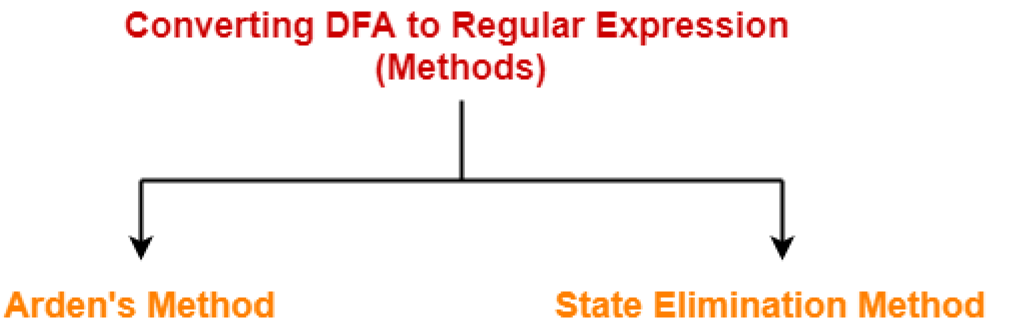
DFA to regular expression conversion



Converting DFA to Regular Expression (Methods)

Arden's Method

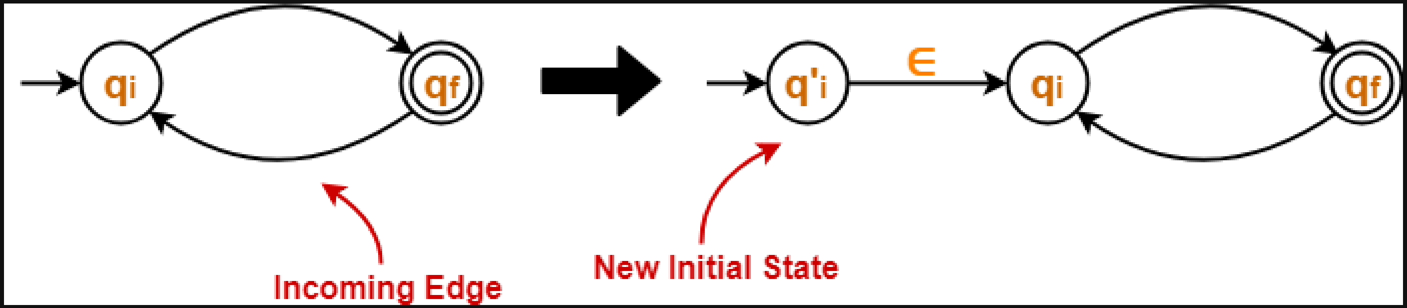
State Elimination Method

State Elimination Method-

This method involves the following steps in finding the regular expression for any given DFA-

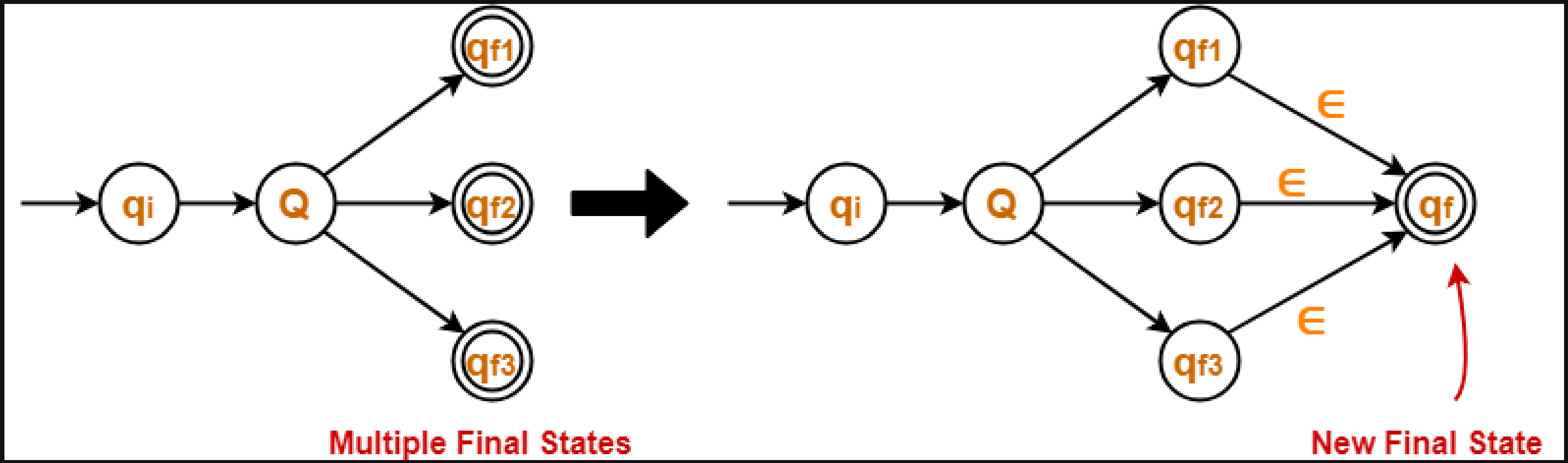
Step-01:

If there exists any incoming edge to the initial state, then create a new initial state having no incoming edge to it.



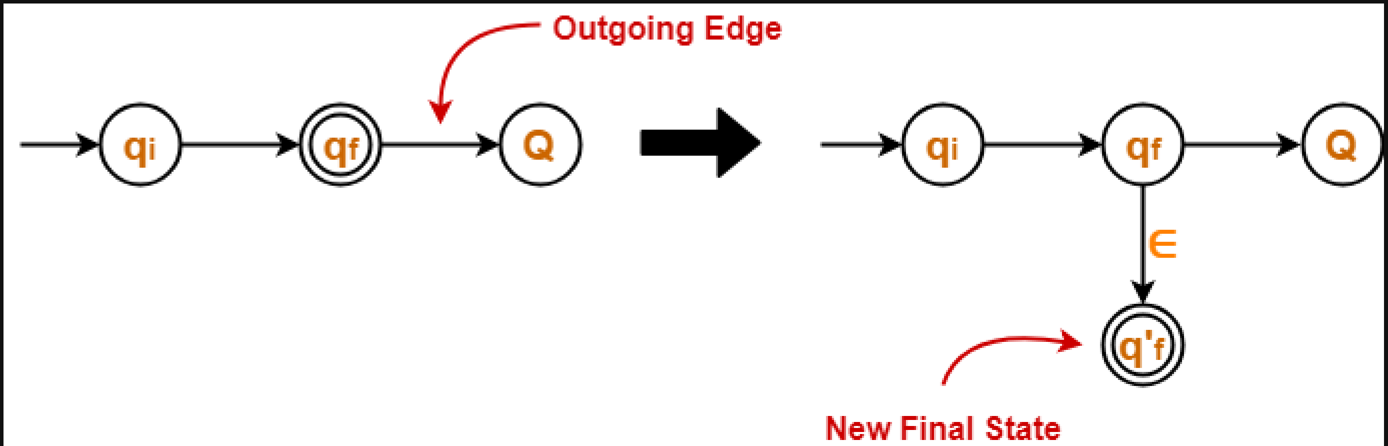
Step-02:

If there exists multiple final states in the DFA, then convert all the final states into non-final states and create a new single final state.



Step-03:

If there exists any outgoing edge from the final state, then create a new final state having no outgoing edge from it.



Step-04:

- Eliminate all the intermediate states one by one.
- These states may be eliminated in any order.

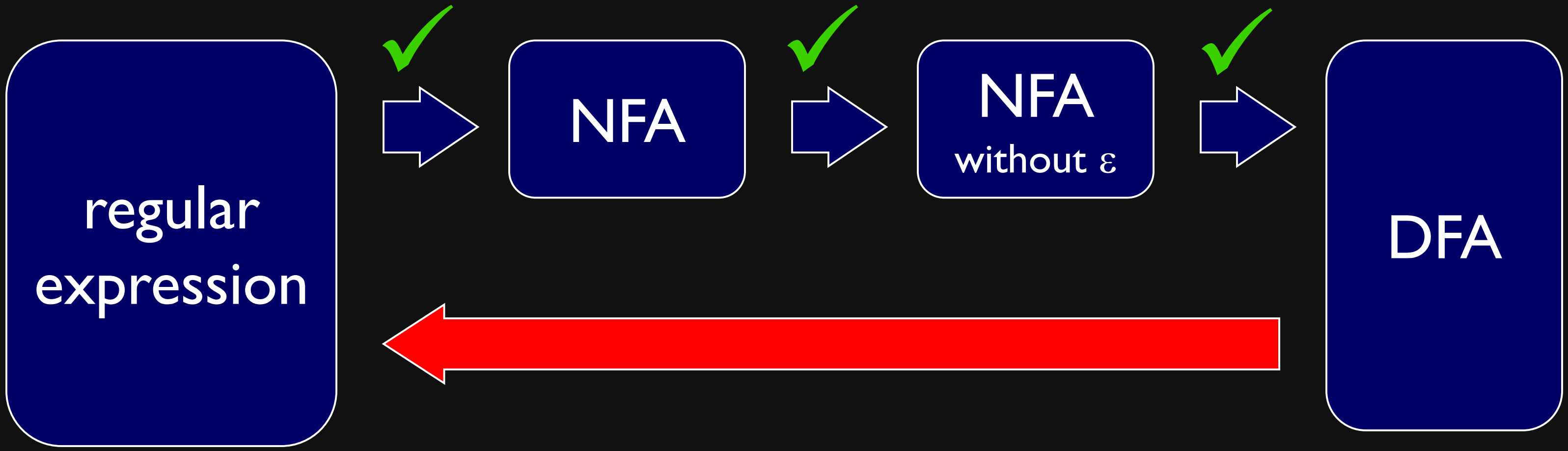
In the end,

- Only an initial state going to the final state will be left.
- The cost of this transition is the required regular expression.

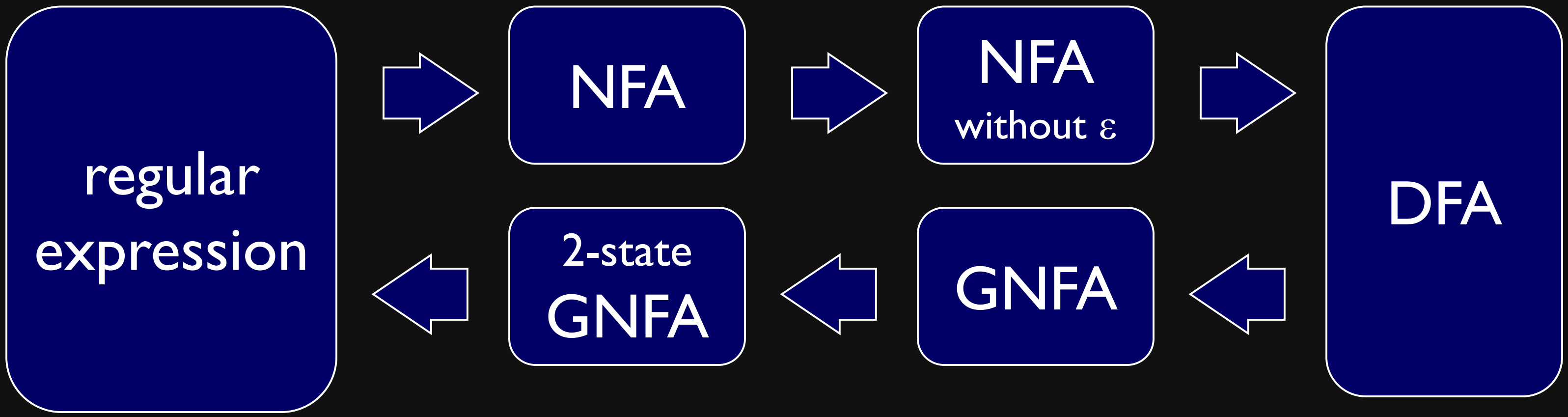
NOTE

The state elimination method can be applied to any finite automata. (NFA, ∈-NFA, DFA etc)

Road map

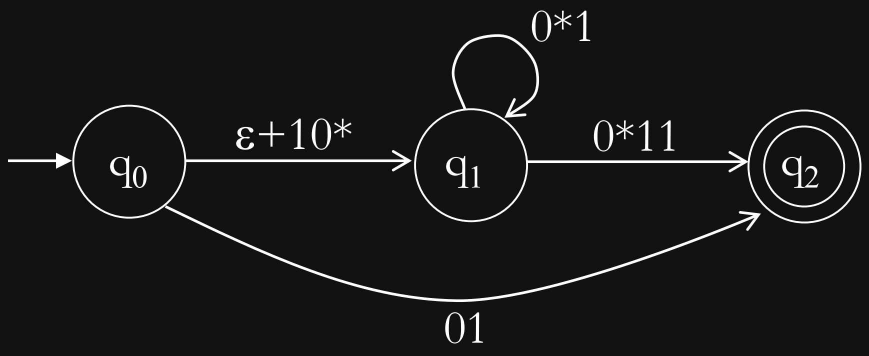


Road map



Generalized NFAs

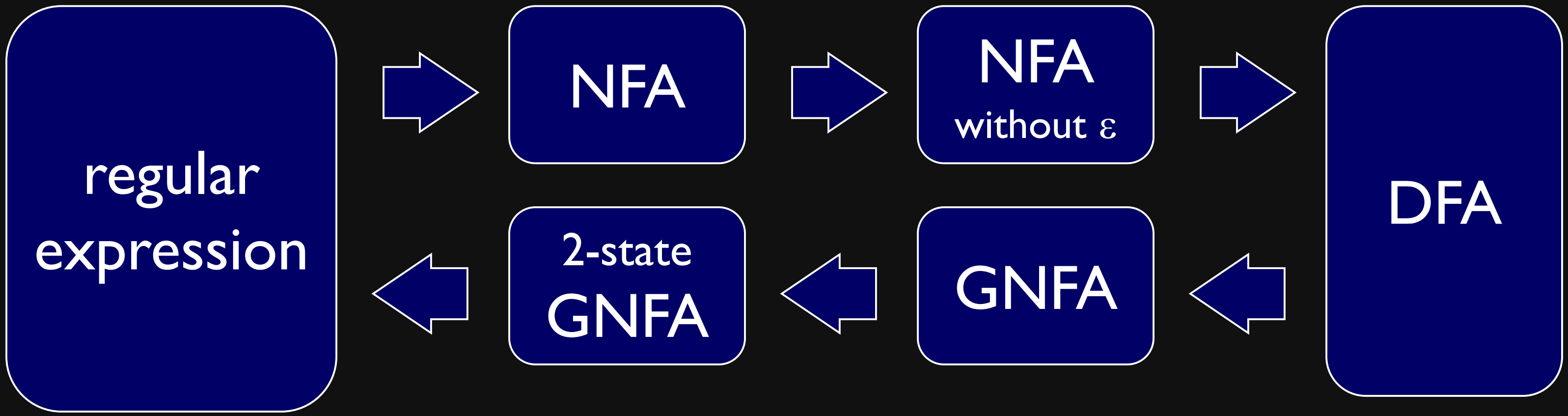
- A generalized NFA is an NFA whose transitions are labeled by regular expressions, like



moreover

- – It has exactly one accept state, different from its start state
- – No arrows come into the start state
- – No arrows go out of the accept state

