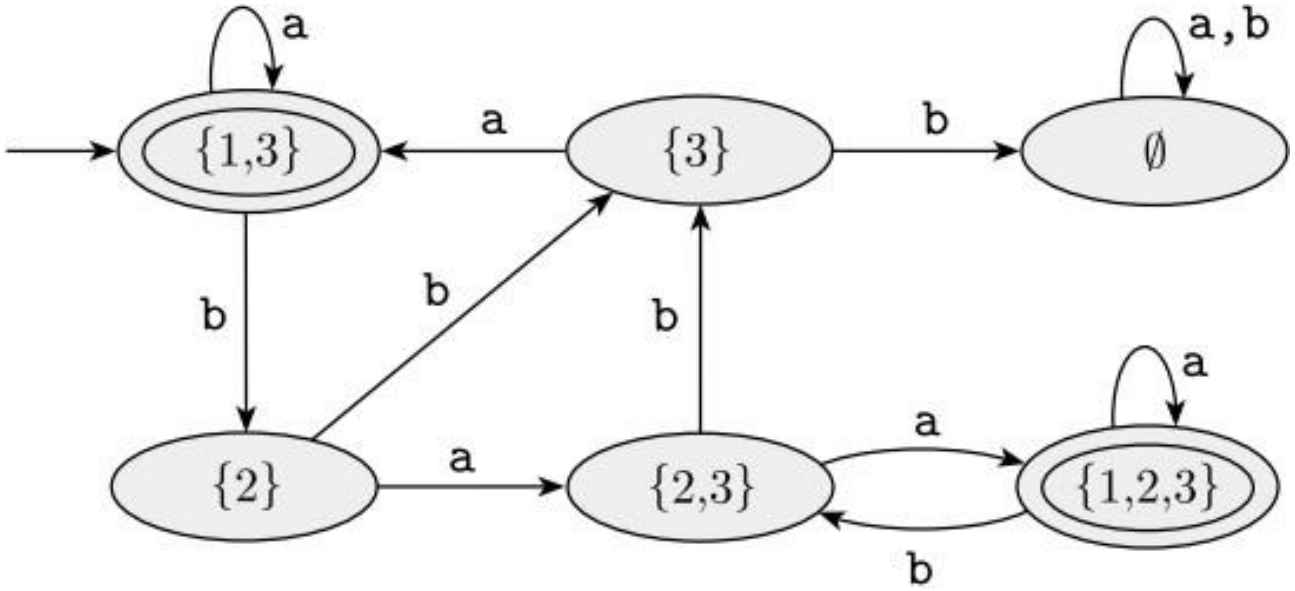


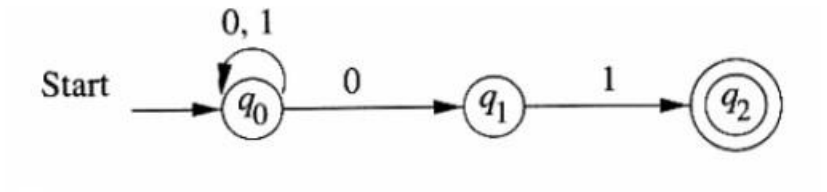
FIGURE 1.44
DFA D after removing unnecessary states

The Extended Transition Function

As for DFA's, we need to extend the transition function δ of an NFA to a function δˆ that

- takes a state q and a string of input symbols w,
- and returns the set of states that the NFA is in if it starts in state q and processes the string w.

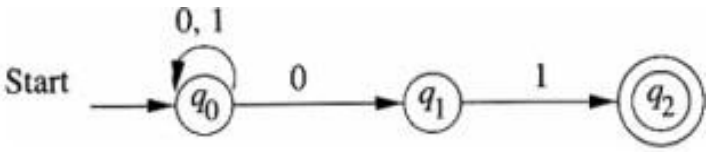
Let us use $\hat{\delta}$ to describe the processing of input 00101 by the NFA of figure.



An NFA accepting all strings that end in 01

The Extended Transition Function - continued

00101



A summary of the steps is:

1. $\hat{\delta}(q_0, \epsilon) = \{q_0\}$.
2. $\hat{\delta}(q_0, 0) = \delta(q_0, 0) = \{q_0, q_1\}$.
3. $\hat{\delta}(q_0, 00) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$.
4. $\hat{\delta}(q_0, 001) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$.
5. $\hat{\delta}(q_0, 0010) = \delta(q_0, 0) \cup \delta(q_2, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$.
6. $\hat{\delta}(q_0, 00101) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$.