UPES

UNIVERSITY WITH A PURPOSE

CSEG 3004
Formal Languages And Automata Theory

UPES
UNIVERSITY WITH A PURPOSE

Content of this lecture

- In this presentation we will discuss the Finite Automata

Basic Building Blocks - Alphabet

- An **alphabet** is a finite set of symbols. e.g.
  - $\Sigma_1 = \{a, b, c, d, \ldots, z\}$: the set of letters in English
  - $\Sigma_2 = \{0, 1, \ldots, 9\}$: the set of (base 10) digits
  - $\Sigma_3 = \{a, b, \ldots, z, \#\}$: the set of letters plus the special symbol #
  - $\Sigma_4 = \{ (,) \}$: the set of open and closed brackets

Basic Building Blocks - String

- A **string** over alphabet Σ is a finite sequence of symbols in Σ. e.g.
  - abfbz is a string over $\Sigma_1 = \{a, b, c, d, \ldots, z\}$
  - 9021 is a string over $\Sigma_2 = \{0, 1, \ldots, 9\}$
  - ab#bc is a string over $\Sigma_3 = \{a, b, \ldots, z, \#\}$
  -))()(() is a string over $\Sigma_4 = \{ (,) \}$
- **Length** of string: Total no of symbols(either same or different) in the string
- **Empty** String: String of length 0 is called empty/null string. It is denoted by ϵ /Λ / λ

Basic Building Blocks - String

- **Concatenation** of two strings
  - x=ab, y=dc then xy=abdc and yx=dcab
  - Concatenation of empty string with any string is that string it self.
  - So empty string acts as string identifier during string concatenation operation. E.g.
  - $\epsilon \,.\, x = x.\, \epsilon = x$

Basic Building Blocks - Prefix

- **Prefix** of string: Obtained by removing zero or more trailing symbols.
- **Proper Prefix** of string: Obtained by removing one or more trailing symbols.
- Let x=abc

Basic Building Blocks - Prefix

- **Prefix** of string: Obtained by removing zero or more trailing symbols.
- **Proper Prefix** of string: Obtained by removing one or more trailing symbols.
- Let x=abc
- Prefix of x are: abc, ab, a, ϵ
- Proper Prefix of x are: ab, a, ϵ

Basic Building Blocks - Suffix

- **Suffix** of string: Obtained by removing zero or more leading symbols.
- **Proper Suffix** of string: Obtained by removing one or more leading symbols.
- Let x=abc

Basic Building Blocks - Suffix

- **Suffix** of string: Obtained by removing zero or more leading symbols.
- **Proper Suffix** of string: Obtained by removing one or more leading symbols.
- Let x=abc
- Suffix of x are: abc, bc, c, ϵ
- Proper Suffix of x are: bc, c, ϵ

Basic Building Blocks - Substring

- **Substring** of string: It is obtained by removing zero or more leading or trailing symbols.
- **Proper Substring** of string: Obtained by removing one or more leading or trailing symbols.
- X=abc

Basic Building Blocks - Substring

- X=abc
- **Substring** of string: It is obtained by removing zero or more leading or trailing symbols.
- **Proper Substring** of string: Obtained by removing one or more leading or trailing symbols.
- Substring of X are: abc, bc, ab, b, a, c, ϵ
- Proper substring of X are: bc, ab, b, a, c, ϵ
- **Any prefix or suffix of any string x must be a substring of x but reverse may or may not be true.**

Basic Building Blocks - Language

- **Language:** It is a set of strings which are formed by concatenation of symbols of the alphabets by following the grammatical rules for that language.
- It may be finite or infinite.
- Σ ={x},
  - L_1= odd length strings
  - L_2= even length strings
  - L_3= any length strings

Basic Building Blocks - Language

- **Language:** It is a set of strings which are formed by concatenation of symbols of the alphabets by following the grammatical rules for that language.
- It may be finite or infinite.
- Σ ={x},
  - L_1= odd length strings = {x, xxx, xxxxx, xxxxxxx,….}
  - L_2= even length strings ={ϵ, xx, xxxx, xxxxxx,…..}
  - L_3= any length strings ={ϵ, x, xx, xxx, xxxx, xxxxx, …..}
- Language consisting of no word is denoted by null set ø={ }
- Language L = { ϵ } is not same as ø

Operation on Language

- If L & M are the two languages then
  - **Concatenation** of L & M will be the language consisting of all the string xy which can be formed by selecting a string x from L and y from M. It is denoted by LM.
  - L={a, bc}
  - M={b, ac}
  - LM=
  - ML=

Operation on Language

- If L & M are the two languages then
  - **Concatenation** of L & M will be the language consisting of all the string xy which can be formed by selecting a string x from L and y from M. It is denoted by LM.
  - L={a, bc}
  - M={b, ac}
  - LM= {ab, aac, bcb, bcac}
  - ML={ba, bbc, aca, acbc}

Operation on Language

- If L & M are the two languages then
  - **Union** of L & M will be collection of all those strings which are either in L or in M. It is denoted by L U M.
  - L={a, ac}
  - M={b, ac}
  - LM=

Operation on Language

- If L & M are the two languages then
  - **Union** of L & M will be collection of all those strings which are either in L or in M. It is denoted by L U M.
  - L={a, ac}
  - M={b, ac}
  - LM= {a, b, ac}

Operation on Language

- If L is a languages then
  - **Closure/Kleen Closure** of L will be the collection of all those strings which will be formed by combining the strings of language L with itself or with other strings with any number of times and in any manner. It is denoted by $L^* = U_{i=0}^{\infty} L^i = \{L^0, L^1, L^2, L^3 \ldots.\}$
  - **Positive Closure:** $L^+ = U_{i=1}^{\infty} L^i = \{L^1, L^2, L^3, L^4, \ldots.\}$
  - If L={a, b} then L^* =
  - If L={a, b} then L^+ =

Operation on Language

- If L is a languages then
  - **Closure/Kleen Closure** of L will be the collection of all those strings which will be formed by combining the strings of language L with itself or with other strings with any number of times and in any manner. It is denoted by $L^* = \bigcup_{i=0}^{\infty} L^i = \{L^0, L^1, L^2, L^3 \ldots.\}$
  - **Positive Closure:** $L^+ = \bigcup_{i=1}^{\infty} L^i = \{L^1, L^2, L^3, L^4, \ldots.\}$
  - If L={a, b} then L^* ={ϵ, a, b, aa, ab, ba, bb, aaa, aab, aba, baa, bba, bab, abb, bbb,…… }
  - If L={a, b} then L^+ ={a, b, aa, ab, ba, bb, aaa, aab, aba, baa, bba, bab, abb, bbb,…… }

Regular Expression

- A regular expression is a notation to represent a language.
- It must be general enough that it represents all the strings of the language.
- Also it must exclude all those strings which are not part of the language.

Regular Expression Construction Rules

- A regular expression over Σ is an expression formed using the following rules:
  - Regular expression for $\emptyset$ is $\emptyset$ itself.
  - Regular expression for L={ε} is ε
  - For every a $\in \Sigma$, that symbol a is a regular expression it self
  - If R and S are regular expressions for language L_1 & L_2 respectively, then
    - RS will be the regular expression corresponding L_1L_2
    - R+S will be the regular expression corresponding $L_1 + L_2$
    - R* will be the regular expression corresponding to L_1^*

Regular Expression Construction Rules

- While solving regular expression, closure has highest precedence. It is followed by concatenation and union in the end.
- Language associated with regular expression is called Regular Language.
- Every regular language must have regular expression.
- Only regular languages have regular expression.
- Languages which does not have regular expressions are called non regular language.

Regular Expression Numerical

- Describe the following sets by regular expressions:
- (a) {101}
- (b) {abba}
- (c) {01, 10}
- (d) {λ, ab}
- (e) {abb, a, b, bba}
- (f) {λ, 0, 00, 000.... }
- (g) {1, 11, 111 ... }

Regular Expression Numerical

- Describe the following sets by regular expressions:
- (a) {101}=101
- (b) {abba}=abba
- (c) {01, 10} =01+10
- (d) {λ, ab} = λ +ab
- (e) {abb, a, b, bba} =abb+a+b+bba
- (f) {λ, 0, 00, 000.... } =0^*
- (g) {1, 11, 111 ... }=1(1)*

Regular Expression Numerical

- Let Σ ={x}, find the regular expression for the following languages:
  - Odd length string

Regular Expression Numerical

- Let Σ ={x}, find the regular expression for the following languages:
  - Odd length string = {x, xxx, xxxxx, xxxxxxx,……}=x(xx)*

Regular Expression Numerical

- Let Σ ={x}, find the regular expression for the following languages:
  - Even length string

Regular Expression Numerical

- Let Σ ={x}, find the regular expression for the following languages:
  - Even length string= {x^0, xx, xxxx, xxxxxx,……}=$(xx)^*$

Regular Expression Numerical

- Let Σ ={x}, find the regular expression for the following languages:
  - Any length string

Regular Expression Numerical

- Let Σ ={x}, find the regular expression for the following languages:
  - Any length string= {x^0, x, xx, xxx, xxxx, xxxxx, xxxxxx, xxxxxxx,……}=x^*

Regular Expression Numerical

- Let Σ ={a, b}, find the regular expression for the following languages:
  - Odd length string

Regular Expression Numerical

- Let Σ ={a, b}, find the regular expression for the following languages:
  - Odd length string ={a, b, aaa, aba, aab, baa, bba, bab, abb, bbb,…..}
  - =(a+b)((a+b)(a+b))*

Regular Expression Numerical

- Let Σ ={a, b}, find the regular expression for the following languages:
  - Even length string

Regular Expression Numerical

- Let Σ ={a, b}, find the regular expression for the following languages:
  - Even length string ={ε , aa, ab, ba, bb,,,,}=((a+b)(a+b))*

Regular Expression Numerical

- Let Σ ={a, b}, find the regular expression for the following languages:
  - Any length string

Regular Expression Numerical

- Let Σ ={a, b}, find the regular expression for the following languages:
  - Any length string={ε ,a, b, aa, ab, ba, bb, aaa, aba,…..}=$(a+b)^*$

Regular Expression Numerical

- Let Σ ={a, b}, find the regular expression for the following languages:
  - Start with a

Regular Expression Numerical

- Let Σ ={a, b}, find the regular expression for the following languages:
  - Start with a ={a, ab, aa, aaa, abb, aba, aab,..}=$a(a+b)^*$

Regular Expression Numerical

- Let Σ ={a, b}, find the regular expression for the following languages:
  - End with b

Regular Expression Numerical

- Let Σ ={a, b}, find the regular expression for the following languages:
  - End with b={b, ab, bb, aab, abb, bab, bbb,… }=$(a+b)^*b$

Regular Expression Numerical

- Let Σ ={a, b}, find the regular expression for the following languages:
  - End with ab

Regular Expression Numerical

- Let Σ ={a, b}, find the regular expression for the following languages:
  - End with ab={ab, bab, aab, bbab, baab, abab, aaab,….}=$(a+b)^*ab$

Regular Expression Numerical

- Let Σ ={a, b}, find the regular expression for the following languages:
  - Start with a and end with b

Regular Expression Numerical

- Let Σ ={a, b}, find the regular expression for the following languages:
  - Start with a and end with b = {ab, aab, abb, aaab, aabb, abab, abbb, …} = $a(a+b)^*b$

Regular Expression Numerical

- Let Σ ={a, b}, find the regular expression for the following languages:
  - String of length three

Regular Expression Numerical

- Let Σ ={a, b}, find the regular expression for the following languages:
  - String of length three ={aaa, aab, aba, baa, bba, bab, abb, bbb}= (a+b) (a+b) (a+b)

Regular Expression Numerical

- Let Σ ={a, b}, find the regular expression for the following languages:
  - Must have at least one a

Regular Expression Numerical

- Let Σ ={a, b}, find the regular expression for the following languages:
  - Must have at least one a={a, ba, ab, aa, aaa, aba, baa, aab, bba, bab, abb, …}= $(a+b)^*a(a+b)^*$

Regular Expression Numerical

- Let Σ ={a, b}, find the regular expression for the following languages:
  - Must have at least two a

Regular Expression Numerical

- Let Σ ={a, b}, find the regular expression for the following languages:
  - Must have at least two a={aa, aaa, aba, baa, aab,…}= $(a+b)^*a(a+b)^*a(a+b)^*$

Regular Expression Numerical

- Let Σ ={a, b}, find the regular expression for the following languages:
  - Have exactly two a

Regular Expression Numerical

- Let Σ ={a, b}, find the regular expression for the following languages:
  - Have exactly two a = {aa, baa, aba, aab, … }=$b^*ab^*ab^*$

Regular Expression Numerical

- Find the regular expression for the identifiers of the C language:

Regular Expression Numerical

- Find the regular expression for the identifiers of the C language:
- Rules for identifiers of C language are:
  - First character must be a letter or underscore
  - Max. length of identifier can be 32 and minimum can be one
  - Identifier is composed of letter (small or caps), underscore and digits
  - Let I represents the set of all small or capital alphabets
  - Let U represents the set containing underscore
  - Let D represents the set of all digits
  - Let E represents the set containing absylan
  - $(I+U)(I+U+D+E)^{31}$

Regular Expression Numerical

- Let Σ ={0, 1}, find the regular expression for the following languages:
  - Every zero is immediately followed by at least two one's

Regular Expression Numerical

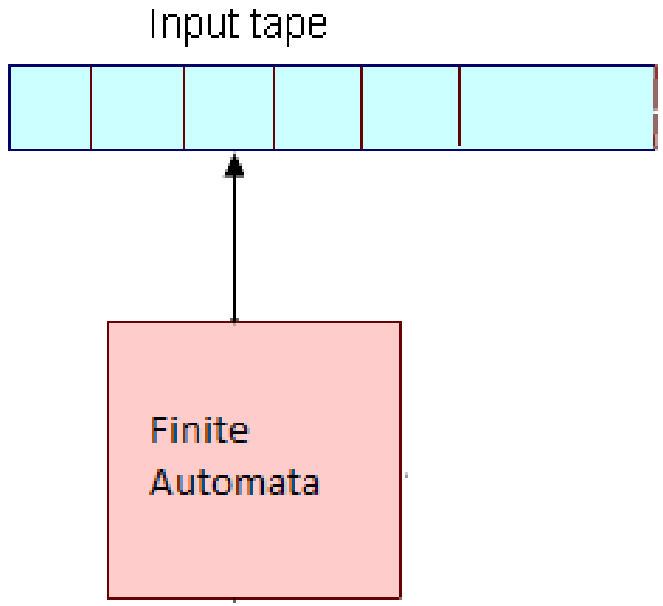
- Let Σ ={0, 1}, find the regular expression for the following languages:
  - Every zero is immediately followed by at least two one's
  - Possible strings in the language are= {λ, 1,11,011, 111,…} = $(1+011)^*$

Finite Automata

- Any system which is at each moment in one of the finite number of discrete state and move among these states in a predictable way in response to individual input signal can be modeled by finite automata. FA is collection of 3 things:
  - A finite set of states such that one of the state is called start/initial sate and some/none will be designated as final/accepting/halting state.
  - An alphabet of input symbols from which string of symbols are formed.
  - A finite set of transition rules which tell for each state and for each input letter which next state to go.
- FA acts as language recognizer.

Finite Automata - As a Machine

- It represents a machine which read the single symbol at a time from input tape by strictly moving in one direction (either left to right or right to left).
- It only shows the success/failure, true/false or acceptance/rejection by ending at halting/ non halting state.



Finite Automata - Applications

- For the designing of lexical analysis of a compiler.
- For recognizing the pattern using regular expressions.
- For the designing of the combination and sequential circuits using Mealy and Moore Machines.
- Used in text editors.
- For the implementation of spell checkers.

Finite Automata - Limitations

- It does not have any memory.
- It does not gives any output.
- It only shows the success/failure, true/false or acceptance/rejection by ending at halting/ non halting state.
- Head movement is in only one direction.
- Input tape is read only.

Finite Automata-Types

- It is of three types:
  - Deterministic Finite Automata (DFA)
  - Non Deterministic Finite Automata (NDFA)
  - Non Deterministic Finite Automata with epsilon (NDFA-ϵ)
- Power (in terms of recognition) of all the three variants of FA are same. Corresponding to every NDFA-ϵ we can have equivalent NDFA. Corresponding to every NDFA we can have equivalent DFA.
- If there exist FA corresponding to any language then that language must be regular and must also have regular expression.
- Regular expression ↔ Regular Language ↔ FA

Deterministic Finite Automata (DFA)-Theoretical Definition

- It is that type of finite automata, in which after reading a symbol from a current state, system can move at most towards one state (same/different) i.e. moving of machine from a state corresponding to a symbol is deterministic (only one option) not deterministic. From any state there must be at most one out going edge corresponding to symbol read.
- Also by reading epsilon(λ) system can’t change the state i.e. no edge with labeled epsilon(λ) will be there in DFA.

Deterministic Finite Automata (DFA)-Mathematical Definition

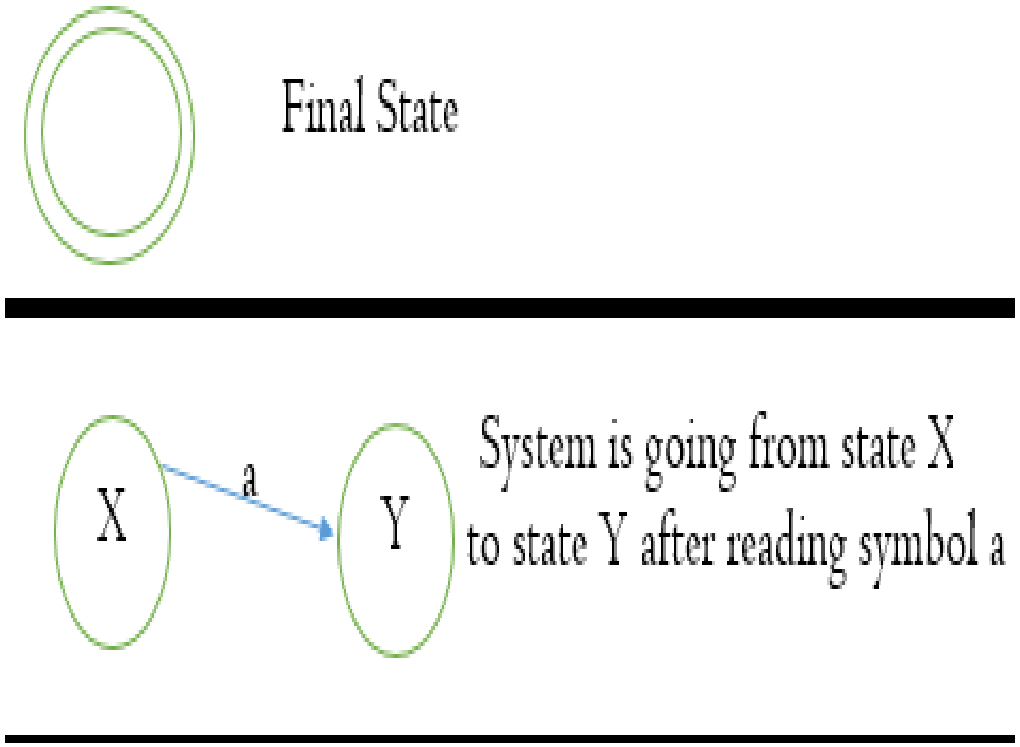
- A DFA M is defined as a collection of 5 tuples $(Q, \Sigma, \delta, q_0, F)$ where
  - Q is a finite set of states
  - Σ is an alphabet
  - $\delta : Q \times \Sigma \rightarrow Q$ is a transition function

    e.g. $\delta(X, a)=Y$ implies that by reading symbol 'a' from state X machine is moving on state Y, where $X, Y \in Q$
  - $q_0 \in Q$ is the initial state (only one)
  - $F \subseteq Q$ is a set of accepting/starting/halting/terminating states (can be zero, one or more than one).
  - $\delta(X, \epsilon)=X$, i.e. in DFA, system can't change the state by reading ϵ

- When we represent the FA diagrammatically it is called transition diagram while its tabular representation is called transition table.

