Theory of Computation (CT-502)

Course Instructor
ANUJ GHIMIRE

DISCLAIMER

- *This document does not claim any originality and cannot be used as a substitute for prescribed textbooks.*
- *The information presented here is merely a collection from various sources as well as freely available material from internet and textbooks.*
- *The ownership of the information lies with the respective authors or institutions.*

Chapter 2

Finite Automata

Automata

- Automata is a Greek word which means ***"self computing"***.
- Automata theory is a branch of the theory of computation which deals with the study of abstract machines and their capacities for computation.
- An abstract machine is called the automata which includes the design and analysis of the computing devices.
- Automata are the mathematical models that can perform computations on strings of symbols according to a set of rules.

Automata

- These computational devices are mainly used to solve language recognition problems.
- Language recognition problems are whose which are used to determine whether a word belongs to a language.
- Simply, finite automata (FA) are good models of computers with a extremely limited amount of memory.

Finite Automata / FSM

- Finite Automata (FA) is the simplest machine to recognize patterns.
- Finite automata or Finite State Machine (FSM) is the model of computation that accepts / rejects regular languages therefore it is also called as language recognizer.
- It captures all possible states and transition that a machine can take while responding to a sequence of input symbols
- It has a finite set of states, edge that lead from one state to another and edge labelled with a symbol.

Finite Automata / FSM

- It has a set of states and rules for moving from one state to another but it depends on applied input symbol.
- A finite automata consists of five tuples (components) as: (S, I, O, F, G) where:

S: Set of finite number of states.

I : Set of finite input symbols.

O : Finite set of outputs

F : State Relation or Transition Function

G: Output Relation

Finite Automata / FSM

- A state relation (F) is of the form:

  (current state, input) → next state

- A output relation (G) is of the form

  (current state, input) → (next state, output)

- ***ON/OFF switch as FSM:***

  M=(S, I, O, F, G) is a FSM where:

S={ON, OFF}

I={Press},

O={Device ON, Device OFF}

F Consist of:

(ON, Press) → *OFF*

(OFF, Press) → *ON*

G Consist of:

(ON, Press) → *(OFF, Device OFF)*

(OFF, Press) → *(ON, Device ON)*

Finite Automata / FSM

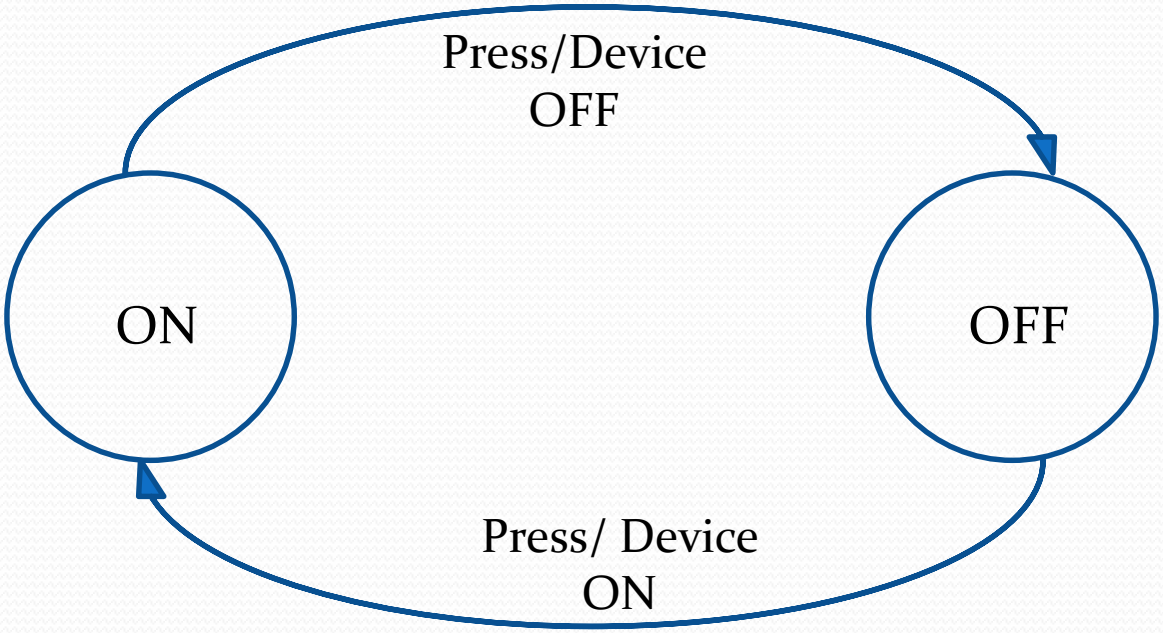


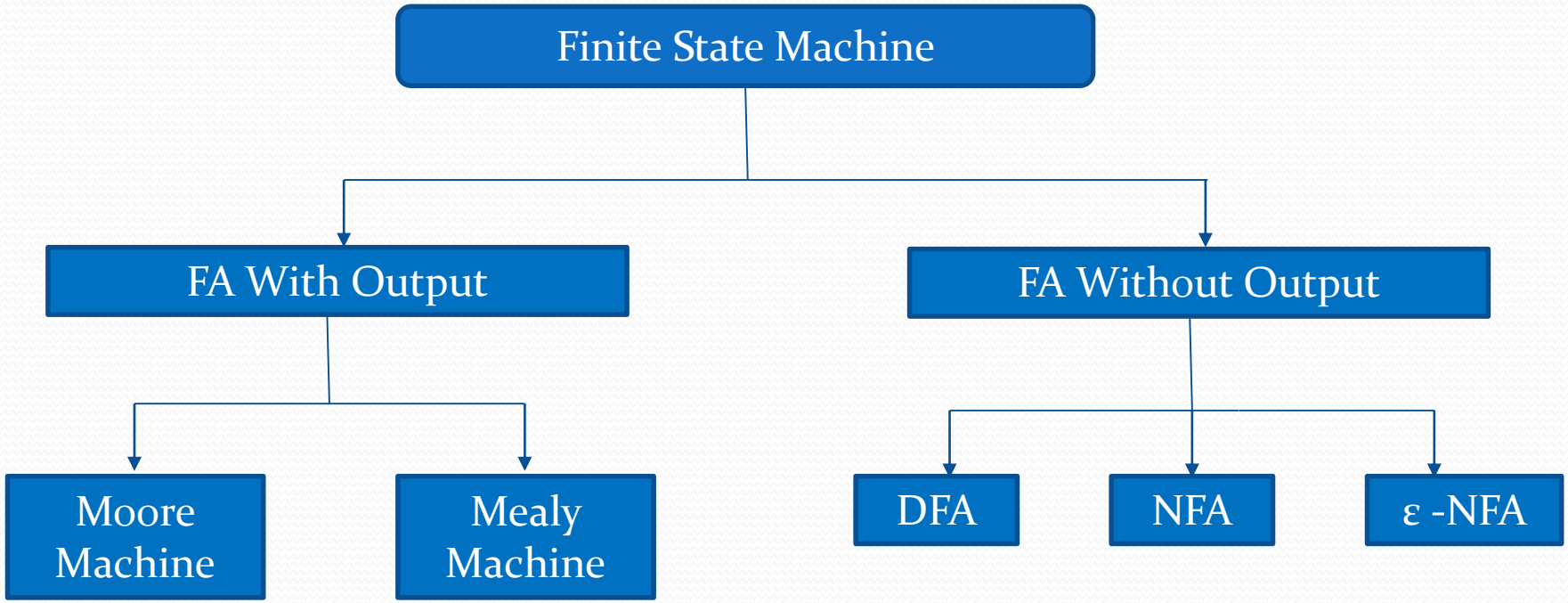
Fig: State Diagram or Transition Diagram

Finite Automata / FSM

- The finite state machines are applicable in computer science and data networking.
- For example:
  - Finite-state machines are basis for programs for spell checking, indexing, grammar, searching large bodies of text, transforming text using markup languages such as XML and HTML.
  - Network protocols that specify how computers communicate.

Finite Automata / FSM

- Categories of Finite State Machine



Deterministic Finite Automata (DFA)

- A Deterministic Finite Automata (DFA) is a finite automata, in which for each input symbol there is exactly one and only transition for each state.
- In DFA null (∈) string is not allowed i.e., DFA cannot change state without any character.
- Formally, DFA is defined by five tuples:

M= (Q, ∑, δ, q_o, F) where:

Q: Set of finite number of states.

∑ : Set of finite input symbols.

q_o : Initial /Starting State ($q_o \in Q$)

F: Set of Final States ($F \subseteq Q$)

Deterministic Finite Automata (DFA)

δ : Transition Function of the form

$$Q \times \Sigma \rightarrow Q$$

- This is the function which describes the change of states during the transition.
- This mapping is usually represented by a transition table or a transition diagram.

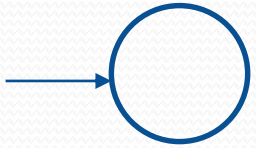
Representation of DFA

- DFA or any Finite Automata (FA) can be represented using:
  - Transition Diagram and
  - Transition Table
- The ***Transition Diagram*** is a directed labelled graph in which each vertex or node represents a state.
- The directed edges of the graph show the transition of a state and every edges are labelled with input.

Representation of DFA

- The notations used in transition diagram are:

Normal States



Initial / Starting States

Represents the transition from one state to another

Terminating States

Representation of DFA

- Let us consider a DFA as

$M = (Q, \Sigma, \delta, q_0, F)$

$Q = \{q_0, q_1, q_2, q_3\}$

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

δ : Transition Function of the form $Q \times \Sigma \rightarrow Q$ are:

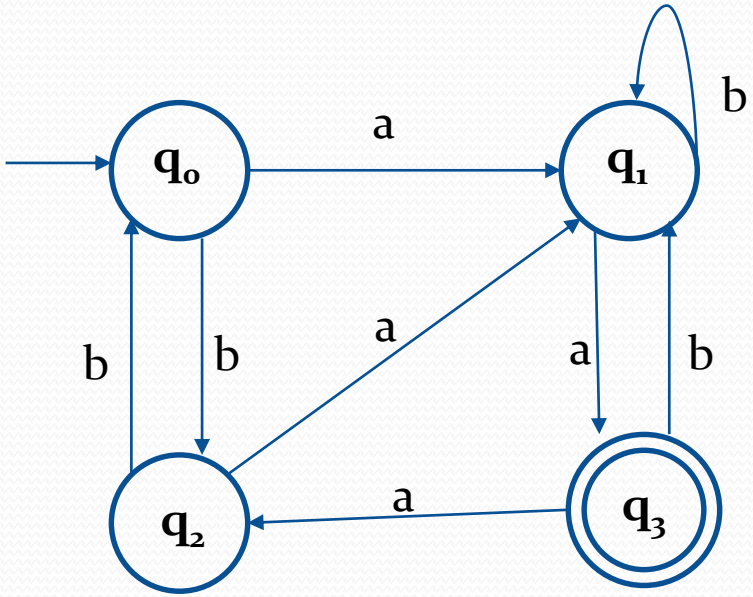
$(q_0, a) \rightarrow q_1$
$(q_0, b) \rightarrow q_2$
$(q_1, a) \rightarrow q_3$
$(q_1, b) \rightarrow q_1$
$(q_2, a) \rightarrow q_1$
$(q_2, b) \rightarrow q_0$
$(q_3, a) \rightarrow q_2$
$(q_3, b) \rightarrow q_1$

$q_0 : q_0$

$F: q_3$

Representation of DFA

- Transition Diagram of given DFA



Representation of DFA

- A ***Transition Table*** is a tabular representation of a transition function of finite automata.
- The rows of the table correspond to the states, and the columns correspond to the input.
- The intersection of each row and the column i.e. each cell corresponding to the next state.
- Here the initial state or starting state is marked with arrow head and the final state are marked with (*) symbol

Representation of DFA

- Let us consider a DFA as

$M = (Q, \Sigma, \delta, q_o, F)$

$Q = \{q_0, q_1, q_2, q_3\}$

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

δ : Transition Function of the form $Q \times \Sigma \rightarrow Q$ are:

$\delta(q_0, a) \rightarrow q_1$
$\delta(q_0, b) \rightarrow q_2$
$\delta(q_1, a) \rightarrow q_3$

$\delta(q_1, b) \rightarrow q_1$
$\delta(q_2, a) \rightarrow q_1$
$\delta(q_2, b) \rightarrow q_0$

$\delta(q_3, a) \rightarrow q_2$
$\delta(q_3, b) \rightarrow q_1$

$q_0 : q_0$

$F: q_3$

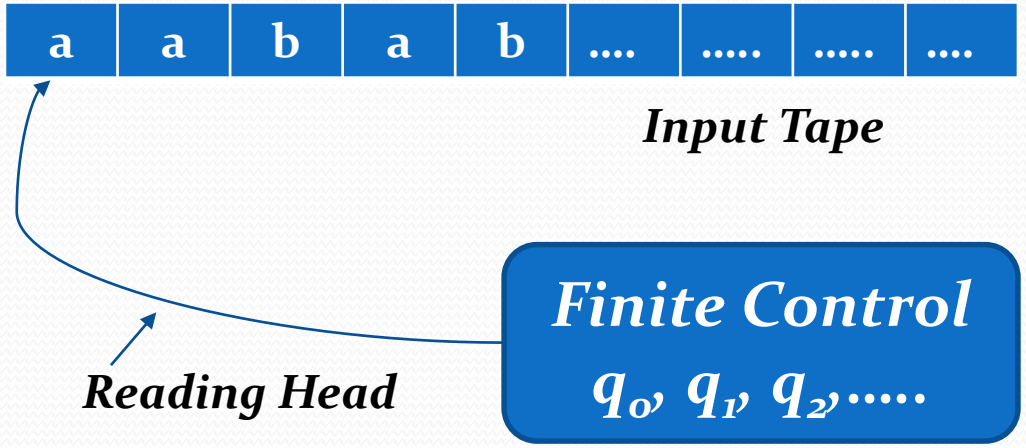
Representation of DFA

- A Transition Table of given DFA

Q/Σ	a	b
→ q_0	q_1	q_2
q_1	q_3	q_1
q_2	q_1	q_0
* q_3	q_2	q_1

Processing of String

- For processing any string by an automata, it has three blocks:
  - Input Tape
  - Reading Head
  - Finite Control



- ***Input Tape*** of automata is responsible for storing the symbol of input string.
- It is divided into number of square cell, each cell stores the symbol of input string.

Processing of String

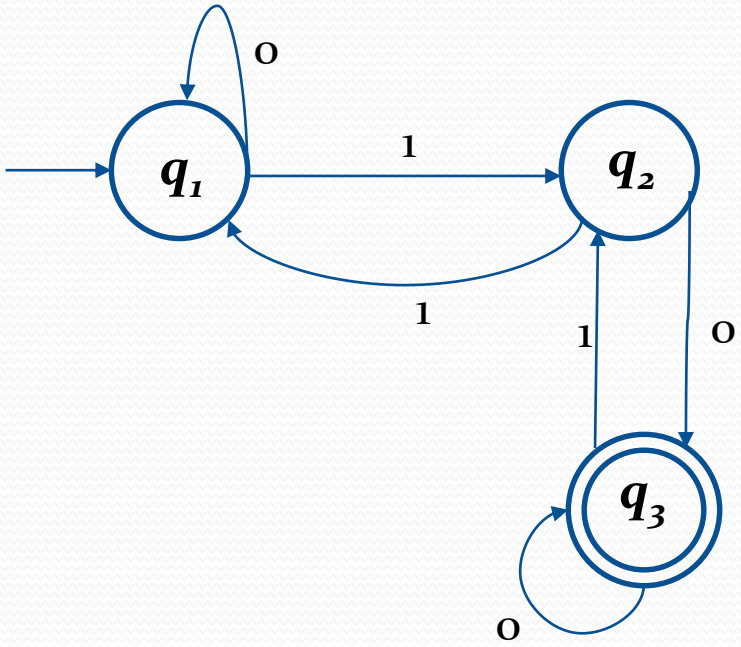
- ***Reading Head*** scans the symbol from the input tape, each cell at a time in one direction only, either from left to right or right to left.
- The processing of FA is re controlled by ***Finite Control*** i.e. all the states, current state, next state obtained after each transition function must be defined in the finite control.
- Finite control is also responsible for giving next state regularly and the state obtained after each transition is tracked by finite control.

