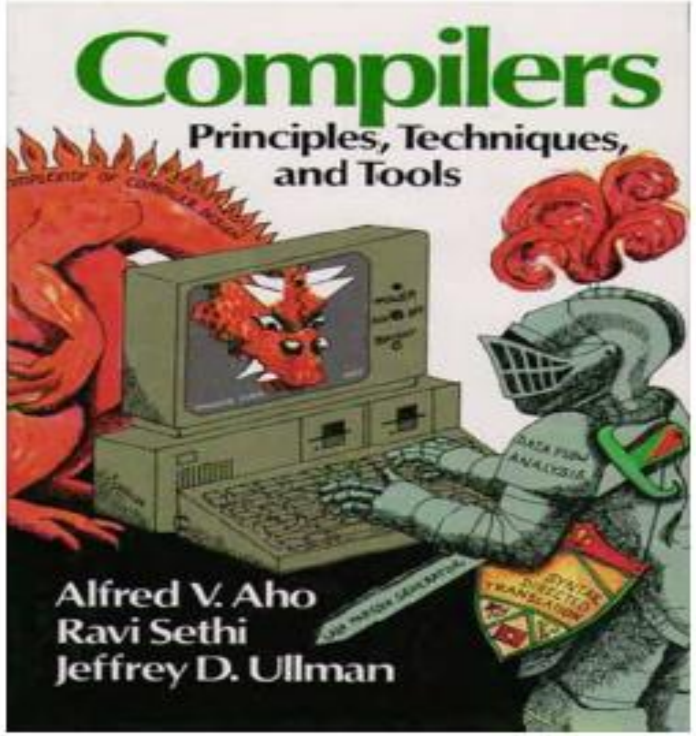
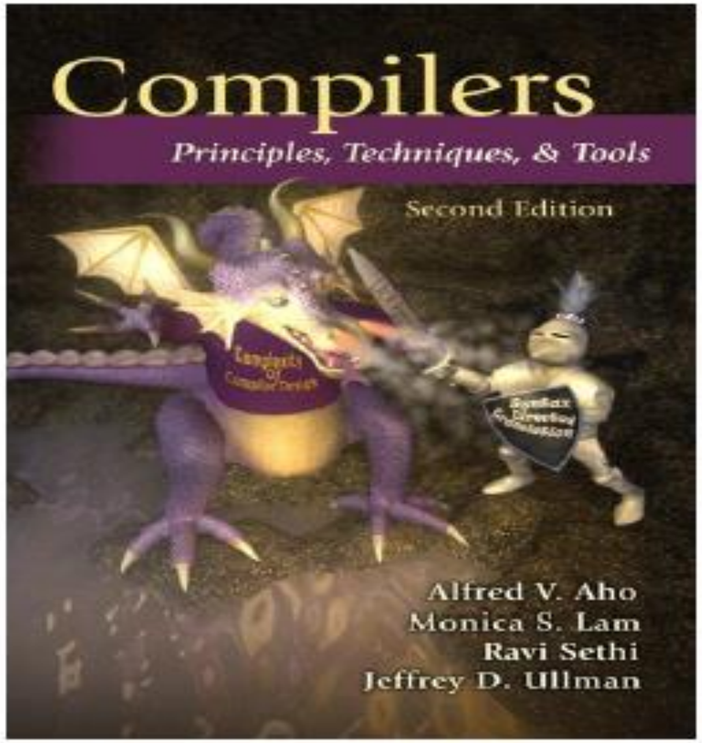
Compiler Design
Unit I – Part 1

1. Finite automation – deterministic
2. *Finite automation - non deterministic*
3. *Conversion of NFA to DFA*
4. Minimization of DFA

Reference Book:

- Compilers: Principles, Techniques, and Tools, 2/E.
  - **Alfred V. Aho**, *Columbia University*
  - **Ravi Sethi**, *Avaya Labs*
  - **Jeffrey D. Ullman**, *Stanford University*





Commonly Known as
Dragon Book

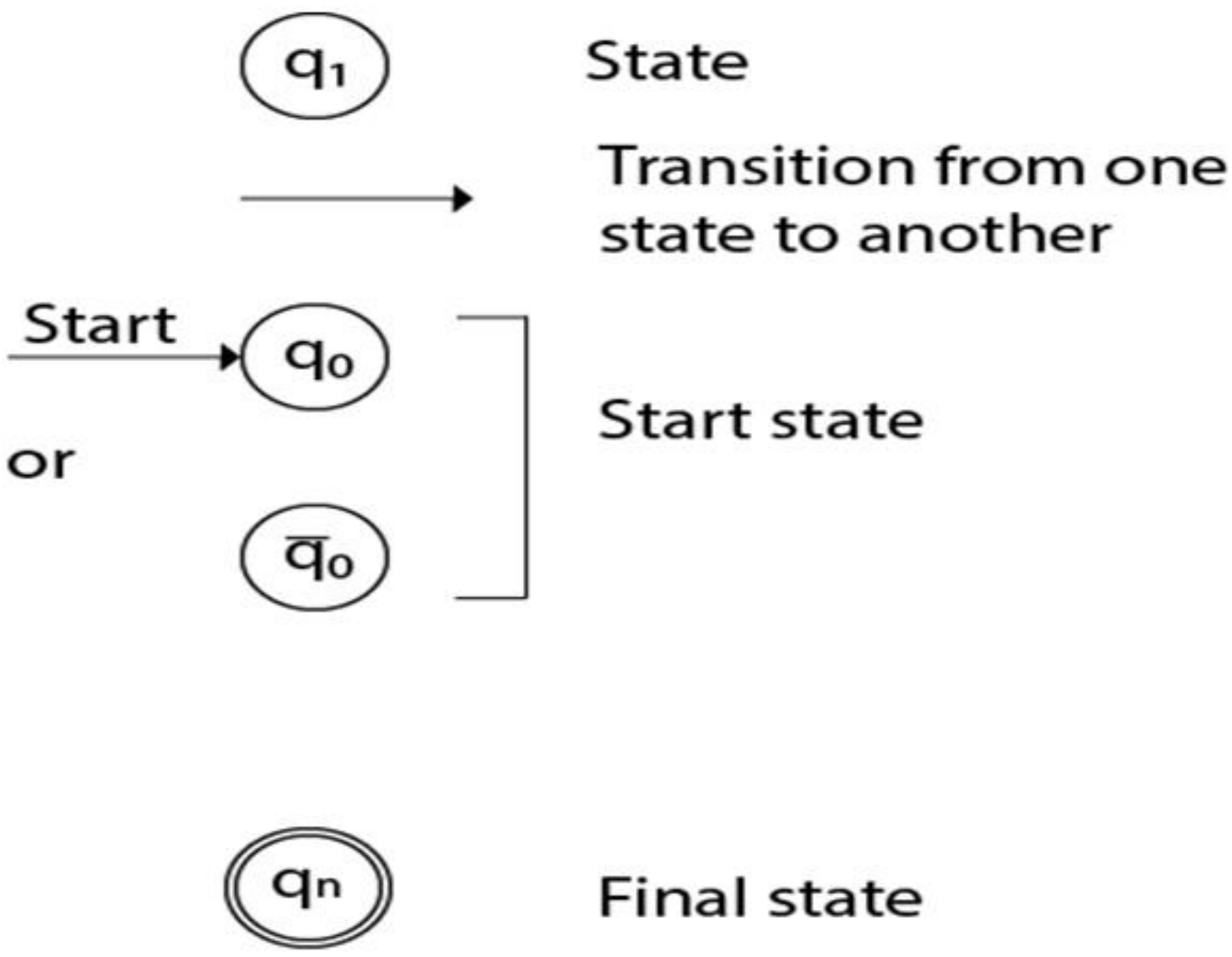
2nd Edition with Monica S. Lam, *Stanford University*