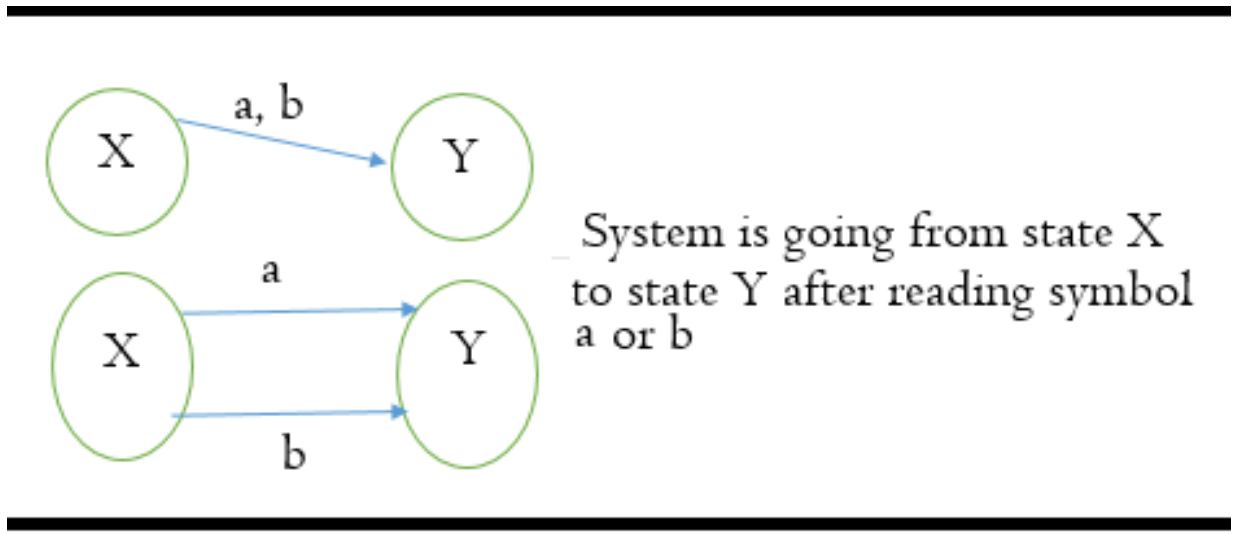
Deterministic Finite Automata (DFA)-Notation

- In transition diagram, state will be represented by circle while accepting states will be represented by circle inside circle.



Deterministic Finite Automata (DFA)-Notation

- In transition diagram, state will be represented by circle while accepting states will be represented by circle inside circle.



Deterministic Finite Automata (DFA)

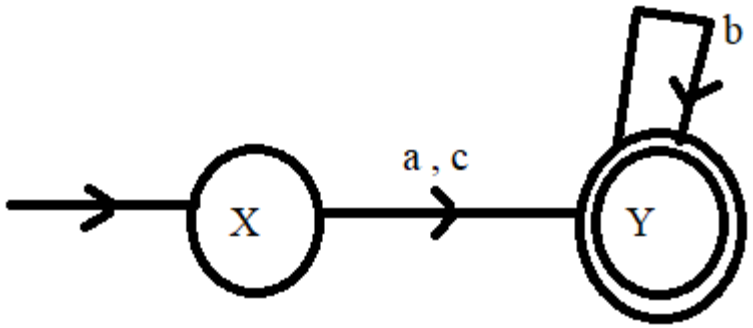
- In DFA, when we say language L is accepted by machine M then it means that all the strings in L would leave machine in final state and any string x which does not belong to L will never leave the machine into final state.
- In mathematical form this can be written as follows:
  - L(M)= Language accepted my machine M
  - L(M)= { x | x is accepted by DFA M, i.e. $\delta^*(q_0, x) \epsilon A\}$

Deterministic Finite Automata (DFA)-Numericals

- Σ={a,b,c}, regular expression= $(a+c)b^*$

Deterministic Finite Automata (DFA)-Numericals

- Σ={a,b,c}, regular expression= (a+c)b^*



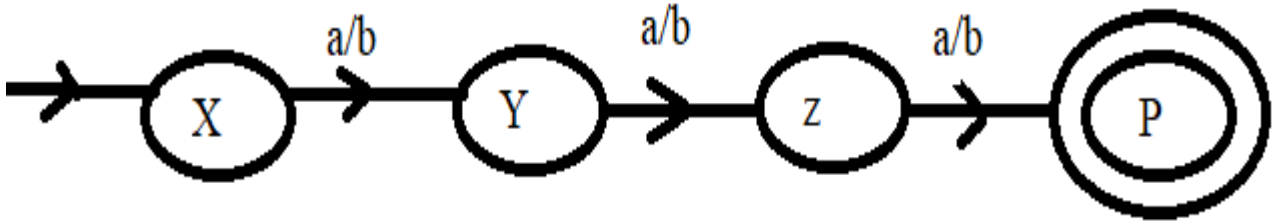
	a	b	c
X	Y	--	Y
Y	--	Y	--

Deterministic Finite Automata (DFA)-Numericals

- Σ={a, b}
  - only three length string is permitted

Deterministic Finite Automata (DFA)-Numericals

- Σ={a, b}
  - only three length string is permitted



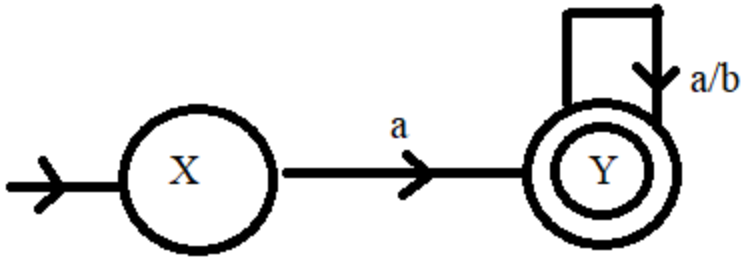
	a	b
X	Y	Y
Y	Z	Z
Z	P	P
P	--	--

Deterministic Finite Automata (DFA)-Numericals

- Σ={a, b}
  - Starting symbol must be a

Deterministic Finite Automata (DFA)-Numericals

- Σ={a, b}
  - Starting symbol must be a



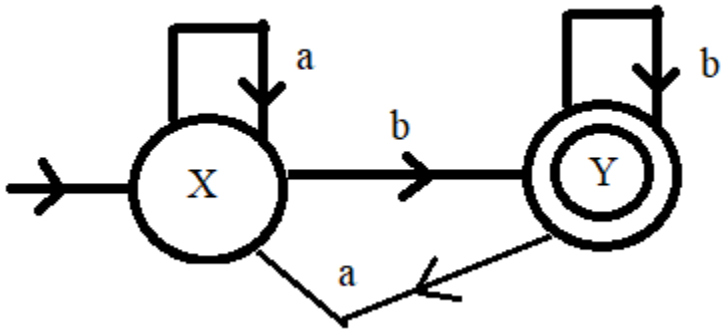
	a	b
→ X	Y	---
Y	Y	Y

Deterministic Finite Automata (DFA)-Numericals

- Σ={a, b}
  - Ending symbol must be b

Deterministic Finite Automata (DFA)-Numericals

- Σ={a, b}
  - Ending symbol must be b= (a+b)*b

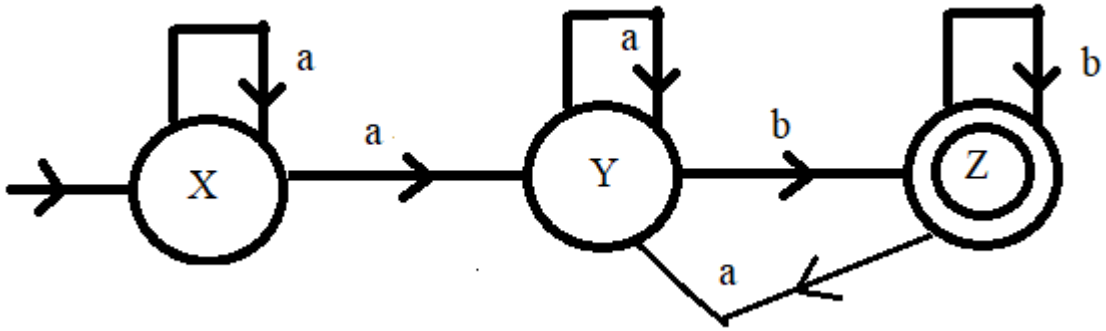


Deterministic Finite Automata (DFA)-Numericals

- Σ={a, b}
  - Starting symbol must be a and ending symbol must be b

Deterministic Finite Automata (DFA)-Numericals

- Σ={a, b}
  - Starting symbol must be a and ending symbol must be b= a(a+b)*b

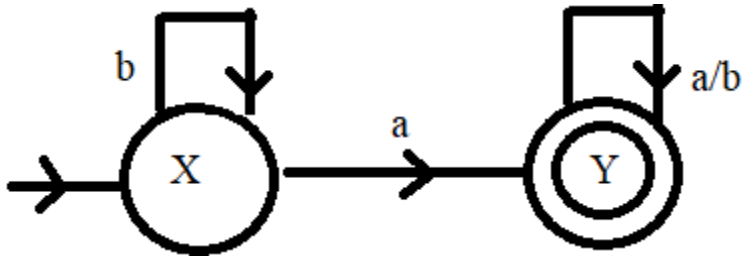


Deterministic Finite Automata (DFA)-Numericals

- Σ={a, b}
  - At least one a must be there

Deterministic Finite Automata (DFA)-Numericals

- Σ={a, b}
  - At least one a must be there=(a+b) *a(a+b)*

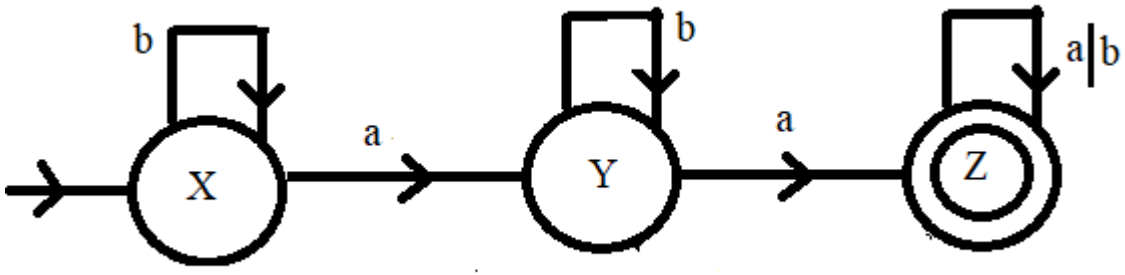


Deterministic Finite Automata (DFA)-Numericals

- Σ={a, b}
  - At least two a must be there

Deterministic Finite Automata (DFA)-Numericals

- Σ={a, b}
  - At least two a must be there=(a+b)*a(a+b)*a(a+b)*

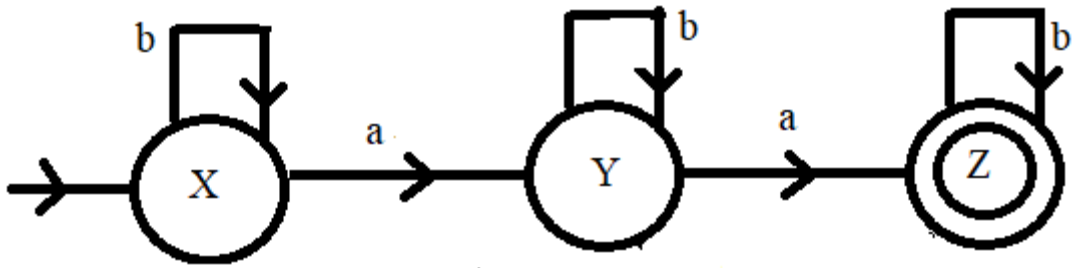


Deterministic Finite Automata (DFA)-Numericals

- Σ={a, b}
  - Exactly two a will be there

Deterministic Finite Automata (DFA)-Numericals

- Σ={a, b}
  - Exactly two a will be there

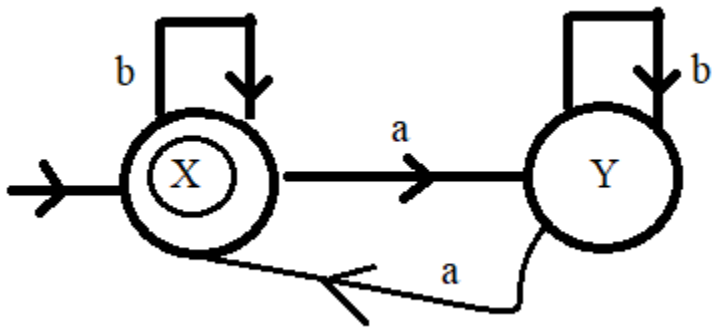


Deterministic Finite Automata (DFA)-Numericals

- Σ={a, b}
  - Number of a must be even = (b*ab*a)*b*

Deterministic Finite Automata (DFA)-Numericals

- Σ={a, b}
  - Number of a must be even = (b*ab*a)*b*

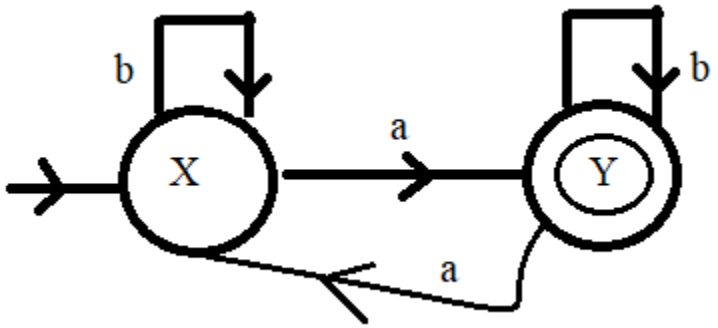


Deterministic Finite Automata (DFA)-Numericals

- Σ={a, b}
  - Number of a must be odd = (b + ab*a)*ab* =b * a(b * ab * a) * b *

Deterministic Finite Automata (DFA)-Numericals

- Σ={a, b}
  - Number of a must be odd = (b + ab*a)*ab* =b * a(b * ab * a) * b *

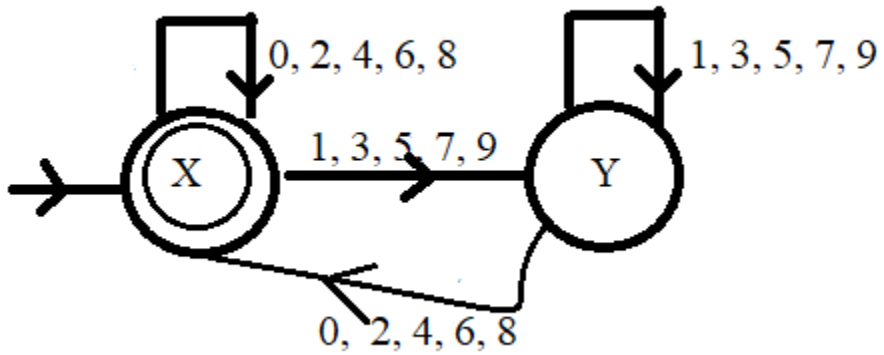


Deterministic Finite Automata (DFA)-Numericals

- Σ={0,1,2,3,4,5,6,7,8,9}
  - Draw DFA for divisible by 2

Deterministic Finite Automata (DFA)-Numericals

- Σ={0,1,2,3,4,5,6,7,8,9}
  - Draw DFA for divisible by 2

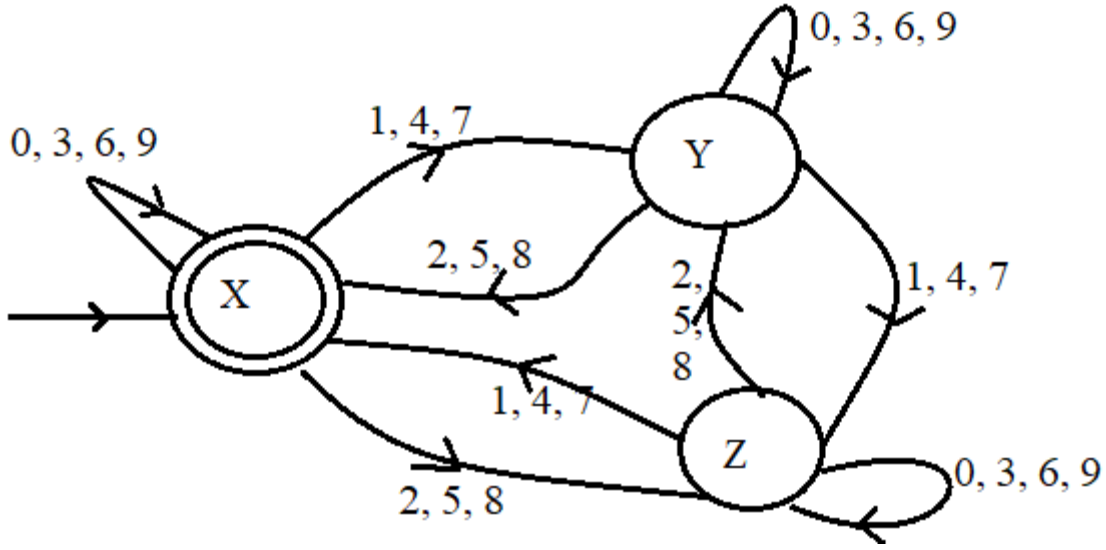


Deterministic Finite Automata (DFA)-Numericals

- Σ={0,1,2,3,4,5,6,7,8,9}
  - Draw DFA for divisible by 3

Deterministic Finite Automata (DFA)-Numericals

- Σ={0,1,2,3,4,5,6,7,8,9}
  - Draw DFA for divisible by 3

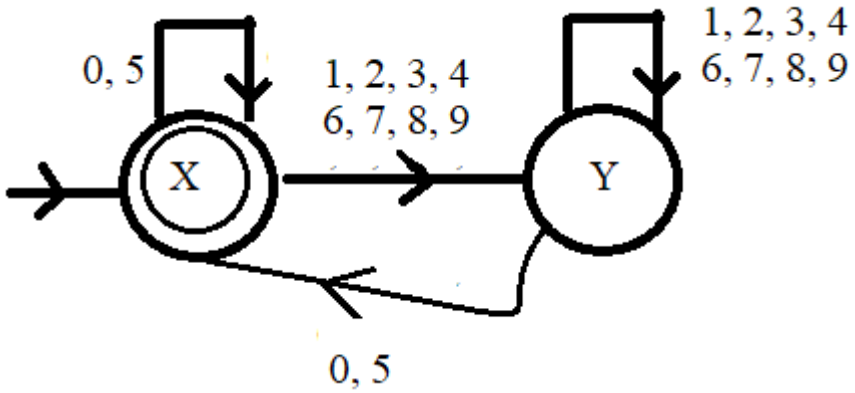


Deterministic Finite Automata (DFA)-Numericals

- Σ={0,1,2,3,4,5,6,7,8,9}
  - Draw DFA for divisible by 5

Deterministic Finite Automata (DFA)-Numericals

- Σ={0,1,2,3,4,5,6,7,8,9}
  - Draw DFA for divisible by 5



Non Deterministic Finite Automata (NDFA)