Processing of String

- Initially the reading head is placed at the left must cell of input tape and then it scans each cell in the defined direction.
- When the input string becomes empty and it the finite control gives one of the defined final states as the next state, the string is said to be ***accepted*** otherwise ***rejected***.

Processing of String

- Consider the following DFA and check whether the string ***w=00111010100*** is accepted or rejected



Processing of String

- Solution,

The transition function of given DFA is written as:

$\delta(q_1,0) \rightarrow q_1$ $\quad$ $\delta(q_2,0) \rightarrow q_3$

$\delta(q_1,1) \rightarrow q_2$ $\quad$ $\delta(q_2,1) \rightarrow q_1$

$\delta(q_3,0) \rightarrow q_3$

$\delta(q_3,1) \rightarrow q_2$

Given string ***w=0011110100***

Processing of String

Now,

$$\begin{aligned}\delta(q_1, 001110100) &\rightarrow \delta(q_1, 01110100)\\ &\rightarrow \delta(q_1, 1110100)\\ &\rightarrow \delta(q_2, 110100)\\ &\rightarrow \delta(q_1, 10100)\\ &\rightarrow \delta(q_2, 0100)\\ &\rightarrow \delta(q_3, 100)\\ &\rightarrow \delta(q_2, 00)\\ &\rightarrow \delta(q_3, 0)\\ &\rightarrow \delta(q_3, \varepsilon)\end{aligned}$$

Here, after processing each of the input string the DFA is in the final state hence the given string ***w=001110100*** is accepted by given DFA.

Designing DFA

- Procedure for Designing DFA
  - Analyze the set of strings i.e. language which is accepted by the DFA.
  - Draw the transition diagram as per the string that the language generates.
  - Confirm that every state is check for the output state for every input symbol.
  - Confirm that no state have two different output state for a single input symbol.
  - Confirm that there is only one initial state and at least one final state in transition diagram of DFA.

Designing DFA (Example_1)

- *Design DFA over ∑{a,b} that accepts the string with zero or more* ***a*** .

Solution,

We need to construct the DFA as M= (Q, ∑, δ, q_o, F) where:

Q: Set of finite number of states.

∑ : Set of finite input symbols.

q_o : Initial /Starting State ($q_o \in Q$)

F: Set of Final States ($F \subseteq Q$)

δ : Transition Function of the form

$$Q \times \sum \rightarrow Q$$

Set of string that the DFA accepts are:

{ε, a, aa, aaa, aaaa, aaaaa……}

Designing DFA (Example_1)

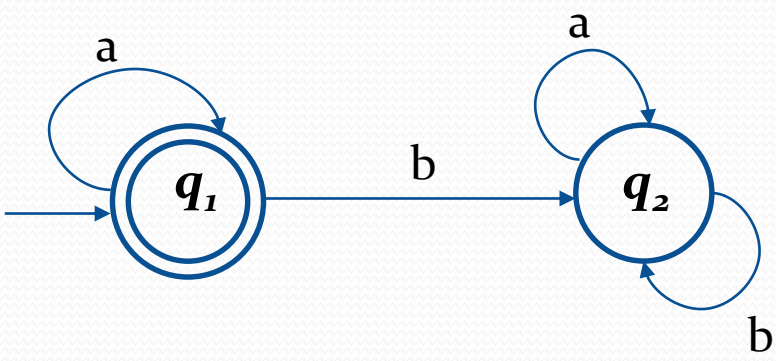


Figure: DFA that accepts the string with zero or more ***a***

Designing DFA (Example_1)

- Now the required DFA is

$M = (Q, \Sigma, \delta, q_o, F)$

$Q = \{q_1, q_2\}$

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

δ : Transition Function of the form $Q \times \Sigma \rightarrow Q$ are:

$$(q_1, a) \rightarrow q_1$$
$$(q_1, b) \rightarrow q_2$$
$$(q_2, a) \rightarrow q_2$$
$$(q_2, b) \rightarrow q_2$$

$q_0 : q_1$

$F : q_1$

Designing DFA (Example_2)

- *Design DFA over ∑{a,b} that accepts the string ending with* ***bab*** *and check w=baabbbababbbab is accepted or not.*

Solution,

We need to construct the DFA as M= (Q, ∑, δ, q_o, F) where:

Q: Set of finite number of states.

∑ : Set of finite input symbols.

q_o : Initial /Starting State ($q_o \in Q$)

F: Set of Final States ($F \subseteq Q$)

δ : Transition Function of the form

$$Q \times \Sigma \rightarrow Q$$

Set of string that the DFA accepts are:

{***bab***, a***bab***, ab***bab***, aa***bab***, bbab***bab***, babaa***bab***,.....}

Designing DFA (Example_2)

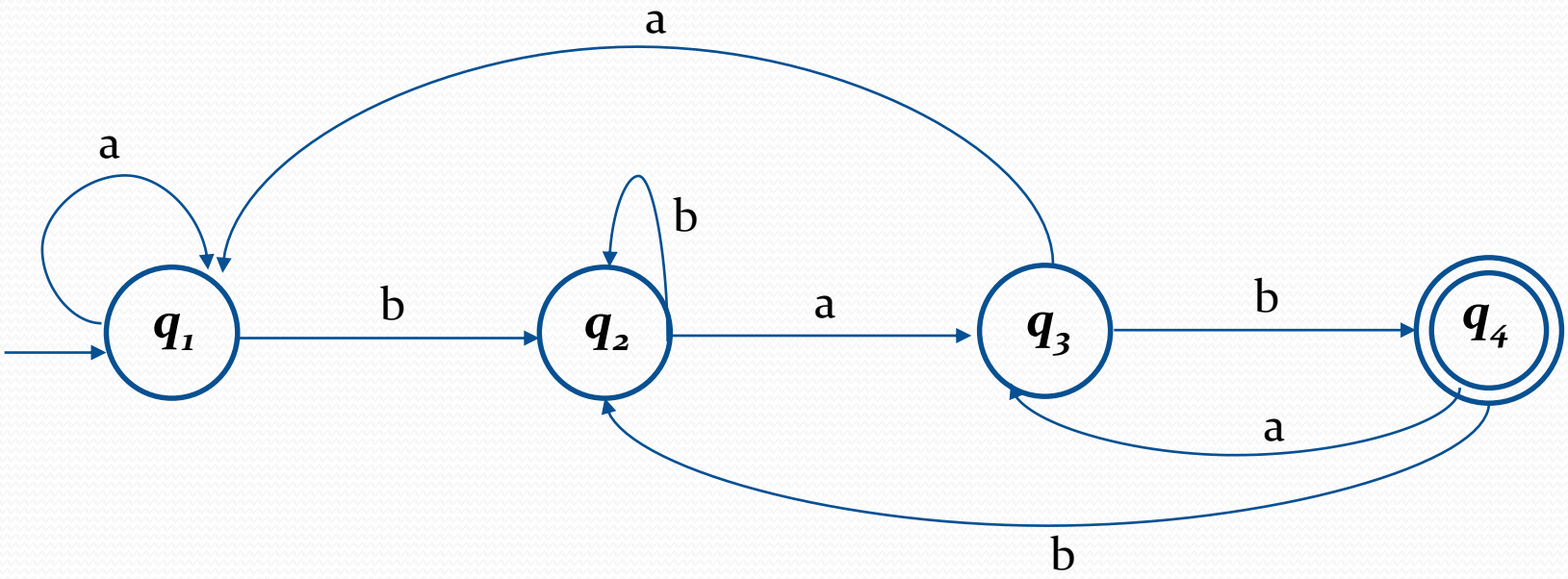


Figure: DFA over the {a,b} that accepts the string ending with ***bab***

Designing DFA (Example_2)

- Now the required DFA is

$M = (Q, \Sigma, \delta, q_o, F)$

$Q = \{q_1, q_2, q_3, q_4\}$

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

δ : Transition Function of the form $Q \times \Sigma \rightarrow Q$ are:

$(q_1, a) \rightarrow q_1$
$(q_1, b) \rightarrow q_2$
$(q_2, a) \rightarrow q_3$
$(q_2, b) \rightarrow q_2$
$(q_3, a) \rightarrow q_1$
$(q_3, b) \rightarrow q_4$
$(q_4, a) \rightarrow q_3$
$(q_4, b) \rightarrow q_2$

$q_0 : q_1$

$F: q_4$

Designing DFA (Example_2)

Now, check for w=baabbbababbab

$\delta(q_1, baabbbababbab) \rightarrow \delta(q_2, aabbbababbab)$
$\rightarrow \delta(q_3, abbbababbab)$
$\rightarrow \delta(q_1, bbbababbab)$
$\rightarrow \delta(q_2, bbababbab)$
$\rightarrow \delta(q_2, bababbab)$
$\rightarrow \delta(q_2, ababbab)$
$\rightarrow \delta(q_3, babbab)$
$\rightarrow \delta(q_4, abbab)$
$\rightarrow \delta(q_1, bbab)$
$\rightarrow \delta(q_2, bab)$
$\rightarrow \delta(q_2, ab)$
$\rightarrow \delta(q_3, b)$
$\rightarrow \delta(q_4, \varepsilon)$

Here, after processing each of the input string the DFA is in the final state hence the given string ***w=baabbbababbab*** is accepted by given DFA.

Designing DFA (Example_3)

- *Design DFA over ∑{0,1} that accepts the string that starts with **110**.*

Solution,

We need to construct the DFA as M= (Q, ∑, δ, q_o, F) where:

Q: Set of finite number of states.

∑ : Set of finite input symbols.

q_o : Initial /Starting State ($q_o \in Q$)

F: Set of Final States ($F \subseteq Q$)

δ : Transition Function of the form

$$Q \times \sum \rightarrow Q$$

Set of string that the DFA accepts are:

{***110***, ***110***11, ***110***00, ***110***1011, ***110***0111, ***110***00110,.....}

Designing DFA (Example_3)

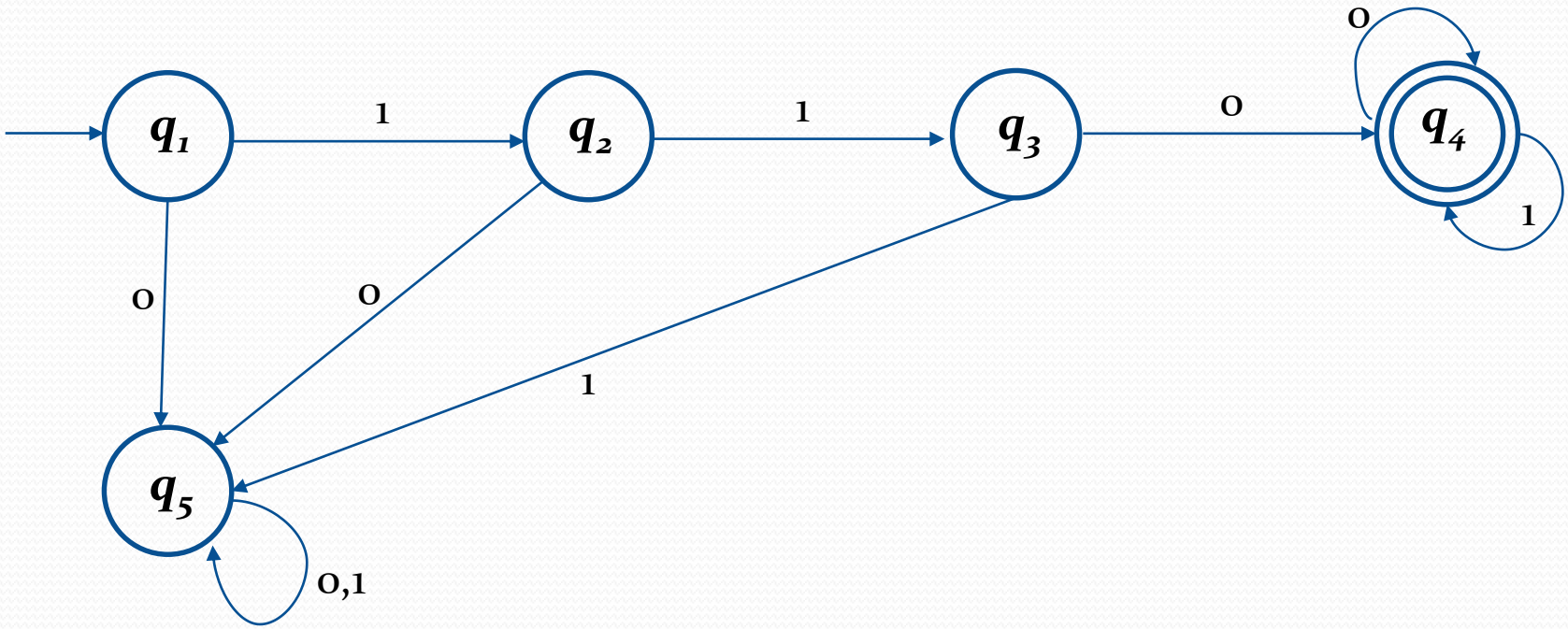


Figure: DFA over the {a,b} that accepts the string starting with ***110***

Designing DFA (Example_3)

- Now the required DFA is

$M = (Q, \Sigma, \delta, q_0, F)$

$Q = \{q_1, q_2, q_3, q_4, q_5\}$

$\Sigma = \{0, 1\}$

δ : Transition Function of the form $Q \times \Sigma \rightarrow Q$ are:

$(q_1, 0) \rightarrow q_5$
$(q_1, 1) \rightarrow q_2$
$(q_2, 0) \rightarrow q_5$
$(q_2, 1) \rightarrow q_3$
$(q_3, 0) \rightarrow q_4$
$(q_3, 1) \rightarrow q_5$
$(q_4, 0) \rightarrow q_4$
$(q_4, 1) \rightarrow q_4$
$(q_5, 0) \rightarrow q_5$
$(q_5, 1) \rightarrow q_5$

$q_0 : q_1$

$F: q_4$

Designing DFA (Example_4)

- *Design DFA over {0,1} that accepts the string with even number of 0 and odd number of 1.*

Solution,

We need to construct the DFA as M= (Q, ∑, δ, q_0, F) where:

Q: Set of finite number of states.

∑ : Set of finite input symbols.

q_0 : Initial /Starting State ($q_0 \in Q$)

F: Set of Final States ($F \subseteq Q$)

δ : Transition Function of the form

$$Q \times \sum \rightarrow Q$$

Set of string that the DFA accepts are:

{1, 001, 111, 01011, 01101, 1001010, 110111000, 00101010110,.....}

Designing DFA (Example_4)

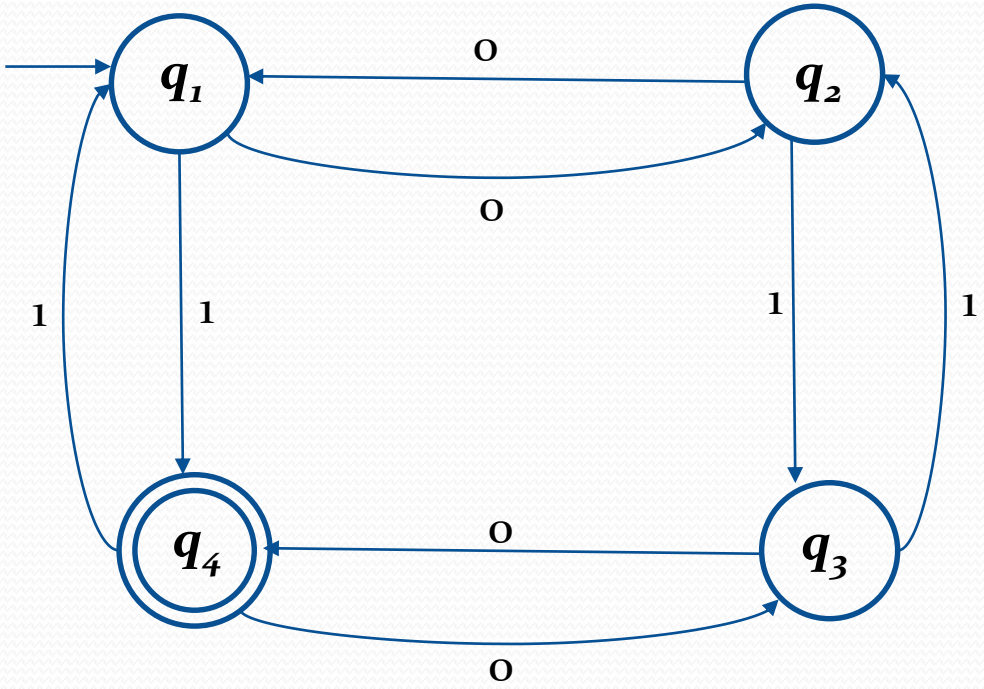


Figure: DFA over the {0,1} that accepts the string with even **0** *and odd* **1**

Designing DFA (Example_4)

- Now the required DFA is

$M = (Q, \Sigma, \delta, q_0, F)$

$Q = \{q_1, q_2, q_3, q_4\}$

$\Sigma = \{0, 1\}$

δ : Transition Function of the form $Q \times \Sigma \rightarrow Q$ are:

$(q_1, 0) \rightarrow q_2$ $(q_1, 1) \rightarrow q_4$ $(q_2, 0) \rightarrow q_1$ $(q_2, 1) \rightarrow q_3$ $(q_3, 0) \rightarrow q_4$ $(q_3, 1) \rightarrow q_2$ $(q_4, 0) \rightarrow q_3$ $(q_4, 1) \rightarrow q_1$

$q_0 : q_1$

$F: q_4$

Designing DFA (Example_5)

- *Design a deterministic finite automaton M that accepts the language L= {w ϵ {a, b} * : w does not contain three consecutive b's }*

Solution,

We need to construct the DFA as M= (Q, ∑, δ, q_o, F) where:

Q: Set of finite number of states.