Finite Automata

- Finite automata are used to recognize patterns.
- It takes the string of symbol as input and changes its state accordingly. When the desired symbol is found, then the transition occurs.
- At the time of transition, the automata can either move to the next state or stay in the same state.
- Finite automata have two states, **Accept state** or **Reject state**. When the input string is processed successfully, and the automata reached its final state, then it will accept.

A finite automaton is a collection of 5-tuple (Q, ∑, δ, q0, F), where:

- Q: finite set of states
- ∑: finite set of the input symbol
- q0: initial state
- F: **final** state
- δ: Transition function



Finite Automata Model:

- Finite automata can be represented by input tape and finite control.
- **Input tape:** It is a linear tape having some number of cells. Each input symbol is placed in each cell.
- **Finite control:** The finite control decides the next state on receiving particular input from input tape. The tape reader reads the cells one by one from left to right, and at a time only one input symbol is read.

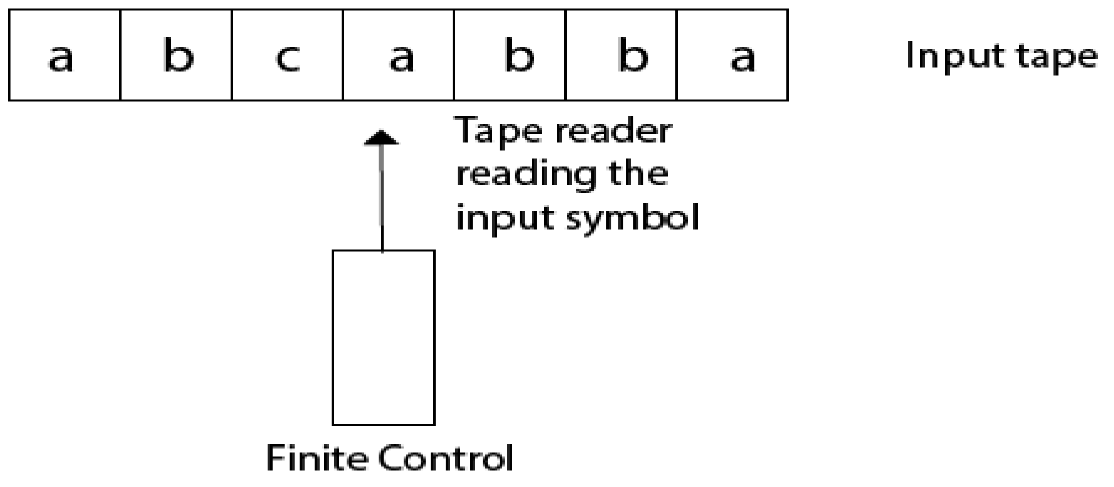


Fig :- Finite automata model

There are two types of finite automata:

- DFA(deterministic finite automata)
- NFA(non-deterministic finite automata)

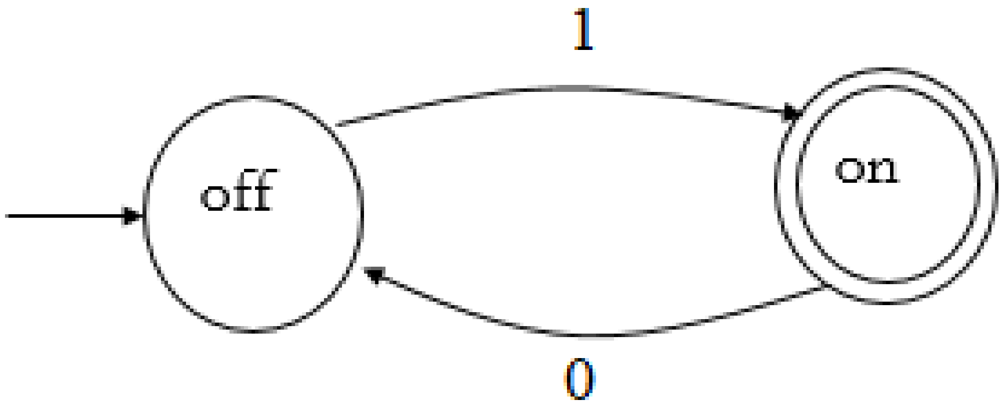
1. DFA

- DFA refers to deterministic finite automata. Deterministic refers to the uniqueness of the computation. In the DFA, the machine goes to one state only for a particular input character. DFA does not accept the null move.

2. NFA

- NFA stands for non-deterministic finite automata. It is used to transmit any number of states for a particular input. It can accept the null move.

- Every DFA is NFA, but NFA is not DFA.
- There can be multiple final states in both NFA and DFA.
- DFA is used in Lexical Analysis in Compiler.
- NFA is more of a theoretical concept.



Deterministic Finite Automata (DFA)

- The finite automata are called deterministic finite automata if the machine is read an input string one symbol at a time.
- In DFA, there is only one path for specific input from the current state to the next state.
- DFA does not accept the null move, i.e., the DFA cannot change state without any input character.
- DFA can contain multiple final states. It is used in Lexical Analysis in Compiler.

A DFA is a collection of 5-tuples same as we described in the definition of FA.

