COMPILER DESIGN

UNIT 1

PART 3

Pragya gaur

GRAMMAR

- the formal definition of the syntax of a programming language is usually called ‘Grammar’
- A Grammar consist of a set of rules to specify the sequence of characters that form allowable program in the language being defined.

- Formal Grammar : A Grammar specified using a strictly defined notations
- Equivalent Grammar : two grammar are equivalent if they produce the same language.

```
   S -> bS
S -> Sb
S -> a
```

```
S -> bS
S -> aT
T -> bT
T -> ε
```

- Regular Grammar: Regular grammar are special cases of BNF grammar that turn out to be equivalent to the finite state automata.
- Regular Expression: a form of language definition that is equivalent to FSA and regular grammar.
  - a special text string for describing a search pattern.

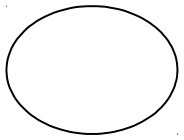
FINITE AUTOMATA

- Regular expressions = specification
- Finite automata = implementation

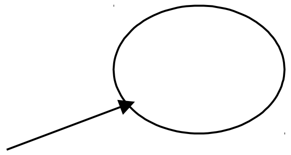
- A finite automaton consists of
  - An input alphabet Σ
  - A set of states S
  - A start state n
  - A set of accepting states F ⊆ S
  - A set of transitions state $\rightarrow^{\text{input}}$ state

FINITE AUTOMATA STATE GRAPHS

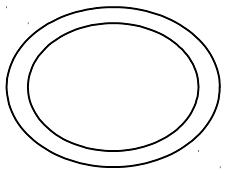
- A state



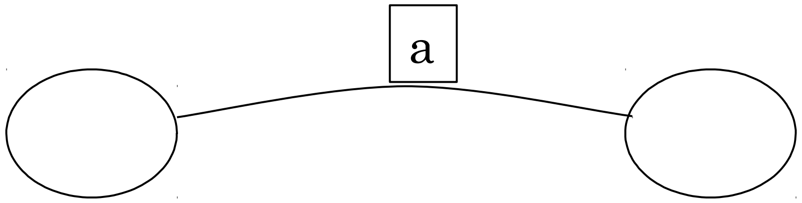
- The start state



- An accepting state

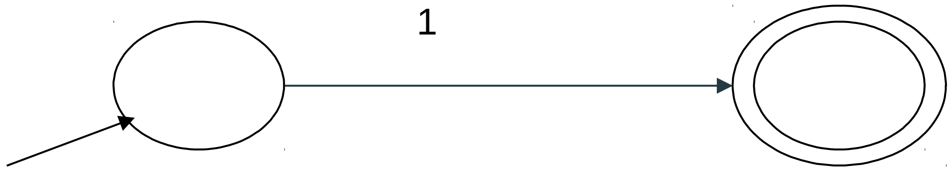


- A transition



A SIMPLE EXAMPLE

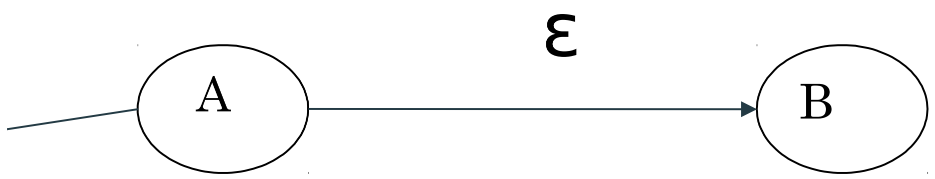
- A finite automaton that accepts only “1”



- A finite automaton accepts a string if we can follo
- w transitions labeled with the characters in the string from the start to some accepting state

EPSILON MOVES

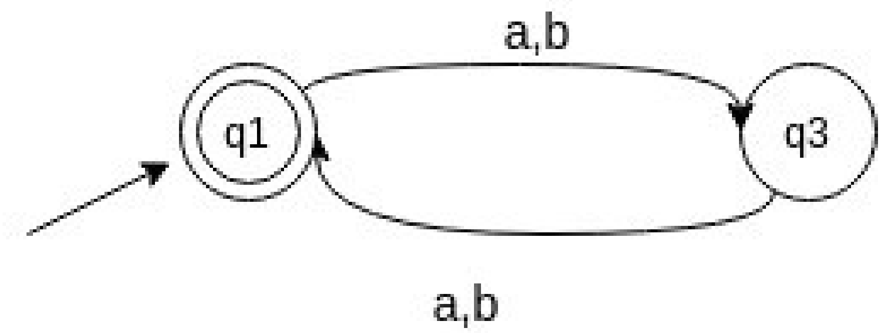
- Another kind of transition: ε-moves



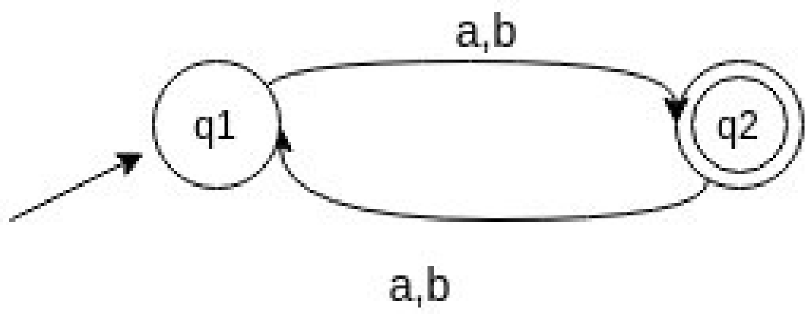
- Machine can move from state A to state B without reading input

A SIMPLE EXAMPLE

DFA for W ∈ a,b
|w| mod 2=0



DFA for W ∈ a,b
|w| mod 2=1



DFA for W ∈ a,b
|w| mod 3=0

DETERMINISTIC AND NONDETERMINISTIC AUTOMATA

- Deterministic Finite Automata (DFA)
  - One transition per input per state
  - No ε-moves
- Nondeterministic Finite Automata (NFA)
  - Can have multiple transitions for one input in a given state
  - Can have ε-moves

ARDEN'S THEOREM STATE THAT:

If P and Q are two regular expressions over Σ, and if P does not contain ϵ , then the

following equation in R given by **R = Q + RP**

has an unique solution i.e.,

R = QP*

Let's start by taking this equation as equation (i)

R = Q + RP ......(i)

Now, replacing R by R = QP*, we get,

R = Q + QP*P

Taking Q as common,

R = Q(ϵ + P*P)
R = QP*

EXAMPLES

Initial and final state-q1

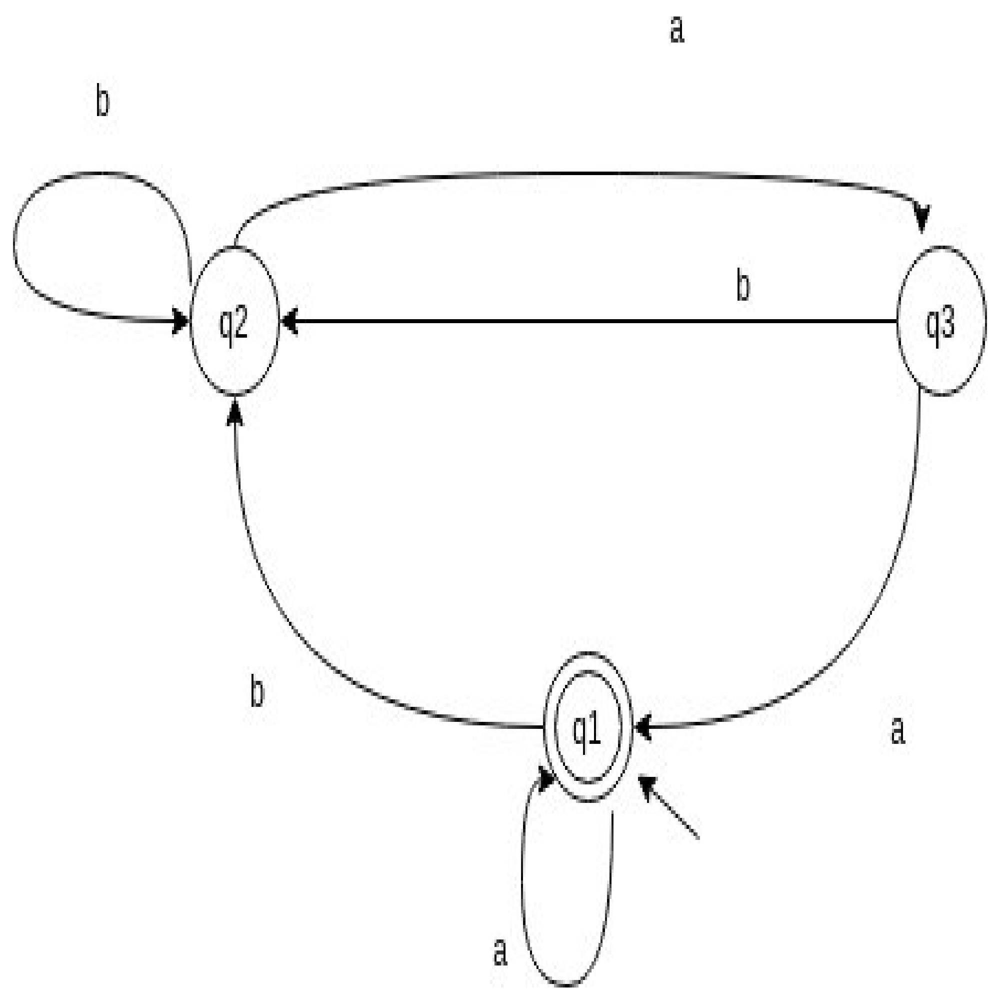
Step-01:
equations are
q1=q1 a + q3 a + ϵ (1
q2= q1 b + q2 b + q3 b (2
q3= q2 a (3

Bring state q2 in the form R = Q + RP.

q2= q1 b + q2 b + (q2 a) b
= q1 b + q2(b + a b)

