Syllabus : Unit – I : Fundamentals and Finite Automata

Strings - Alphabets and languages - Finite state systems – Basic Definitions - Finite Automata - Deterministic finite automata – Non deterministic finite automata - Equivalence of DFA and NFA - Equivalence of NFA with and without ε –moves - Minimization of FA - Finite automata with output – More machines and mealy machines.

Introduction

- **Definition of TOC**

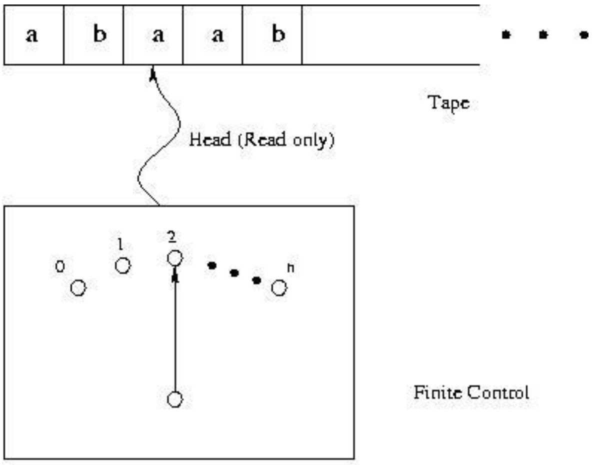
  TOC describes the basic ideas and models underlying computing. TOC suggests various abstract models of computation, represented mathematically.

- **History of Theory of Computation**
  - ✓ 1936 Alan Turing invented the Turing machine, and proved that there exists an unsolvable problem.
  - ✓ 1940's Stored-program computers were built.
  - ✓ 1943 McCulloch and Pitts invented finite automata.
  - ✓ 1956 Kleene invented regular expressions and proved the equivalence of regular expression and finite automata
  - ✓ 1956 Chomsky defined Chomsky hierarchy, which organized languages recognized by different automata into hierarchical classes.
  - ✓ 1959 Rabin and Scott introduced nondeterministic finite automata and proved its equivalence to (deterministic) finite automata.
  - ✓ 1950's-1960's More works on languages, grammars, and compilers
  - ✓ 1965 Hartmantis and Stearns defined time complexity, and Lewis, Hartmantis and Stearns defined space complexity.
  - ✓ 1971 Cook showed the first NP-complete problem, the satisfiability problem.
  - ✓ 1972 Karp Showed many other NP-complete problems.
  - ✓ 1976 Diffie and Helllman defined Modern Cryptography based on NP-complete problems.
  - ✓ 1978 Rivest, Shamir and Adelman proposed a public-key encryption scheme, RSA.

Finite State systems

A finite automaton can also be thought of as the device shown below consisting of a tape and a control circuit which satisfy the following conditions:

- ✓ The tape has the left end and extends to the right without an end.
- ✓ The tape is dividing into squares in each of which a symbol can be written prior to the start of the operation of the automaton.
- ✓ The tape has a read only head.
- ✓ The head is always at the leftmost square at the beginning of the operation.
- ✓ The head moves to the right one square every time it reads a symbol. It never moves to the left. When it sees no symbol, it stops and the automaton terminates its operation.
- ✓ There is a finite control which determines the state of the automaton and also controls the movement of the head.



Finite Automaton

Basic Definitions

- **Symbol :**

  Symbol is a character.
  Example : a,b,c,… , 0,1,2,3,….9 and special characters.

- **Alphabet :**

  An alphabet is a finite, nonempty set of symbol. It is denoted by ∑.
  Example :
  a) ∑ = {0,1}, the set of binary alphabet.
  b) ∑ = {a,b……..z}, the set of all lowercase letters.
  c) ∑ = {+, &,…..}, the set of all special characters.

- **String or Word :**

  A string is a finite set sequence of symbols chosen from some alphabets.
  Example :
  a) 0111010 is a string from the binary alphabet ∑ = {0,1}
  b) aabbaacab is a string from the alphabet ∑ = {a,b,c}

- **Empty String :**

  The empty string is the string with zero occurrences of symbols (no symbols). It is denoted by ϵ.

- **Length of String :**

  The length of a string is number of symbols in the string. It denoted by |w|.

  Example :
  w = 010110101 from binary alphabet ∑ = {0,1}
  Length of a string |w| = 9

✓ Power of an Alphabet:

- ✓ If ∑ is an alphabet, we can express the set of all strings of certain length from that alphabet by using an exponential notation. It is denoted by $\sum^k$ is the set of strings of length k, each of whose symbols is in ∑.

Example :

∑ = {0,1} has 2 symbols

i) $\sum^1$ = {0,1} (∴ $2^1 = 2$)

ii) $\sum^2$ = {00, 01, 10, 11} (∴ $2^2 = 4$)

iii) $\sum^3$ = {000,001,010,011,100,101,110,111} (∴ $2^3 = 8$)

- ✓ The set of strings over an alphabet ∑ is usually denoted by $\sum^*$.
  For instance, $\sum^* = \{0,1\}^* = \{\epsilon,0,1,00,01,10,11\}$
  (∴ $\sum^*=\sum^0\cup\sum^1\cup\sum^2\ldots\ldots$) - with ϵ symbol.

- ✓ The set of strings over an alphabet ∑ excluding ϵ is usually denoted by $\sum^+$.
  For instance, $\sum^+ = \{0,1\}^+ = \{0,1,00,01,10,11\}$
  (∴ $\sum^+=\sum^* - \{\epsilon\}$ or $\sum^1\cup\sum^2\cup\sum^3\ldots\ldots$)
  - without ϵ symbol.

✓ Concatenation of String

Join the two or more strings. Let x and y be two strings. Concatenation of strings x and y is appending symbols of y to right end of x.

$x = a_1a_2a_3\ldots\ldots\ldots\ldots\ldots a_n$ and $y = b_1b_2b_3\ldots\ldots\ldots\ldots\ldots b_n$

Concatenation of String $xy = a_1a_2a_3\ldots\ldots a_n\, b_1b_2b_3\ldots\ldots b_n$

Example :

s = ababa and t = cdcddc

Concatenation st = ababacdcddc

✓ Languages:

If Σ is an alphabet, and L ⊆ Σ* then L is a language.

Examples:

- The set of legal English words
- The set of legal C programs
- The set of strings consisting of n 0's followed by n 1's
  { ϵ, 01,0011,000111, …}

✓ Operations on Languages

- ✓ **Complementation**
  Let *L* be a language over an alphabet Σ. The complementation of *L*, denoted by $\overline{L}$, is Σ*–*L*.
- ✓ **Union**
  Let L_1 and L_2 be languages over an alphabet Σ. The union of L_1 and L_2, denoted by $L_1\cup L_2$, is {x | x is in L_1 or L_2}.
- ✓ **Intersection**
  Let L_1 and L_2 be languages over an alphabet Σ. The intersection of L_1 and L_2, denoted by $L_1\cap L_2$, is { x | x is in L_1 and L_2}.

- **Concatenation**
  Let L_1 and L_2 be languages over an alphabet Σ. The concatenation of L_1 and L_2, denoted by $L_1 \cdot L_2$, is $\{w_1 \cdot w_2 | w_1$ is in L_1 and w_2 is in $L_2\}$.
- **Reversal**
  Let L be a language over an alphabet Σ. The reversal of L, denoted by L^r, is $\{w^r | w$ is in L$\}$.
- **Kleene's closure**
  Let L be a language over an alphabet Σ. The Kleene's closure of L, denoted by L*, is {x | for an integer n ≥ 0 $x = x_1 x_2 \ldots x_n$ and $x_1, x_2, \ldots, x_n$ are in L}.

$$L^* = \bigcup_{i=0}^{\infty} L^i \qquad (\text{e.g. } a^* = \{\varepsilon, a, aa, aaa, \ldots\})$$

- **Positive Closure**
  Let L be a language over an alphabet Σ. The closure of L, denoted by L+, is { x |for an integer n ≥ 1, $x = x_1 x_2 \ldots x_n$ and x_1, x2 , …, x_n are in L}

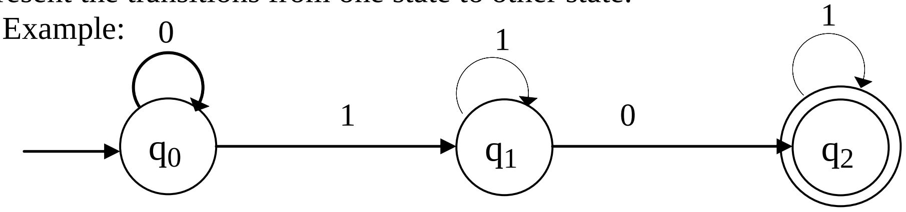
$$L^+ = \bigcup_{i=1}^{\infty} L^i \qquad (\text{e.g. } a^* = \{a, aa, aaa, \ldots\})$$

Finite Automata

Automaton is an abstract computing device. It is a mathematical model of a system, with discrete inputs, outputs, states and set of transitions from state to state that occurs on input symbols from alphabet Σ.

- **It representations:**
  - Graphical (Transition Diagram or Transition Table)
  - Tabular (Transition Table)
  - Mathematical (Transition Function or Mapping Function)
- **Formal Definition of Finite Automata**
  A finite automaton is a 5-tuples; they are $M=(Q, \Sigma, \delta, q_0, F)$
  where
  Q is a finite set called the states
  Σ is a finite set called the alphabet
  $\delta : Q \times \Sigma \rightarrow Q$ is the transition function
  $q_0 \in Q$ is the start state also called initial state
  $F \subseteq Q$ is the set of accept states, also called the final states

- **Transition Diagram (Transition graph)**
  It is a directed graph associated with the vertices of the graph corresponds to the states of the finite automata. (or) It is a 5-tuple graph used state and edges represent the transitions from one state to other state.
  Example:



Start or Initial State

Final or Accepting State

✓ **Transition Table.**

It is the tabular representation of the DFA. For a transition table the transition function is used.

Example:

States	Input	
	0	1
→{q0}	{q1}	{q0}
{q1}	-	{q2}
*{q2}	-	-

✓ **Transition Function.**

- The mapping function or transition function denoted by δ.
- Two parameters are passed to this transition function: (i) current state and (ii) input symbol.
- The transition function returns a state which can be called as next state.

δ (current_state, current_input_symbol) = next_state

Example:

δ (q0, a) = q1

✓ **Computation of a Finite Automaton**

- The automaton receives the input symbols one by one from left to right.
- After reading each symbol, M1 moves from one state to another along the transition that has that symbol as its label.
- When M1 reads the last symbol of the input it produces the output: accept if M1 is in an accept state, or reject if M1 is not in an accept state.

✓ **Applications**

- It plays an important role in complier design.
- In switching theory and design and analysis of digital circuits automata theory is applied.
- Design and analysis of complex software and hardware systems.
- To prove the correctness of the program automata theory is used.
- To design finite state machines such as Moore and mealy machines.
- It is base for the formal languages and these formal languages are useful of the programming languages.

✓ **Types of Finite Automata**

- Finite Automata without output
  - Deterministic Finite Automata (DFA)
  - Non-Deterministic Finite Automata (NFA or NDFA)
  - Non-Deterministic Finite Automata with ε move (ε-NFA or ε-NDFA)
- Finite Automata with output
  - Moore Machine
  - Mealy Machine

Deterministic Finite Automata (DFA)

Deterministic Finite Automaton is a FA in which there is **only one path for a specific input from current state to next state.** There is a unique transition on each input symbol.

✓ Formal Definition of Deterministic Finite Automata

A finite automaton is a 5-tuples; they are M=(Q, Σ, δ, q_0, F)

where

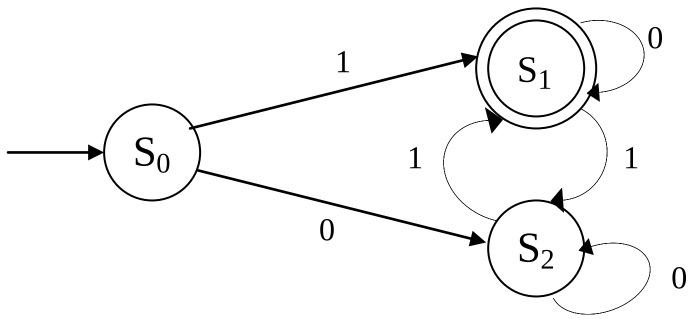
Q is a finite set called the states

Σ is a finite set called the alphabet

$\delta : Q \times \Sigma \rightarrow Q$ is the transition function

$q_0 \in Q$ is the start state also called initial state

$F \subseteq Q$ is the set of accept states, also called the final states



Non-Deterministic Finite Automata (NDFA or NFA)

Non-Deterministic Finite Automaton is a FA in which there **many paths for a specific input from current state to next state.**

✓ Formal Definition of Non-Deterministic Finite Automata

A finite automaton is a 5-tuples; they are M=(Q, Σ, δ, q_0, F)

where

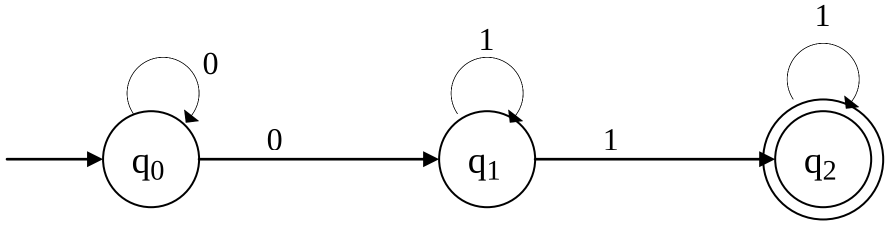
Q is a finite set called the states

Σ is a finite set called the alphabet

$\delta : Q \times \Sigma \rightarrow 2^Q$ is the transition function

$q_0 \in Q$ is the start state also called initial state

$F \subseteq Q$ is the set of accept states, also called the final states



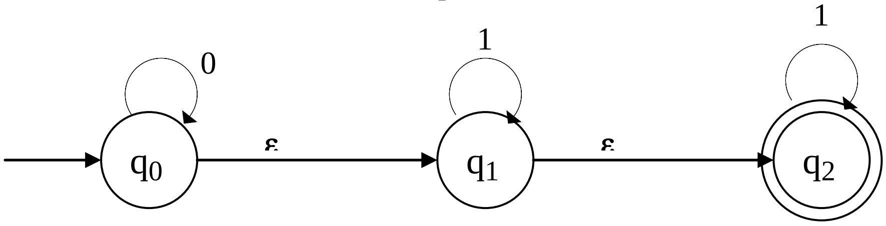
Finite Automaton with ε- moves

The finite automata is called NFA when there exists **many paths for a specific input or ε from current state to next state.** The **ε is a character** used to indicate null string.

- ✓ Formal Definition of Non-Deterministic Finite Automata

  A finite automaton is a 5-tuples; they are $M=(Q, \Sigma, \delta, q_0, F)$
  where

  Q is a finite set called the states
  Σ is a finite set called the alphabet
  $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$ is the transition function
  $q_0 \in Q$ is the start state also called initial state
  $F \subseteq Q$ is the set of accept states, also called the final states

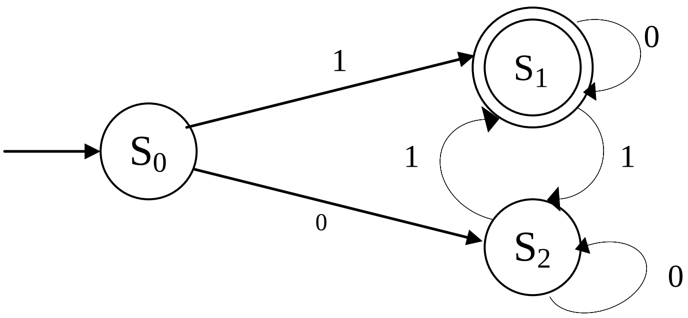


Differentiate DFA and NFA

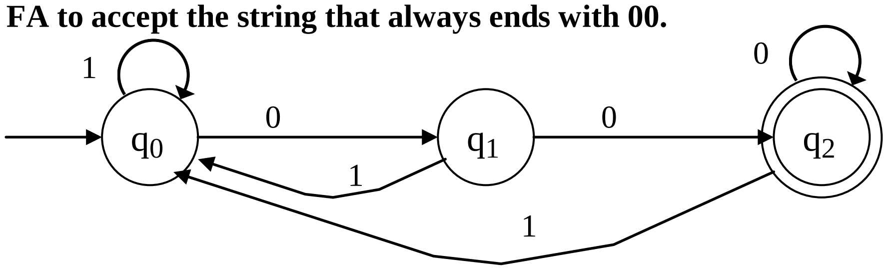
Sl. No	DFA	NFA
1.	DFA is Deterministic Finite Automata	NFA is Non-Deterministic Finite Automata
2.	For given state, on a given input we reach to deterministic and unique state.	For given state, on a given input we reach to more than one state.
3.	DFA is a subset of NFA	Need to convert NFA to DFA in the design of complier.
4.	$\delta : Q \times \Sigma \rightarrow Q$ Example: $\delta(q_0, a) = \{q_1\}$	$\delta : Q \times \Sigma \rightarrow 2^Q$ Example : $\delta(q_0, a) = \{q_1, q_2\}$

Problems for Finite Automata

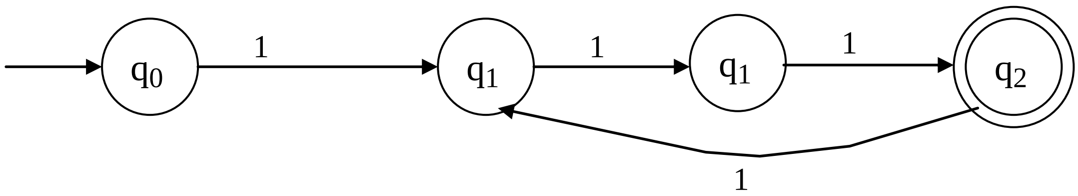
1. **Design FA which accepts odd number of 1's and any number of 0's.**



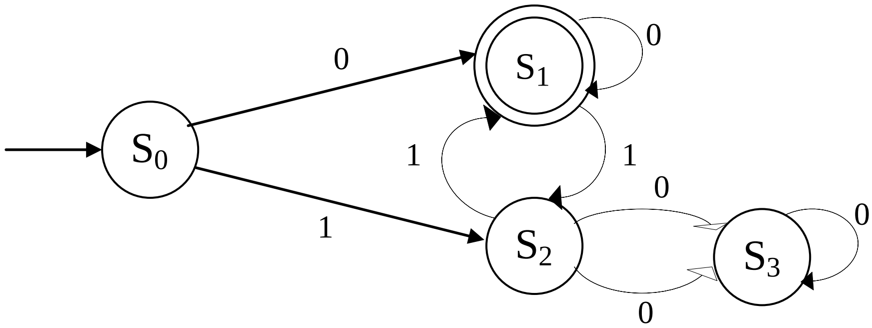
2. **Design FA to accept the string that always ends with 00.**



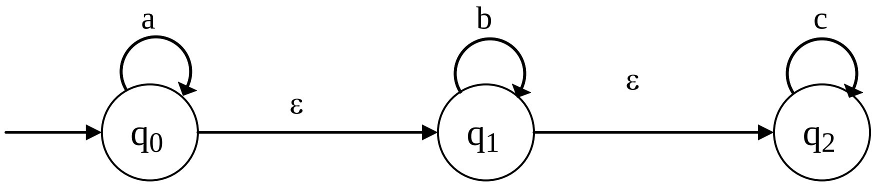
3. Design FA to check whether given unary number is divisible by three.



4. Design FA to check whether given binary number is divisible by three.



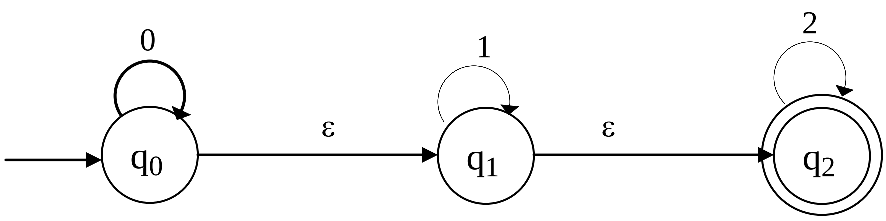
5. Obtain the ε closure of states q0 and q1 in the following NFA with ε transition.



Solution:

ε - CLOSURE {q0} = {q0, q1,q2}
ε - CLOSURE {q1} = {q1,q2}

6. Obtain ε closure of each state in the following NFA with ε move.



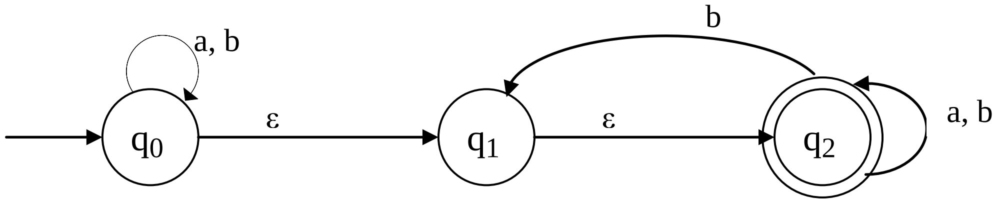
Solution:

ε - CLOSURE {q0} = {q0, q1,q2}
ε - CLOSURE {q1} = {q1,q2}
ε - CLOSURE {q2} = {q2}

<u>Tutorial:</u>

7. **Design Finite Automata which accepts the only 0010 over the input Σ = {0, 1}.**
8. **Design Finite Automata which checks whether given binary number is even or odd over the input Σ = {0, 1}.**
9. **Design Finite Automata which accepts only those strings which starts with 'a' and end with 'b' over the input Σ = {a, b}.**

10. **Design a DFA to accept the language L = {w | w has both an even number of 0's and an even number of 1's.**
11. **Design a DFA to accept the language L = {w | w has both an odd number of 0's and an odd number of 1's.**
12. **Obtain ε closure of each state in the following NFA with ε move.**



Equivalence of NFA and DFA

For every NFA, there exists an equivalent DFA.

Theorem:

For every NFA, there exists a DFA which simulates the behavior of NFA. If L is the set accepted by NFA, then there exists a DFA which also accepts L.

Or

Let L be a set accepted by NFA (L(M)), then there exists a DFA that accepts (L(M′)).

Proof:

Let M = (Q, Σ, δ, q_0, F) be an NFA for language L, then define DFA M′ = (Q′, Σ′, δ′, q_0′, F′).

- The states of M′ are all the subset of M.
- The elements in Q′ will be denoted by [q_1, q_2, q_3, … , q_i] and the elements in Q are denoted by {q_1, q_2, q_3, … , q_i}.
- Initial state of NFA is q_0, and also an initial state of DFA is q_0′ =[q_0].
- we define

  δ′ ([q_1, q_2, q_3, …, q_i],a) = [p_1, p_2, p_3, …, p_i]

  if only if

  δ({q_1, q_2, q_3, …, q_i},a) = {p_1, p_2, p_3, …, p_i}

  This means that whenever in NFA, at the current state {q_1, q_2, q_3, …, q_i} if we get input 'a' and it goes to the next states {p_1, p_2, p_3, …, p_i} then while constructing DFA for it the current state is assumed to be [q_1, q_2, q_3, …, q_i]. At this state, the input is 'a' and it goes to the next state is assumed to be [p_1, p_2, p_3, …, p_i]. On applying transition function on each of the state's q_1, q_2, q_3, …, q_i the new state may be any of the state's from p_1, p_2, p_3, …, p_i.

Theorem can be proved with the induction method by assuming length of input string 'x'.

δ′(q_0′, x) = [q_1, q_2, q_3, …, q_i]

if only if

δ(q_0, x) = {q_1, q_2, q_3, …, q_i}

Basis method:

If the input string length is 0. ie. **|x|=0** where x = {ε}, then $q_0' = [q_0]$.

Induction method:

If we assume that the hypothesis is true for the length of input string is less than or equal to 'm'. Then if 'xa' is a length of string is m+1. Hence the transition function (δ′) could be written as,

$$\boldsymbol{\delta' (q_0', xa) = \delta' (\delta' (q_0', x),a)}$$

By induction hypothesis,

$\delta'(q_0', x) = [p_1, p_2, p_3, \ldots, p_i]$

if only if

$\delta(q_0, x) = \{p_1, p_2, p_3, \ldots, p_i\}$

By definition of δ′

$\delta'([p_1, p_2, p_3, \ldots, p_i], a) = [r_1, r_2, r_3, \ldots, r_i]$

if only if

$\delta(\{p_1, p_2, p_3, \ldots, p_i\}, a) = \{r_1, r_2, r_3, \ldots, r_i\}$

Thus,

$\boldsymbol{\delta' (q_0', xa) = [r_1, r_2, r_3, \ldots, r_i]}$

if only if

$\boldsymbol{\delta (q_0, xa) = \{r_1, r_2, r_3, \ldots, r_i\}}$

Shown by induction hypothesis,

L(M) = L(M′)

Extended Transition Function (δ″ or δ^)

This is used to represent transition functions with a string of input symbols 'w' and returns a set of states. It is represented by δ″ or δ^

Suppose w = xa

$\delta (q, x) = \{p_1, p_2, p_3, \ldots, p_k\}$

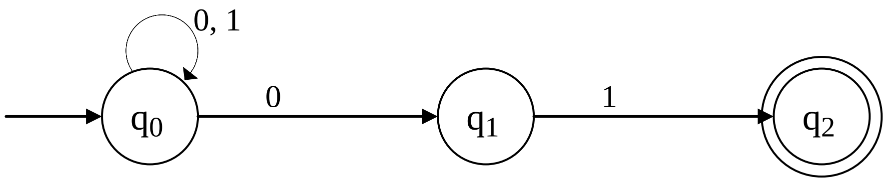
then

$$\bigcup_{i=0}^{\infty} \delta''(p_i,a) = \{r_1, r_2, r_3, \ldots, r_m\}$$

$\delta''(p_i, xa) = \delta''(\delta(q,x) a))$

Example Problems for Converting NFA into DFA

1. Obtain the DFA equivalent to the following NFA.



Solution :

The transition table for given **NFA** can be drawn as follows

States	Input	
	0	1
→{q0}	{q0}{q1}	{q0}
{q1}	-	{q2}
*{q2}	-	-

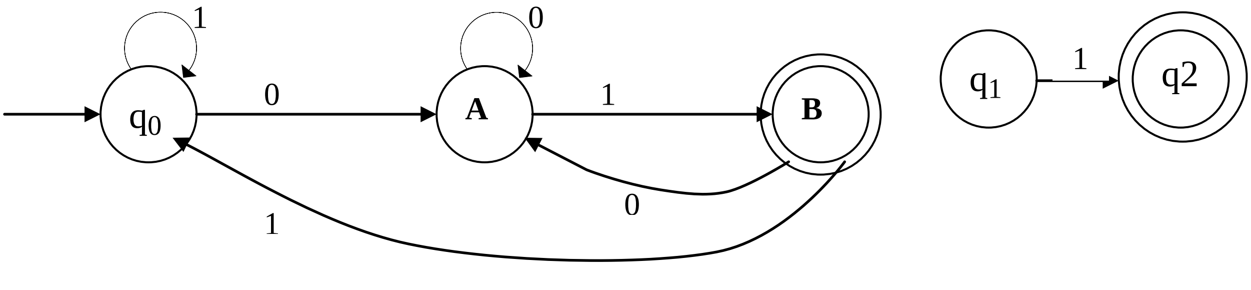
Let the DFA M′ = (Q′, Σ′, δ′, q_0′, F′) then, transition function (δ′) will be computed as,

δ′([q0], 0) = [q0, q1] - a new state - **A**
δ′([q0], 1) = [q0]
δ′([q1], 0) = -
δ′([q1], 1) = [q2]
δ′([q2], 0) = -
δ′([q2], 1) = -
δ′([qo,q1],0) = [q0,q1]
δ′([qo,q1],1) = [q0,q2] a new state - **B**
δ′([qo,q2],0) = [q0,q1]
δ′([qo,q2],0) = [q0]

The transition table for DFA

States	Input	
	0	1
→[q0]	[q0, q1]	[q0]
[q1]	-	[q2]
*[q2]	-	-
[q0, q1]	[q0, q1]	[q0, q2]
*[q0, q2]	[q0, q1]	[q0]

The transition diagram for DFA



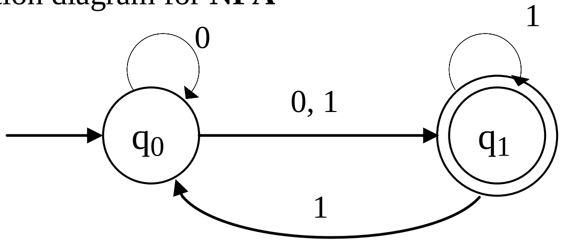
2. Let M = ({q0, q1}, {0,1}, δ, q0, {q1}) be NFA. Where δ (q0, 0) = {q0, q1}, δ (q0, 1) = {q1}, δ (q1, 0) = {ϕ}, δ (q1, 1) = {q0, q1}. Construct its equivalent DFA.

Solution :

The transition table for **NFA**

States	Input	
	0	1
→{q0}	{q0}{q1}	{q1}
*{q1}	ϕ	{q0}{q1}

The transition diagram for **NFA**



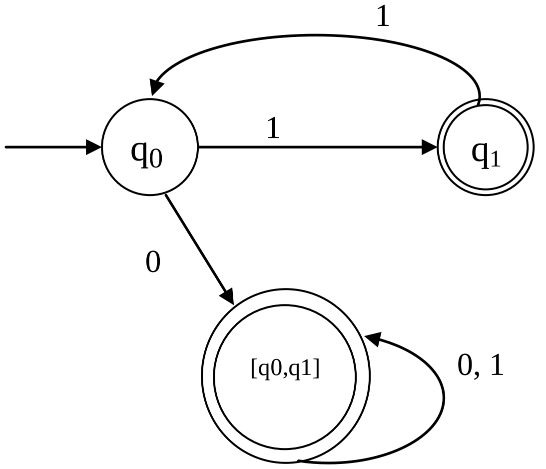
Let the DFA M′ = (Q′, Σ′, δ′, q_0′, F′) then, transition function (δ′) will be computed as,

δ′ ([q0], 0) = [q0, q1] -a new state **A**
δ′([q0], 1) = [q1]
δ′([q1], 0) = ϕ
δ′([q1], 1) = [q0]
δ′([q0,q1],0) = [q0,q1]
δ′([qo,q1],1) = [q0,q1]

The transition table for DFA

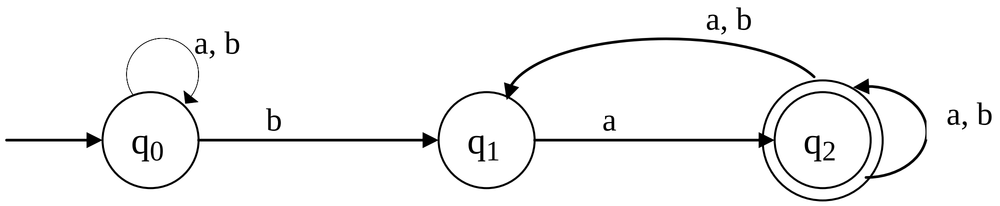
States	Input	
	0	1
→[q0]	[q0, q1]	[q1]
*[q1]	ϕ	[q0]
*[q0, q1]	[q0, q1]	[q0, q1]

The transition diagram for DFA



Tutorial:

3. **Obtain the DFA equivalent to the following NFA.**



4. **Let M = ({q0, q1,q2,q3}, {0,1}, δ, q0, {q2,q3}) be NFA. Where δ (q0, 0) = {q0, q1}, δ (q0, 1) = {q1}, δ (q1, 0) = {q2,q3}, δ (q1, 1) = {q0, q1}, δ (q2, 0) = {q2}, δ (q2, 1) = {q0, q3}, δ (q3, 0) = {q3}, δ (q3, 1) = {q2, q3}, Construct its equivalent DFA.**

Equivalence of NDFA's with and without ε-moves

Theorem:

If L is accepted by NFA with ε-moves, then there exists L which is accepted by NFA without ε-moves.

Proof:
Let M = (Q, Σ, δ, q_0, F) be an NFA with ε-moves for language L, then define NFA without ε-moves M′ = (Q′, Σ′, δ′, q_0′, F′).

- The elements in Q′ will be denoted by [q_1, q_2, q_3, … , q_i] and the elements in Q are denoted by {q_1, q_2, q_3, … , q_i}.
- Initial state of NFA with ε-moves is q_0, and also an initial state of NFA without ε-moves is q_0′ =[q_0].
- F′ =
- δ′ can be denoted by δ′′ with some input.

Basis:

|X| = 1, where X is a symbol 'a'.
δ′(q0,a) = δ′′(q0,a)

Induction:

|X| > 1, Let X = wa
δ′(q0,wa) = δ′(δ′′(q0,w),a)

By induction hypothesis,

δ′(q0,w) = δ′′(q0,w) = p

Now we will show that

δ′(p,a) = δ(q0,wa)

But,

δ′(p,a) = δ′(q,a) = δ′′(q,a) as p = δ′′(q0,w)

We have

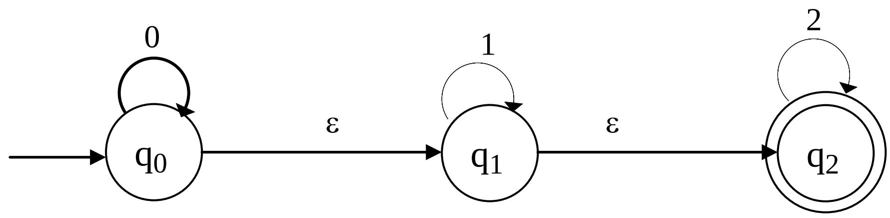
δ′′(q,a) = δ′′(q0,wa)

Thus by definition of δ′′

δ′(q0,wa) = δ′′(q0,wa)

Example Problems for Converting NFA with ε into NFA without ε

1. Construct NFA without ε from NFA with ε.



Solution:

Find the ε – closure function of all states:

ε – closure (q0) = {q0, q1, q2}
ε – closure (q1) = {q1, q2}
ε – closure (q2) = {q2}

ε – closure (q0)
= { q0,q1,q2}

Compute δ′ function:

δ′(q0,0) = δ″ (q0,0) = ε – closure (δ(δ′(q0,ε),0))
= ε – closure (δ({q0,q1,q2},0))
= ε – closure (q0) = **{q0,q1,q2}**

δ′(q0,1) = δ″ (q0,1) = ε – closure (δ(δ′(q0,ε),1))
= ε – closure (δ({q0,q1,q2},1))
= ε – closure (q1) = **{q1,q2}**

δ′(q0,2) = δ″ (q0,2) = ε – closure (δ(δ′(q0,ε),2))
= ε – closure (δ({q0,q1,q2},2))
= ε – closure (q2) = **{q2}**

δ′(q1,0) = δ″ (q1,0) = ε – closure (δ(δ′(q1,ε),0))
= ε – closure (δ({q1,q2},0))
= ε – closure (ϕ) = **{ϕ}**

δ′(q1,1) = δ″ (q1,1) = ε – closure (δ(δ′(q1,ε),1))
= ε – closure (δ({q1,q2},1))
= ε – closure (q1) = **{q1,q2}**

δ′(q1,2) = δ″ (q1,2) = ε – closure (δ(δ′(q1,ε),2))
= ε – closure (δ({q1,q2},2))
= ε – closure (q2) = **{q2}**

δ′(q2,0) = δ″ (q2,0) = ε – closure (δ(δ′(q2,ε),0))
= ε – closure (δ({q2},0))
= ε – closure (ϕ) = **{ϕ}**

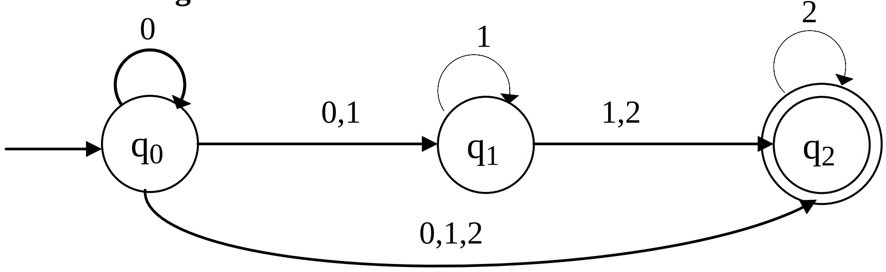
δ′(q2,1) = δ″ (q2,1) = ε – closure (δ(δ′(q2,ε),1))
= ε – closure (δ({q2},1))
= ε – closure (ϕ) = **{ϕ}**

δ′(q2,2) = δ″ (q2,2) = ε – closure (δ(δ′(q2,ε),2))
= ε – closure (δ({q2},2))
= ε – closure (q2) = **{q2}**

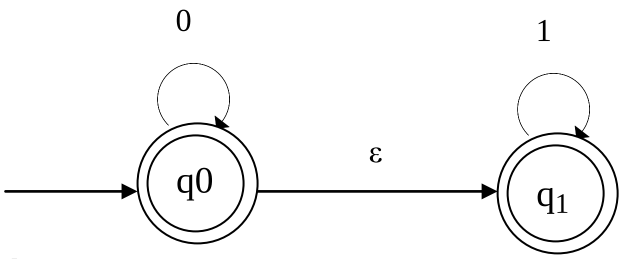
The transition table for NFA

States	Input		
	0	1	2
→q0	{q0,q1,q2}	{q1,q2}	{q2}
q1	{ϕ}	{q1,q2}	{q2}
*q2	{ϕ}	{ϕ}	{q2}

The transition diagram for NFA



2. Construct NFA without ε from NFA with ε.



Solution:

<u>Find the ε – closure function of all states:</u>

ε – closure (q0) = {q0, q1}
ε – closure (q1) = {q1}

<u>Compute δ′ function:</u>

δ′(q0,0) = δ″ (q0,0) = ε – closure (δ(δ′(q0,ε),0))
= ε – closure (δ({q0,q1},0))
= ε – closure (q0) = **{q0,q1}**

δ′(q0,1) = δ″ (q0,1) = ε – closure (δ(δ′(q0,ε),1))
= ε – closure (δ({q0,q1},1))
= ε – closure (q1) = **{q1}**

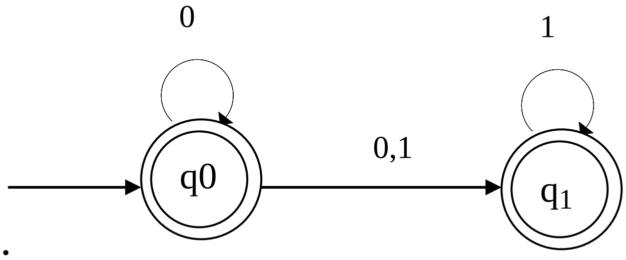
δ′(q1,0) = δ″ (q1,0) = ε – closure (δ(δ′(q1,ε),0))
= ε – closure (δ({q1},0))
= ε – closure (ϕ) = **{ϕ}**

δ′(q1,1) = δ″ (q1,1) = ε – closure (δ(δ′(q1,ε),1))
= ε – closure (δ({q1},1))
= ε – closure (q1) = **{q1}**

The transition table for NFA

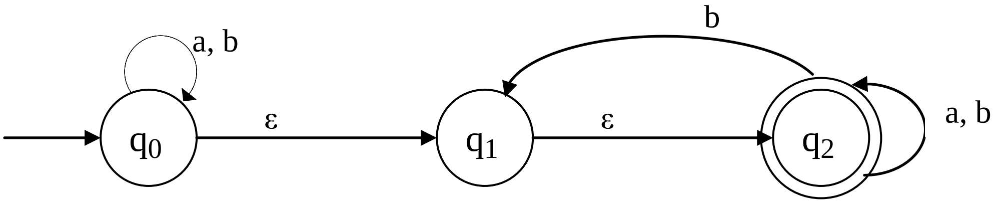
States	Input	
	0	1
→*q0	{q0,q1}	{q1}
*q1	{ϕ}	{q1}

The transition diagram for NFA



Tutorial:

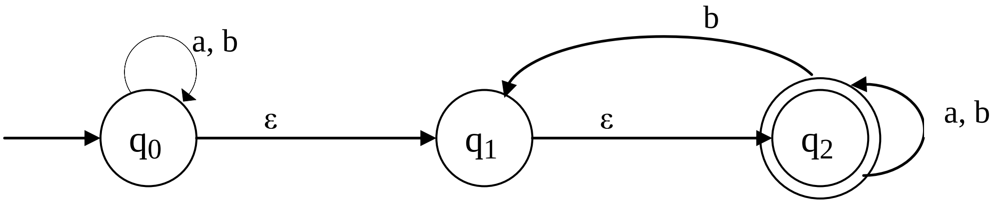
1. **Obtain the NFA equivalent to the following NFA with ε-move.**



2. **Let M = ({q0, q1,q2,q3}, {0,1}, δ, q0, {q2,q3}) be ε-NFA.**
Where δ (q0, 0) = {q0, q1}, δ (q0, 1) = {q1}, δ (q1, 0) = {q2,q3}, δ (q1, ε) = {q1}, δ (q1, 1) = {q0, q1}, δ (q2, 0) = {q2}, δ (q2, ε) = {q3}, δ (q2, 1) = {q0, q3,}, δ (q3, 0) = {q3}, δ (q3, 1) = {q2, q3}, δ (q3, ε) = {q0}. Construct its equivalent NFA.