Converting a DFA to a GNFA



regular expression

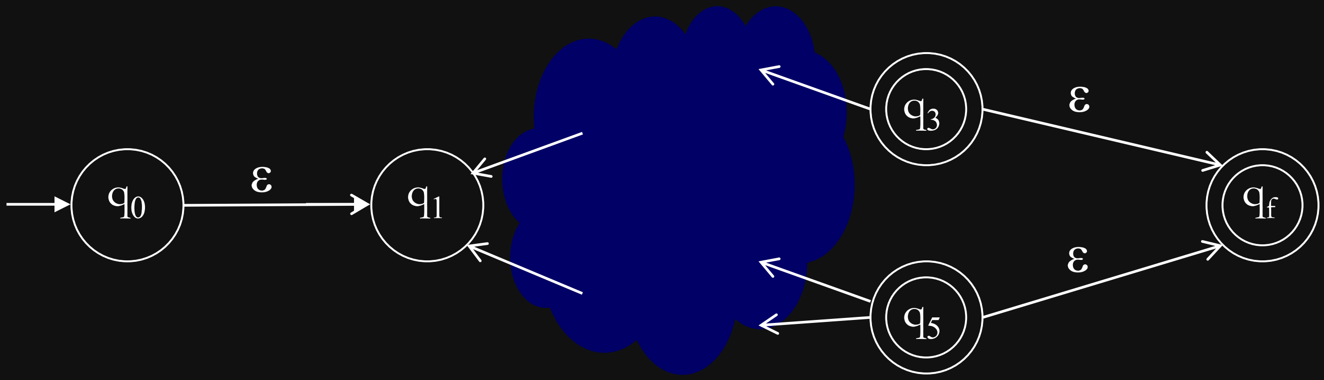
NFA

NFA without ε

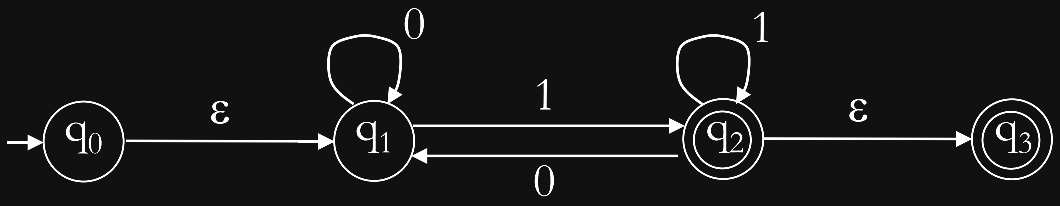
DFA

GNFA

2-state GNFA



Conversion example

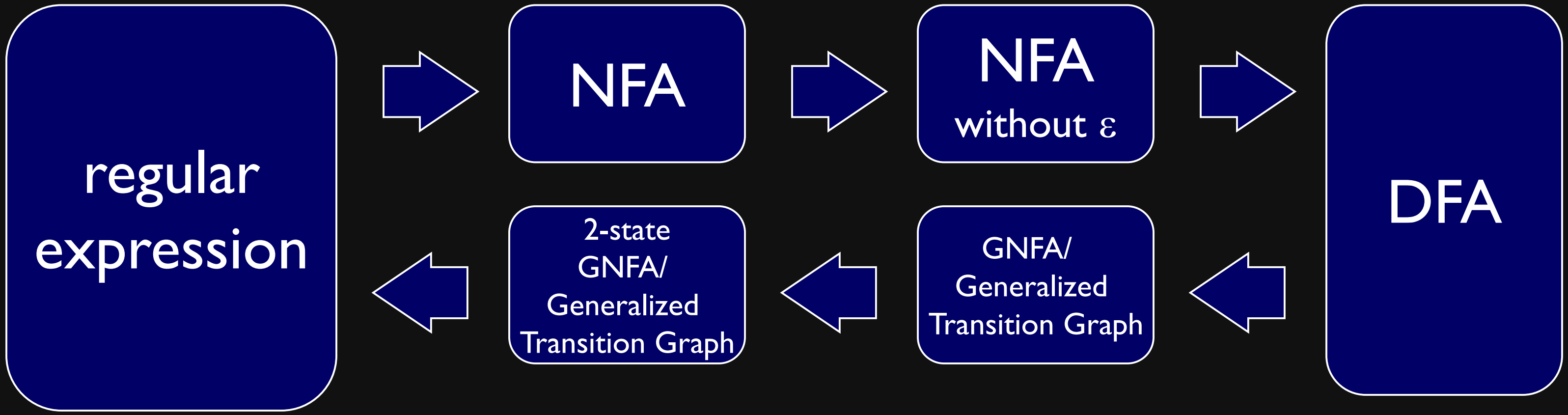


✓ – It has exactly one accept state, different from its start state

✓ – No arrows come into the start state

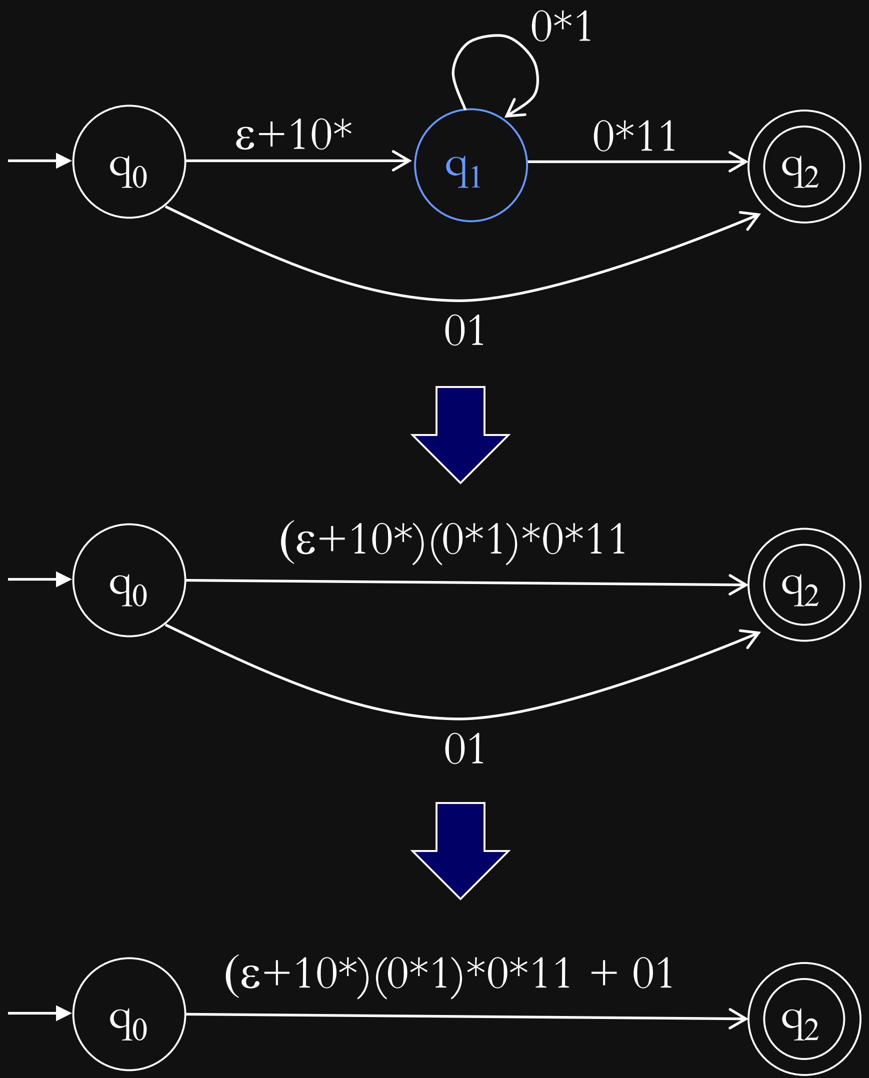
✓ – No arrows go out of the accept state

GNFA state reduction


regular expression
NFA
NFA without ε
DFA
GNFA/ Generalized Transition Graph
2-state GNFA/ Generalized Transition Graph

From any GNFA, we can eliminate every state but the start and accept states

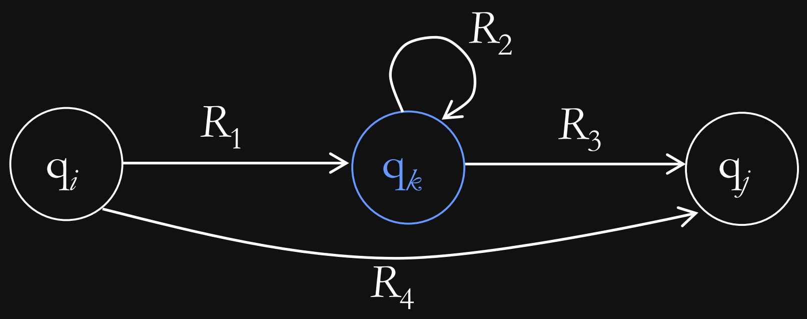
State elimination



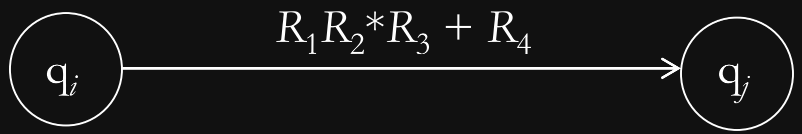
State elimination – general method

- To eliminate state q_k, for every pair of states (q_i, q_j)

Replace

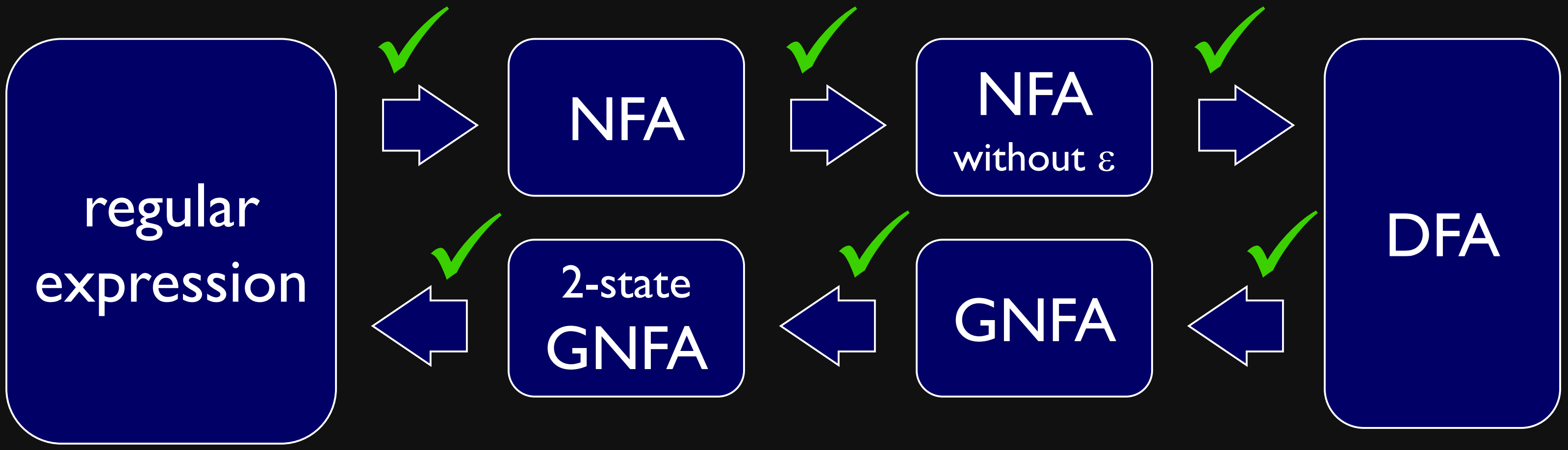


by



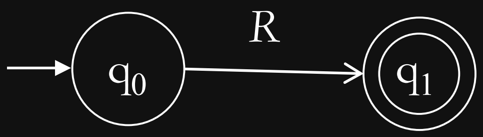
remember to do this even when $q_i = q_j$!

Road map



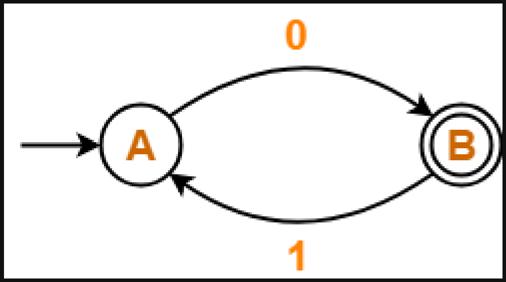
regular expression ✓ → NFA ✓ → NFA without ε ✓ → DFA

DFA ✓ → GNFA ✓ → 2-state GNFA ✓ → regular expression



A 2-state GNFA is the same as a regular expression R!

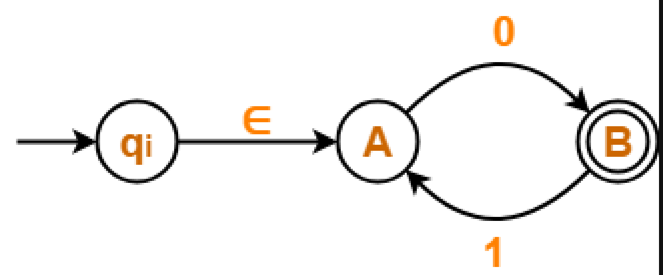
Problem-01:

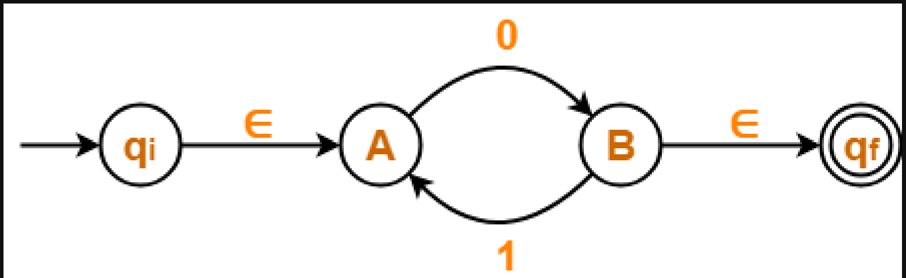


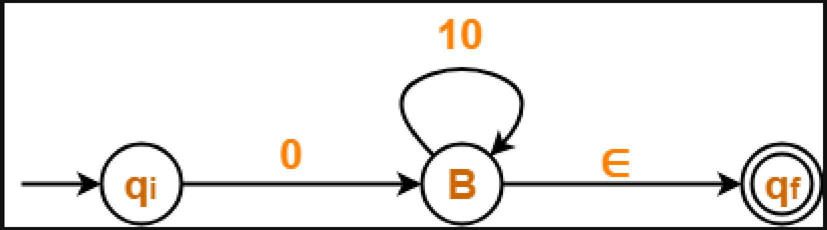
NOTE-

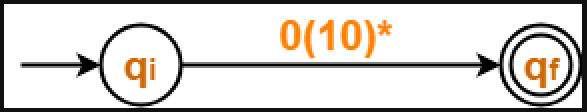
In the above question,

- If we first eliminate state B and then state A, then regular expression would be = (01)*0.
- This is also the same and correct.