by arden’s theorem

q2=q1 b (b+ab)* (4



EXAMPLES

Using (3) in (1), we get-

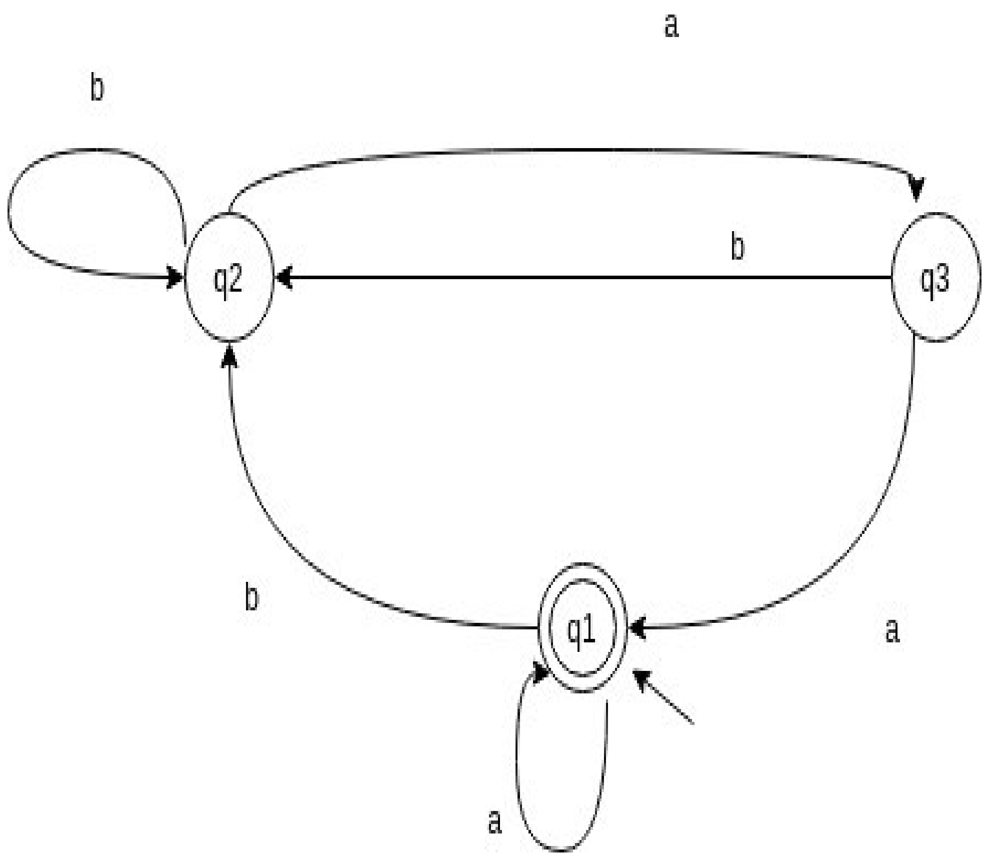
q1=q1 a + q3 a + ϵ
=q1a+(q2 a)a + ϵ

Using (4) in (1), we get-

=q1 a+q1 b (b+ab)* a a + ϵ
=q1 (a+b (b+ab)* a a) + ϵ (5

Using Arden’s Theorem in (5), we get-

q1= ϵ + q1 (a+b (b+ab)* a a)
q1=(a+b(b+ab)*aa)*



Thus, Regular Expression for the given DFA = (a+b(b+ab)*aa)*

ARDEN'S THEOREM

Construct the regular expression for the given DFA

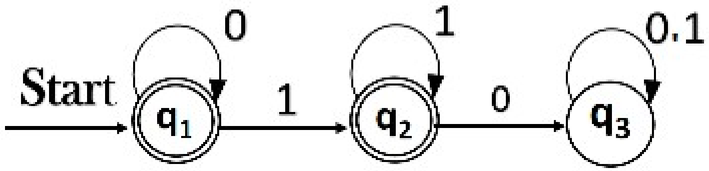
Solution:

Let us write down the equations

q1 = q1 0 + ε

Since q1 is the start state, so ε will be added, and the input 0 is coming to q1 from q1 hence we write

State = source state of input × input coming to it



Similarly,

q2 = q1 1 + q2 1
q3 = q2 0 + q3 (0+1)

Since the final states are q1 and q2, we are interested in solving q1 and q2 only. Let us see q1 first

$q1 = q1\ 0 + \varepsilon$

We can re-write it as

$q1 = \varepsilon + q1\ 0$
$R \quad = Q + \quad R \ \ P$

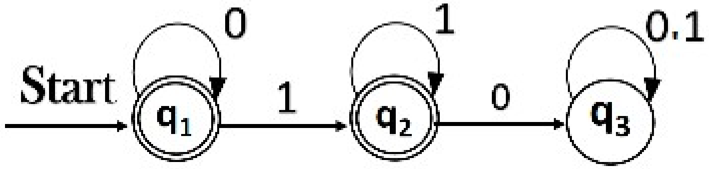
Which is similar to $R = Q + RP$,

and gets reduced to $R = OP^*$.

Assuming $R = q1$, $Q = \varepsilon$, $P = 0$

We get

$q1 = \varepsilon.(0)^*$
$q1 = 0^*$



$q1 = 0^*$

Substituting the value into q2, we will get

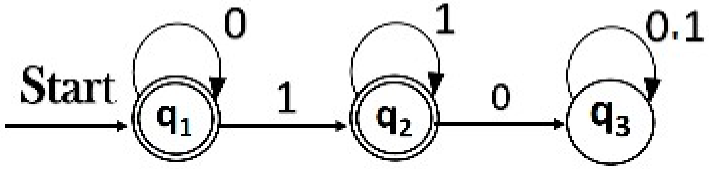
$q2 = q1\ 1 + q2\ 1$
$q2 = 0^*\ 1 + q2\ 1$

$R \quad = \ Q \quad + \ R \ \ P$

$q2 = 0^*\ 1\ (1)^* \quad (R = Q + RP \ \rightarrow \ Q P^*)$

The regular expression is given by

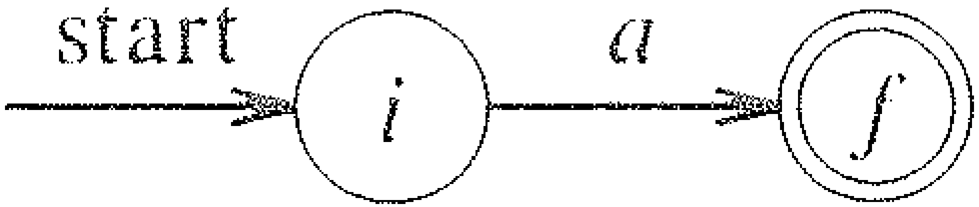
$r = q1 + q2$
$= 0^* + 0^*\ 1(1)^*$
$r = 0^* + 0^*\ 1\ 1^*$



THOMPSON CONSTRUCTION

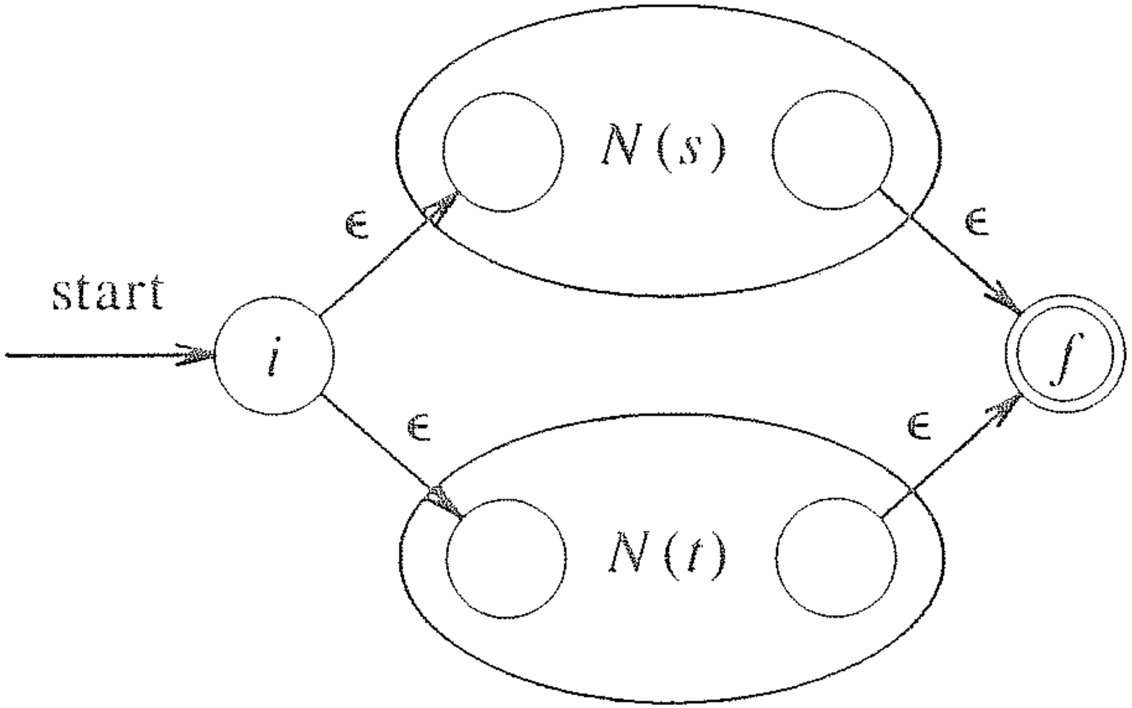
Construction of an NFA from a Regular Expression