Example Problems for Converting NFA with ε-move into DFA

1. **Construct DFA from the following ε-NFA.**



Solution:

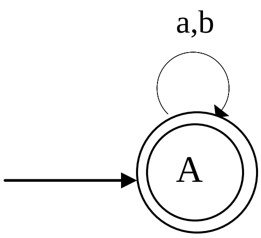
ε – closure (q0) = {q0, q1, q2} → A new state in DFA

ε – closure (δ (A, a)) = ε – closure (q0,q2)
= {q0, q1, q2} → A
ε – closure (δ (A, b)) = ε – closure (q0,q1,q2)
= {q0, q1, q2} → A

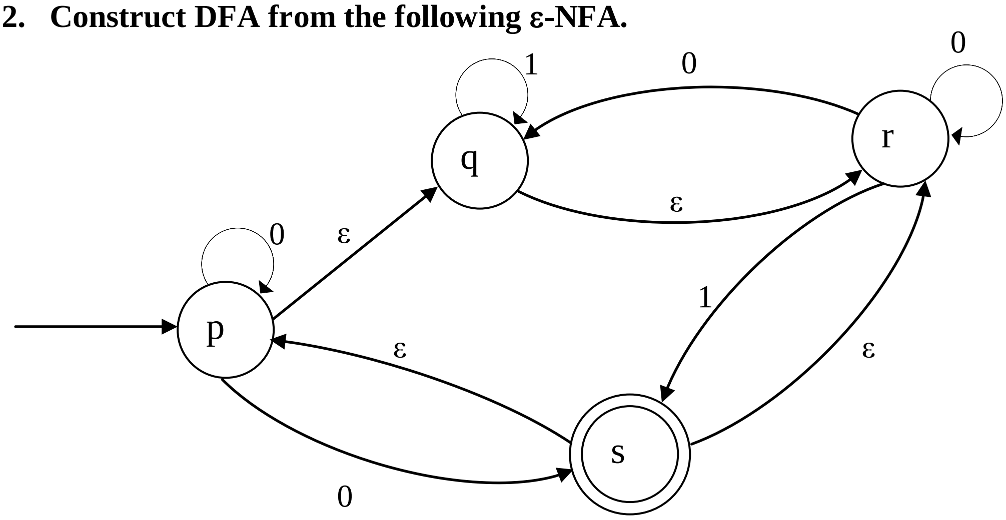
The transition table for DFA

States	Input	
	A	b
→*A	A	A

The transition diagram for DFA



2. Construct DFA from the following ε-NFA.



Solution:

ε – closure (p) = {p,q,r} → A new state in DFA

ε – closure (δ (A, 0)) = ε – closure (p,r) = ε – closure (p) ∪ ε – closure (r)
= {p,q,r} ∪ {r,s} = {p,q,r,s} → **B new state in DFA**

ε – closure (δ (A, 1)) = ε – closure (q,s) = ε – closure (q) ∪ ε – closure (s)
= {q,r} ∪ {p,q,r,s} = {p,q,r,s} → **B**

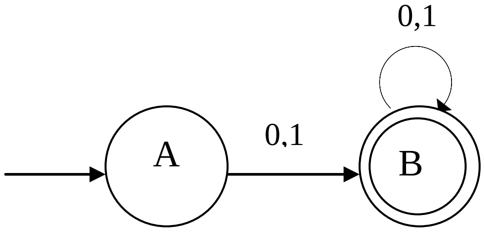
ε – closure (δ (B, 0)) = ε – closure (p,r) = ε – closure (p) ∪ ε – closure (r)
= {p,q,r} ∪ {r,s} = {p,q,r,s} → **B**

ε – closure (δ (B, 1)) = ε – closure (q,s) = ε – closure (q) ∪ ε – closure (s)
= {q,r} ∪ {p,q,r,s} = {p,q,r,s} → **B**

The transition table for DFA

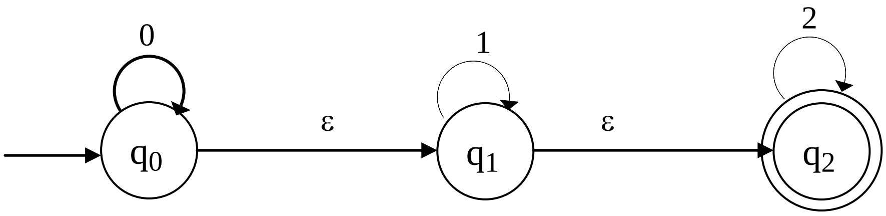
States	Input	
	0	1
→A	B	B
*B	B	B

The transition diagram for DFA



Tutorial:

1. **Obtain the DFA equivalent to the following NFA with ε-move.**



2. **Let M = ({q0, q1,q2,q3}, {0,1}, δ, q0, {q2,q3}) be ε-NFA.
Where δ (q0, 0) = {q0, q1}, δ (q0, 1) = {q1}, δ (q1, 0) = {q2,q3}, δ (q1, ε) = {q1}, δ (q1, 1) = {q0, q1}, δ (q2, 0) = {q2}, δ (q2, ε) = {q3}, δ (q2, 1) = {q0, q3,}, δ (q3, 0) = {q3}, δ (q3, 1) = {q2, q3}, δ (q3, ε) = {q0}. Construct its equivalent DFA.**

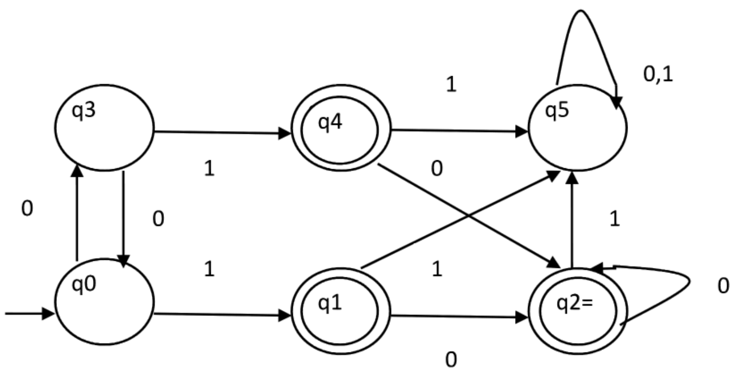
Minimization of DFA

- ✓ DFA minimization stands for converting a given DFA to its equivalent DFA with minimum number of states.
- ✓ Suppose there is a DFA M = (Q, ∑, q0, δ, F) which recognizes a language L. Then the minimized DFA M = (Q', ∑, q0, δ', F') can be constructed for language L as:

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.
2. Initialize k = 1
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.
4. Stop when $P_k = P_{k-1}$ (No change in partition)
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.

Example:

Consider the following DFA into minimized DFA.



Solution:

Transition Table for DFA

States	**Inputs**	
	0	**1**
→q0	q3	q1
*q1	q2	q5
*q2	q2	q5
q3	q0	q4
*q4	q2	q5
q5	q5	q5

Step 1: Divide into two sets. One set is containing final states and other set containing non-final states.

States	**Inputs**		**Partition (P_0)**
	0	**1**	
→q0	q3	q1	Non-Final States
q3	q0	q4	
q5	q5	q5	
*q1	q2	q5	Final States
*q2	q2	q5	
*q4	q2	q5	

Step 2: To calculate P_1, we will check whether sets of partition P_0 can be partitioned or not:

For set { q1, q2, q4 } :

- δ (q1, 0) = δ (q2, 0) = q2 and δ (q1, 1) = δ (q2, 1) = q5, So q1 and q2 are not distinguishable.
- Similarly, δ (q1, 0) = δ (q4, 0) = q2 and δ (q1, 1) = δ (q4, 1) = q5, So q1 and q4 are not distinguishable.
- So, q2 and q4 are not distinguishable. So, {q1, q2, q4} set will not be partitioned in P1.

States	**Inputs**		**Partition (P_0)**
	0	**1**	
→q0	q3	q1	Non-Final States
q3	q0	q4	
q5	q5	q5	
***q1**	**q2**	**q5**	Final States
***q2**	**q2**	**q5**	
***q4**	**q2**	**q5**	

Step 3: Remove q2 and q4 row from the table and replace q2 and q4 into q1 where however present in the table.

States	Inputs		Partition (P_0)
	0	1	
→q0	q3	q1	Non-Final States
q3	q0	~~q4~~ q1	
q5	q5	q5	
*q1	q1	q5	Final States

Step 4:

- δ (q0, 0) = q3 and δ (q3, 0) = q0 - Moves of q0 and q3 on input symbol 0 are q3 and q0 respectively which are in same set in partition P0.
- δ (q0, 1) = δ (q3, 1) = q1 - Moves of q0 and q3 on input symbol 1 is q1 which are in same set in partition P0.
- So, q0 and q3 are not distinguishable.

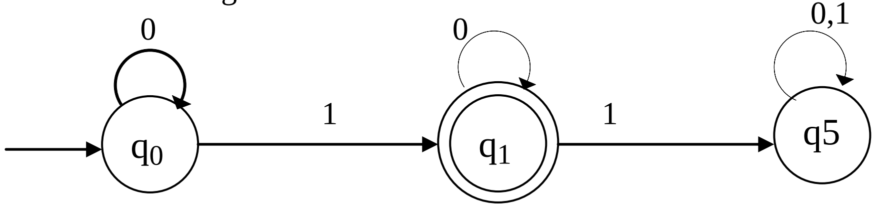
Step 5: Remove q3 row from the table and replace q3 into q0 where however present in the table.

States	Inputs		Partition (P_0)
	0	1	
→q0	~~q3~~ q0	q1	Non-Final States
q3	q0	q1	
q5	q5	q5	
*q1	q1	q5	Final States

Step 6: Final Transition Table for DFA (no more not distinguishable)

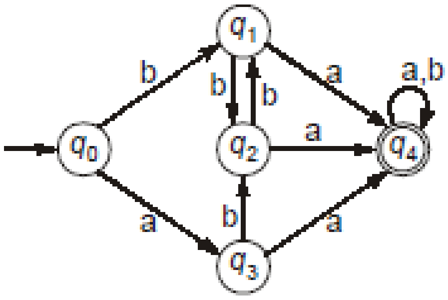
States	Inputs		Partition (P_0)
	0	1	
→q0	q0	q1	Non-Final States
q5	q5	q5	
*q1	q1	q5	Final States

Step 7: Transition Diagram for minimized DFA

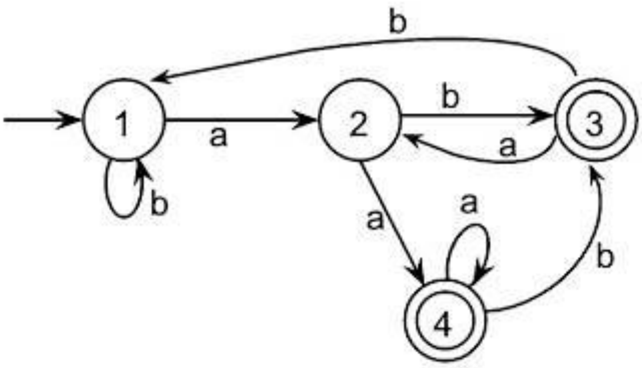


Tutorial:

1. Consider the following DFA into minimized DFA.

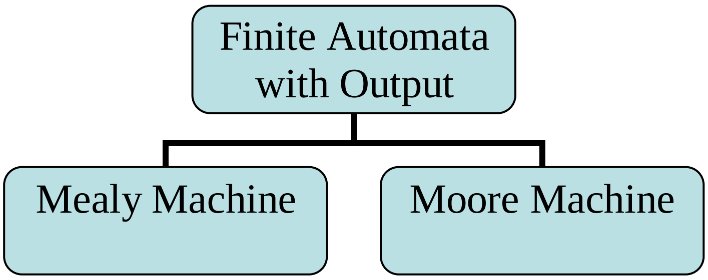


2. Consider the following DFA into minimized DFA.



Finite automata with Output

- ✓ Finite automata may have outputs corresponding to each transition. There are two model or machine for finite automata with output.



Mealy Machine

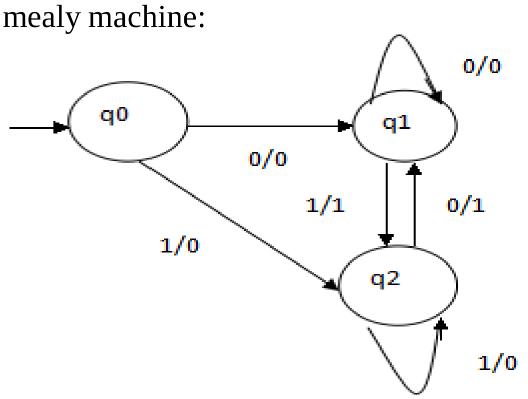
- ✓ A Mealy Machine is an FSM whose output depends on the present state as well as the present input.
- ✓ The value of the output function z(t) depends only on the present state q(t) and present input λ (t), i.e. z(t) = λ (q(t), x(t))
- ✓ The length of output for a mealy machine is equal to the length of input. If input string ε, the output string is also ε.

✓ It can be described by a 6 tuples M = (Q, ∑, ∆, δ, λ, q0)
where
- **Q** is a finite set of states.
- ∑ is a finite set of input symbols
- ∆ is a finite set of output symbols
- **δ** is the input transition function where δ: Q × ∑ → Q
- λ is the output transition function where λ : Q × ∑ → ∆
- q0 is the initial state

✓ Transition table of mealy machine:

Present State	**Input = 0**		**Input = 1**	
	Next State	**Output**	**Next State**	**Output**
→q0	q1	0	q2	0
q1	q1	0	q2	1
q2	q1	1	q2	0

✓ Transition diagram of mealy machine:



Moore Machine

✓ Moore machines are FSM whose output depends on the present state as well as the previous state.

✓ The value of the output function z(t) depends only on the present state q(t) and independent of the current input x(t), i.e. z(t) = λ (q(t))

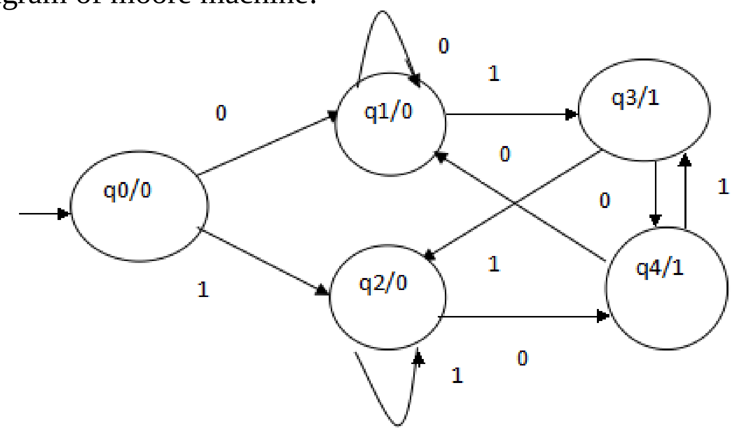
✓ The length of output for a moore machine is greater than input by 1. If input string ε, the output string is ∆= λ (q(t)).

✓ It can be described by a 6 tuples M = (Q, ∑, ∆, δ, λ, q0)
where
- **Q** is a finite set of states.
- ∑ is a finite set of input symbols
- ∆ is a finite set of output symbols
- **δ** is the input transition function where δ: Q × ∑ → Q
- λ is the output transition function where λ : Q → ∆
- q0 is the initial state

✓ Transition table of moore machine:

Present State	Next State		Output
	Input = 0	**Input = 1**	
→q0	q1	q2	0
q1	q1	q3	0
q2	q4	q2	0
q3	q4	q2	1
q4	q1	q3	1

✓ Transition diagram of moore machine:



Mealy Machine vs. Moore Machine

Mealy Machine	Moore Machine
Output depends both upon the present state and the present input	Output depends only upon the present state.
Generally, it has fewer states than Moore Machine.	Generally, it has more states than Mealy Machine.
The value of the output function is a function of the transitions and the changes, when the input logic on the present state is done.	The value of the output function is a function of the current state and the changes at the clock edges, whenever state changes occur.
Mealy machines react faster to inputs. They generally react in the same clock cycle.	In Moore machines, more logic is required to decode the outputs resulting in more circuit delays. They generally react one clock cycle later.

Transforming Mealy Machine into Moore Machine

- ✓ Transform Mealy Machine into Moore Machine for the given input string and the output string as same (except for the first symbol).
- ✓ **Algorithm:**
  - **Step 1:** Look into the next state column for any state (example q0,q1, …. qi) and determine the number of different outputs associated with qi in that column (output column values are same or different).
  - **Step 2:** qi into several different states. The number of such states being equal to the number of outputs associated with qi.
  - **Step 3:** qi replaced by qi0 for output 0 and qi1 for output 1
  - **Step 4:** Convert Mealy Structure to Moore Structure
  - **Step 5:** Add new start state with output 0 and next states same as the next states of first state.

- ✓ **Example:**

  Consider the Mealy machine described by the transition table given below. To construct a Moore machine, this is equivalent to mealy machine.

Present State	a = 0		a = 1	
	Next State	Output	Next State	Output
→q1	q3	0	q2	0
q2	q1	1	q4	0
q3	q2	1	q1	1
q4	q4	1	q3	0

Solution:

Step 1: Look into the next state column for any state (example q0,q1, …. qi) and determine the number of different outputs associated with qi in that column (output column values are same or different).

Present State	a = 0		a = 1		Determine same or different output
	Next State	Output	Next State	Output	
→q1	q3	0	q2	0	same
q2	q1	1	q4	0	different
q3	q2	1	q1	1	same
q4	q4	1	q3	0	different

Step 2: q2 split into q20 and q21 states. Similarly q4 split into q40 and q41.

Present State	a = 0		a = 1	
	Next State	Output	Next State	Output
→q1	q3	0	q2	0
q2 < q20, q21	q1	1	q4	0
q3	q2	1	q1	1
q4 < q40, q41	q4	1	q3	0

Present State	a = 0		a = 1	
	Next State	Output	Next State	Output
→q1	q3	0	q2	0
q20	q1	1	q4	0
q21	q1	1	q4	0
q3	q2	1	q1	1
q40	q4	1	q3	0
q41	q4	1	q3	0

Step 3: q2 replaced by q20 for output 0 and q21 for output 1, similarly q4 replaced by q40 for output 0 and q41 for output 1

Present State	a = 0		a = 1	
	Next State	Output	Next State	Output
→q1	q3	0	q20	0
q20	q1	1	q40	0
q21	q1	1	q40	0
q3	q21	1	q1	1
q40	q41	1	q3	0
q41	q41	1	q3	0

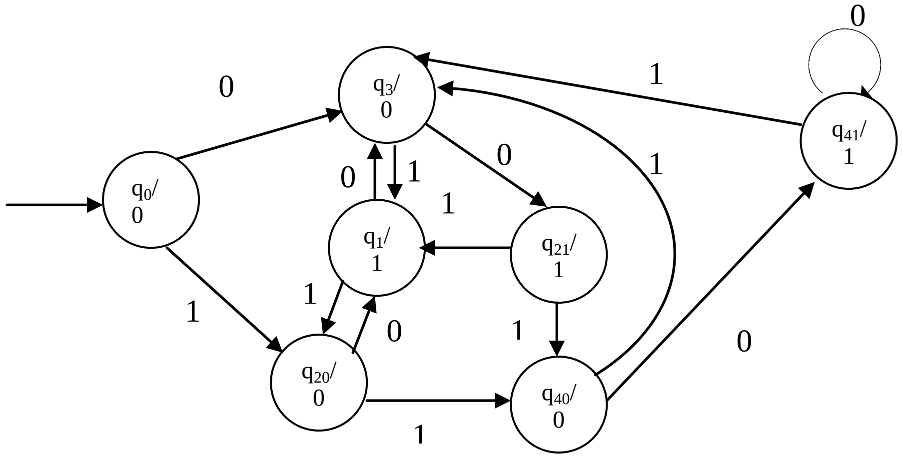
Step 4: Convert Mealy Structure to Moore Structure

Present State	Next State		Output
	a = 0	a = 1	
q1	q3	q20	1
q20	q1	q40	0
q21	q1	q40	1
q3	q21	q1	0
q40	q41	q3	0
q41	q41	q3	1

Step 5: Add new start state with output 0 and next states same as the next states of first state.

Present State	Next State		Output
	a = 0	a = 1	
→q0	q3	q20	0
q1	q3	q20	1
q20	q1	q40	0
q21	q1	q40	1
q3	q21	q1	0
q40	q41	q3	0
q41	q41	q3	1

Transition Diagram for Moore Machine

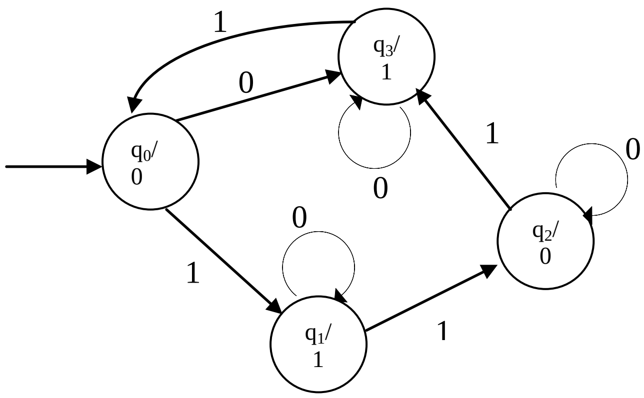


Transforming Moore Machine into Mealy Machine

- ✓ Transform Mealy Machine into Moore Machine for the given input string and the output string as same.
- ✓ **Algorithm:**
  - **Step 1:** Remove output column from moore table and add output column to mealy table
  - **Step 2:** Fill the output column from moore table.

Example:

Consider the Moore machine described by the transition diagram given below. To construct a Mealy machine, which is equivalent to moore machine.



Transition Table for Moore Machine

Present State	Next State		Output
	a = 0	a = 1	
→q0	q3	q1	0
q1	q1	q2	1
q2	q2	q3	0
q3	q3	q0	1

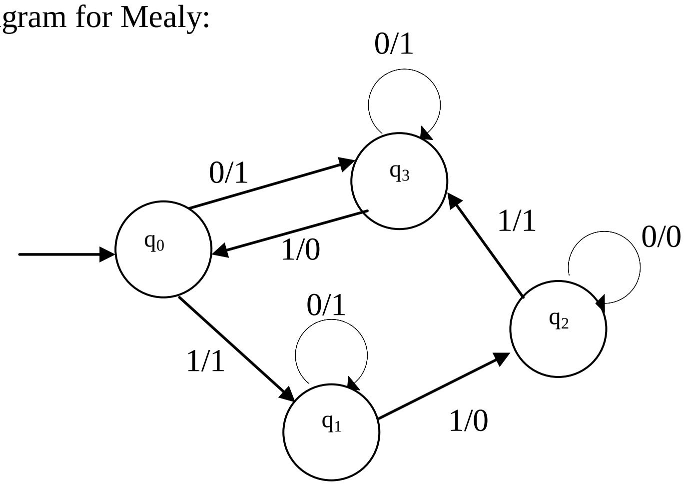
Solution:

Step 1: Remove output column from moore table and add output column to mealy table

Transition Table for Mealy:

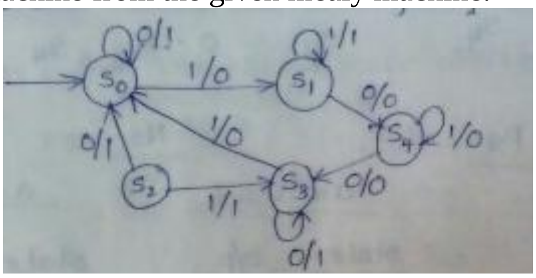
Present State	a = 0		a = 1	
	Next State	Output	Next State	Output
→q0	q3	1	q1	1
q1	q1	1	q2	0
q2	q2	0	q3	1
q3	q3	1	q0	0

Transition Diagram for Mealy:

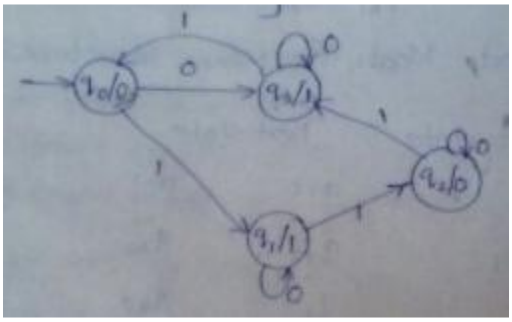


Tutorial Problems:

1. Construct the moore machine from the given mealy machine.



2. Construct the moore machine from the given mealy machine.



Syllabus : Unit – II : Regular Expressions and Regular sets

Regular expressions – Regular languages - Identity rules for regular expressions – Equivalence of finite automata and regular expressions – Pumping lemma for regular sets – Applications of the Pumping lemma - Closure proportions of regular sets (Without proof)

Equivalence of finite Automaton and regular expressions

Regular Languages

A language is called regular language if there exists a finite automaton that recognizes it. For example finite automaton M recognizes the language L if L = {w | M accepts w}.

✓ Operations on Regular Languages

Let A and B be languages. We define regular operations union, concatenation, and star as follows:

- Union : $A \cup B = \{x \mid x \in A \vee x \in B\}$
- Concatenation : $A \circ B = \{xy \mid x \in A \wedge y \in B\}$
- Star : $A^* = \{x_1x_2 \ldots x_k \mid k \geq 0 \wedge x_i \in A, 1 \leq i \leq k\}$

Regular Expression

Let Σ be an alphabet. The regular expressions over Σ and the sets that they denote are defined recursively as follows:

a. Ø is a regular expression and denotes the empty set.
b. ε is a regular expression and denotes the set {ε}
c. For each 'a'∈ Σ, 'a' is a regular expression and denotes the set {a}.
d. If 'r' and 's' are regular expressions denoting the languages L_1 and L_2 respectively then
   - ✓ Union : r + s is equivalent to $L_1 \cup L_2$
   - ✓ Concatenation : rs is equivalent to L_1L_2
   - ✓ Closure : r^* is equivalent to L_1^*

Problems for Regular Expression

1. Write the regular expression for the language accepting all combinations of a's over the set ∑ = {a}.

L = { a,aa,aaa,………………….}

R= a^* (i.e. kleen closure)

2. Write regular expression for the language accepting the strings which are starting with 1 and ending with 0, over the set ∑ = {0,1}.

L = { 10,1100,1010,100010………………….}

R= $1(0+1)^*0$

3. Show that (0*1*)* = (0+1)*.

LHS : (0*1*)* = { ε, 0,1,00,11,0011,011,0011110………………}
RHS : (0+1)* = { ε, 0,1,00,11,0011,011,0011110………………}
Hence
LHS = RHS is proved

4. Show that (r+s)* ≠ r* + s*.

LHS : (r+s)* = { ε, r,s,rs,rr,ss,rrrsssr,……………….}
RHS : r* + s* = { ε, r,rr,rrr…………}U { ε, s,ss,sss,…………}
= { ε, r,rr,rrr,s,ss,ssss……………..}
Hence
LHS ≠ RHS is proved

5. Describe the following by regular expression

a. L1 = the set of all strings of 0's and 1's ending in 00.

b. L2 = the set of all strings of 0's and 1's beginning with 0 and ending with .

r1 = (0+1)*00
r2 = 0(0+1)*1

6. Show that (r*)* = r* for a regular expression r.

LHS = r* = { ε, r,rr,rrr, …………….)
(r*)* = { ε, r,rr,rrr, …………….)*
(r*)* = { ε, r,rr,rrr, …………….) = r*
LHS = RHS

7. If L = {The language starting and ending with 'a' and having any combinations of b's in between, that what is r?

r1 = a b*a

**8. Give regular expression for L= L1 ∩ L2 over alphabet {a,b}
where L1 = all strings of even length,
L2 = all strings starting with 'b'.**

r = r1 + r2
r = $a^n b^n$ + b (a+b)*

Regular Expressions & Languages

Syllabus:

Regular Expression - FA and Regular Expressions - proving languages not to be regular - closure properties of regular languages - Equivalence and minimation of Automata.

Regular Expressions:

- The languages accepted by FA are easily descripted by simple expressions called regular expressions.

- The Regular Expression is very effective way to represent any language.

Definition:

Let Σ be an alphabet which is used to denote the input set.

The regular expression over Σ can be defined as follows,

- φ is a regular expression which denotes the empty set.
- ε is a regular expression and denotes the set {ε}
- for each 'a' in Σ, 'a' is a regular expression and denotes the set {a}.
- If r and s are regular expressions denoting the languages L_1 and L_2 respectively, then
  - * r+s is equivalent to $L_1 \cup L_2$. ie. union
  - * rs is " to $L_1 L_2$. ie concatenation
  - * r^* is " to L_1^*. ie closure
  - * r^+ is " to L_1^+. ie positive closure.

Problem :

1. Write the regular expression for the language accepting all combinations of a's over the set $\Sigma = \{a\}$

$L = \{\varepsilon, a, aa, aaa, \ldots\}$

$R = a^*$

2. Design the r.e for the language accepting all combinations of 'a' except the null string over $\Sigma = \{a\}$.

$L = \{a, aa, aaa, \ldots\}$

$R = a^+$

3. Design the r.e for the language containing all the string containing any number of a's and b's.

$L = \{\varepsilon, a, b, ab, aabb, abab, aabbb, \ldots\}$

$r.e = (a+b)^*$

4. Design the r.e for the language containing all the string containing any number of a's and b's. except the null string.

$L = \{a, b, ab, aabb, abab, aabbb, \ldots\}$

$r.e = (a+b)^+$

5. Construct the r.e for the language accepting all the strings which are ending with 00 over the set $\Sigma = \{0,1\}$.

$L = \{00, 0000, 1100, 01000, \ldots\}$

$r.e = (0+1)^*00$

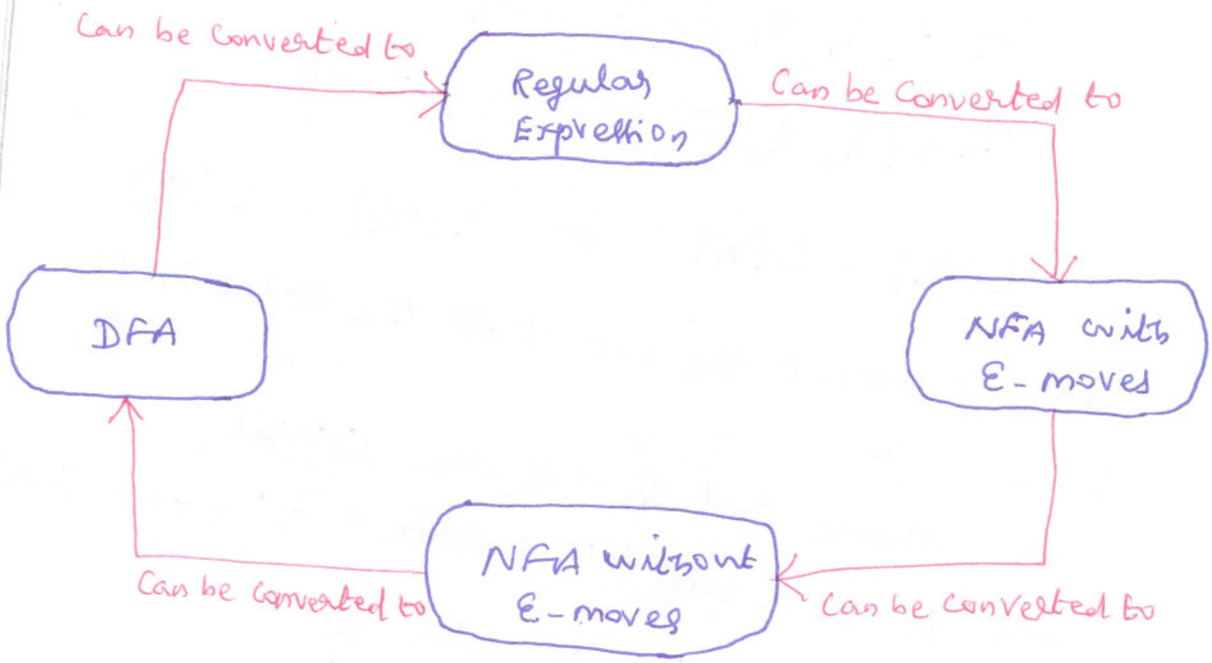
6. Write r.e to denote a language L which accepts all the strings which begin or end with either 00 or 11.

$L = \{00100, 11011, 101000, 010111 \ldots\}$

$L = (0+1)^*(00+11)$.

Equivalence of FA and R.E.

- There is a close relationship between a FA & R.E.
- Can show this relation by the following figure.



Theorem :-

Let r be a r.e. Then there exists an NFA with ε-transitions that accepts $L(r)$.

Proof:

Basis (zero operators):

- Now, since r has zero operators ϕ, ε or a for some 'a' in Σ.

(i) $r = \varepsilon$ (ii) $r = \phi$ (iii) $r = a$

Induction (one or more operators):

- This theorem can be true for 'n' no. of operators
- $n \geq 1$
- r.e contains equal to or more than one operator.
- In any type of r.e there are only three cases possible
  (i) Union (ii) Concatenation (iii) Closure

Case 1 : (Union)

Let $r = r_1 + r_2$ where r_1 and r_2 be the r.e.

There exists two NFA's

$M_1 = (Q_1, \Sigma_1, \delta_1, q_1, \{f_1\})$ &

$M_2 = (Q_2, \Sigma_2, \delta_2, q_2, \{f_2\})$ with

$L(M_1) = L(r_1)$ & $L(M_2) = L(r_2)$

→ Q_1 - represent the set of all the states in M_1

→ Q_2 - " " " in M_2

we assume that Q_1 & Q_2 are disjoint.

Let 'q_0' be a new initial state & 'f_0' a new final state.

Construction of Machine 'M' will be

$M = (Q_1 \cup Q_2 \cup \{q_0, f_0\}, \Sigma_1 \cup \Sigma_2, \delta, q_0, \{f_0\})$

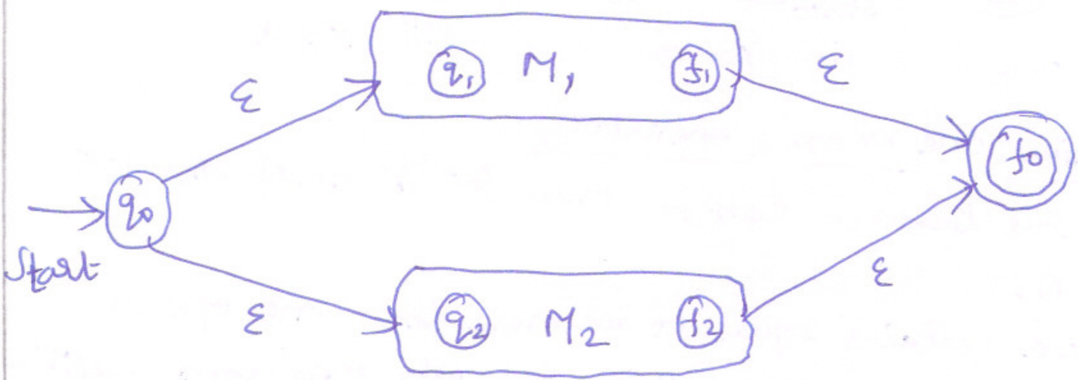
where δ is defined by

(i) $\delta(q_0, \varepsilon) = \{q_1, q_2\}$

(ii) $\delta(q, a) = \delta_1(q, a)$ for q in $Q_1 - \{f_1\}$ and a in $\Sigma_1 \cup \{\epsilon\}$

(iii) $\delta(q, a) = \delta_2(q, a)$ for q in $Q_2 - \{f_2\}$ and a in $\Sigma_2 \cup \{\varepsilon\}$

(iv) $\delta(f_1, \varepsilon) = \delta(f_2, \varepsilon) = \{f_0\}$.