- At a particular state, there can be more than one edge with the same label i.e. by reading the same symbol system can go on more than one states.
- System can't change the state by reading **epsilon.**
- Mathematically NDFA M is defined as a collection of 5 tuples $(Q, \Sigma, \delta, q_0, F)$ where
  - Q is a finite set of states
  - Σ is an alphabet
  - **$\delta : Q \times \Sigma \rightarrow 2^Q$ is a transition function**
  - $q_0 \in Q$ is the initial state (only one)
  - $F \subseteq Q$ is a set of accepting/starting/halting/terminating states (can be zero, one or more than one).
  - $\delta(X, \epsilon)=X$, i.e. in NDFA, system can't change the state by reading ϵ
  - $\delta(X, a)$= set of all possible states on which machine can reach by reading a from X

Non Deterministic Finite Automata (NDFA)

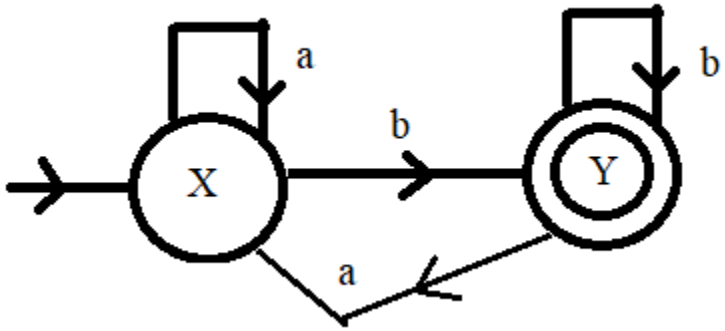
- A string is accepted by NDFA, if there is a sequence of moves such that we begin at the start state and end at final state. We ignore other redundant state.
- In NDFA, when we say language L is accepted by machine M then it means that all the strings in L would leave machine in final state and any string x which does not belong to L will never leave the machine into final state. In mathematical form this can be written as follows:
  - L(M)= Language accepted my machine M
  - L(M)= { x | x is accepted by NDFA M, i.e. $\delta^*(q_0, x) \cap A \neq \Phi\}$

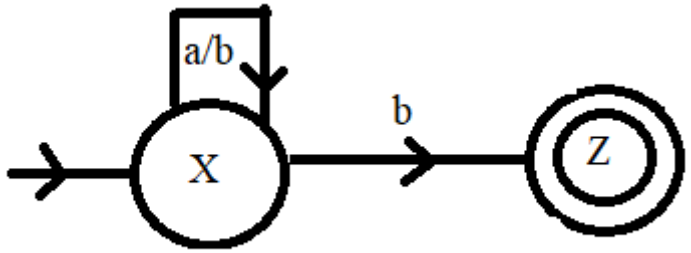
Non Deterministic Finite Automata (DFA)-Numericals

- Σ={a, b}
  - Ending symbol must be b

Non Deterministic Finite Automata (DFA)-Numericals

- Σ={a, b}
  - Ending symbol must be b= (a+b)*b



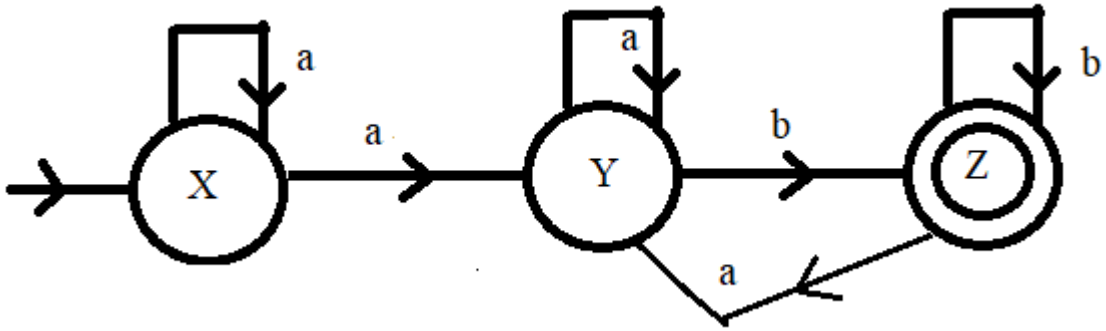


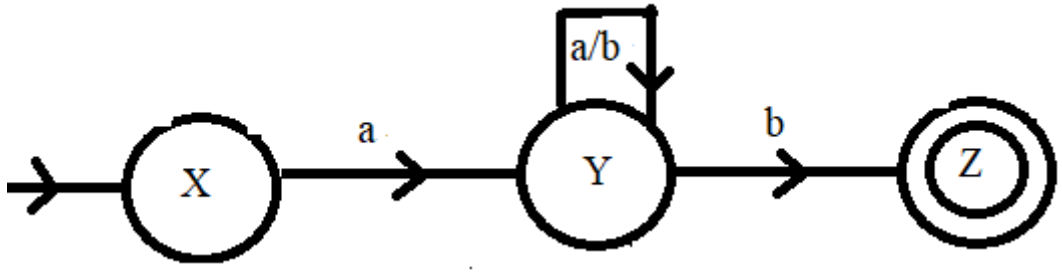
Non Deterministic Finite Automata (DFA)-Numericals

- Σ={a, b}
  - Starting symbol must be a and ending symbol must be b

Non Deterministic Finite Automata (DFA)-Numericals

- Σ={a, b}
  - Starting symbol must be a and ending symbol must be b= a(a+b)*b



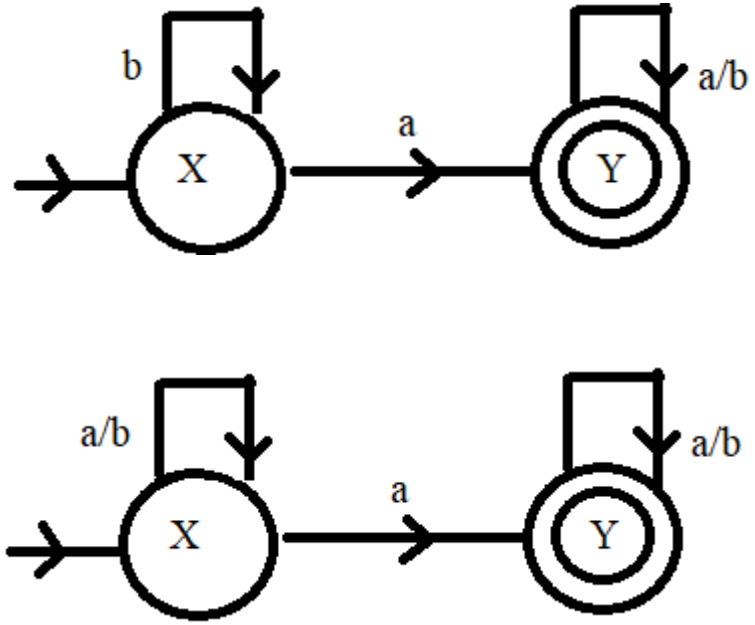


Non Deterministic Finite Automata (DFA)-Numericals

- Σ={a, b}
  - At least one a must be there

Non Deterministic Finite Automata (DFA)-Numericals

- Σ={a, b}
  - At least one a must be there=(a+b) *a(a+b)*

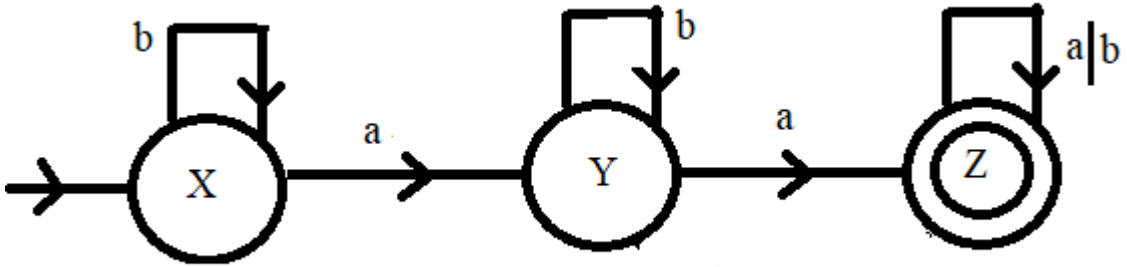


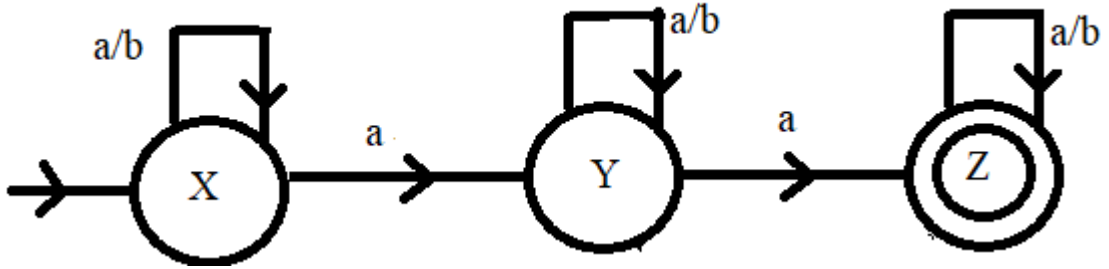
Non Deterministic Finite Automata (DFA)-Numericals

- Σ={a, b}
  - At least two a must be there

Non Deterministic Finite Automata (DFA)-Numericals

- Σ={a, b}
  - At least two a must be there=(a+b)*a(a+b)*a(a+b)*





Non Deterministic Finite Automata (NDFA) with epsilon

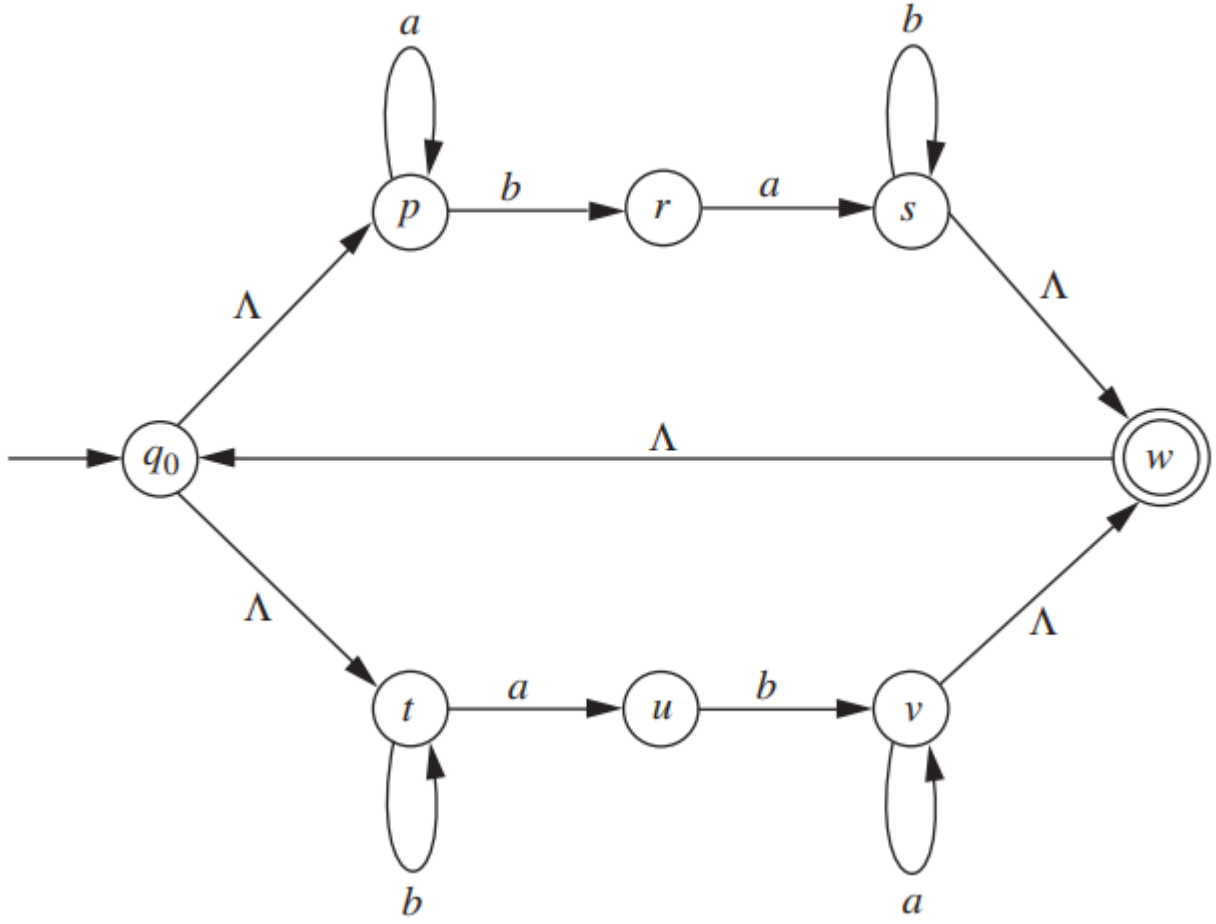
- At a particular state, there can be more than one edge with the same label i.e. by reading the same symbol system can go on more than one states.
- System can change the state by reading **epsilon**.
- Mathematically NDFA M is defined as a collection of 5 tuples $(Q, \Sigma, \delta, q_0, F)$ where
  - Q is a finite set of states
  - Σ is an alphabet
  - **$\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$ is a transition function**
  - $q_0 \in Q$ is the initial state (only one)
  - $F \subseteq Q$ is a set of accepting/starting/halting/terminating states (can be zero, one or more than one).
  -

Non Deterministic Finite Automata (NDFA) with epsilon

- A string is accepted by NDFA with epsilon, if there is a sequence of moves such that we begin at the start state and end at final state. We ignore other redundant state.
- In NDFA with epsilon, when we say language L is accepted by machine M then it means that all the strings in L would leave machine in final state and any string x which does not belong to L will never leave the machine into final state.
- In mathematical form this can be written as follows:
  - L(M)= Language accepted my machine M
  - L(M)= { x | x is accepted by NDFA M, i.e. $\delta^*(q_0, x) \cap A \neq \Phi\}$

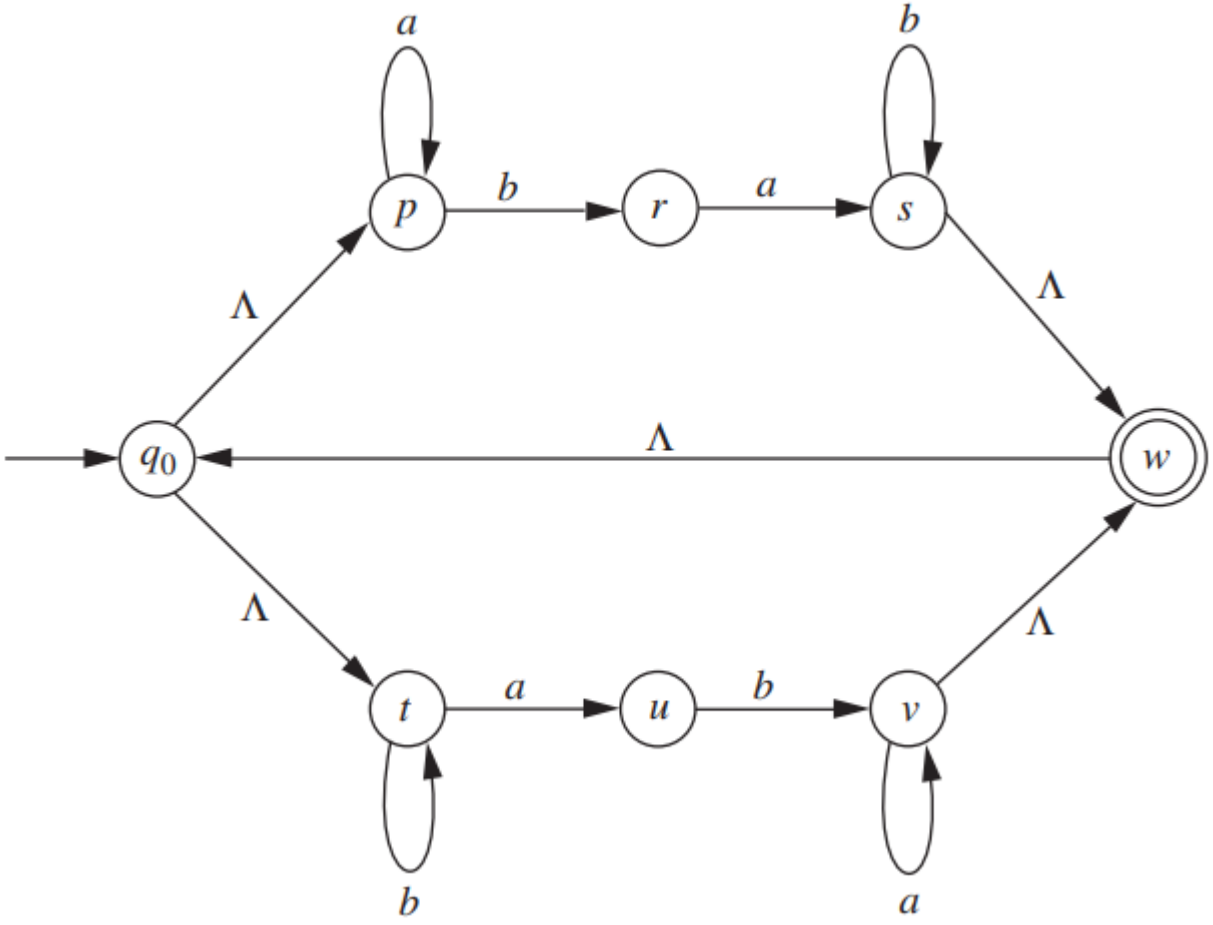
Non Deterministic Finite Automata (NDFA) with epsilon