BASIS: For expression a construct the NFA



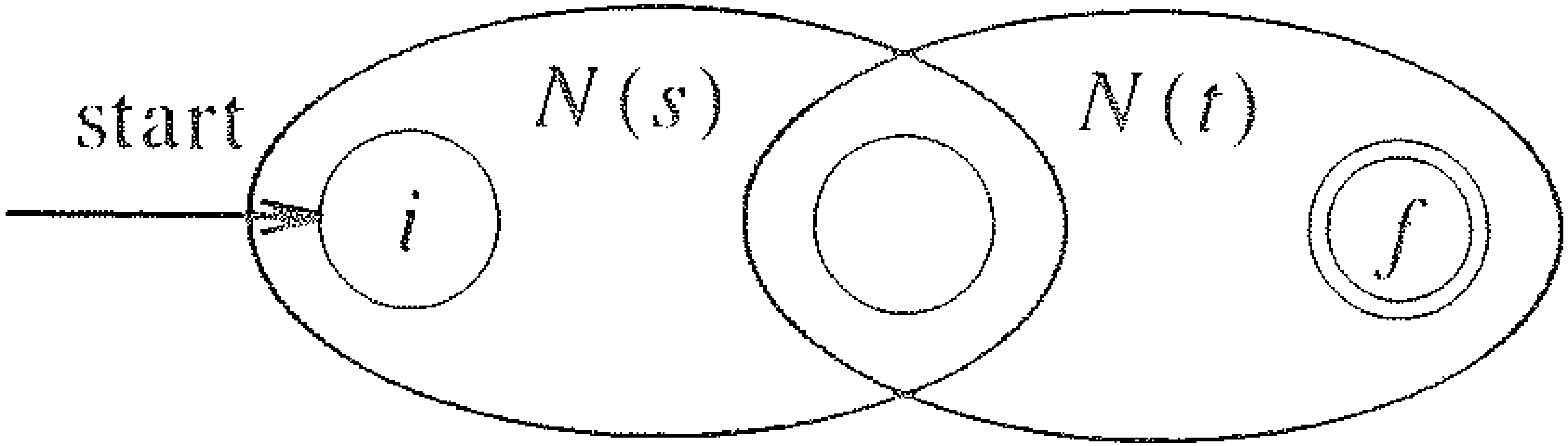
THOMPSON CONSTRUCTION

For the regular expression s|t,



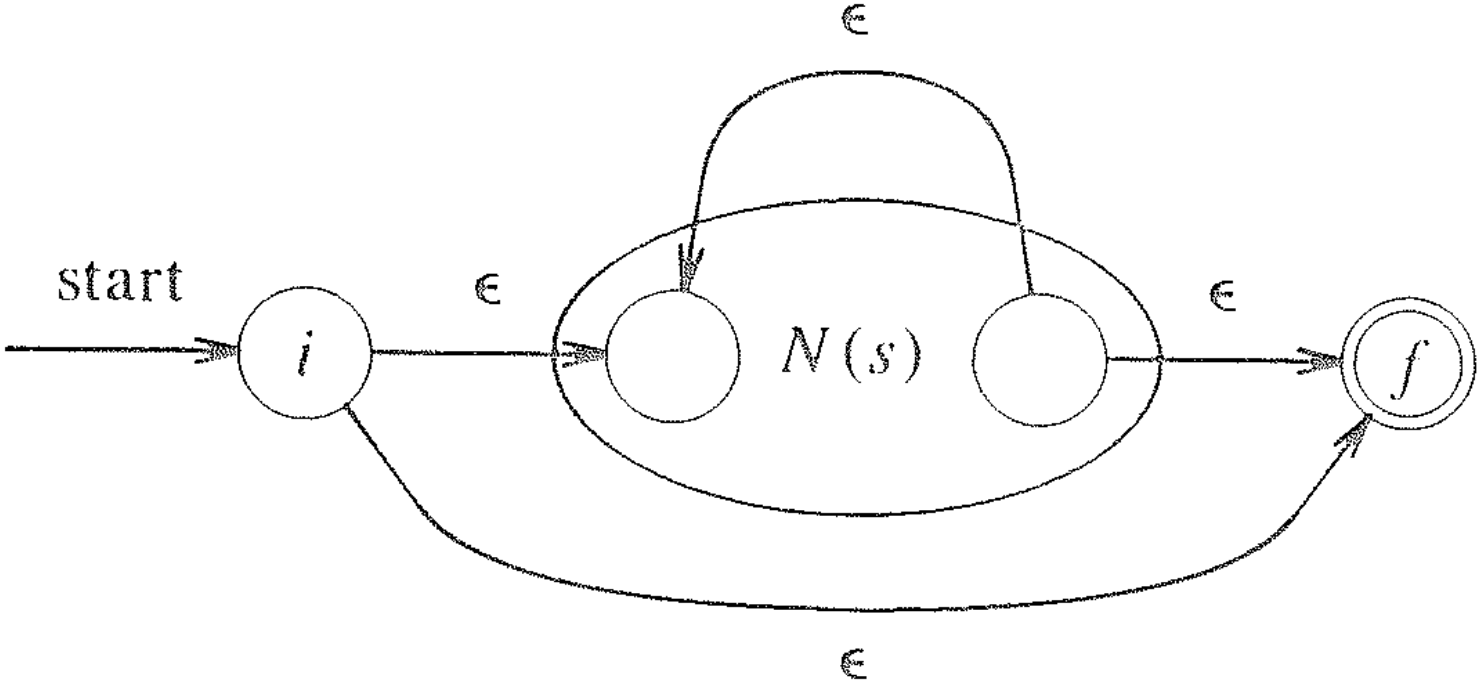
THOMPSON CONSTRUCTION

For the regular expression st,



THOMPSON CONSTRUCTION

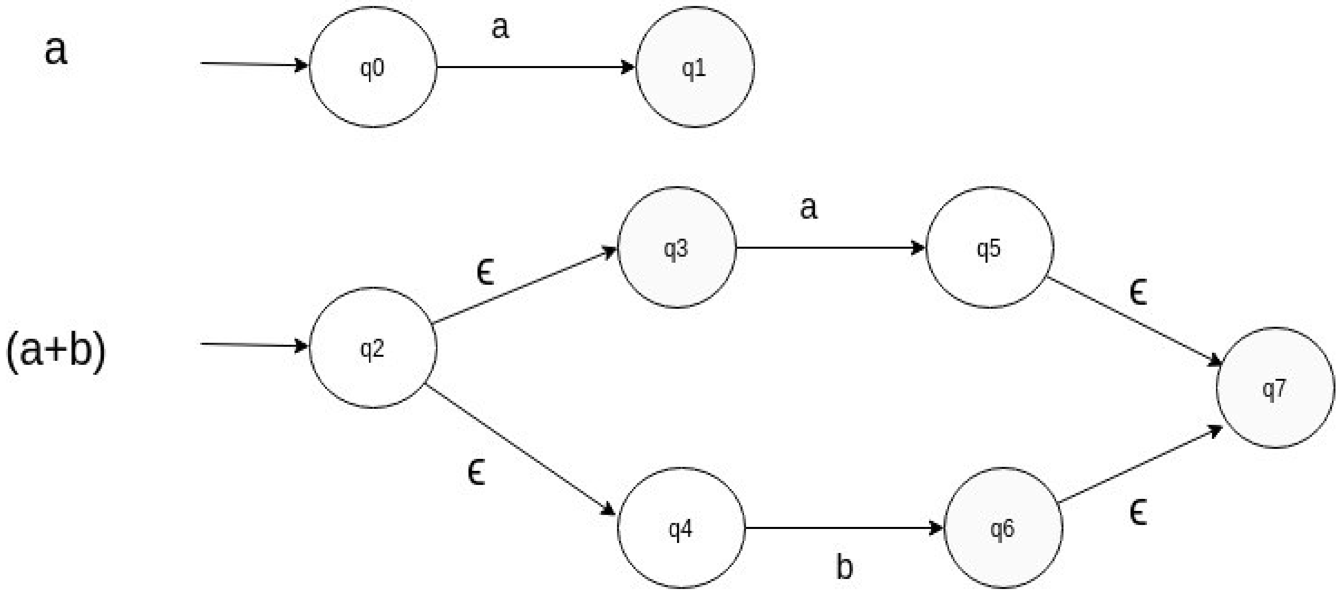
For the regular expression S*,



EXAMPLE

construct an NFA for r= a(a+b)*bb

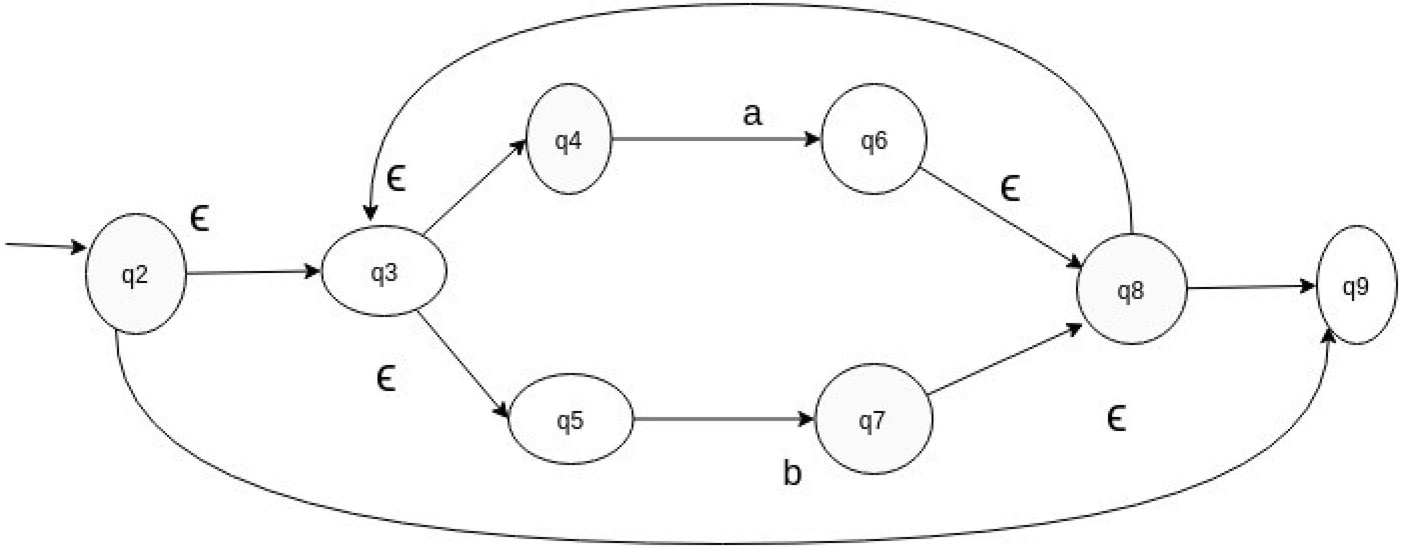
r1 = a,
r2=(a+b)
r3=(a+b)*
r4=bb



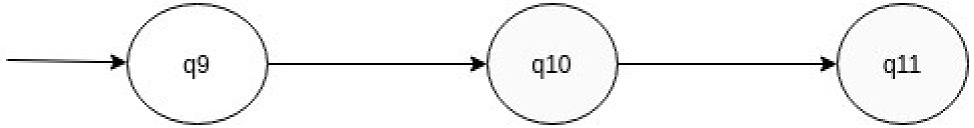
EXAMPLF

a(a+b)*bb

(a+b)*

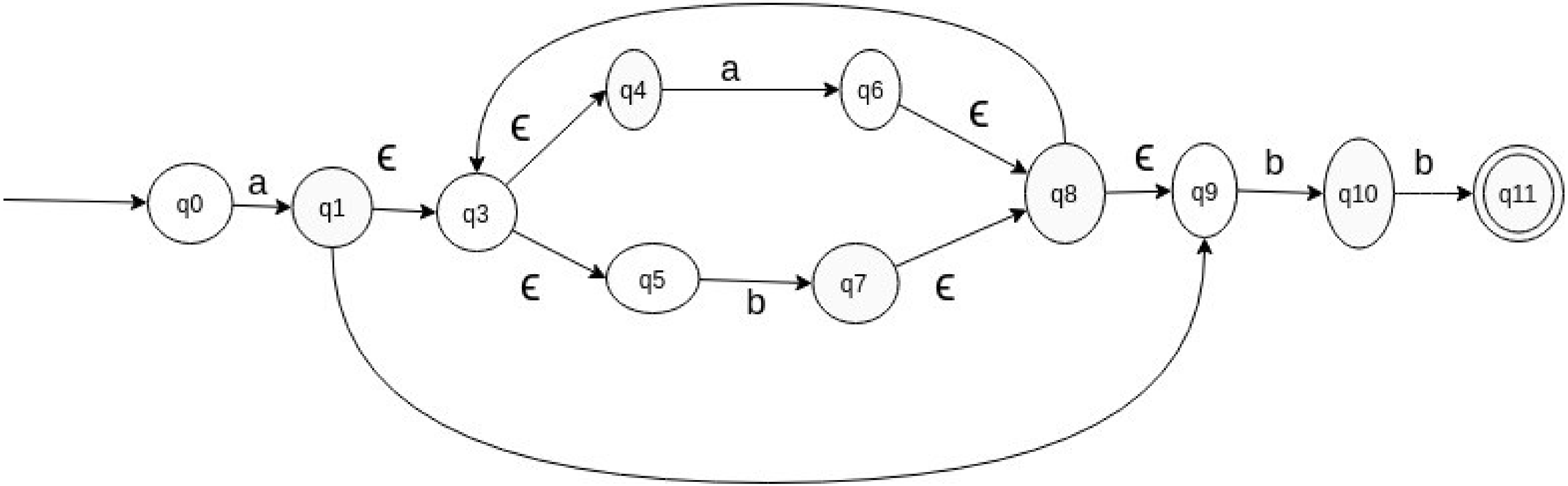


bb

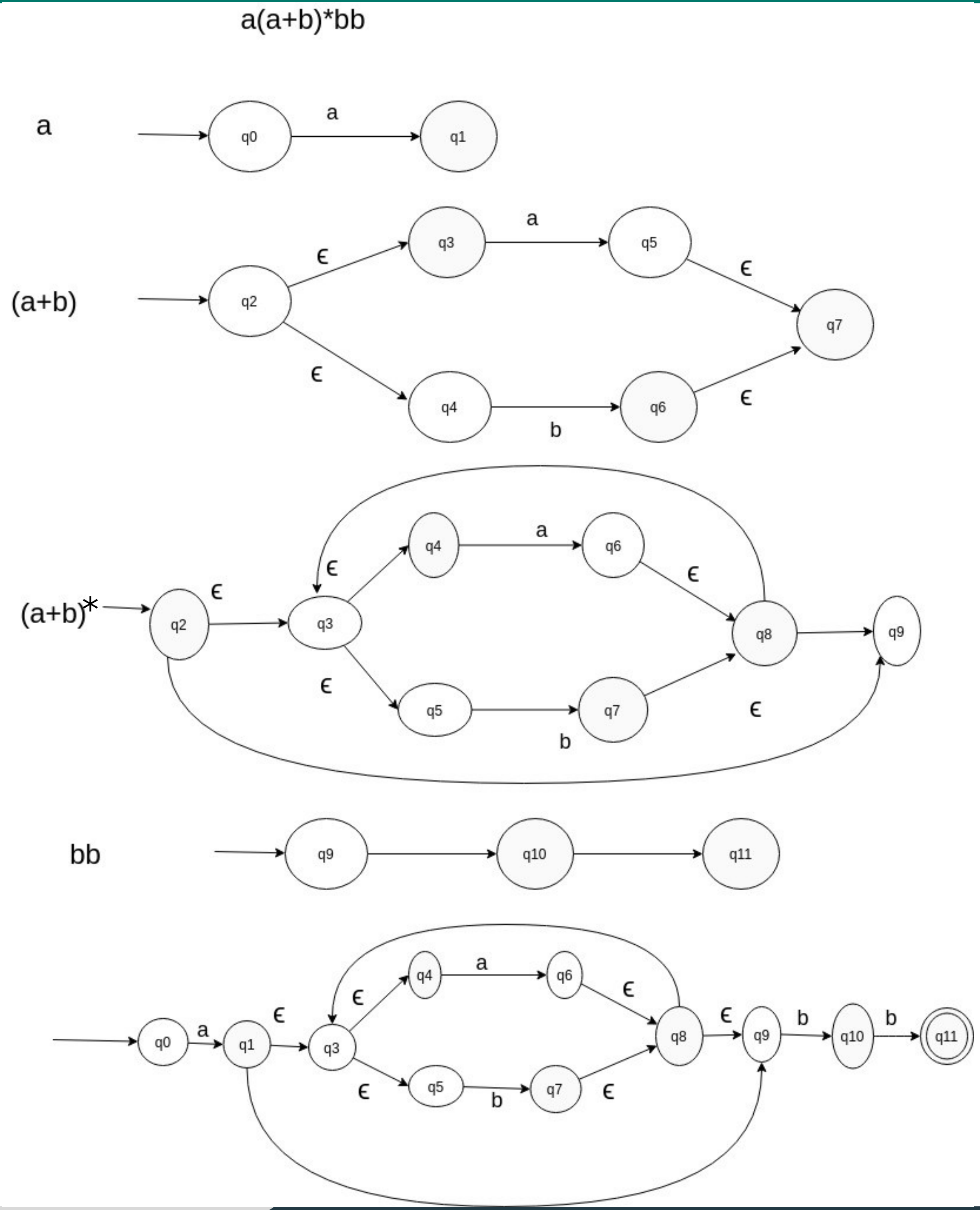


EXAMPLE

a(a+b)*bb



EXAMPLE


a(a+b)*bb
a
q0
a
q1
(a+b)
q2
ϵ
q3
a
q5
ϵ
q7
ϵ
q4
b
q6
ϵ
(a+b)*
q2
ϵ
q3
ϵ
q4
a
q6
ϵ
q8
q9
ϵ
q5
b
q7
ϵ
bb
q9
q10
q11
q0
a
q1
ϵ
q3
ϵ
q4
a
q6
ϵ
q8
ϵ
q9
b
q10
b
q11
ϵ
q5
b
q7
ϵ

EXAMPLE

- (a|b)*a
- (b+ab*a)*
- 0*10*

THE SUBSET CONSTRUCTION ALGORITHM

The algorithm for constructing a DFA from a given NFA such that it recognizes the same language is called subset construction.

Operation	Definition
∈-*closure*(*s*)	set of NFA states reachable from state *s* on ∈-transition
∈-*closure*(*T*)	set of NFA states reachable from some *s* in *T* on ∈-transition
move(*T*, *a*)	set of NFA states to which there is transition on input *a* from some state *s* in the set *T*

THE SUBSET CONSTRUCTION ALGORITHM

```
Initialize: Let  $\in$-closure( $s_0$ )  be the only state in Dstates ( of the
DFA )
Repeat:    while there are unmarked states  T  in  Dstates  do
               mark T