∑ : Set of finite input symbols.

q_o : Initial /Starting State ($q_o \in Q$)

F: Set of Final States ($F \subseteq Q$)

δ : Transition Function of the form

$$Q \times \sum \rightarrow Q$$

Set of string that the DFA accepts are:

{a, bb, aab, aabb, aabba, babbab, bbabbaba,……}

Designing DFA (Example_5)

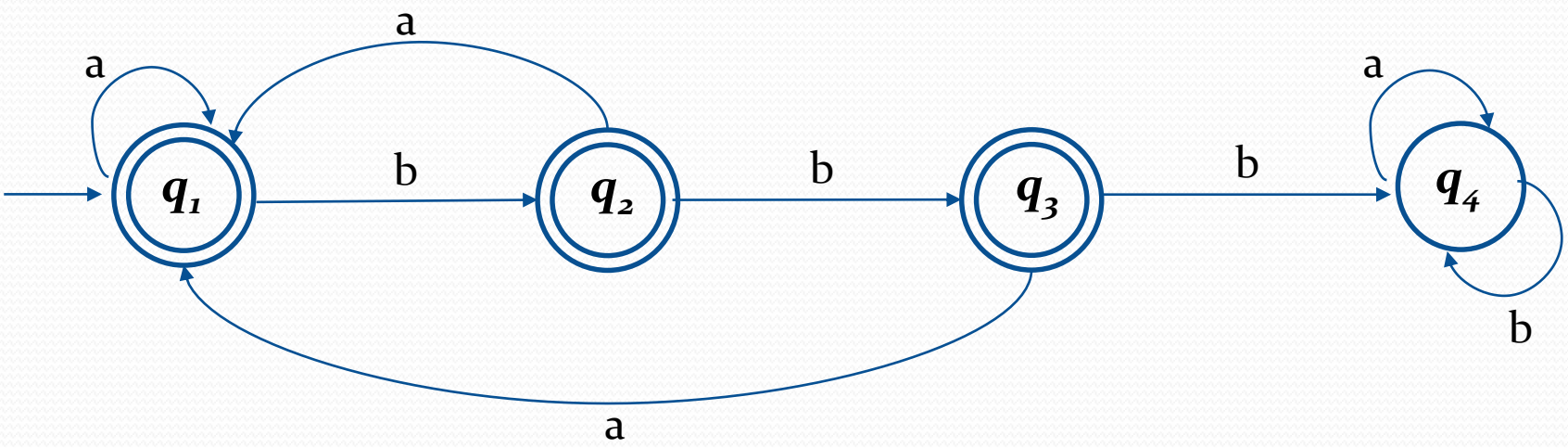


*Figure: DFA that accepts the language L= {w ϵ {a, b} * : w does not contain three consecutive b's }*

Designing DFA (Example_5)

- Now the required DFA is

$M = (Q, \Sigma, \delta, q_0, F)$

$Q = \{q_1, q_2, q_3, q_4\}$

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

δ : Transition Function of the form $Q \times \Sigma \rightarrow Q$ are:

$(q_1, a) \rightarrow q_1$
$(q_1, b) \rightarrow q_2$
$(q_2, a) \rightarrow q_1$
$(q_2, b) \rightarrow q_3$
$(q_3, a) \rightarrow q_1$
$(q_3, b) \rightarrow q_4$
$(q_4, a) \rightarrow q_4$
$(q_4, b) \rightarrow q_4$

$q_0 : q_1$

$F: \{q_1, q_2, q_3\}$

Non-Deterministic Finite Automata (NFA)

- In NFA the for a particular input symbol machine can move to any combination of states.
- In other word the exact state to which the machine moves cannot be determined in non deterministic finite automata.
- The automaton, as it reads the input string, may choose at each step to go into anyone of these legal next states; the choice is not determined by anything in our model, and is therefore said to be non-deterministic.
- ***NFA does not contain any dead state.***

Non-Deterministic Finite Automata (NFA)

- Formally, NFA is defined by five tuples (similar to DFA):

$$M = (Q, \Sigma, \delta, q_o, F) \text{ where:}$$

Q: Set of finite number of states.

Σ : Set of finite input symbols.

q_o : Initial /Starting State ($q_o \in Q$)

F: Set of Final States ($F \subseteq Q$)

δ : Transition Function of the form

$$Q \times \Sigma \rightarrow 2^Q$$

Non-Deterministic Finite Automata (NFA)

- The difference between DFA and the NFA is in the type of transition function (δ).
- In NFA, δ is a function that takes a state and input symbol as arguments and returns a set of zero, one, or more states as output but in case of DFA exactly only one state is returned an output.
- In fact every DFA is a NFA but not every NFA is a DFA. ***But there may exists an equivalent DFA for every NFA.***

Non-Deterministic Finite Automata (NFA)

- Consider a NFA as:

$M = (Q, \Sigma, \delta, q_o, F)$

$Q = \{q_1, q_2, q_3, q_4\}$

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

δ : Transition Function of the form $Q \times \Sigma \rightarrow 2^Q$ are:

$(q_1, a) \rightarrow \{q_1, q_2\}$
$(q_1, b) \rightarrow q_3$
$(q_2, a) \rightarrow q_3$
$(q_2, b) \rightarrow \phi$
$(q_3, a) \rightarrow \phi$
$(q_3, b) \rightarrow \{q_3, q_4\}$
$(q_4, a) \rightarrow \{q_4, q_2\}$
$(q_4, b) \rightarrow \{q_1, q_3\}$

$q_0 : q_1$

$F: q_4$

Non-Deterministic Finite Automata (NFA)

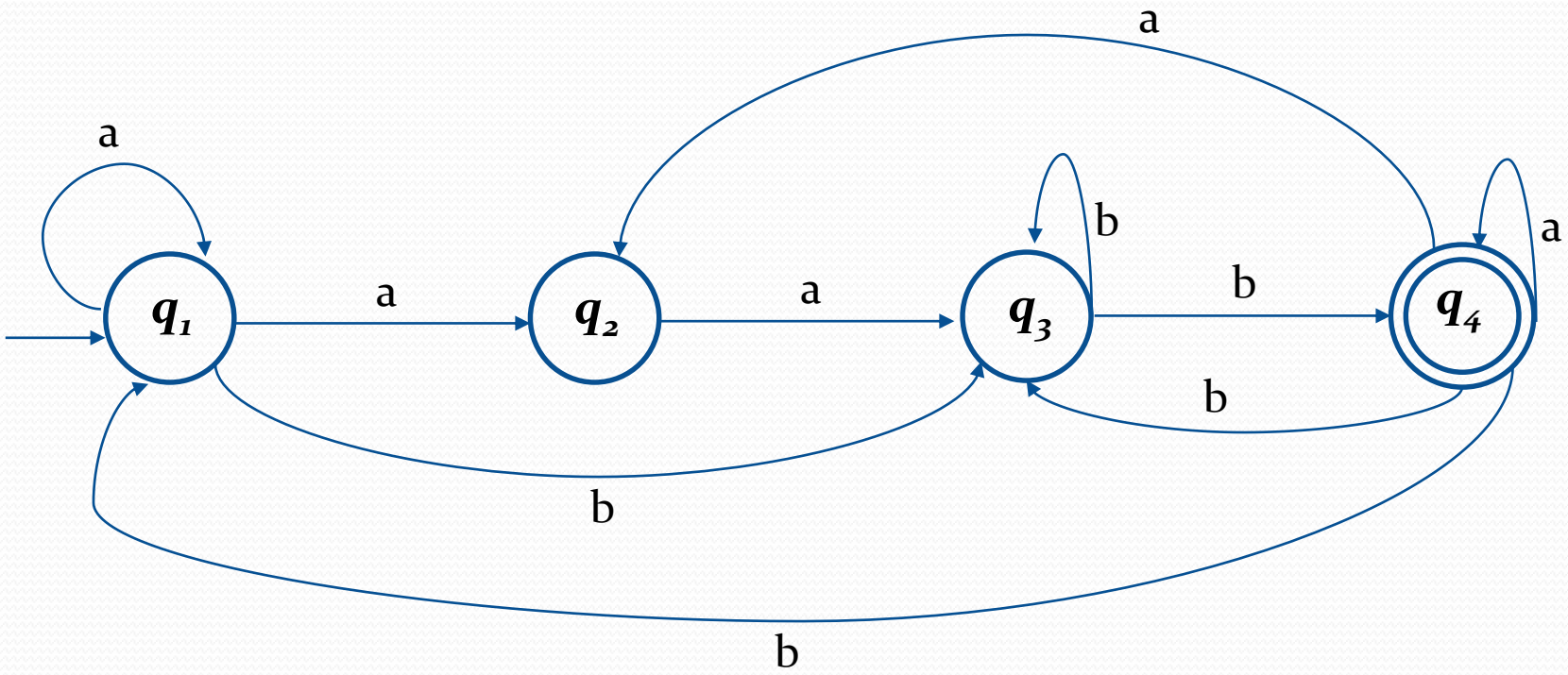


Figure: NFA

Non-Deterministic Finite Automata (NFA)

- A Transition Table of given NFA

Q/Σ	a	b
→ q_1	$\{q_1, q_2\}$	q_3
q_2	q_3	ϕ
q_3	ϕ	$\{q_3, q_4\}$
* q_4	$\{q_2, q_4\}$	$\{q_1, q_3\}$

Non-Deterministic Finite Automata (NFA)

- Consider the language L = (ab+aba)*, and draw its DFA

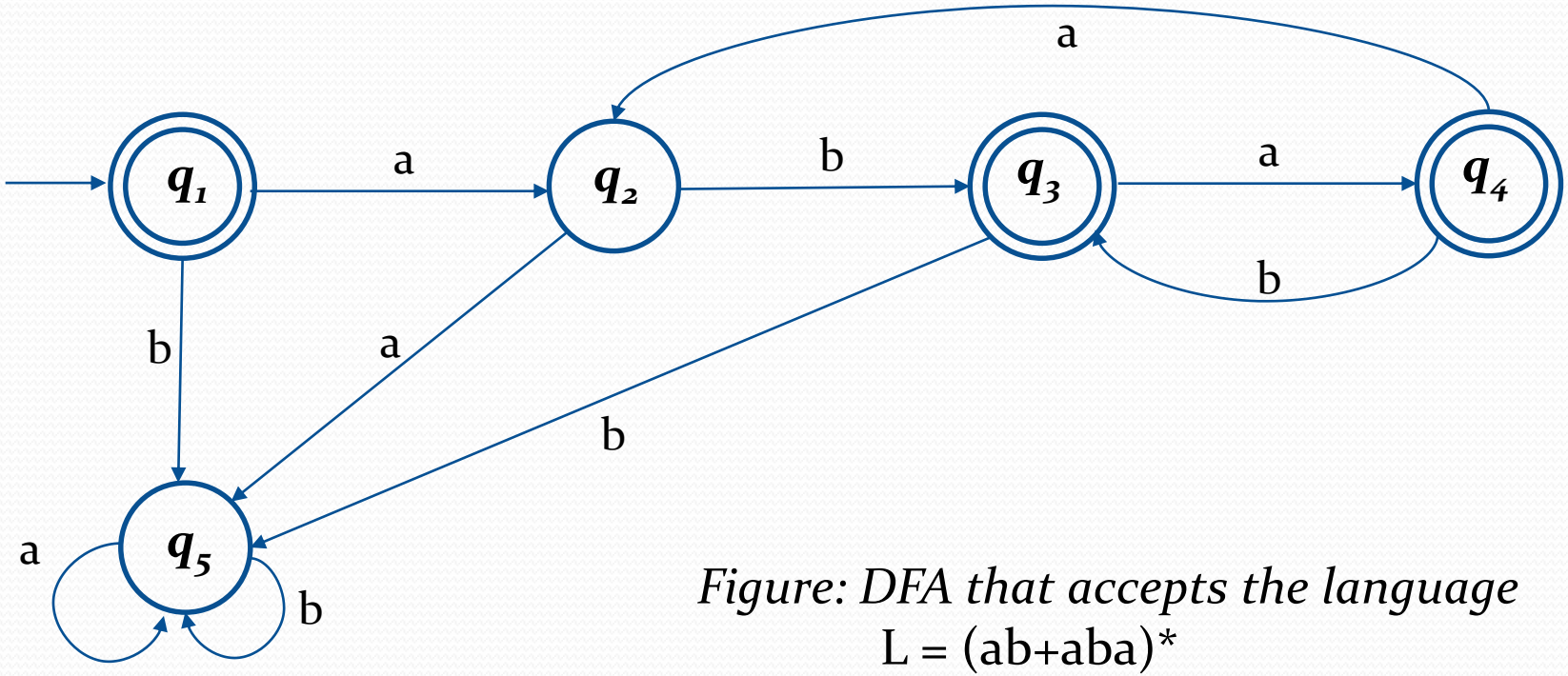


Figure: DFA that accepts the language L = (ab+aba)*

Non-Deterministic Finite Automata (NFA)

- For this language L = (ab+aba)*, we can have the NFA as

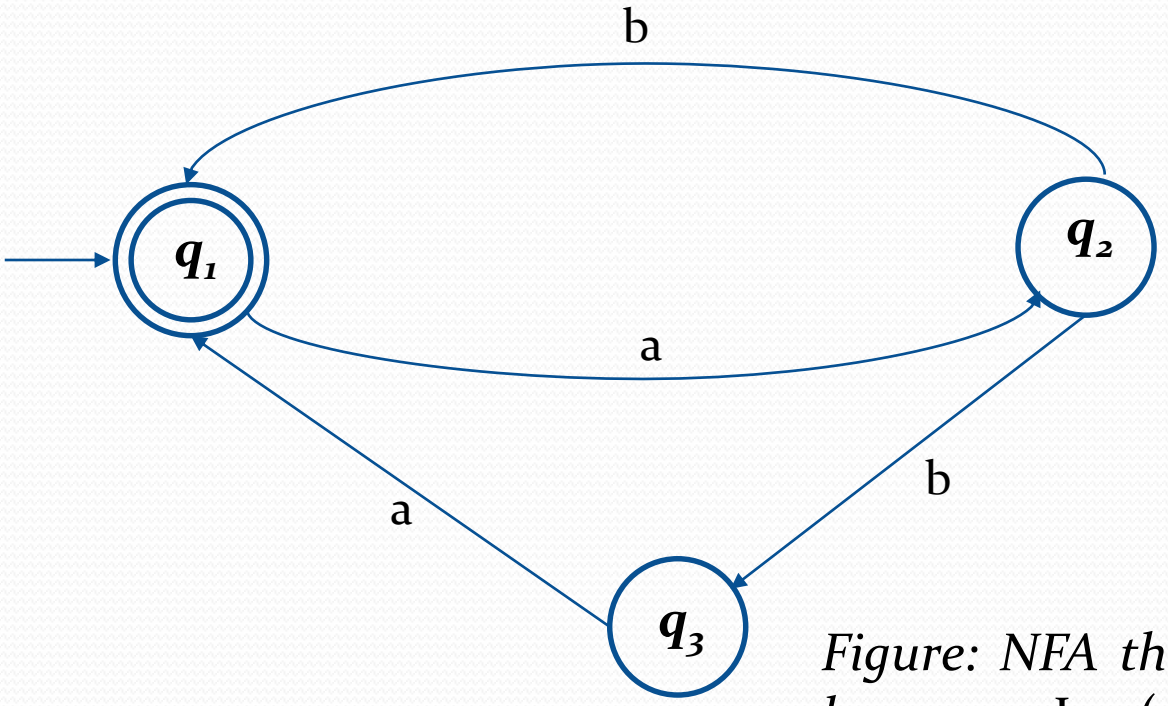


Figure: NFA that accepts the language L = (ab+aba)*

Non-Deterministic Finite Automata (NFA)_Example_1

- Construct an NDFA for the language
  $(ba)^* \cup (bab)^*$.