- Find epsilon closure of all states



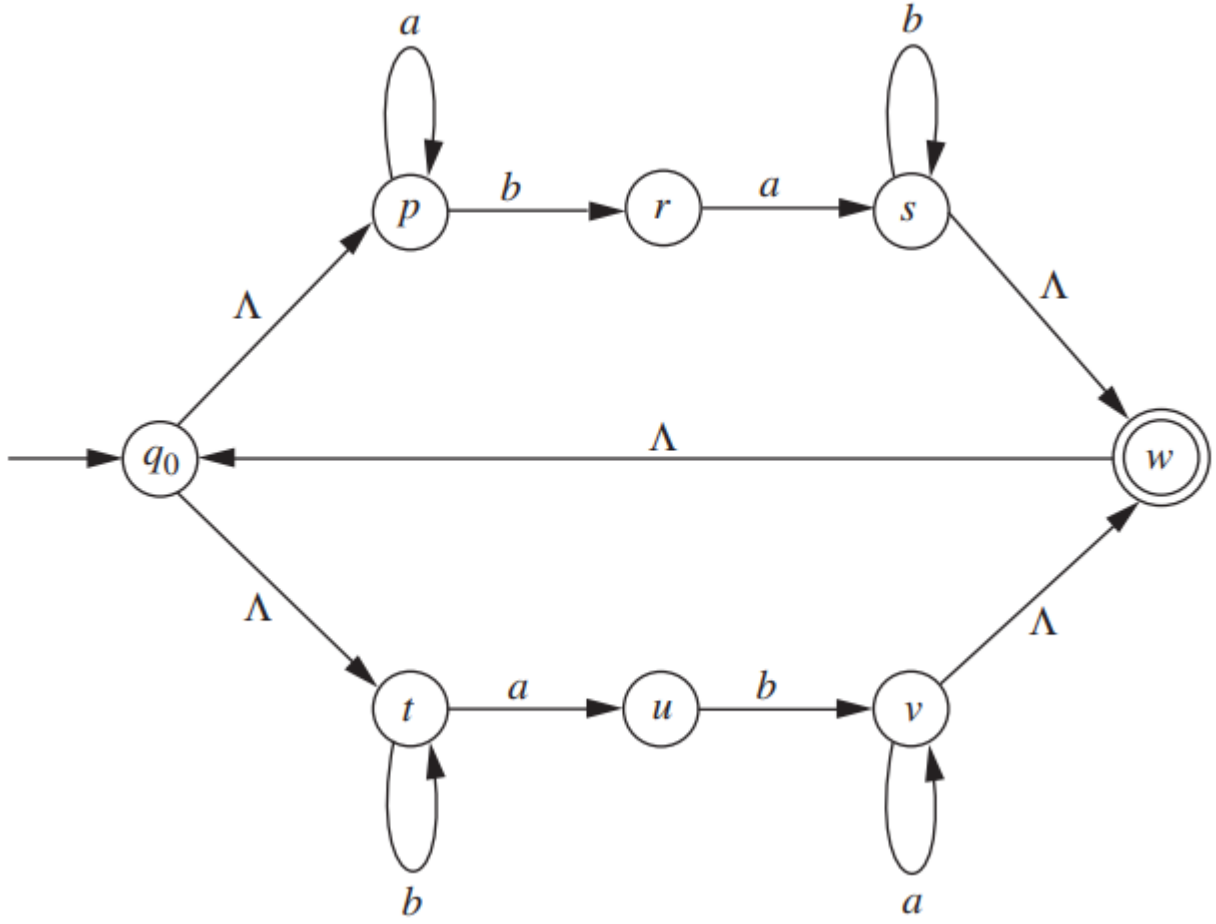
Non Deterministic Finite Automata (NDFA) with epsilon

- Find epsilon closure of all states
- $\Lambda(q_0)= \{q_0,p,t\}$
- $\Lambda(p)= \{p\}$
- $\Lambda(t)= \{t\}$
- $\Lambda(r)= \{r\}$
- $\Lambda(u)= \{u\}$
- $\Lambda(s)= \{s,w, q_0,p,t\}$
- $\Lambda(v)= \{v,w, q_0,p,t\}$
- $\Lambda(w)= \{w, q_0,p,t\}$



Non Deterministic Finite Automata (NDFA) with epsilon

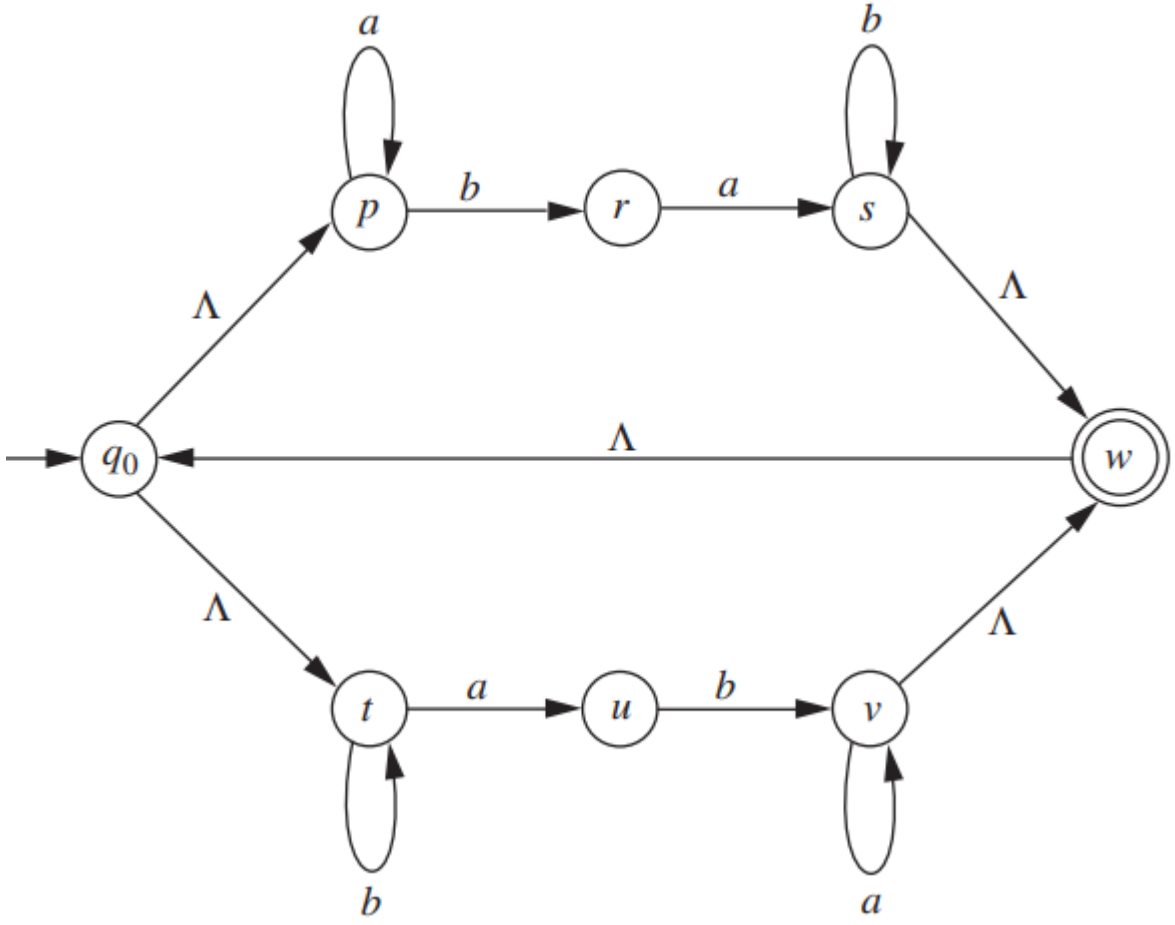
- Find $\delta^*(q_0, a)$



Non Deterministic Finite Automata (NDFA) with epsilon

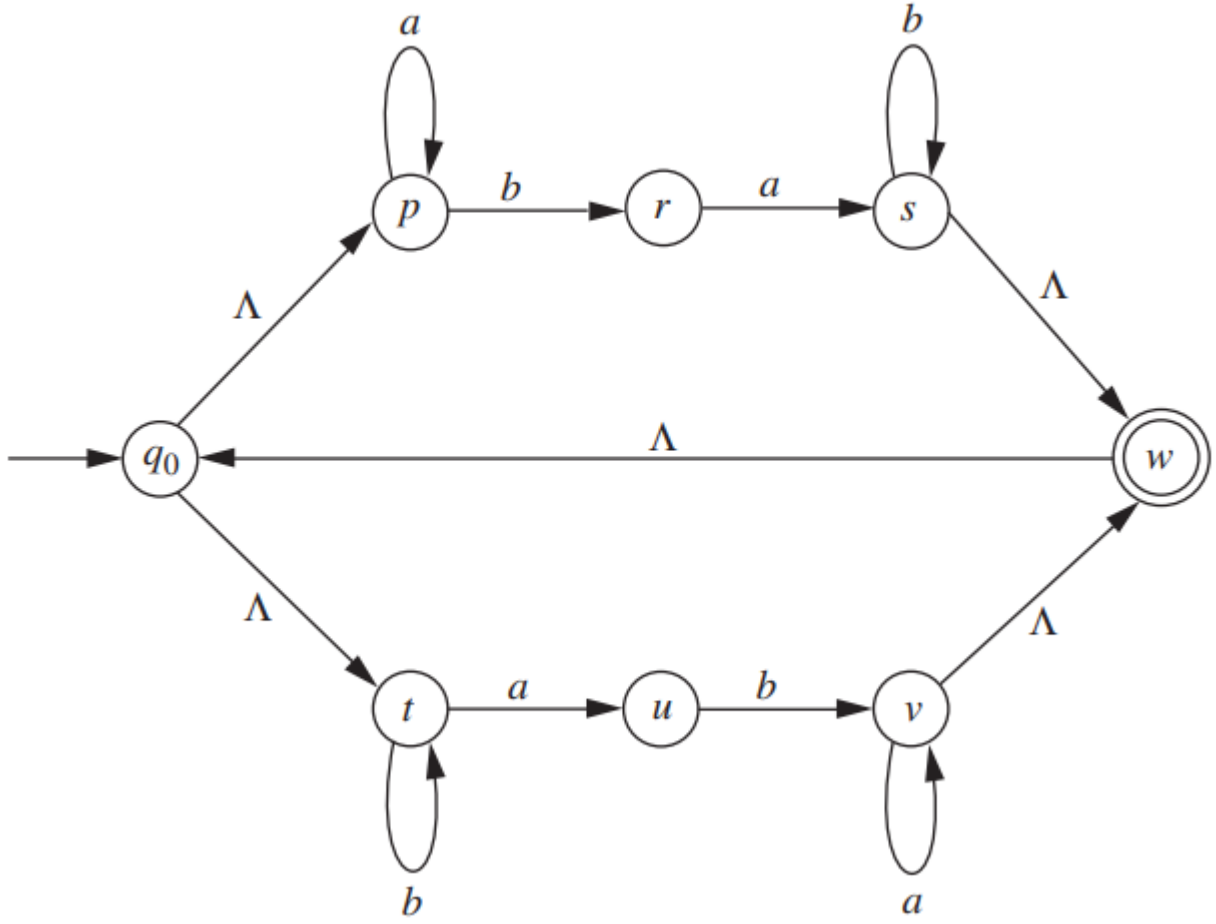
- Find $\delta^*(q_0, a)$

$$\begin{aligned}\delta^*(q_0, a) &= \Lambda\left(\bigcup\{\delta(k, a) \mid k \in \delta^*(q_0, \Lambda)\}\right)\\ &= \Lambda\left(\delta(q_0, a) \cup \delta(p, a) \cup \delta(t, a)\right)\\ &= \Lambda\left(\emptyset \cup \{p\} \cup \{u\}\right)\\ &= \Lambda(\{p, u\})\\ &= \{p, u\}\end{aligned}$$



Non Deterministic Finite Automata (NDFA) with epsilon

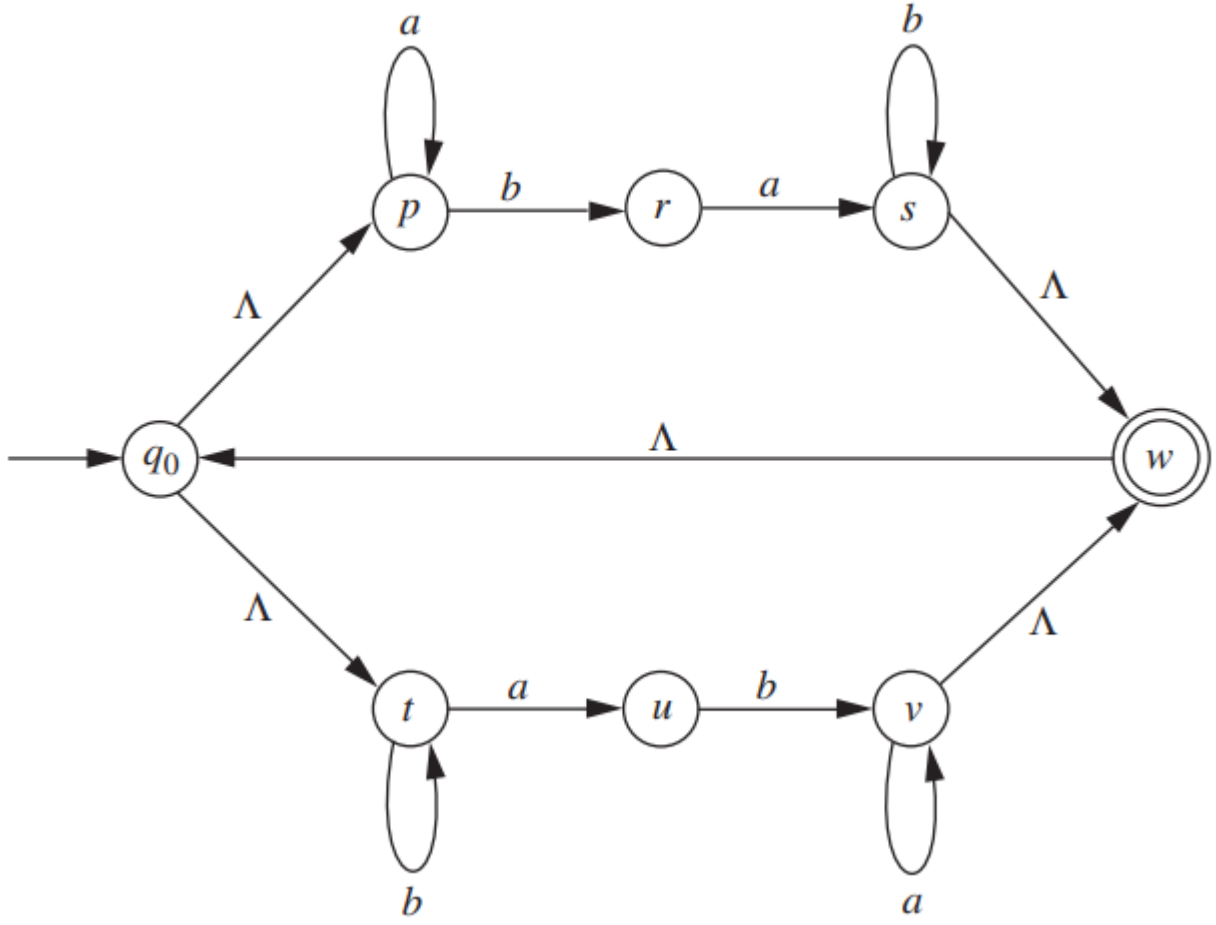
- Find $\delta^*(q_0, ab)$



Non Deterministic Finite Automata (NDFA) with epsilon

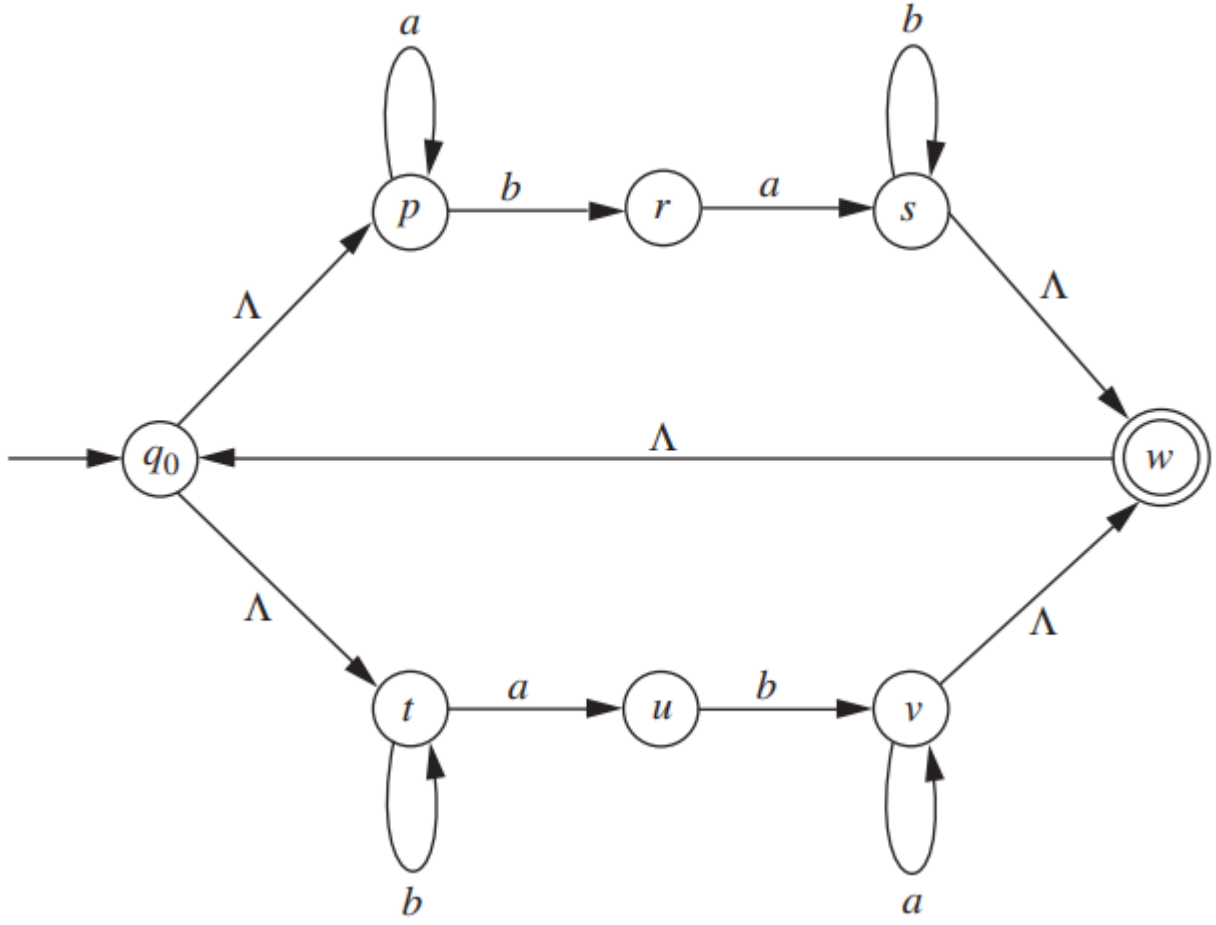
- Find $\delta^*(q_0, ab)$

$$\begin{aligned}\delta^*(q_0, ab) &= \Lambda\left(\bigcup\{\delta(k, b) \mid k \in \{p, u\}\}\right)\\ &= \Lambda(\delta(p, b) \cup \delta(u, b))\\ &= \Lambda(\{r, v\})\\ &= \{r, v, w, q_0, p, t\}\end{aligned}$$