               for each symbol  a  do
                   U =    $\in$-closure( move( T, a ) )
                   if  U  is not in Dstates then
                       add  U  as unmarked state in  Dstates
                   Dtran[ T, a ] = U
               end
           end
```

EXAMPLE

Steps to Convert NFA with ε-move to DFA :

Step 1 : *Take ∈ closure for the beginning state of NFA as beginning state of DFA.*

Step 2 : *Find the states that can be traversed from the present for each input symbol*

(union of transition value and their closures for each states of NFA present in current state of DFA).

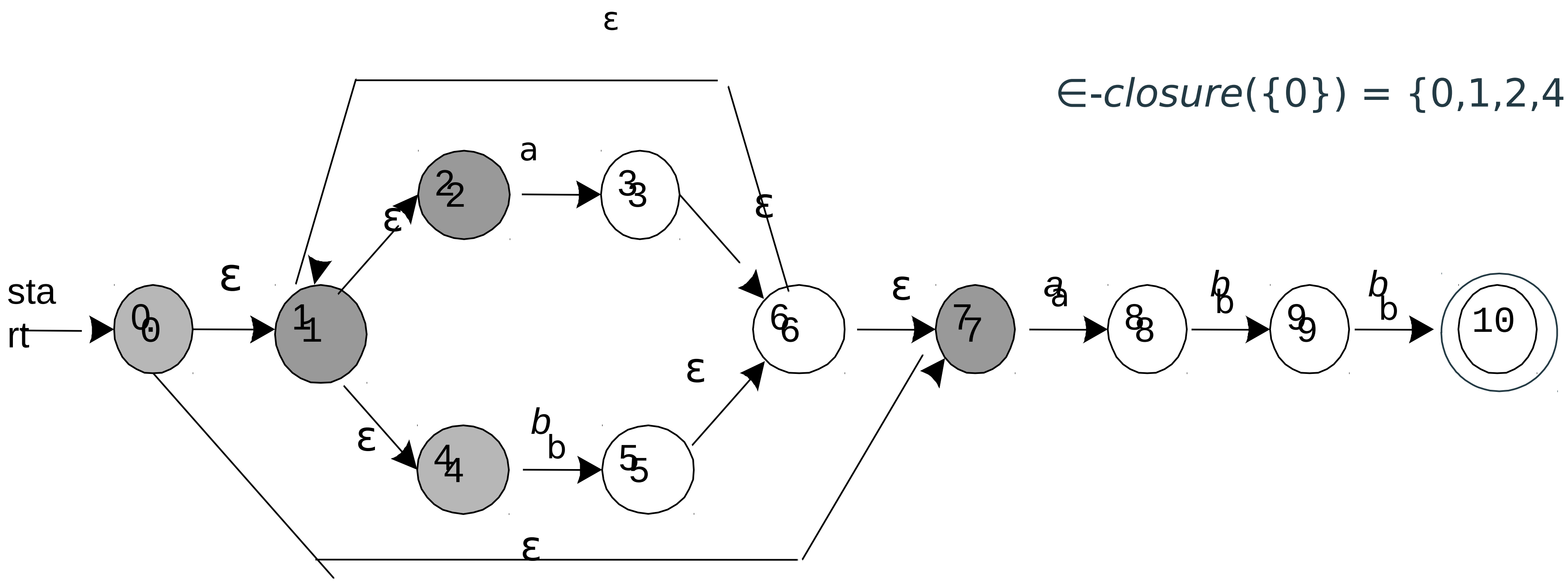
Step 3 : *If any new state is found take it as current state and repeat step 2.*

Step 4 : *Do repeat Step 2 and Step 3 until no new state present in DFA transition table.*

Step 5 : *Mark the states of DFA which contains final state of NFA as final states of DFA.*

EXAMPLE

Convert the NFA for the expression: (*a* | *b*)**abb*
into a DFA using the subset construction algorithm.



∈-*closure*({0}) = {0,1,2,4,7} = *A*

EXAMPLE

∈-*closure*({3, 8})=∈-*closure*(3) U ∈-*closure*(8)

∈-*closure*(3)={,1,2,3,4,6,7}

∈-*closure*(8)={8}

∈-*closure*({3, 8})={1,2,3,4,6,7,8}

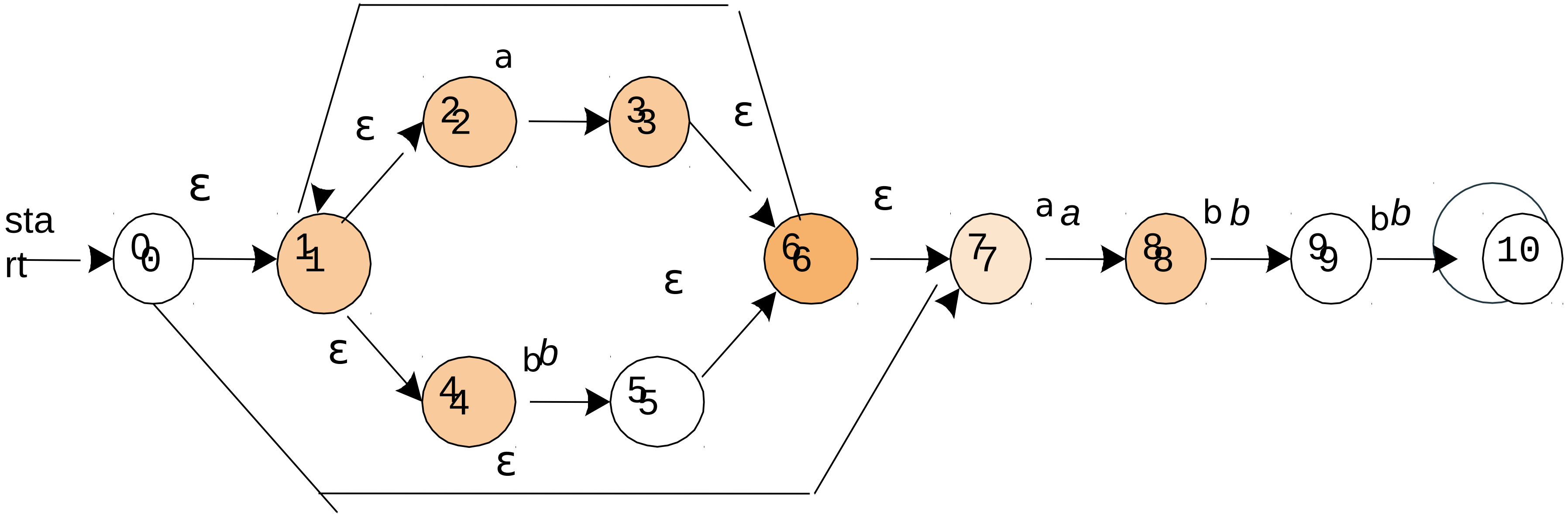
δ’(A,a)= ∈-*closure*(*move*(A,*a*))

=∈-*closure*(*move*({0,1,2,4,7},*a*))

=∈-*closure*({3, 8})

={1,2,3,4,6,7,8}

δ’(A,a) = *B*



EXAMPLE

$\delta'(A,b) = \in\text{-}closure(move(A,b))$

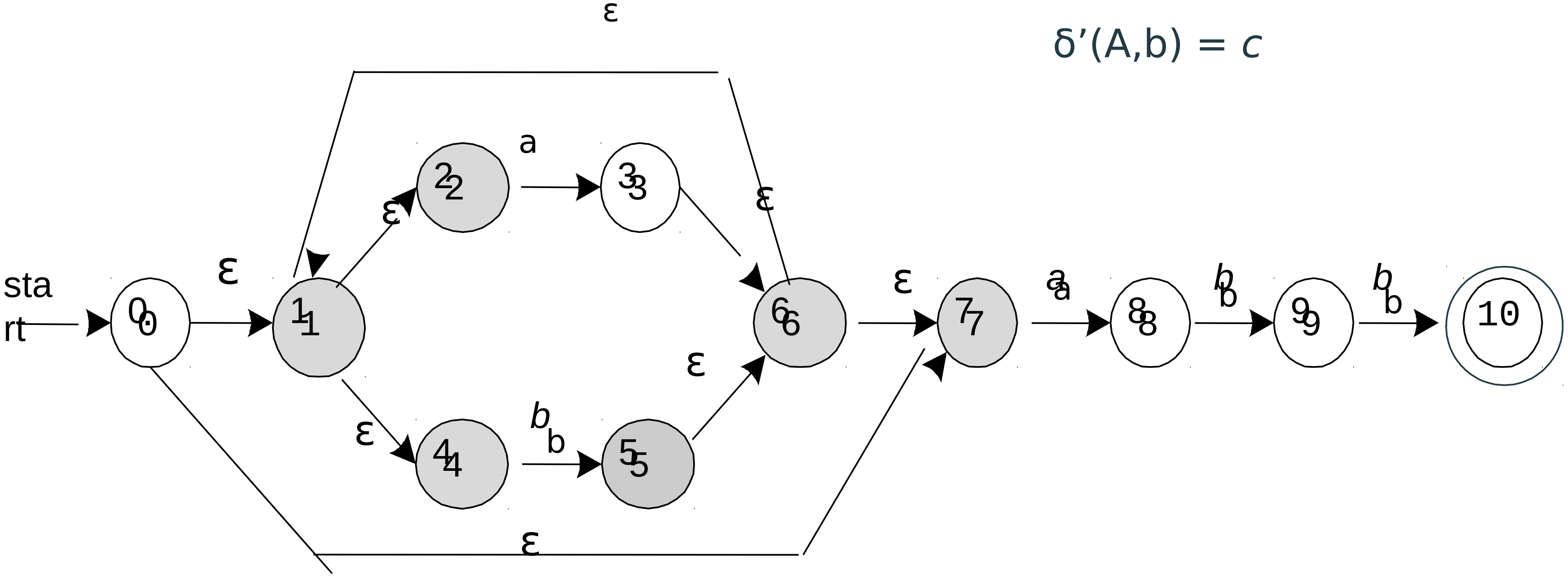
$= \in\text{-}closure(move(\{0,1,2,4,7\},b))$