Solution,

We need to construct the NDFA as M= (Q, ∑, δ, q_0, F) where:

Q: Set of finite number of states.

∑ : Set of finite input symbols.

q_0 : Initial /Starting State ($q_0 \in Q$)

F: Set of Final States ($F \subseteq Q$)

δ : Transition Function of the form

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

Set of string that the DFA accepts are:

{ε, ba, bab, babbabbab, babbabbabbab, bababa……}

Non-Deterministic Finite Automata (NFA)_Example_1

- For this language L = (ba)* ∪ (bab)*, we can have the NFA as

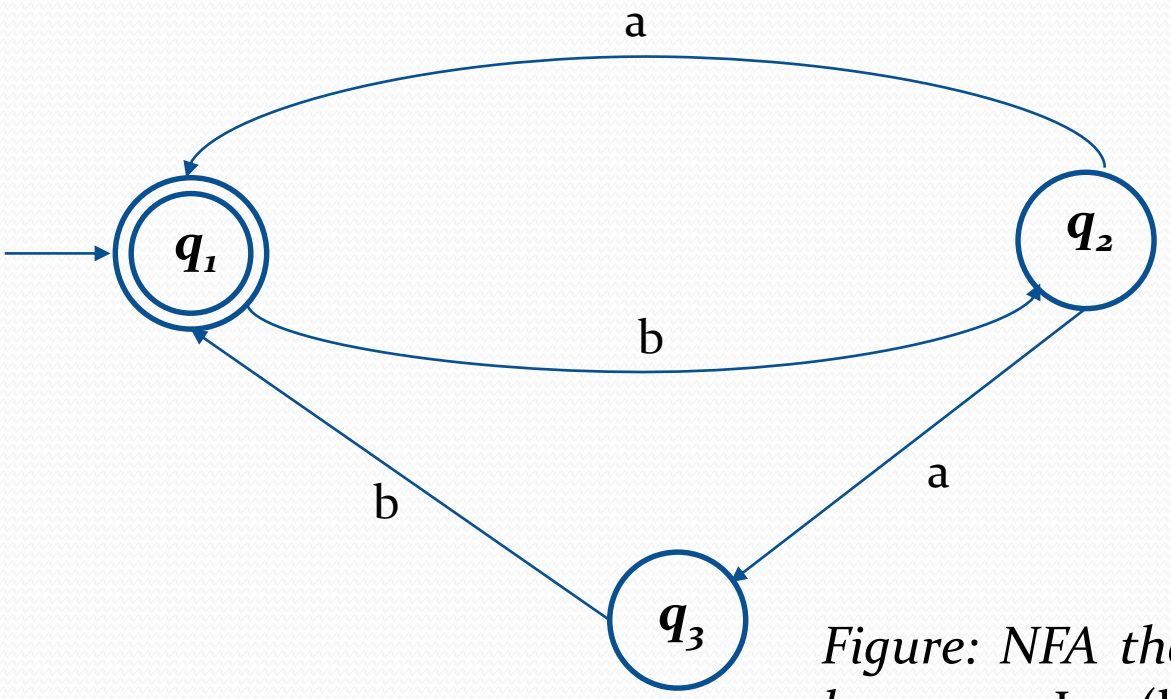


Figure: NFA that accepts the language L = (ba)* ∪ (bab)*

Non-Deterministic Finite Automata (NFA)_Example_1

- Consider a NFA as:

$M = (Q, \Sigma, \delta, q_o, F)$

$Q = \{q_1, q_2, q_3\}$

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

δ : Transition Function of the form $Q \times \Sigma \rightarrow 2^Q$ are:

$(q_1, a) \rightarrow \phi$

$(q_1, b) \rightarrow q_2$

$(q_2, a) \rightarrow \{q_1, q_3\}$

$(q_2, b) \rightarrow \phi$

$(q_3, a) \rightarrow \phi$

$(q_3, b) \rightarrow q_1$

$q_0 : q_1$

$F: q_1$

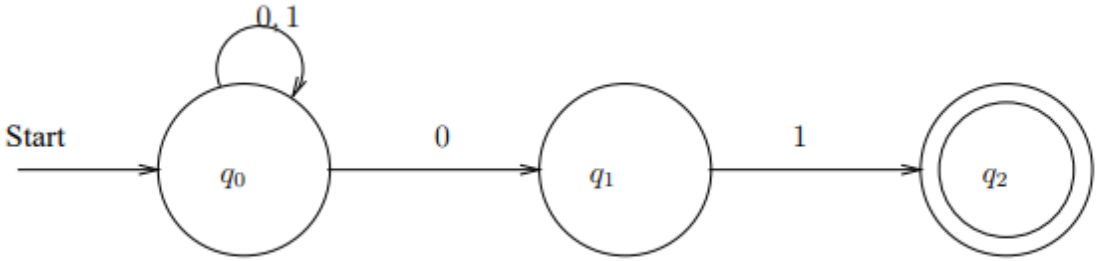
Non-Deterministic Finite Automata (NFA)

- Here we can see that the complex DFA can be described with the simple NFA.
- Non-deterministic finite automata can be a much more convenient device to design than a deterministic finite automata.
- Nondeterministic devices are not meant as realistic models of computers.
- They are simply a useful notational generalization of finite automata, as they can greatly simplify the description of these automata.

Extended Transition Function of NFA

- The extended transition function of NFA, denoted by $\hat{\delta}$ is a transition function that takes two arguments as input, a state $q \in Q$ & a string w and returns a set of states that the NFA is in, if it starts in q & processes the string w.
- It describes what happens when we start in any state and follow any sequence of inputs.

Extended Transition Function of NFA_Example



Processing $w = 00101$

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. $\hat{\delta}(q_0, 00101) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$

Equivalence of DFA and NFA

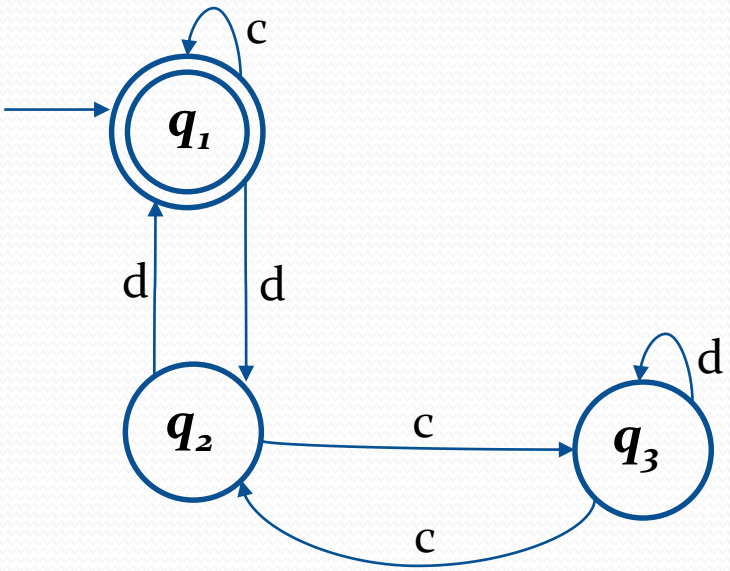
- The equivalence of two finite automata determines whether the two automata's are equal or not. i.e. the two automata perform the same function.
- The languages they accept will be same. So, in order to identify two automata are equivalent or not there are certain steps we need to follows.
- They are:

Equivalence of DFA and NFA

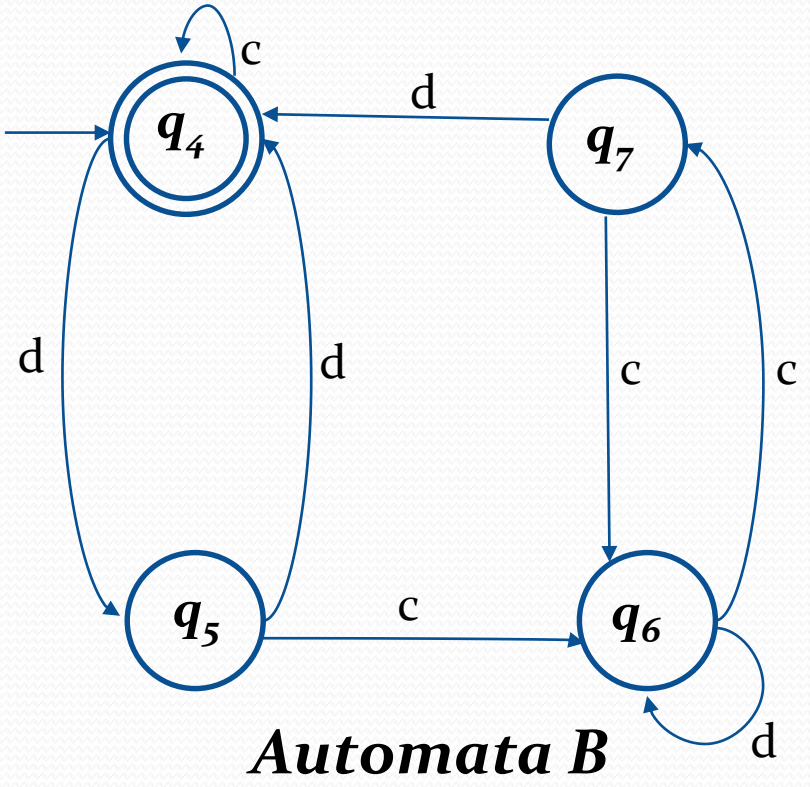
- *If initial state is final state of one automata, then in second automata also initial state must be final state for them to be equivalent.*
- *For any pair of states* $\{q_i, q_j\}$ *the transition for input* $\boldsymbol{a \in \Sigma}$ *is defined by* $\{q_a, q_b\}$ *where* $\delta(q_i, a) = q_a$ *and* $\delta(q_j, a) = q_b$. *The two automata are not equivalent of for a pair* $\{q_a, q_b\}$ *one is intermediate state and the other is final state.*

Equivalence of DFA and NFA

- *Example:*



Automata A



Automata B

Equivalence of DFA and NFA

- *In automata A, q_0 is the initial state as well as final state and In automata B also q_4 is the initial state as well as the final state so it satisfies our first condition.*
- *Now we check second condition, for that let we first construct pairs of states from two automata and check transition in the table as follows:-*

Equivalence of DFA and NFA

States	Inputs	
	c	d
$\{q_1,q_4\}$	$\{q_1,q_4\}$ (FS,FS)	$\{q_2,q_5\}$ (IS,IS)
$\{q_2,q_5\}$	$\{q_3,q_6\}$ (IS,IS)	$\{q_1,q_4\}$ (FS,FS)
$\{q_3,q_6\}$	$\{q_2,q_7\}$ (IS,IS)	$\{q_3,q_6\}$ (IS,IS)
$\{q_2,q_7\}$	$\{q_3,q_6\}$ (IS,IS)	$\{q_1,q_4\}$ (FS,FS)

IS: Intermediate State
FS: Final State

Equivalence of DFA and NFA

- Now we can see that all the pairs checked have transition in ***c*** and ***d*** both are either initial states or final states. This satisfies the second condition.
- Hence both automation A and B are equivalent.

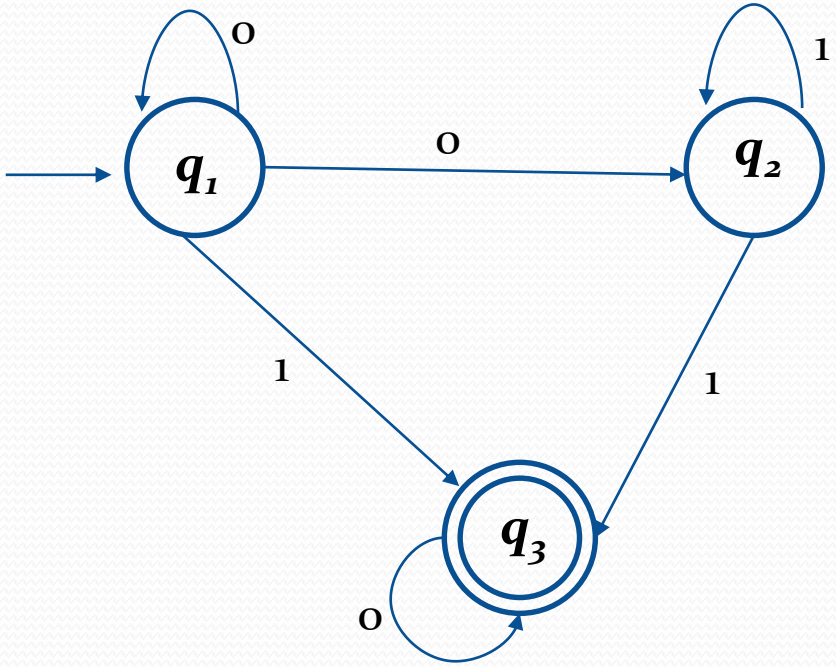
Note: While making or checking transition if we found two pairs are not same (i.e., both are not intermediate states or final states) then we stop there and conclude as not equivalent.

Conversion of NFA to DFA

- Remember that every DFA is an special NFA but not vice versa.
- But there is an equivalent DFA for every NFA.
- We use sub-set construction method to convert NFA into DFA
- Steps are:
  - Construct a transition table of given NFA.
  - Identify all the new states from the transition table.
  - Find the transition for each new states in term of input symbols.
  - The process is continued until transition for all the new states are identified.
  - Finally draw a transition diagram by using all states obtained.

Conversion of NFA to DFA_Example(1)

- Convert the following NFA into DFA



Conversion of NFA to DFA_Example(1)

- Solution,

The transition table of the given NFA is given by:

Q/Σ	0	1
→q_1	{q_1,q_2}	q_3
q_2	ϕ	{q_2,q_3}
*q_3	q_3	ϕ

Let δ' be the transition function of the DFA being constructed.

Conversion of NFA to DFA_Example(1)

Now we need to find the transition for all the states in the transition table for each input symbol.

Since q_1 is the initial state of given NFA so the starting state of DFA will be also q_1.