- The construction of machine M is shown the transition from 'q_0' to 'f_0' must begin by going to q_1 or q_2 on ε.
- If the path goes to q_1 then it follows the path in M_1 and goes to the state f_1 and then goto f_0 on ε.
- Similarly if the path goes to q_2 to $f_2 \rightarrow f_0$ through M_2

Thus $L(M) = L(M_1) \cup L(M_2)$ is proved.

Case : 2 (concatenation)

- Let $r = r_1 r_2$ where r_1 & r_2 are two r.e.
- The M_1 & M_2 denotes the two machines

Showthat $L(M_1) = L(r_1)$ & $L(M_2) = L(r_2)$

- construction of Machine M will be

$M = (Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, \delta, \{q_1\}, \{f_2\})$

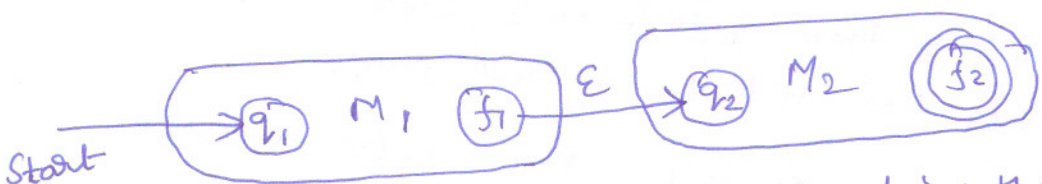
where δ is defined by

(i) $\delta(q, a) = \delta_1(q_1, a)$ for q in $Q_1 - \{f_1\}$ and 'a' in $\Sigma_1 \cup \{\varepsilon\}$.

(ii) $\delta(f_1, \varepsilon) = \{q_2\}$

(iii) $\delta(q, a) = \delta_2(q, a)$ for q in Q_2 & a in $\Sigma_2 \cup \{\varepsilon\}$

- The machine M is shown in the figure.



- The initial state is q_1 by some input 'a' the next state will be f_1. And on receiving ε the transition will be from f_1 to q_2 & final state will be f_2.

(b)

- The transition from q_2 to f_2 will be on receiving some input b.

Thus $L(M) = xy$
i.e. x is in $L(M_1)$ & y is in $L(M_2)$

Hence

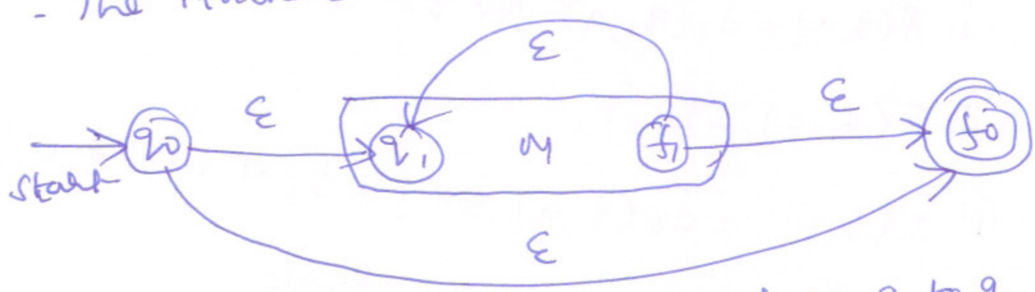
$$\boxed{L(M) = L(M_1) \cdot L(M_2)}$$ is proved.

Case : 3 (closure)

- Let $r = r_1^*$ where r_1 be a r.e.
- The Machine M_1 is show that $L(M_1) = L(r_1)$
- construction of Machine M will be

$M = (Q_1 \cup \{q_0, f_0\}, \Sigma_1, \delta, q_0, \{f_0\})$
where δ is defined by
(i) $\delta(q_0, \varepsilon) = \delta(f_1, \varepsilon) = \{q_1, f_0\}$
(ii) $\delta(q, a) = \delta_1(q, a)$ for q in $Q_1 - \{f_1\}$ and a in $\Sigma_1 \cup \{\varepsilon\}$.

- The Machine M will be



- The Machine M_1 shows that from q_0 to q_1 on ε.
- Similarly from q_0 to f_0 on ε there is a path. The path exists from f_1 to q_1 a back path.
- Similarly a transition from f_1 to f_0 on ε.

Thus $$\boxed{L(M) = L(M_1)^*}$$ is proved.

Problems:

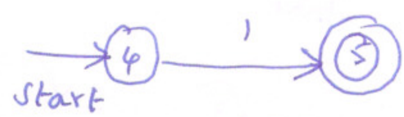
① Convert r.e into NFA with ε-move for the following

$$r = (0+1)^* 11$$

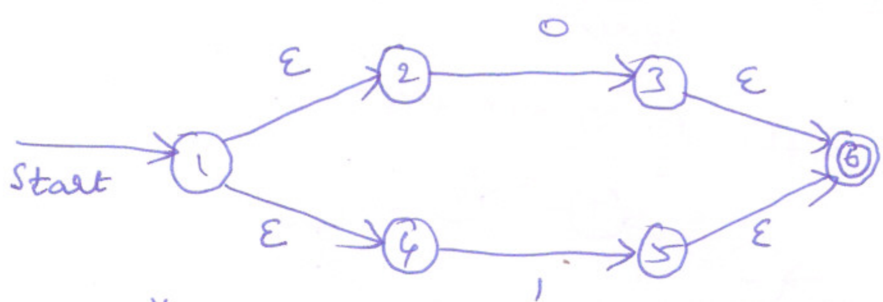
Solution:

$r_1 = 0$

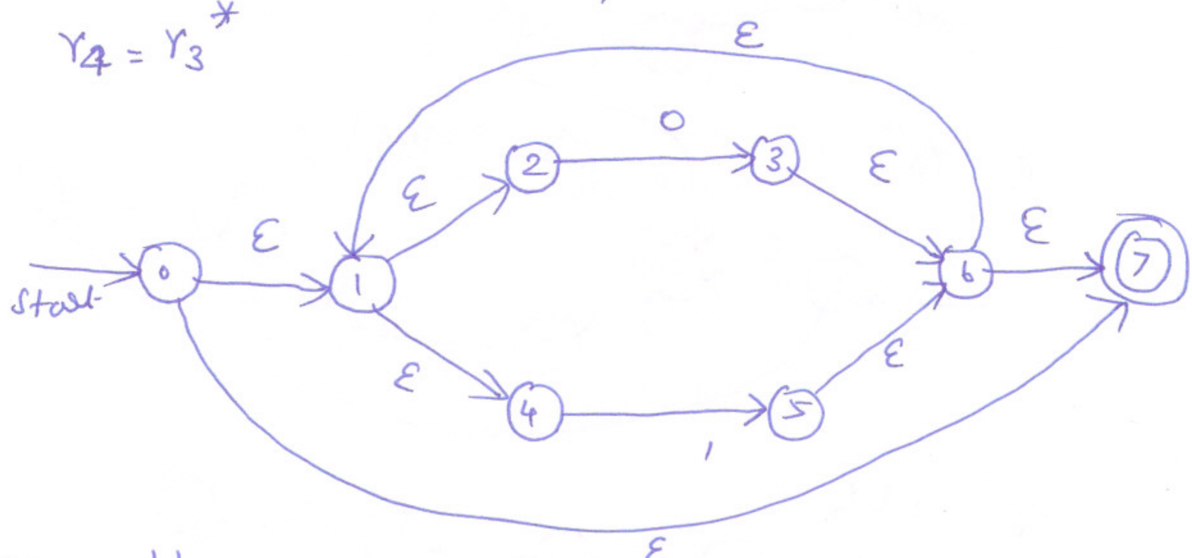
$r_2 = 1$



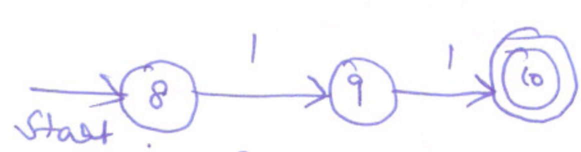
$r_3 = r_1 + r_2$



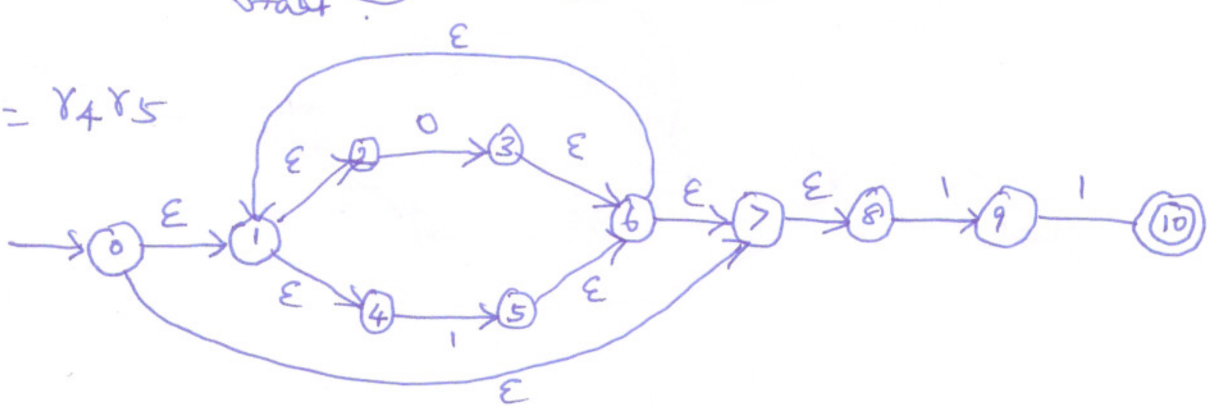
$r_4 = r_3^*$



$r_5 = 11$



$r = r_4 r_5$

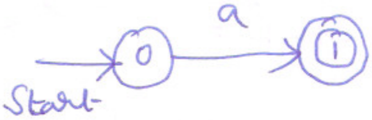


(2) Convert r.e into NFA with ε-move, NFA without ε-move, DFA and minimization of DFA.

$r = a(a+b)^* abb$

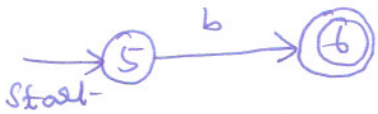
Solution:

$r_1 = a$

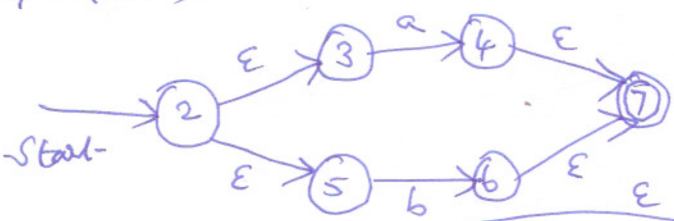


$r_2 = a$

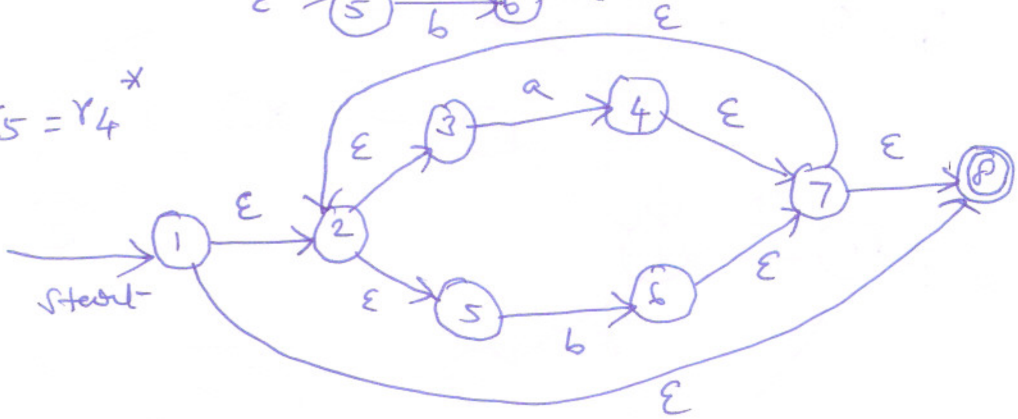
$r_3 = b$



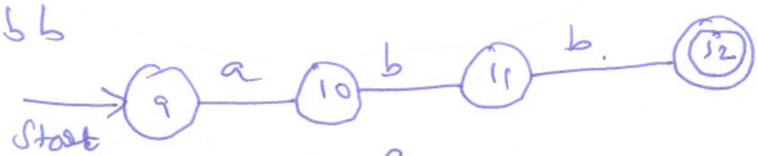
$r_4 = (a+b)$ i.e. $r_4 = (r_2 + r_3)$



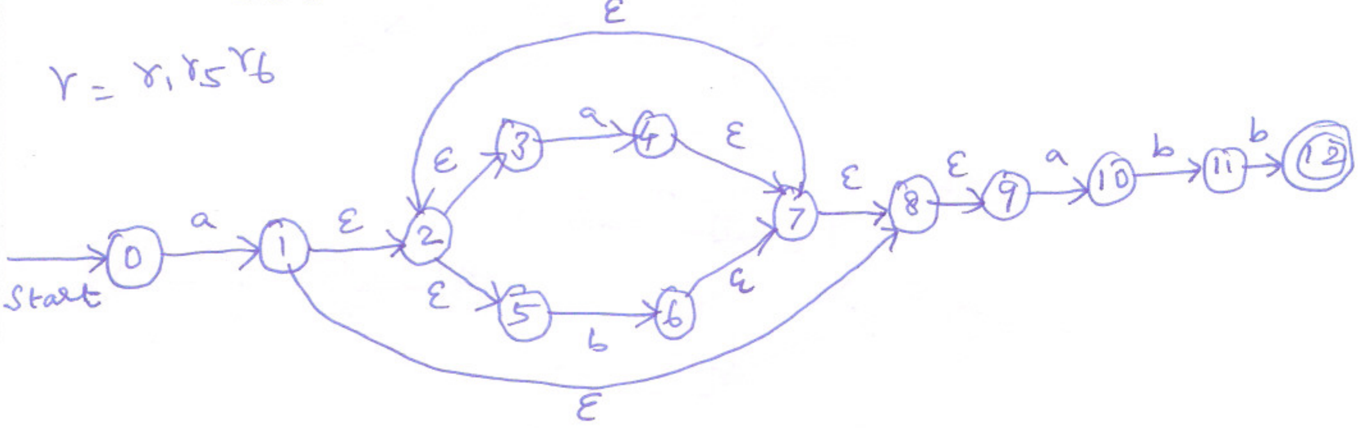
$r_5 = r_4^*$



$r_6 = abb$



$r = r_1 r_5 r_6$



NFA with ε-move to NFA without ε-move :

From Fig (1)

$\varepsilon\text{-closure}(0) = \{0\}$

$\delta'(0,a) = \delta''(0,a)$

$= \varepsilon\text{-closure}(\delta(\delta''(0,\varepsilon)), a)$ → ε-closure (0)

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

$= \varepsilon\text{-closure}(1)$

$= \{1, 2, 3, 5, 8, 9\}$

$\delta'(0,b) = \delta''(0,b)$

$= \varepsilon\text{-closure}(\delta(\delta''(0,\varepsilon)), b)$

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

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

$= \{\phi\}$

$\delta'(1,a) = \delta''(1,a)$

$= \varepsilon\text{-closure}(\delta(\delta''(1,\varepsilon)), a)$

$= \varepsilon\text{-closure}(\delta(\{1,2,3,5,8,9\}), a)$

$= \varepsilon\text{-closure}(4,10)$

$= \{4,7,8,9,2,3,5\} = \{2,3,4,5,7,8,9,10\}$

$\delta'(1,b) = \delta''(1,b)$

$= \varepsilon\text{-closure}(\delta(\delta''(1,\varepsilon)), b)$

$= \varepsilon\text{-closure}(\delta(1,2,3,5,8,9), b)$

$= \varepsilon\text{-closure}(6)$

$= \{6,7,8,9,2,3,5\} = \{2,3,5,6,7,8,9\}$

$\delta'(2,a) = \delta''(2,a)$

$= \varepsilon\text{-closure}(\delta(\delta''(2,\varepsilon)), a)$

$= \varepsilon\text{-closure}(\delta(2,3,5), a)$

$= \varepsilon\text{-closure}(4) = \{2,3,4,5,7,8,9\}$

$$\delta'(2,b) = \delta''(2,b)$$
$$= \varepsilon\text{-closure}(\delta(\delta''(2,\varepsilon)), b)$$
$$= \varepsilon\text{-closure}(\delta(2,3,5), b)$$
$$= \varepsilon\text{-closure}(6)$$
$$= \{2,3,5,6,7,8,9\}$$

$$\delta'(3,a) = \delta''(3,a)$$
$$= \varepsilon\text{-closure}(\delta(\delta''(3,\varepsilon)), a)$$
$$= \varepsilon\text{-closure}(\delta(3,a))$$
$$= \varepsilon\text{-closure}(4) = \{2,3,4,5,7,8,9\}$$

$$\delta'(3,b) = \delta''(3,b)$$
$$= \varepsilon\text{-closure}(\delta(\delta''(3,\varepsilon)), b)$$
$$= \varepsilon\text{-closure}(\delta(3,b))$$
$$= \varepsilon\text{-closure}(\phi) = \{\phi\}$$

$$\delta'(4,a) = \delta''(4,a)$$
$$= \varepsilon\text{-closure}(\delta(\delta''(4,\varepsilon)), a)$$
$$= \varepsilon\text{-closure}(\delta(2,3,4,5,7,8,9); a)$$
$$= \varepsilon\text{-closure}(4,10)$$
$$= \{2,3,4,5,7,8,9,10\}$$

$$\delta'(4,b) = \delta''(4,b)$$
$$= \varepsilon\text{-closure}(\delta(\delta''(4,\varepsilon)), b)$$
$$= \varepsilon\text{-closure}(\delta(2,3,4,5,7,8,9), b)$$
$$= \varepsilon\text{-closure}(6)$$
$$= \{2,3,5,6,7,8,9\}$$

$$\delta'(5,a) = \delta''(5,a)$$
$$= \varepsilon\text{-closure}(\delta(\delta''(5,\varepsilon)), a)$$
$$= \varepsilon\text{-closure}(\delta(5,a))$$
$$= \varepsilon\text{-closure}(\phi) = \{\phi\}$$

$\delta'(5,b) = \delta''(5,b)$
$= \epsilon\text{-closure}(\delta(\delta''(5,\epsilon)),b)$
$= \epsilon\text{-closure}(\delta(5,b))$
$= \epsilon\text{-closure}(6) = \{2,3,5,6,7,8,9\}$

$\delta'(6,a) = \delta''(6,a)$
$= \epsilon\text{-closure}(\delta(\delta''(6,\epsilon)),a)$
$= \epsilon\text{-closure}(\delta(2,3,5,6,7,8,9),a)$
$= \epsilon\text{-closure}(4,10) = \{2,3,4,5,7,8,9,10\}$

$\delta'(6,b) = \delta''(6,b)$
$= \epsilon\text{-closure}(\delta(\delta''(6,\epsilon)),b)$
$= \epsilon\text{-closure}(\delta(2,3,5,6,7,8,9),b)$
$= \epsilon\text{-closure}(6) = \{2,3,5,6,7,8,9\}$

$\delta'(7,a) = \delta''(7,a)$
$= \epsilon\text{-closure}(\delta(\delta''(7,\epsilon)),a)$
$= \epsilon\text{-closure}(\delta(2,3,5,7,8,9),a)$
$= \epsilon\text{-closure}(4,10) = \{2,3,4,5,7,8,9,10\}$

$\delta'(7,b) = \delta''(7,b)$
$= \epsilon\text{-closure}(\delta(\delta''(7,\epsilon)),b)$
$= \epsilon\text{-closure}(\delta(2,3,5,7,8,9),b)$
$= \epsilon\text{-closure}(6) = \{2,3,5,6,7,8,9\}$

$\delta'(8,a) = \delta''(8,a)$
$= \epsilon\text{-closure}(\delta(\delta''(8,\epsilon)),a)$
$= \epsilon\text{-closure}(\delta(8,9),a)$
$= \epsilon\text{-closure}(10) = \{10\}$

$\delta'(8,b) = \delta''(8,b)$
$= \epsilon\text{-closure}(\delta(\delta''(8,\epsilon)),b)$
$= \epsilon\text{-closure}(\delta(8,9),b)$
$= \epsilon\text{-closure}(\phi) = \{\phi\}$

$$\delta'(9,a) = \delta''(9,a)$$
$$= \varepsilon\text{-closure}(\delta(\delta''(9,\varepsilon)),a)$$
$$= \varepsilon\text{-closure}(\delta(9,a))$$
$$= \varepsilon\text{-closure}(10) = \{10\}$$

$$\delta'(9,b) = \delta''(9,b)$$
$$= \varepsilon\text{-closure}(\delta(\delta''(9,\varepsilon)),b)$$
$$= \varepsilon\text{-closure}(\delta(9,b))$$
$$= \varepsilon\text{-closure}(\phi) = \{\phi\}$$

$$\delta'(10,a) = \delta''(10,a)$$
$$= \varepsilon\text{-closure}(\delta(\delta''(10,\varepsilon)),a)$$
$$= \varepsilon\text{-closure}(\delta(10,a))$$
$$= \varepsilon\text{-closure}(\phi) = \{\phi\}$$

$$\delta'(10,b) = \delta''(10,b)$$
$$= \varepsilon\text{-closure}(\delta(\delta''(10,\varepsilon)),b)$$
$$= \varepsilon\text{-closure}(\delta(10,b))$$
$$= \varepsilon\text{-closure}(11) = \{11\}$$

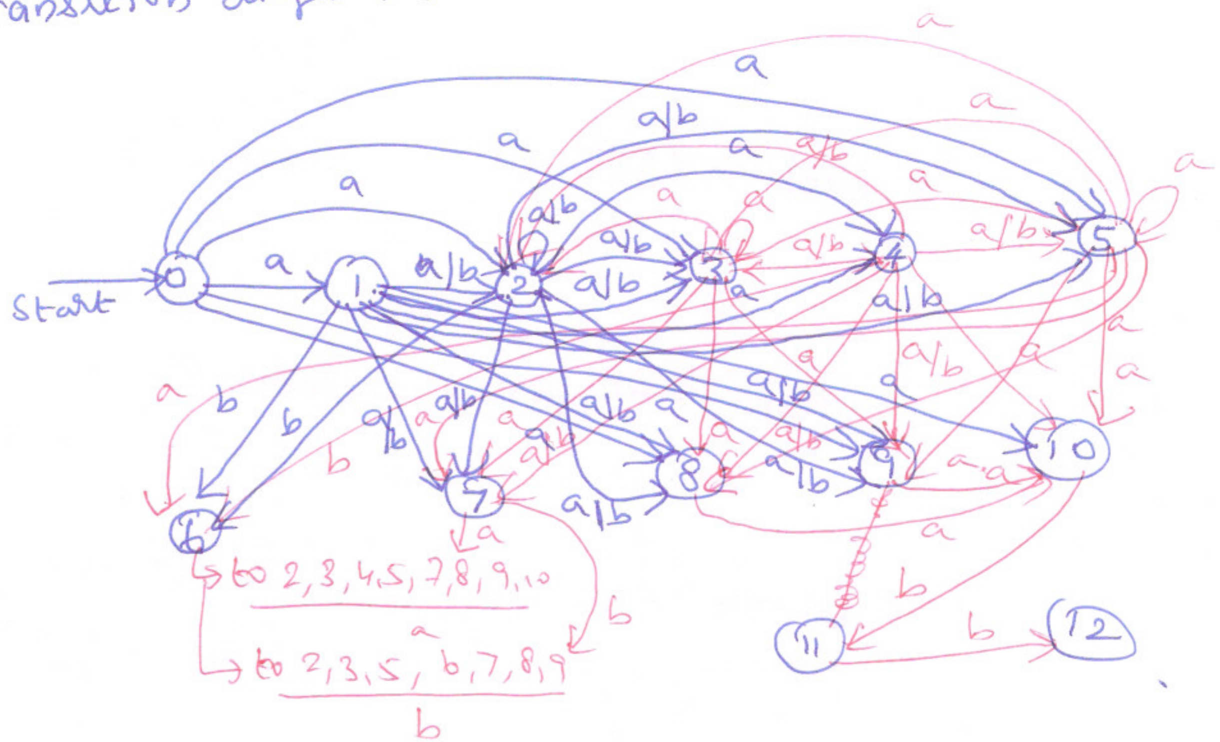
$$\delta'(11,a) = \delta''(11,a)$$
$$= \varepsilon\text{-closure}(\delta(\delta''(11,\varepsilon)),a)$$
$$= \varepsilon\text{-closure}(\delta(11,a))$$
$$= \varepsilon\text{-closure}(\phi) = \{\phi\}$$

$$\delta'(11,b) = \delta''(11,b)$$
$$= \varepsilon\text{-closure}(\delta(\delta''(11,\varepsilon)),b)$$
$$= \varepsilon\text{-closure}(\delta(11,b))$$
$$= \varepsilon\text{-closure}(12) = \{12\}$$

$$\delta'(12,a) = \delta''(12,a)$$
$$= \varepsilon\text{-closure}(\delta(\delta''(12,\varepsilon)),a)$$
$$= \varepsilon\text{-closure}(\delta(12,a))$$
$$= \varepsilon\text{-closure}(\phi) = \{\phi\}$$

$$\delta'(12,b) = \delta''(12,b)$$
$$= \varepsilon\text{-closure}(\delta(\delta''(12,\varepsilon)),a)$$
$$= \varepsilon\text{-closure}(\delta(12,b)$$
$$= \varepsilon\text{-closure}(\phi) = \{\phi\}$$

Transition diagram for NFA with ε-move.



Convert NFA with ε-move into DFA.

From Fig ①.

ε-closure (0) = {0} → (A) new state

ε-closure (δ (A, a)) = ε-closure (1)
= {1, 2, 3, 5, 8, 9} → (B)

ε-closure (δ (A, b)) = ε-closure (φ) = φ.

ε-closure (δ (B, a)) = ε-closure (4, 10)
= {2, 3, 4, 5, 7, 8, 9, 10} → (C)

ε-closure (δ (B, b)) = ε-closure (6)
= {2, 3, 5, 6, 7, 8, 9} → (D)

ε-closure (δ (C, a)) = ε-closure (4, 10)
= {2, 3, 4, 5, 7, 8, 9, 10} → (C)

ε-closure (δ (C, b)) = ε-closure (6, 11)
= {2, 3, 5, 6, 7, 8, 9, 11} → (E)

ε-closure $(\delta(D,a))$ = ε-closure $(4,10)$
$= \{2,3,4,5,7,8,9,10\} \rightarrow$ (C)

ε-closure $(\delta(D,b))$ = ε-closure (6)
$= \{2,3,5,6,7,8,9\} \rightarrow$ (D)

ε-closure $(\delta(E,a))$ = ε-closure $(4,10)$
$= \{2,3,4,5,7,8,9,10\} \rightarrow$ (C)

ε-closure $(\delta(E,b))$ = ε-closure $(6,12)$
$= \{2,3,5,6,7,8,9,12\} \rightarrow$ (F)

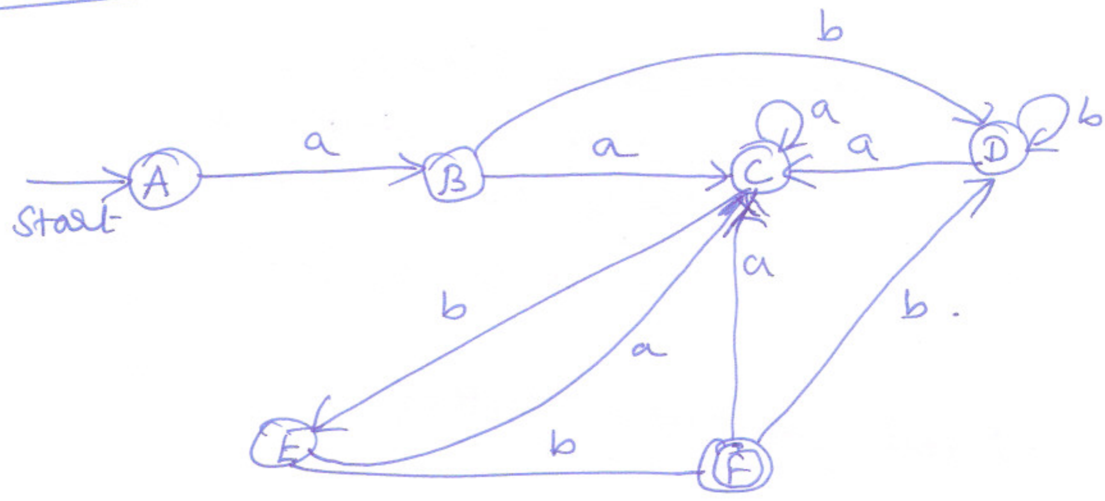
ε-closure $(\delta(F,a))$ = ε-closure $(4,10)$
$= \{2,3,4,5,7,8,9,10\} \rightarrow$ (C)

ε-closure $(\delta(F,b))$ = ε-closure (6)
$= \{2,3,5,6,7,8,9\} \rightarrow$ (D)

DFA Transition Table:

State	Input a	Input b
→ A	B	ϕ
B	C	D
C	C	E
D	C	D
E	C	F
* F	C	D.

DFA diagram:



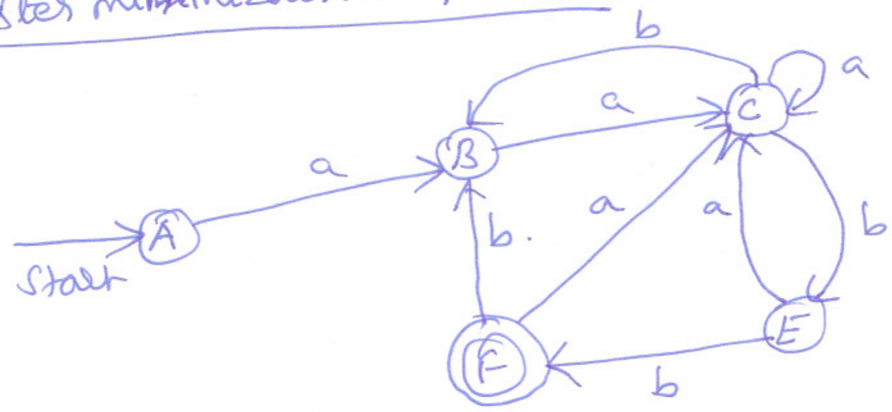
Minimization of DFA

State	Input a	Input b
*A	B	ϕ
B	C	D
C	C	E
D	C	D
E	C	F
*F	C	D

⟹

State	Input a	Input b
*A	B	ϕ
B	C	B
C	C	E
E	C	F
*F	C	B

After minimization of DFA:



Regular Expression from DFA

- There are three methods
  - Direct substitution method
  - Using Ad theorems
  - By State elimination Technique.

Direct substitution Method:

Theorem : 2

If L is accepted by a DFA, then L is denoted by a regular expression.

Proof:

Let L be the set accepted by DFA.

$M = (Q, \Sigma, \delta, q_1, F)$

where $Q = \{q_1, q_2, q_3, \ldots, q_n\}$

Let R_{ij}^{k} be the set of strings that takes M from state q_i to state q_j without entering and leaving through any state numbered higher than k.

ie if $x \in R_{ij}^{k}$ then

$\delta(q_i, x) = q_j$ & if $\delta(q_j, y) = q_l$

for any y which is of the form

$x = y x_1 \ldots x_n$, then $l \leq k$ or $y = \varepsilon$ & $l = i$ or

$x = y$ & $j = l$

- Note that R_{ij}^{k} denotes all strings that take q_i to q_j as there is no state numbered greater than n.

- We can define R_{ij}^{k} recursively as follows, when the input string is given, M moves from q_i to q_j without passing through a state higher than q_k are either.

(i) In R_{ij}^{k-1} (ie. they never pass through a state as - high as q_k)

(ii) Composed of a string in R_{ik}^{k-1} (which takes M to q_k first time) followed by zero or more string in R_{kk}^{k-1} (which take M from q_k back to q_k without passing through q_k or a higher-numbered state) followed by string in R_{kj}^{k-1} (which takes M from state q_k to q_j)

ie. $R_{ij}^{k-1} = R_{ij}^{k} \cup R_{ik}^{k-1} (R_{kk}^{k-1})^* R_{kj}^{k-1}$ → ①

$$R_{ij}^{0} = \begin{cases} a & : [\delta(q_i, a) = q_j] \text{ if } i \neq j \} \\ a & : [\delta(q_i, a) = q_j] \cup \{\varepsilon\} \text{ if } i = j. \end{cases}$$

- We show for each i, j, k there exists a r.e r_{ij}^{k} denoting the R_{ij}^{k}. We proceed by induction of k.

<u>Basis Step:</u>

When $k=0$, $R_{ij}^{0} = \{a\}$ or $\{\varepsilon\}$

Thus r_{ij}^{0} can be written as $a_1 + a_2 + \dots + a_p$, if $i \neq j$

(or)

$a_1 + a_2 + \dots + a_p + \varepsilon$, if $i = j$

where $a_1, a_2, \dots, a_p$ is a set of symbols 'a'.

Show that $\delta(q_i, a) = q_j$

If there are such 'a's then ϕ or (ε in the case $i=j$) is r_{ij}^{0}.

<u>Induction Step:</u>

Assume by induction that, for each l and m, there exists a r.e r_{lm}^{k-1}

Show that $L(r_{lm}^{k-1}) = R_{lm}^{k-1}$

Hence by ① there exists a r.e r_{ij}^{k}

Show that

$$r_{ij}^{k} = r_{ik}^{k-1} \cdot (r_{kk}^{k-1})^{*} r_{kj}^{k-1} + r_{ij}^{k-1}$$

- This completes the induction as the recursive formula for R_{ij}^{k} involves only the r.e operations union, concatenation & closure.

- To complete the proof we observe that

$$T(M) = \bigcup_{q_j \in F} R_{ij}^{n}$$

- Since R_{ij}^{n} denotes the set of all strings that take q_l to q_j. T(M) is denoted by the r.e.

$$r_{ij_1}^{n} + r_{ij_2}^{n} + \cdots + r_{ij_p}^{n}$$

where $F = \{q_{j_1}, q_{j_2}, \ldots, q_{j_p}\}$

Hence the proved.

Identities for r.e :

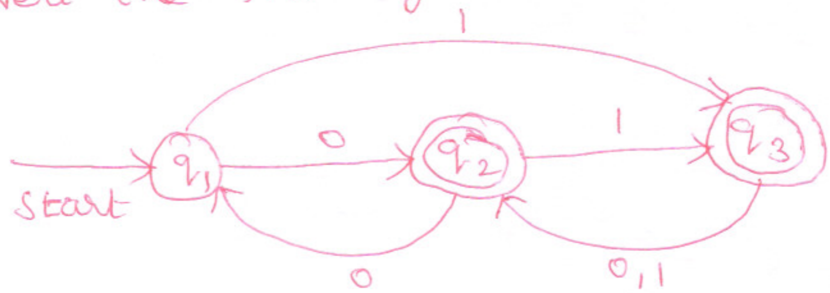
1. $\phi + R = R$
2. $\phi R = R\phi = \phi$
3. $\varepsilon R = R\varepsilon = R$
4. $\varepsilon^{*} = \varepsilon$ & $\phi^{*} = \varepsilon$
5. $R + R = R$
6. $R^{*} R^{*} = R^{*}$
7. $RR^{*} = R^{*} R$
8. $(R^{*})^{*} = R^{*}$
9. $\varepsilon + RR^{*} = R^{*} = \varepsilon + R^{*} R$
10. $(PQ)^{*} P = P(QP)^{*}$
11. $(P+Q)^{*} = (P^{*} Q^{*})^{*} = (P^{*} + Q^{*})^{*}$
12. $(P+Q)R = PR + QR$ &
13. $R(P+Q) = RP + RQ$.