$= \in\text{-}closure(\ \{5\}\ )$

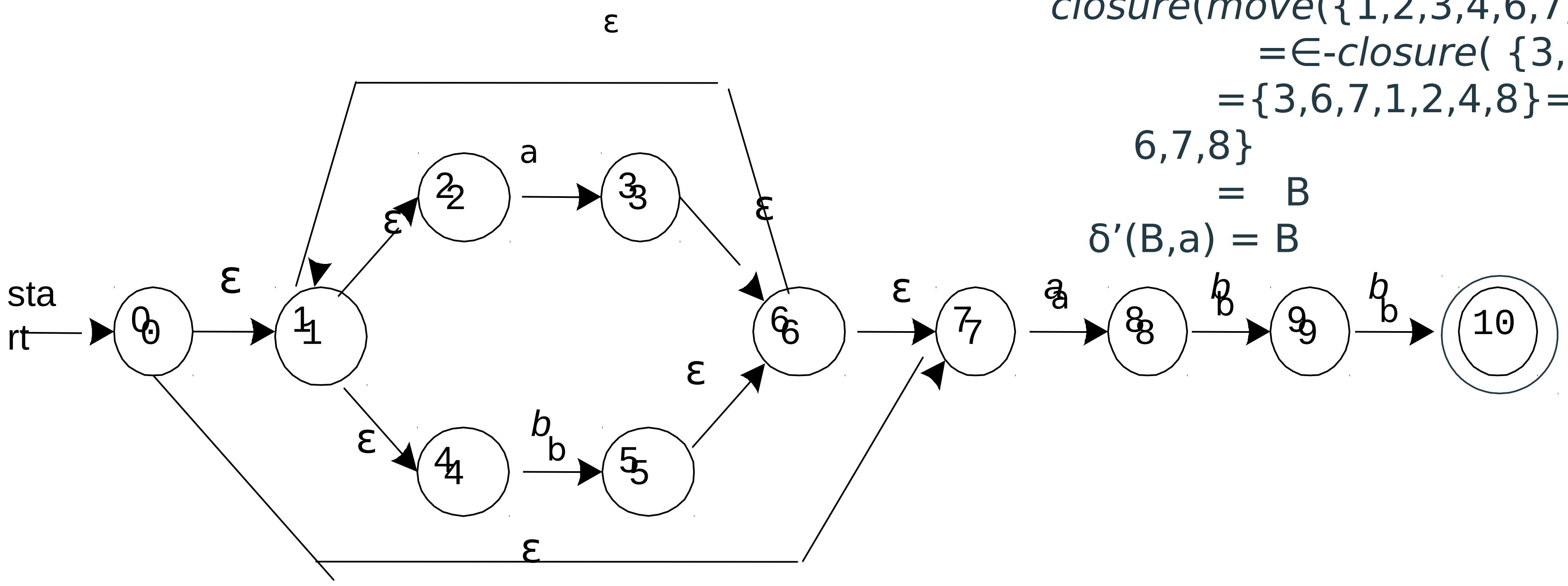
$= \{1,2,4,5,6,7\}$

$= C$

$\delta'(A,b) = c$



EXAMPLE



{1,2,3,4,6,7,8}=B

δ’(B,a) = ∈-*closure*(*move*(B,a))

=∈-*closure*(*move*({1,2,3,4,6,7,8},*a*))

=∈-*closure*({3,8})

={3,6,7,1,2,4,8}=={1,2,3,4, 6,7,8}

= B

δ’(B,a) = B

EXAMPLE

{1,2,3,4,6,7,8}=B

δ'(B,b) = ∈-*closure*(*move*(B,b))

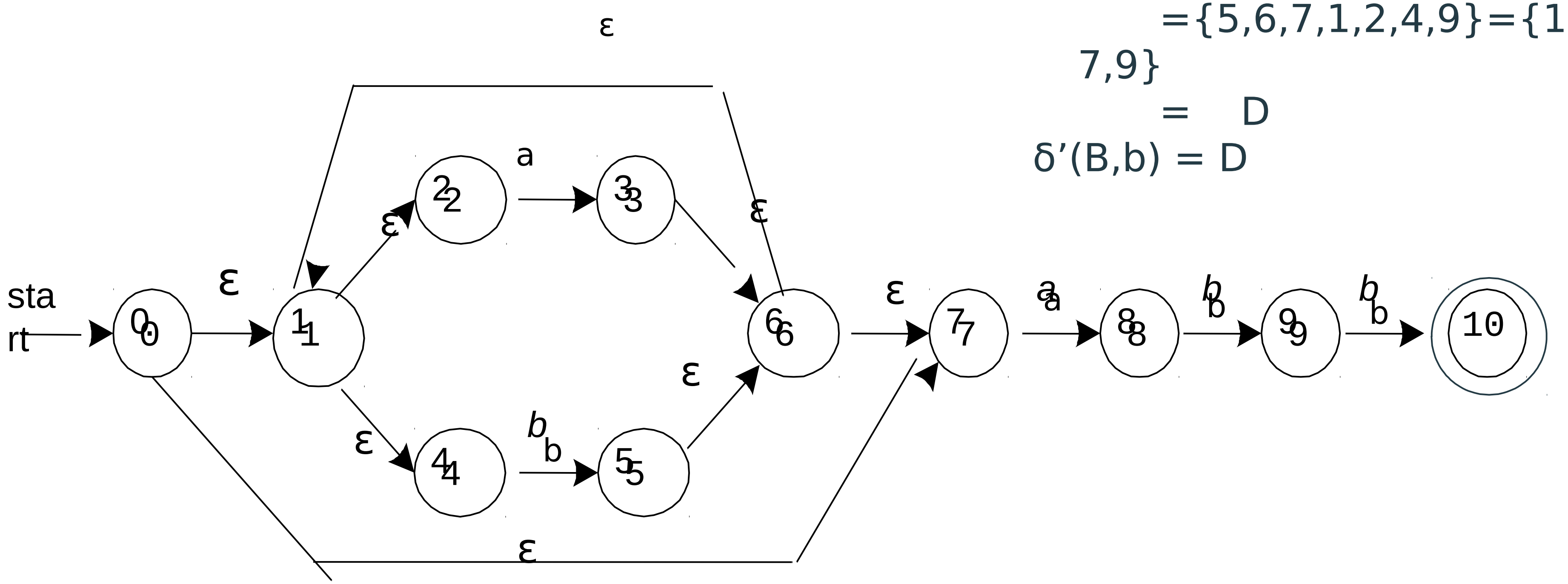
=∈-*closure*(*move*({1,2,3,4,6,7,8},*b*))

=∈-*closure*({5,9})

={5,6,7,1,2,4,9}={1,2,4,5,6,7,9}

= D

δ'(B,b) = D



EXAMPLE

{1,2,4,5,6,7}

δ’(C,a) = ∈-*closure*(*move*(C,a))

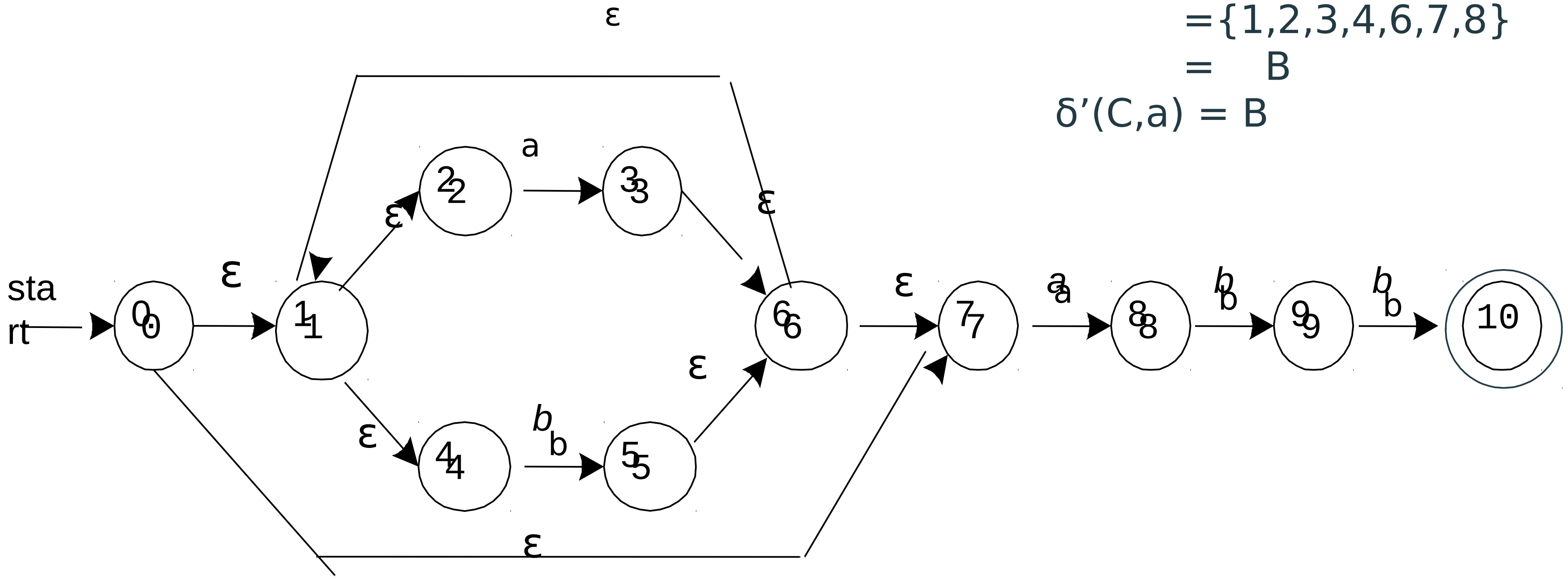
=∈-*closure*(*move*({1,2,4,5,6,7},*a*))

=∈-*closure*({3,8})

={1,2,3,4,6,7,8}

= B

δ’(C,a) = B



EXAMPLE

{1,2,4,5,6,7}

δ’(C,b) = ∈-*closure*(*move*(C,b))