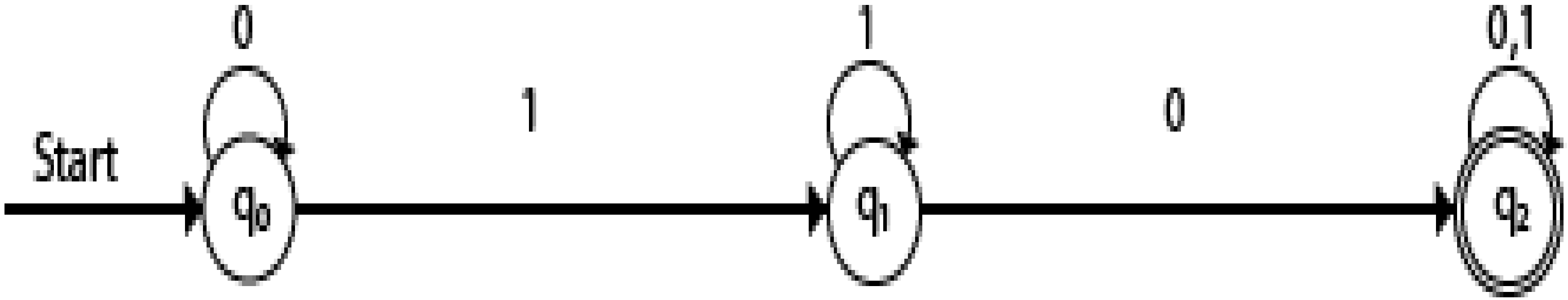
- Q: finite set of states
- ∑: finite set of the input symbol
- q0: initial state
- F: **final** state
- δ: Transition function

Transition function can be defined as:

- $\delta: Q \times \Sigma \rightarrow Q$

Example 1:

- Q = {q0, q1, q2}
- ∑ = {0, 1}
- q0 = {q0}
- F = {q2}

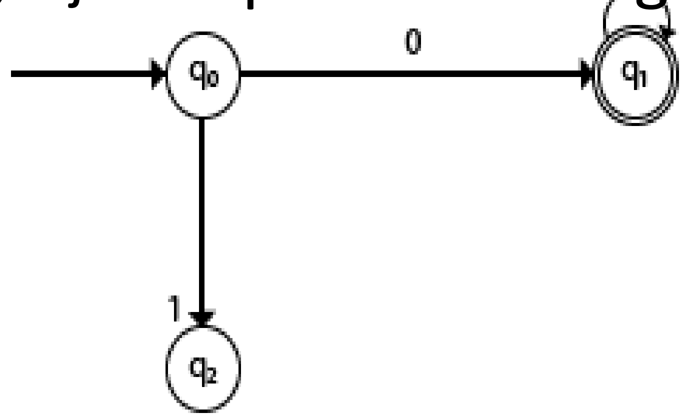


Transition Table:

Present State	Next state for Input 0	Next State of Input 1
→q0	q0	q1
q1	q2	q1
*q2	q2	q2

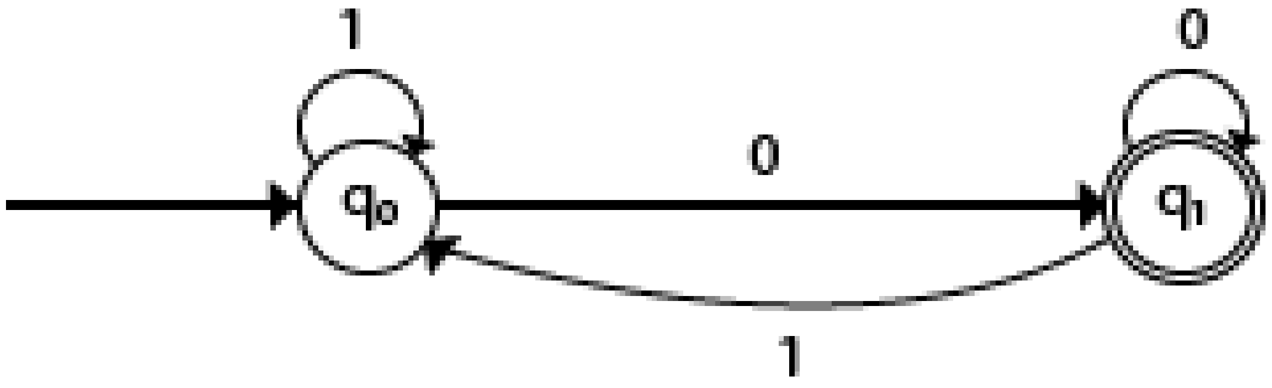
Example 2:

- DFA with ∑ = {0, 1} accepts all starting with 0.



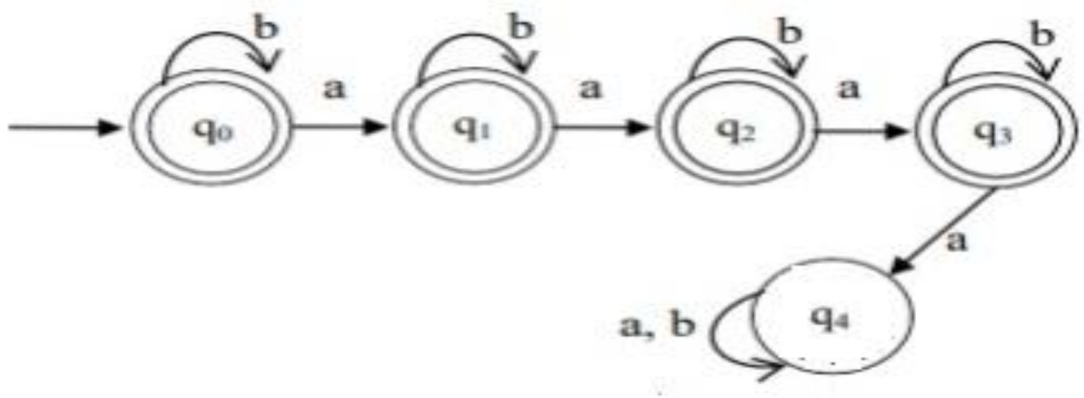
Example 3:

- DFA with ∑ = {0, 1} accepts all ending with 0.