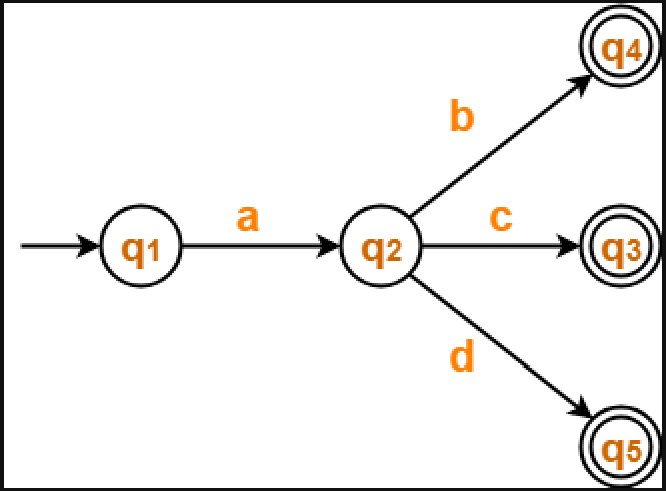


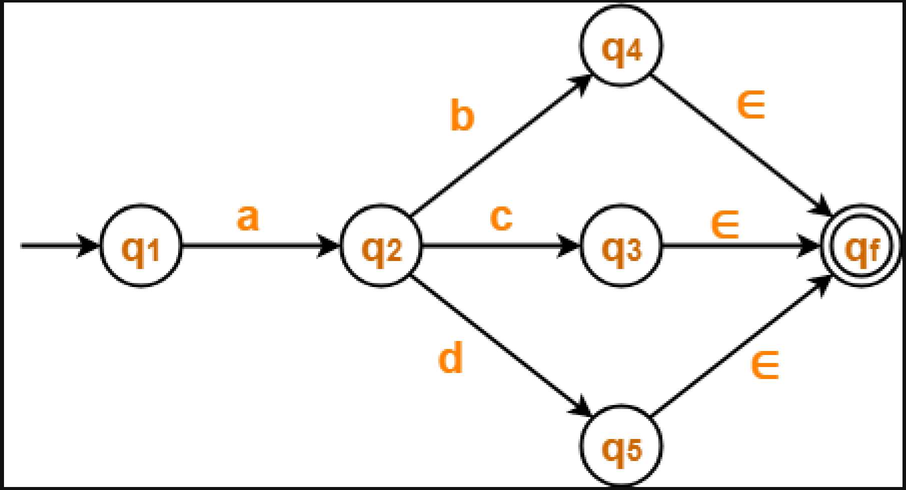


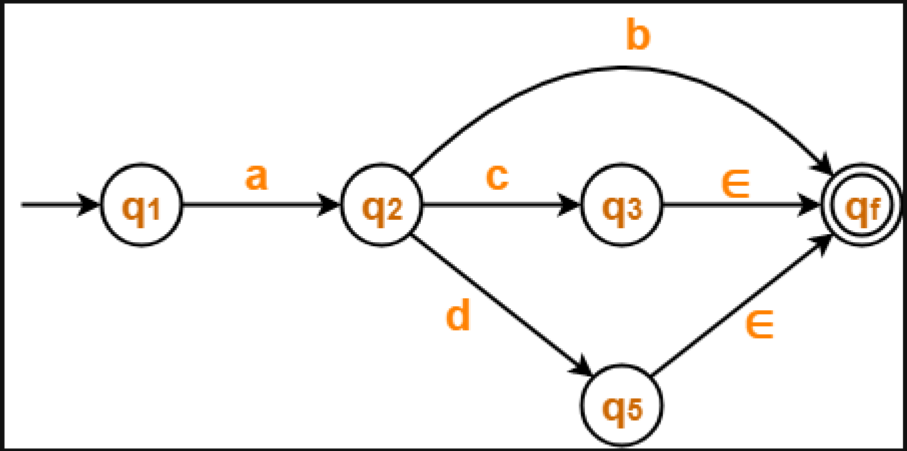


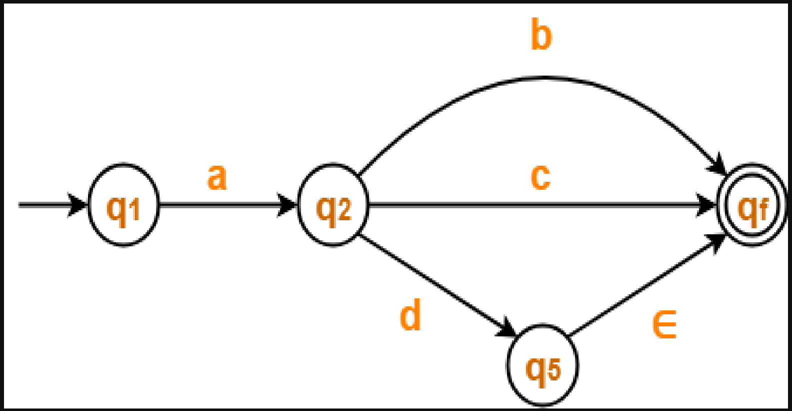
Regular Expression = 0(10)*

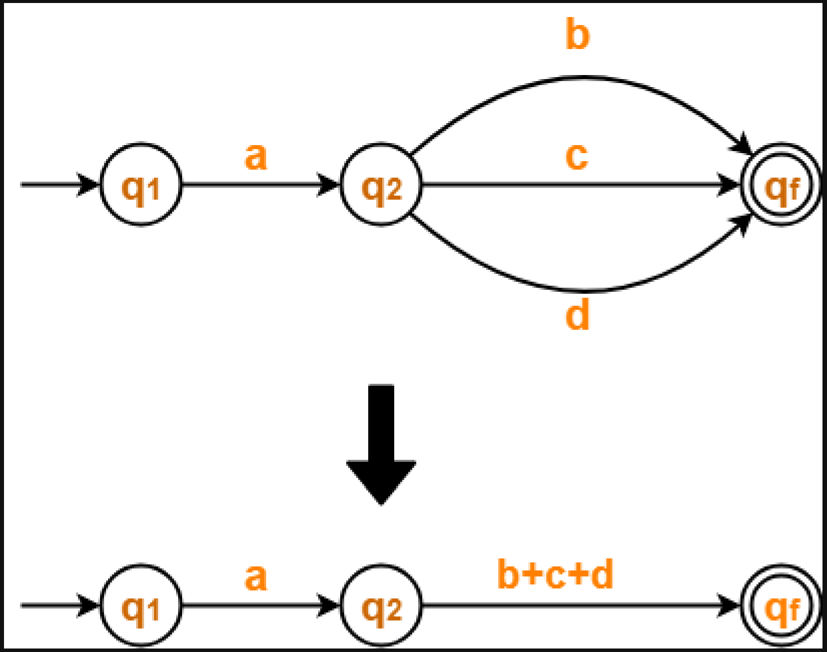
Problem 02

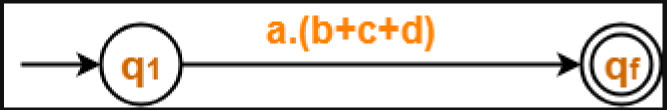

q1
a
q2
b
q4
c
q3
d
q5


q1
a
q2
b
q4
∈
c
q3
∈
qf
d
q5
∈


b
q1
a
q2
c
q3
∈
qf
d
∈
q5

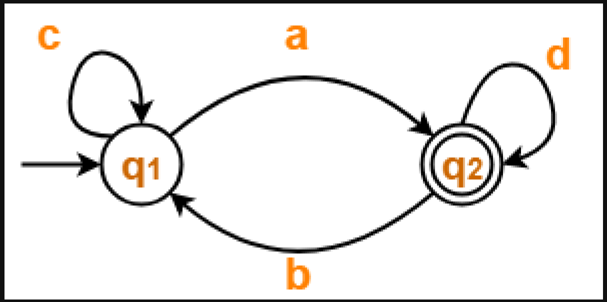

b
q1
a
q2
c
qf
d
∈
q5



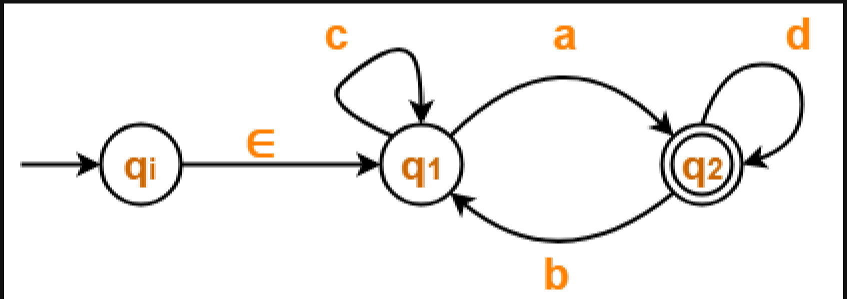


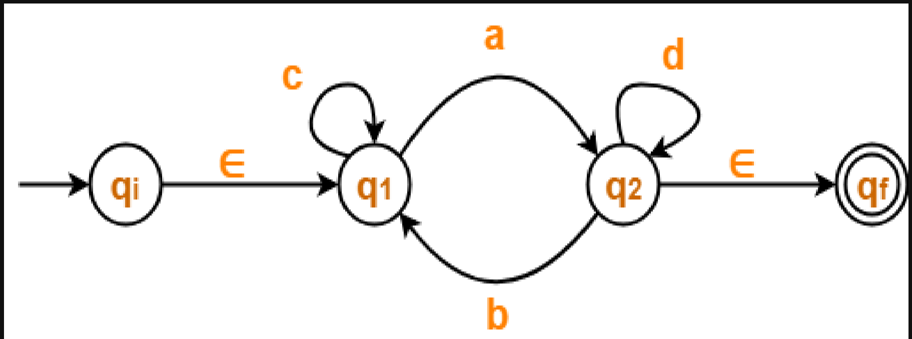
Regular Expression = a(b+c+d)