Step 1: Find the transition for initial state

$\delta'(\{q_1\},0) \rightarrow \delta(q_1,0) \rightarrow \{q_1,q_2\}$ ***(new state)***

$\delta'(\{q_1\},1) \rightarrow \delta(q_1,1) \rightarrow \{q_3\}$ ***(new state)***

Step 2: Find the transition for new state

$\delta'(\{q_1,q_2\},0)$

$\rightarrow \delta(q_1,0) \cup \delta(q_2,0)$

$\rightarrow \{q_1,q_2\} \cup \{\phi\}$

$\{q_1,q_2\}$

Conversion of NFA to DFA_Example(1)

$\delta'(\{q_1,q_2\},1)$

$\rightarrow \delta(q_1,1) \cup \delta(q_2,1)$

$\rightarrow \{q_3\} \cup \{q_2,q_3\}$

$\{q_2,q_3\}$ ***(new state)***

$\delta'(\{q_2,q_3\},0)$

$\rightarrow \delta(q_2,0) \cup \delta(q_3,0)$

$\rightarrow \{\phi\} \cup \{q_3\}$

$\{q_3\}$

Conversion of NFA to DFA_Example(1)

$\delta'(\{q_2,q_3\},1)$

$\rightarrow \delta(q_2,1) \cup \delta(q_3,1)$

$\rightarrow \{q_2,q_3\} \cup \{\phi\}$

$\{q_2,q_3\}$

$\delta'(\{q_3\},0) \rightarrow \delta(q_3,0) \rightarrow \{q_3\}$

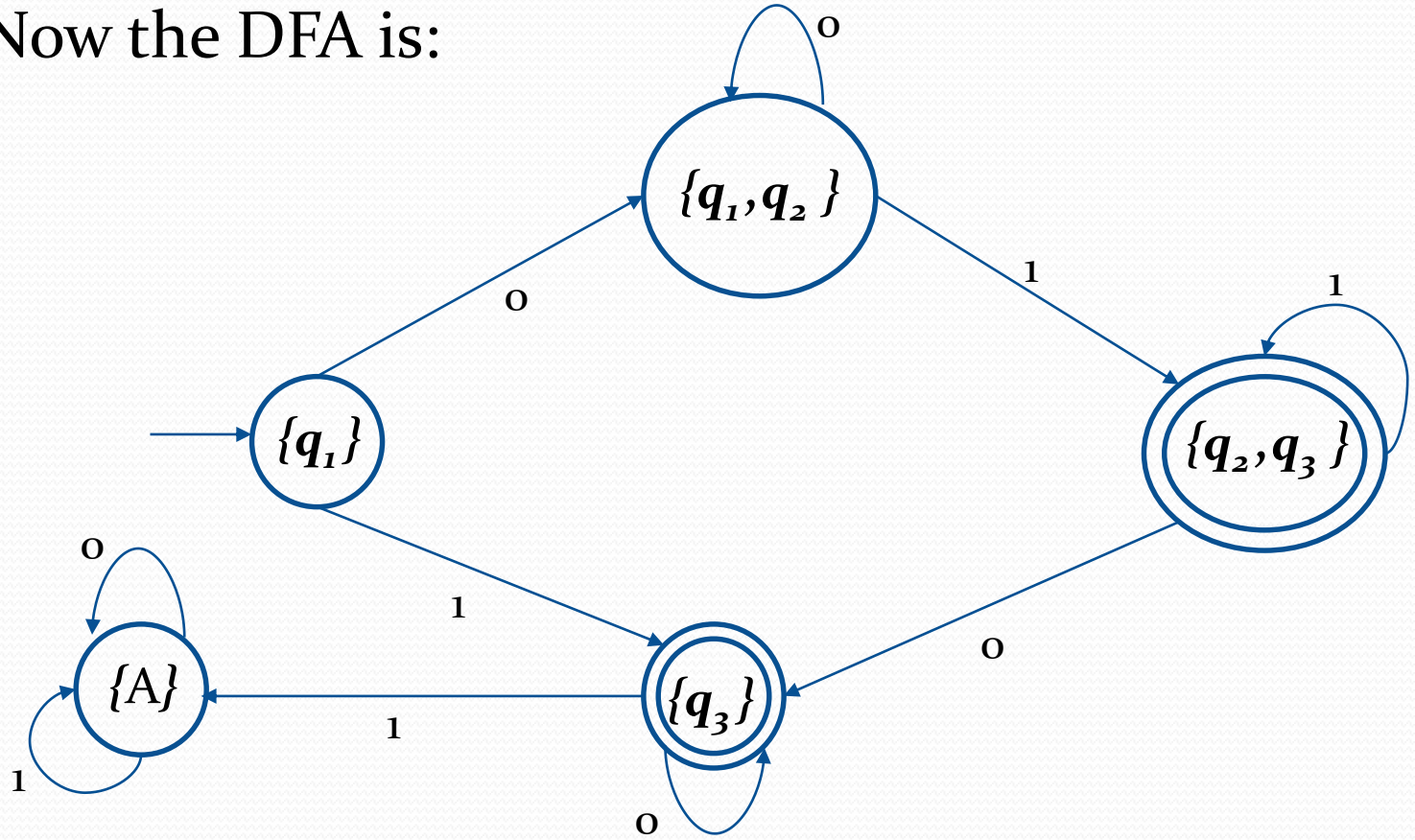
$\delta'(\{q_3\},1) \rightarrow \delta(q_3,1) \rightarrow \{\phi\}$ In DFA there cannot be empty state so we need to define the ***dead state.***

So,

$\boldsymbol{\delta'(\{q_3\},1) \rightarrow \delta(q_3,1) \rightarrow \{A\}}$ here A is dead state.

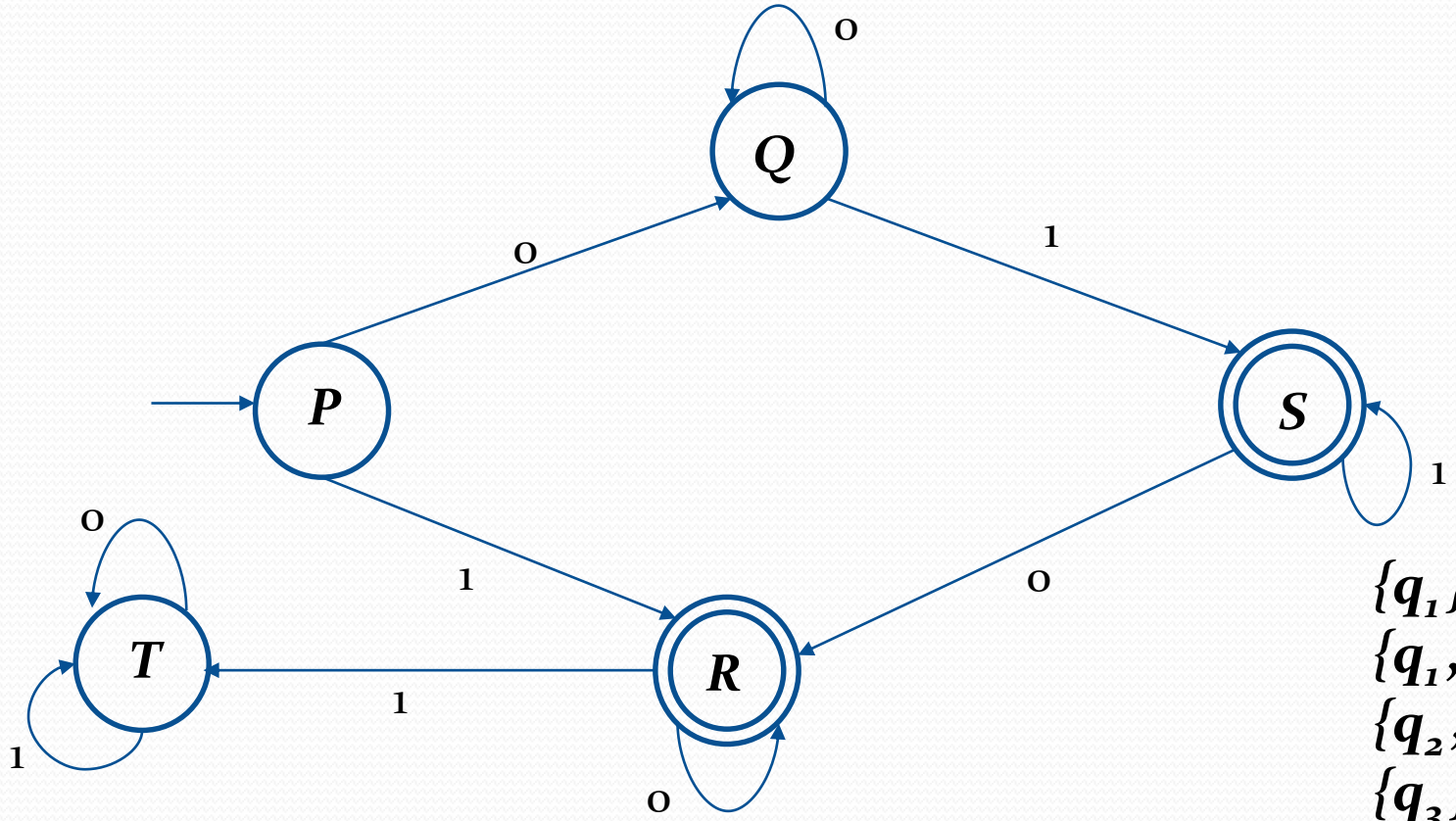
Conversion of NFA to DFA_Example(1)

Now the DFA is:



Conversion of NFA to DFA_Example(1)

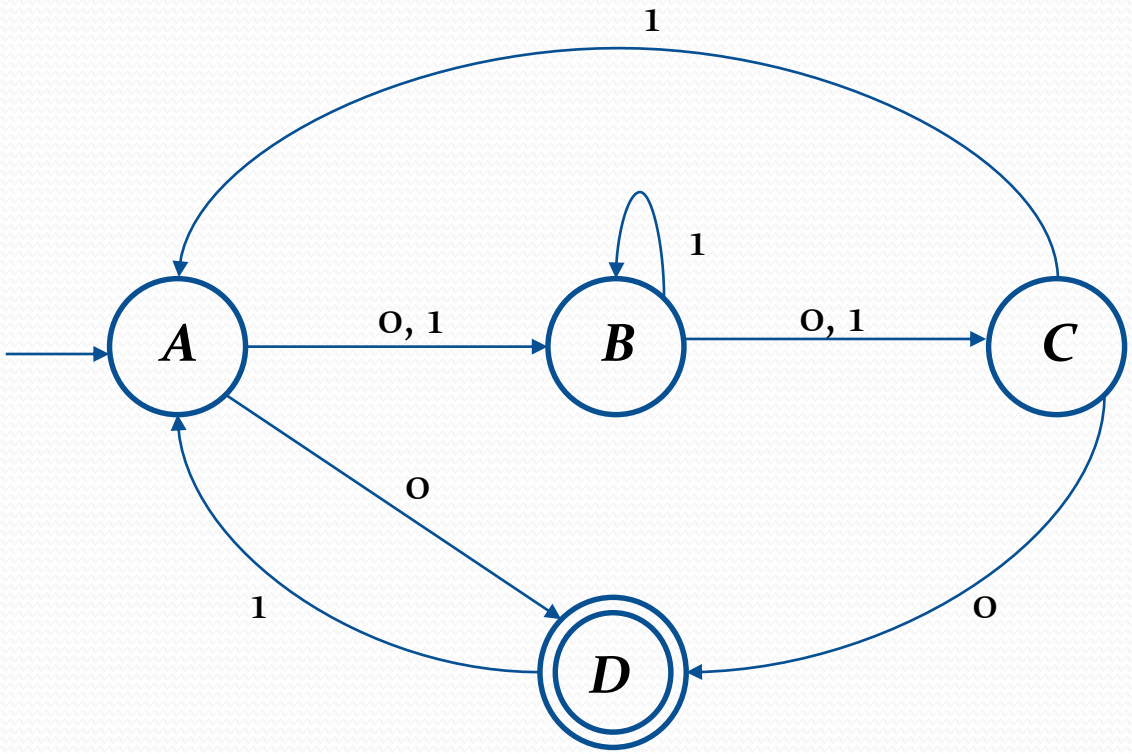
This DFA can be redrawn as :



$\{q_1\}=P$
$\{q_1,q_2\}=Q$
$\{q_2,q_3\}=S$
$\{q_3\}=R$
$\{A\}=T$

Conversion of NFA to DFA_Example(2)

- Convert the following NFA into DFA



Conversion of NFA to DFA_Example(2)

- Solution,

The transition table of the given NFA is given by:

Q/Σ	0	1
→A	{B,D}	B
B	C	{B,C}
C	D	A
*D	Φ	A

Let δ' be the transition function of the DFA being constructed.

Conversion of NFA to DFA_Example(2)

Now we need to find the transition for all the states in the transition table for each input symbol.

Since ***A*** is the initial state of given NFA so the starting state of DFA will be also ***A*.**

Step 1: Find the transition for initial state

$\delta'(\{A\},0) \rightarrow \delta(A,0) \rightarrow \{B,D\}$ ***(new state)***

$\delta'(\{A\},1) \rightarrow \delta(A,1) \rightarrow \{B\}$ ***(new state)***

Step 2: Find the transition for new state

$\delta'(\{B,D\},0)$

$\rightarrow \delta(B,0) \cup \delta(D,0)$

$\rightarrow \{C\} \cup \{\phi\}$

$\{C\}$ ***(new state)***

Conversion of NFA to DFA_Example(2)

$\delta'(\{B,D\},1)$

$\rightarrow \delta(B,1) \cup \delta(D,1)$

$\rightarrow \{B,C\} \cup \{A\}$

$\{A,B,C\}$ ***(new state)***

$\delta'(\{B\},0) \rightarrow \delta(B,0) \rightarrow \{C\}$

$\delta'(\{B\},1) \rightarrow \delta(B,1) \rightarrow \{B,C\}$ ***(new state)***

$\delta'(\{C\},0) \rightarrow \delta(C,0) \rightarrow \{D\}$ ***(new state)***

$\delta'(\{C\},1) \rightarrow \delta(C,1) \rightarrow \{A\}$ ***(old state)***

Conversion of NFA to DFA_Example(2)

δ' ({A,B,C},0)

$\rightarrow \delta$ (A,0) $\cup \delta$ (B,0) $\cup \delta$ (C,0)

$\rightarrow$ {B,D} $\cup$ {C} $\cup$ {D}

{B,C,D} ***(new state)***

δ' ({A,B,C},1)

$\rightarrow \delta$ (A,1) $\cup \delta$ (B,1) $\cup \delta$ (C,1)

$\rightarrow$ {B} $\cup$ {B,C} $\cup$ {A}

{A,B,C} ***(old state)***

Conversion of NFA to DFA_Example(2)

$\delta'(\{B,C\},0)$

$\rightarrow \delta(B,0) \cup \delta(C,0)$

$\rightarrow \{C\} \cup \{D\}$

$\{C,D\}$ ***(new state)***

$\delta'(\{B,C\},1)$

$\rightarrow \delta(B,1) \cup \delta(C,1)$

$\rightarrow \{B,C\} \cup \{A\}$

$\{A,B,C\}$ ***(old state)***

Conversion of NFA to DFA_Example(2)

$\delta'(\{D\},0) \rightarrow \delta(D,0) \rightarrow \{\phi\}$ In DFA there cannot be empty state so we need to define the ***dead state.***

So,

$\delta'(\{D\},0) \rightarrow \delta(D,1) \rightarrow \{Q\}$ here Q is dead state.

$\delta'(\{D\},1) \rightarrow \delta(D,1) \rightarrow \{A\}$ ***(old state)***

$\delta'(\{B,C,D\},0)$

$\rightarrow \delta(B,0) \cup \delta(C,0) \cup \delta(D,0)$

$\rightarrow \{C\} \cup \{D\} \cup \{\phi\}$

$\{C,D\}$

Conversion of NFA to DFA_Example(2)

δ' ({B,C,D},1)

$\rightarrow \delta$ (B,1) $\cup \delta$ (C,1) $\cup \delta$ (D,1)

$\rightarrow$ {B,C} $\cup$ {A} $\cup$ {A}

{A,B,C} ***(old state)***

δ' ({C,D},0)

$\rightarrow \delta$ (C,0) $\cup \delta$ (D,0)

$\rightarrow$ {D} $\cup$ {ϕ}

{D} ***(old state)***

δ' ({C,D},1)