Non Deterministic Finite Automata (NDFA) with epsilon

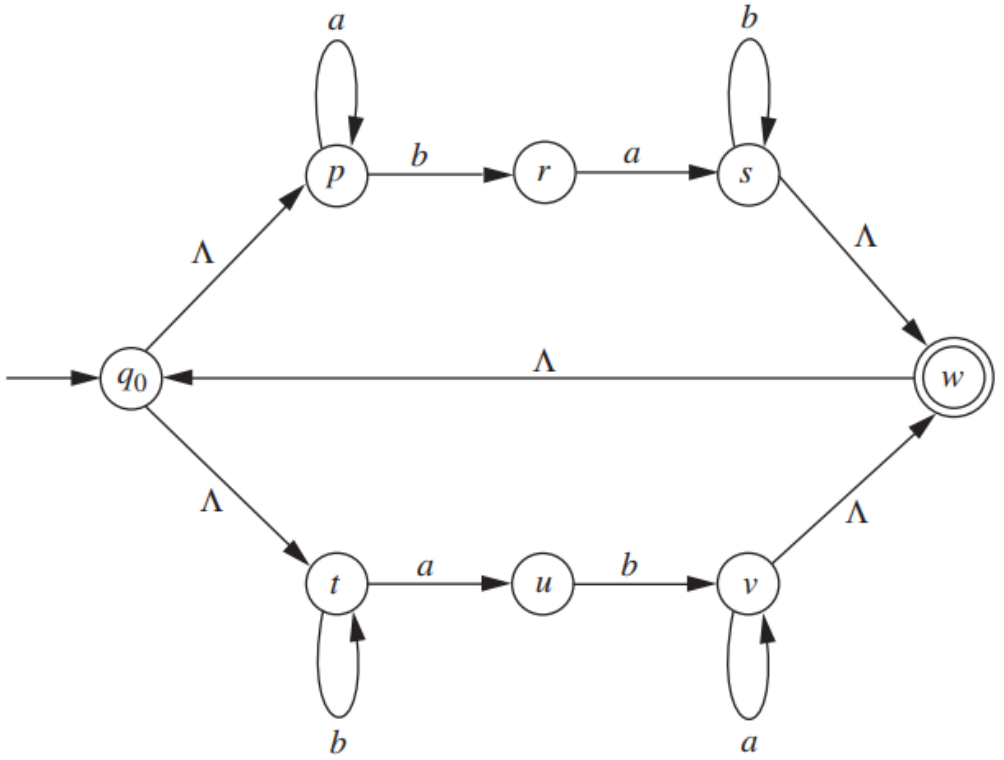
- Find $\delta^*(q_0, aba)$



Non Deterministic Finite Automata (NDFA) with epsilon

- Find $\delta^*(q_0, aba)$

$$\delta^*(q_0, aba) = \Lambda\left(\bigcup\{\delta(k, a) \mid k \in \{r, v, w, q_0, p, t\}\}\right)$$
$$= \Lambda(\delta(r, a) \cup \delta(v, a) \cup \delta(w, a) \cup \delta(q_0, a) \cup \delta(p, a) \cup \delta(t, a))$$
$$= \Lambda(\{s\} \cup \{v\} \cup \emptyset \cup \emptyset \cup \{p\} \cup \{u\})$$
$$= \Lambda(\{s, v, p, u\})$$
$$= \{s, v, p, u, w, q_0, t\}$$



NDFA to DFA

- Suppose there is an NDFA N < Q, ∑, q0, δ, F > which recognizes a language L. Then the equivalent DFA D < Q', ∑, q0, δ', F' > recognizing the same language L can be constructed as follows:
  - Step 1: Initially Q' = ϕ.
  - Step 2: Add q0 to Q'.
  - Step 3: Find the possible set of states that can be traversed from the present state for each input symbol using transition function of NDFA. If the new set of state is not part of Q' then add it to Q'.
  - Step 4: For each new state added in Q', repeat step 3.
  - Step 5: Final state of DFA will be all states with contain F (final states of NFA)

NDFA to DFA

- Consider the following NDFA

State	a	b
q0	q0,q1	q0
q1		q2
q2		

NDFA to DFA

- Consider the following NDFA

State	a	b
q0	q0,q1	q0
q1		q2
q2		

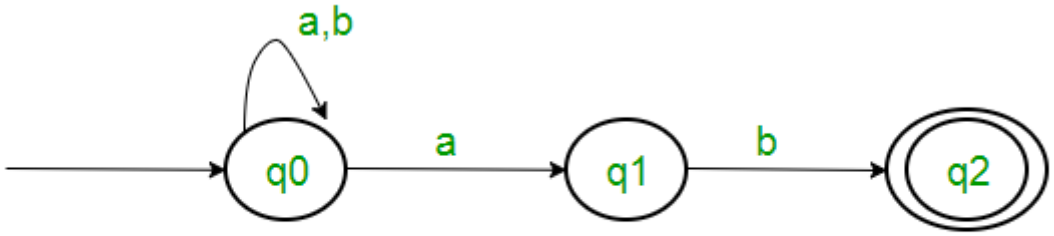


Figure 1

NDFA to DFA

- Consider the following NDFA

State	a	b
q0	q0,q1	q0
q1		q2
q2		

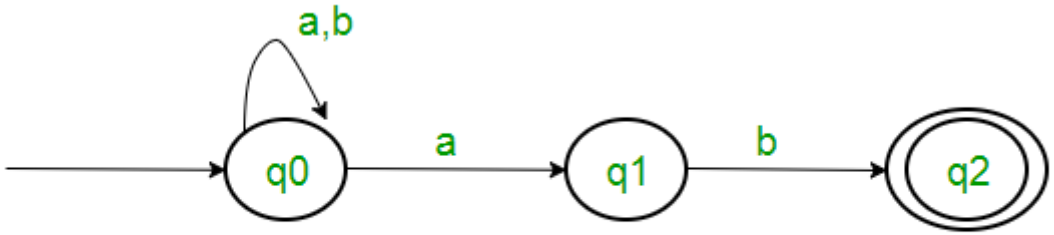


Figure 1

State	a	b
q0	{q0,q1}	q0
{q0,q1}	{q0,q1}	{q0,q2}
{q0,q2}	{q0,q1}	q0

NDFA to DFA

- Consider the following NDFA

State	a	b
q0	q0,q1	q0
q1		q2
q2		

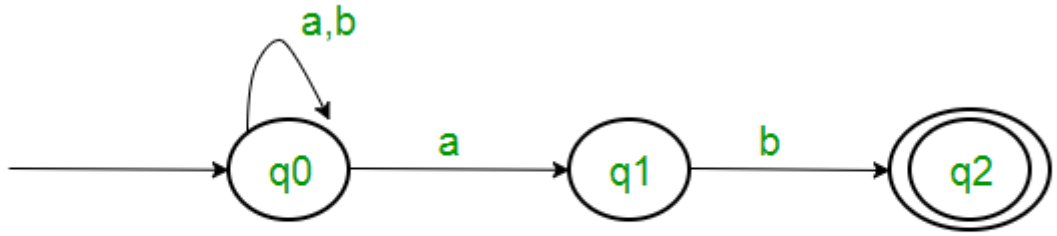


Figure 1

State	a	b
q0	{q0,q1}	q0
{q0,q1}	{q0,q1}	{q0,q2}
{q0,q2}	{q0,q1}	q0

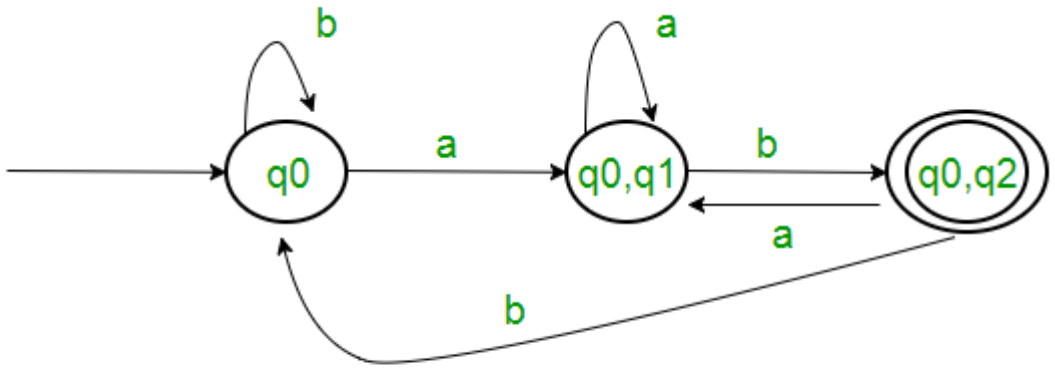


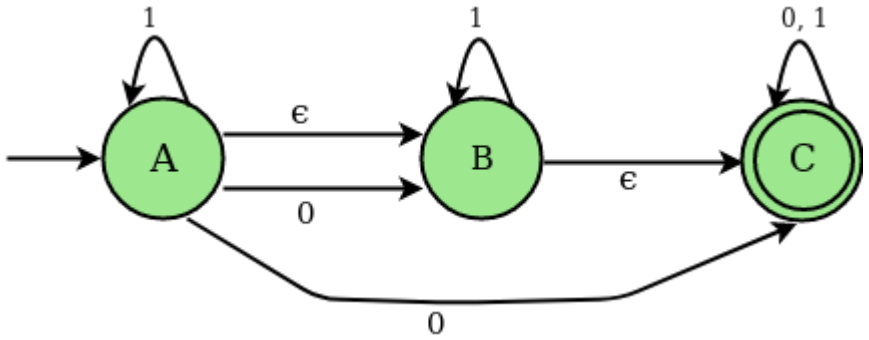
Figure 2

NDFA with epsilon to DFA

- Suppose there is an NDFA N < Q, ∑, q0, δ, F > which recognizes a language L. Then the equivalent DFA D < Q', ∑, q0, δ', F' > recognizing the same language L can be constructed as follows:
  - Step 1: Initially Q' = ϕ.
  - Step 2: Add ∈ closure of q0 to Q' as starting state of equivalent DFA.
  - Step 3: Find the possible set of states that can be traversed from the present state for each input symbol using transition function of NDFA and take ∈ closure of output to form the final set of states. If the new set of state is not part of Q' then add it to Q'.
  - Step 4: For each new state added in Q', repeat step 3.
  - Step 5: Final state of DFA will be all states with contain F (final states of NFA)

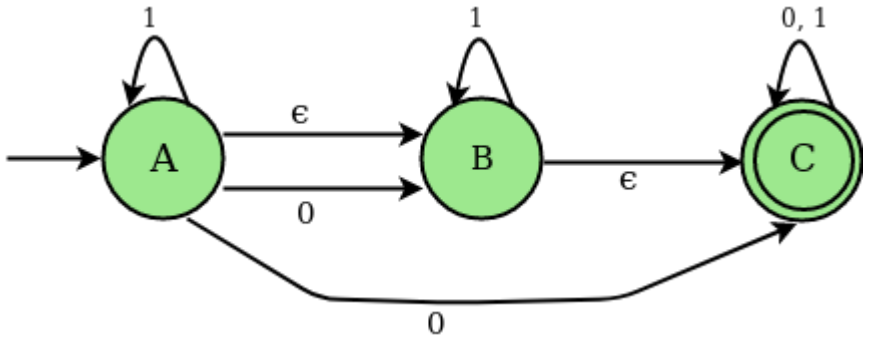
NDFA with epsilon to DFA

- Consider the following NDFA with epsilon



NDFA with epsilon to DFA

- Consider the following NDFA with epsilon

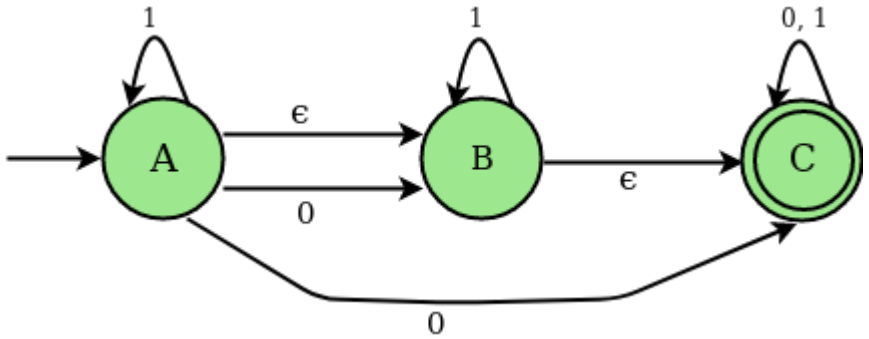


- ∈-closure(A) : {A, B, C}, ∈-closure(B) : {B, C}, ∈-closure(C) : {C}

NDFA with epsilon to DFA

- Consider the following NDFA with epsilon

STATES	0	1	EPSILON
A	B, C	A	B
B	–	B	C
C	C	C	–

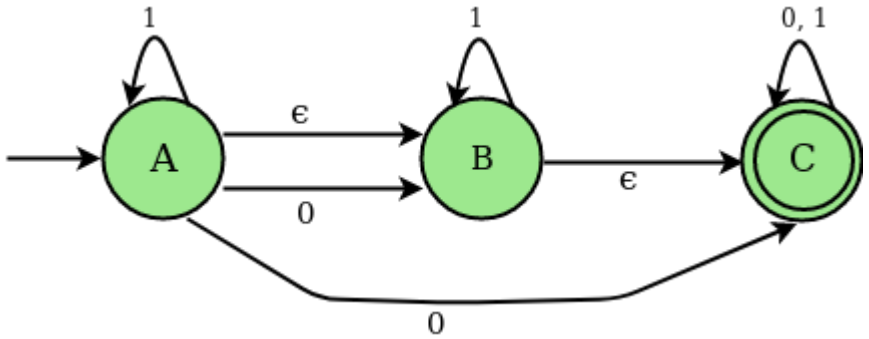


- ∈-closure(A) : {A, B, C}, ∈-closure(B) : {B, C}, ∈-closure(C) : {C}

NDFA with epsilon to DFA

- Consider the following NDFA with epsilon

STATES	0	1	EPSILON
A	B, C	A	B
B	–	B	C
C	C	C	–



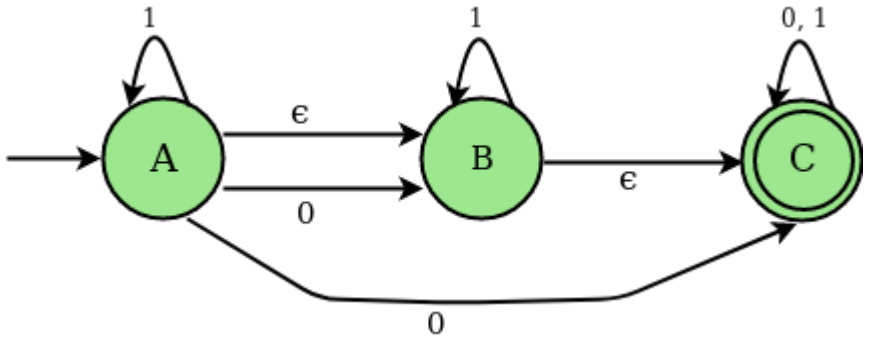
- ∈-closure(A) : {A, B, C}, ∈-closure(B) : {B, C}, ∈-closure(C) : {C}

STATES	0	1
A, B, C	B, C	A, B, C
B, C	C	B, C
C	C	C

NDFA with epsilon to DFA

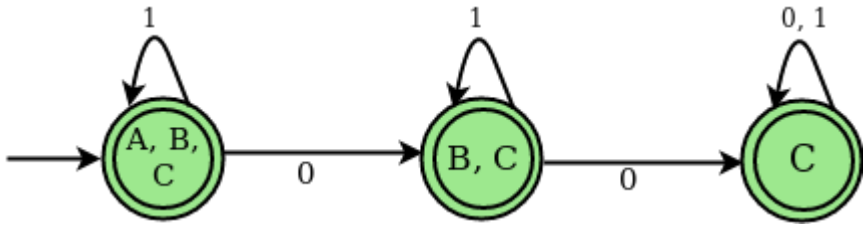
- Consider the following NDFA with epsilon

STATES	0	1	EPSILON
A	B, C	A	B
B	–	B	C
C	C	C	–



- ∈-closure(A) : {A, B, C}, ∈-closure(B) : {B, C}, ∈-closure(C) : {C}

STATES	0	1
A, B, C	B, C	A, B, C
B, C	C	B, C
C	C	C

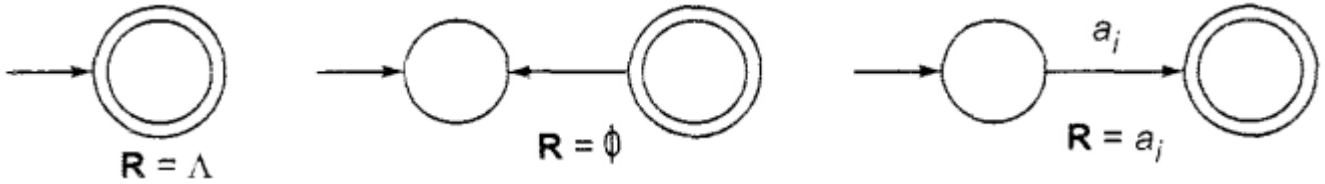


Kleens Theorem-Statement

- It states that any regular language is accepted by an FA and conversely that any language accepted by an FA is regular.
- So it can be divided into two parts
- Part 1: Any regular language is accepted by a finite automaton.
- Part 2: Language accepted by finite automaton is regular.