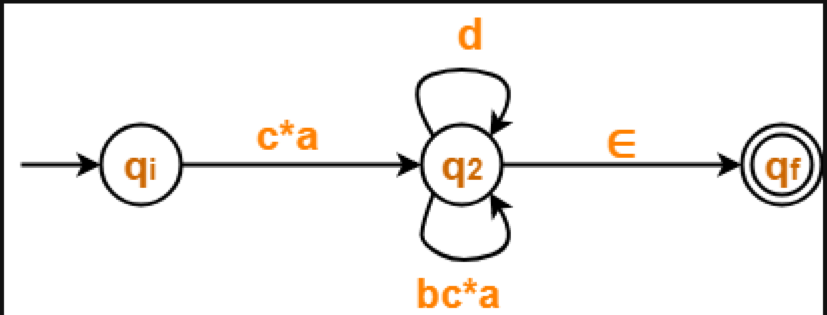
Problem 03

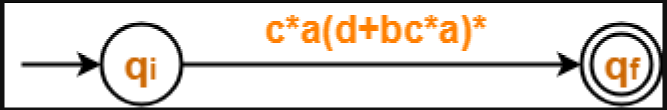


Regular Expression = c*a(d+bc*a)*

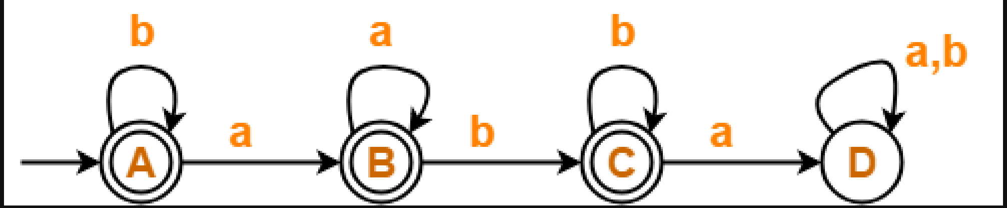


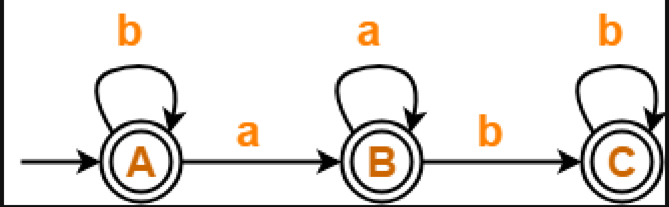


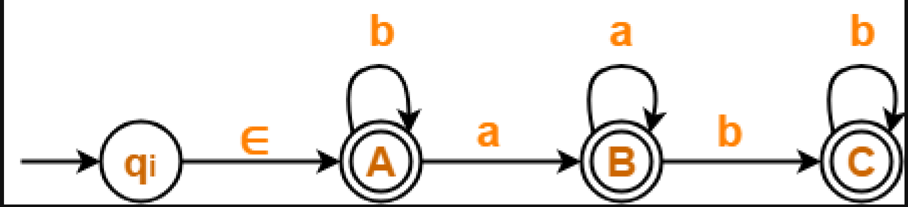


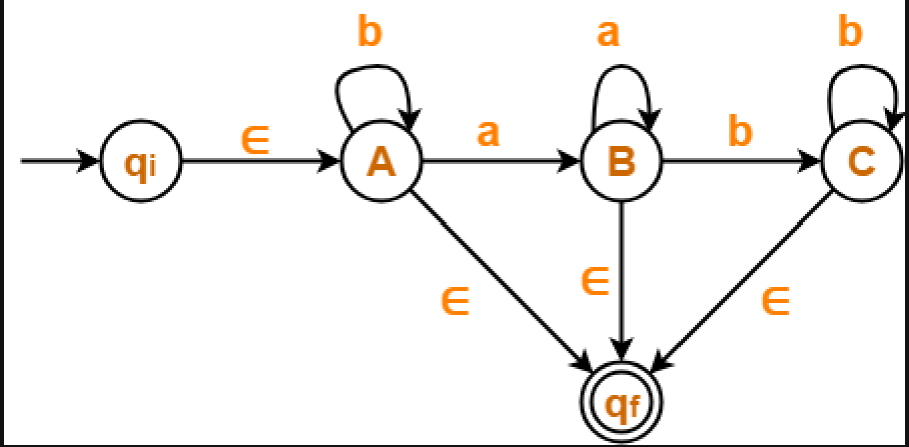


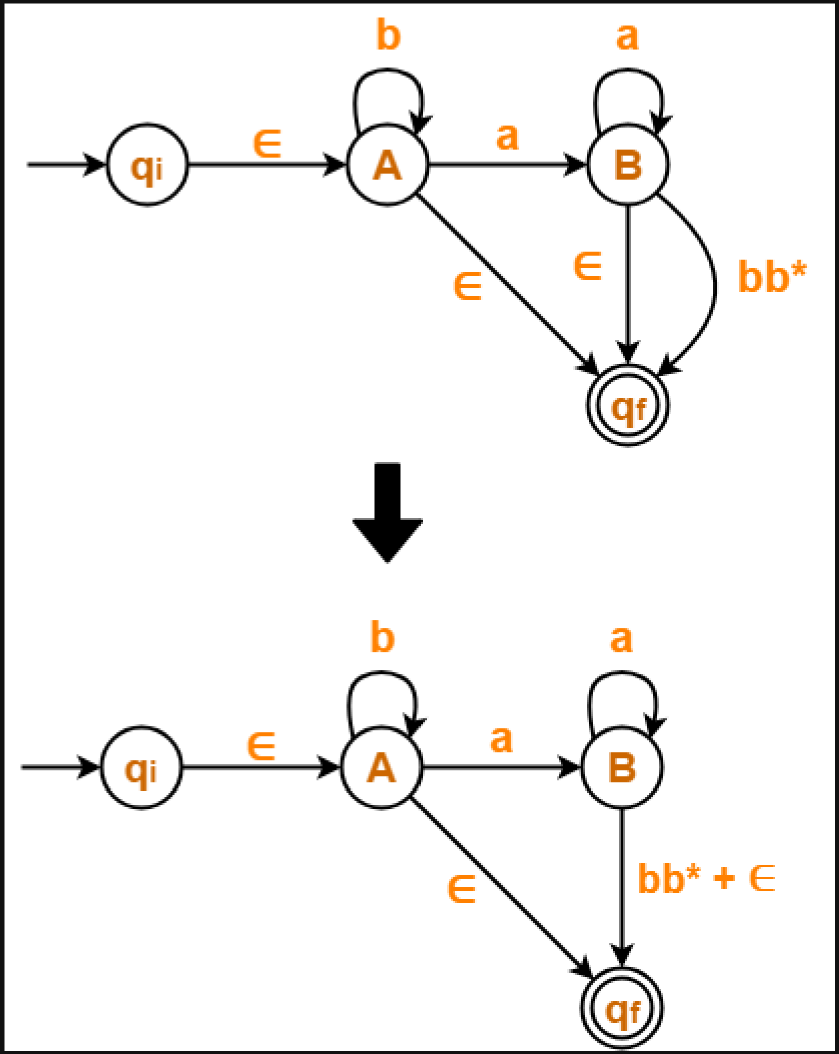
Problem 04

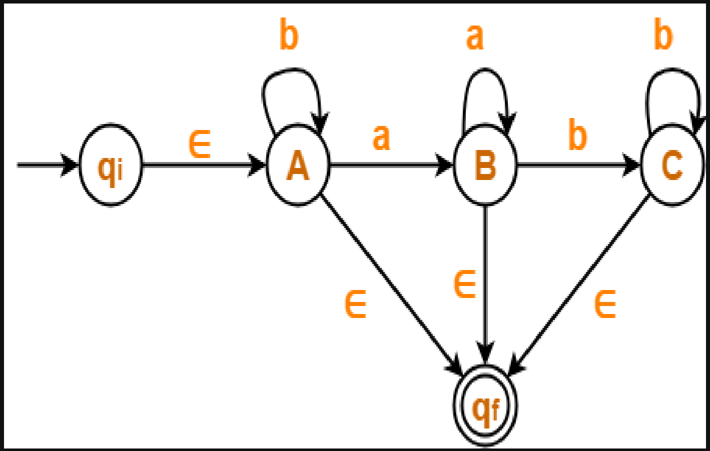


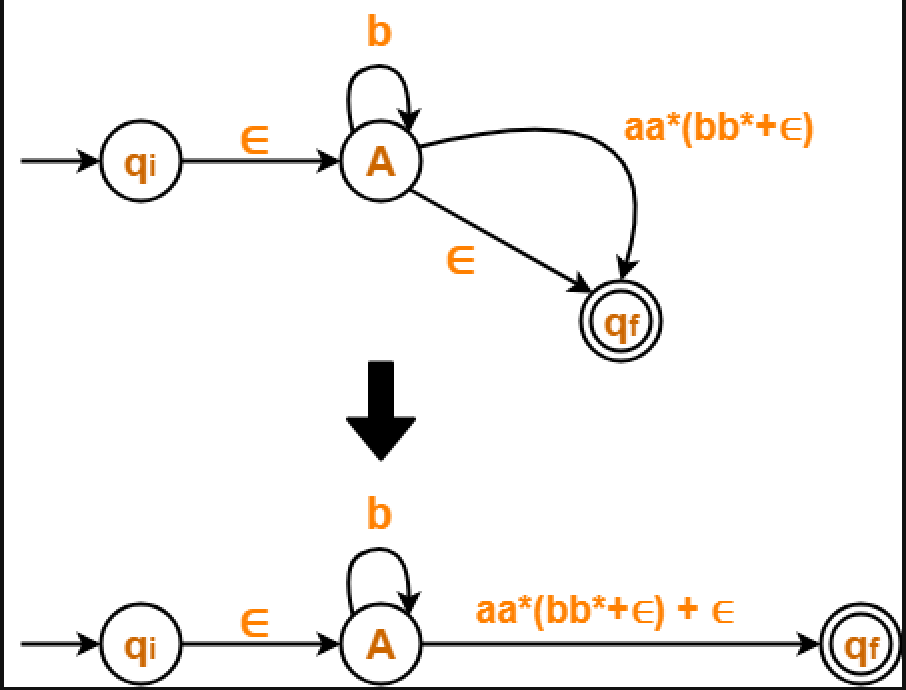






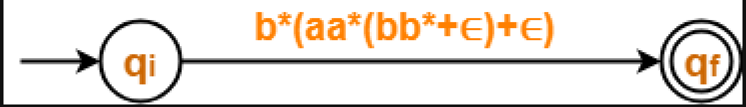

b
a
qi
∈
A
a
B
∈
∈
bb*
qf
b
a
qi
∈
A
a
B
∈
bb* + ∈
qf


b
a
b
qi
∈
A
a
B
b
C
∈
∈
∈
qf



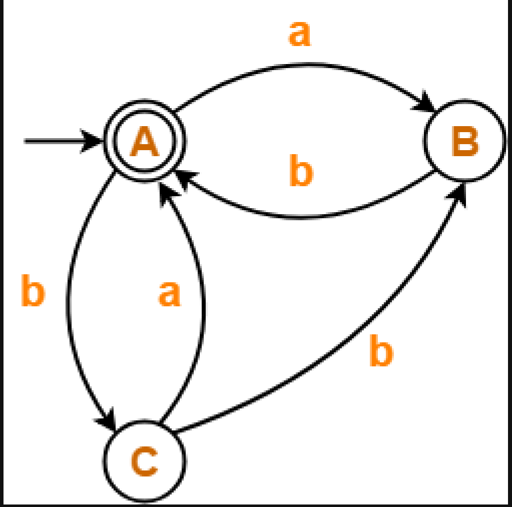
Regular Expression =
b*(aa*(bb*+∈)+∈)

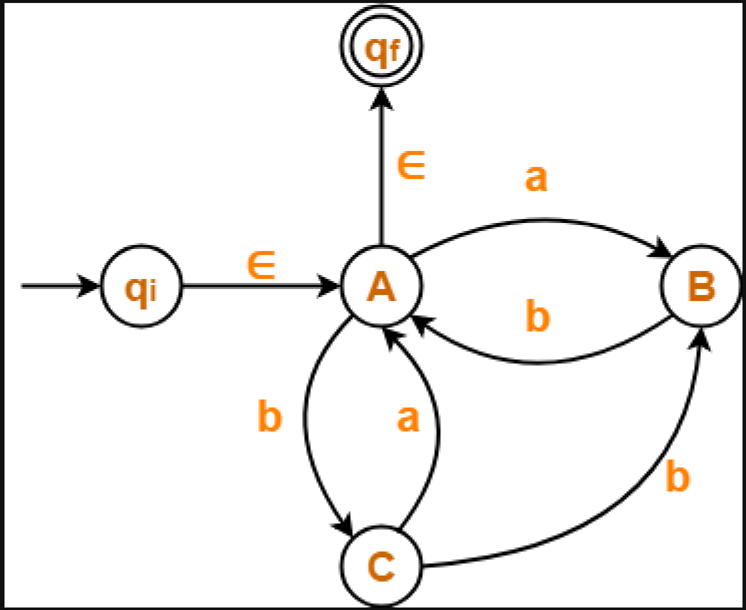
We know, bb* + ∈ = b*
So, we can also write-

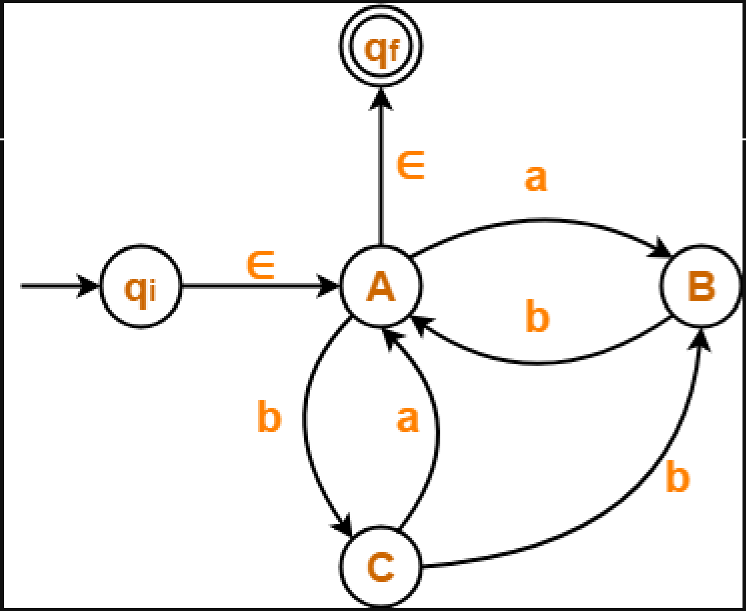


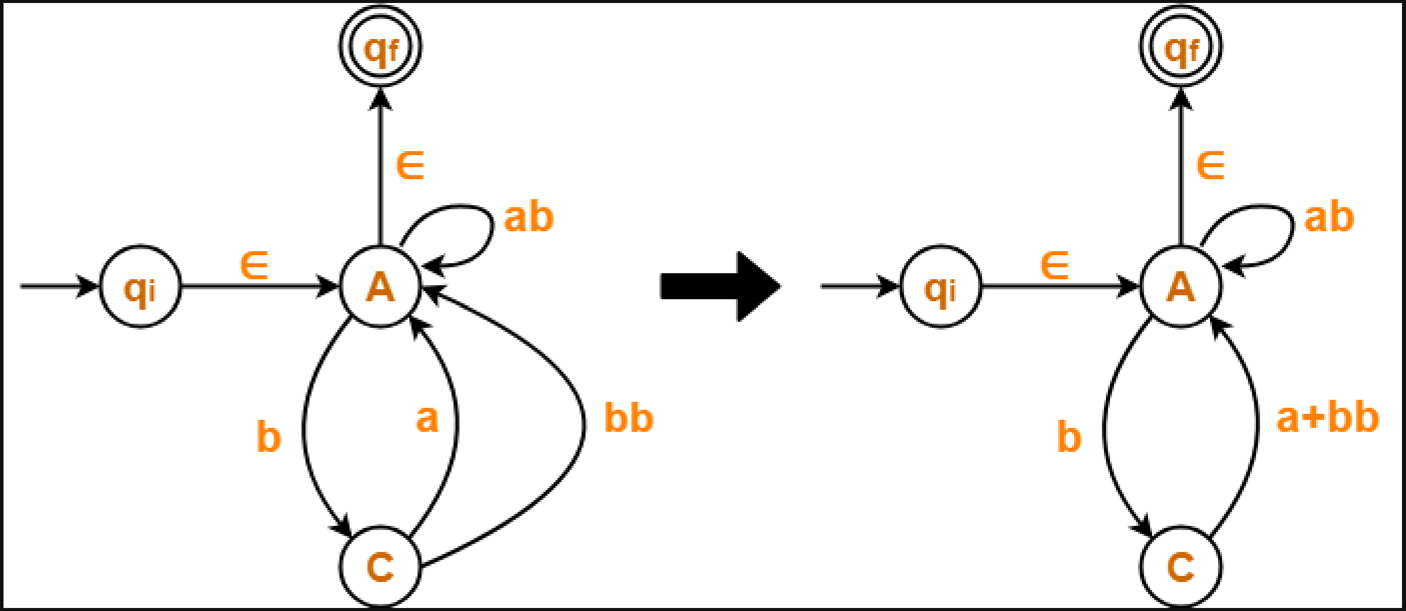
Regular Expression =
b*(aa*b*+∈)