Examples

1. $(\varepsilon+1)^* = 1^*$
2. $1^*(\varepsilon+1) = 1^*$
3. $(\varepsilon+1)+1^* = 1^*$
4. $0+1^*0 = 1^*0$
5. $0+01^* = 01^*$
6. $\varepsilon+00^* = 0^*$
7. $(1+0)^* = (1^*0^*)^*$
8. $\phi 0 = \phi$
9. $\phi+0 = 0$
10. $\phi^* = \varepsilon$
11. $11^* \neq 1^*$
12. $1^*1 \neq 1^*$
13. $(01)^* \neq 0^*1^*$
14. $\varepsilon 0 = (\varepsilon^*)0 = 0$
15. $(01)^*0 = 0(10)^*$
16. $(0^*1)^*0^* = (0+1)^*$
17. $\varepsilon+\varepsilon = \varepsilon$
18. $\varepsilon \cdot \varepsilon = \varepsilon$
19. $\varepsilon(\varepsilon^*) = (\varepsilon^*)\varepsilon = \varepsilon$
20. $1+1 = 1$
21. $0+0 = 0$
22. $\varepsilon+(00)^* = (00)^*$
23. $0+0(00)^* = 0(00)^*$
24. $1+1(11)^* = 1(11)^*$
25. $(00)^*(\varepsilon+00) = (00)^*$

Problems

1. Convert the following to a r.e.



Solution:

$$r.e = R_{12}^{(3)} + R_{13}^{(3)}$$

$$R_{ij}^{(0)} = \begin{cases} \varepsilon + a_1 + a_2 + \cdots + a_l, & \text{if } i=j \\ a_1 + a_2 + a_3 \cdots + a_l, & \text{if } i \neq j \end{cases}$$

K = 0

$R_{11}^{(0)} = \varepsilon$ $\quad R_{21}^{(0)} = 0$ $\quad R_{31}^{(0)} = \phi$

$R_{12}^{(0)} = 0$ $\quad R_{22}^{(0)} = \varepsilon$ $\quad R_{32}^{(0)} = 0+1$

$R_{13}^{(0)} = 1$ $\quad R_{23}^{(0)} = 1$ $\quad R_{33}^{(0)} = \varepsilon$

K = 1

$$R_{ij}^{K} = R_{ij}^{(K-1)} + R_{jK}^{(K-1)} \left(R_{KK}^{(K-1)}\right)^* R_{Kj}^{(K-1)}$$

$$R_{ij}^{(1)} = R_{ij}^{(0)} + R_{jK}^{(0)} \left(R_{KK}^{(0)}\right)^* R_{Kj}^{(0)}$$

$$R_{11}^{(1)} = R_{11}^{(0)} + R_{11}^{(0)} \left(R_{11}^{(0)}\right)^* R_{11}^{(0)}$$

$$= \varepsilon + \varepsilon(\varepsilon^*)\varepsilon$$

$$= \varepsilon + \varepsilon = \varepsilon$$

$$\therefore \boxed{R_{11}^{(1)} = \varepsilon}$$

$$R_{12}^{(1)} = R_{12}^{(0)} + R_{21}^{(0)} \left(R_{11}^{(0)}\right)^* R_{12}^{(0)}$$

$$= 0 + 0(\varepsilon)^* 0 = 0 + 0 = 0$$

$$\therefore \boxed{R_{12}^{(1)} = 0}$$

$$R_{13}^{(1)} = R_{13}^{(0)} + R_{11}^{(0)} \left(R_{11}^{(0)}\right)^* R_{13}^{(0)}$$

$$= 1 + \varepsilon(\varepsilon)^* \cdot 1 = 1 + 1 = 1$$

$$\therefore \boxed{R_{13}^{(1)} = 1}$$

$$R_{21}^{(1)} = R_{21}^{(0)} + R_{21}^{(0)} (R_{11}^{(0)})^* R_{11}^{(0)}$$

$$= 0 + 0(\varepsilon)^* \cdot \varepsilon = 0 + 0 = 0$$

$$\therefore \boxed{R_{21}^{(1)} = 0}$$

$$R_{22}^{(1)} = R_{22}^{(0)} + R_{21}^{(0)} (R_{11}^{(0)})^* R_{12}^{(0)}$$

$$= \varepsilon + 0(\varepsilon)^* 0 = \varepsilon + 00(\varepsilon)^* = \varepsilon + 00$$

$$\therefore \boxed{R_{22}^{(1)} = \varepsilon + 00}$$

$$R_{23}^{(1)} = R_{23}^{(0)} + R_{21}^{(0)} (R_{11}^{(0)})^* R_{13}^{(0)}$$

$$= 1 + 0(\varepsilon)^* 1 = 1 + 01$$

$$\therefore \boxed{R_{23}^{(1)} = 1 + 01}$$

$$R_{31}^{(1)} = R_{31}^{(0)} + R_{31}^{(0)} (R_{11}^{(0)})^* R_{11}^{(0)}$$

$$= \phi + \phi(\varepsilon)^* \varepsilon = \phi + \phi = \phi$$

$$\therefore \boxed{R_{31}^{(1)} = \phi}$$

$$R_{32}^{(1)} = R_{32}^{(0)} + R_{31}^{(0)} (R_{11}^{(0)})^* R_{12}^{(0)}$$

$$= (0+1) + \phi(\varepsilon)^* 0 = (0+1) + \phi = 0+1$$

$$R_{33}^{(1)} = R_{33}^{(0)} + R_{31}^{(0)} (R_{11}^{(0)})^* R_{13}^{(0)}$$

$$= \varepsilon + \phi(\varepsilon)^* 1 = \varepsilon + \phi = \varepsilon .$$

$\underline{K = 2}$

$$R_{11}^{(2)} = R_{11}^{(1)} + R_{12}^{(1)} (R_{22}^{(1)})^* R_{21}^{(1)}$$

$$= \varepsilon + 0 (\varepsilon + 00)^* 0$$

$$= \varepsilon + 0 (00)^* 0 = \varepsilon + (00)^* = (00)^*$$

$$\therefore \boxed{R_{11}^{(2)} = (00)^*}$$

$$R_{12}^{(2)} = R_{12}^{(1)} + R_{12}^{(1)} (R_{22}^{(1)})^* R_{22}^{(1)}$$

$$= 0 + 0 (\varepsilon + 00)^* (\varepsilon + 00)$$

$$= 0 + 0 (00)^* (\varepsilon + 00)$$

$$= 0 + 0 (00)^* = 0 (00)^*$$

$$\therefore \boxed{R_{12}^{(2)} = 0 (00)^*}$$

$$R_{13}^{(2)} = R_{13}^{(1)} + R_{12}^{(1)} (R_{22}^{(1)})^* R_{23}^{(1)}$$

$$= 1 + 0 (\varepsilon + 00)^* (1 + 01)$$

$$= 1 + 0 (00)^* (1 + 01)$$

$$= 1 + (00)^* (\varepsilon + 0) 1$$

$$= 1 + 0^* 1 = 0^* 1.$$

$$\therefore \boxed{R_{13}^{(2)} = 0^* 1}$$

$$R_{21}^{(2)} = R_{21}^{(1)} + R_{22}^{(1)} (R_{22}^{(1)})^* R_{21}^{(1)}$$

$$= 0 + (\varepsilon + 00) (\varepsilon + 00)^* 0$$

$$= 0 + (\varepsilon + 00) (00)^* 0$$

$$= 0 + (00)^* 0 = (00)^* 0$$

$$\therefore \boxed{R_{21}^{(2)} = (00)^* 0}$$

$$R_{22}^{(2)} = R_{22}^{(1)} + R_{22}^{(1)} (R_{22}^{(1)})^* R_{22}^{(1)}$$

$$= (\varepsilon+00) + (\varepsilon+00)(\varepsilon+00)^*(\varepsilon+00)$$

$$= (\varepsilon+00) + (\varepsilon+00)(00)^*(\varepsilon+00)$$

$$= (\varepsilon+00) + (00)^*(\varepsilon+00)$$

$$= (\varepsilon+00) + (00)^* = 00^*$$

$$\therefore \boxed{R_{22}^{(2)} = 00^*}$$

$$R_{23}^{(2)} = R_{23}^{(1)} + R_{22}^{(1)} (R_{22}^{(1)})^* R_{23}^{(1)}$$

$$= (1+01) + (\varepsilon+00)(\varepsilon+00)^*(1+01)$$

$$= (1+01) + (\varepsilon+00)(00)^*(1+01)$$

$$= (1+01) + (00)^*(1+01)$$

$$= (1+01) + (00)^*(\varepsilon+0)1$$

$$= (1+01) + 0^*1 = 1+01+0^*1 = 1+0^*1$$

$$= 0^*1. \quad \therefore \boxed{R_{23}^{(2)} = 0^*1}$$

$$R_{31}^{(2)} = R_{31}^{(1)} + R_{32}^{(1)} (R_{22}^{(1)})^* R_{21}^{(1)}$$

$$= \phi + (0+1)(\varepsilon+00)^*0 = (0+1)(00)^*0$$

$$\therefore \boxed{R_{31}^{(2)} = (0+1)(00)^*0}$$

$$R_{32}^{(2)} = R_{32}^{(1)} + R_{32}^{(1)} (R_{22}^{(1)})^* R_{22}^{(1)}$$

$$= (0+1) + (0+1)(\varepsilon+00)^*(\varepsilon+00)$$

$$= (0+1) + (0+1)(00)^*(\varepsilon+00)$$

$$= (0+1) + (0+1)(00)^*$$

$$= (0+1)(00)^*$$

$$\therefore \boxed{R_{32}^{(2)} = 0+1(00)^*}$$

(229)

$$R_{33}^{(2)} = R_{33}^{(1)} + R_{32}^{(1)} \left(R_{22}^{(1)}\right)^* R_{23}^{(1)}$$

$$= \varepsilon + (0+1)(\varepsilon + 00)^* (1+01)$$

$$= \varepsilon + (0+1)(00)^* (1+01)$$

$$= \varepsilon + (0+1)(00)^* (\varepsilon + 0)1$$

$$= \varepsilon + (0+1) 0^* 1$$

$$= \varepsilon + (0+1) 0^* 1$$

$\gamma_{12}^{(3)} + \gamma_{13}^{(3)}$ r.e.

$$\gamma_{12}^{(3)} = \gamma_{12}^{(2)} + \gamma_{13}^{(2)} \left(\gamma_{33}^{(2)}\right)^* \gamma_{32}^{(2)}$$

$$= 0(00)^* + 0^*1 \left(\varepsilon + (0+1) 0^* 1\right)^* (0+1)(00)^*$$

$$= 0(00)^* + 0^*1 \left((0+1) 0^* 1\right)^* (0+1)(00)^*$$

$$\gamma_{13}^{(3)} = \gamma_{13}^{(2)} + \gamma_{13}^{(2)} \left(\gamma_{33}^{(2)}\right)^* \gamma_{32}^{(2)}$$

$$= 0^*1 + 0^*1 \left(\varepsilon + (0+1) 0^* 1\right)^* (\varepsilon + 0(0+1) 0^* 1)$$

$$= 0^*1 \left((0+1) 0^* 1\right)^*.$$

Hence Regular Expression is

$$\text{r.e} = \gamma_{12}^{(3)} + \gamma_{13}^{(3)} = 0^*1\left((0+1) 0^* 1\right)^* (\varepsilon + (0+1)(00)^*) + 0(00)^*.$$

$$\boxed{\text{r.e.} = 0^*1\left((0+1) 0^* 1\right)^* \left(\varepsilon + (0+1)(00)^*\right) + 0(00)^*}$$

$$R_{33}^{(2)} = R_{33}^{(1)} + R_{32}^{(1)} \left(R_{22}^{(1)}\right)^* R_{23}^{(1)}$$

$$= \varepsilon + (0+1)(\varepsilon + 00)^* (1+01)$$

$$= \varepsilon + (0+1)(00)^* (1+01)$$

$$= \varepsilon + (0+1)(00)^* (\varepsilon + 0)1$$

$$= \varepsilon + (0+1)\, 0^* 1$$

$$= \varepsilon + (0+1)\, 0^* 1$$

$r_{12}^{(3)} + r_{13}^{(3)}$ r.e.

$$r_{12}^{(3)} = r_{12}^{(2)} + r_{13}^{(2)} \left(r_{33}^{(2)}\right)^* r_{32}^{(2)}$$

$$= 0(00)^* + 0^*1\left(\varepsilon + (0+1)\, 0^*1\right)^* (0+1)(00)^*$$

$$= 0(00)^* + 0^*1\left((0+1)\, 0^*1\right)^* (0+1)(00)^*$$

$$r_{13}^{(3)} = r_{13}^{(2)} + r_{13}^{(2)} \left(r_{33}^{(2)}\right)^* r_{32}^{(2)}$$

$$= 0^*1 + 0^*1\left(\varepsilon + (0+1)\, 0^*1\right)^* \left(\varepsilon + 0(0+1)\, 0^*1\right)$$

$$= 0^*1\left((0+1)\, 0^*1\right)^*.$$

Hence Regular Expression is

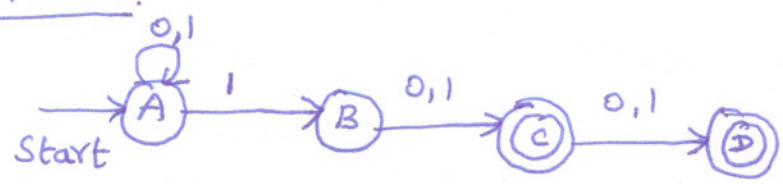
$$r.e = r_{12}^{(3)} + r_{13}^{(3)} = 0^*1\left((0+1)\, 0^*1\right)^* \left(\varepsilon + (0+1)(00)^*\right) + 0(00)^*.$$

$$\boxed{r.e. = 0^*1\left((0+1)\, 0^*1\right)^* \left(\varepsilon + (0+1)(00)^*\right) + 0(00)^*}$$

Method : 2 - State Elimination Technique

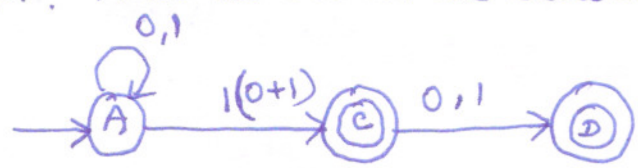
- The construction of r.e is complex for more no. of states. So, apply state elimination Technique.

Problem:



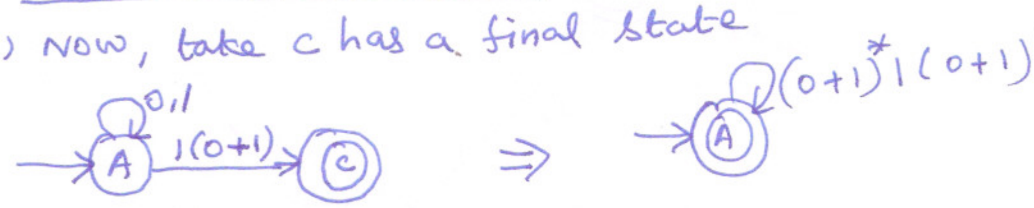
Solution:

Step 1: First to eliminate state B, then find the r.e.



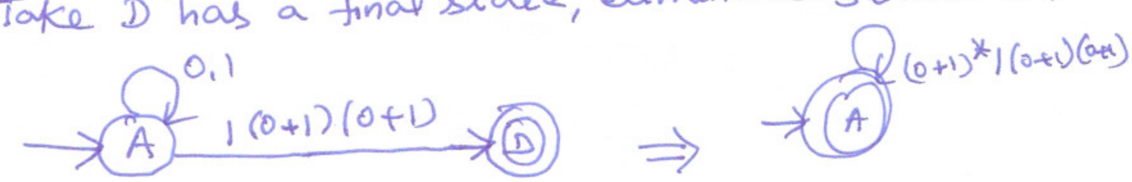
Step 2: There are two final state

(i) Now, take C has a final state



$r.e_1 = (0+1)^* 1 (0+1)$

(ii) Take 'D' has a final state, eliminate state C.



$r.e._2 = (0+1)^* 1 (0+1)(0+1)$

$r.e = r.e.1 + r.e.2 \qquad \therefore R = R_1 + R_2$

$R = (0+1)^* 1 (0+1) + (0+1)^* 1 (0+1)(0+1)$

$R = (0+1)^* 1 (0+1) (\varepsilon + (0+1))$

$\boxed{R = (0+1)^* 1 (0+1)(\varepsilon + 0 + 1)}$

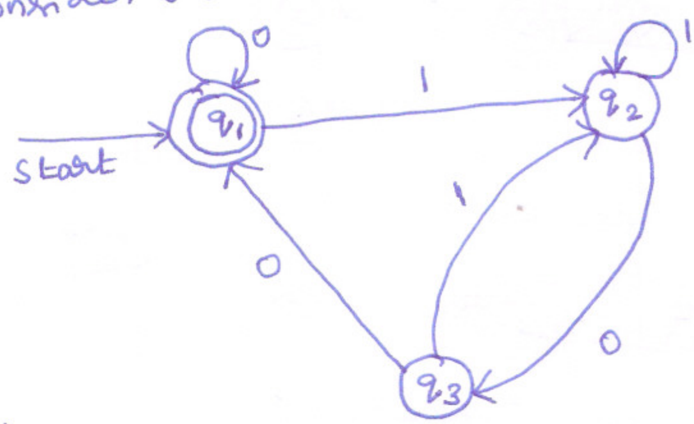
Method 3: Arden's Theorem

Let P and Q be two regular expressions over Σ. If P does not contain ε, then, the equation in $R = Q + RP$ has a solution $R = QP^*$

- Using this theorem, it is easy to find the r.e.
- The conditions to apply this theorem are
  - * Finite automata does not have ε-moves.
  - * It has only one start state.

Problem:

Consider the r.e for the given finite automata.



Solution:

The transition are defined by the equations.

$$q_1 = q_1 0 + q_3 0 + \varepsilon \longrightarrow (1)$$

$$q_2 = q_1 1 + q_2 1 + q_3 1 \longrightarrow (2)$$

$$q_3 = q_2 0 \longrightarrow (3)$$

Substitute (3) in (2)

$$q_2 = q_1 1 + q_2 1 + (q_2 0) 1$$

$$q_2 = q_1 1 + q_2 1 + q_2 01$$

$$\underset{R}{q_2} = \underset{Q}{q_1 1} + \underset{R}{q_2} \underset{P}{(1 + 01)}$$

$$q_2 = q_1 1 (1 + 01)^* \rightarrow (4) \therefore \text{By Theorem.}$$

Substitute ③ in ①

$$q_1 = q_1 0 + q_3 0 + \varepsilon \rightarrow ①$$

$$q_1 = q_1 0 + (q_2 0) 0 + \varepsilon$$

$$q_1 = q_1 0 + q_2 00 + \varepsilon \longrightarrow ⑤$$

Substitute ④ in ⑤

$$q_1 = q_1 0 + q_1 1(1+01)^* 00 + \varepsilon$$

$$q_1 = q_1 (0 + 1(1+01)^* 00) + \varepsilon$$

$$\underset{R}{q_1} = \underset{Q}{\varepsilon} + \underset{R}{q_1}\underset{P}{(0 + 1(1+01)^* 00)}$$

$$q_1 = \varepsilon(0+1(1+01)^* 00)^*$$

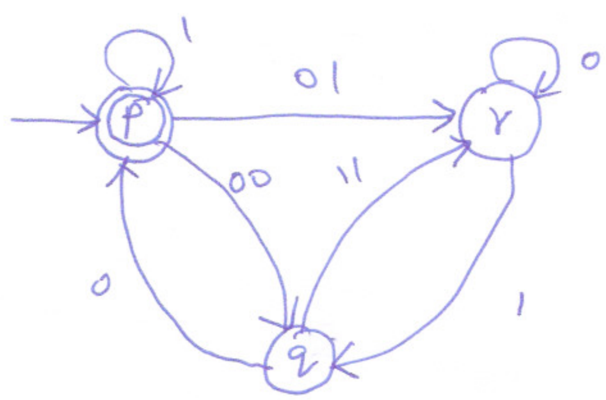
$$q_1 = (0+1(1+01)^* 00)^*$$

Hence q_1 is a final state, then regular expression is

$$\boxed{r.e = (0+1(1+01)^* 00)^*}$$

Tutorial :

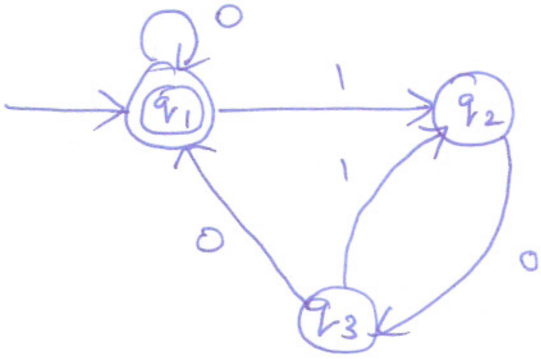
Find the r.e for the given DFA using Arden's theorem.



Ans :

$$\boxed{Re = (00+010^* 1)(10+110^* 1)^* 01^*.}$$

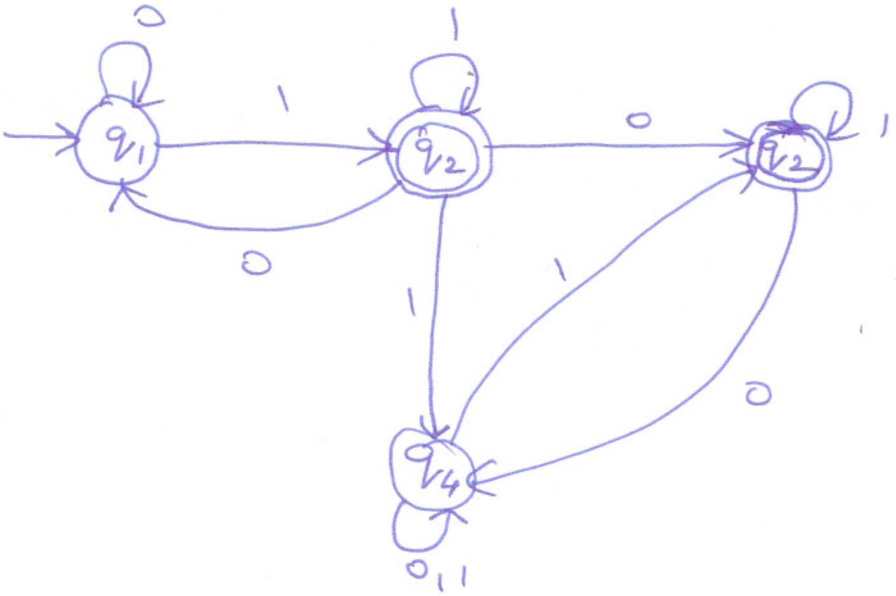
(2) Find the r.e for the given FA using SET.



Ans :

$$R.E = (0+1+(1+01)^{*}00)^{*}.$$

(3) Convert FA into R.E using Arden's theorem



Ans,

R.E = value in q_2 + value in q_3

1. Show that $(0^* 1^*)^* = (0+1)^*$

$LHS = (0^* 1^*)^*$

$= \{\varepsilon, 0, 1, 00, 11, 1111, 01, 10, 0011, \ldots\}$

$L = \{$any combination of 0's, any combinations of 1's, any combinations of 0's & 1's with $\varepsilon\}$

$RHS = (0+1)^*$

$= \{\varepsilon, 0, 1, 00, 11, 1111, 01, 10, 0011, \ldots\}$

$L = \{$any combinations of 0's, 1's & $\varepsilon\}$

Hence LHS = RHS is proved.

2. Show that $(ab)^* \neq (a^* b^*)$

$LHS = (ab)^* = \{\varepsilon, ab, abab, ababab, \ldots\}$

$RHS = (a^* b^*) = \{\varepsilon, a, b, ab, aa, bb, aabb, \ldots\}$

Hence LHS = RHS is proved.

3. Show that $(r+s)^* \neq r^* + s^*$

$LHS = (r+s)^* = \{\varepsilon, r, s, rs, sr, rrss, \ldots\}$

$= \{$any combinations of r, s & $\varepsilon\}$

$RHS = r^* + s^* = \{\varepsilon, r, s, rr, ss, \ldots\}$

$= \{$any combinations of only r or only s with $\varepsilon\}$

∴ LHS ≠ RHS is proved.

(4) Prove that

$r(s+t) = rs + rt$

$LHS = r(s+t)$

$= rs + rt$

$= RHS.$

So, LHS = RHS is proved.

(5) Show that $S = (r^*)^* = r^*$ for a r.e

Consider $r = 0$.

$LHS = (r^*)^* = (0^*)^*$

$= \{\varepsilon, 0, 00, 000, \ldots\}$

$RHS = r^* = 0^*$

$= \{\varepsilon, 0, 00, 000, \ldots\}$

So, $\boxed{LHS = RHS}$ is proved.

Applications of Regular Expression:

* It is used in UNIX.
* It is used in lexical Analysis
* Finding pattern in a Text.

Proving Languages not to be regular:

- Powerful Technique which is used to prove that certain languages are not regular is pumping lemma.

Principle:

- For a string of length > n accepted by the DFA, the walk through of a DFA must contain a cycle.
- Repeating the cycle an arbitrary no. of times, it should yield another string accepted by the DFA.

* It is also proved that a given infinite language is not regular.

Theorem: (Pumping lemma)

- Let L be a regular language, then there is exists a constant 'n' (the no. of states that accepts L) such that if z is any word in L, then

  (i) $z = uvw$

  (ii) $|uv| \leq n$

  (iii) $|v| \geq 1$

  (iv) $uv^i w \in L$ for all $i \geq 0$.

Proof:

If a language is regular, it is accepted by a DFA $M = (Q, \Sigma, \delta, q_0, F)$ with n numbers of states

Consider an input of n or more symbols says

$a_1, a_2, \ldots a_m$, $m \geq n$ & for $i = 1, 2 \ldots, m$

Let $\delta(q_0, a_1, a_2, \ldots a_i) = q_i$

It is not possible for each of the n+1 states

$q_0, q_1 \ldots\ldots q_n$ to be distinct;

Since there are only 'n' different states

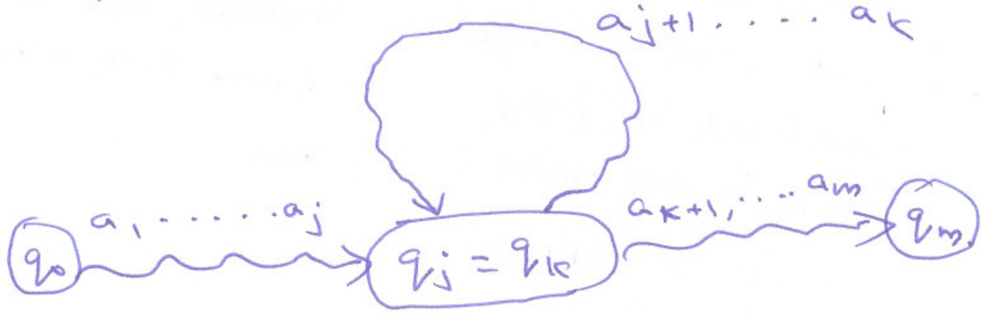
Thus there are two integers j & k, $0 \leq j < k \leq n$

Show that $q_j = q_k$.

The path labeled

$a_1, a_2 \ldots\ldots a_m$ is shown in the

diagram.



~~Since~~ Since $j<k$, the string $a_{j+1} \ldots a_k$ is of length atleast 1, and since $k \leq n$, its length is no more than ~~one~~ n.

If q_m is in F, ie. $a_1, a_2, \ldots a_m$ is in L(M),

then $a_1, a_2, \ldots a_j a_{k+1} a_{k+2} \ldots a_m)$

$= \delta(\delta(q_0, a_1, \ldots a_j), a_{k+1} \ldots a_m)$

$= \delta(q_j, a_{k+1} \ldots a_m)$

$= \delta(q_k, a_{k+1} \ldots a_m) = q_m$

Thus $a_1, \ldots a_j (a_{j+1} \ldots a_k)^i a_{k+1} \ldots a_m$ is in L(M).
for any $i \geq 0$.

Syllabus : Unit – III : Regular Grammars and Context Free Grammars

Types of Grammars - Regular grammars – Right Linear and Left Linear grammars - Equivalence of regular grammar and Finite Automata - Context free Grammars - Motivation and introduction - Derivations - Leftmost derivation - Rightmost derivation - Derivation tree – Ambiguity - Simplification of CFG's - Chomsky Normal Form - Greibach Normal Form.

Introduction

- ✓ **Language**: "A language is a collection of sentences of finite length all constructed from a finite alphabet of symbols."
- ✓ **Grammar**: "A grammar can be regarded as a device that enumerates the sentences of a language."

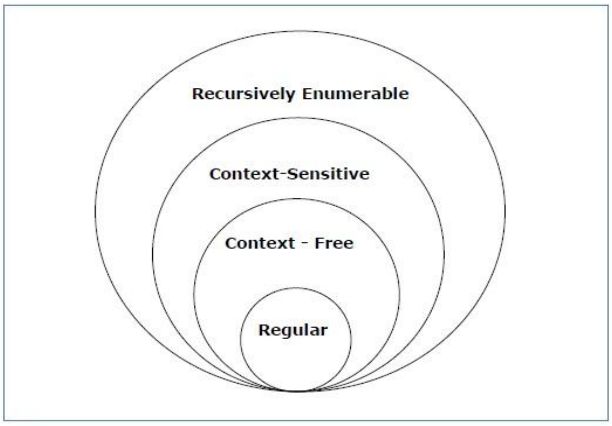
- ✓ A formal grammar is a quad-tuple G = (N, T, P, S)
  where
  - N is a finite set of non-terminals
  - T is a finite set of terminals and is disjoint from N
  - P is a finite set of production rules of the form $w \in (N \cup T)^* \rightarrow w \in (N \cup T)^*$
  - $S \in N$ is the start symbol

- ✓ **Chomsky Hierarchy (Types of grammars)**

Class	Grammars	Languages	Automaton	Rules
Type-0	Unrestricted Grammar	Recursively enumerable Language	Turing machine	Rules are of the form: $\alpha \rightarrow \beta$, where α and β are arbitrary strings over a vocabulary V and $\alpha \neq \varepsilon$
Type-1	Context-sensitive Grammar	Context-sensitive Language	Linear-bounded automaton	Rules are of the form: $\alpha A \beta \rightarrow \alpha B \beta$ $S \rightarrow \varepsilon$ where $A, S \in N$ $\alpha, \beta, B \in (N \cup T)^*$ $B \neq \varepsilon$
Type-2	Context-free Grammar	Context-free Language	Pushdown automaton	Rules are of the form: $A \rightarrow \alpha$ where $A \in N$ $\alpha \in (N \cup T)^*$
Type-3	Regular Grammar	Regular Language	Finite automaton	Rules are of the form: $A \rightarrow \varepsilon$ $A \rightarrow \alpha$ $A \rightarrow \alpha B$ where $A, B \in N$ and $\alpha \in T$

✓ **Scope of each type of grammar**

A figure shows the scope of each type of grammar:



✓ **Type - 3 Grammar**

- Type-3 grammars generate regular languages. Type-3 grammars must have a single non-terminal on the left-hand side and a right-hand side consisting of a single terminal or single terminal followed by a single non-terminal.
- The productions must be in the form

  X → a
  X → aY

  where X, Y ∈ N (Non terminal) and a ∈ T (Terminal)
- The rule S → ε is allowed if S does not appear on the right side of any rule.
- Example

  X → ε
  X → a | aY
  Y → b

✓ **Type - 2 Grammar**

- Type-2 grammars generate context-free languages. These languages generated by these grammars are be recognized by a non-deterministic pushdown automaton.
- The productions must be in the form

  A → γ
  where A ∈ N (Non terminal) and γ ∈ (T ∪ N)* .
- Example

  S → X a
  X → a
  X → aX
  X → abc
  X → ε

✓ **Type - 1 Grammar**

- Type-1 grammars generate context-sensitive languages.
- The productions must be in the form

  α A β → α γ β
  Where A ∈ N (Non-terminal) and α, β, γ ∈ (T ∪ N)*

- The strings α and β may be empty, but γ must be non-empty.
- The rule S → ε is allowed if S does not appear on the right side of any rule. The languages generated by these grammars are recognized by a linear bounded automaton.
- Example

  AB → AbBc
  A → bcA
  B → b

✓ **Type - 0 Grammar**

- Type-0 grammars generate recursively enumerable languages. The productions have no restrictions. They are any phase structure grammar including all formal grammars.
- They generate the languages that are recognized by a Turing machine.
- The productions can be in the form of

  α → β

  where α is a string of terminals and non-terminals with at least one non-terminal and α cannot be null. β is a string of terminals and non-terminals.
- Example

  S → ACaB
  Bc → acB
  CB → DB
  aD → Db

Regular grammars

✓ **Formal Definition of Regular Grammars**

- A regular grammar is a mathematical object, G, with four components,

  G = (N, T, P, S)

  Where

  N is a nonempty, finite set of non-terminal symbols
  T is a finite set of terminal symbols
  P is a set of grammar rules, each of one having one of the forms

  A → aB
  A → a
  A → ε, for A, B ∈ N, a ∈ T, and ε the empty string

  S is the start symbol S ∈ N

✓ **Definition: The Language Generated by a Regular Grammar**

- Let *G* = (*N*, *T*, *P*, *S*) be a regular grammar. We define the *language generated by G* to be L(*G*)
- L(*G*) = {*w* | *S* ⇒ * *w*, where *w* ∈*T**}

Linear grammar

✓ A linear grammar is a context-free grammar that has at most one non-terminal symbol on the right hand side of each grammar rule.

S → aA
A → aB
B → Bb

Left Linear grammars

- ✓ A left linear grammar is a linear grammar in which the non-terminal symbol always occurs on the left side.
- ✓ In a grammar if all productions are in the form

  $A \to B\,\alpha$

  $A \to \alpha$ where $A, B \in V$ and $\alpha \in T^*$

- ✓ Example

  A → Aa / Bb / b

Right Linear grammars

- ✓ A right linear grammar is a linear grammar in which the non-terminal symbol always occurs on the right side.
- ✓ In a grammar if all productions are in the form

  $A \to \alpha\, B$

  $A \to \alpha$ where $A, B \in V$ and $\alpha \in T^*$

- ✓ Example

  A → aA / bB / b

Converting Left Linear grammars into Right Linear grammars

- ✓ Algorithm:
  1. If the left linear grammar has a rule S → a, then make that a rule in the right linear grammar
  2. If the left linear grammar has a rule A → a, then add the following rule to the right linear grammar: S → aA
  3. If the left linear grammar has a rule B → Aa, add the following rule to the right linear grammar: A → aB
  4. If the left linear grammar has a rule S → Aa, then add the following rule to the right linear grammar: A → a

- ✓ Example 1:

Left Linear Grammar	→	Right Linear Grammar
S → Aa		S → abA
A → ab		A → a

- ✓ Example 2:

  Left Linear Grammar

  S → Ab
  S → Sb
  A → Aa
  A → a

  Step 1: Make new non-terminal

  $S_0 \to S$
  S → Ab
  S → Sb
  A → Aa
  A → a

Step 2: If the left linear grammar has this rule A → p, then add the following rule to the right linear grammar: S → pA

Left Linear Grammar	Left Linear Grammar
S_0 → S	S_0 → aA
S → Ab	
S → Sb	
A → Aa	
A → a	

Step 3: If the left linear grammar has a rule B → Ap, add the following rule to the right linear grammar: A → pB

Left Linear Grammar	Left Linear Grammar
S_0 → S	S_0 → aA
S → Ab	A → bS
S → Sb	A → aA
A → Aa	S → bS
A → a	

Step 4: If the left linear grammar has S → Ap, then add the following rule to the right linear grammar: A → p

Left Linear Grammar	Left Linear Grammar
S_0 → S	S_0 → aA
S → Ab	A → bS
S → Sb	A → aA
A → Aa	S → bS
A → a	S → ε

Step 5: Equivalent Right Linear Grammar:

S_0 → aA
A → bS
A → aA
S → bS
S → ε

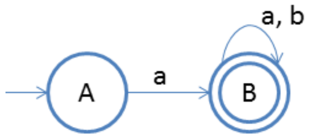
Equivalence of regular grammar and Finite Automata

✓ **Conversion of Finite Automata to Right Linear Regular Grammar**

1. Algorithm:
   1. Repeat the process for every state.
   2. Begin the process from start state.
   3. Write the production as the output followed by the state on which the transition is going.
   4. And at the last add ε because that's required to end the derivation.

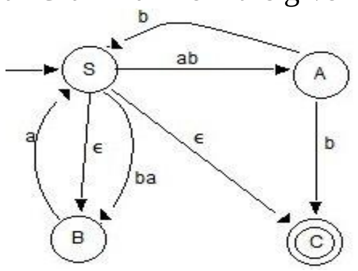
✓ Problems for Finite Automata to Right Linear Regular Grammar:

1. Construct Right Linear Grammar from the given Finite Automata



1) Pick start state and output is on symbol 'a' we are going on state B
So, we will write as :
A → aB
2) Then we will pick state B and then we will go for each output. So, we will get the below production.
B→aB/bB/ε
3) So, final we got right linear grammar as:
A → aB
B → aB/bB/ε

2. Construct Right Linear Grammar from the given Finite Automata



1) Pick start state and output is on symbol 'ab' we are going on state A
So, we will write as :
S → abA
2) Pick start state and output is on symbol 'ba' we are going on state B
So, we will write as :
S → baA
3) Pick start state and output is on symbol 'ε ' we are going on state B and C
So, we will write as :
S → B and S → ε (C is final state)
4) Then we will pick state A and then we will go for each output. So, we will get the below production.
A→ bS and A→ b (C is final state)
5) Then we will pick state B and then we will go for each output. So, we will get the below production.
B→ aS
6) Then we will pick state C and then we will go for each output. So, we will get the below production.
C→ ε
7) So, final we got right linear grammar as:
S → abA / baA / B / ε
A→ bS / b
B → aS
C → ε

✓ **<u>Conversion of Regular language to Right Linear Regular Grammar</u>**

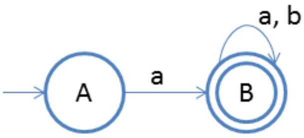
Algorithm:

1. Construct Finite automata from regular language.
2. Repeat the process for every state.
3. Begin the process from start state.
4. Write the production as the output followed by the state on which the transition is going.
5. And at the last add ε because that's required to end the derivation.

✓ **Problems for Regular language to Right Linear Regular Grammar:**

3. Construct Regular language from the given Finite Automata
   L = {All strings start with 'a' over Σ = (a+b)*}.

1) Construct Finite automata from given regular language.



2) Pick start state and output is on symbol 'a' we are going on state B
   So, we will write as :
   A → aB
3) Then we will pick state B and then we will go for each output. So, we will get the below production.
   B→aB/bB/ε
4) So, final we got right linear grammar as:
   A → aB
   B → aB/bB/ε

✓ **<u>Conversion of Regular expression to Right Linear Regular Grammar</u>**

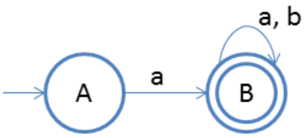
Algorithm:

1. Construct Finite automata from regular expression.
2. Repeat the process for every state.
3. Begin the process from start state.
4. Write the production as the output followed by the state on which the transition is going.
5. And at the last add ε because that's required to end the derivation.

✓ **Problems for Regular language to Right Linear Regular Grammar:**

4. Construct Regular Expression from the given Finite Automata
   r = a(a+b)*

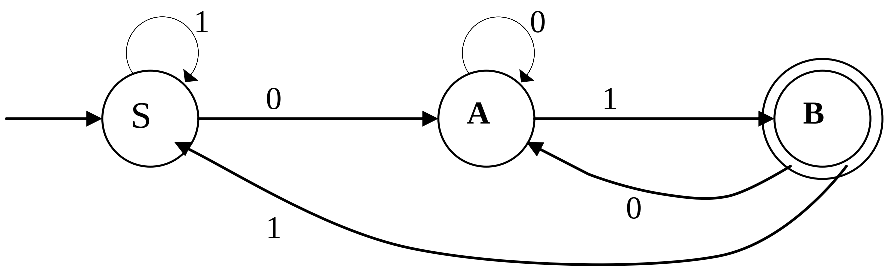
1) Construct Finite automata from given regular expression.