- **Example 4:**

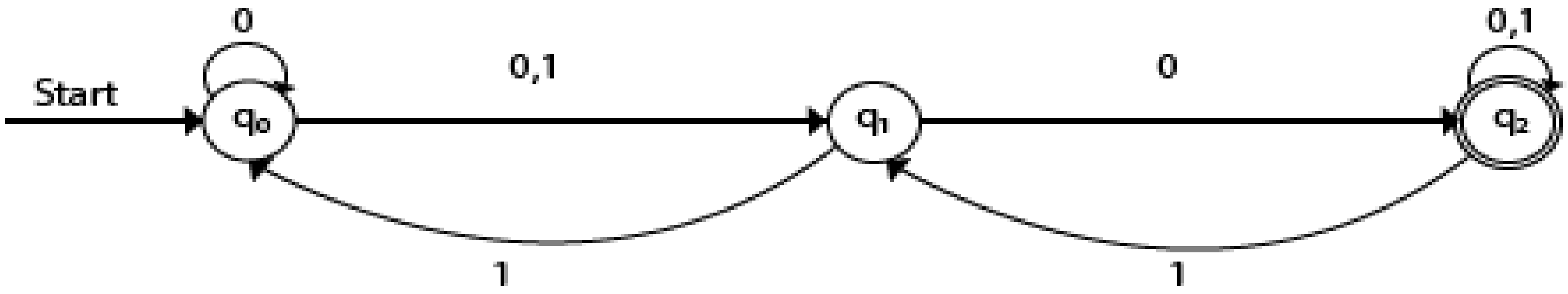
- Design a DFA that accepts at most 3 a''s



Non-Deterministic finite automata(NFA)

- The finite automata are called NFA when there exist many paths for specific input from the current state to the next state.
- Every NFA is not DFA, but each NFA can be translated into DFA.
- NFA is defined in the same way as DFA but with the following two exceptions, it contains multiple next states, and it contains ε transition.
- NFA also has five states same as DFA, but with different transition function, as shown follows:

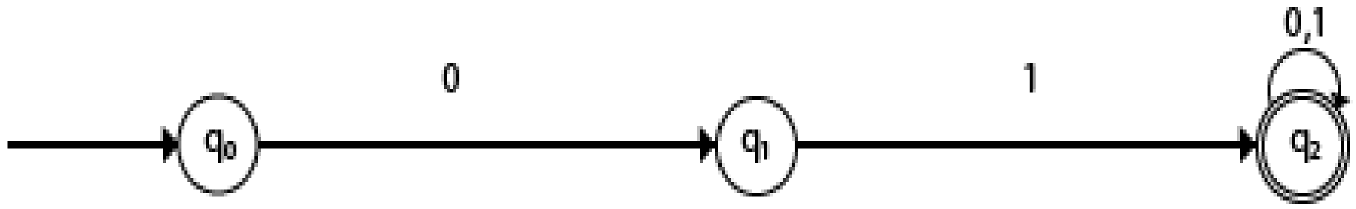
$\delta: Q \times \sum \rightarrow 2^Q$



Present State	Next state for Input 0	Next State of Input 1
→q0	q0, q1	q1
q1	q2	q0
*q2	q2	q1, q2

Example:

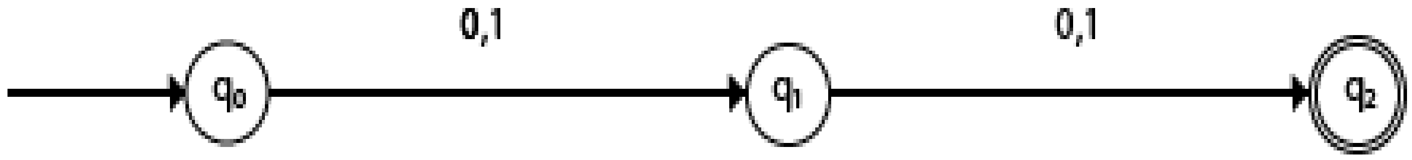
- NFA with ∑ = {0, 1} accepts all strings with 01.



Present State	Next state for Input 0	Next State of Input 1
→q0	q1	ε
q1	ε	q2
*q2	q2	q2

Example:

- NFA with ∑ = {0, 1} and accept all string of length atleast 2.

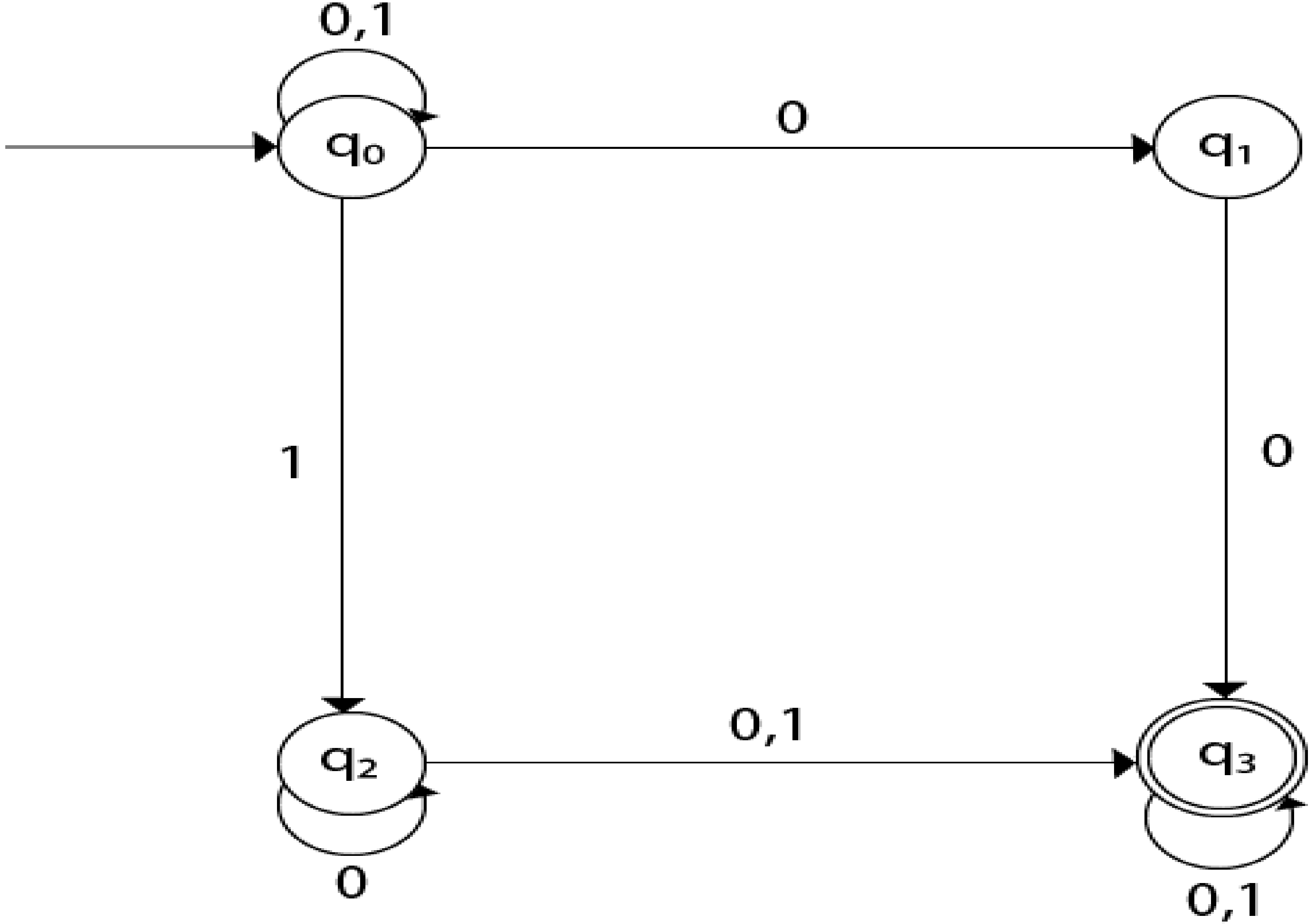


Present State	Next state for Input 0	Next State of Input 1
→q0	q1	q1
q1	q2	q2
*q2	ε	ε