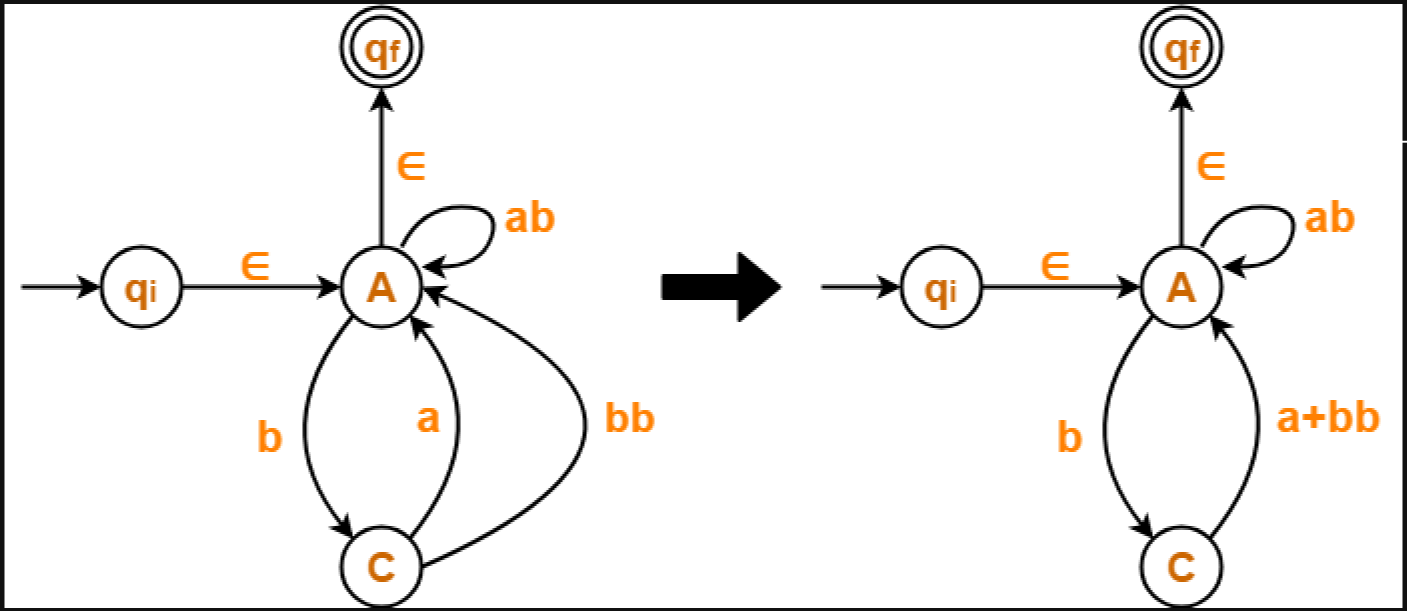
Problem 05

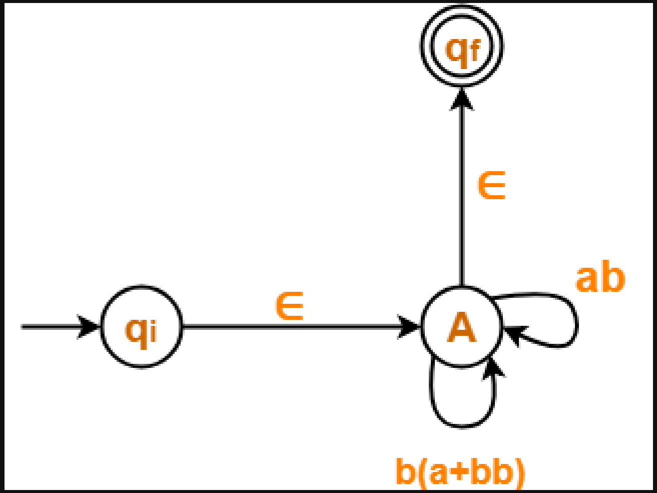




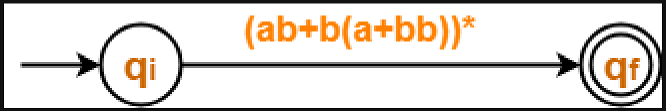

qf
∈
a
qi
∈
A
B
b
b
a
b
C


qf
∈
ab
qi
∈
A
b
a
bb
C
qf
∈
ab
qi
∈
A
b
a+bb
C

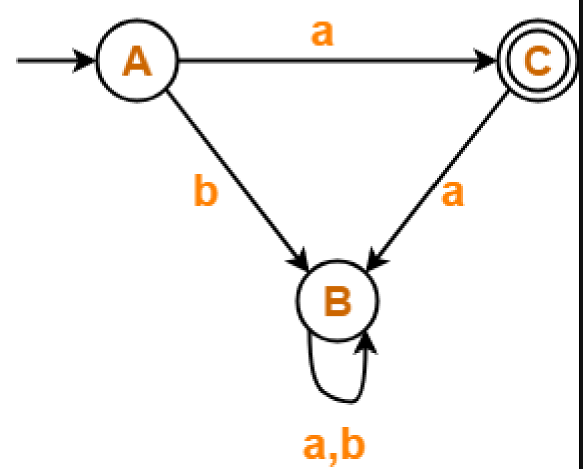




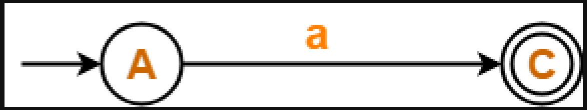
Regular Expression = (ab + b(a+bb))*



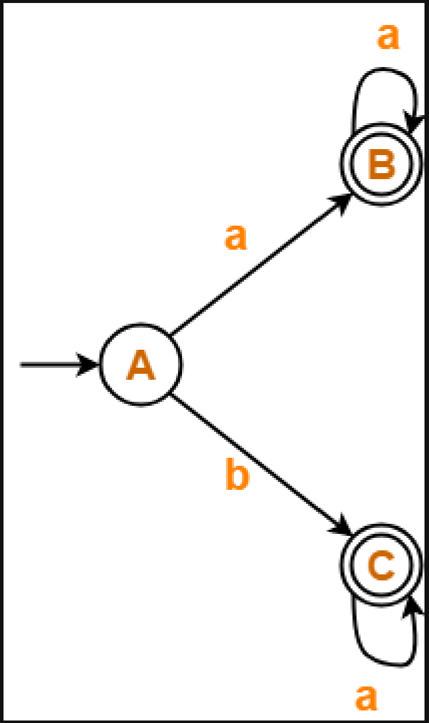
Problem 06

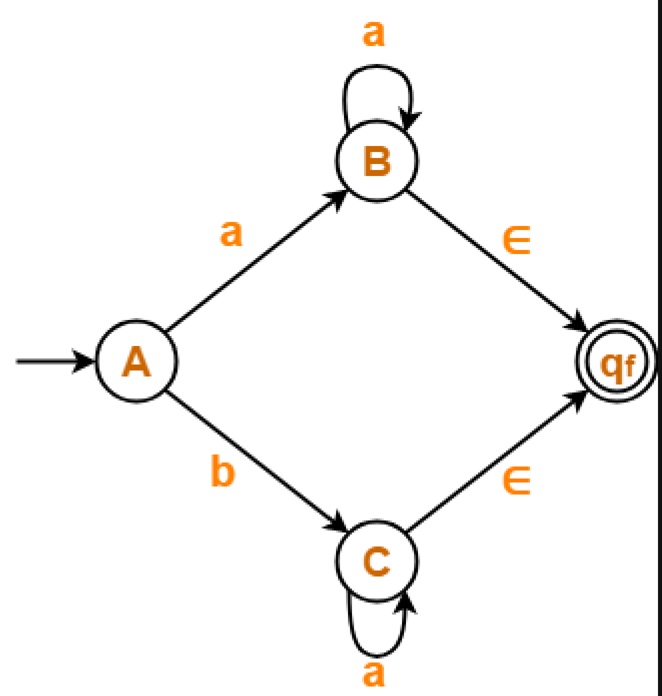


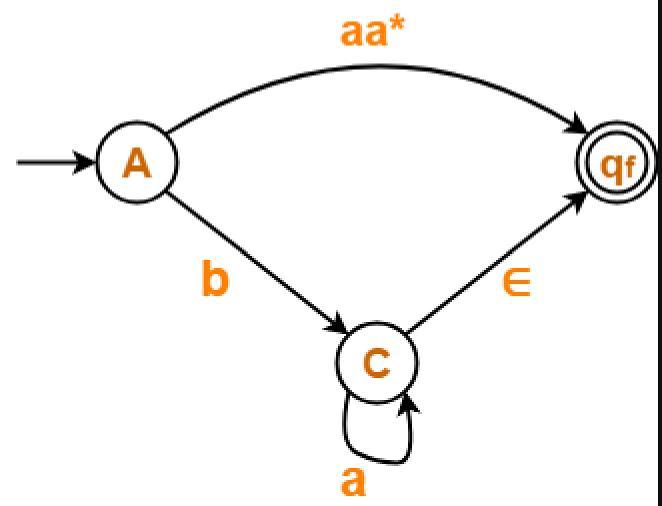
Regular Expression = a

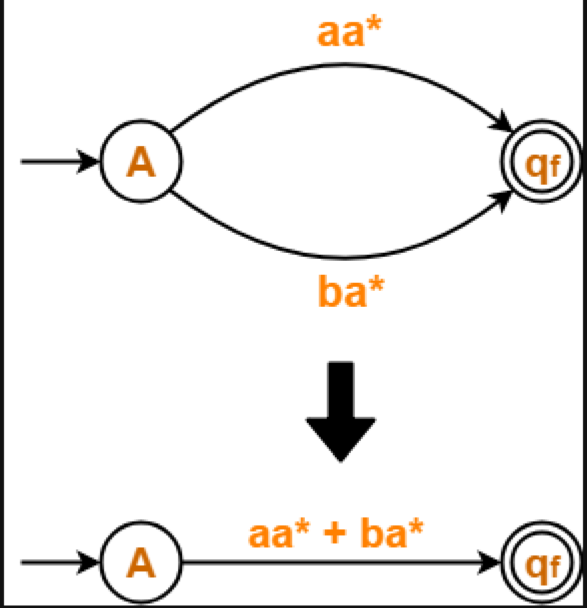


Problem 07

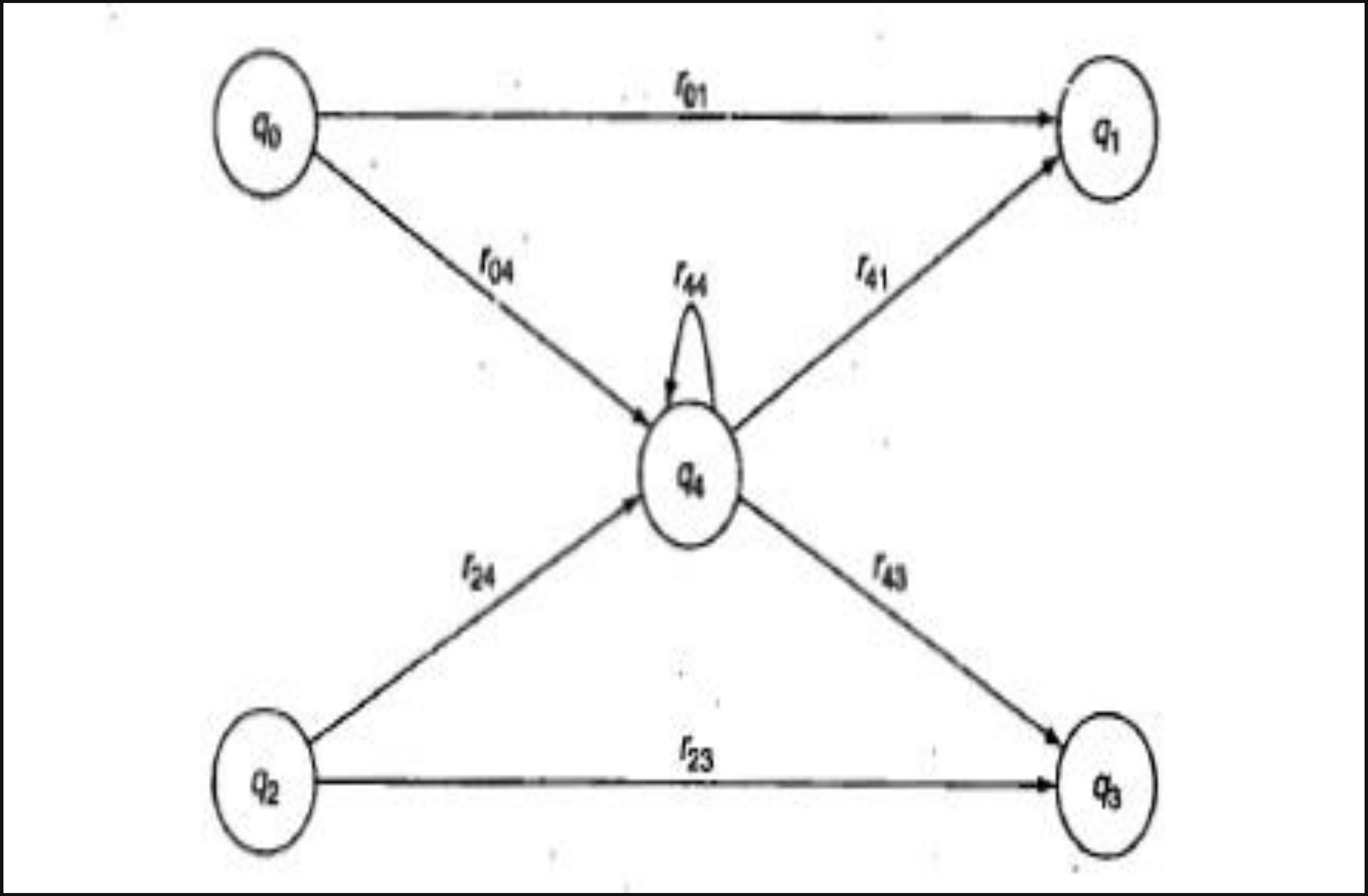


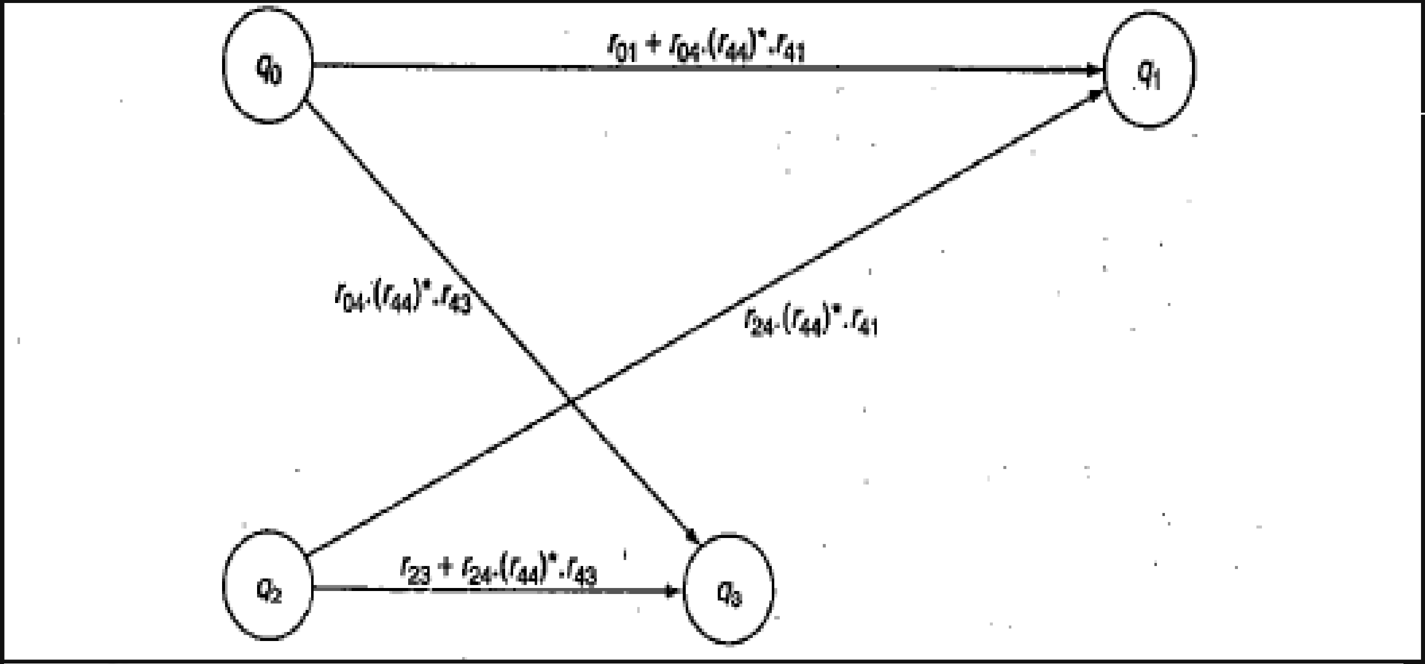


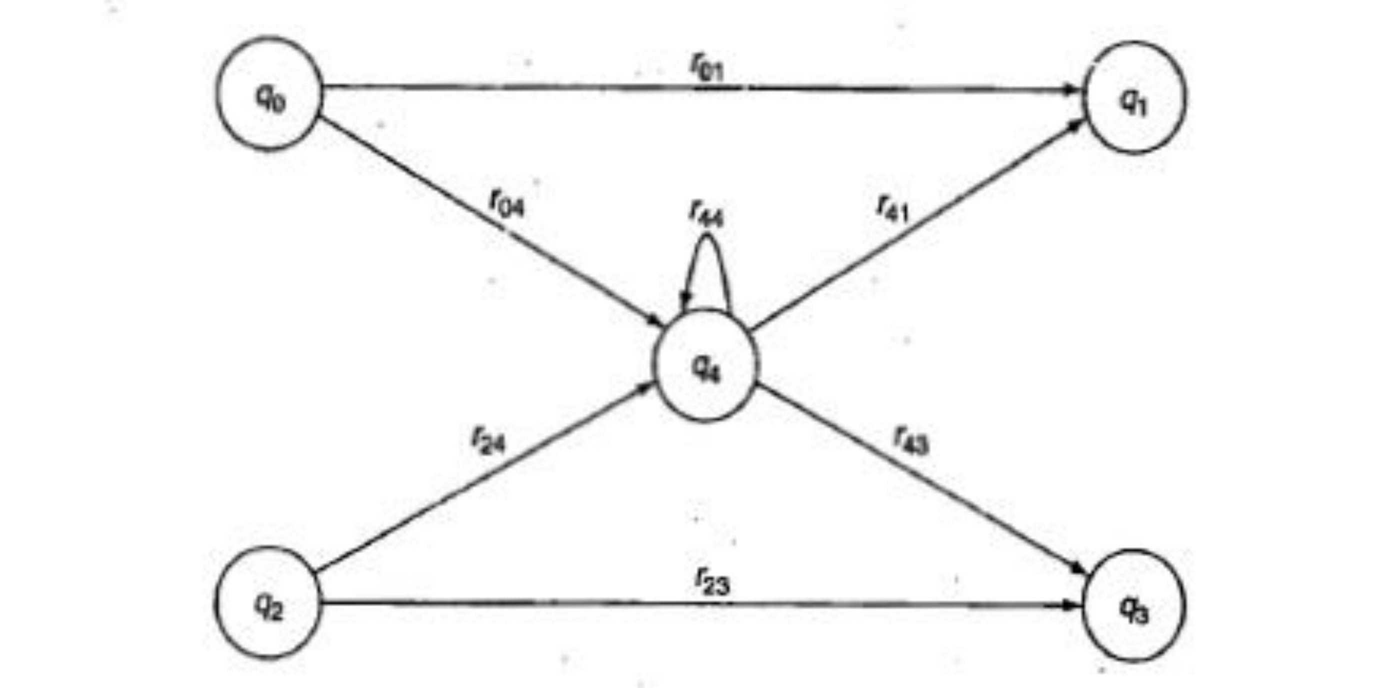