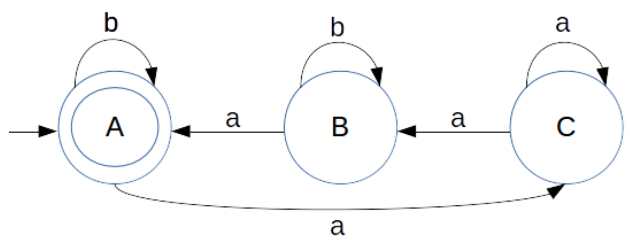
2) Pick start state and output is on symbol 'a' we are going on state B
   So, we will write as :
   A → aB
3) Then we will pick state B and then we will go for each output. So, we will get the below production.
   B→aB/bB/ε
4) So, final we got right linear grammar as:
   A → aB
   B → aB/bB/ε

Tutorial Questions:

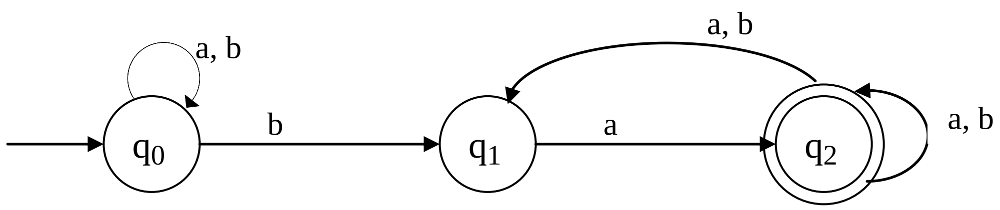
5. Construct Right Linear Grammar from the given Finite Automata



6. Construct Right Linear Grammar from the given Finite Automata



7. Construct Right Linear Grammar from the given Finite Automata



8. Construct Right Linear Grammar from the following Regular Languages.
   a. L = {All the strings starting and ending with 'a' and having any combinations of b's in between over Σ = (a, b)}.
   b. L = {The set of all strings of 0's and 1's ending in 00 over Σ = (0, 1)}.
   c. L = {The set of all strings of 0's and 1's beginning with 0 and ending with 1 over Σ = (0, 1)}.

9. Construct Right Linear Grammar from the following Regular Expressions.
   a. r = (0+1)*11
   b. r = a(a+b)*b

✓ **<u>Conversion of Right Linear Regular Grammar to Finite Automata</u>**

Algorithm:

Given a regular grammar G, a finite automata accepting L(G) can be obtained as follows:

1. The number of states in the automata will be equal to the number of non-terminals plus one. Each state in automata represents each non-terminal in the regular grammar. The additional state will be the final state of the automata. The state corresponding to the start symbol of the grammar will be the initial state of automata. If L(G) contains ϵ that is start symbol is grammar devices to ϵ, then make start state also as final state.
2. The transitions for automata are obtained as follows:
   - For every production A → aB, then make δ(A, a) = B that is make an are labeled 'a' from A to B.
   - For every production A → a, then make δ(A, a) = final state.
   - For every production A → ϵ, then make δ(A, ϵ) = A and A will be final state.

✓ **Problems for Right Linear Regular Grammar to Finite Automata**

1. Construct a Finite Automata from the given Right Linear Grammar

A → aB/bA/b
B → aC/bB
C → aA/bC/a

Solution:

Step 1: Take the 'A' productions, then will make transition functions

A → aB → δ(A, a) = B
A → bA → δ(A, b) = A
A → b → δ(A, b) = Final State

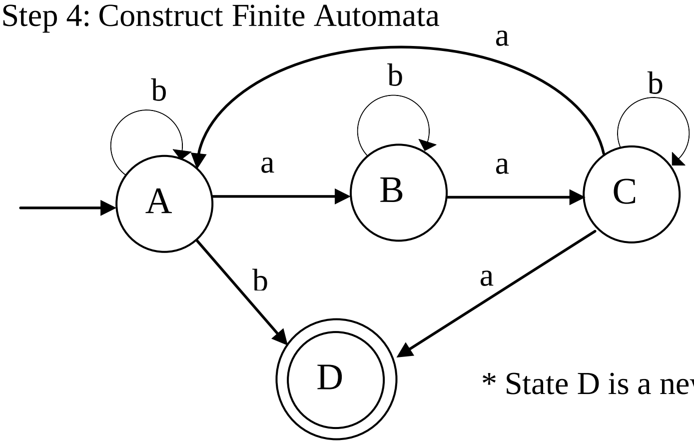
Step 2: Take the 'B' productions, then will make transition functions

B → aC → δ(B, a) = C
B → bB → δ(B, b) = B

Step 3: Take the 'C' productions, then will make transition functions

C → aA → δ(C, a) = A
C → bC → δ(C, b) = C
C → b → δ(C, b) = Final State

Step 4: Construct Finite Automata



* State D is a new final State

2. Construct a Finite Automata from the given Right Linear Grammar

   S → A / B / ε
   A → 0S/1B/0
   B → 0S/1A/1

Solution:

Step 1: Take the 'S' productions, then will make transition functions

S → A → δ(S, ε) = A
S → B → δ(S, ε) = B
S → ε → δ(S, ε) = S and S is make Final State

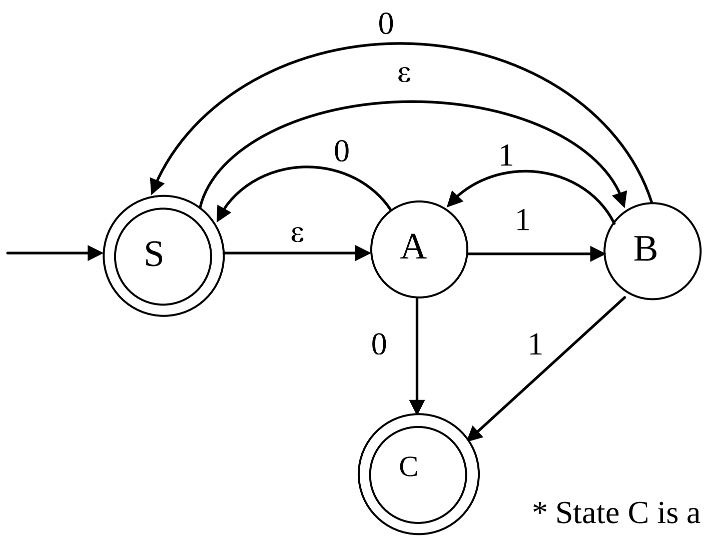
Step 2: Take the 'A' productions, then will make transition functions

A → 0S → δ(A, 0) = S
A → 1B → δ(A, 1) = B
A → 0 → δ(A, 0) = Final State

Step 3: Take the 'B' productions, then will make transition functions

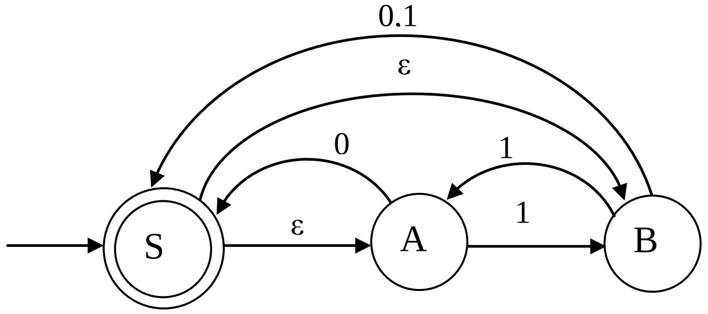
B → 0S → δ(B, 0) = S
B → 1A → δ(B, 1) = A
B → 1 → δ(B, 1) = Final State

Step 4: Construct Finite Automata



* State C is a new final State

Step 5: Reconstructed Finite Automata (after removing state C)



Tutorial Questions:

3. Construct a Finite Automata from the given Right Linear Grammar
   S → abA / baA / B / ε
   A→ bS / b
   B → aS
   C → ε

4. Construct a Finite Automata from the given Right Linear Grammar
   A → aB
   B → aB/bB/ε
5. Give the Finite Automata from the given Right Linear Grammar
   S → 0S/1A/1/0B/0
   A → 0A/1B/0/1
   B → 0B/1A/0/1

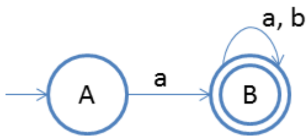
✓ **Conversion of Finite Automata to Left Linear Regular Grammar**

Algorithm:

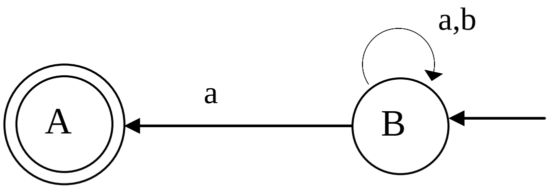
1. Take reverse of the finite automata
2. Remove unreachable state.
3. Then write right linear grammar using the following steps
   i. Repeat the process for every state.
   ii. Begin the process from start state.
   iii. Write the production as the output followed by the state on which the transition is going.
   iv. And at the last add ε because that's required to end the derivation.
4. Then take reverse of the right linear grammar
5. And you will get the final left linear grammar

✓ **Problems for Finite Automata to Left Linear Regular Grammar:**

1. Construct Left Linear Grammar from the given Finite Automata



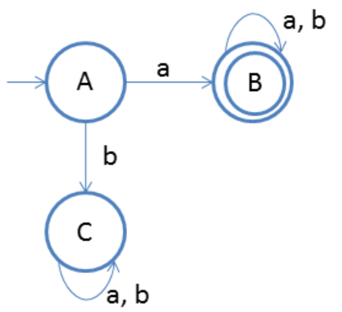
1) Take reverse of the finite automata (make final state as initial state and vice-versa)



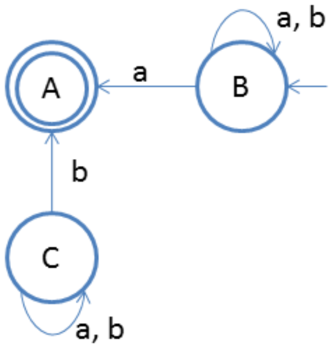
2) Remove unreachable state.
   There is no unreachable state

3) Then write right linear grammar
   a. Pick start state and output is on symbol 'a' we are going on state A and B. So, we will write as :
      B → aA / aB
   b. Pick start state and output is on symbol 'b' we are going on state B. So, we will write as :
      B → bB
   c. Then we will pick state A and then we will go for each output. So, we will get the below production.
      A→ ε
   d. So, final we got right linear grammar as:
      B→ aA / aB / bB
      A → ε
4) Then take reverse of the right linear grammar
   B→ Aa / Ba / Bb
   A → ε
5) Final left linear grammar
   B→ Aa / Ba / Bb
   A → ε

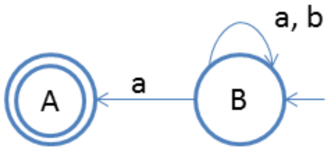
2. Construct Left Linear Grammar from the given Finite Automata



1) Take reverse of the finite automata (make final state as initial state and vice-versa)



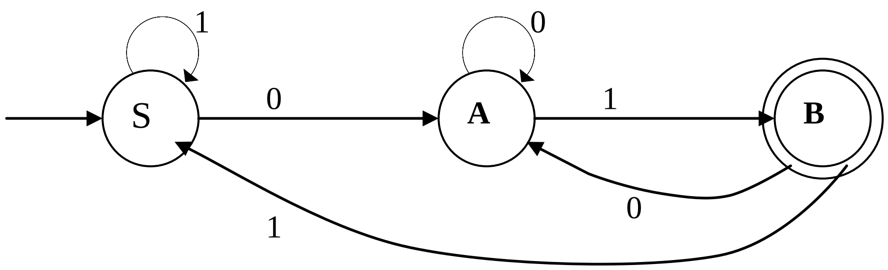
2) Remove unreachable state.
   State C is unreachable state, So remove state from the above FA



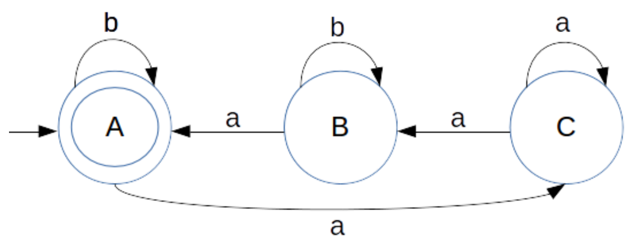
3) Then write right linear grammar
   a. Pick start state and output is on symbol 'a' we are going on state A and B. So, we will write as :
      B → aA / aB
   b. Pick start state and output is on symbol 'b' we are going on state B. So, we will write as :
      B → bB
   c. Then we will pick state A and then we will go for each output. So, we will get the below production.
      A→ ε
   d. So, final we got right linear grammar as:
      B→ aA / aB / bB
      A → ε
4) Then take reverse of the right linear grammar
   B→ Aa / Ba / Bb
   A → ε
5) Final left linear grammar
   B→ Aa / Ba / Bb
   A → ε

Tutorial Questions:

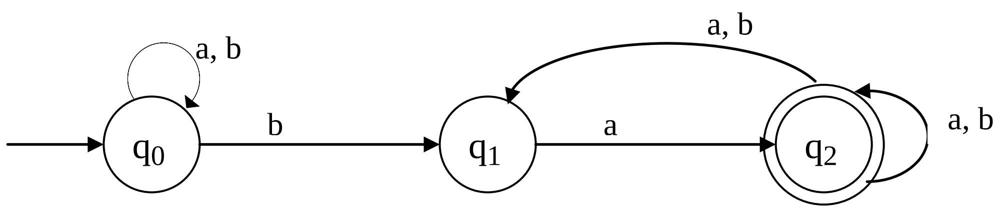
3. Construct Left Linear Grammar from the given Finite Automata



4. Construct Left Linear Grammar from the given Finite Automata



5. Construct Left Linear Grammar from the given Finite Automata



✓ **Conversion of Left Linear Regular Grammar to Finite Automata**

Algorithm:

Given a regular grammar G, a finite automata accepting L(G) can be obtained as follows:

1. Take reverse of CFG
2. The number of states in the automata will be equal to the number of non-terminals plus one. Each state in automata represents each non-terminal in the regular grammar. The additional state will be the final state of the automata. The state corresponding to the start symbol of the grammar will be the initial state of automata. If L(G) contains ϵ that is start symbol is grammar devices to ϵ, then make start state also as final state.
3. The transitions for automata are obtained as follows:
   - For every production A → aB, then make δ(A, a) = B that is make an are labeled 'a' from A to B.
   - For every production A → a, then make δ(A, a) = final state.
   - For every production A → ϵ, then make δ(A, ϵ) = A and A will be final state.
4. Then again take reverse of the FA and that will be our final output
5. Start State: It will be the first production's state
6. Final State: Take those states which end up with input alphabets.

✓ **Problems for Finite Automata to Left Linear Regular Grammar**

1. Construct a Finite Automata from the given Left Linear Grammar

A → Ba/Ab/b
B → Ca/Bb
C → Aa/Cb/a

Solution:

Step 1: Take reverse of CFG

A → aB/bA/b
B → aC/bB
C → aA/bC/a

Step 2: Take the 'A' productions, then will make transition functions

A → aB → δ(A, a) = B
A → bA → δ(A, b) = A
A → b → δ(A, b) = Final State

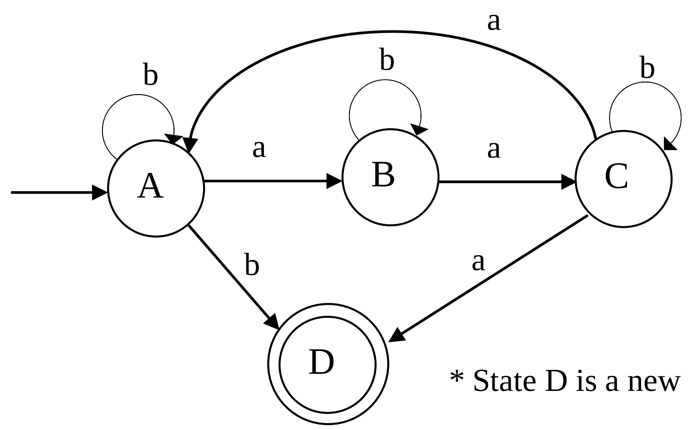
Step 3: Take the 'B' productions, then will make transition functions

B → aC → δ(B, a) = C
B → bB → δ(B, b) = B

Step 4: Take the 'C' productions, then will make transition functions

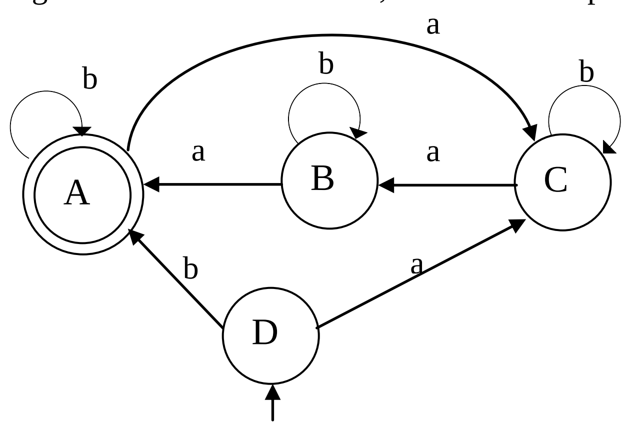
C → aA → δ(C, a) = A
C → bC → δ(C, b) = C
C → b → δ(C, b) = Final State

Step 5: Construct Finite Automata



* State D is a new final State

Step 6: Again take reverse of the FA, this is final output.



2. Construct a Finite Automata from the given Left Linear Grammar

S → A / B / ε
A → S0/B1/0
B → S0/A1/1

Solution:

Step 1: Take reverse of CFG

S → A / B / ε
A → 0S/1B/0
B → 0S/1A/1

Step 2: Take the 'S' productions, then will make transition functions

S → A → δ(S, ε) = A
S → B → δ(S, ε) = B
S → ε → δ(S, ε) = S and S is make Final State

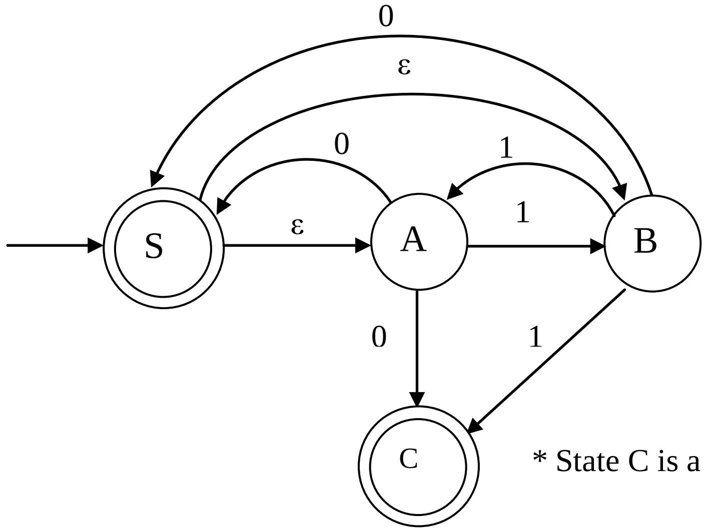
Step 3: Take the 'A' productions, then will make transition functions

A → 0S → δ(A, 0) = S
A → 1B → δ(A, 1) = B
A → 0 → δ(A, 0) = Final State

Step 4: Take the 'B' productions, then will make transition functions

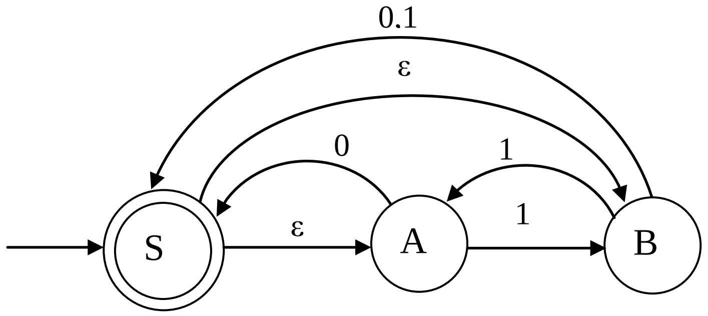
B → 0S → δ(B, 0) = S
B → 1A → δ(B, 1) = A
B → 1 → δ(B, 1) = Final State

Step 5: Construct Finite Automata

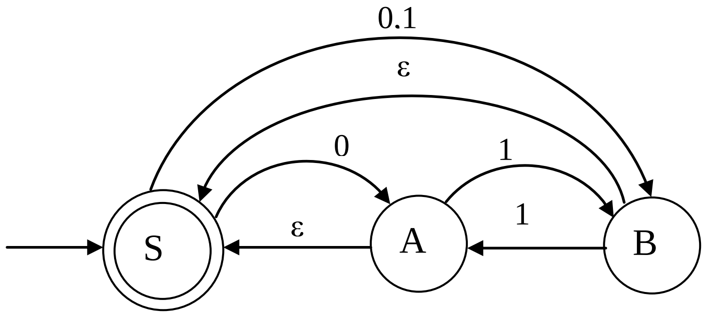


* State C is a new final State

Step 6: Reconstructed Finite Automata (remove state C)



Step 7: Again take reverse of the FA, this is final output.



Tutorial Questions:

3. Construct a Finite Automata from the given Left Linear Grammar
   S → Aab / Aba / B / ε
   A→ Sb / b
   B → Sa
   C → ε
4. Construct a Finite Automata from the given Left Linear Grammar
   A → Ba
   B → Ba/Bb/ε
5. Give the Finite Automata from the given Left Linear Grammar
   S → S0/A1/1/B0/0
   A → A0/B1/0/1
   B → B0/A1/0/1

Context free Grammars

✓ **Motivation and introduction**

- A Context Free Grammar is a "machine" that creates a language.
- A language created by a CF grammar is called A Context Free Language.
- The class of Context Free Languages Properly Contains the class of Regular Languages.

✓ **Definition:**

A Context Free Grammar is consists of four components. They are finite set of non-terminals, finite set of terminals, set of productions and start symbol.

✓ **Formal Definition of Context Free Grammars (CFG)**

- A CFG is a mathematical object, G, with four components,
  G = (N, T, P, S)
  Where
  N is a nonempty, finite set of non-terminal symbols
  T is a finite set of terminal symbols
  P is a set of grammar rules, each of one having one of the forms
  $A \rightarrow \alpha$
  Where $A \in N$ and $\alpha \in (N \cup T)^*$
  S is the start symbol $S \in N$
- Example
  Let G = ({S},{0,1,ε},P,S) be a CFG, where productions are S→ 0S0/1S1/ε

✓ **Context Free Language: The Language Generated by a Regular Grammar**

- Let $G = (N, T, P, S)$ be a regular grammar. We define the *language generated by G* to be L(*G*).
- $L(G) = \{w \mid$ w can be derived from G (or) $S \stackrel{*}{\Rightarrow} w$, where $w \in T^*\}$

✓ **<u>Conversion of Context Free Language (CFL) into Context Free Grammar (CFG)</u>**

1. Construct a CFG representing the set of palindromes over (0+1)*.
   The possible strings are
   {ε,0,1,00,11,000,111,010,101,0000,1111,00100,11011, 01110,10101,....}
   The CFG for a palindrome is given by
   S → 0 / 1 / ε
   S → 0S0 / 1S1

2. Construct a CFG for the language L = $\{a^n$; n is odd$\}$.
   The possible strings are {ε, a, aaa, aaaaa, aaaaaaa, ...}
   The productions are
   G: S → a / aaS

3. Construct a CFG for the language L = $\{a^n b^n ; n \geq 0\}$.
   The possible strings are {ε, ab, aabb, aaabbb, aaaabbbb, ...}
   The productions are
   G: S → ab / aSb / ε

4. Construct a CFG for the language L = $\{0^n1^n ; n \geq 1\}$.
   The possible strings are {01, 0011, 000111, 00001111, ...}
   The productions are
   G: S → 01 / 0S1

5. Construct a CFG for the language L = $\{a^ncb^n ; n \geq 0\}$.
   The possible strings are {c, acb, aacbb, aaacbbb, aaaacbbbb, ...}
   The productions are
   G: S → c / aSb

6. Construct a CFG for the language L = $\{wcw^r ; w \in (a+b)^*\}$.
   The possible strings are {c, aca, bcb, abcba, aacaa, bbcbb, bacab, abacaba, bbacabba,….}
   The productions are
   G: S → aSa / bSb / c

7. Construct a CFG for the language L = $\{aab(bba)^n bab(aab)^n ; n \geq 0\}$.
   The possible strings are {aabbab, aabbbababaab, aabbbabbababaabaab,….}
   The productions are
   G: S → ABCD
   A → aab
   B → bba / bbaB
   C → bab
   D → aab / aabD

Tutorial Questions:

8. Construct a CFG for the language L = $\{a^nbc^m ; n, m \geq 0\}$.
9. Construct a CFG for the language L = $\{0^n1011^n ; n \geq 1\}$.
10. Construct a CFG for the language L = $\{1^n0^m ; n \geq 0, m = n+2\}$.

✓ **<u>Conversion of Context Free Grammar (CFG) into Context Free Language (CFL)</u>**

1. Construct a CFL from the given grammar
   G = ({S}, {0,1, ε} , P, S)
   Where
   S → 0 / 1 / ε
   S → 0S0 / 1S1
   Solution:
   If String Length = 1, The Strings are ε, 0, 1
   If String Length = 2, The Strings are 00, 11
   If String Length = 3, The Strings are 000, 111, 010, 101
   If String Length = 4, The Strings are 0000,1111
   If String Length = 5, The Strings are 00000,11111, 01010, 10101, 11011, 00100, 01110, 10001
   .....
   If String Length > 5, The Strings are 0000 … .001111 … 11
   So, The CFL is
   L = { w; All strings are palindrome over Σ{0,1}}

2. Construct a CFL from the given grammar

   G = ({S}, {0,1, ε} , P, S)

   Where

   S → a / aaS

   Solution:

   If String Length = 1, The String is a
   If String Length = 2, The String is aaa
   If String Length = 3, The String is aaaaa
   If String Length = 4, The String is aaaaaaa
   .....
   If String Length > n, The String is aaa......aaaa, n is odd
   So, The CFL is
   L = {a^n ; n is odd}.

3. Construct a CFL from the given grammar

   G = ({S}, {a, b, c} , P, S)

   Where

   S → aSa / bSb / c

   Solution:

   If String Length = 1, The String is c
   If String Length = 3, The Strings are aca, bcb
   If String Length = 5, The Strings are aacaa, bbcbb, abcba, bacab
   .....
   If String Length > n, The Strings are aaa...c...aaa, bb...c...bb, aba..c..aba, bba....c...bba, ...
   So, The CFL is
   L = {wcw^r ; w ∈ (a+b)*}.

Tutorial Questions:

4. Construct a the CFL from the following grammar

   S → c / aSb

5. Construct a the CFL from the following grammar

   S → ABCD
   A → aab
   B → bba / bbaB
   C → bab
   D → aab / aabD

6. Construct a the CFL from the grammar G = ({S},{a,b},P,S)}, with productions

   S → aSa,
   S → bSb,
   S → ε

Derivations

- ✓ A derivation of a string for a grammar is a sequence of grammar rule applications that transform the start symbol into the string. A derivation proves that the string belongs to the grammar's language. ie. $\boldsymbol{S \overset{*}{\Rightarrow} w}$**, where** $\boldsymbol{w \in T^*}$ **and** $\mathbf{w \in L(G)}$
- ✓ A derivation is fully determined by giving, for each step:
  - o The rule applied in that step
  - o The occurrence of its left-hand side to which it is applied
- ✓ Example

  Consider G whose productions are S → aAS / a, A→ SbA / SS / ba,
  show that S ⇒ aabbaa.

  Solution:

  S ⇒ aAs
  ⇒ aSbAs [A → SbA]
  ⇒ aabAS [S → a]
  ⇒ aabbaS [A → ba]
  ⇒ aabbaa [S → a]
  $S \overset{*}{\Rightarrow}$ aabbaa

Leftmost derivation (LMD)

- ✓ A leftmost derivation is obtained by applying production to the leftmost variable or non-terminal in each step.

  ie. $\boldsymbol{S \underset{lm}{\overset{*}{\Rightarrow}} w}$**, where** $\boldsymbol{w \in T^*}$ **and** $\mathbf{w \in L(G)}$

- ✓ Problems for LMD

1. Consider G whose productions are S → aAS / a, A→ SbA / SS / ba,
   Show that S ⇒ aabbaa.

   Solution:

   S ⇒ a**A**S
   ⇒ a<u>**S**bA</u>S [A→ SbA]
   ⇒ a<u>a</u>b**A**S [S→ a]
   ⇒ aab<u>ba</u>**S** [A→ ba]
   ⇒ aabba<u>a</u> [S→ a]

   $\mathbf{S \underset{lm}{\overset{*}{\Rightarrow}}}$ **aabbaa**

2. Find a left most derivation for "aaabbabbba" with the productions.
   P : S → aB / bA, A → a /S / bAA, B → b / bS / aBB

Solution:

S ⇒ a**B**
⇒ a<u>a**B**B</u> [B→ aBB]
⇒ aa<u>a**B**B</u>B [B→ aBB]
⇒ aaa<u>b</u>**B**B [B→ b]
⇒ aaab<u>b</u>**B** [B→ b]
⇒ aaabb<u>a**B**B</u> [B→ aBB]
⇒ aaabba<u>b</u>**B** [B→ b]
⇒ aaabbab<u>b</u>**S** [B→ bS]
⇒ aaabbabb<u>b**A**</u> [S→ bA]
⇒ aaabbabbb<u>a</u> [A→ a]

$$\mathbf{S} \overset{*}{\underset{lm}{\Rightarrow}} \mathbf{aaabbabbba}$$

Rightmost derivation

- ✓ A rightmost derivation is obtained by applying production to the rightmost variable or non-terminal in each step.

  ie. $S \overset{*}{\underset{rm}{\Rightarrow}} w$, **where** $w \in T^*$ **and** $w \in L(G)$

- ✓ Problems for RMD

1. Consider G whose productions are S → aAS / a, A→ SbA / SS / ba,
   Show that S ⇒ aabbaa.

   Solution:

S ⇒ aA**S**
⇒ a**A**<u>a</u> [S→ a]
⇒ a<u>Sb**A**</u>a [S→ SbA]
⇒ a**S**b<u>ba</u>a [A→ ba]
⇒ aabbaa [S→ a]

$$\mathbf{S} \overset{*}{\underset{rm}{\Rightarrow}} \mathbf{aabbaa}$$

2. Find a right most derivation for "aaabbabbba" with the productions.
   P : S → aB / bA, A → a /S / bAA, B → b / bS / aBB
   Solution:

S ⇒ a**B**	
⇒ aaB**B**	[B→ aBB]
⇒ aaBb**S**	[B→ bS]
⇒ aaBbb**A**	[S→ bA]
⇒ aa**B**bba	[A→ a]
⇒ aaaB**B**bba	[B→ aBB]
⇒ aaa**B**bbba	[B→ b]
⇒ aaab**S**bbba	[B→ bS]
⇒ aaabb**A**bbba	[S→ bA]
⇒ aaabbabbba	[A→ a]

S $\underset{rm}{\overset{*}{\Rightarrow}}$ **aaabbabbba**

- ✓ **Sentential Form or Partial Derivation**
  - A partial derivation is a part of a derivation. The strings are derived from the start symbol is called as Sentential form.
  - If G =(V,T,P,S) is a CFG, then α (V ∪ T)*

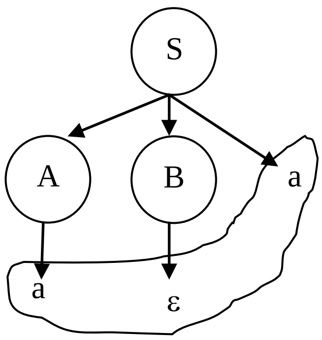
$S \underset{G}{\overset{*}{\Rightarrow}} \alpha$, **where** $\alpha \in (V \cup T)^*$ - **Sentential Form**

$S \underset{lm}{\overset{*}{\Rightarrow}} \alpha$, **where** $\alpha \in (V \cup T)^*$ - **Left Sentential Form**

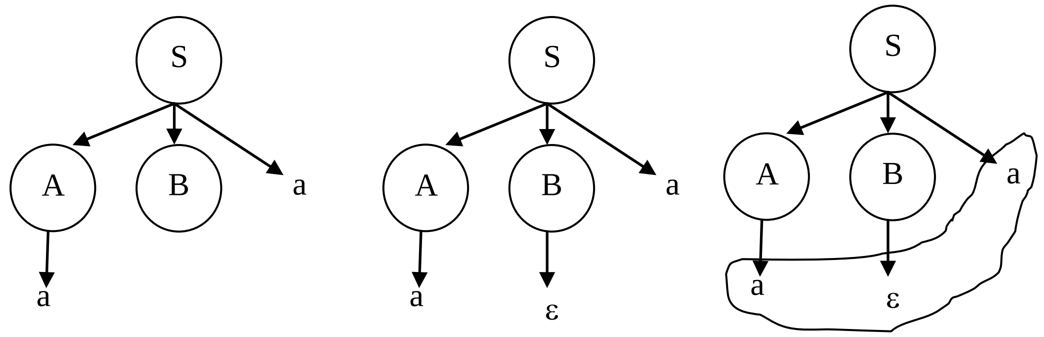
$S \underset{rm}{\overset{*}{\Rightarrow}} \alpha$, **where** $\alpha \in (V \cup T)^*$ - **Right Sentential Form**

Derivation Tree or Parse Tree - (Pictorial representation of derivation)

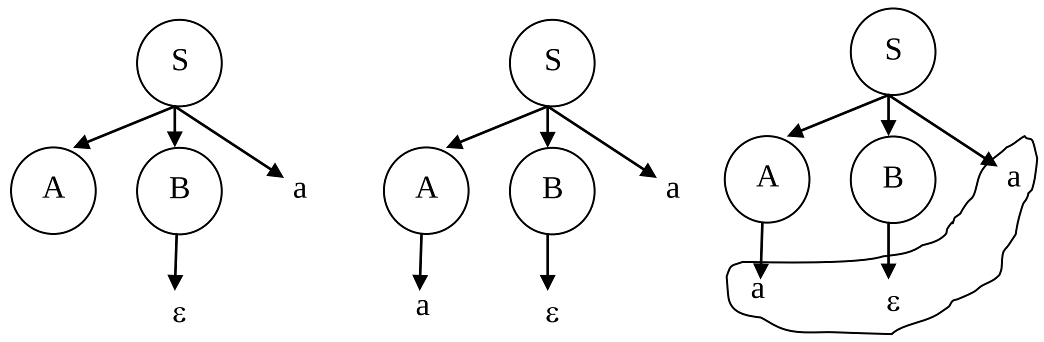
- ✓ A derivation tree or parse tree is an ordered rooted tree that graphically represents the semantic information a string derived from a context-free grammar.
- ✓ Representation Technique
  - Root vertex − Must be labelled by the start symbol.
  - Vertex − Labelled by a non-terminal symbol.
  - Leaves − Labelled by a terminal symbol or ε.



- **Types of Derivation Tree**
  - Leftmost derivation tree
    - A leftmost derivation tree is obtained by applying production to the leftmost vertex in each step.
    - Example: S → ABa, A → a, B → ε



  - Rightmost derivation tree
    - A rightmost derivation tree is obtained by applying production to the rightmost vertex in each step.
    - Example: S → ABa, A → a, B → ε



Ambiguity

- ✓ If a context free grammar G has more than one derivation tree (leftmost or rightmost derivation tree) for some string w∈L(G), it is called an ambiguous grammar. There exist multiple right-most or left-most derivations for some string generated from that grammar.

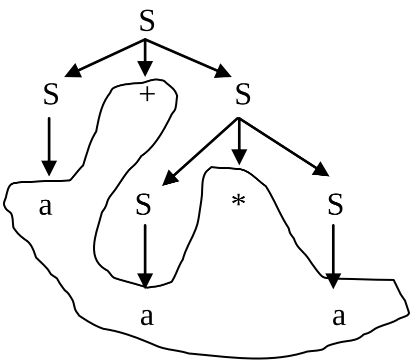
- ✓ **Problems for Ambiguity in Context-Free Grammars**

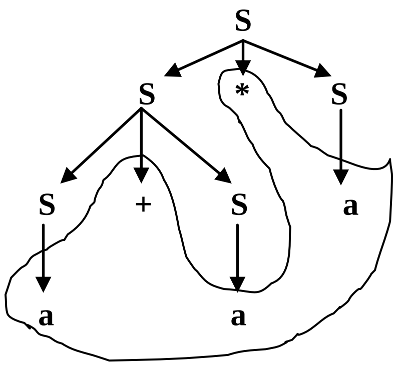
1. Check whether the grammar G with production rules S → S+S / S*S / S / a is ambiguous or not.

   Solution:

   Let's assume a string w = a+a*a

   Parse Tree 1 : Parse Tree 2 :





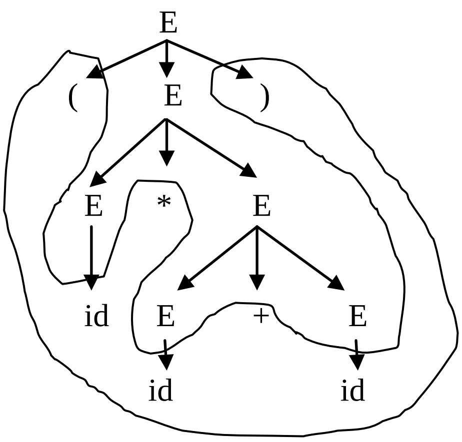
   Thus we have two parse trees, So the given grammar is ambiguous.

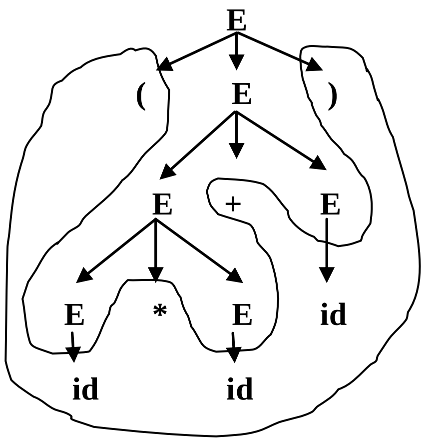
2. Check whether the grammar G with production rules S → E+E / E*E / (E) / id is ambiguous or not.

   Solution:

   Let's assume a string w = (id*id+id)

   Parse Tree 1 : Parse Tree 2 :





   Thus we have two parse trees, so the given grammar is ambiguous.

Tutorial Questions:

1. Show that the grammar defined by the productions
   S → SS / a /b is ambiguous.

2. If G is the grammar S → SbS / a, Show that G is ambiguous.

3. Prove that the grammar defined by the productions
   S → A1B, A → 0A / ε, B → 0B / 1B / ε is unambiguous.

4. Let the production of the grammar be S → 0B / 1A, A → 0 / 0S / 1AA,
   B → 1 / 1S / 0BB and the string 0110.
   a. Find the left most derivation and associated derivation tree.
   b. Find the right most derivation and associated derivation tree.
   c. Find the G is ambiguous or not.
   d. Find a L(G).

5. G denotes the context-free grammar defined by the following rules.
   S→ASB / ab / SS
   A→aA / A
   B→bB / A
   a. Give a left most derivation of “aaabb” in G. Draw the associated parse tree.
   b. Give a right most derivation of “aaabb” in G. Draw the associated parse tree.
   c. Show that G is ambiguous.
   d. Find a L(G).