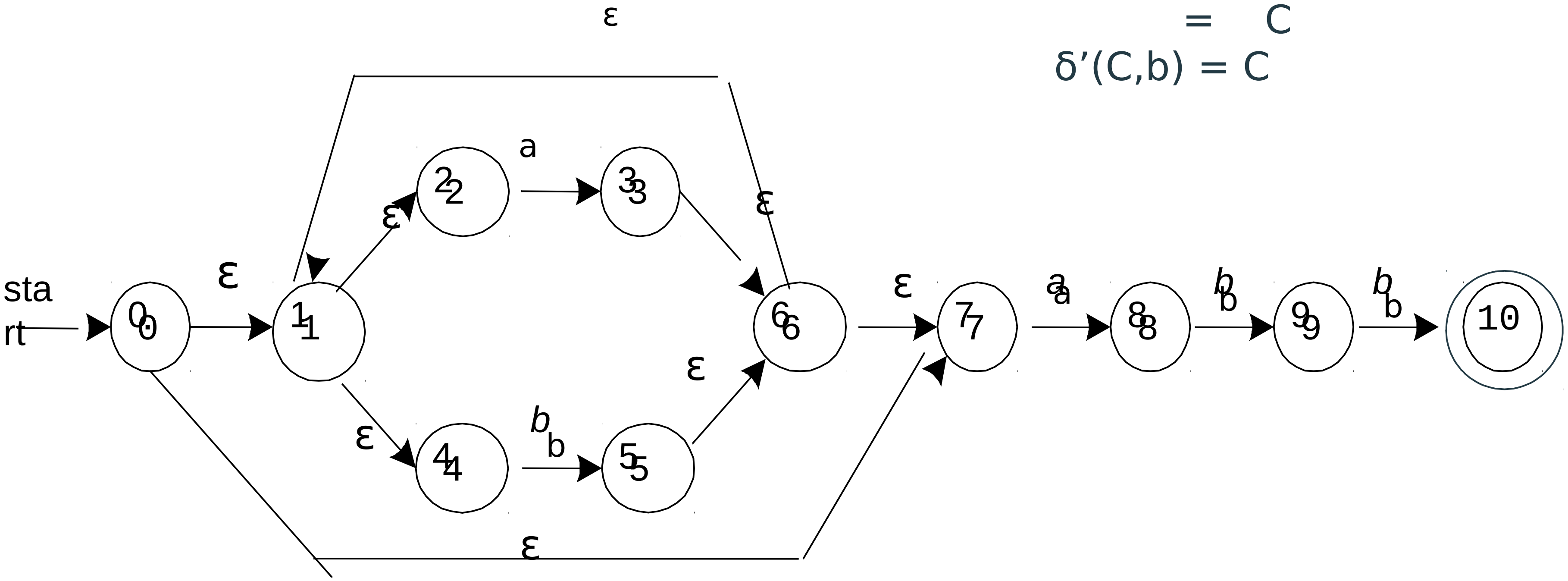
=∈-*closure*(*move*({},*b*))

=∈-*closure*({5})

={1,2,4,5,6,7}

= C

δ’(C,b) = C



EXAMPLE

{1,2,4,5,6,7,9}

δ'(D,a) = ∈-*closure*(*move*(D,a))

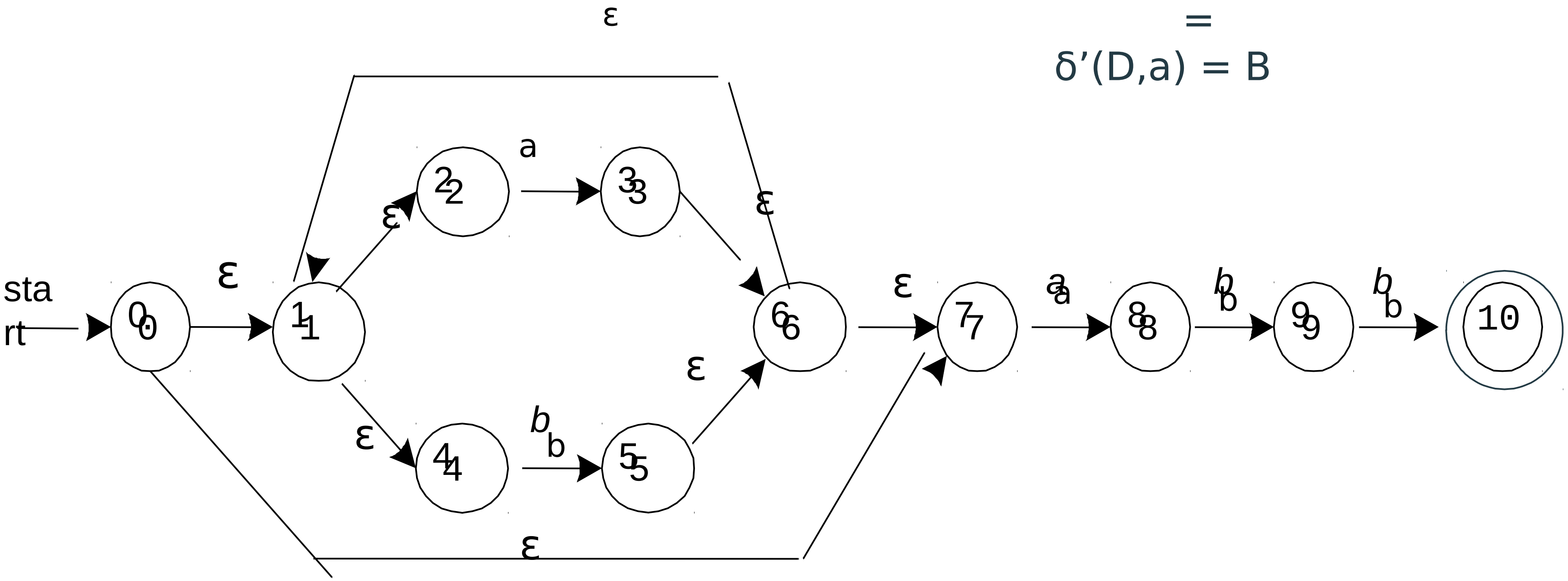
=∈-*closure*(*move*({},*a*))

=∈-*closure*({3,8})

=B

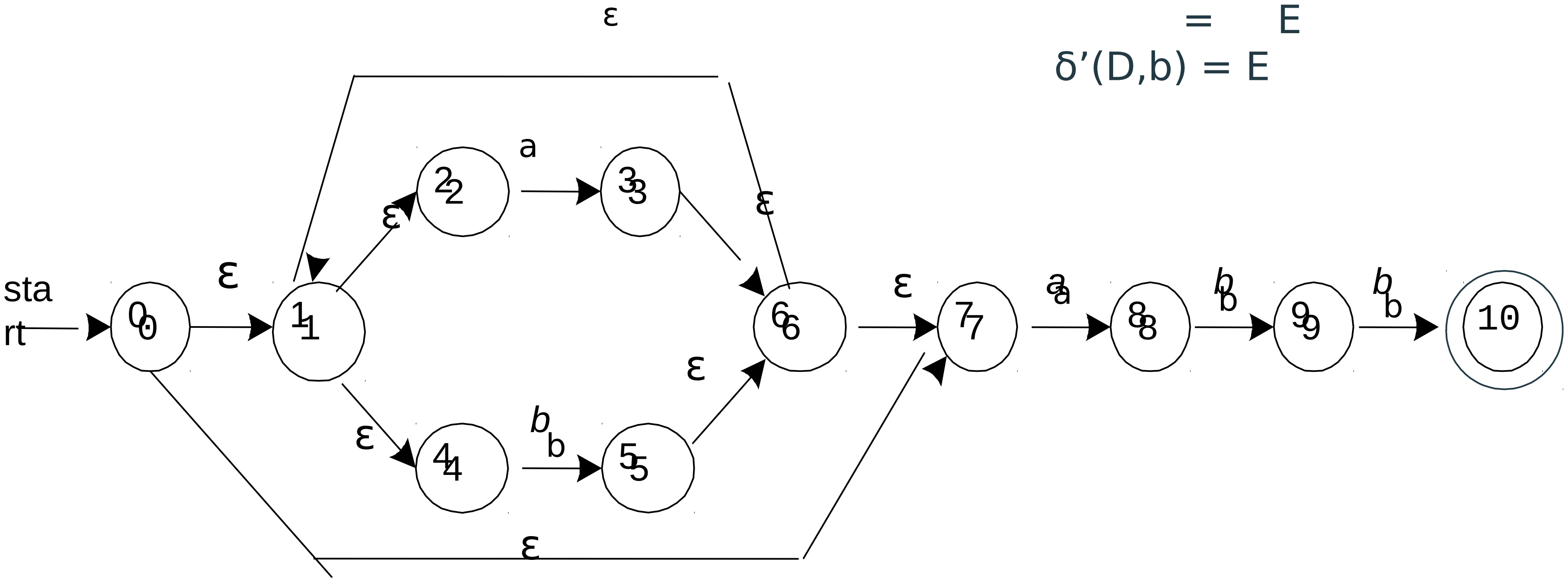
=

δ'(D,a) = B

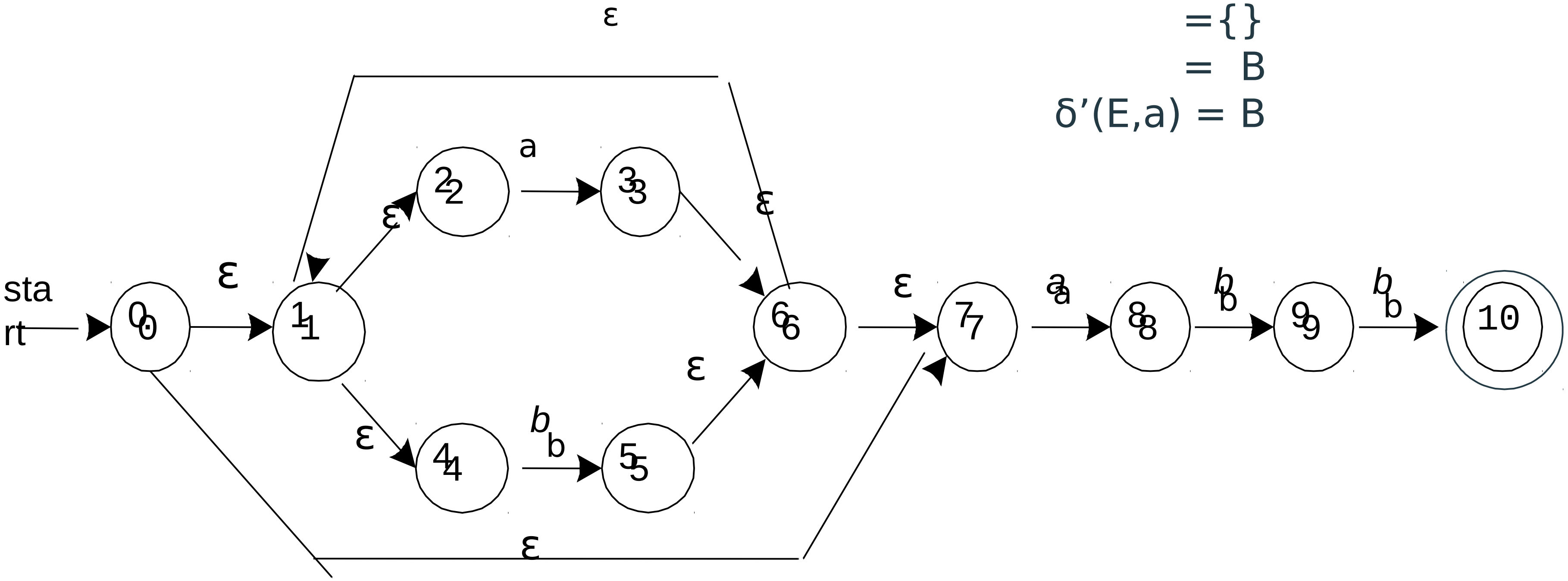


EXAMPLE

δ’(D,b) = ∈-*closure*(*move*(D,b))
=∈-*closure*(*move*({},*b*))
=∈-*closure*({5,10})
={5,6,7,1,2,4,10}=
{1,2,4,5,6,7,10}
= E
δ’(D,b) = E



EXAMPLE



{1,2,4,5,6,7,10}

δ’(E,a) = ∈-*closure*(*move*(E,a))

=∈-*closure*(*move*({1,2,4,5,6,7,10},*a*))

=∈-*closure*({3,8})

={}

= B

δ’(E,a) = B

EXAMPLE

{1,2,4,5,6,7,10}

δ'(E,b) = ∈-*closure*(*move*(E,b))