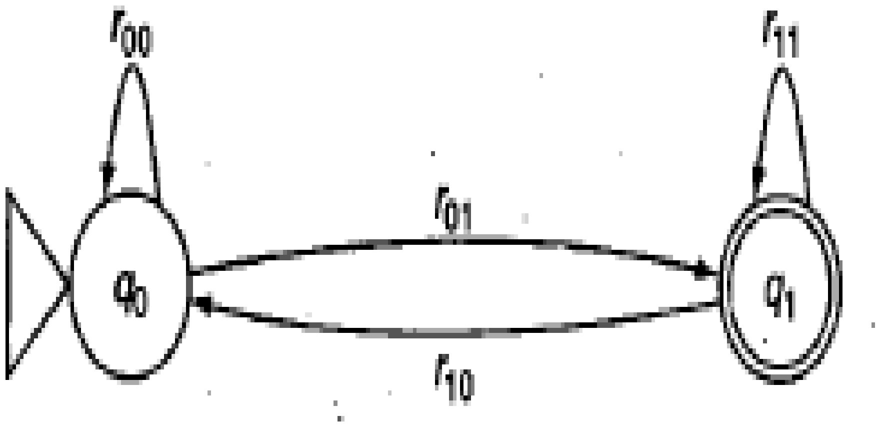


Regular Expression = aa* + ba*


q_0
r_{01}
q_1
r_{04}
r_{44}
r_{41}
q_4
r_{24}
r_{43}
q_2
r_{23}
q_3

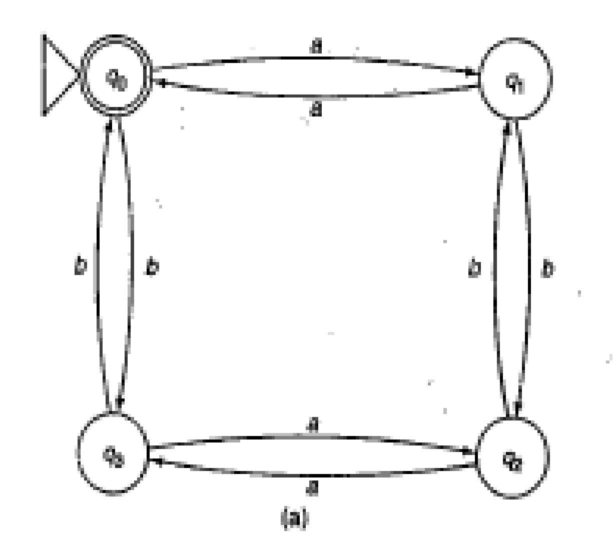

q_0
$r_{01} + r_{04}.(r_{44})^*.r_{41}$
q_1
$r_{04}.(r_{44})^*.r_{43}$
$r_{24}.(r_{44})^*.r_{41}$
q_2
$r_{23} + r_{24}.(r_{44})^*.r_{43}$
q_3


q_0
r_{01}
q_1
r_{04}
r_{44}
r_{41}
q_4
r_{24}
r_{43}
q_2
r_{23}
q_3

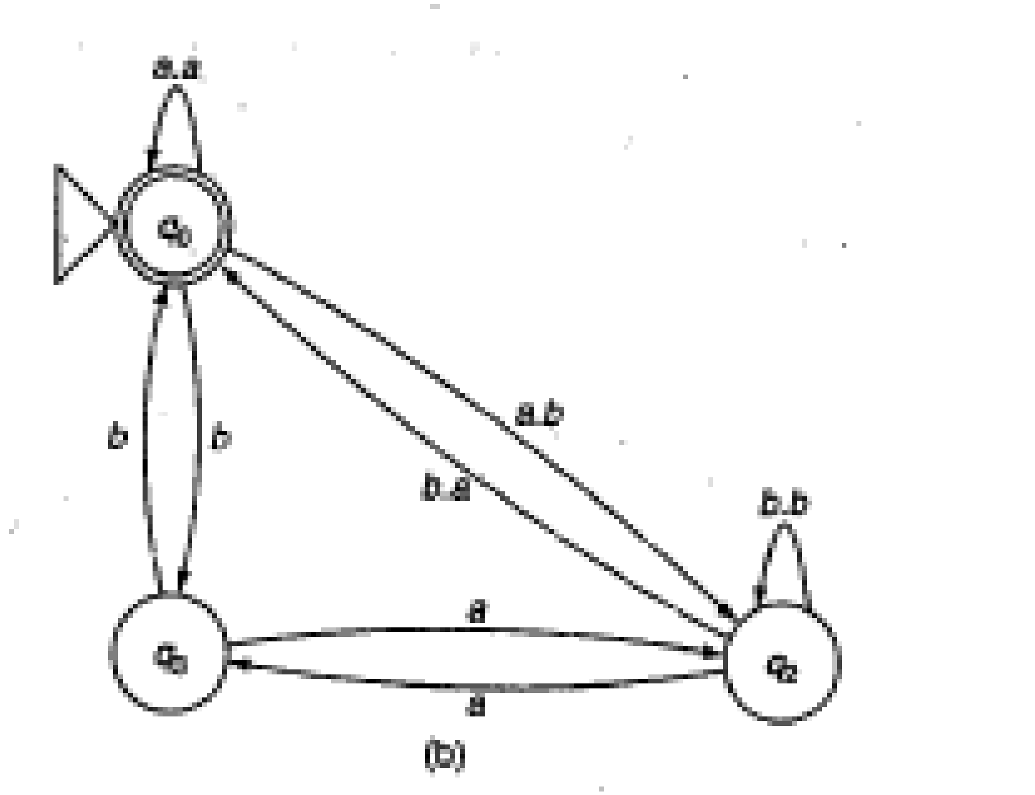


$$(r_{00} + r_{01} \cdot (r_{11})^* \cdot r_{10})^* \cdot r_{01} \cdot (r_{11} + r_{10} \cdot (r_{00})^* r_{01})^*$$

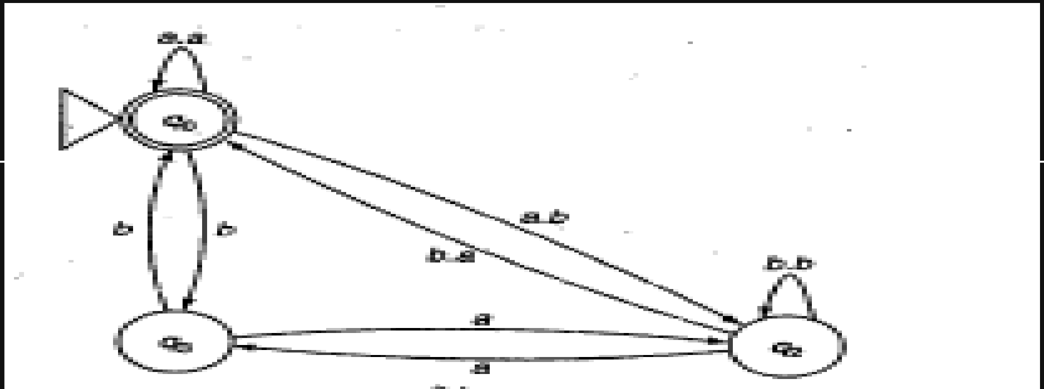
example



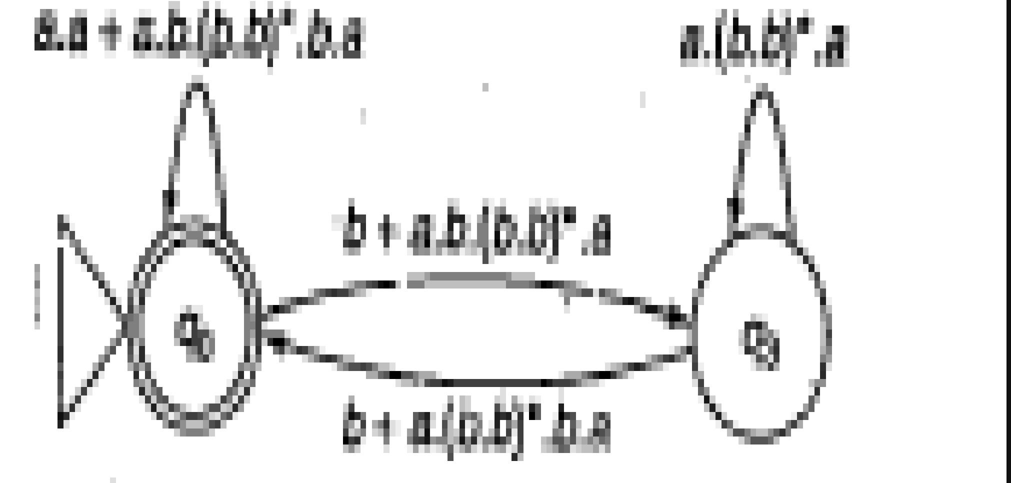
(a)



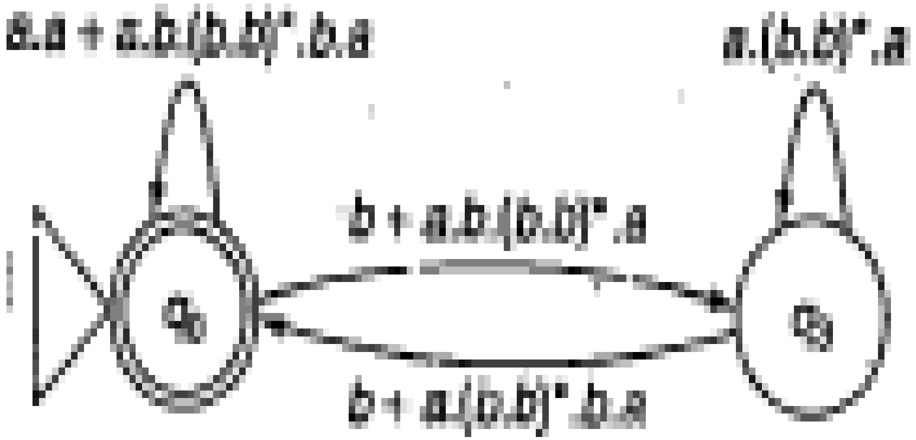
(b)