Kleens Theorem-Part 1

- Part 1: Any regular language is accepted by a finite automaton.
- We will prove this through Mathematical Induction:
- Now as we know that corresponding to every regular language there must be a regular expression.
- Basic Step: Smallest regular expression will be either ε, $\varnothing$, or a single symbol. FA, corresponding to these will be



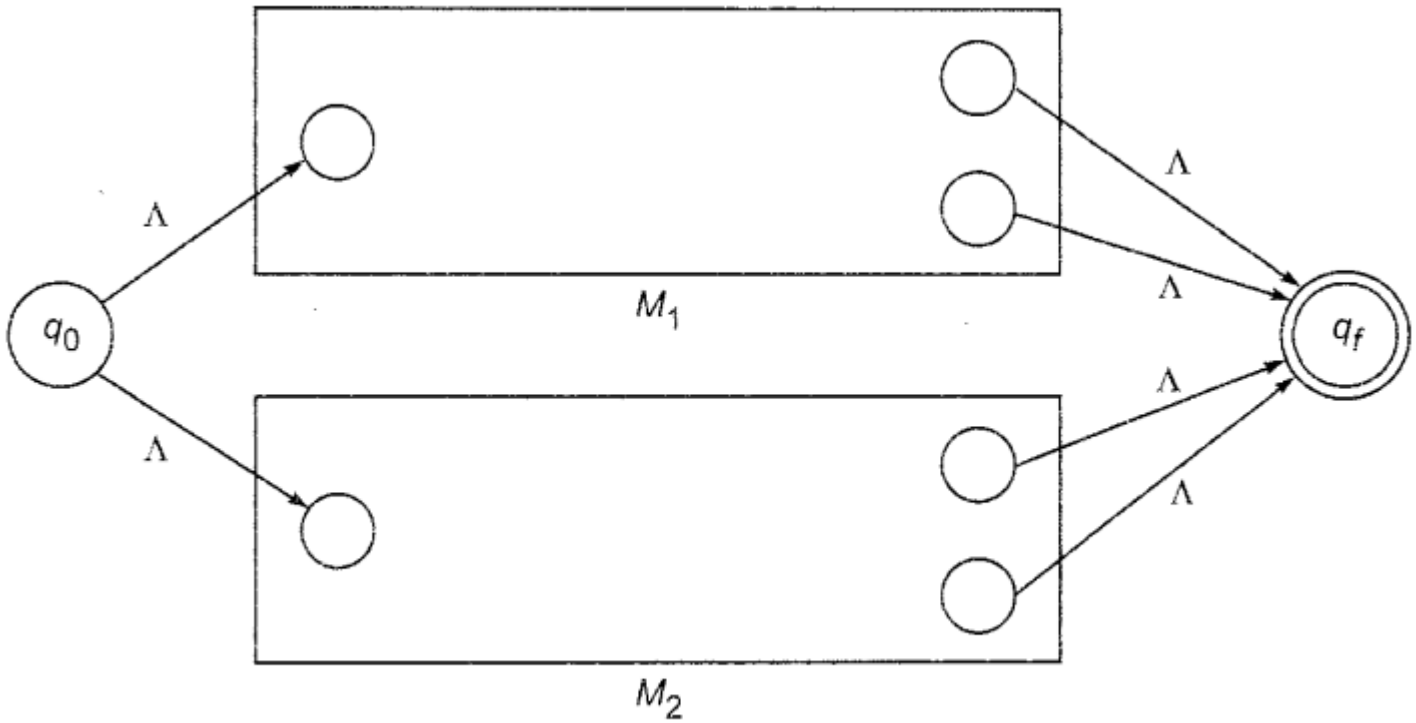
- So we have proved that there exist FA for these

Kleens Theorem-Part 1

- Induction Hypothesis: Now we assume that exist FA corresponding to a RE having n characters
- Step 3: Now we have to prove that there exist FA corresponding to a RE R having n+1 characters. Now new character can be introduced through union, concatenation or closure operations only.

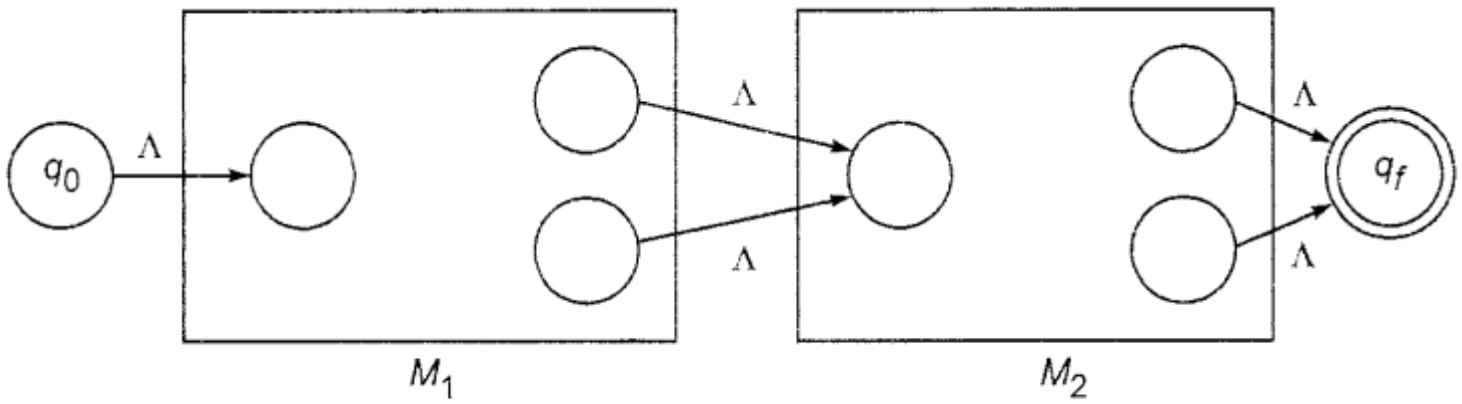
  R=P+Q R=PQ R=P*
- Here P & Q are regular expressions having n characters, so there must exist FA corresponding to P & Q. By induction Hypothesis, M1 and M2 are FA corresponding to L(P) and L(Q).

Kleens Theorem-Part 1



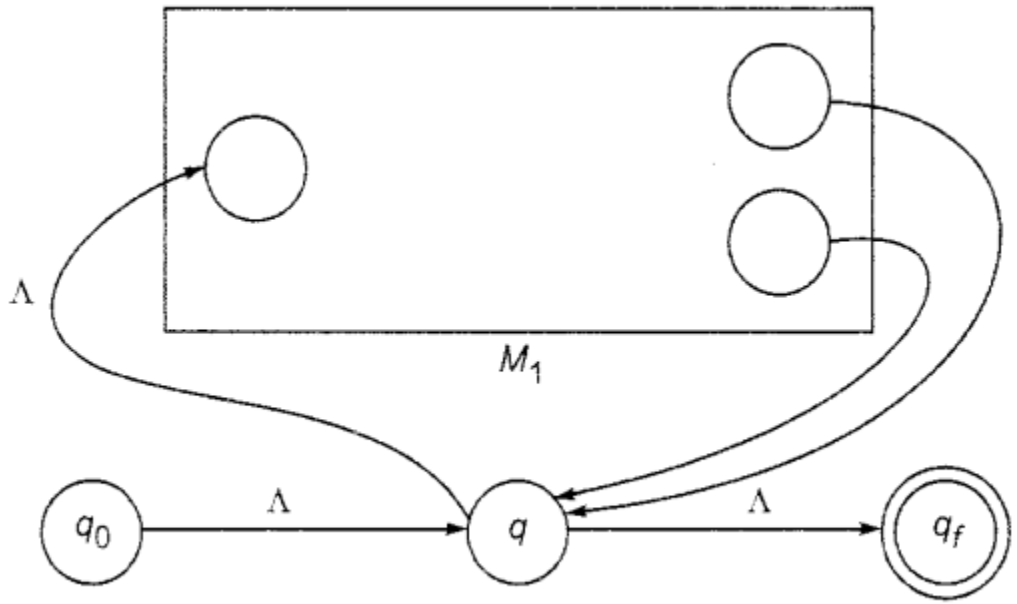
NDFA accepting $L(\mathbf{P} + \mathbf{Q})$

Kleens Theorem-Part 1



NDFA accepting $L(\mathbf{PQ})$

Kleens Theorem-Part 1



NDFA accepting $L(\mathbf{P}^*)$

Steps to find FA of a RE

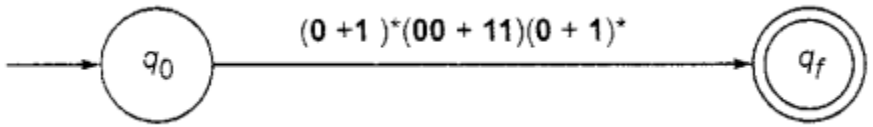
- Construct the transition graph equivalent to the given regular expression using epsilon moves.
- Construct the transition table for the transition graph constructed in previous step.
- Convert the NDFA with epsilon constructed in previous step into the equivalent DFA

FA of a RE - Numerical

Construct the finite automaton equivalent to the regular expression $(\mathbf{0} + \mathbf{1})^*(\mathbf{00} + \mathbf{11})(\mathbf{0} + \mathbf{1})^*$

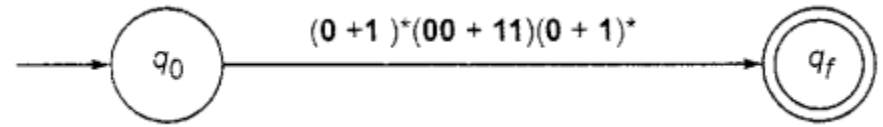
FA of a RE - Numerical

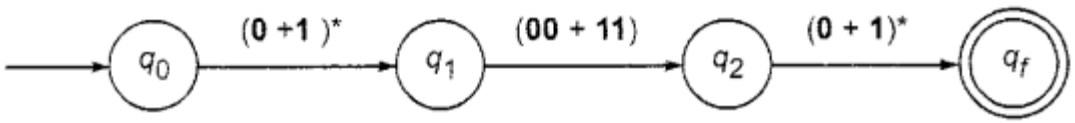
Construct the finite automaton equivalent to the regular expression $(0 + 1)^*(00 + 11)(0 + 1)^*$



FA of a RE - Numerical

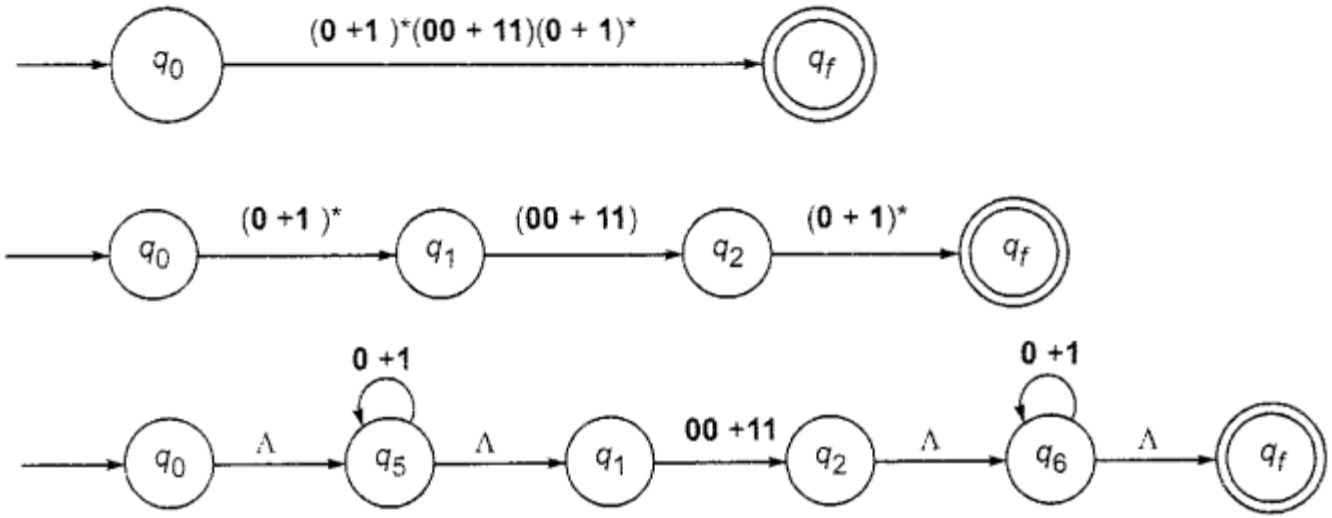
Construct the finite automaton equivalent to the regular expression $(0 + 1)^*(00 + 11)(0 + 1)^*$





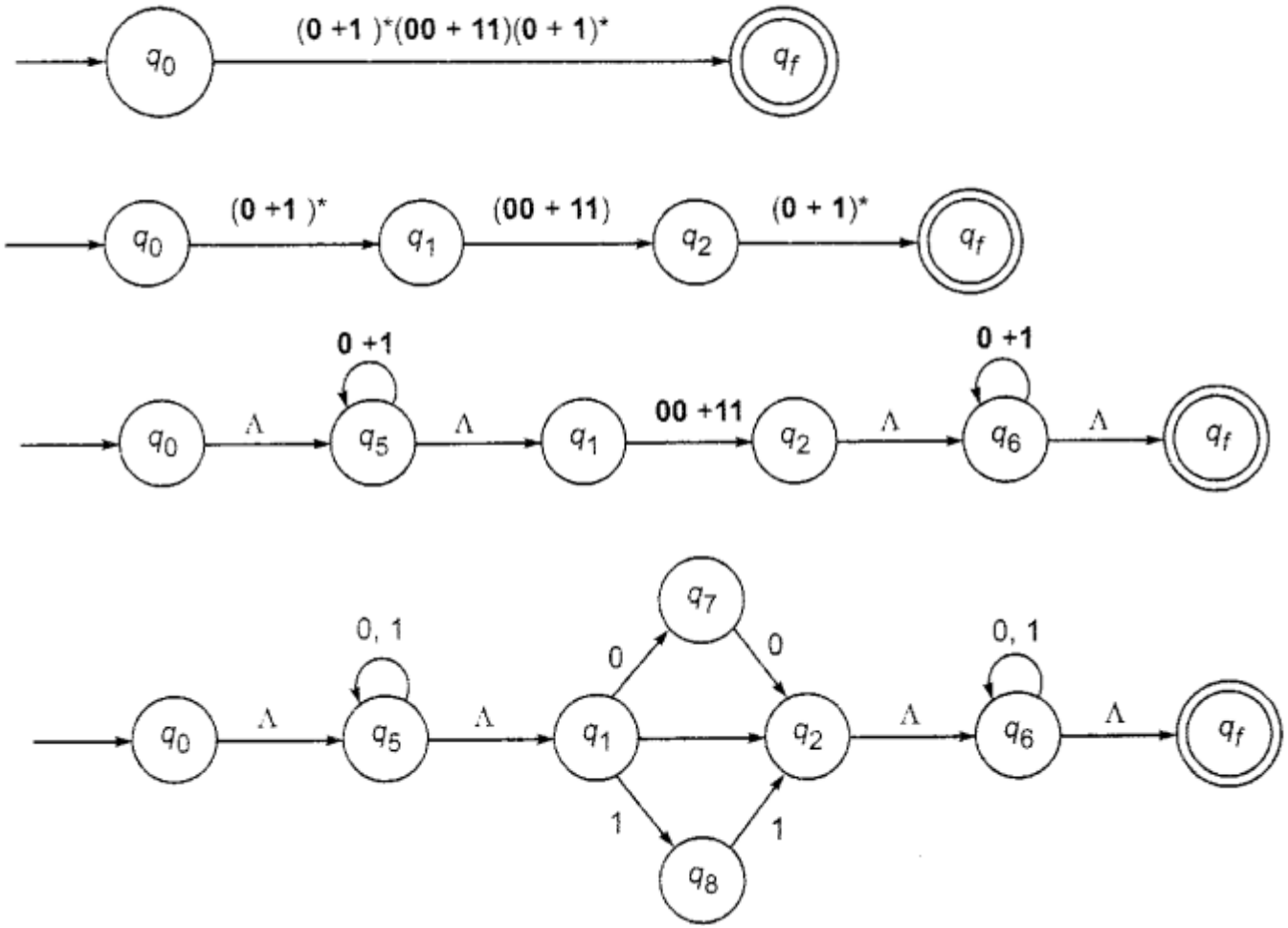
FA of a RE - Numerical

Construct the finite automaton equivalent to the regular expression $(0 + 1)^*(00 + 11)(0 + 1)^*$



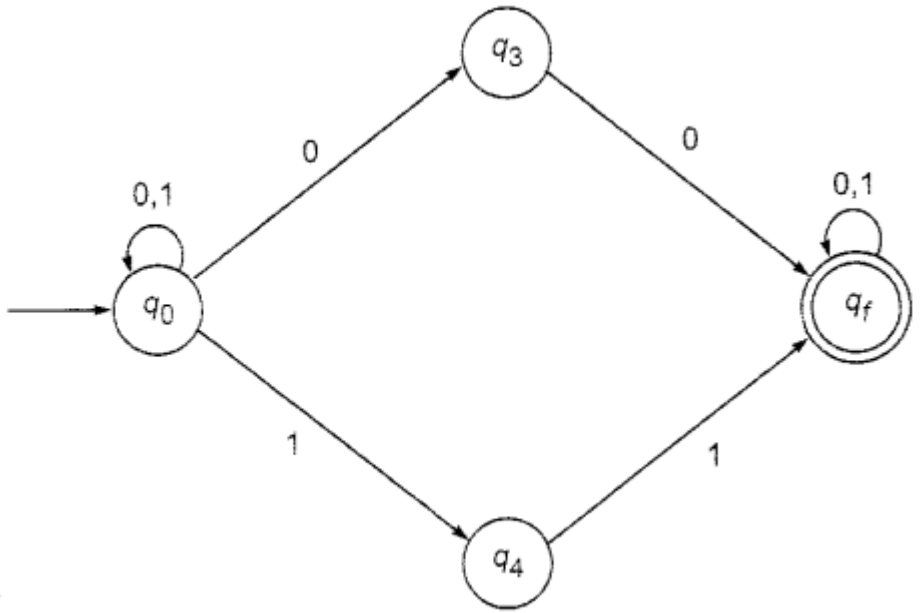
FA of a RE - Numerical

Construct the finite automaton equivalent to the regular expression $(0 + 1)^*(00 + 11)(0 + 1)^*$



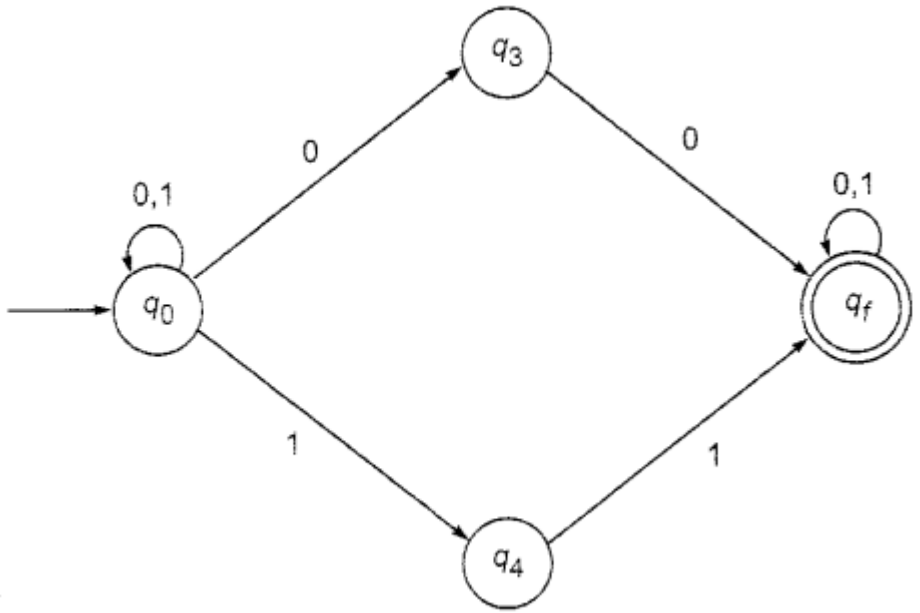
FA of a RE - Numerical

Construct the finite automaton equivalent to the regular expression $(\mathbf{0} + \mathbf{1})^*(\mathbf{00} + \mathbf{11})(\mathbf{0} + \mathbf{1})^*$