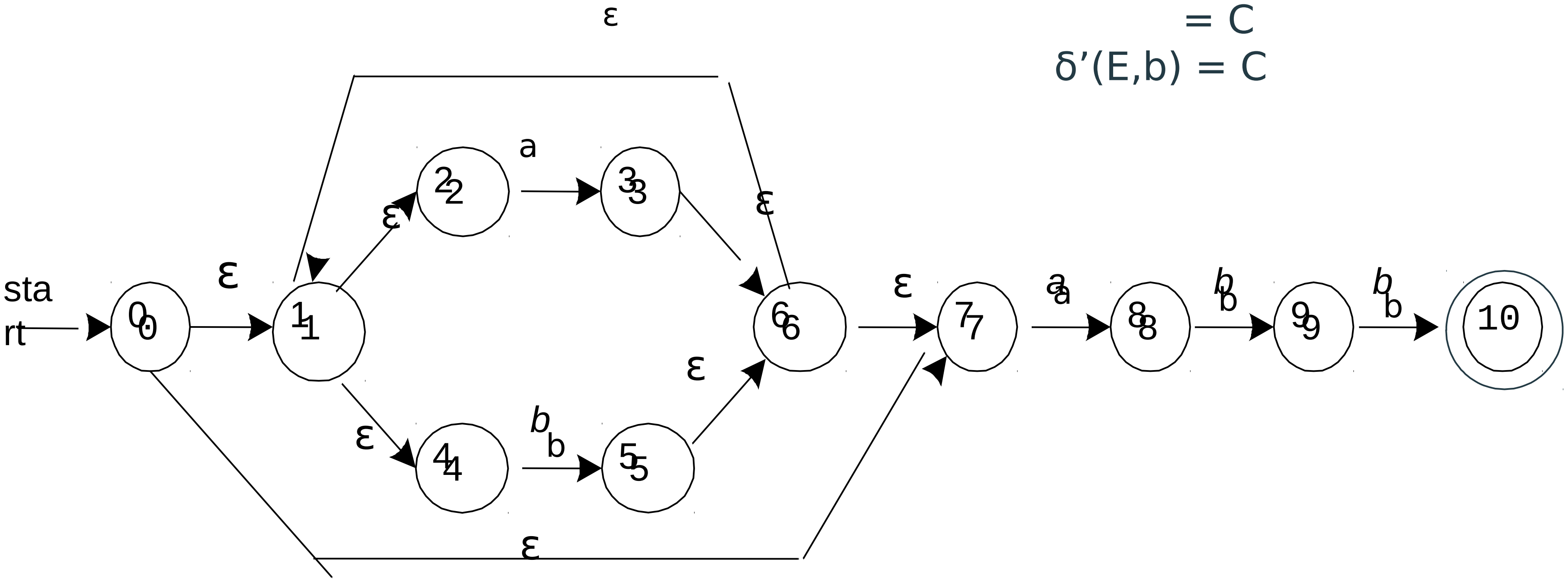
=∈-*closure*(*move*({},*b*))

=∈-*closure*({5})

={}

= C

δ'(E,b) = C



EXAMPLE

$\in$-*closure*({0}) = {0,1,2,4,7} = *A*

A

$\in$-*closure*(*move*({0,1,2,4,7},*a*))=$\in$-*closure*({3, 8})={1,2,3,4,6,7,8}= *B*

$\in$-*closure*(*move*({1,2,4,7},*b*))=$\in$-*closure*({5})={1,2,4,5,6,7}= *C*

B

$\in$-*closure*(*move*({1,2,3,4,6,7,8},*a*) =$\in$-*closure*({3, 8})= *B*

$\in$-*closure*(*move*({1,2,3,4,6,7,8},*b*) =$\in$-*closure*({5, 9})={1,2,4,5,6,7,9}= *D*

C

$\in$-*closure*(*move*({1,2,4,5,6,7},*a*))=$\in$-*closure*({3,8})= *B*

$\in$-*closure*(*move*({1,2,4,5,6,7},*b*))=$\in$-*closure*({5})= *C*

D

$\in$-*closure*(*move*({1,2,4,5,6,7,9},*a*))=$\in$-*closure*({3,8})= *B*

$\in$-*closure*(*move*({1,2,4,5,6,7,9},*b*))=$\in$-*closure*({5,10})={1,2,4,5,6,7,10}= *E*

E

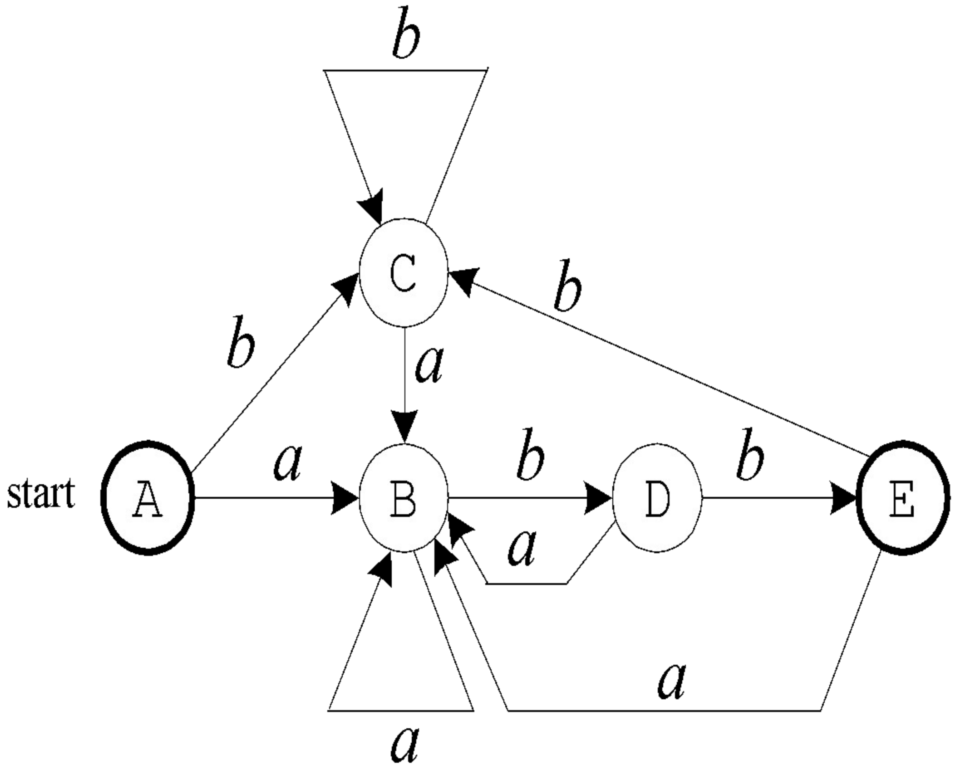
$\in$-*closure*(*move*({1,2,4,5,6,7,10},*a*))=$\in$-*closure*({3,8})= *B*

$\in$-*closure*(*move*({1,2,4,5,6,7,10},*b*))=$\in$-*closure*({5})= *C*

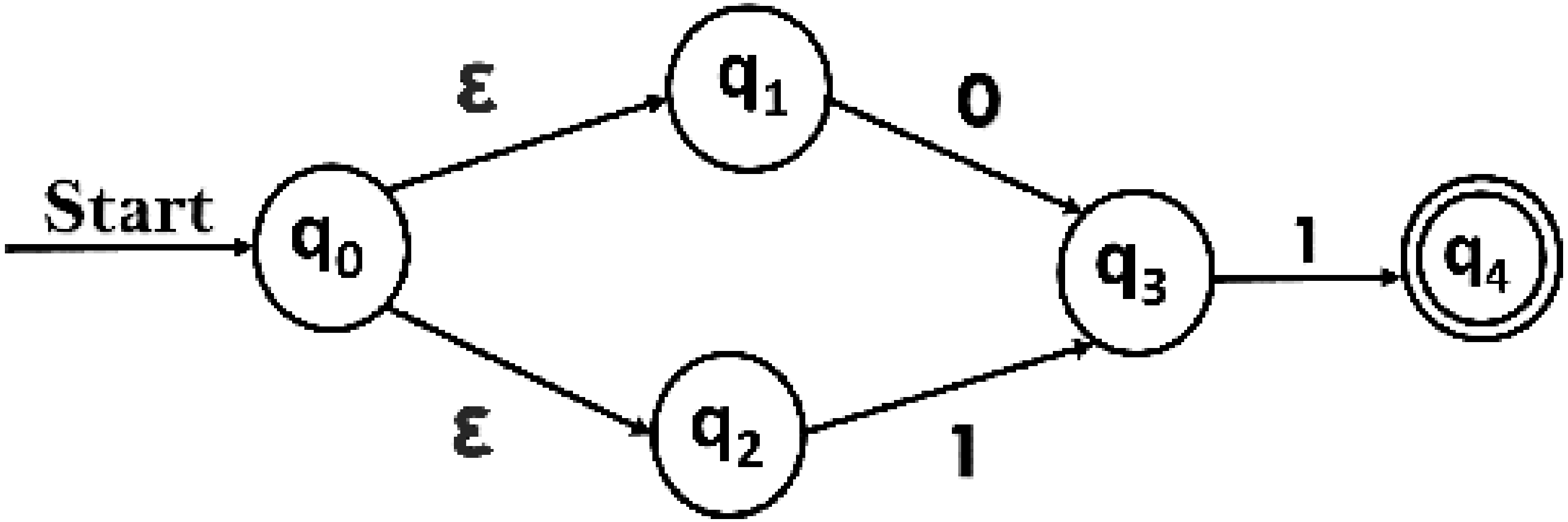
EXAMPLE

The transition table for this DFA becomes:

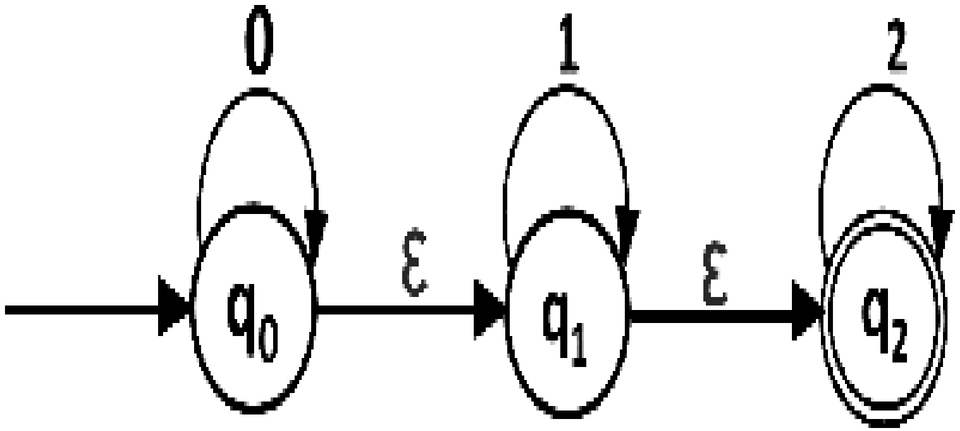
state	*a*	*b*
A	*B*	*C*
B	*B*	*D*
C	*B*	*C*
D	*B*	*E*
E	*B*	*C*