Simplification of CFG’s

- ✓ In a CFG, it may happen that all the production rules and symbols are not needed for the derivation of strings. Besides, there may be some null productions, useless symbols and unit productions. Elimination of these productions and symbols is called simplification of CFGs.
- ✓ Simplification essentially comprises of the following steps
  - Elimination of Useless Symbols or Productions
  - Elimination of Null Productions (ie. ε)
  - Elimination of Unit Productions

- **Elimination of Useless Symbols or Productions**
  - The productions that can never take part in derivation of any string are called useless productions. Similarly, a symbol that can never take part in derivation of any string is called a useless symbol or variable.
  - Example

1. Eliminate the useless symbols or productions from the given grammar

G: S → abS / abA / abB
A → cd
B → aB
C → dc

Solution:

Step 1:

The production 'B →aB' is useless because there is no way it will ever terminate. If it never terminates, then it can never produce a string, then remove all the productions in which variable 'B' occurs.

After eliminating B production and B symbols:

G1: S → abS / abA
A → cd
C → dc

Step 2:

The production 'C → dc' is useless because the variable 'C' will never occur in derivation of any string, then remove all the productions in which variable 'C' occurs.

After eliminating C production:

G2: S → abS / abA
A → cd

Step 3: Resultant Grammar

G': S → abS / abA

A → cd

Tutorial Questions:

2. Eliminate the useless symbols or productions from the given grammar

S → AC / B, A → a, C → c / BC, E → aA / ε

3. Remove the useless symbol from the given context free grammar:

S → aB / bX
A → Bad / bSX / a
B → aSB / bBX
X → SBD / aBx / ad

✓ **Elimination of Null Productions (ie. ε)**

- The productions A → ε are called ε productions (also null productions). These productions can only be removed from those grammars that do not generate ε (an empty string).
- To remove null productions, we first have to find all the nullable variables. A variable A is called nullable if ε can be derived from A.
  - For all the productions A→ ε , A is a nullable variable.
  - For all the productions of type B → A1A2…An, where all 'Ai's are nullable variables, B is also a nullable variable.
- If all the variables on the RHS of the production are nullable , then we do not add A → ε to the new grammar.
- Example:

1. Eliminate the ε productions from the given grammar

G: S → ABCd
A → BC
B → bB / ε
C → cC / ε

Solution:

Step 1: Remove the productions B→ε and C→ε

G: S → ABCd / ACd / ABd / Ad
A → BC / C / B / ε
B → bB / b
C → cC / c

Step 2: Remove the production A→ε

G: S → ABCd / ACd / ABd / Ad / BCd / Cd / Bd / d
A → BC / C / B
B → bB / b
C → cC / c

Step 2: Resultant Grammar

G': S → ABCd / ACd / ABd / Ad / BCd / Cd / Bd / d
A → BC / C / B
B → bB / b
C → cC / c

Tutorial Questions:

2. Eliminate the ε productions from the given grammar

   S → ABAC
   A → aA / ε
   B → bB / ε
   C → c

3. Remove the ε productions from the given grammar

   S → ASA / aB / b, A → B, B → b / ε

✓ **Elimination of Unit Productions**

- Any production rules in the form A → B where A, B ∈ Non-terminal is called unit production.
- Steps for eliminate unit productions:
  - Step 1: To remove A → B, add production A → x to the grammar rule whenever B → x occurs in the grammar. [x ∈ Terminal, x can be Null]
  - Step 2: Delete A → B from the grammar.
  - Step 3: Repeat from step 1 until all unit productions are removed.

- Example

1. Eliminate the unit production from the given grammar

   G: S → Aa / B
   A →b / B
   B → A / a

   Solution:

   Step 1: Remove the production B→ A

   G: S → Aa / B
   A →b / A / a
   B → A / a

   Step 2: Remove the production A→A

   G: S → Aa / B
   A →b / a
   B → A / a

   Step 3: Remove the production B→A

   G: S → Aa / B
   A → b / a
   B → b / a

Step 4: Remove the production S→B

G: S → Aa / b / a
A → b / a
B → b / a

Step 4: Resultant Grammar

G': S → Aa / b / a
A → b / a
B → b / a

Tutorial Questions:

2. Eliminate the useless symbols or productions from the given grammar

   S → XY, X → a, Y → Z / b, Z → M, M → N, N → a

3. Remove the useless symbol from the given context free grammar:

   S → AB
   A → a
   B → C / b
   C → D
   D → E
   E → a

4. Consider the grammar

   S→ 0A0 / 1B1 / BB
   A→ C
   B→ S / A
   C→ S / ε and simplify using the same order

   a. Eliminate ε-Productions
   b. Eliminate unit productions
   c. Eliminate useless symbols

Normal Form

- ✓ A CFG is convert into a specific form is called as Normal forms.
- ✓ There are two types of Normal Norms.
  - Chomsky Normal Form (CNF)
  - Greibach Normal Form (GNF)

Chomsky Normal Form (CNF)

✓ A CFG is said to be in Chomsky Normal Form if every production is of one of these two forms:

1. Non-Terminal → Non-Terminal . Non-Terminal

   Example: A → BC where A,B,C∈V (right side is two Non-Terminal).

2. Non-Terminal → Terminal

   Example: A → a where a ∈ T (right side is a single Terminal).

✓ Algorithms for converting CFG into CNF:

Step 1: Eliminate Null productions.

Step 2: Eliminate Unit productions.

Step 3: Eliminate Useless Symbols or Productions.

Step 4: Replace each production $A \rightarrow B_1 \ldots B_n$ where n > 2 with $A \rightarrow B_1C$. Where $C \rightarrow B_2 \ldots B_n$. Repeat this step for all productions having more than two non-terminals in the right side.

Step 5: If the right side of any production is in the form A → aB where a is a terminal and A, B are non-terminal, then the production is replaced by A → XB and X → a. Repeat this step for every production which is in the form A → aB.

✓ Problems for converting CGF into CNF:

1. Consider the Grammar G = ({S,A,B},{a,b}, P, S} as the productions

   S → bA / aB
   A → bAA / aS / a
   B → aBB / bS / b.
   Convert it into CNF.

   Solution:

   Step 1: Eliminate Null productions.
   There is no Null production.
   Step 2: Eliminate Unit productions.
   There is no Unit production.
   Step 3: Eliminate Useless Symbols or Productions.
   There is no Useless Symbols or Productions.
   Step 4: Find the productions which are already in CNF.
   A → a
   B → b
   Step 5: Replace all remaining productions into CNF.
   Non-Terminal → Non-Terminal . Non-Terminal
   Non-Terminal → Terminal

   i) S → bA
   $S \rightarrow C_bA$
   $C_b \rightarrow b$

ii) $S \rightarrow aB$

$S \rightarrow C_aB$

$C_a \rightarrow a$

iii) A $\rightarrow bAA$

$A \rightarrow C_bD_1$

$D_1 \rightarrow AA$

$C_b \rightarrow b$

iv) A $\rightarrow aS$

$A \rightarrow C_aS$

$C_a \rightarrow a$

v) B $\rightarrow aBB$

$B \rightarrow C_a D_2$

$D_2 \rightarrow BB$

$C_a \rightarrow a$

v) B $\rightarrow bS$

$B \rightarrow C_bS$

$C_b \rightarrow b$

Step 3: Final Resultant Grammar

G: $S \rightarrow C_bA$ / C_aB

$A \rightarrow C_bD_1$ / C_aS / a

$B \rightarrow C_a D_2$ / C_bS / b

$D_1 \rightarrow AA$

$D_2 \rightarrow BB$

$C_a \rightarrow a$

$C_b \rightarrow b$

2. Convert the given grammar into CNF.

G = ({S,A,B},{a,b}, P, S}

The Productions are

S→ 0A0 / 1B1 / BB

A→ C

B→ S / A

C→ S / ε.

Solution:

Step 1: Eliminate ε-Productions

1.1 Remove the production C→ ε

S→ 0A0 / 1B1 / BB

A→ S / ε

B→ S / A

C→ S

1.2 Remove the production A→ ε

S→ 0A0 / 00 / 1B1 / BB

A→ S

B→ S / ε

C→ S

1.3 Remove the production B→ ε

S→ 0A0 / 00 / 1B1 / 11 / BB / B
A→ S
B→ S
C→ S

Step 2: Eliminate Unit productions.

2.1 Remove the production C→ S

S→ 0A0 / 00 / 1B1 / 11 / BB / B
A→ S
B→ S

2.2 Remove the production B→ S

S→ 0A0 / 00 / 1S1 / 11 / SS / S
A→ S

2.3 Remove the production A→ S

S→ 0S0 / 00 / 1S1 / 11 / SS / S

2.4 Remove the production S→ S

S→ 0S0 / 00 / 1S1 / 11 / SS

Step 3: Eliminate useless symbols

There is no Unit production.
Resultant Grammar (after simplifications)
G' : S→ 0S0 / 00 / 1S1 / 11 / SS

Step 4: Find the productions which are already in CNF.

S→ SS

Step 5: Replace all productions into CNF.

Non-Terminal → Non-Terminal . Non-Terminal
Non-Terminal → Terminal

i) S→ 0S0

S → AB
B→ SA
A→ 0

ii) S→ 00

S → AA
A → 0

iii) S→ 1S1

S → DC
C → SD
D → 1

iv) S→ 11

S → DD
D → 1

Step 5: Resultant Grammar

G’: **S → AB / AA / DC / DD**
B→ SA
A→ 0
C → SD
D → 1

Tutorial Questions:

3. Convert the following CFG to CNF
   S → ASA / aB
   A → B / S
   B → b / ε
4. Convert the following CFG to CNF
   S → AB / Aa
   A→ aAA / a
   B→ bBB / b
5. Find a grammar in Chomsky Normal form equivalent to
   S→aAD
   A→aB / bAB
   B→b
   D→d
6. Consider G = ({S,A}, {a,b}, P, S} where P consists of
   S→aAS / a
   A→ SbA / SS / ba
   Convert it to its equivalent CNF

<u>Greibach Normal Form (GNF)</u>

- ✓ A CFG is said to be in Greibach Normal Form if every production is of one of these two forms:
  1. Non-Terminal → Terminal . Any no. of Non-Terminal

     Example: A → aBC or
  2. Non-Terminal → Terminal

     Example: A → a (right side is a single Terminal).

     (Or)

     A → aα , where a∈T and α∈V*

- ✓ <u>Algorithms for converting CFG into GNF:</u>

  Step 1: Eliminate Null productions.

  Step 2: Eliminate Unit productions.

  Step 3: Eliminate Useless Symbols or Productions.

  Step 4: Check whether the CFG is already in CNF and convert it to CNF if it is not.

Step 5: Rename the variables like A_1, A_2, ...An starting with S = A_1. (A_i in ascending order of i)

Step 6: No need to modify the productions like $A_i \rightarrow A_j\gamma$ where i < j

Step 7: Modify the productions like $A_i \rightarrow A_j\gamma$ where $i \geq j$

(a) If $A_i \rightarrow A_j\, \gamma$ where i > j, then substitute for A_j productions.
Suppose $A_j \rightarrow A_k / A_L$, then the new set of productions are
$A_i \rightarrow A_k\, \gamma / A_L\gamma$

(b) It $A_i \rightarrow A_j\, \gamma$ where i = j, then do the following steps:
Introduce a new variable B_i
Then
$B_i \rightarrow A_k$
$B_i \rightarrow \gamma\, B_i$
and remove the production $A_i \rightarrow A_j\, \gamma$

(c) For each production $A_i \rightarrow \beta$ where β does not begin with A_i , then add the production
$A_i \rightarrow \beta\, B_i$

Step 7: Convert all the productions into GNF form. $A \rightarrow a\alpha$ where $a \in T$ and $\alpha \in V^*$

✓ <u>Problems for converting CFG into GNF:</u>

1. Consider the Grammar G = ({S,A,B},{a,b}, P, S} as the productions

$S \rightarrow AB$
$A \rightarrow BS / b$
$B \rightarrow SA / a$
Convert it into CNF.

<u>Solution:</u>

Step 1: Eliminate Null productions.
There is no Null production.
Step 2: Eliminate Unit productions.
There is no Unit production.
Step 3: Eliminate Useless Symbols or Productions.
There is no Useless Symbols or Productions.
Step 4: All production rules are already in CNF form.
Step 5: Rename the variables S, A, B as A_1, A_2, A_3 respectively.

$A_1 \rightarrow A_2\, A_3$ ------ (1)
$A_2 \rightarrow A_3\, A_1 / b$ ------ (2)
$A_3 \rightarrow A_1\, A_2 / a$ ------ (3)

In (1), i < j, no need to modify the production.
In (2), i < j, no need to modify the production.
In (3), i > j, substitute A_1 productions in (3)
$A_3 \rightarrow A_2\, A_3\, A_2 / a$ -------(4)

In (4), i > j, substitute A_2 productions in (4)
$A_3 \rightarrow A_3\, A_1\, A_3\, A_2 / b\, A_3\, A_2 / a$ -------(5)

In (5), i = j, introduce new non-terminal B_3, then B_3 productions are

$B_3 \rightarrow A_1\ A_3\ A_2 \ /\ A_1\ A_3\ A_2\ B_3$ and

$A_3 \rightarrow b\ A_3\ A_2 \ /\ a$ has been modified to

$A_3 \rightarrow b\ A_3\ A_2 \ /\ a \ /\ b\ A_3\ A_2\ B_3 \ /\ a\ B_3$

Step 6: Resultant productions are

$A_1 \rightarrow A_2\ A_3$ -------- (1)

$A_2 \rightarrow A_3\ A_1 \ /\ b$ -------- (2)

$B_3 \rightarrow A_1\ A_3\ A_2 \ /\ A_1\ A_3\ A_2\ B_3$ -------- (3)

$\mathbf{A_3 \rightarrow b\ A_3\ A_2 \ /\ a \ /\ b\ A_3\ A_2\ B_3 \ /\ a\ B_3}$ **-------- (4)**

Step 7: Convert into GNF form

Non-Terminal = Terminal .any no. of Non-Terminals

Non-Terminal = Terminal

Substitute A_2 in (1)

$A_1 \rightarrow A_3\ A_1\ A_3 \ /\ b\ A_3$ -------- (5)

Substitute A_3 in (5)

$\mathbf{A_1 \rightarrow b\ A_3\ A_2\ A_1\ A_3 \ /\ a\ A_1\ A_3 \ /\ b\ A_3\ A_2\ B_3\ A_1\ A_3 \ /}$
$\mathbf{a\ B_3\ A_1\ A_3 \ /\ b\ A_3}$

Substitute A_3 in (2)

$\mathbf{A_2 \rightarrow b\ A_3\ A_2\ A_3\ A_1 \ /\ a\ A_3\ A_1 \ /\ b\ A_3\ A_2\ B_3\ A_3\ A_1 \ /}$
$\mathbf{a\ B_3\ A_3\ A_1 \ /\ b}$

Substitute A_1 in (3)

$\mathbf{B_3 \rightarrow b\ A_3\ A_2\ A_3\ A_2 \ /\ a\ A_3\ A_2 \ /\ b\ A_3\ A_2\ B_3\ A_3\ A_2 \ /\ a\ B_3\ A_3\ A_2 \ /}$
$\mathbf{b\ A_3\ A_2\ A_3\ A_2\ B_3 \ /\ a\ A_3\ A_2\ B_3 \ /\ b\ A_3\ A_2\ B_3\ A_3\ A_2\ B_3 \ /}$
$\mathbf{a\ A_3\ A_2\ B_3 B_3}$

Step 8: The equivalent GNF productions are

$\mathbf{A_1 \rightarrow b\ A_3\ A_2\ A_1\ A_3 \ /\ a\ A_1\ A_3 \ /\ b\ A_3\ A_2\ B_3\ A_1\ A_3 \ /\ a\ B_3\ A_1\ A_3 \ /\ b\ A_3}$
$\mathbf{A_2 \rightarrow b\ A_3\ A_2\ A_3\ A_1 \ /\ a\ A_3\ A_1 \ /\ b\ A_3\ A_2\ B_3\ A_3\ A_1 \ /\ a\ B_3\ A_3\ A_1 \ /\ b}$
$\mathbf{A_3 \rightarrow b\ A_3\ A_2 \ /\ a \ /\ b\ A_3\ A_2\ B_3 \ /\ a\ B_3}$
$\mathbf{B_3 \rightarrow b\ A_3\ A_2\ A_3\ A_2 \ /\ a\ A_3\ A_2 \ /\ b\ A_3\ A_2\ B_3\ A_3\ A_2 \ /\ a\ B_3\ A_3\ A_2}$
$\mathbf{B_3 \rightarrow b\ A_3\ A_2\ A_3\ A_2\ B_3 \ /\ a\ A_3\ A_2\ B_3 \ /\ b\ A_3\ A_2\ B_3\ A_3\ A_2\ B_3}$
$\mathbf{B_3 \rightarrow a\ A_3\ A_2\ B_3 B_3}$

Tutorial Questions:

2. Convert the following CFG to GNF

   $S \rightarrow AA \ /\ a$
   $A \rightarrow SS \ /\ b$
   (or)

   Convert the following CFG to GNF

   $A_1 \rightarrow A_2A_2 \ /\ a$
   $A_2 \rightarrow A_1A_1 \ /\ b$

3. Convert the following CFG to GNF

   $S \rightarrow AB \ /\ Aa$
   $A \rightarrow aAA \ /\ a$
   $B \rightarrow bBB \ /\ b$

4. Convert the following CFG to GNF

   $S \rightarrow ABA$	$A \rightarrow aA \ /\ \varepsilon$	$B \rightarrow bB \ /\ \varepsilon$

Syllabus : Unit – IV : Push Down Automata

Definitions - Model of PDA – Acceptance by PDA - Design of PDA - Equivalence of PDA's and CFL's - Deterministic PDA - pumping lemma for CFL - Closure properties of CFL (Without proof)

Definition for Push Down Automata

- ✓ Formal Definition of Pushdown Automaton

A pushdown automaton consists of seven tuple

$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$,

Where

Q - Finite set of states

Σ - Finite input alphabet

Γ - Finite alphabet of pushdown symbols

δ - Transition function $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow Q \times \Gamma$

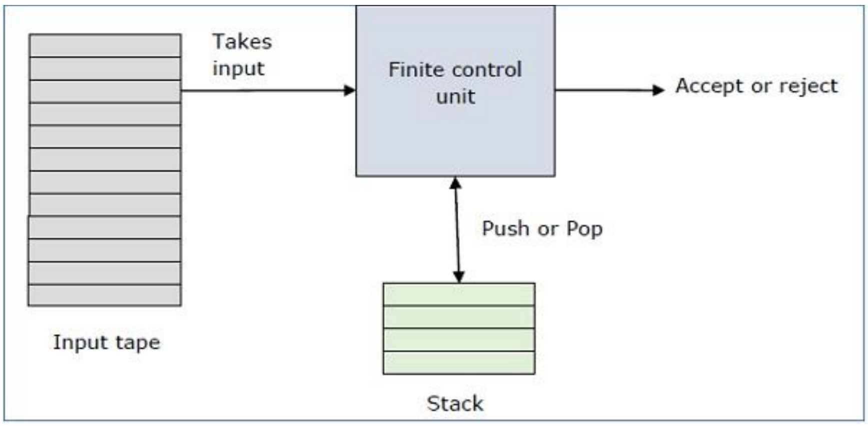
q_0 - start / initial state $q_0 \in Q$

Z_0 - start symbol on the pushdown $Z_0 \in \Gamma$

F - set of final states $F \in Q$

Model of PDA

- ✓ Pushdown Automata is a finite automaton with extra memory called stack which helps Pushdown automata to recognize Context Free Languages.
- ✓ A DFA can remember a finite amount of information, but a PDA can remember an infinite amount of information.
- ✓ The PDA consists of a finite set of states, a finite set of input symbols and a finite set of push down symbols.
- ✓ The finite control has control of both the input tape and the push down store.
- ✓ The stack head scans the top symbol of the stack.
- ✓ A pushdown automaton has three components:
  - o input tape
  - o control unit, and
  - o stack with infinite size.
- ✓ A stack does two operations:
  - o Push − a new symbol is added at the top.
  - o Pop − the top symbol is read and removed.



Acceptance by PDA

- ✓ There are two different ways to Acceptance by PDA
  - Acceptance by Final State
    - In final state acceptability, a PDA accepts a string when, after reading the entire string, the PDA is in a final state. From the starting state, we can make moves that end up in a final state with any stack values. The stack values are irrelevant as long as we end up in a final state.
    - Let M = (Q, ∑, Γ, δ, q_0, Z_0, F) be a PDA, then the language accepted by the set of final states F is

      L(M) = {w ; (q_0, w, z_0) ⊢* (p, ε, γ), p ∈ F, γ ∈ ⊢*}
  - Acceptance by Empty Stack
    - In empty stack acceptability, a PDA accepts a string when, after reading the entire string and also stack is empty, the PDA is in any state.
    - Let M = (Q, ∑, Γ, δ, q_0, Z_0, {q}) be a PDA, then the language accepted by the empty stack is:

      N(M) = {w ; (q0, w, z_0) ⊢* (q, ε, ε), q ∈ Q}

Instantaneous Description (ID)

- ✓ The ID must record the state and stack contains

  If M = (Q, ∑, Γ, δ, q_0, Z_0, F) be a PDA
  then

  (q_0, aw, zα) ⊢ (q_0, w, βα) if δ(q, a, z) = (p, β)

Equivalence of Acceptance of PDA from Empty Stack to Final state

If L is N(M_1) (the language accepted by empty stack) for some PDA M_1, then L is L(M_2) (language accepted by final state) for some PDA M_2 i.e. L = N(M_1) = L(M_2)

(or)

Prove that if L=N(P_N) for some PDA P_N = (Q, Σ, Γ, δ, q_0, Z_0, F), then there is a PDA P_F such that L=L(P_F).

(or)

If L is L(M_2) for some PDA M_2 then N(M_1)=L(M_2),L is N(M_1) for some PDA M_1.

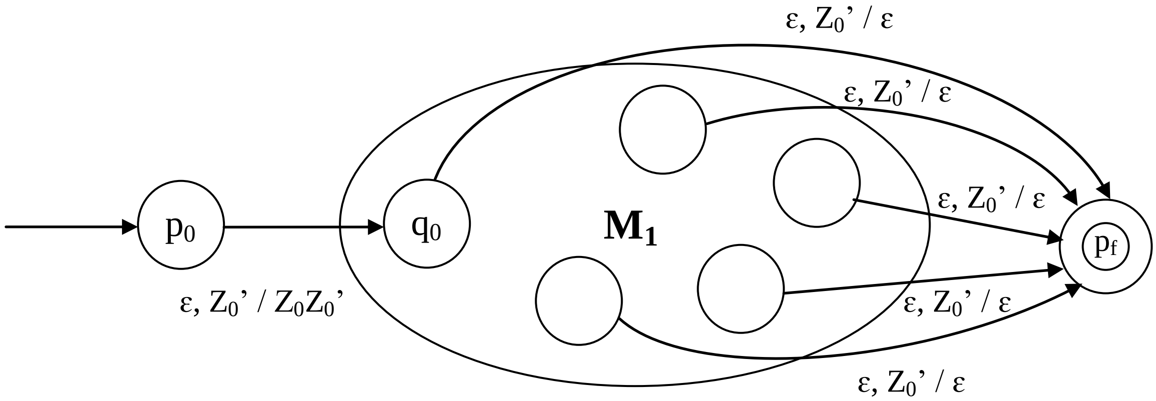
Theorem:

If M_1 = (Q, Σ, Γ, δ, q_0, Z_0, ∅) is a PDA accepting L by empty store, then construct a PDA M_2 = (Q', Σ', Γ', δ', q_0', Z_0', F) which accepts L by final state i.e., L = N(M_1) = L(M_2).

Proof:

M_2 is constructed in such a way that

a) by the initial state moves M_2 of , it reaches an initial id of M_1
b) by the final move of B, it reaches its final state.
c) all intermediate moves of B are in A.



Let us define M_2 as follows

$M_2 = (Q', \Sigma', \Gamma', \delta', q_0', Z_0', F)$

Where

$Q' = Q \cup \{p_0, p_f\}$

$\Sigma' = \Sigma$

$\Gamma = \Gamma \cup \{Z_0'\}$

$F' = \{p_f\}$ - New final state (not in Q)

$q_0' = p_0$ - New start state

Z_0' = New start symbol for stack.

δ' is given by rules:

R_1: $\delta'(p_0, \varepsilon, Z_0') = \{(q_0, Z_0Z_0')\}$

R_2: $\delta'(q, a, Z) = \delta(q, a, Z)$ for all q in Q, a in $(\Sigma \cup \varepsilon)$ and Z in Γ.

R_3: $\delta'(q, \varepsilon, Z_0') = \{(p_f, \varepsilon)\}$.

- By Rule R_1, the PDA M_2 moves from initial ID of M_2 to an initial ID of M_1. R_1 gives a 'ε' move. As a result of R_1, M_2 moves to the initial state of A with the start symbol z_0 on top of the stack.
- By Rule R_2 is used to simulate M_1. Once M_2 reaches an initial ID of M_1, R_2 is used to simulate moves of M_1. We can repeatedly apply R_2 until Z_0' is pushed to the top of the stack.
- By Rule R_3 is also a 'ε' move. Using R_3, M_2 moves to new final state p_f by erasing Z_0' in stack.

We have to show $N(M_1) = L(M_2)$.

Let $w \in N(M_1)$ then by definition of $N(M_1)$,

$M_1 : (q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon)$ for some $q \in Q$

By theorem

$(q, x, \alpha) \vdash^* (p, y, \beta) \Rightarrow (q, xw, \alpha y) \vdash^* (p, yw, \beta\gamma)$

we get

$M_1 : (q_0, w, Z_0Z_0') \vdash^* (q, \varepsilon, Z_0')$

Since empty store (δ) is a subset of δ' i.e. $\delta \subset \delta'$

we have

$M_2 : (q_0, w, Z_0Z_0') \vdash^* (q, \varepsilon, Z_0')$

Therefore we conclude that

$$M_2 : (p_0, w, z_0') \vdash (q_0, w, zz_0') \vdash^* (q, \varepsilon, z_0') \vdash (p_f, \varepsilon, \varepsilon)$$

Equivalence of Acceptance of PDA from final state to empty stack

If L is $N(M_1)$ for some PDA M_1, then L if L(M2) for some PDA M2.

(or)

If A= (Q, Σ, Γ, δ, q_0, Z_0, F) accept by final state, we can find a PDA B, accepting L by empty store i.e., L = T(A) = N(B).

If M_1 = (Q, Σ, Γ, δ, q_0, Z_0, F) accept by final state, we can find a PDA M_2, accepting L by empty store i.e., L = $L(M_1) = N(M_2)$.

Theorem:

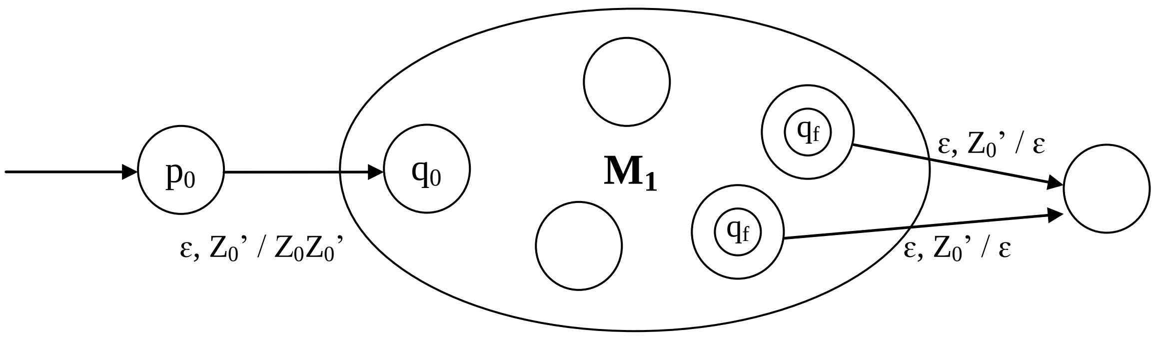
If M_1 = (Q, Σ, Γ, δ, q_0, Z_0, F) is a PDA accepting L by final state, then construct a PDA M_2 = (Q', Σ', Γ', δ', q_0', Z_0', ϕ) which accepts L by empty store.

i.e., L = $L(M_1) = N(M_2)$.

Proof:

M_2 is constructs from M_1 in such a way that

a) by the initial move of M_2 as initial ID of M_1 is reached.
b) once M_2 reaches an initial ID of M_1, it behaves like M_1 until a final state of M_1 is reached.
c) when M_2 reaches final state of M_1, it checks whether the input string is exhausted. Then M_2 simulates M_1 or it erases all the symbols in stack.



Let us define M_2 as follows

M_2 = (Q', Σ', Γ', δ', q_0', Z_0', ϕ)

Where

Q'= Q ∪ {p_0, p}

Σ' = Σ

Γ = Γ ∪{Z_0'}

F' = {p} - New final state (not in Q)

q_0' = p_0 - New start state

Z_0' = New start symbol for stack.

δ' is given by rules:

R_1 : δ'(p_0, ε, Z_0') = {(q_0, Z_0Z_0')}

R_2 : δ'(q_0, ε, Z) = {(q_f, ε)} for all Z ∈ Γ ∪ {Z_0'}.

R_3 : δ'(q, a, Z) = δ(q, a, Z) for all a ∈ Z, q ∈ Q, Z ∈ Γ.

R_4 : δ'(q, ε, Z) = δ(q, ε, z) ∪ {(p, ε)} for all Z ∈ Γ ∪ {Z_0'} and q ∈ F.

- Using R_1, M_2 enters an initial ID of M1 and start symbol Z_0 is placed on top of stack.
- R_2 is a ε move, using this M_2 erases all the symbols on stack.
- R_3 is used to make M_2 simulate M_1 until it reaches the final state of M_1.

We have to show that $L(M_1) = N(M_2)$

Let $w \in L(M_1)$ then

$M_1: (q_0, w, z_0) \vdash^* (q, \varepsilon, \alpha)$ for some $q \in F$, $\alpha \in \vdash^*$

Since $\delta' \subseteq \delta$ and by theorem

$M_1: (q, x, \alpha) \vdash^* (p, y, \beta) \Rightarrow (q, xw, \alpha y) \vdash^* (p, yw, \beta\gamma)$

We can write has

$M_2: (q_0, w, Z_0Z_0') \vdash^* (q, \varepsilon, \alpha z_0')$

Then M_2 can be computed has

$M_2: (p_0, w, Z_0') \vdash (q_0, w, ZZ_0')$
$\vdash^* (q, \varepsilon, Z_0')$
$\vdash (p_f, \varepsilon, \varepsilon)$

Design of PDA

1. Construct a PDA that accepts $L = \{a^n b^n ; n \geq 1\}$ accepted by Final State.

Solution:

Let $M = (Q, \sum, \Gamma, \delta, q_0, Z_0, F)$ be a PDA

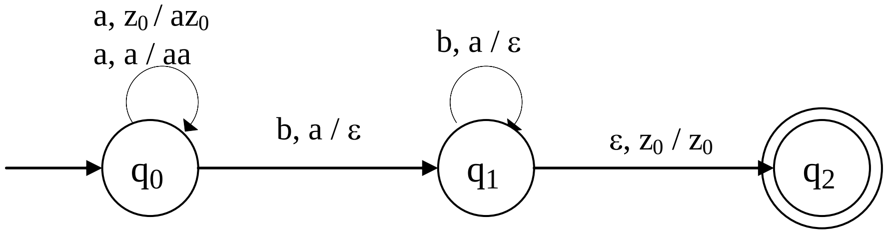
The productions are:

1. $\delta(q_0, a, z_0) = \{(q_0, az_0)\}$ } Push operations
2. $\delta(q_0, a, a) = \{(q_0, aa)\}$

3. $\delta(q_0, b, a) = \{(q_1, \varepsilon)\}$ } Pop operations
4. $\delta(q_1, b, a) = \{(q_1, \varepsilon)\}$

5. $\delta(q_1, \varepsilon, z_0) = \{(q_2, z_0)\}$ - Accept the Final State

Transition Diagram:



2. Construct a PDA that accepts $L = \{a^n b^n ; n \geq 1\}$ accepted by empty stack.

Solution:

Let $M = (Q, \sum, \Gamma, \delta, q_0, Z_0, F)$ be a PDA

The productions are:

1. $\delta(q_0, a, z_0) = \{(q_0, az_0)\}$
2. $\delta(q_0, a, a) = \{(q_0, aa)\}$

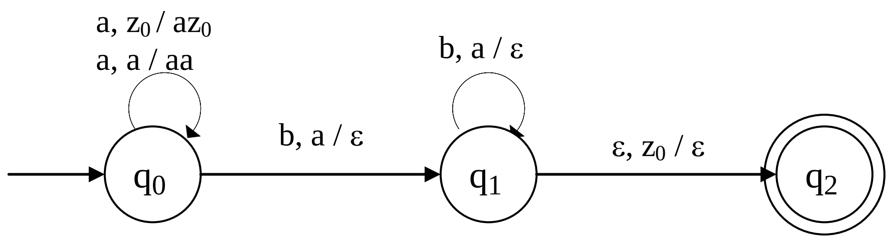
} Push operations

3. $\delta(q_0, b, a) = \{(q_1, \varepsilon)\}$
4. $\delta(q_1, b, a) = \{(q_1, \varepsilon)\}$

} Pop operations

5. $\delta(q_1, \varepsilon, z_0) = \{(q_2, \varepsilon)\}$ - Accept the empty stack

Transition Diagram:



3. Construct a PDA that accepts $L = \{0^n 1^n ; n \geq 0\}$ accepted by Final State.

Solution:

Let $M = (Q, \sum, \Gamma, \delta, q_0, Z_0, F)$ be a PDA

The productions are:

1. $\delta(q_0, 0, z_0) = \{(q_0, 0z_0)\}$
2. $\delta(q_0, \varepsilon, z_0) = \{(q_2, z_0)\}$
3. $\delta(q_0, 0, 0) = \{(q_0, 00)\}$

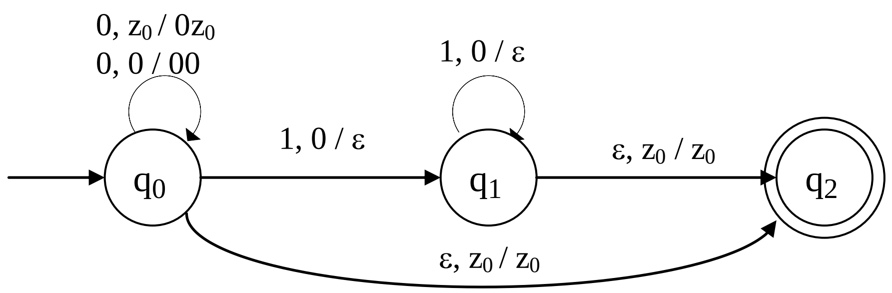
} Push operations

4. $\delta(q_0, 1, 0) = \{(q_1, \varepsilon)\}$
5. $\delta(q_1, 1, 0) = \{(q_1, \varepsilon)\}$

} Pop operations

6. $\delta(q_1, \varepsilon, z_0) = \{(q_2, z_0)\}$ - Accept the Final State

Transition Diagram:



4. Construct a PDA that accepts $L = \{0^n 1^n ; n \geq 0\}$ accepted by empty stack.

Solution:

Let $M = (Q, \sum, \Gamma, \delta, q_0, Z_0, F)$ be a PDA

The productions are:

1. $\delta(q_0, 0, z_0) = \{(q_0, 0z_0)\}$
2. $\delta(q_0, \varepsilon, z_0) = \{(q_2, \varepsilon)\}$
3. $\delta(q_0, 0, 0) = \{(q_0, 00)\}$

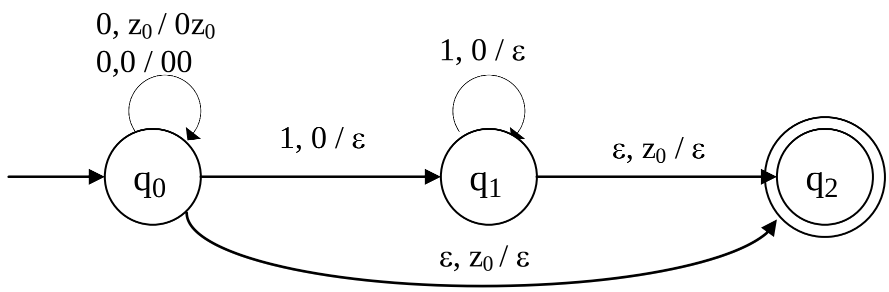
} Push operations

4. $\delta(q_0, 1, 0) = \{(q_1, \varepsilon)\}$
5. $\delta(q_1, 1, 0) = \{(q_1, \varepsilon)\}$

} Pop operations

6. $\delta(q_1, \varepsilon, z_0) = \{(q_2, \varepsilon)\}$ - Accept the empty stack

Transition Diagram:



5. Construct a PDA that accepts $L = \{wcw^R ; w \in (a+b)^*\}$ accepted by Final State.

Solution:

Let $M = (Q, \sum, \Gamma, \delta, q_0, Z_0, F)$ be a PDA

The productions are:

1. $\delta(q_0, a, z_0) = \{(q_0, az_0)\}$
2. $\delta(q_0, b, z_0) = \{(q_0, bz_0)\}$
3. $\delta(q_0, a, a) = \{(q_0, aa)\}$
4. $\delta(q_0, b, b) = \{(q_0, bb)\}$
5. $\delta(q_0, a, b) = \{(q_0, ab)\}$
6. $\delta(q_0, b, a) = \{(q_0, ba)\}$

} Push operations

7. $\delta(q_0, c, a) = \{(q_1, a)\}$
8. $\delta(q_0, c, b) = \{(q_1, b)\}$

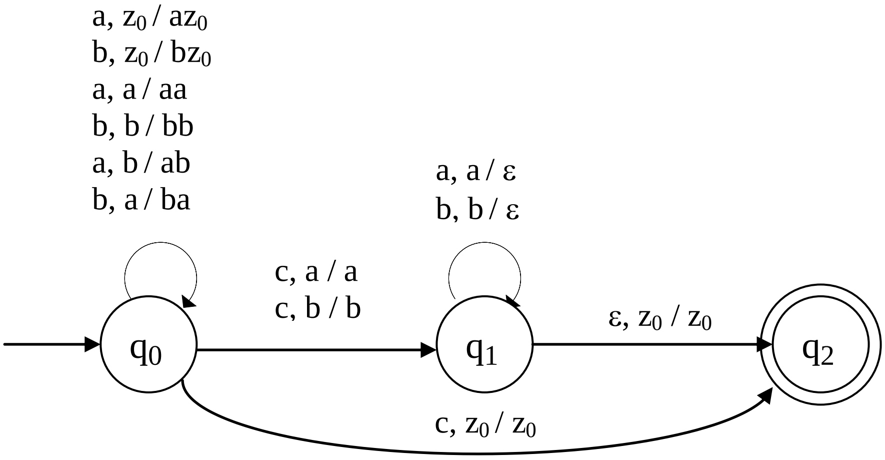
} Accept the separator ‘c’

9. $\delta(q_1, a, a) = \{(q_1, \varepsilon)\}$
10. $\delta(q_1, b, b) = \{(q_1, \varepsilon)\}$

} Pop operations

11. $\delta(q_1, \varepsilon, z_0) = \{(q_2, z_0)\}$ - Accept the Final State

Transition Diagram:



6. Construct a PDA that accepts L = {wcw^R ; w ∈ (a+b)*} accepted by empty stack.

Solution:

Let M = (Q, ∑, Γ, δ, q_0, Z_0, F) be a PDA

The productions are:

1. $\delta(q_0, a, z_0) = \{(q_0, az_0)$
2. $\delta(q_0, b, z_0) = \{(q_0, bz_0)$
3. $\delta(q_0, a, a) = \{(q_0, aa)$
4. $\delta(q_0, b, b) = \{(q_0, bb)$
5. $\delta(q_0, a, b) = \{(q_0, ab)$
6. $\delta(q_0, b, a) = \{(q_0, ba)$

} Push operations

7. $\delta(q_0, c, a) = \{(q_1, a)\}$
8. $\delta(q_0, c, b) = \{(q_1, b)\}$

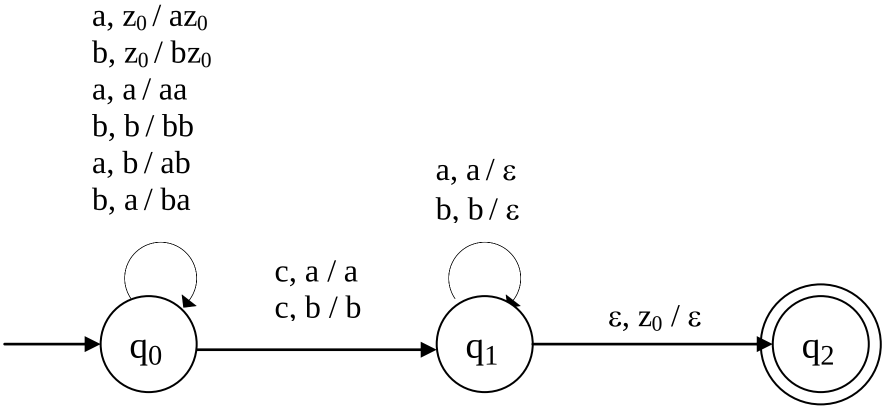
} Accept the separator 'c'

9. $\delta(q_1, a, a) = \{(q_1, \varepsilon)\}$
10. $\delta(q_1, b, b) = \{(q_1, \varepsilon)\}$

} Pop operations

11. $\delta(q_1, \varepsilon, \varepsilon) = \{(q_2, \varepsilon)\}$ - Accept the empty stack

Transition Diagram:



7. Design a PDA that accepts $L = \{ww^R \,;\, w \in (0+1)^*\}$ accepted by final state.

(or)

Design a PDA for even length palindrome.