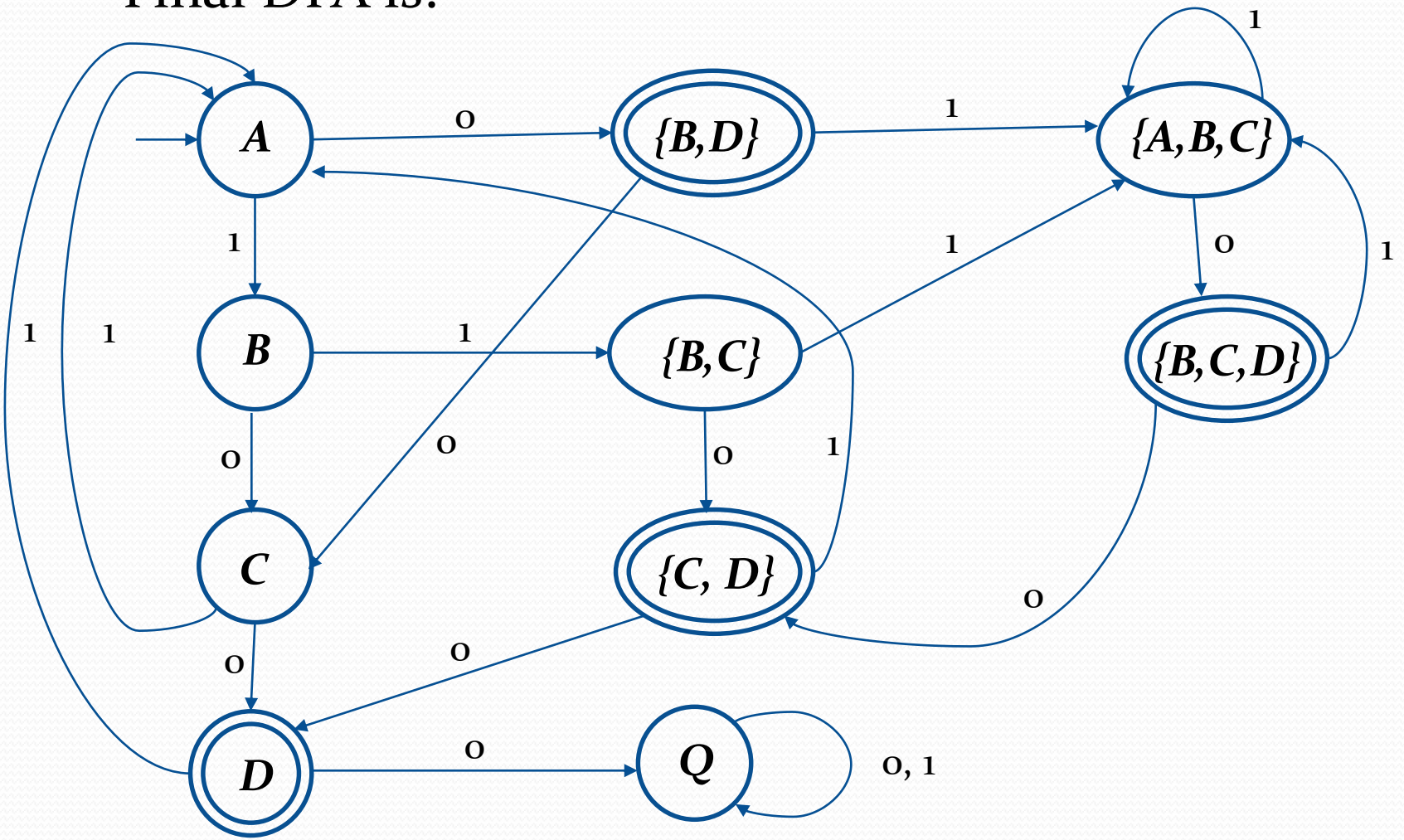
$\rightarrow \delta$ (C,1) $\cup \delta$ (D,1)

$\rightarrow$ {A} $\cup$ {A}

{A} ***(old state)***

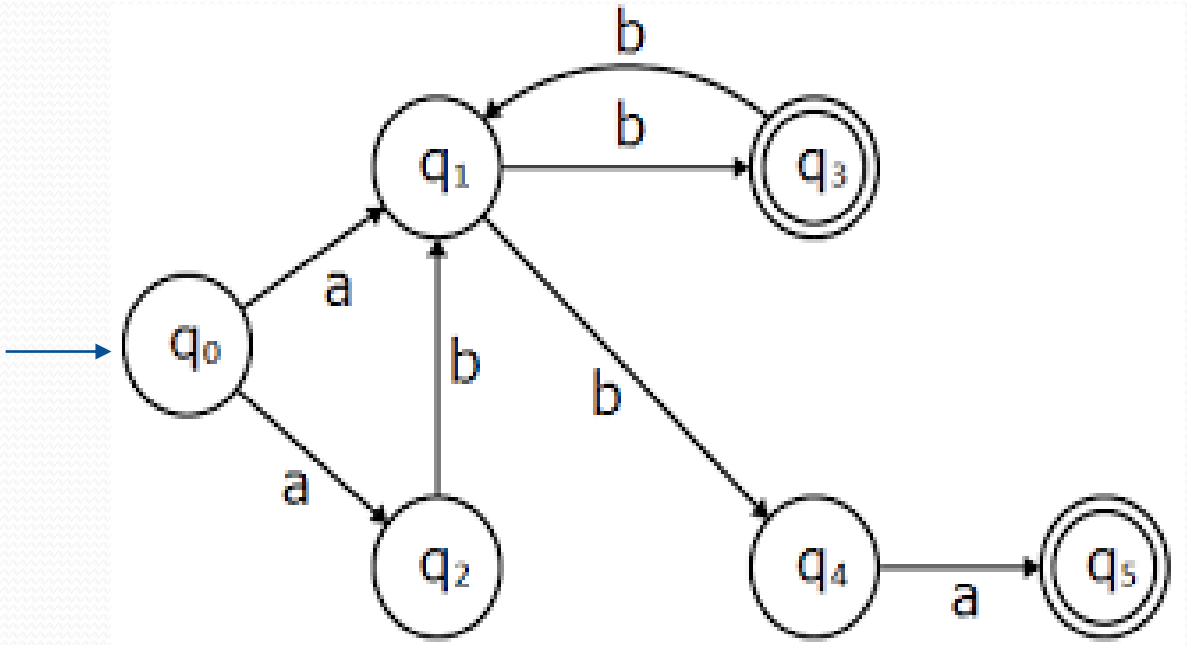
Conversion of NFA to DFA_Example(2)

- Final DFA is:



Conversion of NFA to DFA_Example(3)

- Convert the following NFA into DFA



Try Yourself

Minimization of DFA

- We can have different DFA for the same problem with varying in the number of states.
- But the DFA, with lesser number of states is considered to be the best.
- So, we try to reduce the number of states in the DFA while maintaining its capability to recognize the given language.
- This procedure is known as minimization of DFA.

Minimization of DFA

- Minimization of DFA is required to obtain the minimal version of any DFA which consists of the minimum number of states possible.
- Let we are designing a DFA, it can be designed with variable states by different designer which can perform the same operation and are equivalent.
- Obviously DFA with less states among the different DFA will be better with performance.
- Hence, minimization of DFA is required.

Minimization of DFA

Steps for minimizing of DFA:

- We use the equivalence approach for minimizing DFA i.e. we create the series of equivalence.
  - ***Step 1:*** Draw the transition table of given DFA.
  - ***Step 2:*** Divide the set of states into two sets such that one set will contain all final states and other set will contain non-final states. ***(called as 0-equivalence)***.
  - ***Step 3:*** For ***1-equivalence*** take sets more than one state from ***0-equivalence*** then check if they are ***1- equivalence*** to one another or not. If equivalent then keep in same set otherwise in separate set.

Minimization of DFA

- Two states (q_i, q_j) are in ***1-equivalence*** if for any input symbol a, $\delta(q_i, a)$ and $\delta(q_j, a)$ generates state A and B. If these state A and B are in same sets in ***0-equivalence.***
- Then q_i ***and*** q_j can be kept in same set otherwise not.
- Similarly proceed for all except single sets.

- ***Step 4:*** For ***2-equivalence*** take sets more than one state from ***1-equivalence*** then check if they are ***2-equivalence*** to one another or not. If equivalent then keep in same set otherwise in separate set.

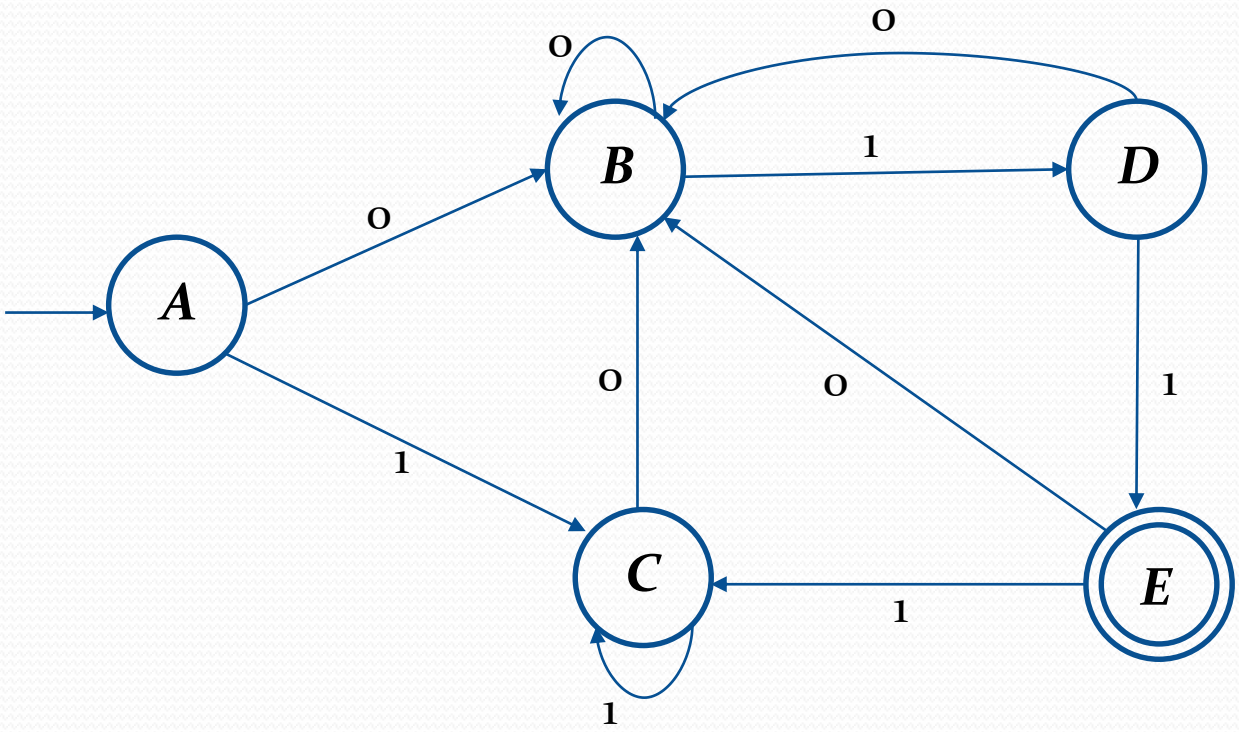
Minimization of DFA

- Two states (q_i, q_j) are in ***2-equivalence*** if for any input symbol a, $\delta(q_i, a)$ and $\delta(q_j, a)$ generates state A and B. If these state A and B are in same sets in ***1-equivalence.***
- Then q_i ***and*** q_j can be kept in same set otherwise not.
- Similarly proceed for all except single sets.

- ***Step 5:*** Repeat until two successive ***equivalence*** gives the same result.

Minimization of DFA_Example(1)

- Minimize the given DFA



Minimization of DFA_Example(1)

Solution,

Construct the transition table for the given DFA

Q/Σ	0	1
→A	B	C
B	B	D
C	B	C
D	B	E
*E	B	C

Minimization of DFA_Example(1)

Now based on the transition table we create the series of equivalence

0-equivalence-Here we create the separate set of the states i.e. one set containing the non-final states and other set containing the final states

0-equivalence: {A,B,C,D} and {E}.

1-equivalence- Here we take two state from ***0-equivalence*** and check for their equivalency according to input symbol.

For state A and B: A on input 0 transits to B

B on input 0 transits to B

A on input 1 transits to C

B on input 1 transits to D

Minimization of DFA_Example(1)

The transition set of states are {B} and {C,D}.

These {B} and {C,D} are in the same set in the ***0-equivalence*** so in ***1-equivalence*** state A and B will be on same set.

For state A and C: A on input 0 transits to B

C on input 0 transits to B

A on input 1 transits to C

C on input 1 transits to C

The transition set of states are {B} and {C}.

These {B} and {C} are in the same set in the ***0-equivalence*** so in ***1-equivalence*** state A and C will be on same set.

Minimization of DFA_Example(1)

For state A and D: A on input 0 transits to B

D on input 0 transits to B

A on input 1 transits to C

D on input 1 transits to E

The transition set of states are {B} and {C,E}.

Here {C,E} are not in the same set in the ***0-equivalence*** so in ***1-equivalence*** state A and D will not be on same set.

So, final ***1-equivalence: {A,B,C}, {D} and {E}.***

Minimization of DFA_Example(1)

2-equivalence- Here we take two state from ***1-equivalence*** and check for their equivalency according to input symbol.

For state A and B: A on input 0 transits to B

B on input 0 transits to B

A on input 1 transits to C

B on input 1 transits to D

The transition set of states are {B} and {C,D}.

Here {C,D} are not in the same set in the ***1-equivalence*** so in ***2-equivalence*** state A and B will not be on same set.

Minimization of DFA_Example(1)

For state A and C: A on input 0 transits to B

C on input 0 transits to B

A on input 1 transits to C

C on input 1 transits to C

The transition set of states are {B} and {C}.

These {B} and {C} are in the same set in the ***1-equivalence*** so in ***2-equivalence*** state A and C will be on same set.

So, final ***2-equivalence: {A,C},{B}, {D} and {E}.***

Minimization of DFA_Example(1)

3-equivalence- Here we take two state from ***2-equivalence*** and check for their equivalency according to input symbol.

For state A and C: A on input 0 transits to B

C on input 0 transits to B

A on input 1 transits to C

C on input 1 transits to C

The transition set of states are {B} and {C}.

Here {B} and {C} are in the same set in the ***2-equivalence*** so in ***3-equivalence*** state A and C will not be on same set.

So, final ***3-equivalence: {A,C},{B}, {D} and {E}.***

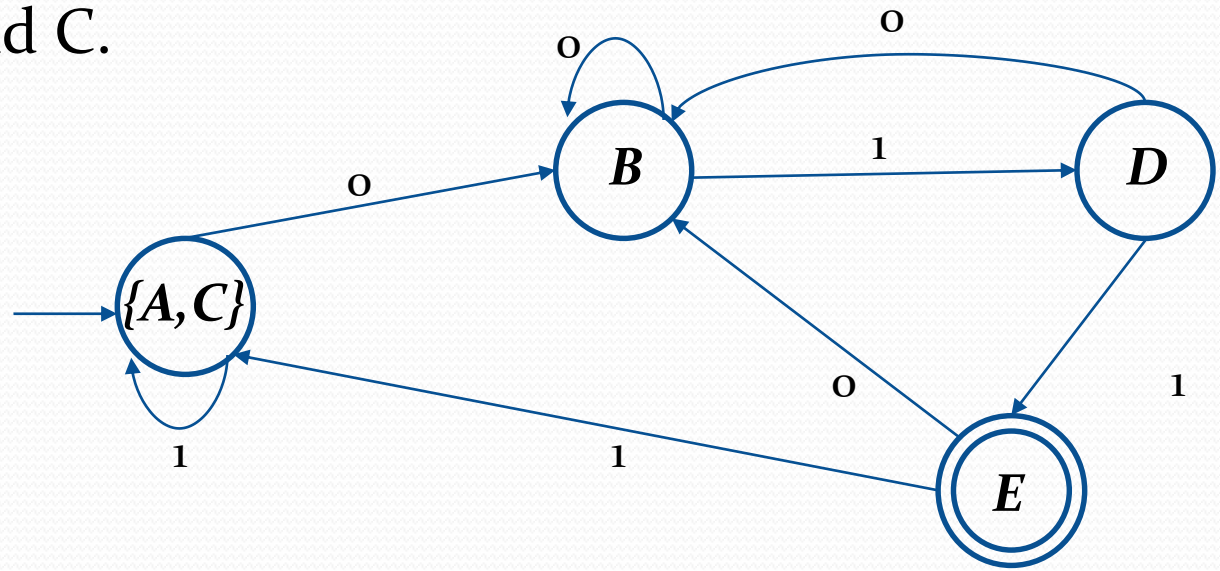
Minimization of DFA_Example(1)

Now we found that, two successive equivalence gives the same result.

i.e. 2-equivalence and 3-equivalence are same.