FA of a RE - Numerical

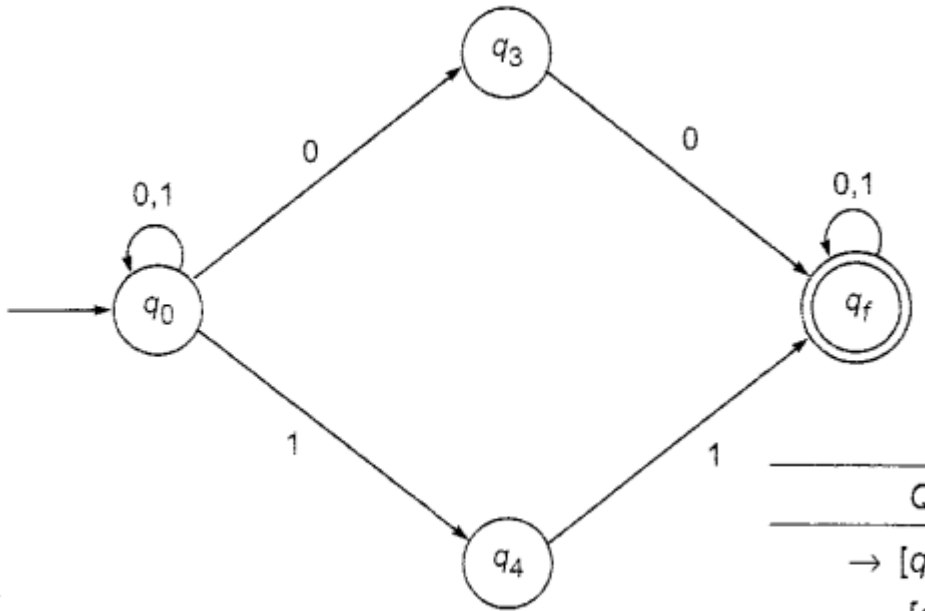
Construct the finite automaton equivalent to the regular expression $(\mathbf{0} + \mathbf{1})^*(\mathbf{00} + \mathbf{11})(\mathbf{0} + \mathbf{1})^*$



State/Σ	0	1
→ q_0	q_0, q_3	q_0, q_4
q_3	q_f	
q_4		q_f
(q_f)	q_f	q_f

FA of a RE - Numerical

Construct the finite automaton equivalent to the regular expression $(\mathbf{0} + \mathbf{1})^*(\mathbf{00} + \mathbf{11})(\mathbf{0} + \mathbf{1})^*$

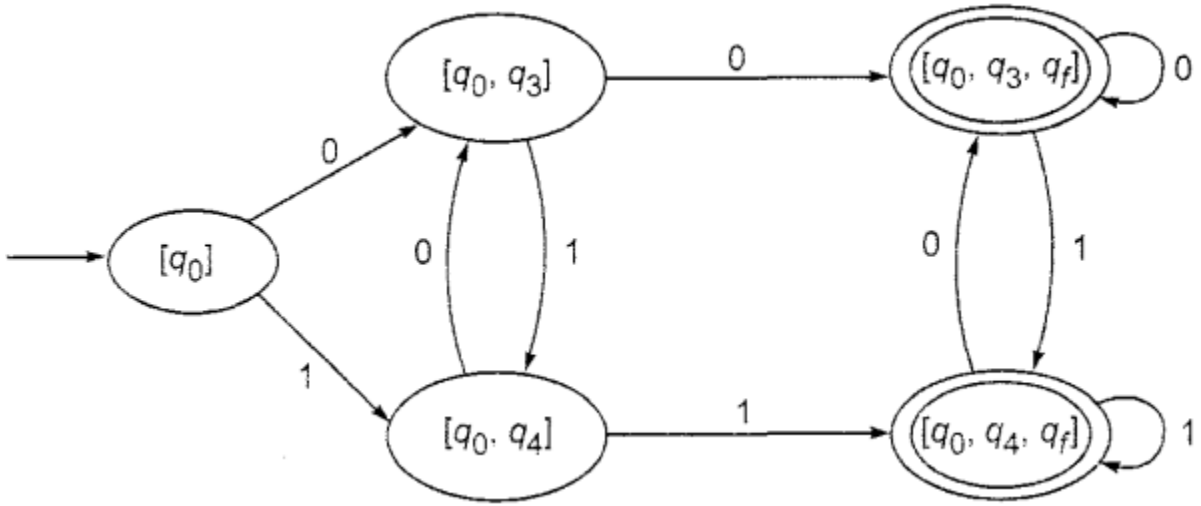


State/Σ	0	1
→ q_0	q_0, q_3	q_0, q_4
q_3	q_f	
q_4		q_f
(q_f)	q_f	q_f

Q	0	1
→ $[q_0]$	$[q_0, q_3]$	$[q_0, q_4]$
$[q_0, q_3]$	$[q_0, q_3, q_f]$	$[q_0, q_4]$
$[q_0, q_4]$	$[q_0, q_3]$	$[q_0, q_4, q_f]$
$[q_0, q_3, q_f]$	$[q_0, q_3, q_f]$	$[q_0, q_4, q_f]$
$[q_0, q_4, q_f]$	$[q_0, q_3, q_f]$	$[q_0, q_4, q_f]$

FA of a RE - Numerical

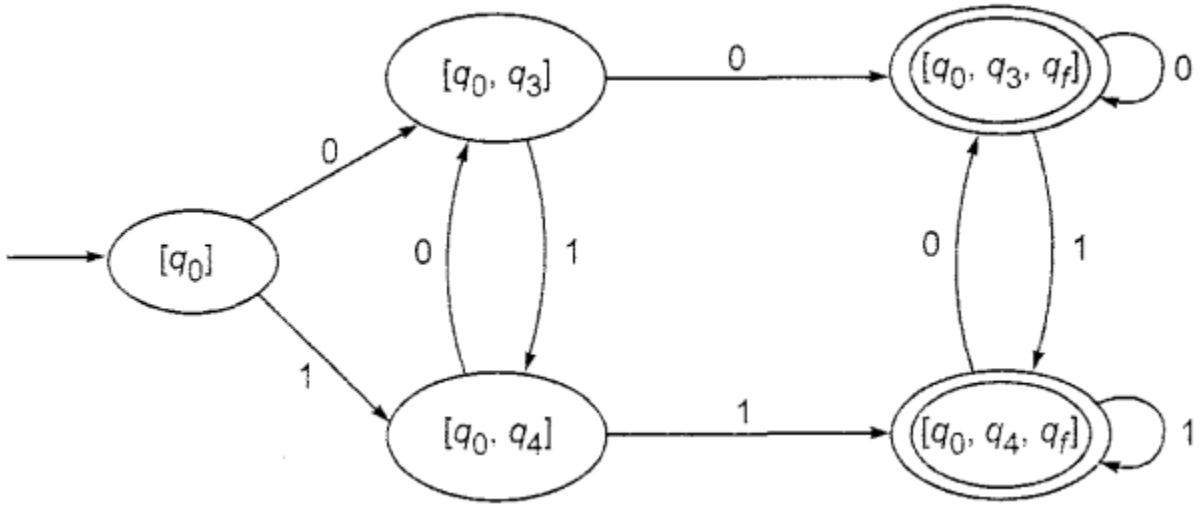
Construct the finite automaton equivalent to the regular expression $(\mathbf{0} + \mathbf{1})^*(\mathbf{00} + \mathbf{11})(\mathbf{0} + \mathbf{1})^*$

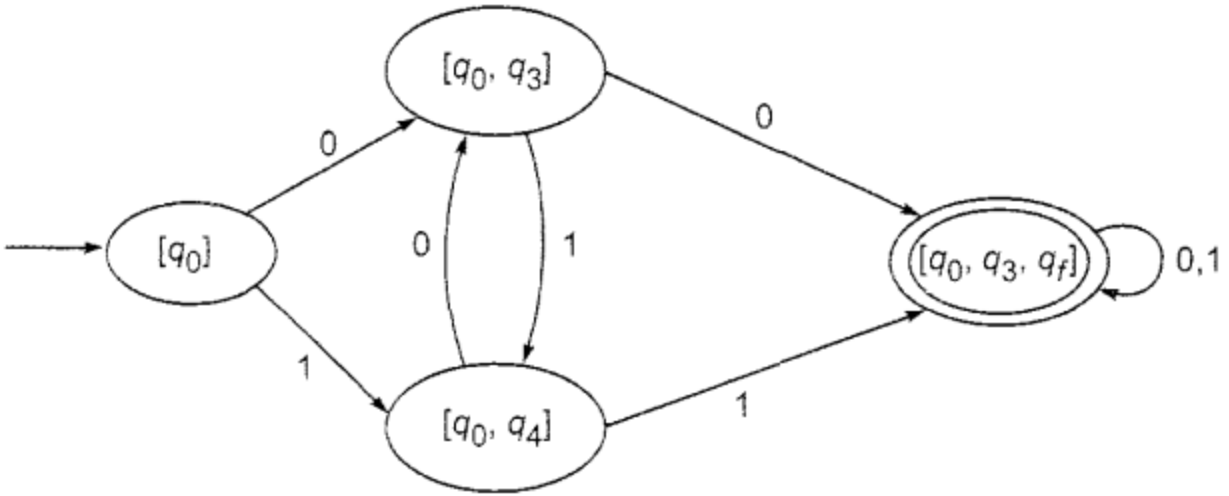


Q	0	1
→ $[q_0]$	$[q_0, q_3]$	$[q_0, q_4]$
$[q_0, q_3]$	$[q_0, q_3, q_f]$	$[q_0, q_4]$
$[q_0, q_4]$	$[q_0, q_3]$	$[q_0, q_4, q_f]$
$[q_0, q_3, q_f]$	$[q_0, q_3, q_f]$	$[q_0, q_4, q_f]$
$[q_0, q_4, q_f]$	$[q_0, q_3, q_f]$	$[q_0, q_4, q_f]$

FA of a RE - Numerical

Construct the finite automaton equivalent to the regular expression $(\mathbf{0} + \mathbf{1})^*(\mathbf{00} + \mathbf{11})(\mathbf{0} + \mathbf{1})^*$



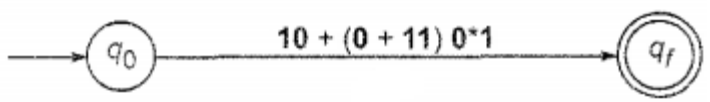


FA of a RE - Numerical

Construct a DFA with reduced states equivalent to the r.e. $\mathbf{10 + (0 + 11))0^*1}$

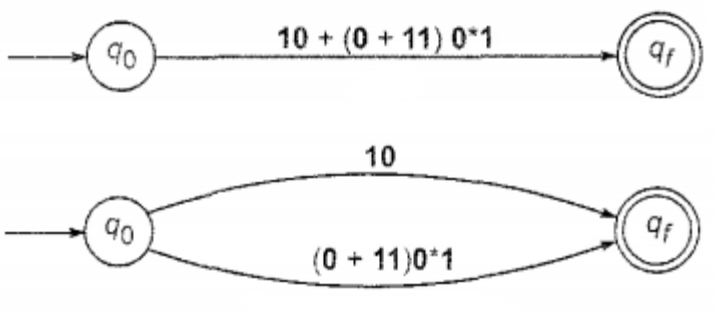
FA of a RE - Numerical

Construct a DFA with reduced states equivalent to the r.e. **10 + (0 + 11))0*1**



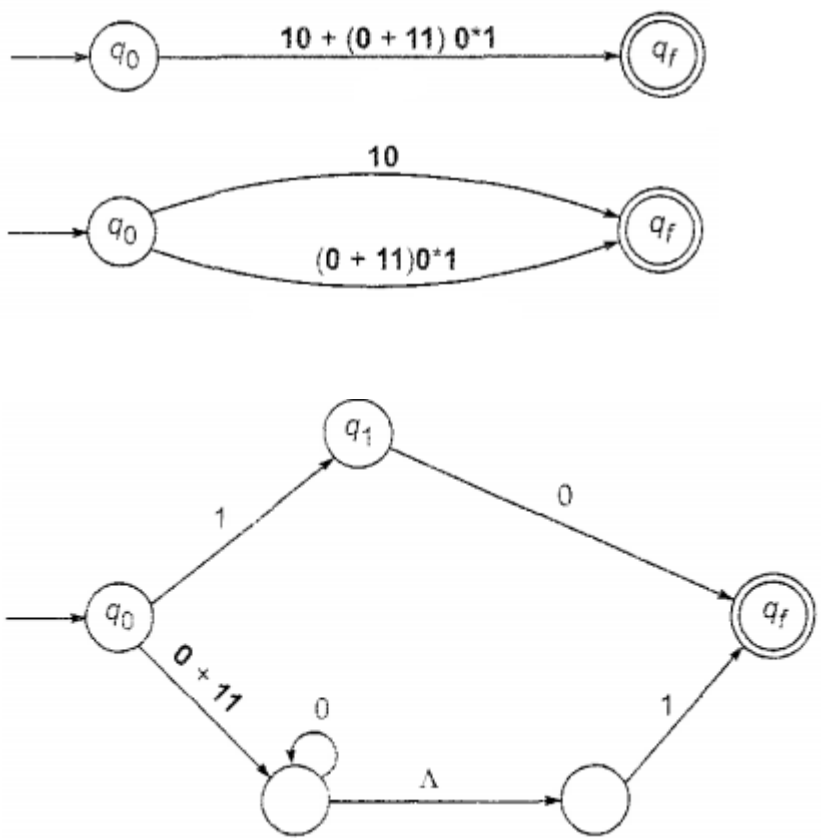
FA of a RE - Numerical

Construct a DFA with reduced states equivalent to the r.e. **10 + (0 + 11))0*1**



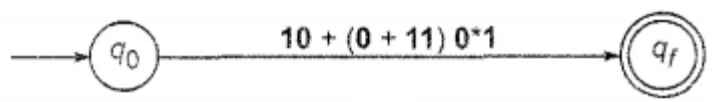
FA of a RE - Numerical

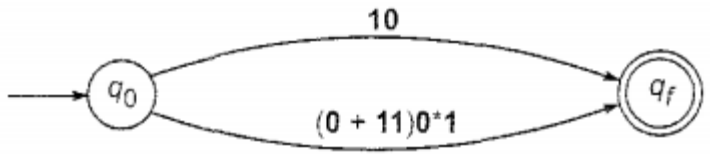
Construct a DFA with reduced states equivalent to the r.e. **10 + (0 + 11))0*1**