Design a NFA for the transition table as given below

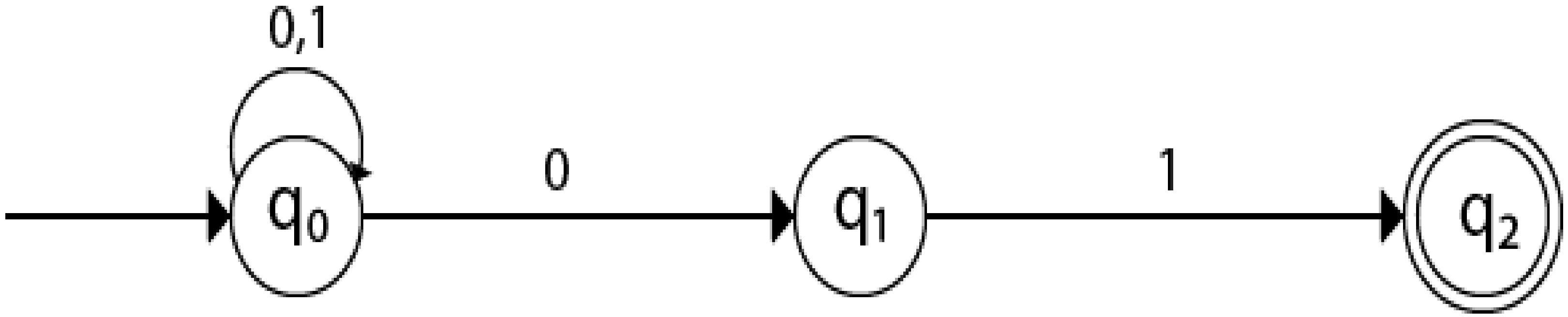
:

Present State	0	1
→q0	q0, q1	q0, q2
q1	q3	ε
q2	q2, q3	q3
→q3	q3	q3

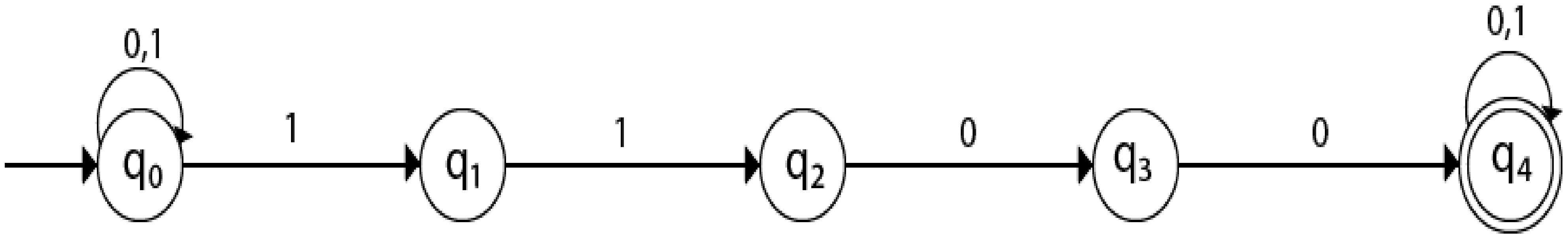

0,1
q_0
0
q_1
1
0
q_2
0,1
q_3
0
0,1

Example:

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



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



Conversion of NFA to DFA:

Step 1 − Create state table from the given NDFA.

Step 2 − Create a blank state table under possible input alphabets for the equivalent DFA.

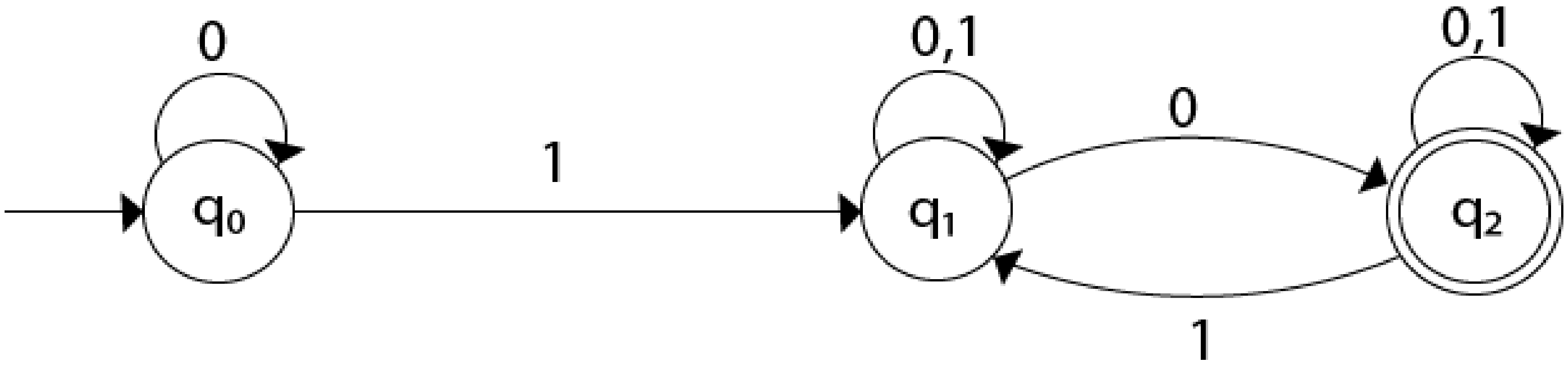
Step 3 − Mark the start state of the DFA by q0 (Same as the NDFA).

Step 4 − Find out the combination of States $\{Q_0, Q_1,\ldots , Q_n\}$ for each possible input alphabet.

Step 5 − Each time we generate a new DFA state under the input alphabet columns, we have to apply step 4 again, otherwise go to step 6.