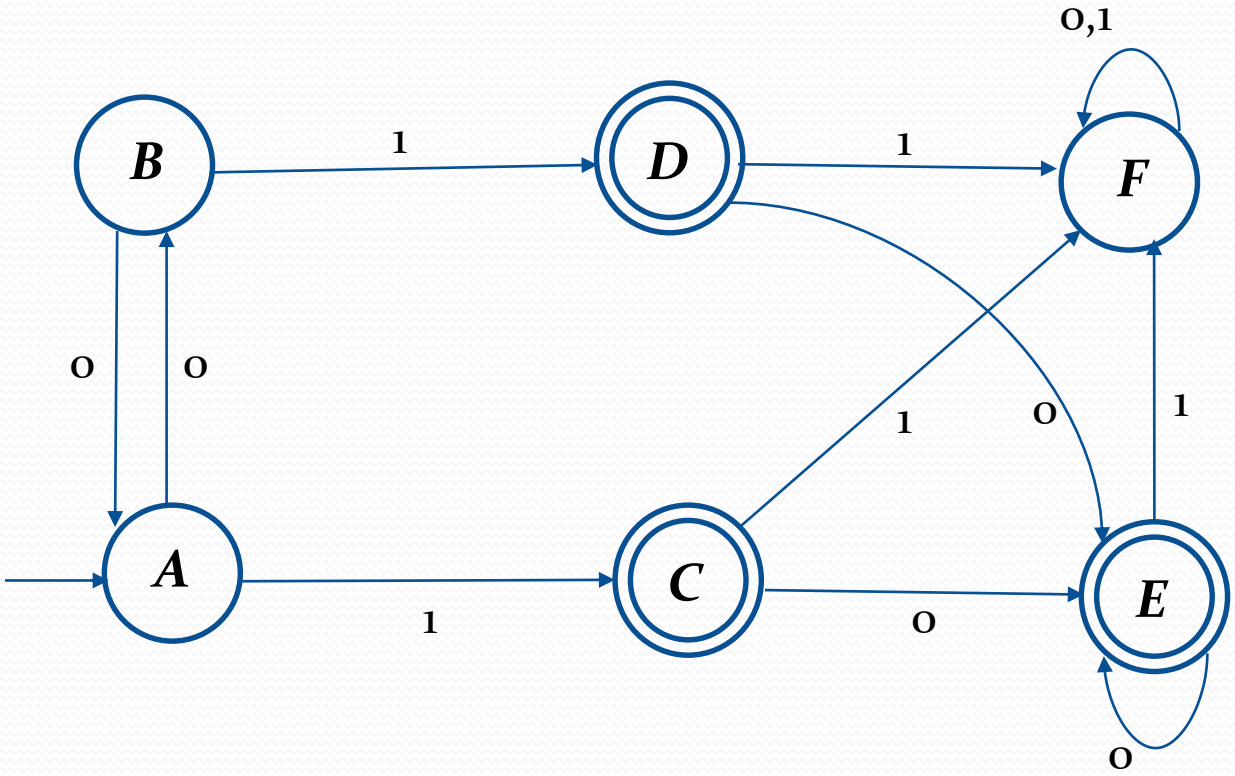
{A,C},{B}, {D} and {E}.

So, we stop here and construct minimized DFA by combining state A and C.



Minimization of DFA_Example(2)

- Minimize the given DFA using equivalence principle



ε-Non-Deterministic Finite Automata (ε -NFA)

- If there is a transition for ***ε (empty input)*** then the automata is called ε-NFA.
- This is another extension of finite automata that allows a transition on ***ε, the empty string*** which means it can switches to new states without reading an input symbol.
- This capability does not expand the class of languages that can be accepted by finite automata, but it does give some added "programming convenience".

ε-Non-Deterministic Finite Automata (ε -NFA)

- Formally, ε-NFA is defined by five tuples (similar to DFA and NFA):

$$M = (Q, \Sigma, \delta, q_o, F) \textbf{ where:}$$

Q: Set of finite number of states.

Σ : Set of finite input symbols.

q_o : Initial /Starting State ($q_o \in Q$)

F: Set of Final States ($F \subseteq Q$)

δ : Transition Function of the form

$$Q \times \Sigma \cup \{\varepsilon\} \rightarrow 2^Q$$

ε-Non-Deterministic Finite Automata (ε -NFA)

- ε-NFA can be shown as:

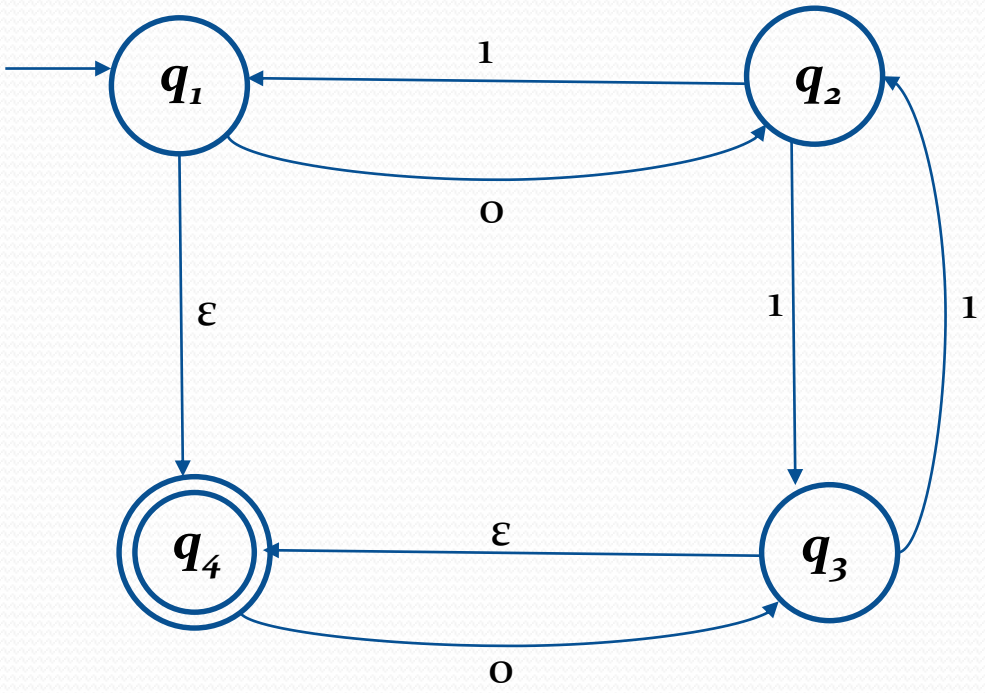


Figure: ε-NFA

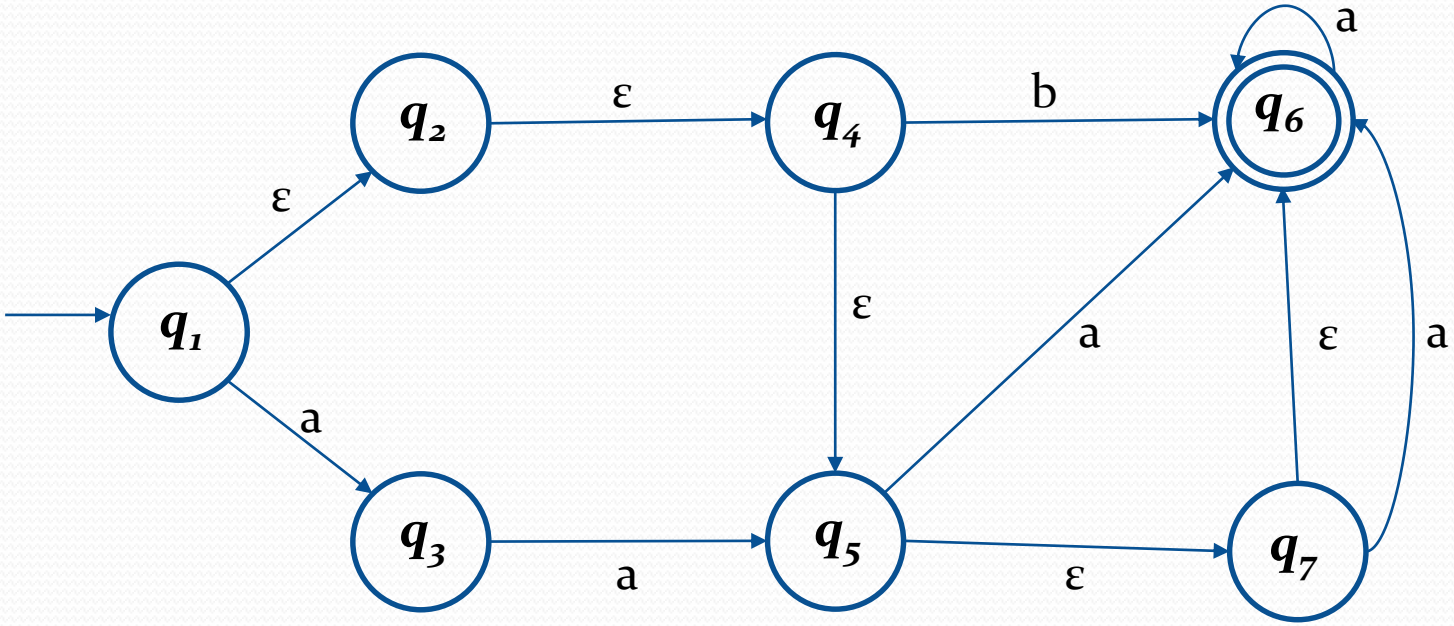
ε-Closure of a State (ε*)

- Epsilon closure of a state ***q*** is the set that contains the state ***q*** itself and all other states that can be reached from state ***q*** by following ***ε transitions***.
- Epsilon closure of a state ***q*** denoted by ***ε-closure (q)*** is the set of states that can be reached without reading any input symbol from state ***q***.
- To obtain ***ε-closure (q)***, follow all transitions out of ***q*** that are labelled ***ε***.
- After we get to another state by following ***ε***, follow the ***ε***-transitions out of those states & so on, eventually finding every state that can be reached from ***q*** along any path whose arcs are all labeled.

ε-Closure of a State (ε*)

- Formally, we can define ***ε***-closure of the state ***q*** as;
  - Basis: state ***q*** is in ***ε-closure (q)***.
  - Induction: If state ***p*** is reached with ***ε-transition*** from state ***q***, ***p*** is in ***ε-closure (q)*** and if there is an arc from ***p*** to ***r*** labeled ***ε***, then ***r*** is in ***ε-closure (q)*** and so on.

ε-Closure of a State (ε*)



ε-closure (q_1)={q_1,q_2,q_4,q_5,q_7,q_6}
ε-closure (q_2)={q_2,q_4,q_5,q_7,q_6}
ε-closure (q_3)={q_3}
ε-closure (q_4)={q_4,q_5,q_7,q_6}
ε-closure (q_5)={q_5,q_7,q_6}

ε-closure (q_6)={q_6}
ε-closure (q_7)={q_7,q_6}

(ε -NFA) Representation

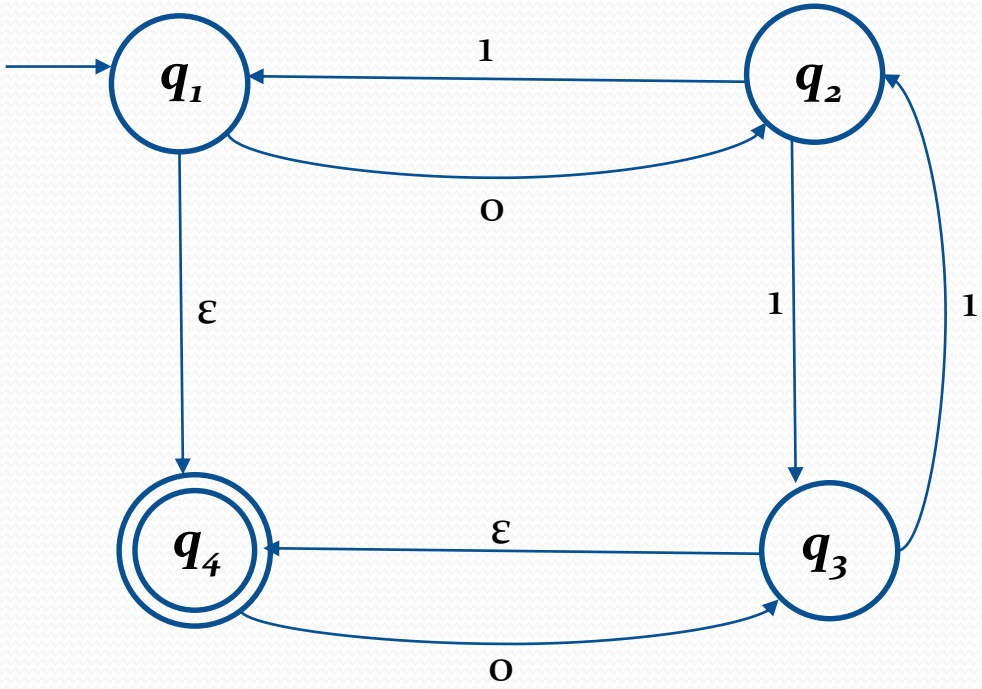


Figure: ε-NFA

(ε -NFA) Representation

- Given ε - NFA as can be defined as:

$M = (Q, \Sigma, \delta, q_0, F)$

$Q = \{q_1, q_2, q_3, q_4\}$

$\Sigma = \{0, 1\}$

δ : Transition Function of the form $Q \times \Sigma \rightarrow 2^Q$ are:

$(q_1, 0) \rightarrow q_2$
$(q_1, 1) \rightarrow \phi$
$(q_1, \varepsilon) \rightarrow \{q_1, q_4\}$
$(q_2, 0) \rightarrow \phi$

$(q_2, 1) \rightarrow \{q_1, q_3\}$
$(q_2, \varepsilon) \rightarrow q_2$
$(q_3, 0) \rightarrow \phi$
$(q_3, 1) \rightarrow q_2$

$(q_3, \varepsilon) \rightarrow \{q_3, q_4\}$
$(q_4, 0) \rightarrow q_3$
$(q_4, 1) \rightarrow \phi$
$(q_4, \varepsilon) \rightarrow q_4$

$q_0 : q_1$

$F: q_4$

(ε -NFA) Representation

- A Transition Table of given ε-NFA

Q/Σ	0	1	ε
q_1	q_2	ϕ	$\{q_1, q_4\}$
q_2	ϕ	$\{q_1, q_3\}$	q_2
Q_3	ϕ	q_2	$\{q_3, q_4\}$
* q_4	q_3	ϕ	q_4

- *Keep in mind that all states contains **ε** and every state on getting ε as input goes to itself. i.e. if **ε** is not given then then goes to itself.*
- *If **ε** is given then transition will be on itself as well as other state where transition is given*

Extended Transition Function of (ε -NFA)

- The extended transition function of ε -NFA denoted by $\hat{\delta}$ is defined by;
  - Basis Step: $\hat{\delta}(q, \varepsilon) = \varepsilon$ -closure (q)
  - Inductive Step: Let w = xa be a string, where x is substring of w without last symbol a and a $\in \sum$ but a $\neq \varepsilon$
- Let $\hat{\delta}$ (q, x)= $\{p_1, p_2, p_3 \ldots\ldots\ldots p_k\}$ i.e. p_i's are the states that can be reached from q following path labeled x which can end with many ε & can have many ε.
- Also let,

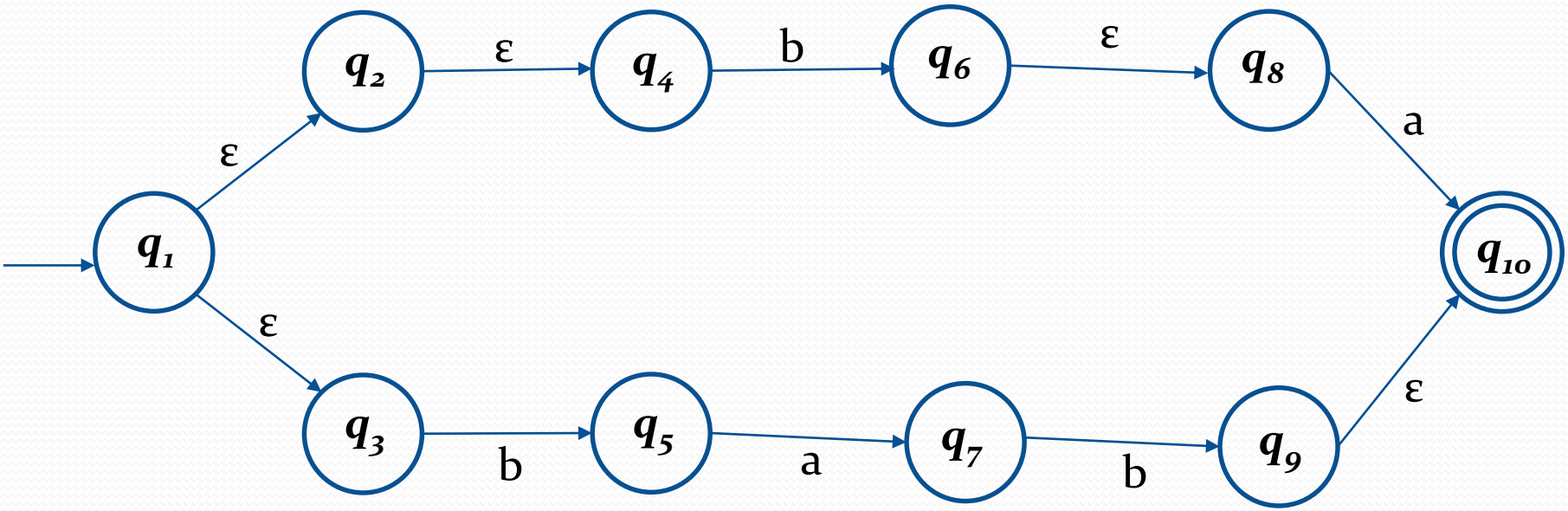
$$\bigcup_{i=1}^{k} \delta(pi, a) = \{r_1, r_2, \ldots\ldots\ldots r_m\}$$

Then, $\delta(q, x) = \bigcup_{j=1}^{m} \varepsilon$ –closure (r_j)

Extended Transition Function of (ε -NFA)

- To compute $\hat{\delta}$ (q, w), first find the ε -closure of starting state.
- Then read the first symbol of w and determine the states of NFA after reading first symbol and take the union of ε -closure of these states.
- Again read the second symbol and determine the states of NFA after reading second symbol from the union of ε -closure of states found in previous step.
- Continue this process until the last symbol is read and finally compute the ε -closure of states that are reached after reading last symbol.