Solution:

Let $M = (Q, \sum, \Gamma, \delta, q_0, Z_0, F)$ be a PDA

The productions are:

1. $\delta(q_0, 0, z_0) = \{(q_0, 0z_0)$
2. $\delta(q_0, 1, z_0) = \{(q_0, 1z_0)$
3. $\delta(q_0, 0, 0) = \{(q_0, 00)$
4. $\delta(q_0, 1, 1) = \{(q_0, 11)$
5. $\delta(q_0, 0, 1) = \{(q_0, 01)$
6. $\delta(q_0, 1, 0) = \{(q_0, 10)$

} Push operations

7. $\delta(q_0, \varepsilon, 0) = \{(q_1, 0)\}$
8. $\delta(q_0, \varepsilon, 1) = \{(q_1, 1)\}$

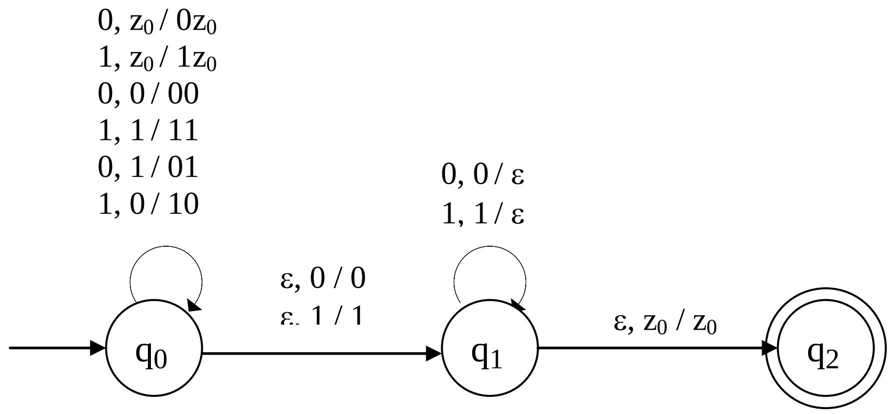
} Accept the separator 'ε'

9. $\delta(q_1, 0, 0) = \{(q_1, \varepsilon)\}$
10. $\delta(q_1, 1, 1) = \{(q_1, \varepsilon)\}$

} Pop operations

11. $\delta(q_1, \varepsilon, z_0) = \{(q_2, z_0)\}$ - Accept the Final State

Transition Diagram:



8. Construct a PDA that accepts $L = \{a^n b^m a^n \,;\, m, n \geq 1\}$ accepted by empty store.

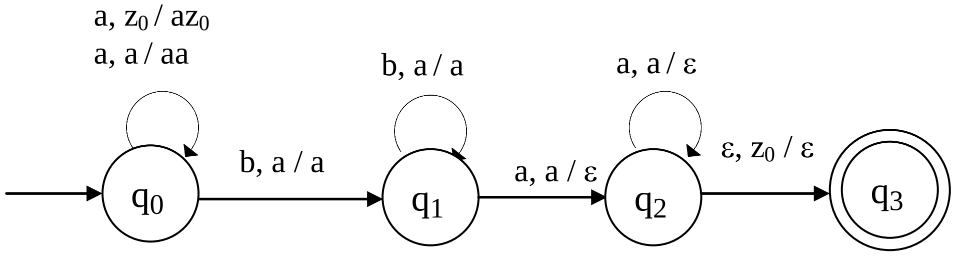
Solution:

Let $M = (Q, \sum, \Gamma, \delta, q_0, Z_0, F)$ be a PDA

The productions are:

1. $\delta(q_0, a, z_0) = \{(q_0, az_0)$
2. $\delta(q_0, a, a) = \{(q_0, aa)$
3. $\delta(q_0, b, a) = \{(q_1, a)$
4. $\delta(q_1, b, a) = \{(q_1, a)$
5. $\delta(q_1, a, a) = \{(q_2, \varepsilon)$
6. $\delta(q_2, a, a) = \{(q_2, \varepsilon)$
7. $\delta(q_2, \varepsilon, z_0) = \{(q_2, \varepsilon)$

Transition Diagram:



9. Design a PDA that accepts $L = \{a^n b^m c^m d^n;\ n, m \geq 1\}$ accepted by empty store and check whether the string w = aaabcddd is accept or not.

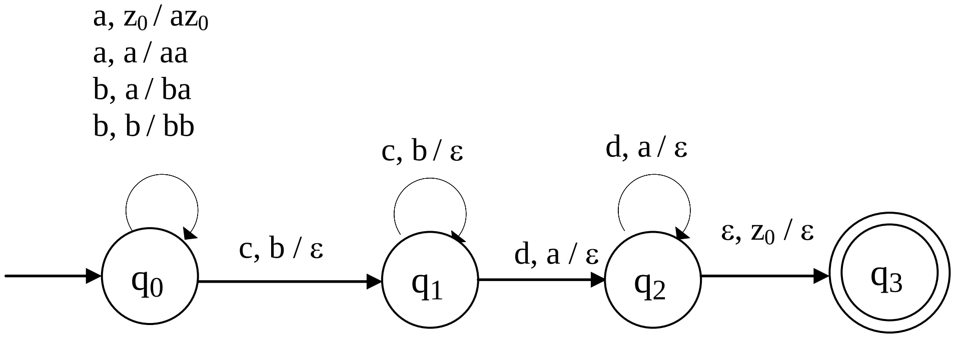
Solution:

Let M = (Q, ∑, Γ, δ, q_0, Z_0, F) be a PDA

The productions are:

1. $\delta(q_0, a, z_0) = \{(q_0, az_0)$
2. $\delta(q_0, a, a) = \{(q_0, aa)$
3. $\delta(q_0, b, a) = \{(q_1, ba)$
4. $\delta(q_0, b, b) = \{(q_1, bb)$
5. $\delta(q_0, c, b) = \{(q_1, \varepsilon)$
6. $\delta(q_1, c, b) = \{(q_1, \varepsilon)$
7. $\delta(q_1, d, a) = \{(q_2, \varepsilon)$
8. $\delta(q_2, d, a) = \{(q_2, \varepsilon)$
9. $\delta(q_2, \varepsilon, z_0) = \{(q_3, \varepsilon)$

Transition Diagram:



String w = aaabcddd

$(q_0, aaabcddd, z_0) \vdash (q_0, aabcddd, az_0)$
$\vdash (q_0, abcddd, aaz_0)$
$\vdash (q_0, bcddd, aaaz_0)$
$\vdash (q_0, cddd, baaaz_0)$
$\vdash (q_1, ddd, aaaz_0)$
$\vdash (q_2, dd, aaz_0)$
$\vdash (q_2, d, az_0)$
$\vdash (q_2, \varepsilon, z_0)$
$\vdash (q_3, \varepsilon, \varepsilon)$ - Hence the string is accepted.

Tutorial Problems:

10. Construct a PDA that accepts L = $\{a^n b^{2n}\ ;\ n \geq 1\}$ accepted by empty stack.
11. Construct a PDA that accepts L = $\{a^n b a^n\ ;\ n > 0\}$ accepted by final state.
12. Design a PDA that accepts L = $\{a^n b a^n\ ;\ n > 0\}$ accepted by final state.
13. Construct a PDA that accepts L = $\{a^n b^m a^n\ ;\ n > 0 \text{ and } m = n+1\}$ accepted by empty store.
14. Construct a PDA that accepts L = $\{a^n b^m;\ n > 0 \text{ and } m \geq n\}$ accepted by empty store.
15. Construct a PDA that accepts L = $\{a^n b^m c^{m-n}\ ;\ m, n \geq 0 \text{ and } m \geq n\}$ and check whether the given string is accepted or not. (a) aabbbbcc (b) aabbc

Equivalence of PDA's and CFL's

i) Conversion of CFG to PDA

Theorem:

For any CFG L, there exists an PDA M such that L=L(M).

Proof:

Let G = (V, T, P, S) be a CFG.

Construct the PDA M that accepts L(G) by empty stack as follows:

$M = (\{q\}, T, V \cup T, \delta, q, S)$

Where transition function δ is defined by:

1. For each variable A, make $\delta(q, \varepsilon, A) = \{(q, \alpha)$ if $A \to \alpha$ is a production of P$\}$.
2. For each terminal a, make $\delta(q, a, a) = \{(q, \varepsilon)\}$.

✓ **Problems for CFG to PDA**

1. Construct a PDA from the following CFG.
   G = ({S, A}, {a, b}, P, S) where the productions are
   $S \to AS / \varepsilon$
   $A \to aAb / Sb / a$

   Solution:

   Let the equivalent PDA, M = ({q}, {a, b}, {a, b, A, S}, δ, q, S)

   where δ:

   $\delta(q, \varepsilon, S) = \{(q, AS), (q, \varepsilon)\}$

   $\delta(q, \varepsilon, A) = \{(q, aAb), (q, Sb), (q, a)\}$

   $\delta(q, a, a) = \{(q, \varepsilon)\}$

   $\delta(q, b, b) = \{(q, \varepsilon)\}$

2. Consider the grammar G = (V, T, P, S) with V = {S}, T = {a, b, c}, and P = {S → aSa, S → bSb, S → c}

   Solution:

   Let the equivalent PDA, M = ({q}, {a, b, c}, {a, b, c, S}, δ, q, S)

   where δ:

   δ(q, ε , S) = {(q, aSa), (q, bSb), (q, c)}

   δ(q, a, a) = {(q, ε)}

   δ(q, b, b) = {(q, ε)}

   δ(q, c, c) = {(q, ε)}

3. Consider the grammar G = (V_N, V_T, P, S) with P = { S → abA / baA / B / ε A→ bS / b, B → aS, C → ε}

   Solution:

   Let the equivalent PDA, M = ({q}, {a, b}, {a, b, S, A, B, C}, δ, q, S)

   where δ:

   δ(q, ε , S) = {(q, abA), (q, baA), (q, B), (q, ε)}

   δ(q, ε , A) = {(q, bS), (q, b)}

   δ(q, ε , B) = {(q, aS)}

   δ(q, ε , C) = {(q, ε)}

   δ(q, a, a) = {(q, ε)}

   δ(q, b, b) = {(q, ε)}

4. Consider the grammar G = (V_N, V_T, P, S)

   Where P :

   S → A / B / ε

   A → 0S/1B/0

   B → 0S/1A/1

   Solution:

   Let the equivalent PDA, M = ({q}, {0, 1}, {0, 1, S, A, B}, δ, q, S)

   where δ:

   δ(q, ε , S) = {(q, A), (q, B), (q, ε)}

   δ(q, ε , A) = {(q, 0S), (q, 1B), (q, 0)}

   δ(q, ε , B) = {(q, 0S), (q, 1A), (q, 1)}

   δ(q, 0, 0) = {(q, ε)}

   δ(q, 1, 1) = {(q, ε)}

5. Construct a PDA that will accept the language generated by the grammar G = ({S, A}, {a, b}, P, S) with the productions S → AA / a, A → SA / b and test whether "abbabb" is in N(M).

Solution:

Let the equivalent PDA, M = ({q}, {a, b}, {a, b, S, A}, δ, q, S)

where δ:

δ(q, ε , S) = {(q, AA), (q, a }

δ(q, ε , A) = {(q, SA), (q, b)}

δ(q, a, a) = {(q, ε)}

δ(q, b, b) = {(q, ε)}

Test whether "abbabb" is in N(M):

δ(q, abbabb , S)	⊢ δ(q, abbabb , AA)	by δ(q, ε , S) = {(q, AA)}
	⊢ δ(q, abbabb , SAA)	by δ(q, ε , A) = {(q, SA)}
	⊢ δ(q, abbabb , aAA)	by δ(q, ε , S) = {(q, a)}
	⊢ δ(q, abbabb , aAA)	by δ(q, a, a) = {(q, ε)}
	⊢ δ(q, bbabb , SAA)	by δ(q, ε , A) = {(q, SA)}
	⊢ δ(q, bbabb , AAAA)	by δ(q, ε , S) = {(q, AA)}
	⊢ δ(q, bbabb , bAAA)	by δ(q, ε , A) = {(q, b)}
	⊢ δ(q, babb , AAA)	by δ(q, b , b) = {(q, ε)}
	⊢ δ(q, babb , bAA)	by δ(q, ε , A) = {(q, b)}
	⊢ δ(q, abb , AA)	by δ(q, b , b) = {(q, ε)}
	⊢ δ(q, abb , SAA)	by δ(q, ε , A) = {(q, SA)}
	⊢ δ(q, abb , aAA)	by δ(q, ε , S) = {(q, a)}
	⊢ δ(q, bb , AA)	by δ(q, a, a) = {(q, ε)}
	⊢ δ(q, bb , bA)	by δ(q, ε , A) = {(q, b)}
	⊢ δ(q, b , A)	by δ(q, b , b) = {(q, ε)}
	⊢ δ(q, b , b)	by δ(q, ε , A) = {(q, b)}
	⊢ δ(q, ε , ε)	by δ(q, b , b) = {(q, ε)}

<u>Tutorial Problems:</u>

6. Consider the grammar G = (V_N, V_T, P, S) and test whether "abbabb" is in N(M).

Where P :

S → abA / baA / B / ε

A→ bS / b

B → aS

C → ε

7. Consider the grammar G = (V, T, P, S)
   Where P :
   $A \rightarrow aB$
   $B \rightarrow aB/bB/\varepsilon$
8. Consider the grammar G = (V, T, P, S) and test whether "0101001" is in N(M).
   Where P :
   $S \rightarrow 0S/1A/1/0B/0$
   $A \rightarrow 0A/1B/0/1$
   $B \rightarrow 0B/1A/0/1$
9. Consider the grammar G = (V, T, P, S)
   Where P :
   $A \rightarrow Ba/Ab/b$
   $B \rightarrow Ca/Bb$
   $C \rightarrow Aa/Cb/a$
10. Consider the grammar G = (V, T, P, S)
   Where P :
   $A \rightarrow aB/bA/b$
   $B \rightarrow aC/bB$
   $C \rightarrow aA/bC/a$
11. Consider the grammar G = (V, T, P, S)
   Where P :
   $S \rightarrow ABCD$
   $A \rightarrow aab$
   $B \rightarrow bba / bbaB$
   $C \rightarrow bab$
   $D \rightarrow aab / aabD$

ii) Conversion of PDA to CFG

Theorem:

If L is N(M) for some PDA M then L is CFL.

Proof:

Let $M = (Q, \sum, \Gamma, \delta, q_0, Z_0, \emptyset)$ be a PDA
Construct the CFG G that accepts L(M) by empty stack as follows:
G = (V, T, P, S)
Where production P is defined by:

- ✓ The productions in P are induced by moves of PDA as follows:

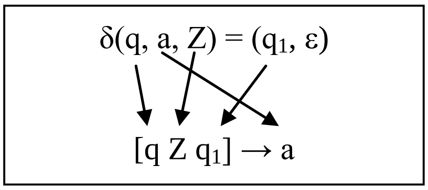
Step 1: Rules for start symbol:

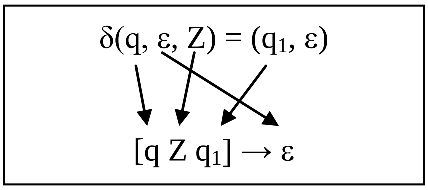
S productions are given by $S \rightarrow [q_0 Z_0 q]$ for every $q \in Q$
For example:
We have two states (q_0, q_1), so two rules for starting variable.
$S \rightarrow [q_0 Z_0 q_0]$
$S \rightarrow [q_0 Z_0 q_1]$

Step 2: Rules for POP operations:

Each erasing move $\delta(q, a, Z) = (q_1, \varepsilon)$ induces production $[q\ Z\ q'] \rightarrow a$

For example:





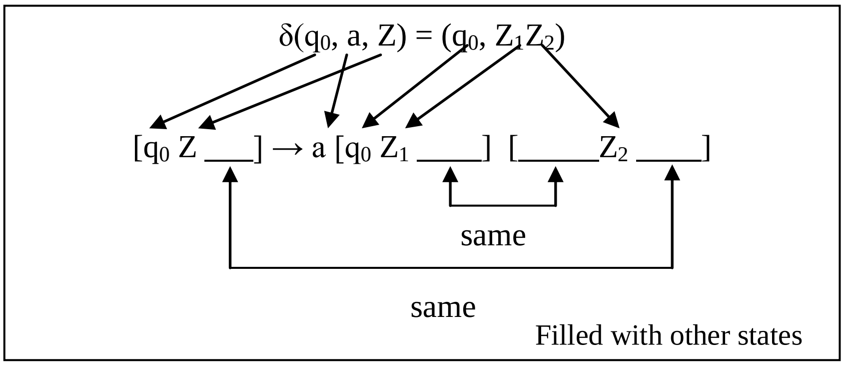
Step 3: Rules for PUSH operations:

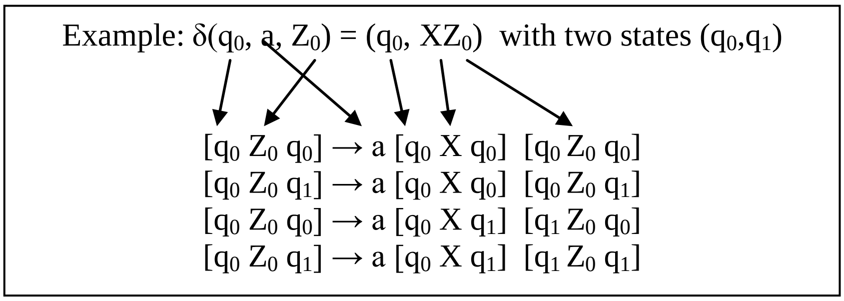
Each non-erasing move $\delta(q, a, Z) = (q', Z_1 Z_2 Z_3 \ldots Z_n)$ induces many productions of form.

$$[q\ Z\ q'] \rightarrow a\ [q_1\ Z_1\ q_2]\ [q_2\ Z_2\ q_3] \ldots\ldots\ldots\ldots\ldots [q_n\ Z_n\ q']$$

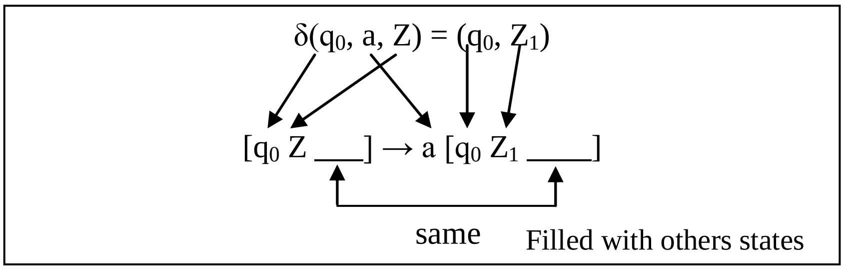
Where each state $q', q_1, q_2, \ldots q_n$ can be any state in Q

General Format 1:





General Format 2:



Example: $\delta(q_0, a, Z_0) = (q_0, X)$ with two states (q_0,q_1)

$[q_0\ Z_0\ q_0] \rightarrow a\ [q_0\ X\ q_0]$
$[q_0\ Z_0\ q_1] \rightarrow a\ [q_0\ X\ q_1]$

✓ **<u>Problems for CFG to PDA</u>**

1. Convert the PDA P= $(\{p, q\},\{0,1\},\{X,Z_0\}, \delta, q, Z_0)$ to a CFG , if is given by
   1. $\delta(q, 1, Z_0) = \{(q, XZ_0)\}$
   2. $\delta(q, 1, X) = \{(q, XX)\}$
   3. $\delta(q, 0, X) = \{(p, X)\}$
   4. $\delta(q, \varepsilon, X) = \{(q, \varepsilon)\}$
   5. $\delta(p, 1, X) = \{(p, \varepsilon)\}$
   6. $\delta(p, 0, Z_0) = \{(q, Z_0)\}$

Solution:

Step 1: Find the push and pop operations:
1. $\delta(q, 1, Z_0) = \{(q, XZ_0)\}$ - Push
2. $\delta(q, 1, X) = \{(q, XX)\}$ - Push
3. $\delta(q, 0, X) = \{(p, X)\}$ - Push
4. $\delta(q, \varepsilon, X) = \{(q, \varepsilon)\}$ - Pop
5. $\delta(p, 1, X) = \{(p, \varepsilon)\}$ - Pop
6. $\delta(p, 0, Z_0) = \{(q, Z_0)\}$ - Push

Step 2: Rules for start symbol:
We have two states q and p.
So, S productions are
1. **$S \rightarrow [q\ Z_0\ q]$**
2. **$S \rightarrow [q\ Z_0\ p]$**

Step 2: Rules for POP operations:
2. 1 Rules for $\delta(q, \varepsilon, X) = \{(q, \varepsilon)\}$ --- (4)
3. **$[q\ X\ q] \rightarrow \varepsilon$**

2. 2 Rules for $\delta(p, 1, X) = \{(p, \varepsilon)\}$ --- (5)
4. **$[p\ X\ p] \rightarrow 1$**

Step 3: Rules for PUSH operations:
3. 1 Rules for $\delta(q, 1, Z_0) = \{(q, XZ_0)\}$ --- (1)
5. **$[q\ Z_0\ q] \rightarrow 1\ [q\ X\ q]\ [q\ Z_0\ q]$**
6. **$[q\ Z_0\ p] \rightarrow 1\ [q\ X\ q]\ [q\ Z_0\ p]$**
7. **$[q\ Z_0\ q] \rightarrow 1\ [q\ X\ p]\ [p\ Z_0\ q]$**
8. **$[q\ Z_0\ p] \rightarrow 1\ [q\ X\ p]\ [p\ Z_0\ p]$**

3. 2 Rules for δ(q,1, X) = {(q,XX)} --- (2)

9. [q X q] → 1 [q X q] [q X q]
10. [q X p] → 1 [q X q] [q X p]
11. [q X q] → 1 [q X p] [p X q]
12. [q X p] → 1 [q X p] [p X p]

3. 3 Rules for δ(q,0, X) = {(p,X)} --- (3)

13. [q X q] → 0 [q X q]
14. [q X p] → 0 [q X p]

3. 4 Rules for δ(p, 0, Z_0) = {(q, Z_0)} --- (6)

15. [p Z_0 q] → 0 [q Z_0 q]
16. [p Z_0 p] → 0 [q Z_0 p]

2. Convert the PDA P= ({q, p}, {0,1},{Z_0, X}, δ, q, Z_0,{p}) to a Context free grammar.
   1. δ(q,0, Z0) ={(q, XZ0)}
   2. δ(q,0, X) = {(q, XX)}
   3. δ(q,1, X) = {(q, X)}
   4. δ(q, ε, X) = {(p, ε)}
   5. δ(p, ε, X) = {(p, ε)}
   6. δ(p,1, X) = {(p, XX)}
   7. δ(p,1, Z0) = {(p, ε)}

Solution:

Step 1: Find the push and pop operations:

1. δ(q, 0, Z_0) ={(q, XZ_0)} - Push
2. δ(q, 0, X) = {(q, XX)} - Push
3. δ(q, 1, X) = {(q, X)} - Push
4. δ(q, ε, X) = {(p, ε)} - Pop
5. δ(p, ε, X) = {(p, ε)} - Pop
6. δ(p,1, X) = {(p, XX)} - Push
7. δ(p,1, Z_0) = {(p, ε)} - Pop

Step 2: Rules for start symbol:

We have two states q and p.
So, S productions are

1. S → [q Z_0 q]
2. S → [q Z_0 p]

Step 2: Rules for POP operations:

2. 1 Rules for δ δ(q, ε, X) = {(p, ε)} --- (4)

3. [q X p] → ε

2. 2 Rules for δ(p, ε, X) = {(p, ε)} --- (5)

4. [p X p] → ε

2. 3 Rules for δ(p,1, Z_0) = {(p, ε)} --- (7)

5. [p Z_0 p] → 1

Step 3: Rules for PUSH operations:

3. 1 Rules for $\delta(q, 0, Z_0) = \{(q, XZ_0)\}$--- (1)

6. $[q\ Z_0\ q] \rightarrow 0\ [q\ X\ q]\ [q\ Z_0\ q]$
7. $[q\ Z_0\ p] \rightarrow 0\ [q\ X\ q]\ [q\ Z_0\ p]$
8. $[q\ Z_0\ q] \rightarrow 0\ [q\ X\ p]\ [p\ Z_0\ q]$
9. $[q\ Z_0\ p] \rightarrow 0\ [q\ X\ p]\ [p\ Z_0\ p]$

3. 2 Rules for $\delta(q, 0, X) = \{(q, XX)\}$--- (2)

10. $[q\ X\ q] \rightarrow 0\ [q\ X\ q]\ [q\ X\ q]$
11. $[q\ X\ p] \rightarrow 0\ [q\ X\ q]\ [q\ X\ p]$
12. $[q\ X\ q] \rightarrow 0\ [q\ X\ p]\ [p\ X\ q]$
13. $[q\ X\ p] \rightarrow 0\ [q\ X\ p]\ [p\ X\ p]$

3. 3 Rules for $\delta(q, 1, X) = \{(q, X)\}$ --- (3)

14. $[q\ X\ q] \rightarrow 1\ [q\ X\ q]$
15. $[q\ X\ p] \rightarrow 1\ [q\ X\ p]$

3. 4 Rules for $\delta(p,1, X) = \{(p, XX)\}$ ---- (6)

16. $[p\ X\ q] \rightarrow 1\ [p\ X\ q]\ [q\ X\ q]$
17. $[p\ X\ p] \rightarrow 1\ [p\ X\ q]\ [q\ X\ p]$
18. $[p\ X\ q] \rightarrow 1\ [p\ X\ p]\ [p\ X\ q]$
19. $[p\ X\ p] \rightarrow 1\ [p\ X\ p]\ [p\ X\ p]$

<u>Tutorial Problems:</u>

1. Construct a Context free grammar G which accepts N(M), where $M=(\{q0,q1\},\{a,b\},\{z0,z\},\delta,q0,z0,\Phi)$ and where δ is given by

   $\delta(q0,b,z0) = \{(q0,zz0)\}$, $\delta(q0, \varepsilon,z0) = \{(q0, \varepsilon)\}$
   $\delta(q0,b,z) = \{(q0,zz)\}$, $\delta(q0,a,z) = \{(q1,z)\}$
   $\delta(q1,b,z) = \{(q1, \varepsilon)\}$, $\delta(q1,a,z0) = \{(q0,z0)\}$

2. Construct the grammar from the given PDA. $M=(\{q0, q1\},\{0,1\},\{X,Z_0\},\delta,q0,Z0,\Phi)$ and where δ is given by

   $\delta(q0,0,z0) = \{(q0,XZ_0)\}$, $\delta(q0,0,X) = \{(q0,XX)\}$,
   $\delta(q0,1,X) = \{(q1, \varepsilon)\}$, $\delta(q1,1,X) = \{(q1, \varepsilon)\}$,
   $\delta(q1, \varepsilon,X) = \{(q1, \varepsilon)\}$, $\delta(q1, \varepsilon, Z_0) = \{(q1, \varepsilon)\}$.

3. Let $M =(\{q0,q1\}, \{0,1\}, \{S,A\}, \delta, q0, Z0, \phi\}$ to be a PDA

   Where δ is given by

   $\delta(q_0, 0, S) = \{(q_0, AS)\}$
   $\delta(q_0, 0, A) = \{(q_0, AA), (q1, S)\}$
   $\delta(q_0, 1, A) = \{(q_1, \varepsilon)\}$
   $\delta(q_1, 1, A) = \{(q_1, \varepsilon)\}$
   $\delta(q_1, \varepsilon, A) = \{(q_1, \varepsilon)\}$
   $\delta(q_1, \varepsilon, S) = \{(q_1, \varepsilon)\}$ Construct a CFG G = (V, T, P, S) generating N (M).

Deterministic PDA

- ✓ In general terms, a deterministic PDA is one in which there is at most one possible transition from any state based on the current input.
- ✓ A deterministic pushdown automaton (DPDA) is a 7-tuple

$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$,
Where

Q - Finite set of states
Σ - Finite input alphabet
Γ - Finite alphabet of pushdown symbols
δ - Transition function $\delta : Q \times \Sigma^* \times \Gamma^* \rightarrow (Q \times \Gamma^*) \cup \{\emptyset\}$
q_0 - start / initial state $q0 \in Q$
Z_0 - start symbol on the pushdown $Z_0 \in \Gamma$
F - set of final states $F \in Q$

Example: Describe a DPDA that can recognize the language {w ; w contains more a's than b's}.

Non-Deterministic PDA

- ✓ In general terms, a non-deterministic PDA is one in which there is more than two possible transition from any state based on the current input.
- ✓ A non-deterministic pushdown automaton (NPDA) is a 7-tuple

$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$,
Where

Q - Finite set of states
Σ - Finite input alphabet
Γ - Finite alphabet of pushdown symbols
δ - Transition function $\delta : Q \times \Sigma^* \times \Gamma^* \rightarrow 2^{(Q \times \Gamma^*)}$
q_0 - start / initial state $q0 \in Q$
Z_0 - start symbol on the pushdown $Z_0 \in \Gamma$
F - set of final states $F \in Q$

Example: Define a NPDA that recognizes the language $\{ww^R ; w \in \Sigma^*\}$.

Pumping Lemma

If **L** is a context-free language, there is a pumping length **p** such that any string $\mathbf{w \in L}$ of length $\geq$ **p** can be written as **w = uvxyz**, where $\mathbf{vy \neq \varepsilon}$, $\mathbf{|vxy| \leq p}$, and for all $\mathbf{i \geq 0}$, $\mathbf{uv^ixy^iz \in L}$.

Applications of Pumping Lemma

Pumping lemma is used to check whether a grammar is context free or not. Let us take an example and show how it is checked.

Problem

1. **Find out whether the language L = {xnynzn | n ≥ 1} is context free or not.**

   Solution

   1. Let L is context free. Then, L must satisfy pumping lemma.
   2. At first, choose a number n of the pumping lemma. Then, take z as 0n1n2n.
   3. Break z into uvwxy, where |vwx| ≤ n and vx ≠ ε.
   4. Hence vwx cannot involve both 0s and 2s, since the last 0 and the first 2 are at least (n+1) positions apart. There are two cases:
   5. Case 1 − vwx has no 2s. Then vx has only 0s and 1s. Then uwy, which would have to be in L, has n 2s, but fewer than n 0s or 1s.
   6. Case 2 − vwx has no 0s.
   7. Here contradiction occurs.
   8. Hence, L is not a context-free language.

2. **The text uses the pumping lemma to show that {ww | w in (0 + 1)*} is not a CFL.**
   1. Suppose L were a CFL.
   2. Let n be L's pumping-lemma constant.
   3. Consider z = 0n10n10n.
   4. We can write z = uvwxy, where |vwx| < n, and |vx| > 1.
   5. Case 1: vx has no 0's.
   6. Then at least one of them is a 1, and uwy has at most one 1, which no string in L does.
   7. Still considering z = 0n10n10n.
   8. Case 2: vx has at least one 0.
   9. vwx is too short (length < n) to extend to all three blocks of 0's in 0n10n10n.
   10. Thus, uwy has at least one block of n 0's, and at least one block with fewer than n 0's.
   11. Thus, uwy is not in L.

Closure properties of CFL (Without proof)

1. CFLs are closed under union
   If L1 and L2 are CFLs, then L1 ∪ L2 is a CFL.
2. CFLs are closed under concatenation
   If L1 and L2 are CFLs, then L1L2 is a CFL.
3. CFLs are closed under Kleene closure
   If L is a CFL, then L^* is a CFL.
4. CFLs are not closed under intersection
   If L1 and L2 are CFLs, then L1 ∩ L2 may not be a CFL.
5. CFLs are not closed under complement
   If L is a CFL, then L may not be a CFL.

Dr. D. Jagadeesan, B.E., M.Tech., Ph.D., MISTE
Professor in CSE,

Syllabus : Unit – V : Turing Machine and Undecidabality

Definition - Model - Language acceptance - Design of Turing Machine - Computable languages and functions - Modifications of Turing machine - Universal Turing machine- Chomsky hierarchy of languages - Grammars and their machine recognizers - Undecidabile Post correspondence problem.

Introduction

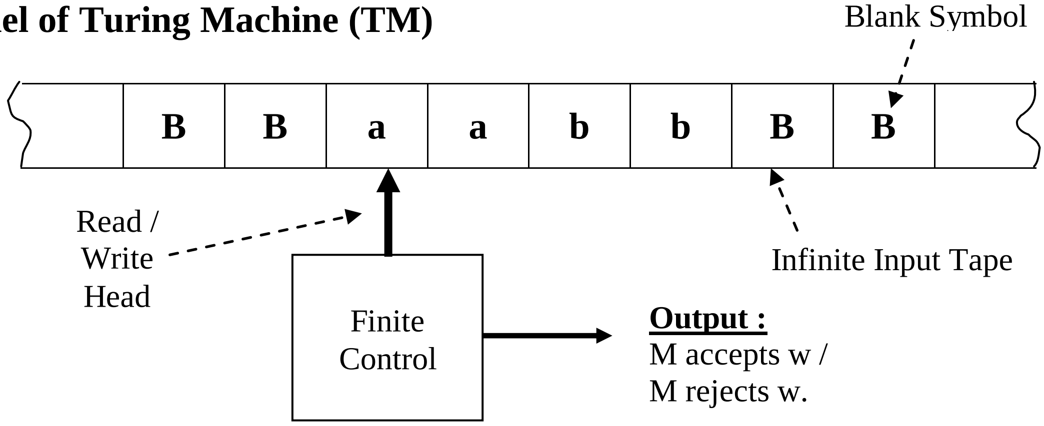
- ✓ A Turing Machine is an accepting device which accepts the languages (recursively enumerable set) generated by type 0 grammars.
- ✓ It was invented in 1936 by Alan Turing.

Definition

- ✓ A Turing Machine (TM) is a mathematical model which consists of
  - o An **infinite length tape** divided into cells, each cell contains a symbol from some finite alphabet. The alphabet contains a special blank symbol (here written as '0') and one or more other symbols. The tape is assumed to be arbitrarily extendable to the left and to the right.
  - o A **head** which reads the input tape.
  - o A **state** register stores the state of the Turing machine.
- ✓ After reading an input symbol, it is replaced with another symbol, its internal state is changed, and it moves from one cell to the right or left. If the TM reaches the final state, the input string is accepted, otherwise rejected.

- ✓ A TM can be formally described as a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$
  Where
  - Q is a finite set of states
  - Σ is the input alphabet
  - Γ is the tape alphabet
  - δ is a transition function; $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$.
  - q_0 is the initial state, $q_0 \in Q$
  - B is the blank symbol, $B \in \Gamma$
  - F is the set of final states, $F \in Q$

Model of Turing Machine (TM)



- TM has three components:
  i. **Finite state control**:
     - It is in one of a finite number of states at each instant, and is connected to the tape head.
  ii. **Tape head:**
     - It is used to scans one of the tape symbol (cell) of the tape at each instant, and is connected to the finite state control. It can read and write symbols from/to the tape, and it can move left and right along the tape.
  iii. **Tape**:
     - It is consists of an infinite number of tape cells, each of which can store one of a finite number of tape symbols at each instant. The tape is infinite both to the left and to the right.