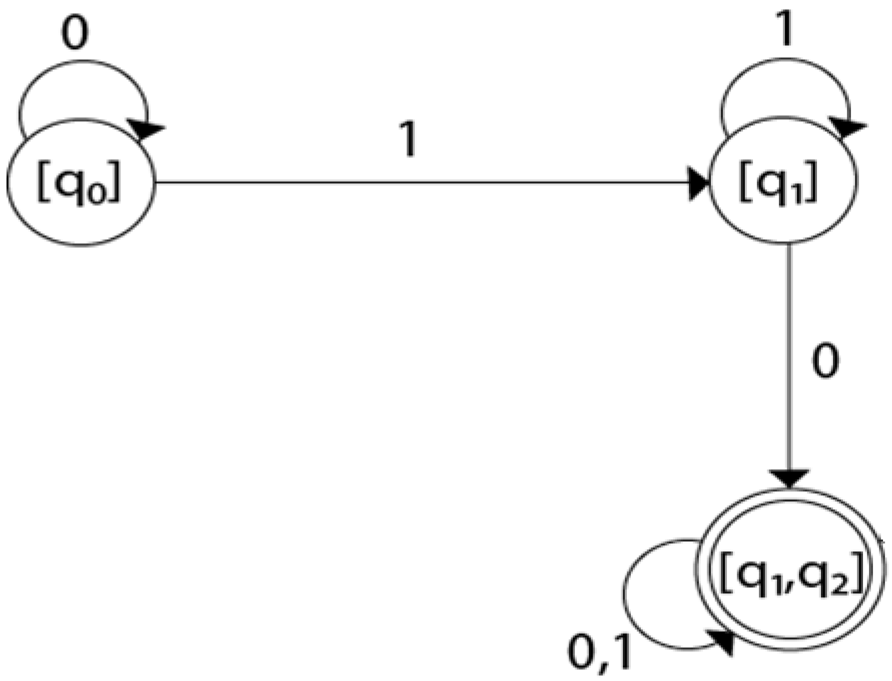
Step 6 − The states which contain any of the final states of the NDFA are the final states of the equivalent DFA.

Example:

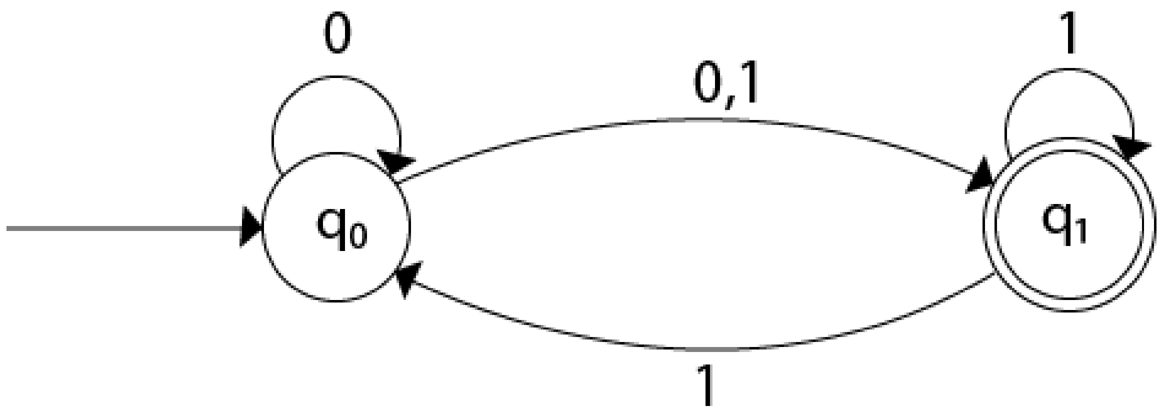


State	0	1
→q0	q0	q1
q1	{q1, q2}	q1
*q2	q2	{q1, q2}

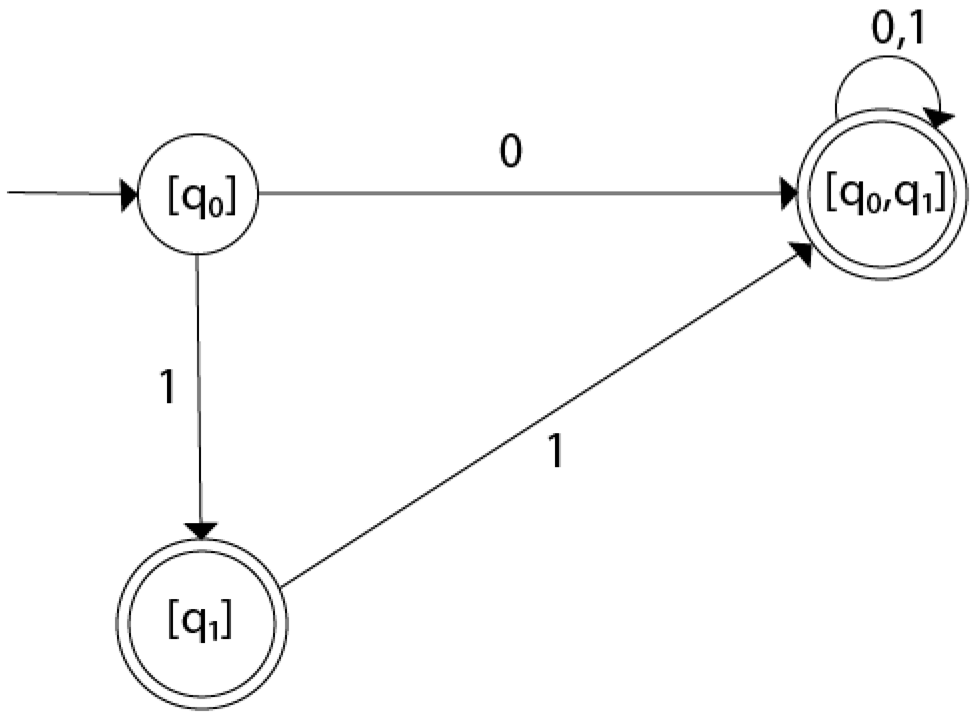
State	0	1
→[q0]	[q0]	[q1]
[q1]	[q1, q2]	[q1]
*[q1, q2]	[q1, q2]	[q1, q2]



Example:



State	0	1
→q0	{q0, q1}	{q1}
*q1	φ	{q0, q1}



State	0	1
→[q0]	[q0, q1]	[q1]
*[q1]	φ	[q0, q1]
*[q0, q1]	[q0, q1]	[q0, q1]

Minimization of DFA

Step 1: Remove all the states that are unreachable from the initial state via any set of the transition of DFA.

Step 2: Draw the transition table for all pair of states.

Step-03: Now, start applying equivalence theorem.

- Take a counter variable k and initialize it with value 0.
- Divide Q (set of states) into two sets such that one set contains all the non-final states and other set contains all the final states.
- This partition is called P0.

Step-04: Increment k by 1.