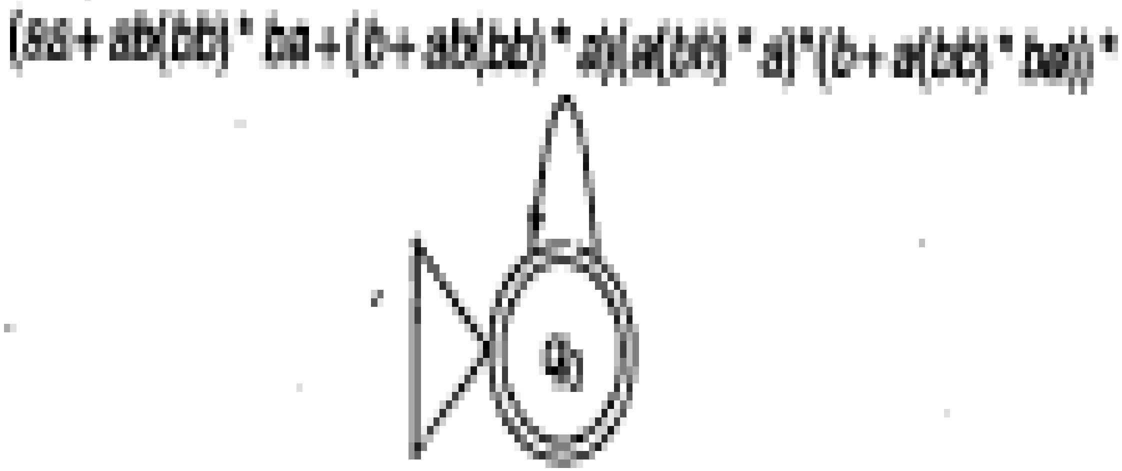
(b)



(c)


a.a + a.b.(b.b)*.b.a
a.(b.b)*.a
b + a.b.(b.b)*.a
q_0
q_1
b + a.(b.b)*.b.a

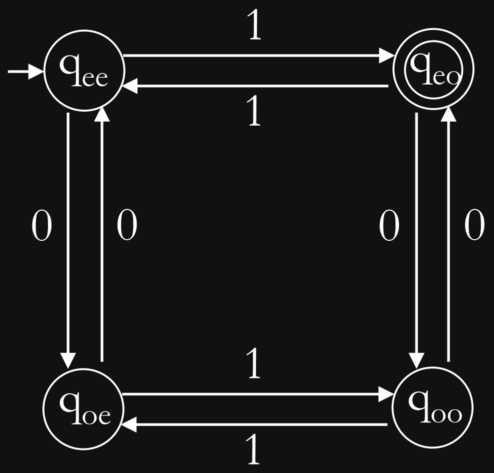
(c)


(aa+ ab(bb) * ba+(b+ ab(bb) * a)(a(bb) * a)*(b+ a(bb) * ba)) *
q_0

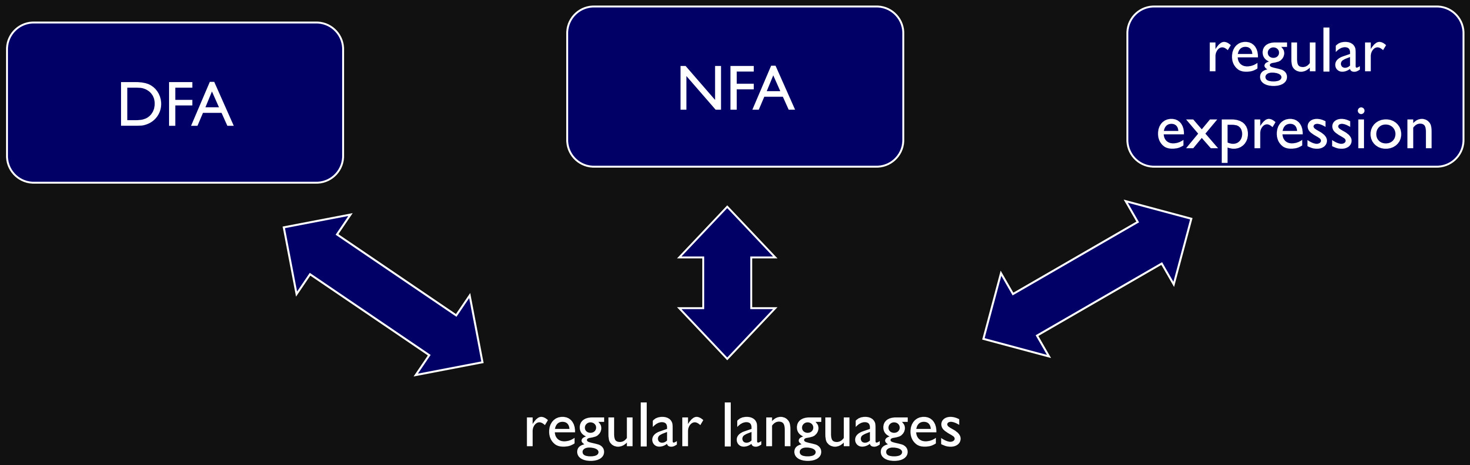
(d)

Exercise

- Even number of 0s and odd number of 1s?



What you need to know



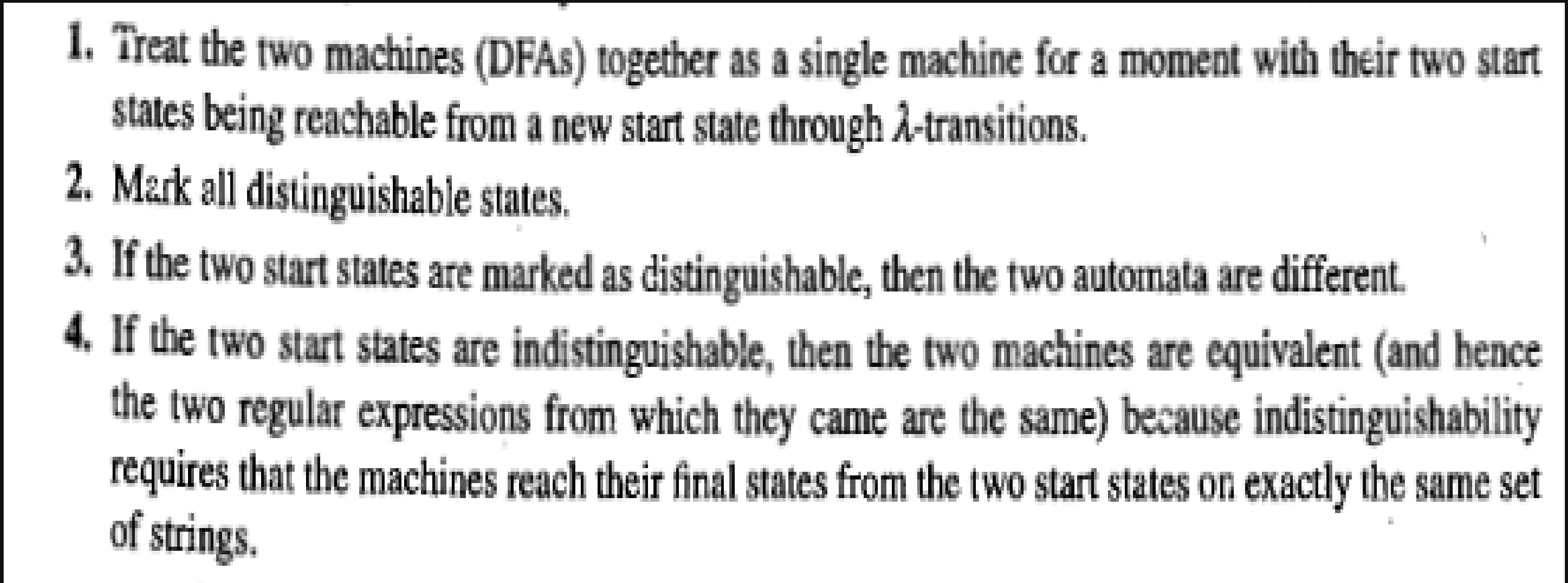
DFA

NFA

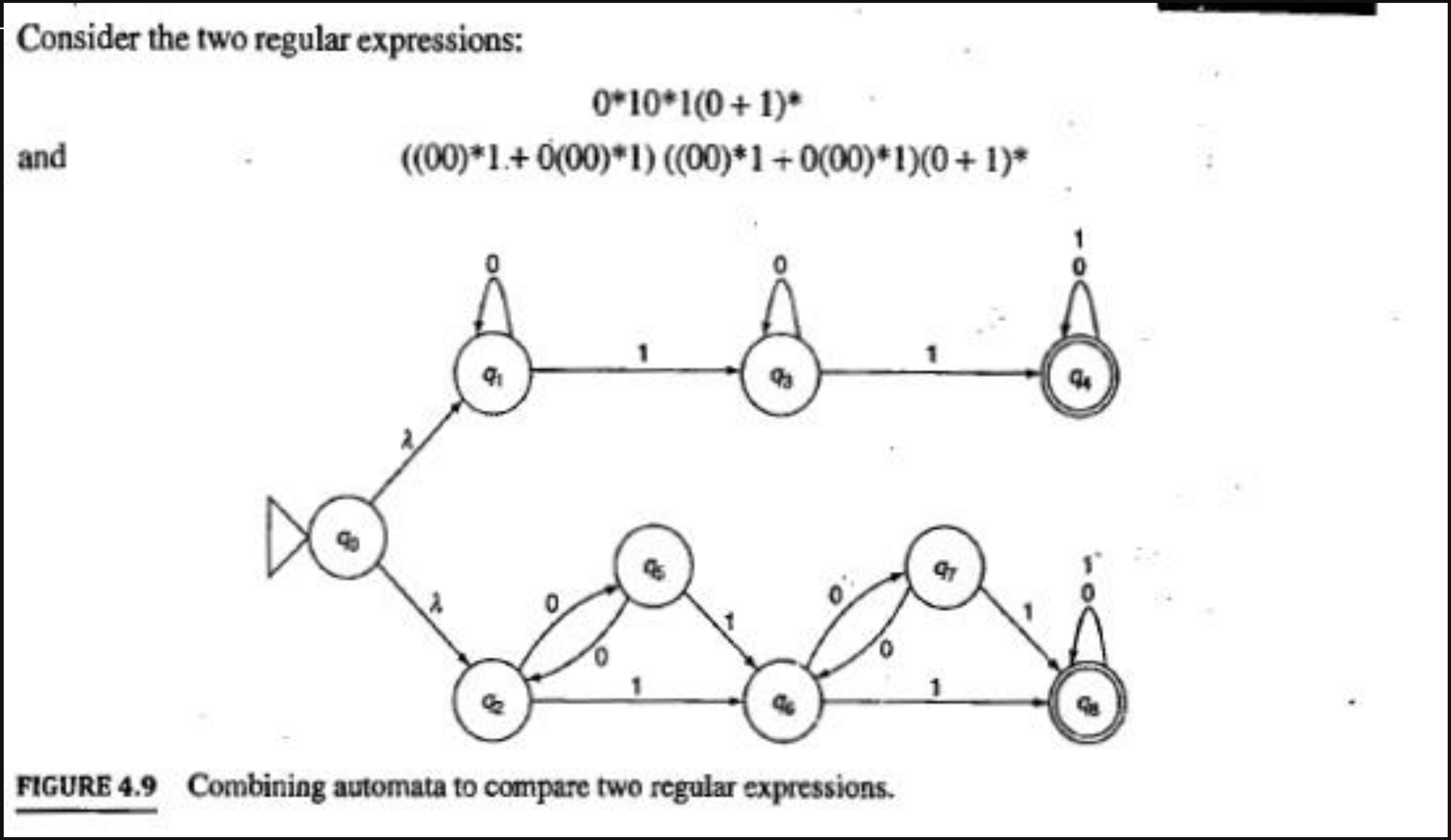
regular expression

regular languages

EQUIVALENCE OF REGULAR EXPRESSIONS



1. Treat the two machines (DFAs) together as a single machine for a moment with their two start states being reachable from a new start state through λ-transitions.
2. Mark all distinguishable states.
3. If the two start states are marked as distinguishable, then the two automata are different.
4. If the two start states are indistinguishable, then the two machines are equivalent (and hence the two regular expressions from which they came are the same) because indistinguishability requires that the machines reach their final states from the two start states on exactly the same set of strings.



Consider the two regular expressions:

$$0^*10^*1(0+1)^*$$

and

$$((00)^*1+0(00)^*1)\,((00)^*1+0(00)^*1)(0+1)^*$$

FIGURE 4.9 Combining automata to compare two regular expressions.

BY IGNORING NEW START STATE q0

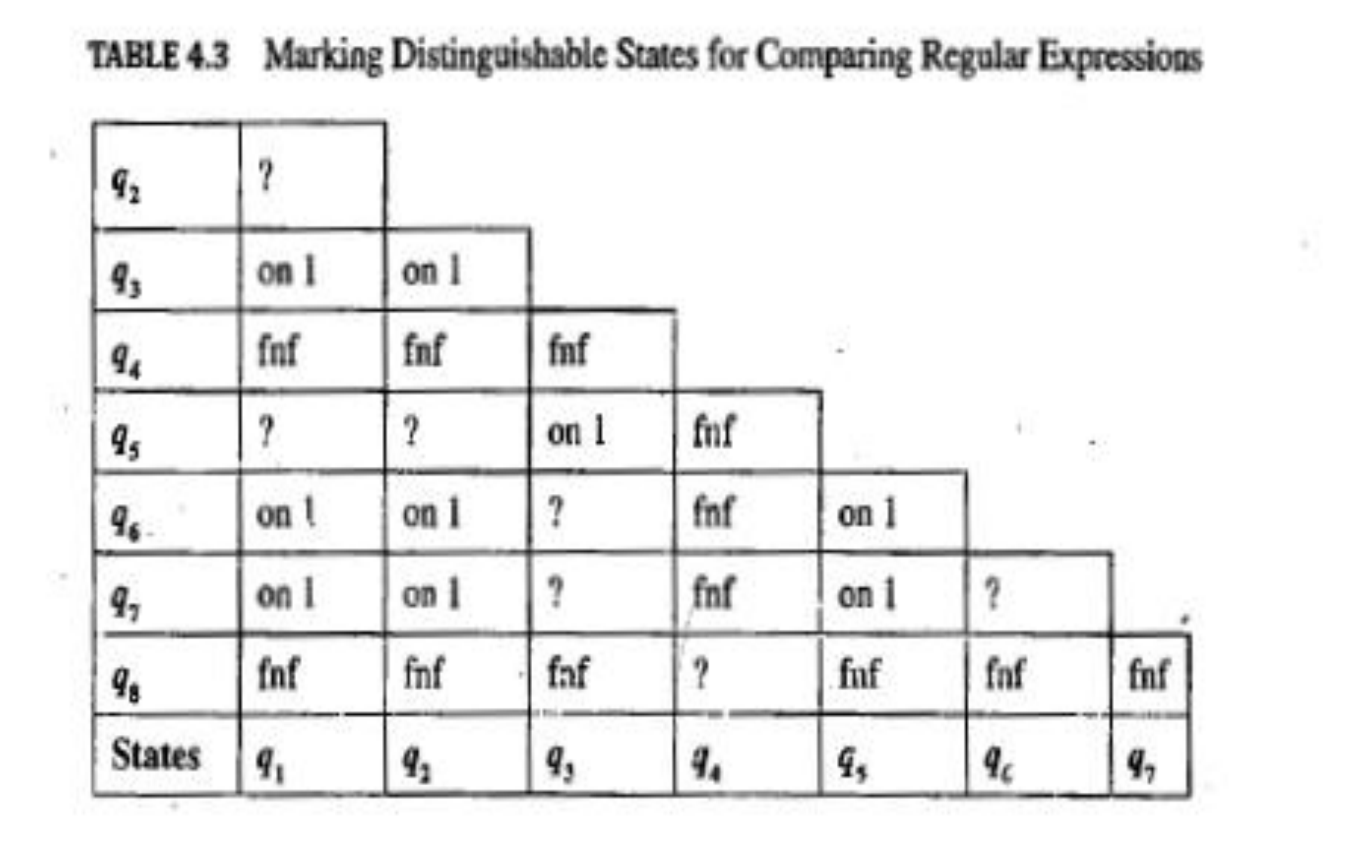
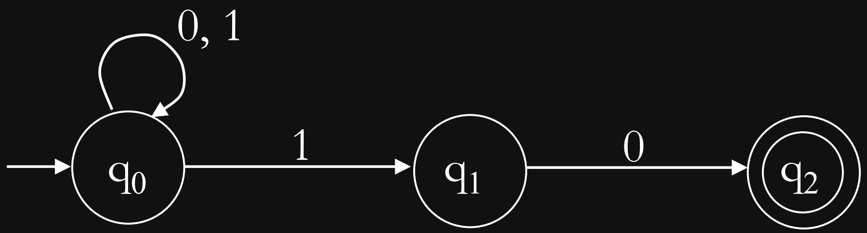