1



2



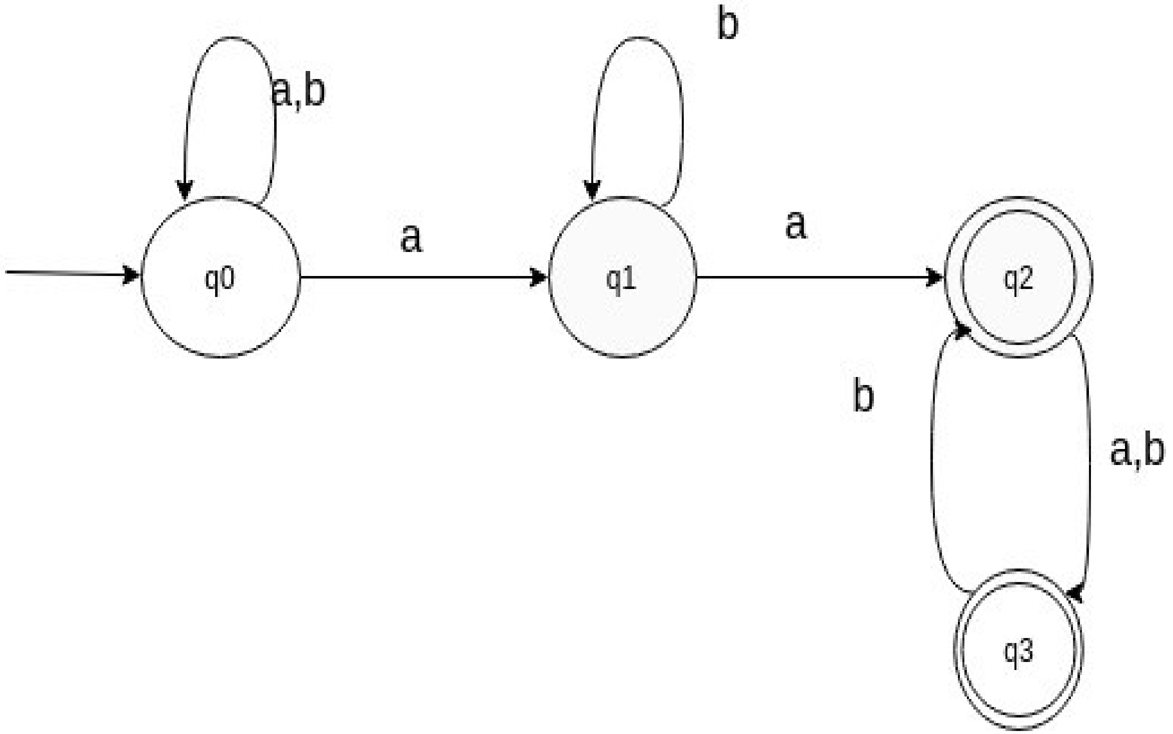
EXAMPLE

δ(q0,a)={q0,q1} B
δ(q0,b)= q0 A
δ(B,a)=δ({q0,q1},a)
δ({q0,q1},a)=δ(q0,a) ∪ δ(q1,a)
={q0,q1} ∪ q2
= {q0,q1,q2} C
δ(B,b)=δ({q0,q1},b)
δ({q0,q1},b)=δ(q0,b) ∪ δ(q1,b)
=q0 ∪ q1
= {q0,q1} B

δ({q0.q1,q2},a)= {q0,q1,q2,q3} D
δ({q0.q1,q2},b)= {q0,q1,q3} E

δ({q0,q1,q2,q3},a)= {q0,q1,q2,q3} D
δ({q0,q1,q2,q3},b)= {q0,q1,q2,q3} D

δ({q0.q1,q3},a)= {q0,q1,q2} C
δ({q0.q1,q3},b)= {q0,q1,q2} C



δ/Σ	a	b
A	B	A
B	C	B
C	D	E
D	D	D
E	C	C

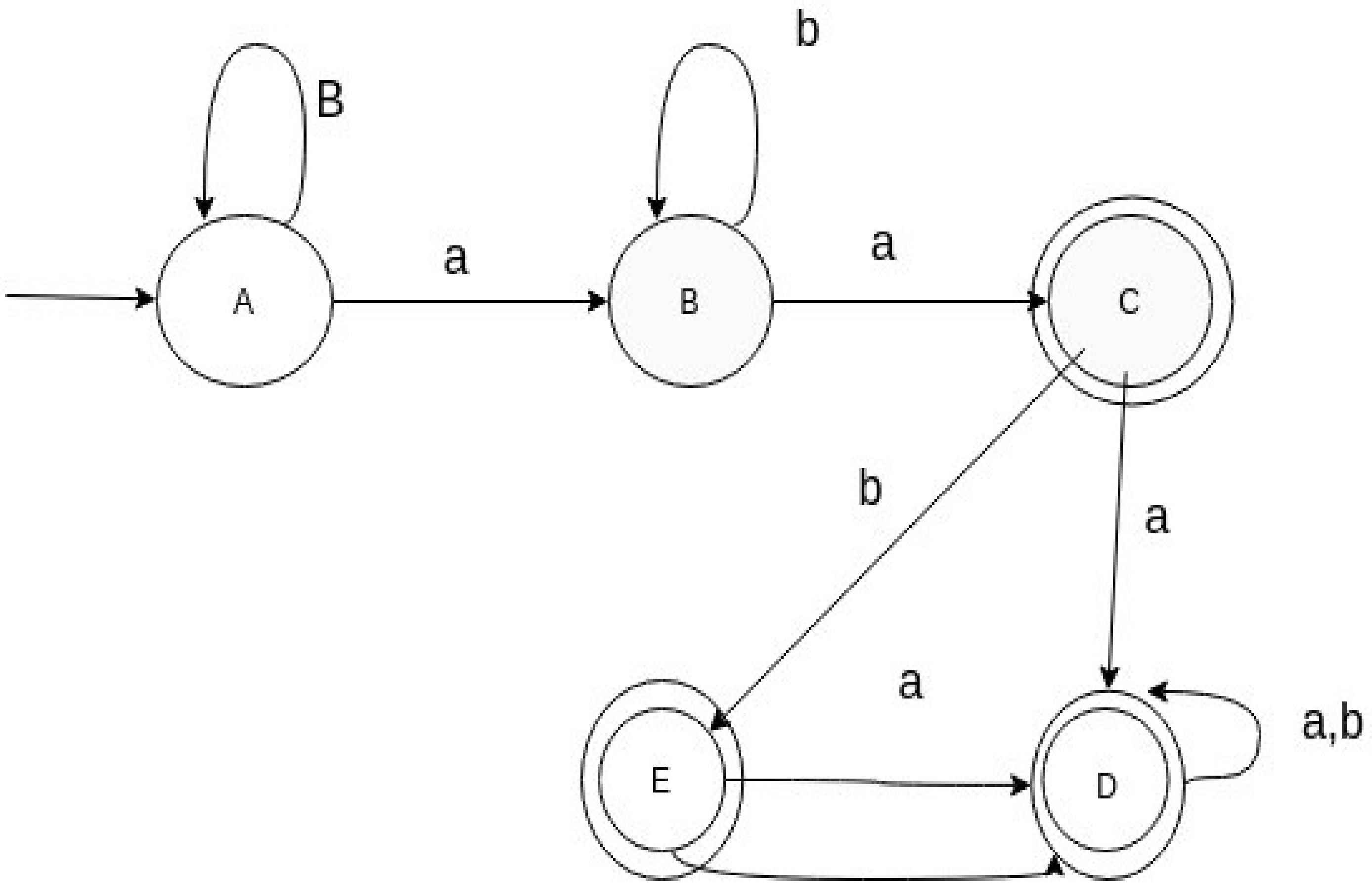
EXAMPLE

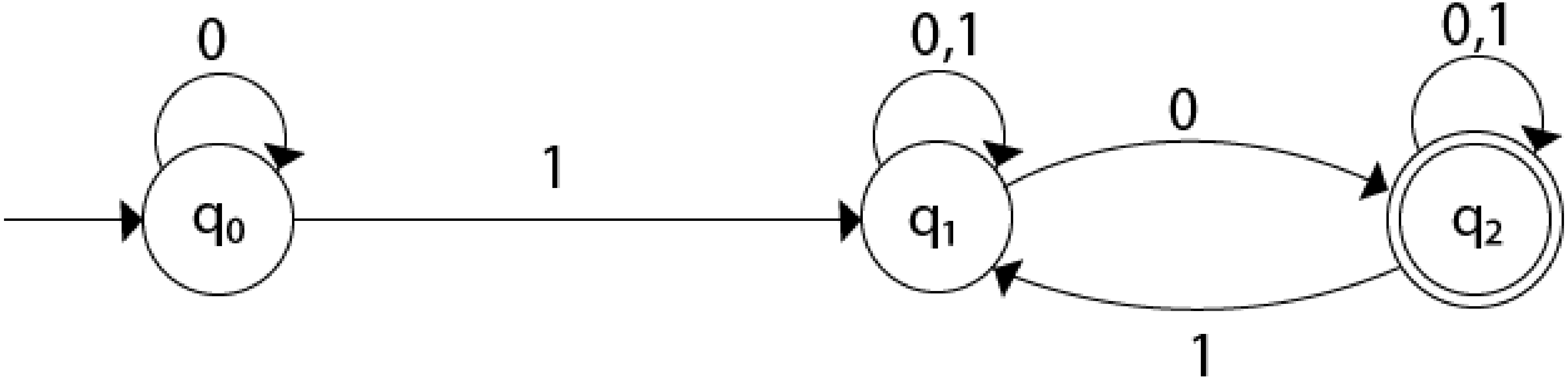
δ/Σ	a	b
q0	q0,q1	q0
q0,q1	q0,q1,q2	q0,q1
q0,q1,q2	q0,q1,q2,q3	q0,q1,q3
q0,q1,q2,q3	q0,q1,q2,q3	q0,q1,q2,q3
q0,q1,q3	q0,q1,q2,q3	q0,q1,q2,q3

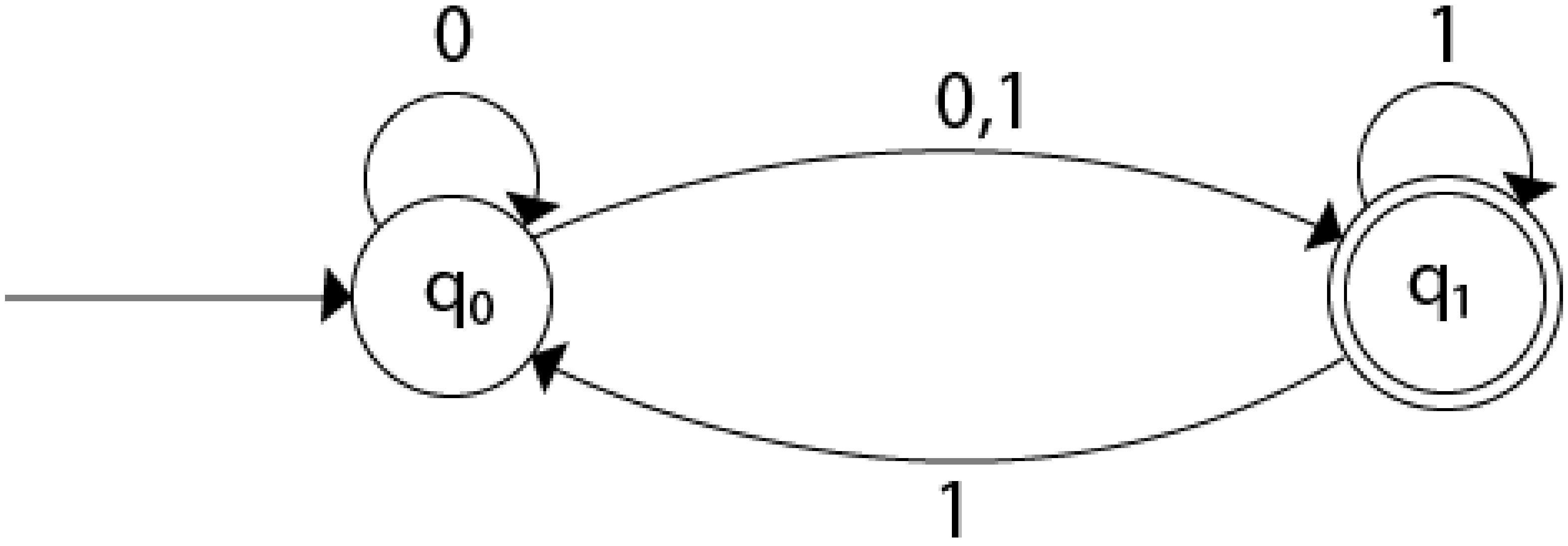
→

δ/Σ	a	b
A	B	A
B	C	B
C	D	E
D	D	D
E	D	D

DFA




0
1
0,1
0
0,1
q_0
q_1
q_2
1


0
0,1
1
q_0
q_1
1