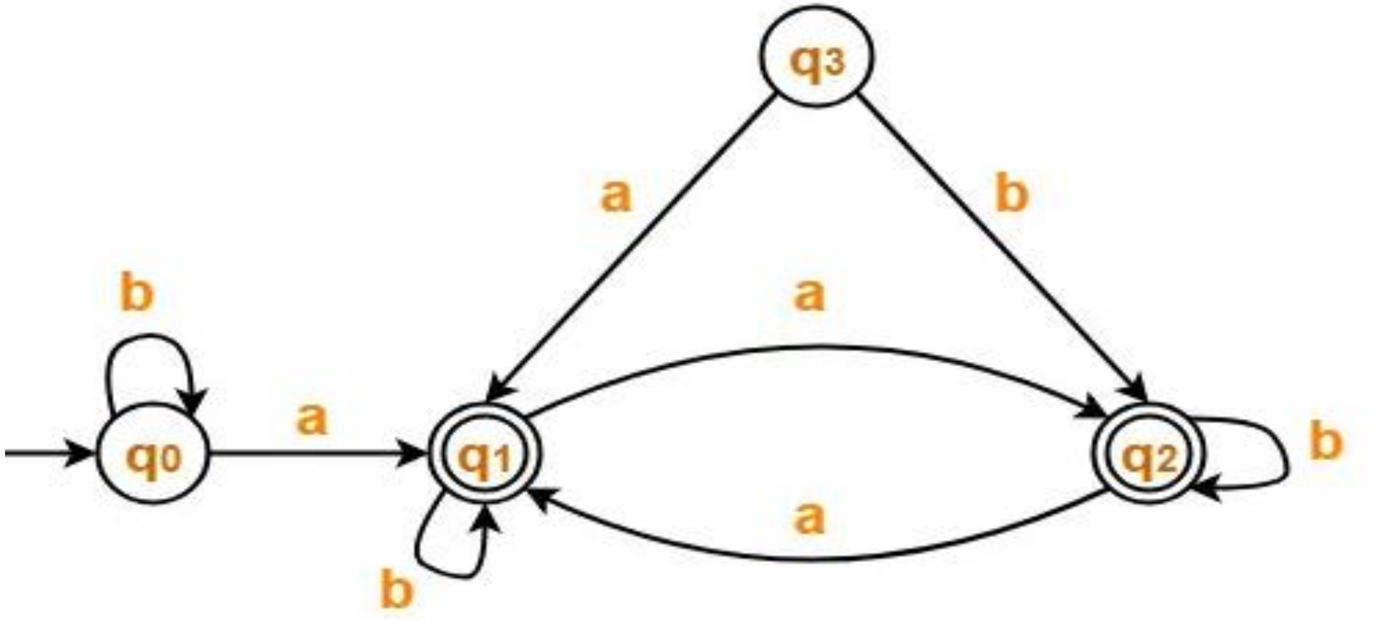
- Find Pk by partitioning the different sets of Pk-1 .
- In each set of Pk-1 , consider all the possible pair of states within each set and if the two states are distinguishable, partition the set into different sets in Pk.

Step-05: Repeat step-04 until no change in partition occurs. In other words, when you find Pk = Pk-1, stop.

Step-06: All those states which belong to the same set are equivalent.

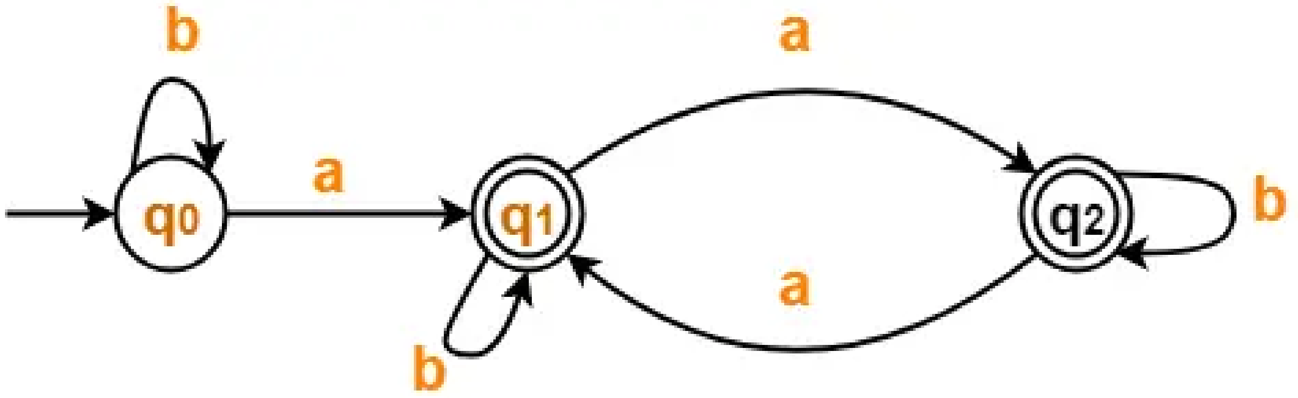
The equivalent states are merged to form a single state in the minimal DFA.

Example 1:



Step-01:

- State q_3 is inaccessible from the initial state.
- So, we eliminate it and its associated edges from the DFA.



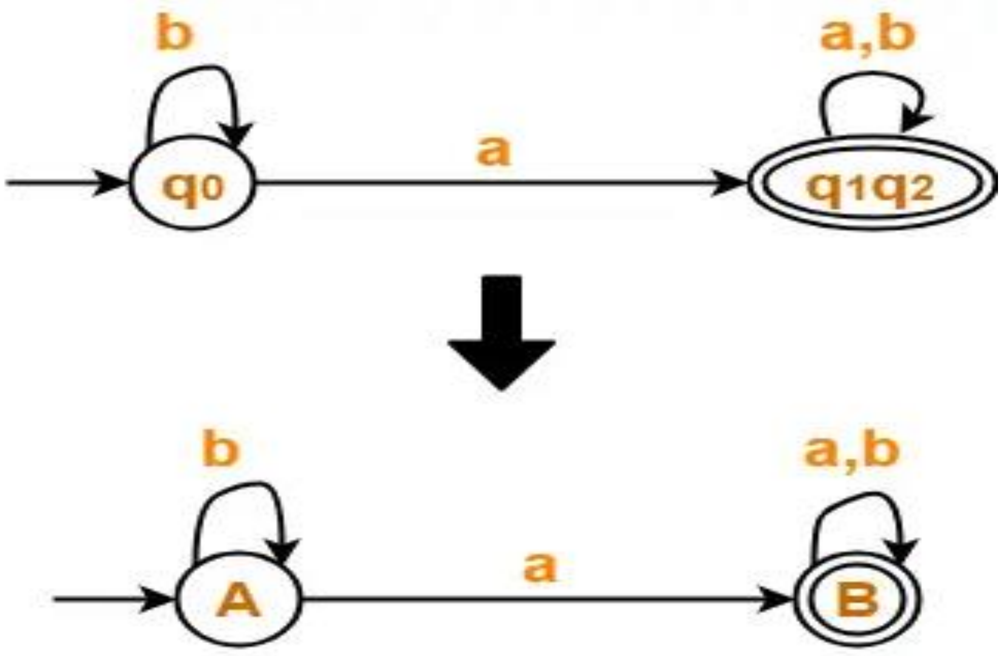
Step-02:

Draw a state transition table-

	a	b
→q0	*q1	q0
*q1	*q2	*q1
*q2	*q1	*q2

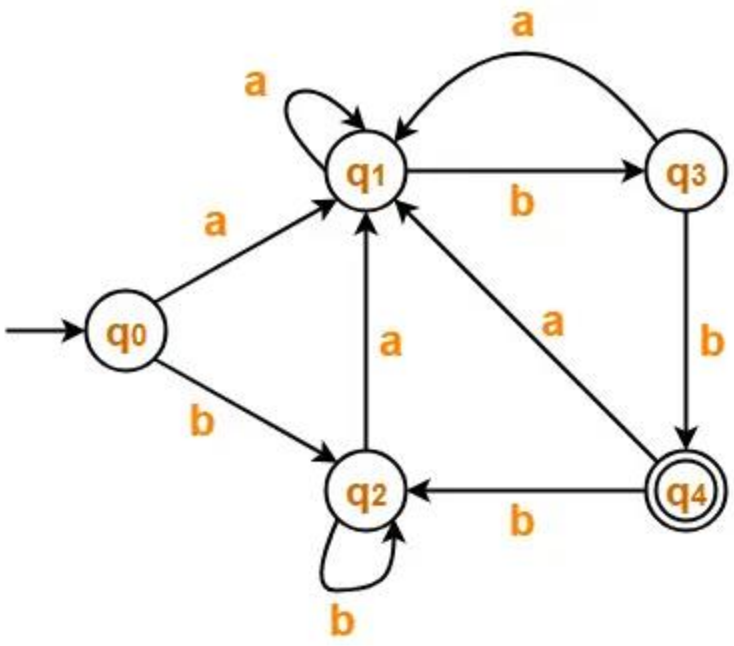
Step 3: $P_0 = \{ q_0 \} \{ q_1 , q_2 \}$
Step 4: $P_1 = \{ q_0 \} \{ q_1 , q_2 \}$

Since $P_1 = P_0$, so we stop.
From P_1, we infer that states q_1 and q_2 are equivalent and can be merged together.



Minimal DFA

Example 2:



Step-01:

The given DFA contains no dead states and inaccessible states.

Step-02:

Draw a state transition table-

	a	b
→q0	q1	q2
q1	q1	q3
q2	q1	q2
q3	q1	*q4
*q4	q1	q2

Step-03:

Now using Equivalence Theorem, we have-

$P_0 = \{ q_0 , q_1 , q_2 , q_3 \} \{ q_4 \}$

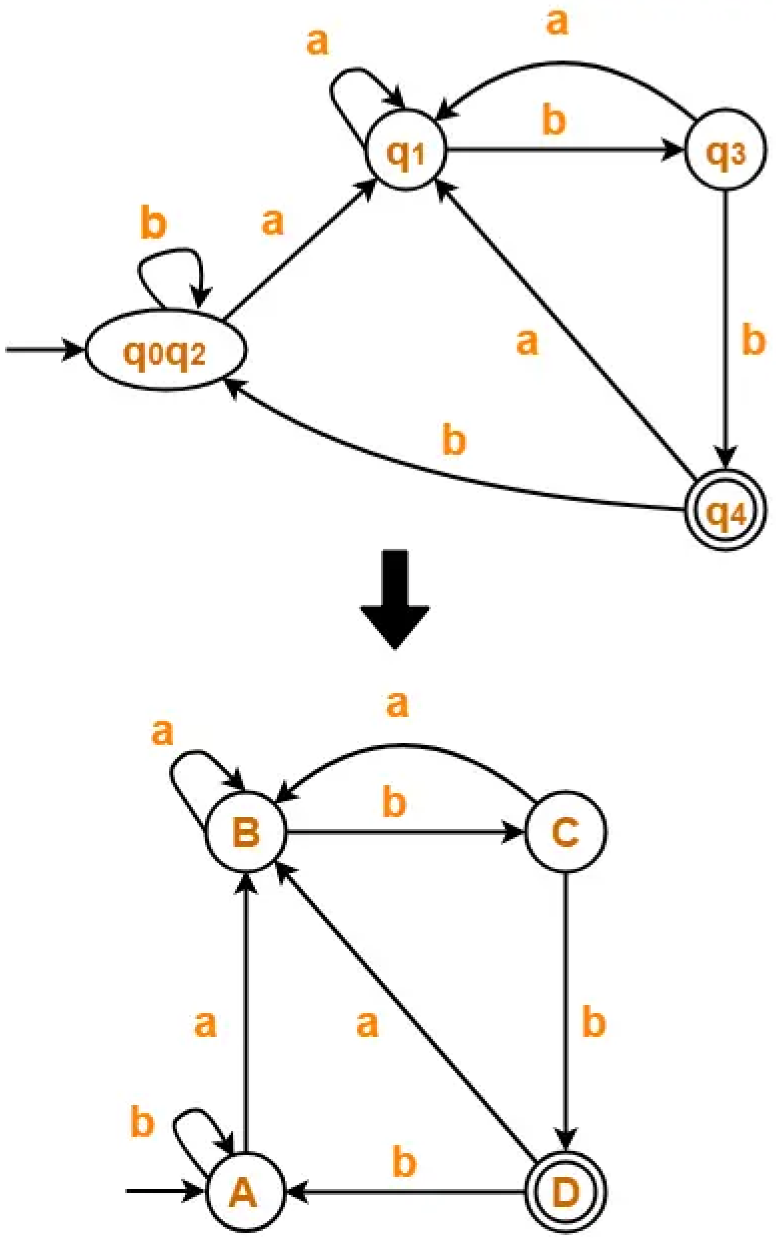
$P_1 = \{ q_0 , q_1 , q_2 \} \{ q_3 \} \{ q_4 \}$

$P_2 = \{ q_0 , q_2 \} \{ q_1 \} \{ q_3 \} \{ q_4 \}$

$P_3 = \{ q_0 , q_2 \} \{ q_1 \} \{ q_3 \} \{ q_4 \}$

Since $P_3 = P_2$, so we stop.

From P_3, we infer that states q_0 and q_2 are equivalent and can be merged together.



Minimal DFA