TABLE 4.3 Marking Distinguishable States for Comparing Regular Expressions

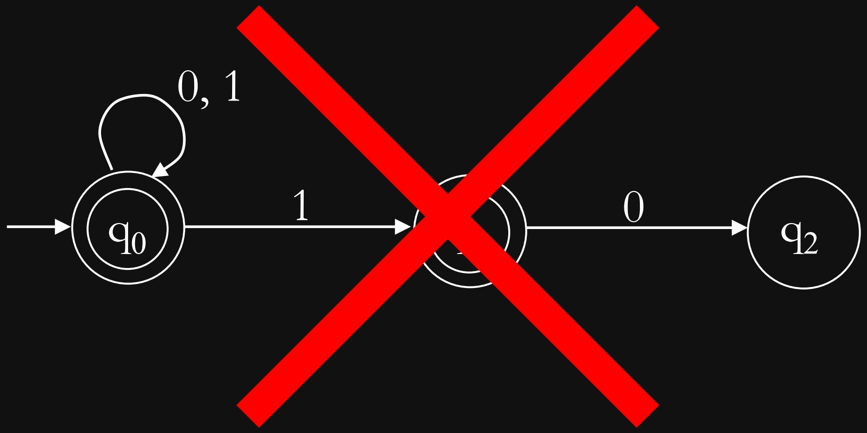
q_2	?						
q_3	on 1	on 1					
q_4	fnf	fnf	fnf				
q_5	?	?	on 1	fnf			
q_6	on 1	on 1	?	fnf	on 1		
q_7	on 1	on 1	?	fnf	on 1	?	
q_8	fnf	fnf	fnf	?	fnf	fnf	fnf
States	q_1	q_2	q_3	q_4	q_5	q_6	q_7

Some pitfalls

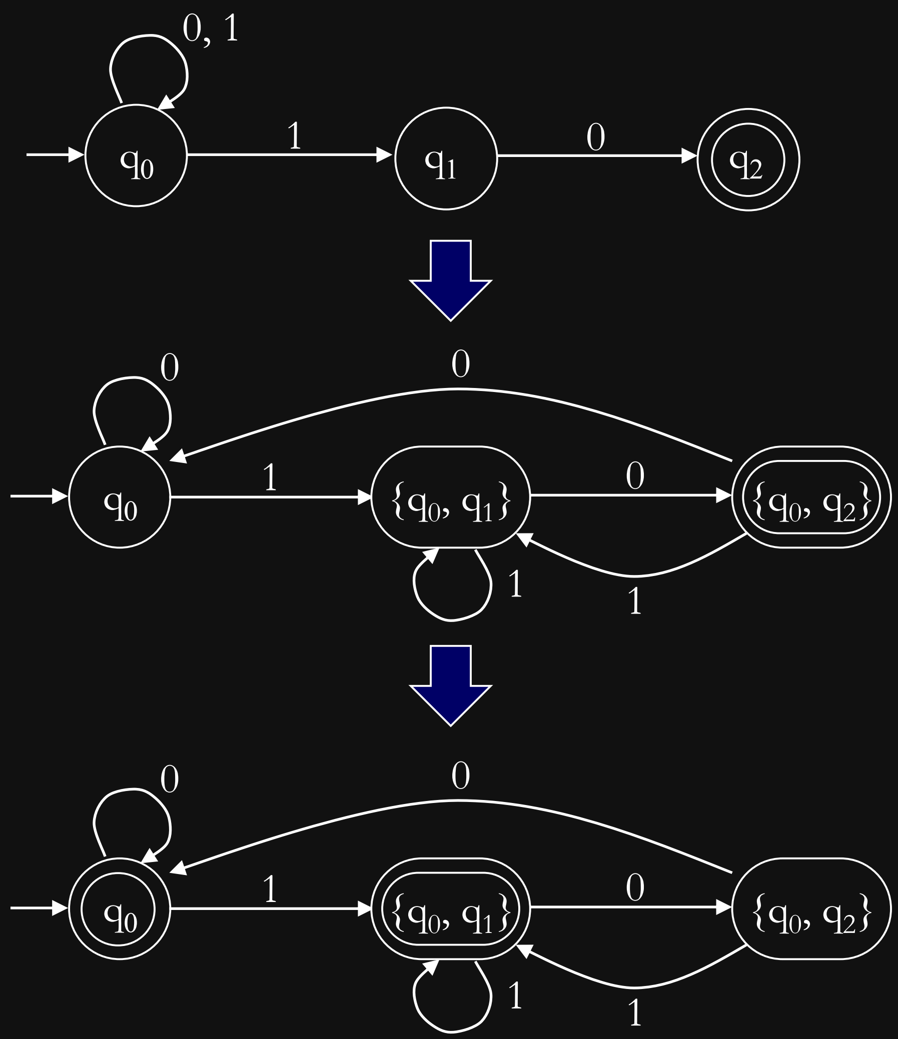
- NFA for all strings that end in 10:



- NFA for all strings that do not end in 10:



How to do it right



NFA for ends in 10

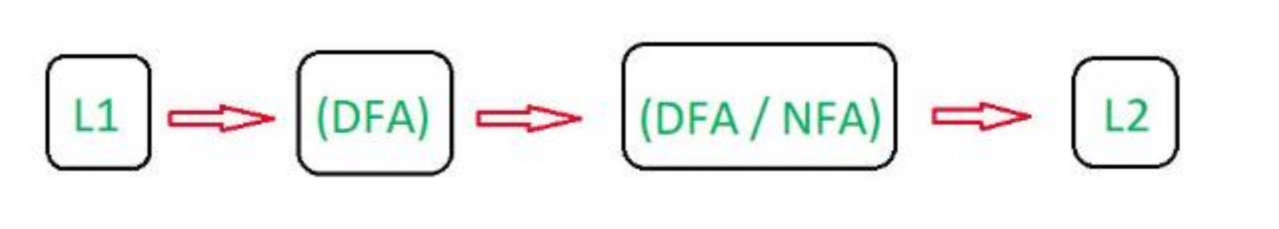
DFA for ends in 10

DFA for does not end in 10

Reversal process in DFA

- **Reversal :** Reversing a language means reversing the each string in the language.
- **Steps to Reversal:**
  - Draw the states as it is.
  - Make final state as initial state and initial states as a final state
  - Reverse the edges
  - Loop will remain same
  - Remove the inappropriate transition state.

- **Note :**
  1. Not every reversal of DFA lead to DFA
  2. Reversal process goes like this:



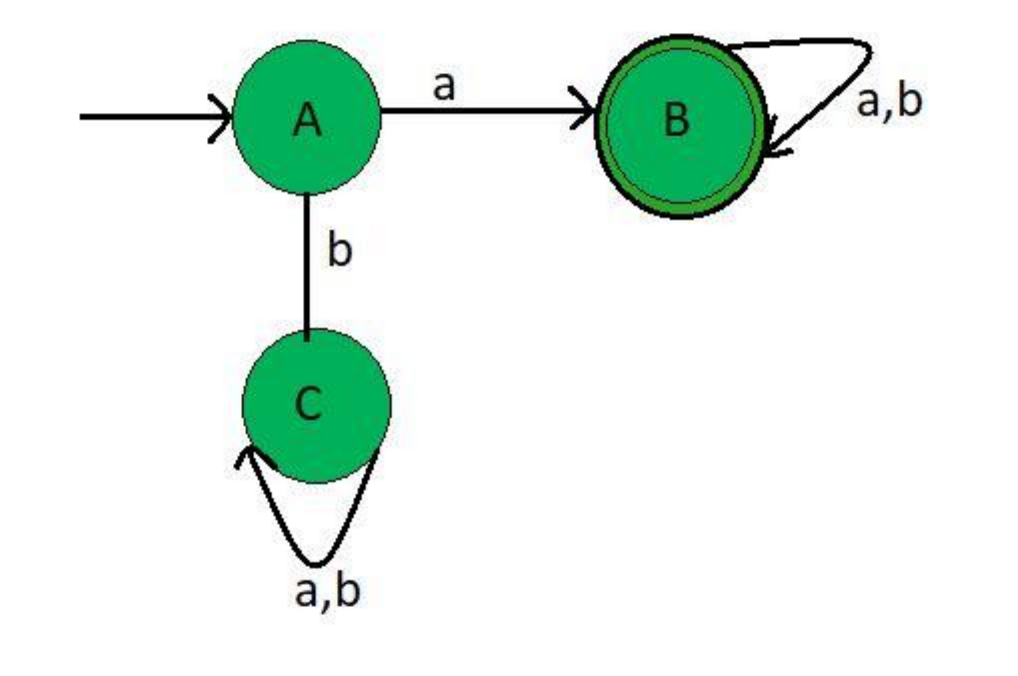
Example-1:

Designing a DFA for the set of string over {a, b} such that string of the language start symbol ‘a’.

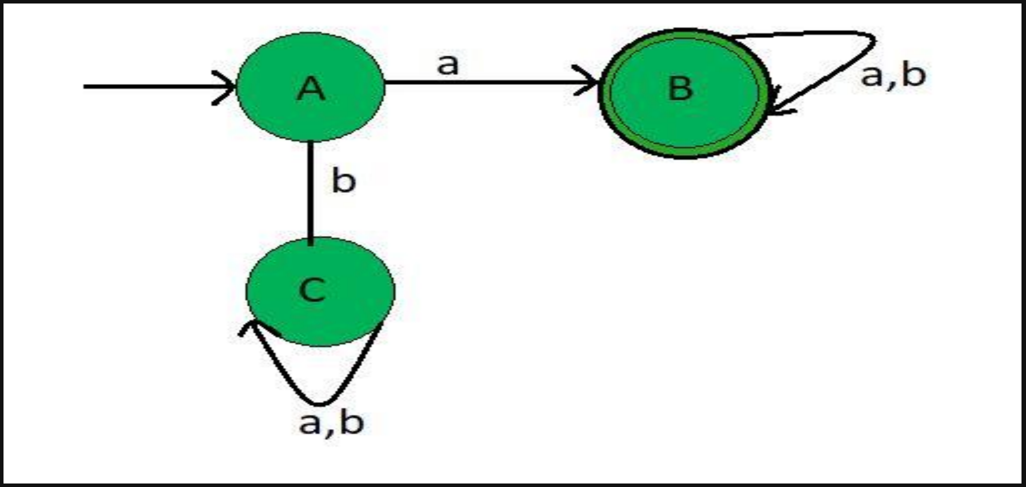
The desired language will be formed:

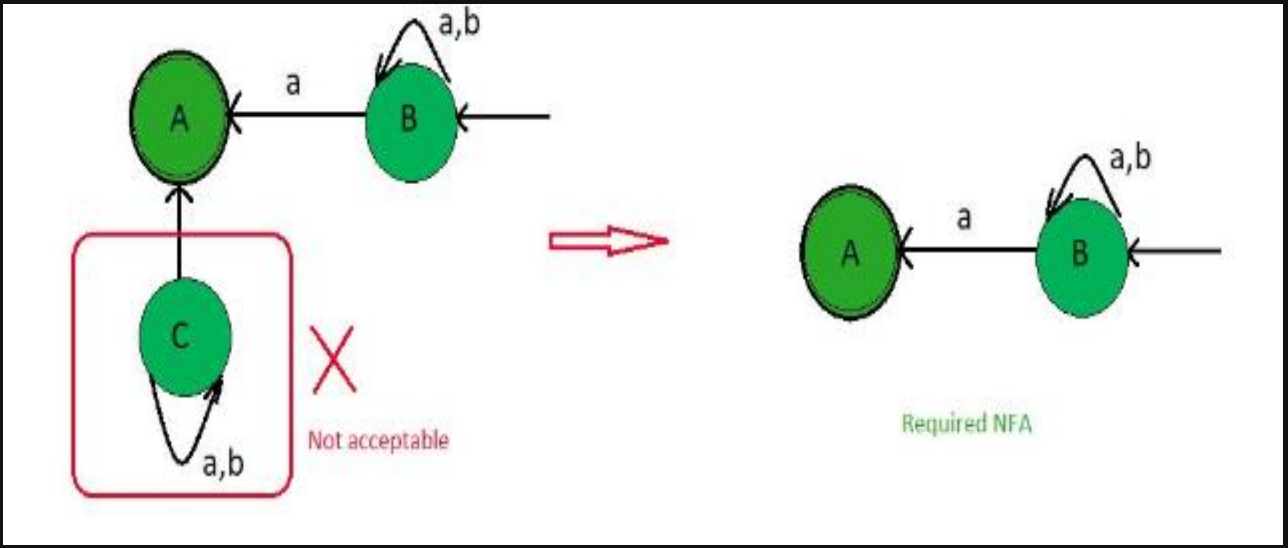
L1 = {a, aa, ab, aab, aaabb, aabab, ...}

- In L1, each string have starting element a. State Transition Diagram for the language L1:



State Transition Diagram of L2 (reverse of L1):




a,b
a
A
B
C
a,b
Not acceptable
a,b
a
A
B
Required NFA