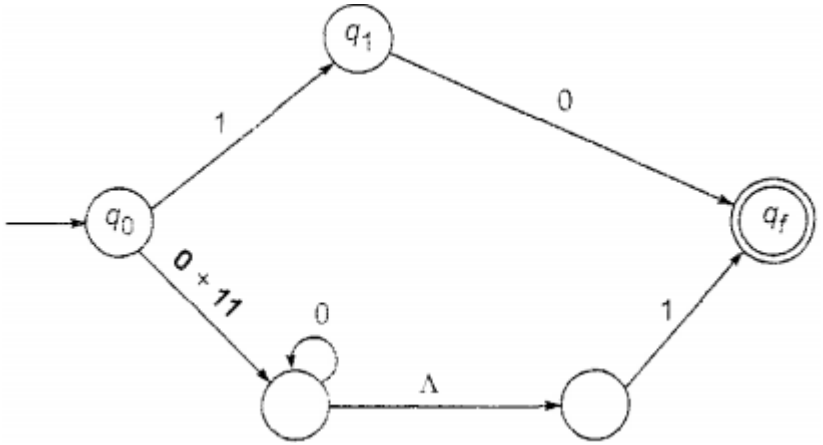


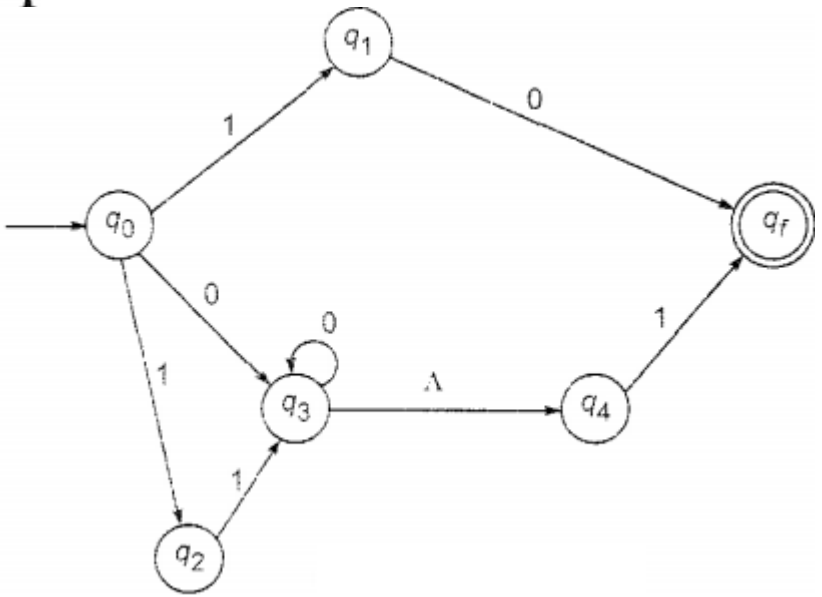
FA of a RE - Numerical

Construct a DFA with reduced states equivalent to the r.e. **10 + (0 + 11))0*1**



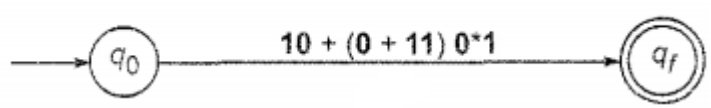


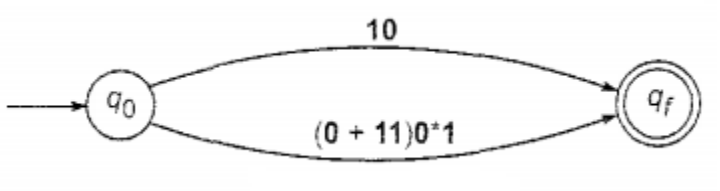


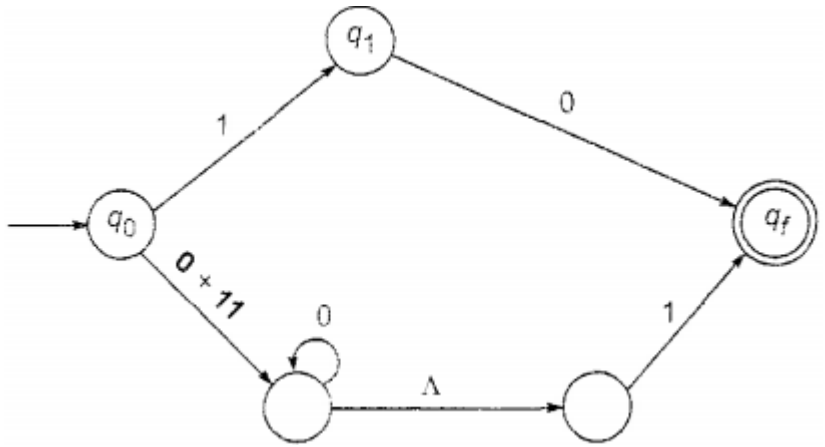


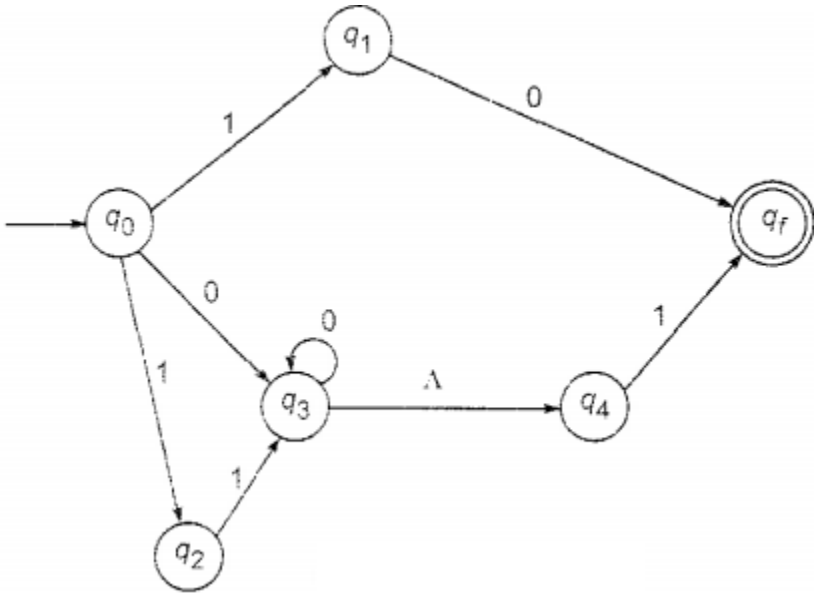
FA of a RE - Numerical

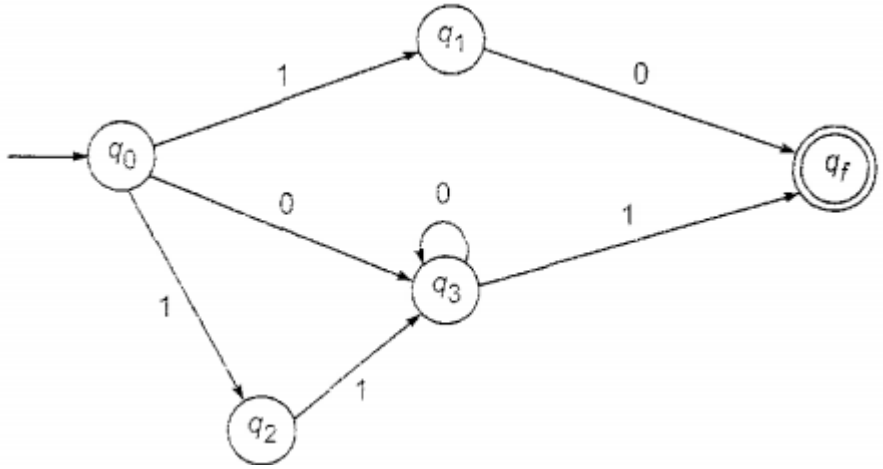
Construct a DFA with reduced states equivalent to the r.e. **10 + (0 + 11))0*1**





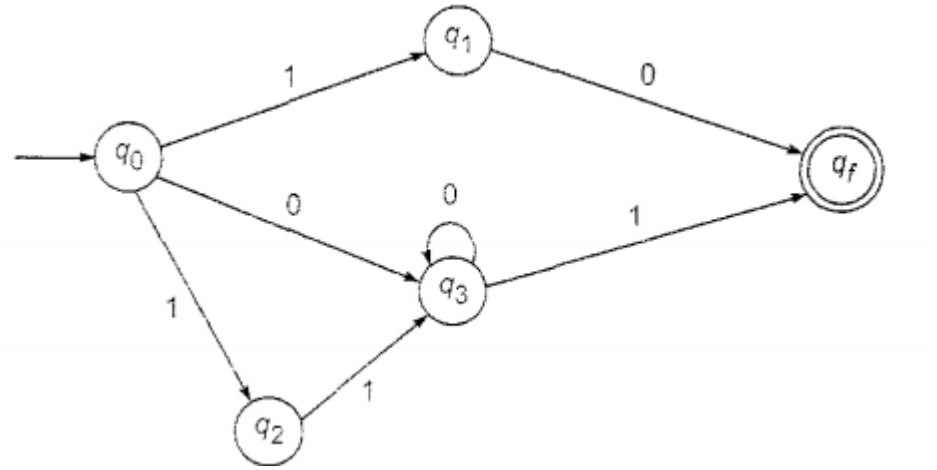






FA of a RE - Numerical

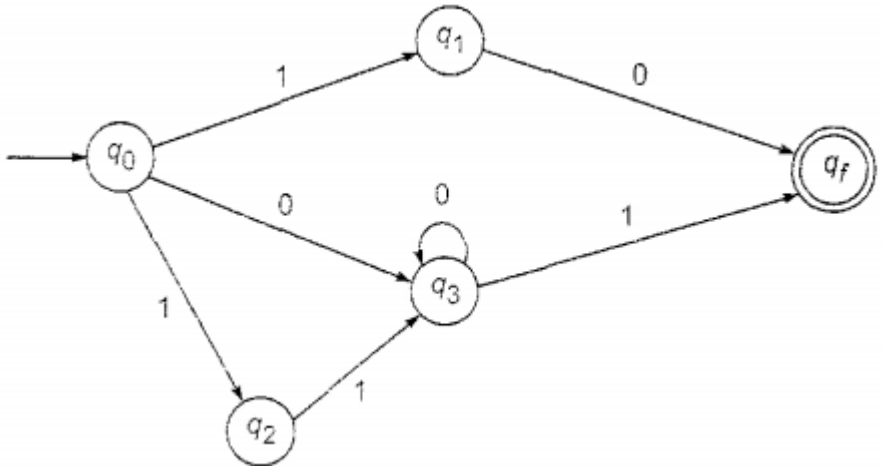
Construct a DFA with reduced states equivalent to the r.e. **10 + (0 + 11))0*1**



State/Σ	0	1
→ q_o	q_3	q_1, q_2
q_1	q_f	
q_2		q_3
q_3	q_3	q_f
(q_f)		

FA of a RE - Numerical

Construct a DFA with reduced states equivalent to the r.e. **10 + (0 + 11))0*1**

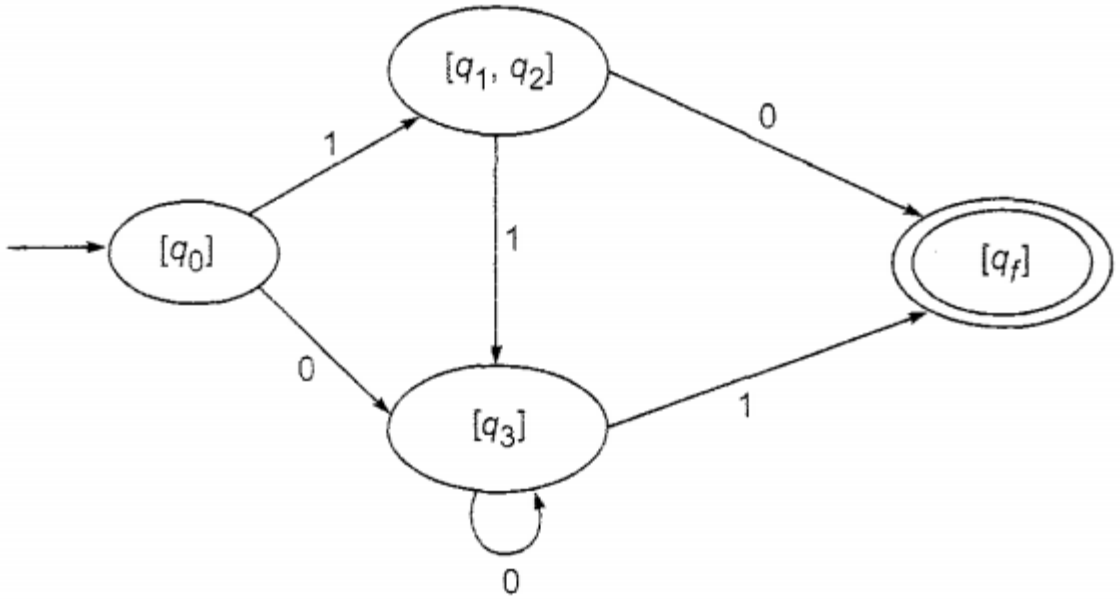


State/Σ	0	1
→ q_o	q_3	q_1, q_2
q_1	q_f	
q_2		q_3
q_3	q_3	q_f
(q_f)		

Q	0	1
→ $[q_0]$	$[q_3]$	$[q_1, q_2]$
$[q_3]$	$[q_3]$	$[q_f]$
$[q_1, q_2]$	$[q_f]$	$[q_3]$
($[q_f]$)	∅	∅
∅	∅	∅

FA of a RE - Numerical

Construct a DFA with reduced states equivalent to the r.e. **10 + (0 + 11))0*1**



Q	0	1
$\rightarrow [q_0]$	$[q_3]$	$[q_1, q_2]$
$[q_3]$	$[q_3]$	$[q_f]$
$[q_1, q_2]$	$[q_f]$	$[q_3]$
$(\overline{[q_f]})$	$\emptyset$	$\emptyset$
$\emptyset$	$\emptyset$	$\emptyset$

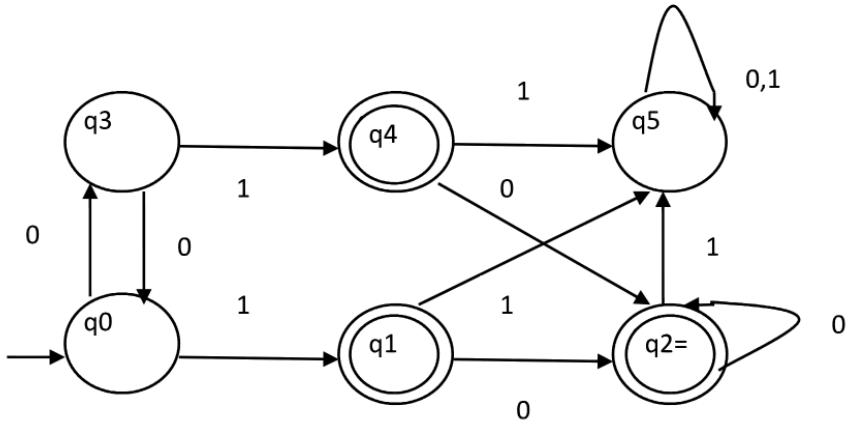
Partition Method of Minimization of DFA

- Suppose there is a DFA D < Q, Σ, q_0, δ, F > which recognizes a language L. Then the minimized DFA D < Q', Σ, q_0, δ', F' > can be constructed for language L as:
  - Step 1: We will divide Q (set of states) into two sets. One set will contain all final states and other set will contain non-final states. This partition is called P_0.
  - Step 2: Initialize k = 1
  - Step 3: Find P_k by partitioning the different sets of P_{k-1}. In each set of P_{k-1}, we will take all possible pair of states. If two states of a set are distinguishable, we will split the sets into different sets in P_k.
  - Step 4: Stop when $P_k = P_{k-1}$ (No change in partition)
  - Step 5: All states of one set are merged into one. No. of states in minimized DFA will be equal to no. of sets in P_k.

Partition Method of Minimization of DFA

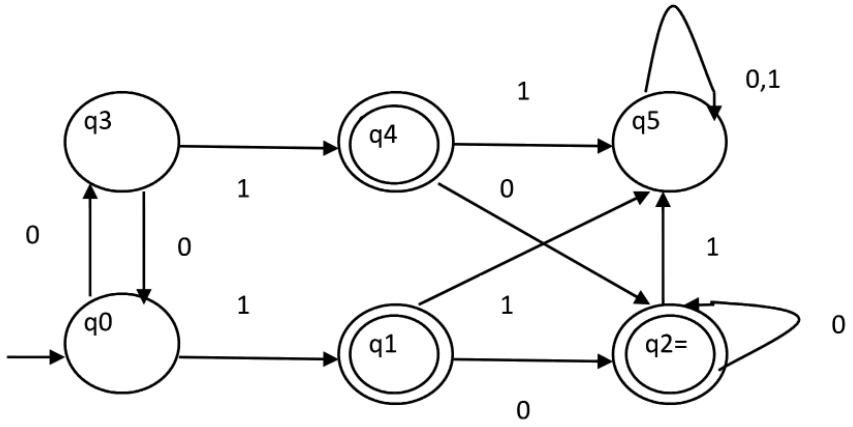
- How to find whether two states in partition P_k are distinguishable?
  - Two states (q_i, q_j) are distinguishable in partition P_k if for any input symbol a, δ (q_i, a) and δ (q_j, a) are in different sets in partition P_{k-1}.

Partition Method of Minimization of DFA - Example



Partition Method of Minimization of DFA - Example

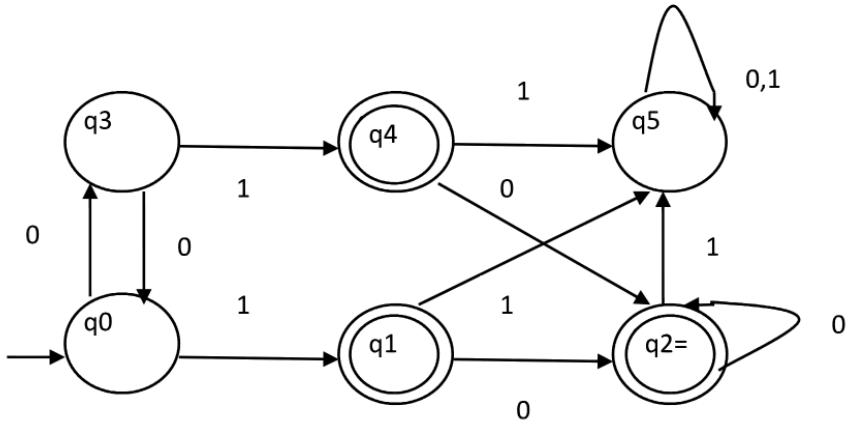
$P_0 = \{ \{ q_1, q_2, q_4 \}, \{ q_0, q_3, q_5 \} \}$.



Partition Method of Minimization of DFA - Example

$P_0 = \{ \{ q_1, q_2, q_4 \}, \{ q_0, q_3, q_5 \} \}$.

$P_1 = \{ \{ q_1, q_2, q_4 \}, \{ q_0, q_3 \}, \{ q_5 \} \}$

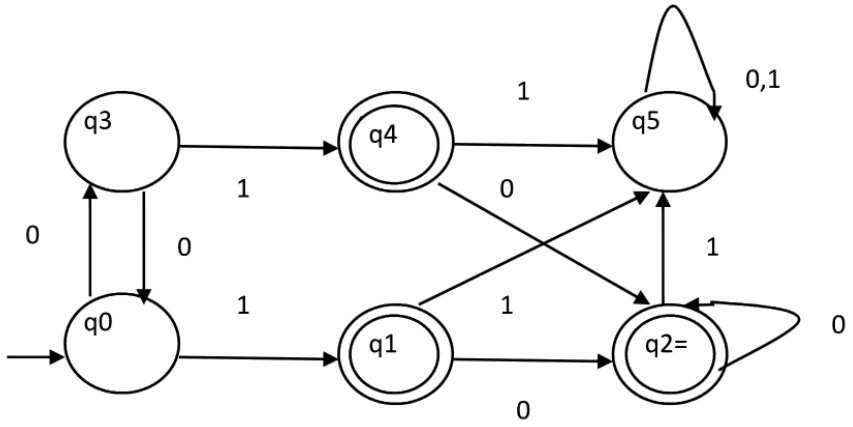


Partition Method of Minimization of DFA - Example

$P_0 = \{ \{ q_1, q_2, q_4 \}, \{ q_0, q_3, q_5 \} \}$.

$P_1 = \{ \{ q_1, q_2, q_4 \}, \{ q_0, q_3\}, \{ q_5 \} \}$

$P_2 = \{ \{ q_1, q_2, q_4 \}, \{ q_0, q_3\}, \{ q_5 \} \}$

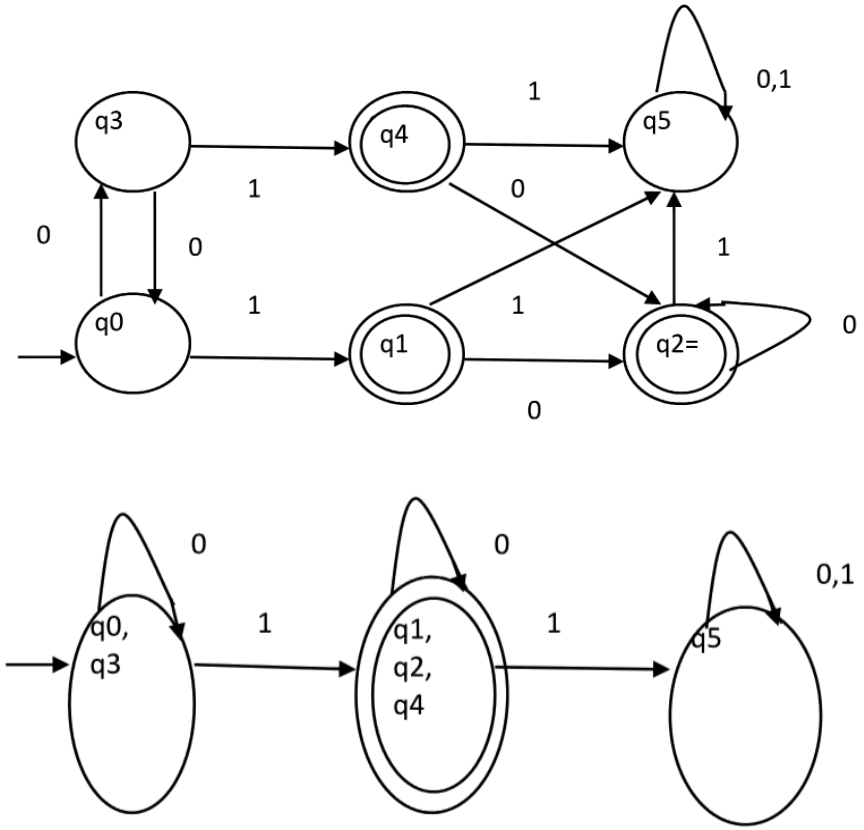


Partition Method of Minimization of DFA - Example

$P_0 = \{ \{ q_1, q_2, q_4 \}, \{ q_0, q_3, q_5 \} \}$.

$P_1 = \{ \{ q_1, q_2, q_4 \}, \{ q_0, q_3\}, \{ q_5 \} \}$

$P_2 = \{ \{ q_1, q_2, q_4 \}, \{ q_0, q_3\}, \{ q_5 \} \}$



THANK YOU

UPES
UNIVERSITY WITH A PURPOSE

Dr Anurag Jain
Assistant Professor (SG), Virtualization Department, School of Computer Science,
University of Petroleum & Energy Studies, Dehradun - 248 007 (Uttarakhand)
Email: anurag.jain@ddn.upes.ac.in , dr.anuragjain14@gmail.com
Mobile: (+91) -9729371188