Extended Transition Function of (ε -NFA)_Example



Find the extended transition for string ***ba***.

Extended Transition Function of (ε -NFA)_Example

- Step-1:

  Compute the extended transition for starting state.

  i.e. $\hat{\delta}(q_1, \varepsilon) = \varepsilon\text{-closure}(q_1) = \{q_1, q_2, q_3, q_4\}$

- Step-2: Compute the transition for ***b*** as:

  $\delta(q_1, b) \cup \delta(q_2, b) \cup \delta(q_3, b) \cup \delta(q_4, b)$

  $= \phi \cup \phi \cup \{q_5\} \cup \{q_6\}$

  $= \{q_5, q_6\}$

Compute the extended transition for the newly obtained state $\{q_5, q_6\}$.

Extended Transition Function of (ε -NFA)_Example

The extended transition for $\{q_5,q_6\}$ is:

i.e. $\hat{\delta}(q_5, \varepsilon) \cup \hat{\delta}(q_6, \varepsilon)$= ε-closure$(q_5) \cup$ ε-closure(q_6)

$= \{q_5\} \cup \{q_6,q_8\}$

$= \{q_5,q_6,q_8\}$

Now Compute the transition for ***a*** as:

$\delta(q_5, a) \cup \delta(q_6, a) \cup \delta(q_8, a)$

$= \{q_7\} \cup \phi \cup \{q_{10}\}$

$=\{q_7,q_{10}\}$

Here the final set of state i.e. $\{q_7,q_{10}\}$ contains one of the final state so the string ***ba*** is accepted.

Removing ε-Transition Function from ε -NFA

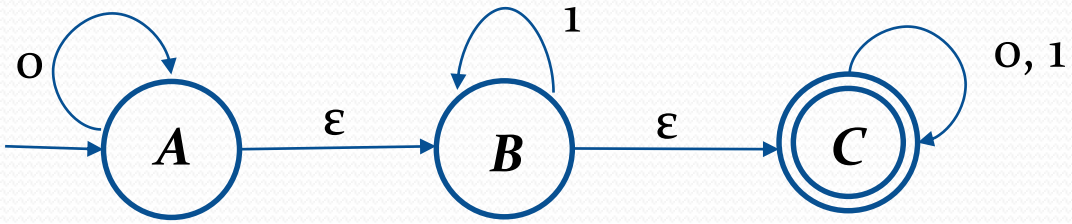
- This process is also considered as the conversion of ε – NFA to NFA.
- We use the concept of ***ε-closure*** to remove the ***ε-transition.***
- The procedure for converting any ***ε*** -NFA to its equivalent NFA is as follows:
  - ***Step-1:***
    - For each state we have to check that where does the state goes on ***ε****. (***ε***** is the set of all states that can be reached from a particular state only by seeing the ***ε*** symbol)

Removing ε-Transition Function from ε -NFA

- ***Step-2:***
  - Now set of states that we got in step-1, have to be checked on which state they go on getting a particular input.
- ***Step-3:***
  - Again the set of states that we got in step-2 are again checked on to which state do they go on ε^* again.

Removing ε-Transition Function from ε -NFA

Example



Here we are removing ε-Transition from above ε-NFA i.e. converting ε –NFA to NFA.

starting state = {A}

input symbol are {0, 1}

Removing ε-Transition Function from ε -NFA

Let, δ_N be the transition function of NFA.

Now we process for each state and input as follows:-

$\delta_N(A, o) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(A),o)$

$= \varepsilon\text{-closure}(\delta(\{A,B,C\},o)$ → $\delta(A,o) \cup \delta(B,o) \cup \delta(C,o)$
$\{A\} \cup \phi \cup \{C\} = \{A,C\}$

$= \varepsilon\text{-closure}(\{A,C\})$

$= \varepsilon\text{-closure}(A) \cup \varepsilon\text{-closure}(C)$

$= \{A,B,C\} \cup \{C\}$

$= \{A,B,C\}$

Removing ε-Transition Function from ε -NFA

$$\delta_N(A, 1) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(A),1)$$
$$= \varepsilon\text{-closure}(\delta(\{A,B,C\},1)$$
$$= \varepsilon\text{-closure}(\{B,C\})$$
$$= \varepsilon\text{-closure}(B) \cup \varepsilon\text{-closure}(C)$$
$$= \{B,C\} \cup \{C\}$$
$$= \{B,C\}$$

$\delta(A,1) \cup \delta(B,1) \cup \delta(C,)$
$\phi \cup \{B\} \cup \{C\} = \{B,C\}$

Removing ε-Transition Function from ε -NFA

$\delta_N(B, 0)$ = ε-closure(δ(ε-closure(B),0)

= ε-closure(δ({B,C},0)

= ε-closure(C)

= {C}

δ(B,0) ∪ δ(C,0)
ϕ ∪{C}= {C}

$\delta_N(B, 1)$ = ε-closure(δ(ε-closure(B),1)

= ε-closure(δ({B,C},1)

= ε-closure({B, C})

= ε-closure(B) ∪ ε-closure(C)

= {B, C} ∪ {C}

= {B, C}

δ(B, 1) ∪ δ(C, 1)
{B} ∪{C}= {B, C}

Removing ε-Transition Function from ε -NFA

$\delta_N(C, 0)$ = ε-closure(δ(ε-closure(C),0)

= ε-closure(δ(C,0))

= ε-closure(C)

= {C}

$\delta_N(C, 1)$ = ε-closure(δ(ε-closure(C),1)

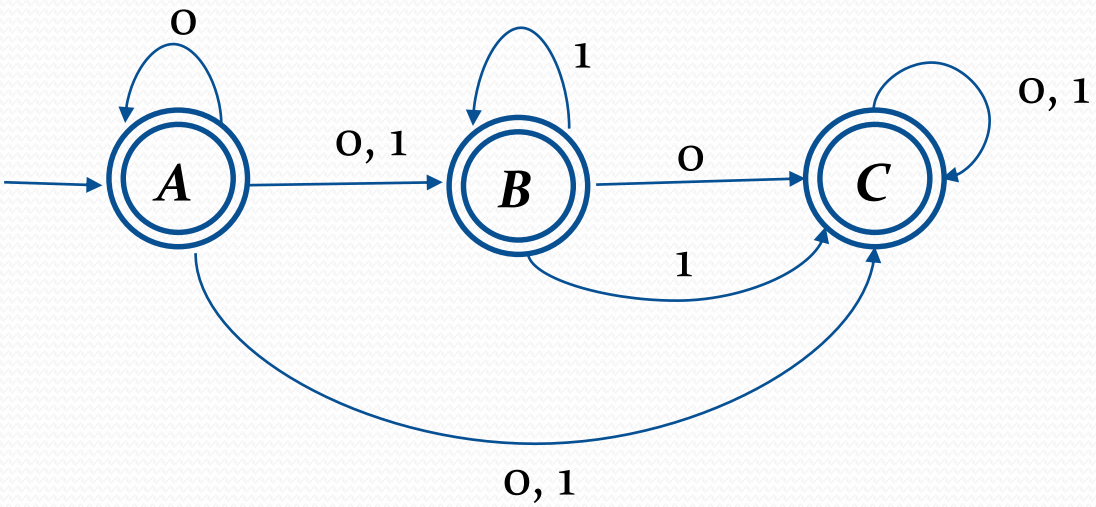
= ε-closure(δ(C, 1)

= ε-closure(C)

= {C}

Removing ε-Transition Function from ε -NFA

Final NFA



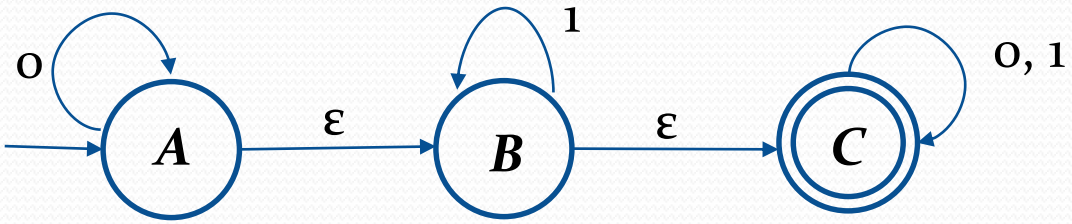
Here A, B and C all are final states in NFA because the final states are those state from where we can reach to the final state of ε –NFA only by ε – input.

Converting ε –NFA to DFA

- Conversion process of ε –NFA to its equivalent DFA is almost similar to converting equivalent to NFA.
- The difference is while converting ε -NFA to NFA we use same start state as it is in ε -NFA but, now for converting ε -NFA to DFA, we will use ε -closure of start state in ε -NFA.
- Also ***step_1 of*** ε –NFA to NFA conversion is ignored ***i.e.*** for each state there is no need to check where does the state transits on ε^* while converting to DFA.

Converting ε –NFA to DFA

Example



Here we are converting ε-NFA to DFA

starting state = ε-closure(A), since A is starting state of ε-NFA

={A, B, C}

input symbol are {0, 1}

Converting ε –NFA to DFA

Let, δ_D be the transition function of DFA.

Now we process for ={A, B, C} and input as follows:-

$\delta_D(\{A,B,C\}\ o) = \varepsilon\text{-closure}(\delta(\{A,B,C\},\ o)$

$= \varepsilon\text{-closure}(\{A,C\})$

$= \varepsilon\text{-closure}(A) \cup \varepsilon\text{-closure}(C)$

$= \{A,B,C\} \cup \{C\}$

$= \{A,B,C\}$

$\delta(A,o) \cup \delta(B,o) \cup \delta(C,o)$
$\{A\} \cup \phi \cup \{C\} = \{A,C\}$

Converting ε –NFA to DFA

$\delta_D(\{A,B,C\}\ 1) = \varepsilon\text{-closure}(\delta(\{A,B,C\}, 1)$

$= \varepsilon\text{-closure}(\{B,C\})$

$= \varepsilon\text{-closure}(B) \cup \varepsilon\text{-closure}(C)$

$= \{B,C\} \cup \{C\}$

$= \{B,C\}$

$\delta(A,1) \cup \delta(B,1) \cup \delta(C,1)$

$\phi \cup \{B\} \cup \{C\} = \{B,C\}$

Converting ε –NFA to DFA

$\delta_D(\{B,C\}\ 0) = \varepsilon\text{-closure}(\delta(\{B,C\}, 0)$

$= \varepsilon\text{-closure}(C)$

$= \{C\}$

$\delta(B,0) \cup \delta(C,0)$

$\phi \cup \{C\} = \{C\}$

$\delta_D(\{B,C\}\ 1) = \varepsilon\text{-closure}(\delta(\{B,C\}, 1)$

$= \varepsilon\text{-closure}(\{B,C\})$

$= \varepsilon\text{-closure}(B) \cup \varepsilon\text{-closure}(C)$

$= \{B, C\} \cup \{C\}$

$= \{B,C\}$

$\delta(B,1) \cup \delta(C,1)$

$\{B\} \cup \{C\} = \{B,C\}$

Converting ε –NFA to DFA

$\delta_D(C, 0) = \varepsilon\text{-closure}(\delta(C),0)$

$= \varepsilon\text{-closure}(C)$

$= \{C\}$

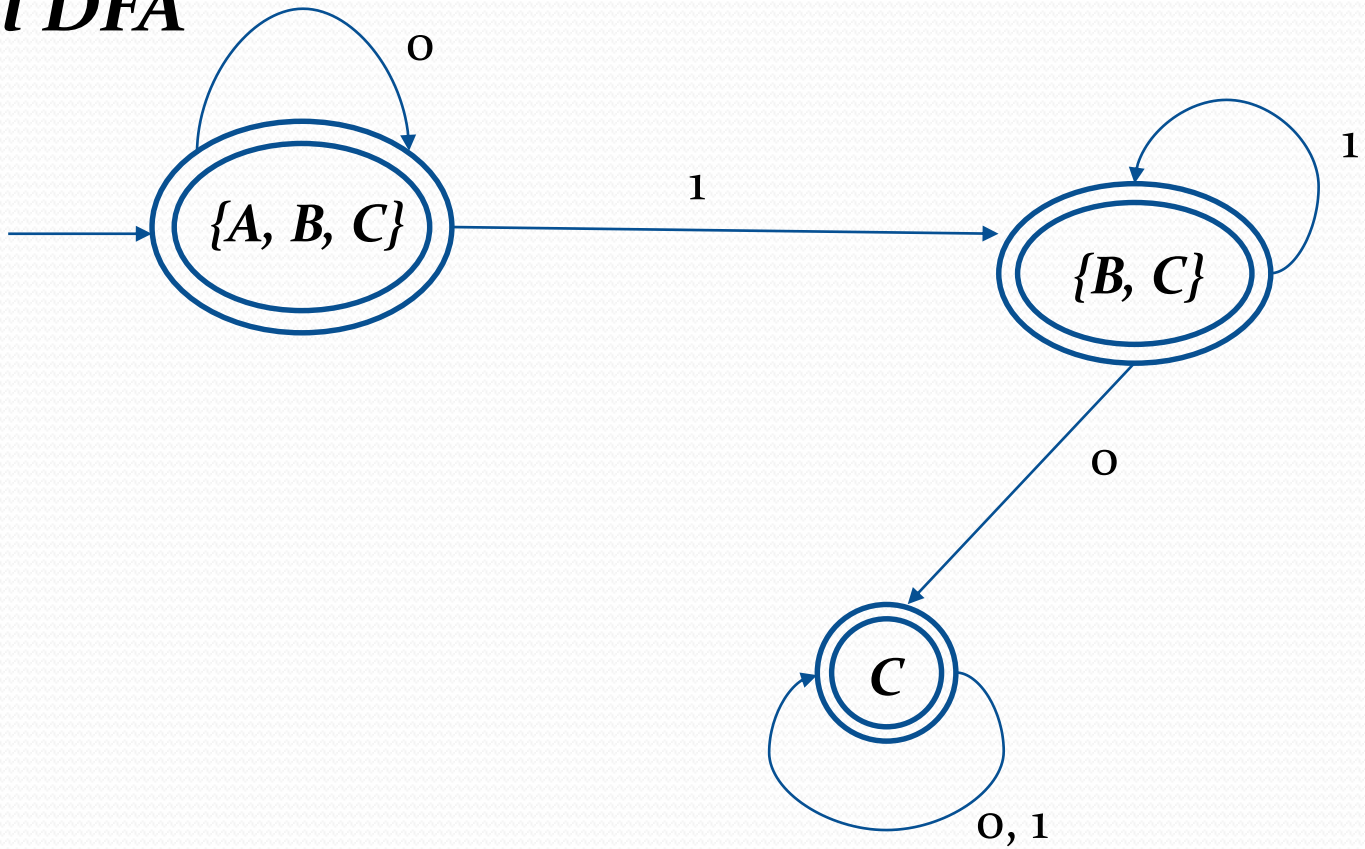
$\delta_D(C, 1) = \varepsilon\text{-closure}(\delta(C),1)$

$= \varepsilon\text{-closure}(C)$

$= \{C\}$

Converting ε –NFA to DFA

Final DFA



Converting ε –NFA to DFA

Question