Language acceptance

- A TM accepts a language if it enters into a final state for any input string w. A language is recursively enumerable (generated by Type-0 grammar) if it is accepted by a Turing machine.
- A string w is accepted by the TM, $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ if $q_0w \vdash^* \alpha_1 q_f \alpha_2$ for some $\alpha_1, \alpha_2 \in \Gamma^*$, $q_f \in F$.
- The language accepted by the TM M is denoted as

$$T(M) = \{w \,;\, w \in \Sigma^*, q_0w \vdash^* \alpha_1 q_f \alpha_2 \text{ for some } \alpha_1, \alpha_2 \in \Gamma^*, q_f \in F\}$$

Moves in a TM

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be a TM. The symbol is used to represent the move.

$\vdash$ - Single move

$\vdash^*$ - Zero or more moves

- $\delta(q, x)$ causes a change in ID of the TM. This is called as a move.

Input head Move to Left side:

- Suppose $\delta(q, x_i) = (p, y, L)$ and the input string to be processed is $x_1x_2x_3 \ldots x_n$ and the head is pointing to symbol x_i.
- Before processing:

  $x_1x_2x_3 \ldots x_{i-1}\, q\, x_i \ldots\ldots x_n$
- After processing:

  $x_1x_2x_3 \ldots x_{i-2}\, q\, x_{i-1}\, y\, x_{i+1} \ldots\ldots x_n$

$$\mathbf{x_1x_2x_3 \ldots x_{i-1}\, q\, x_i \ldots\ldots x_n \vdash x_1x_2x_3 \ldots x_{i-2}\, q\, x_{i-1}\, y\, x_{i+1} \ldots\ldots x_n}$$

Input head Move to Right side:

- Suppose $\delta(q, x_i) = (p, y, R)$ and the input string to be processed is $x_1x_2x_3 \ldots x_n$ and the head is pointing to symbol x_i.
- Before processing:

  $x_1x_2x_3 \ldots x_{i-1}\, q\, x_i \ldots\ldots x_n$
- After processing:

  $x_1x_2x_3 \ldots x_{i-2}\, x_{i-1}\, y\, q\, x_{i+1} \ldots\ldots x_n$

$$\mathbf{x_1x_2x_3 \ldots x_{i-2}\, x_{i-1}\, y\, q\, x_{i+1} \ldots\ldots x_n}$$

Design of Turing Machine

1. Design a TM to recognize the language L ={$a^n b^n$; n>0} and test whether the strings w= "aabb" and "abbb" are accepts or not.

Solution:

The TM is designed using the following steps:

Step 1 : M replaces the leftmost 'a' by 'x' and moves right to the leftmost 'b', replacing it by 'y'.

Step 2 : Then M moves left to find the rightmost 'x' and moves one cell right to the leftmost 'a' and repeat the step 1.

Step 3 : While searching for a 'b', if a blank (B) is encountered, and then M halts without accepting.

Step 4 : After changing a 'b' to 'y', if M finds no more a's, then M checks no more b's remains, M accepting the string else not.

Let M = ({q_0, q_1, q_2, q_3, q_4}, {a, b}, {a, b, B}, δ, q_0, B, {q_4}) be a TM.

δ is defined by:

$\delta(q_0, a) = (q_1, x, R)$ $\delta(q_1, a) = (q_1, a, R)$

$\delta(q_1, y) = (q_1, y, R)$ $\delta(q_1, b) = (q_2, y, L)$

$\delta(q_2, a) = (q_2, a, L)$ $\delta(q_2, y) = (q_0, y, L)$

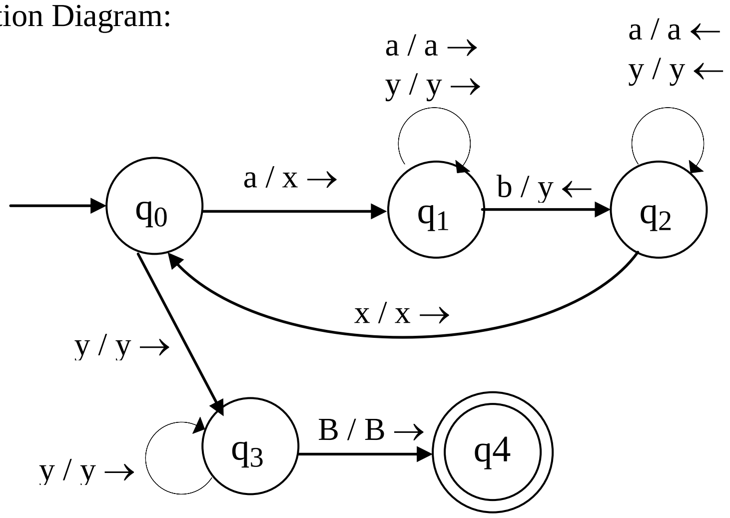
$\delta(q_2, x) = (q_0, x, R)$ $\delta(q_0, y) = (q_3, y, R)$

$\delta(q_3, y) = (q_3, y, R)$ $\delta(q_3, B) = (q_4, B, R)$

Transition Table:

States	Tape Symbols				
	a	b	x	y	B
→q_0	(q_1, x, R)	-	-	(q_0, y, R)	-
q_1	(q_1, a, R)	(q_2, y, L)	-	(q_1, y, R)	-
q_2	(q_2, a, L)	-	(q_0, x, R)	(q_1, y, L)	-
q_3	-	-	-	(q_1, y, R)	(q_4, B, R)
*q_4	-	-	-	-	

Transition Diagram:



i) Test whether the string w = “aabb” is in L(TM)

$\mathbf{q_0}$ aabbB ⊢ x$\mathbf{q_1}$abbB ⊢ xa$\mathbf{q_1}$bbB ⊢ x$\mathbf{q_2}$aybB ⊢ $\mathbf{q_2}$xaybB
⊢ x$\mathbf{q_0}$aybB ⊢ xx$\mathbf{q_1}$ybB ⊢ xxy$\mathbf{q_1}$bB ⊢ xx$\mathbf{q_2}$yyB
⊢ x$\mathbf{q_2}$xyyB ⊢ xx$\mathbf{q_0}$yyB ⊢ xxy$\mathbf{q_3}$yB ⊢ xxyy$\mathbf{q_3}$B
⊢ xxyyB$\mathbf{q_4}$

Hence the string is accepted.

i) Test whether the string w = “abbb” is in L(TM)

$\mathbf{q_0}$ abbbB ⊢ x$\mathbf{q_1}$bbbB ⊢ x$\mathbf{q_1}$bbbB ⊢ $\mathbf{q_2}$xybbB ⊢ x$\mathbf{q_0}$ybbB
⊢ xy$\mathbf{q_3}$bbB

Hence the string is rejected.

2. Design a TM to recognize the language L ={$a^n b^n c^n$; n>0}.

Solution:

The TM is designed using the following steps:

Step 1 : M replaces the leftmost ‘a’ by ‘x’ and moves right to the leftmost ‘b’, replacing it by ‘y’ and moves right to the leftmost ‘c’, replacing it by ‘z’.

Step 2 : Then M moves left to find the rightmost ‘x’ and moves one cell right to the leftmost ‘a’ and repeat the step 1.

Step 3 : While searching for a ‘b’ or ‘c’, if a blank (B) is encountered, and then M halts without accepting.

Step 4 : After changing a ‘b’ to ‘y’ and ‘c’ to ‘z’, if M finds no more a’s, then M checks no more b’s and c’s remains, M accepting the string else not.

Let M = ({$q_0, q_1, q_2, q_3, q_4, q_5$}, {a, b, c}, {a, b, c, B}, δ, q_0, B, {q_5}) be a TM.

δ is defined by:

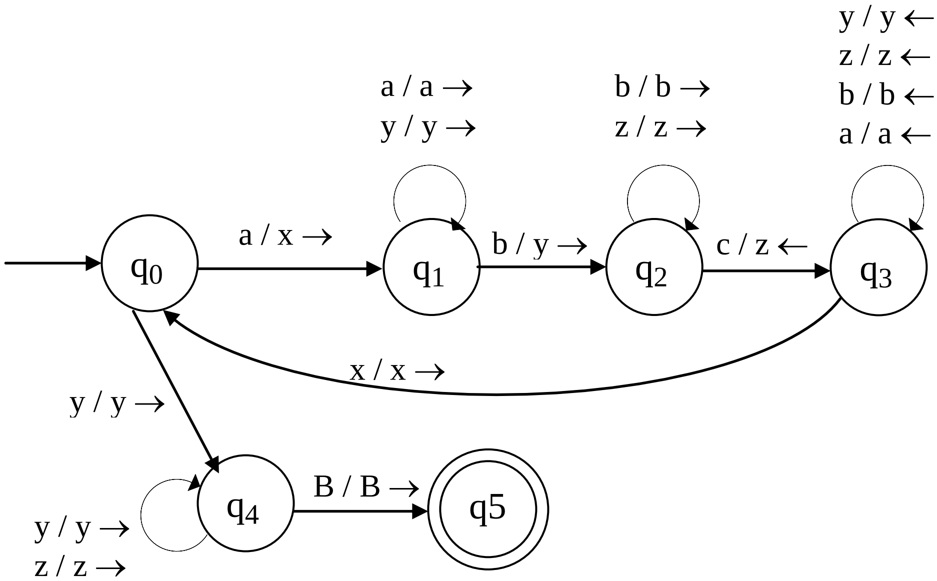
δ (q_0, a) = (q_1, x, R) δ (q_1, a) = (q_1, a, R)
δ (q_1, y) = (q_1, y, R) δ (q_1, b) = (q_2, y, R)
δ (q_2, b) = (q_2, b, R) δ (q_2, z) = (q_2, z, R)
δ (q_2, c) = (q_3, z, L) δ (q_3, z) = (q_3, z, L)
δ (q_3, b) = (q_3, b, L) δ (q_3, y) = (q_3, y, L)
δ (q_3, a) = (q_3, a, L) δ (q_3, x) = (q_0, x, R)
δ (q_0, y) = (q_4, y, R) δ (q_4, y) = (q_4, y, R)
δ (q_4, z) = (q_4, z, R) δ (q_4, B) = (q_5, B, R)

Transition Table:

States	Tape Symbols					
	a	b	c	x	y	z	B
→q_0	(q_1, x, R)	-	-	-	(q_4, y, R)	-	-
q_1	(q_1, a, R)	(q_2, y, R)	-	-	(q_1, y, R)	-	-
q_2	-	(q_2, b, R)	(q_3, z, L)	-	-	(q_2, z, R)	-
q_3	(q_3, a, L)	(q_3, b, L)	-	(q_0, x, R)	(q_3, y, L)	(q_3, z, L)	-

q_4	-	-	-	-	(q_4, y, R)	(q_4, z, R)	(q_5, B, R)
*q_5	-	-	-	-	-	-	-

Transition Diagram:



3. Design a TM to recognize the language L ={ww^R; w ∈ (0+1)*} and check whether the string “010010” is accept or not.

(or)

Design A TM to accept the set of palindrome strings and check whether the string “010010” is accept or not.

Solution

Let M = ({$q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7$}, {a, b, c}, {a, b, c, B}, δ, q_0, B, {q_7}) be a TM.

δ is defined by:

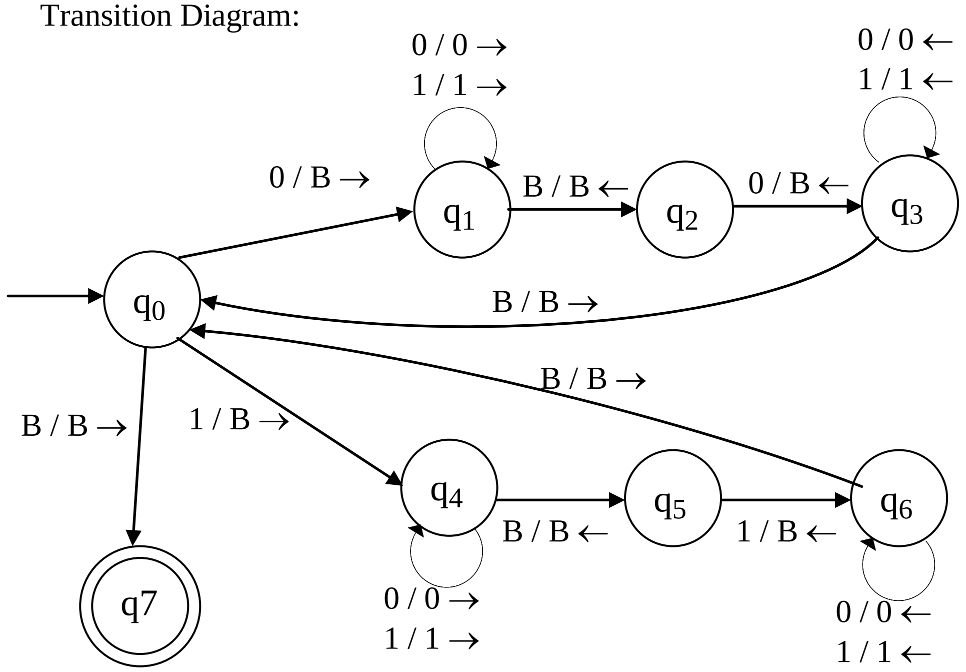
δ (q_0, 0) = (q_1, B, R)
δ (q_1, 0) = (q_1, 0, R)
δ (q_1, 1) = (q_1, 1, R)
δ (q_1, B) = (q_2, B, L)
δ (q_2, 0) = (q_3, B, L)
δ (q_3, 0) = (q_3, 0, L)
δ (q_3, 1) = (q_3, 1, L)
δ (q_3, B) = (q_0, B, R)
δ (q_0, B) = (q_8, B, R)

δ (q_0, 1) = (q_4, B, R)
δ (q_4, 0) = (q_4, 0, R)
δ (q_4, 1) = (q_4, 1, R)
δ (q_4, B) = (q_5, B, L)
δ (q_5, 1) = (q_6, B, L)
δ (q_6, 0) = (q_6, 0, L)
δ (q_6, 1) = (q_6, 1, L)
δ (q_6, B) = (q_0, B, R)

Transition Table:

States	Tape Symbols		
	0	1	B
→q_0	(q_1, B, R)	(q_4, B, R)	(q_8, B, R)
q_1	(q_1, 0, R)	(q_1, 1, R)	(q_2, B, L)
q_2	(q_3, B, L)	-	-
q_3	(q_3, 0, L)	(q_3, 1, L)	(q_0, B, R)
q_4	(q_4, 0, R)	(q_4, 1, R)	(q_5, B, L)
q_5	-	(q_6, B, L)	-
q_6	(q_6, 0, L)	(q_6, 1, L)	(q_0, B, R)
*q_7	-	-	-

Transition Diagram:



Test whether the string "010010" is in L(TM):

$\mathbf{q_0}010010B \vdash B\mathbf{q_1}10010B \vdash B1\mathbf{q_1}0010B \vdash B10\mathbf{q_1}010B$
$\vdash B100\mathbf{q_1}10B \vdash B1001\mathbf{q_1}0B \vdash B10010\mathbf{q_1}B$
$\vdash B1001\mathbf{q_2}0B \vdash B100\mathbf{q_3}1BB \vdash B10\mathbf{q_3}01BB$
$\vdash B1\mathbf{q_3}001BB \vdash B\mathbf{q_3}1001BB \vdash \mathbf{q_3}B1001BB$
$\vdash B\mathbf{q_0}1001BB \vdash BB\mathbf{q_4}001BB \vdash BB0\mathbf{q_4}01BB$
$\vdash BB00\mathbf{q_4}1BB \vdash BB001\mathbf{q_4}BB \vdash BB00\mathbf{q_5}1BB$
$\vdash BB0\mathbf{q_6}0BBB \vdash BB\mathbf{q_6}00BBB \vdash B\mathbf{q_6}B00BBB$
$\vdash BB\mathbf{q_0}00BBB \vdash BBB\mathbf{q_1}0BBB \vdash BBB0\mathbf{q_1}BBB$
$\vdash BBB\mathbf{q_2}0BBB \vdash BB\mathbf{q_3}BBBBB \vdash BBB\mathbf{q_0}BBBB$
$\vdash BBBB\mathbf{q_7}BBB$ - Hence the string is accepted.

4. Design a TM to recognize the language L ={wcw^R; w ∈ (a+b)*}.

Solution

Let M = ({$q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8$}, {a, b, c}, {a, b, c, B}, δ, q_0, B, {q_8}) be a TM.

δ is defined by:

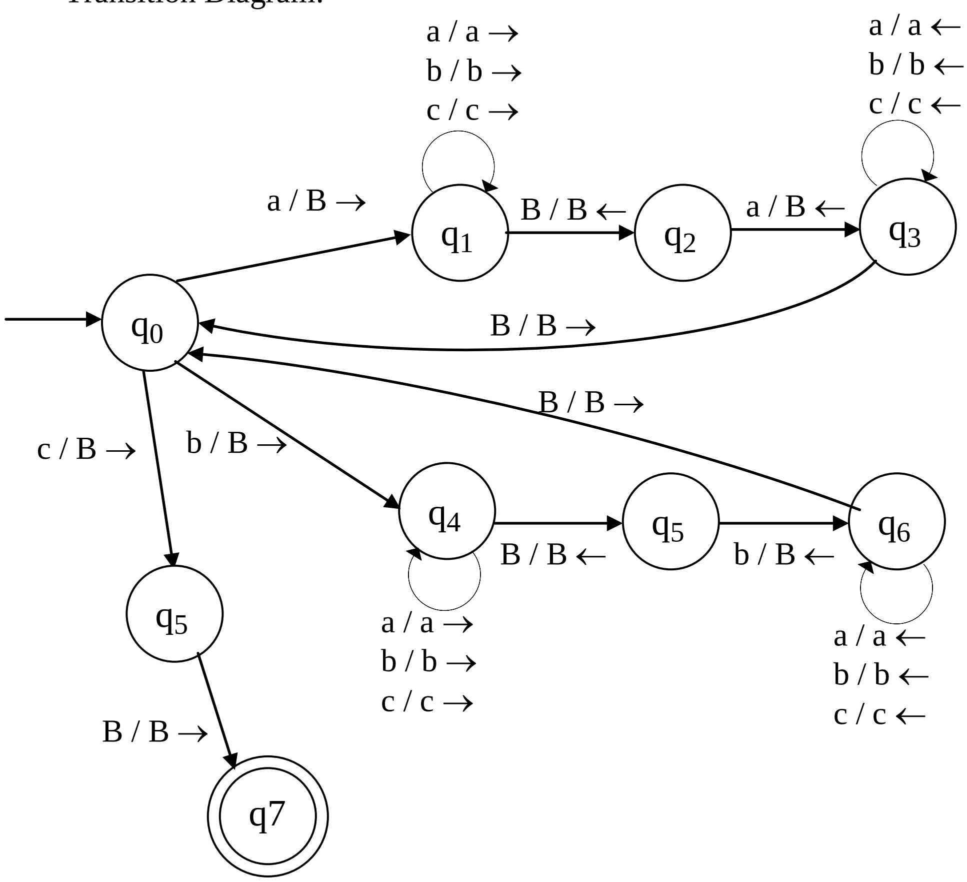
δ (q_0, a) = (q_1, B, R)
δ (q_1, a) = (q_1, a, R)
δ (q_1, b) = (q_1, b, R)
δ (q_1, c) = (q_1, c, R)
δ (q_1, B) = (q_2, B, L)
δ (q_2, a) = (q_3, B, L)
δ (q_3, a) = (q_3, a, L)
δ (q_3, b) = (q_3, b, L)
δ (q_3, c) = (q_3, c, L)
δ (q_3, B) = (q_0, B, R)
δ (q_0, c) = (q_7, B, R)

δ (q_0, b) = (q_4, B, R)
δ (q_4, a) = (q_4, a, R)
δ (q_4, b) = (q_4, b, R)
δ (q_4, c) = (q_4, c, R)
δ (q_4, B) = (q_5, B, L)
δ (q_5, b) = (q_6, B, L)
δ (q_6, a) = (q_6, a, L)
δ (q_6, b) = (q_6, b, L)
δ (q_6, c) = (q_6, c, L)
δ (q_6, B) = (q_0, B, R)
δ (q_7, B) = (q_8, B, R)

Transition Table:

States	Tape Symbols			
	a	b	c	B
→q_0	(q_1, B, R)	(q_4, B, R)	(q_7, B, R)	(q_8, B, R)
q_1	(q_1, 0, R)	(q_1, 1, R)	(q_1, c, R)	(q_2, B, L)
q_2	(q_3, B, L)	-	-	-
q_3	(q_3, 0, L)	(q_3, 1, L)	(q_3, c, L)	(q_0, B, R)
q_4	(q_4, 0, R)	(q_4, 1, R)	(q_4, c, R)	(q_5, B, L)
q_5	-	(q_6, B, L)	-	-
q_6	(q_6, 0, L)	(q_6, 1, L)	(q_6, c, L)	(q_0, B, R)
q_7	-	-	-	(q_8, B, R)
*q_8	-	-	-	-

Transition Diagram:



Tutorial Questions:

5. Design a TM to recognize the language L ={All strings must be equal number of 0's and 1's}.
6. Design a TM to accept the language L ={All strings must be odd number of a's}.
7. Design a TM to accept the language L ={ $a^n b^n c^n d^n$; n > 0}.
8. Design a TM to accept the language L ={ $a^n b^m c^m d^n$; m, n > 0}.
9. Design a TM to accept the language L ={ $a^n b^m$; n > 0 and m = n+2}.
10. Design a TM to accept the language L ={ $a^n bcd^n$; n > 0}.

Computable languages and functions

- ✓ A Turing machine computes a function f : Σ* → Σ* if, for any input word w, it always stops in a configuration where f(w) is on the tape.

- ✓ **Problems:**

1. Construct TM for concatenation of two strings of unary numbers.
   String 1 : 111 and String 2: 11

   Solution:

   Initial content in the tape:

B	1	1	1	0	1	1	B

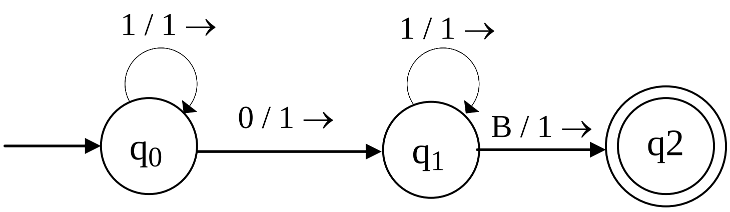
   Step 1 : M replaces the ‘0’ by ‘1’ and moves right to the leftmost ‘B’
   Step 2 : Move to step back, then M replaces the ‘1’ by ‘B’

   Final content in the tape after concatenation:

B	1	1	1	1	1	B	B

   Let M = ({q_0, q_1, q_2}, {1, 0}, {1, 0, B}, δ, q_0, B, {q_2}) be a TM.
   δ is defined by:
   δ (q_0, 1) = (q_0, 1, R)
   δ (q_0, 0) = (q_1, 1, R)
   δ (q_1, 1) = (q_1, 1, R)
   δ (q_1, B) = (q_2, 1, R)

   Transition Diagram:



2. Construct TM for f(x) = x + 3.

Solution:
Assume x = 5 (11111)

Initial content in the tape:

B	1	1	1	1	1	+	1	1	1	B

Step 3 : M replaces the '+' by '1' and moves right to the leftmost 'B'
Step 4 : Move to step back, then M replaces the '1' by 'B'

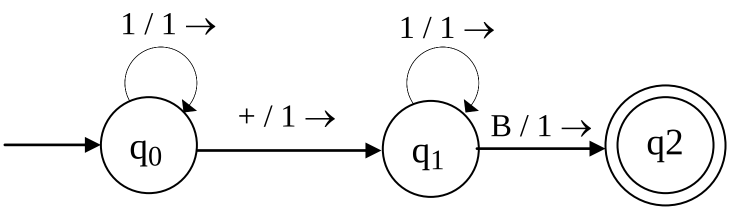
Final content in the tape after processing f(x) = x+3:

B	1	1	1	1	1	1	1	1	B	B

Let M = ({q_0, q_1, q_2}, {1, 0}, {1, 0, B}, δ, q_0, B, {q_2}) be a TM.
δ is defined by:

$\delta(q_0, 1) = (q_0, 1, R)$
$\delta(q_0, +) = (q_1, 1, R)$
$\delta(q_1, 1) = (q_1, 1, R)$
$\delta(q_1, B) = (q_2, 1, R)$

Transition Diagram:



3. Construct TM for f(x, y) = x + y.

Solution:
Assume x = 5 (11111) and y = 3 (111)

Initial content in the tape:

B	1	1	1	1	1	+	1	1	1	B

Step 5 : M replaces the '+' by '1' and moves right to the leftmost 'B'
Step 6 : Move to step back, then M replaces the '1' by 'B'

Final content in the tape after processing f(x, y) = x + y:

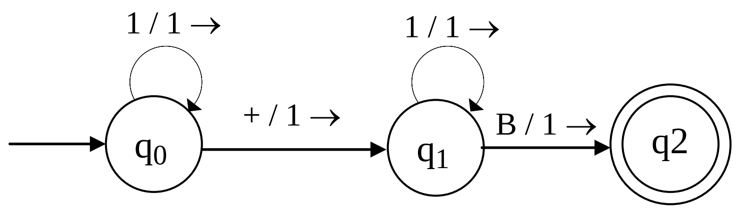
B	1	1	1	1	1	1	1	1	B	B

Let M = ({q_0, q_1, q_2}, {1, 0}, {1, 0, B}, δ, q_0, B, {q_2}) be a TM.

δ is defined by:

δ (q_0, 1) = (q_0, 1, R)
δ (q_0, +) = (q_1, 1, R)
δ (q_1, 1) = (q_1, 1, R)
δ (q_1, B) = (q_2, 1, R)

Transition Diagram:



4. Construct TM for f(x, y) = x – y; x ≥ y.

Solution:
Assume x = 5 (11111) and y = 3 (111)

Initial content in the tape:

B	1	1	1	1	1	-	1	1	1	B

Step 1 : M replaces the leftmost '1' by 'B' and moves right to the leftmost 'B'
Step 2 : Move to step back, then M replaces the '1' by 'B'
Step 3 : Do the step 1 and 2, until no more 1's after '-'
Step 4 : Finally M replaces the '-' by '1'

Final content in the tape after processing f(x, y) = x - y:

B	B	B	B	1	1	B	B	B	B	B

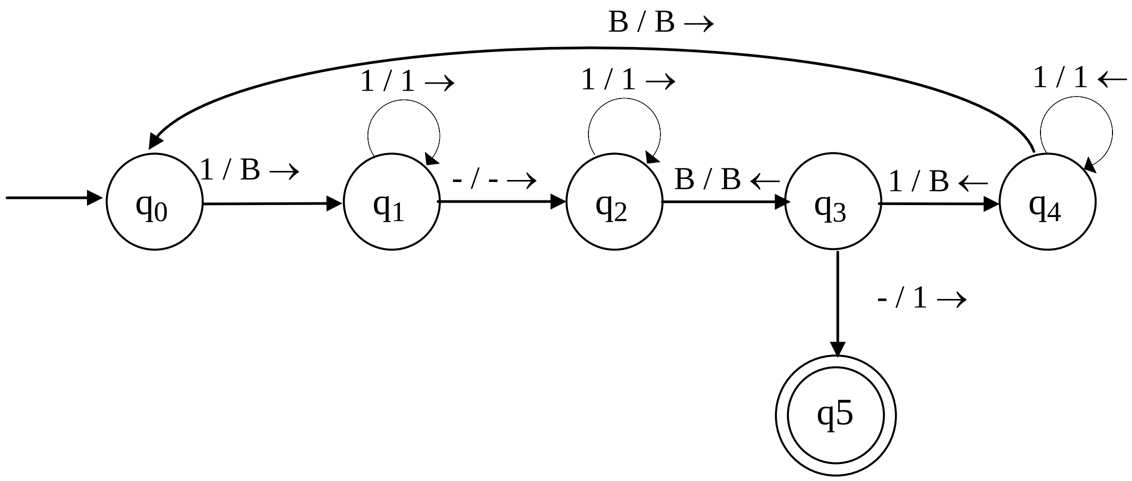
Let M = ({q_0, q_1, q_2}, {1, 0}, {1, 0, B}, δ, q_0, B, {q_2}) be a TM.

δ is defined by:

δ (q_0, 1) = (q_1, B, R)
δ (q_1, 1) = (q_1, 1, R)
δ (q_1, -) = (q_2, -, R)

$\delta(q_2, 1) = (q_2, 1, R)$
$\delta(q_2, B) = (q_3, B, L)$
$\delta(q_3, 1) = (q_3, B, L)$
$\delta(q_4, 1) = (q_4, 1, L)$
$\delta(q_4, B) = (q_0, B, R)$
$\delta(q_3, -) = (q_5, 1, R)$

Transition Diagram:



5. Design a TM to compute f(x, y) = x * y.

Solution:
Initial content in the tape:

B	1	1	1	*	1	1	0	B	B	B	B	B	B	B

Final content in the tape after processing f(x, y) = x * y:

B	X	X	X	*	Y	Y	0	1	1	1	1	1	1	B

Let $M = (\{q_0, q_1, q_2\}, \{1, 0\}, \{1, 0, B\}, \delta, q_0, B, \{q_2\})$ be a TM.
δ is defined by:

States	Tape symbols					
	0	1	X	Y	*	B
→ q_0	(q_1, X, R)	(q_4, X, R)	-	-	-	-
q_1	(q_1, 0, R)	(q_1, 1, R)	-	(q_1, Y, R)	(q_3, *, R)	(q_2, *, L)
q_2	(q_2, 0, L)	(q_2, 1, L)	(q_0, X, R)	(q_2, Y, L)	(q_2, *, L)	-
q_3	(q_3, 0, R)	-	-	-	-	(q_3, 0, L)
q_4	-	(q_5, X, R)	-	(q_4, Y, R)	-	-

q_5	$(q_6, 0, L)$	-	-	-	$(q_7, 0, L)$	-
q_6	-	$(q_6, 1, L)$	$(q_6, 0, L)$	(q_6, Y, L)	-	(q_0, B, R)
$*q_7$	-	(q_7, B, L)	(q_7, B, L)	(q_7, B, L)	-	-

Modifications of Turing machine

Turing Machines with Two Dimensional Tapes

This is a kind of Turing machines that have one finite control, one read-write head and one two dimensional tape. The tape has the top end and the left end but extends indefinitely to the right and down. It is divided into rows of small squares. For any Turing machine of this type there is a Turing machine with a one dimensional tape that is equally powerful, that is, the former can be simulated by the latter.

To simulate a two dimensional tape with a one dimensional tape, first we map the squares of the two dimensional tape to those of the one dimensional tape diagonally as shown in the following tables:

v	v	v	v	v	v	v	. . .	. . .
h	1	2	6	7	15	16	. . .	. . .
h	3	5	8	14	17	26	. . .	. . .
h	4	9	13	18	25	. . .	. . .	. . .
h	10	12	19	24	. . .	. . .	. . .	. . .
h	11	20	23	. . .	. . .	. . .	. . .	. . .
h	21	22	. . .	. . .	. . .	. . .	. . .	. . .
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .

One Dimensional Tape

v	1	v	2	3	h	4	5	6	V	7	8	9	10	h	11	. . .	. . .

The head of a two dimensional tape moves one square up, down, left or right. Let us simulate this head move with a one dimensional tape. Let ***i*** be the head position of the two dimensional tape.

Multitape TM

A multi-tape Turing machine is like an ordinary Turing machine with several tapes. Each tape has its own head for reading and writing. Initially the input appears on tape 1, and the others start out blank.

Universal TM

Universal Turing machine (UTM) is a Turing machine that can simulate an arbitrary Turing machine on arbitrary input.

Turing Machines with Multiple Tapes :

This is a kind of Turing machines that have one finite control and more than one tapes each with its own read-write head. It is denoted by a 5-tuple (Q , Σ , Γ , q_0, δ) . Its transition function is a partial function

δ : Q x (Γ $\cup$ { Δ } $)^n$ -> (Q $\cup$ { h }) x (Γ $\cup$ { Δ } $)^n$ x { R , L , S $\}^n$.
A configuration for this kind of Turing machine must show the current state the machine is in and the state of each tape.

Turing Machines with Multiple Heads :
This is a kind of Turing machines that have one finite control and one tape but more than one read-write heads. In each state only one of the heads is allowed to read and write. It is denoted by a 5-tuple (Q , Σ , Γ , q_0, δ). The transition function is a partial function

δ : Q x { H_1 , H_2 ... , H_n } x (Γ $\cup$ { Δ }) -> (Q $\cup$ { h }) x (Γ $\cup$ { Δ } x { R , L , S }

where H_1 , H_2 ... , H_n denote the tape heads.

Turing Machines with Infinite Tape :
This is a kind of Turing machines that have one finite control and one tape which extends infinitely in both directions. It turns out that this type of Turing machines are only as powerful as one tape Turing machines whose tape has a left end.

Nondeterministic Turing Machines
A nondeterministic Turing machine is a Turing machine which, like nondeterministic finite automata, at any state it is in and for the tape symbol it is reading, can take any action selecting from a set of specified actions rather than taking one definite predetermined action. Even in the same situation it may take different actions at different times. Here an action means the combination of writing a symbol on the tape, moving the tape head and going to a next state. For example let us consider the language L = { ww : w $\in$ { a , b $\}^*$ } .

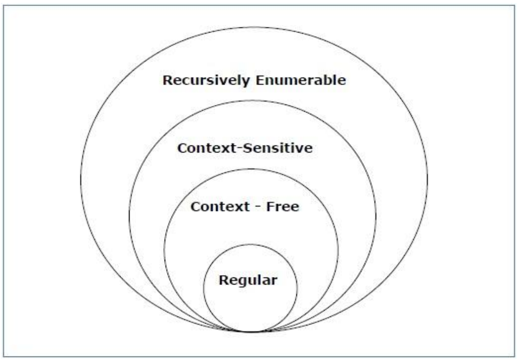
Chomsky hierarchy of languages & Grammars and their machine recognizers

✓ **Chomsky Hierarchy (Types of grammars)**

Class	**Chomsky hierarchy of languages**	**Grammars and their machine recognizers**		Rules
Type-0	Recursively enumerable Language	Unrestricted Grammar	Turing machine	Rules are of the form: $\alpha \to \beta$, where α and β are arbitrary strings over a vocabulary V and $\alpha \neq \varepsilon$
Type-1	Context-sensitive Language	Context-sensitive Grammar	Linear-bounded automaton	Rules are of the form: $\alpha A\beta \to \alpha B\beta$ or $S \to \varepsilon$ where $A, S \in N$ $\alpha, \beta, B \in (N \cup T)^*$ $B \neq \varepsilon$
Type-2	Context-free Language	Context-free Grammar	Pushdown automaton	Rules are of the form: $A \to \alpha$ where $A \in N$, $\alpha \in (N \cup T)^*$
Type-3	Regular Language	Regular Grammar	Finite automaton	Rules are of the form: $A \to \varepsilon$ $A \to \alpha$ $A \to \alpha B$ where $A, B \in N$ and $\alpha \in T$

- ✓ **Scope of each type of grammar**

A figure shows the scope of each type of grammar:



- ✓ **Type - 3 Grammar**
  - Type-3 grammars generate regular languages. Type-3 grammars must have a single non-terminal on the left-hand side and a right-hand side consisting of a single terminal or single terminal followed by a single non-terminal.
  - The productions must be in the form

    X → a

    X → aY

    where X, Y ∈ N (Non terminal) and a ∈ T (Terminal)
  - The rule S → ε is allowed if S does not appear on the right side of any rule.
  - Example

    X → ε

    X → a | aY

    Y → b

- ✓ **Type - 2 Grammar**
  - Type-2 grammars generate context-free languages. These languages generated by these grammars are be recognized by a non-deterministic pushdown automaton.
  - The productions must be in the form

    A → γ

    where A ∈ N (Non terminal) and γ ∈ (T ∪ N)* .
  - Example

    S → X a

    X → a

    X → aX

    X → abc

    X → ε

- ✓ **Type - 1 Grammar**
  - Type-1 grammars generate context-sensitive languages.
  - The productions must be in the form

    α A β → α γ β

    Where A ∈ N (Non-terminal) and α, β, γ ∈ (T ∪ N)*
  - The strings α and β may be empty, but γ must be non-empty.

- The rule S → ε is allowed if S does not appear on the right side of any rule. The languages generated by these grammars are recognized by a linear bounded automaton.
- Example
  AB → AbBc
  A → bcA
  B → b

✓ **Type - 0 Grammar**

- Type-0 grammars generate recursively enumerable languages. The productions have no restrictions. They are any phase structure grammar including all formal grammars.
- They generate the languages that are recognized by a Turing machine.

- The productions can be in the form of
  α → β
  where α is a string of terminals and non-terminals with at least one non-terminal and α cannot be null. β is a string of terminals and non-terminals.
- Example
  S → ACaB
  Bc → acB
  CB → DB
  aD → Db

Undecidability

Phrase Structure Grammar

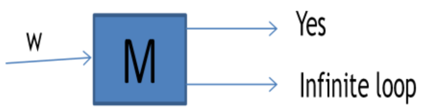
✓ It consists of four components **G = (V, T, P, S)**

Recursive Language

✓ A language is recursive if there exists a Turing Machine that accepts every string of the language and reject every string that is not in the language.

Recursively Enumerable Language

✓ A language is recursive enumerable if there exists a Turing Machine that accepts every string of the language and does not accept strings that are not in the language. The strings that are not in the language may be rejected and it may cause the TM to go to an infinite loop.



Decidability

✓ A language is decidable (recursive) if and only if there is a TM M such that M accepts every string in L and rejects every string not in L (or)

✓ A problem whose language is recursive is said to be a decidable.

<u>Example :</u>

- The strings over {a,b} that consists of alternating a's and b's.
- The strings over {a,b} that contains an equal number of a's and b's

Undecidability

- ✓ A problem is undecidable, if there is no algorithm, that can take as input an instance of the problem and determines whether the answer to that instance is 'Yes' or 'No'.

Example :

- Given a TM M and an input string w, does M halt on input w? (Halting Problem)
- For a fixed machine M, given an input string w, does M halt on input w?
- Membership problem is undecidable.
- State entry problem is undecidable.

Properties of Recursive and Recursively Enumerable Languages

- ✓ Complement of a recursive language is recursive.
- ✓ Union of two recursive languages is recursive.
- ✓ Union of two recursive enumerable languages is also recursively enumerable.
- ✓ L if L and complement of L (L) are recursively enumerable is recursive.

Theorem :

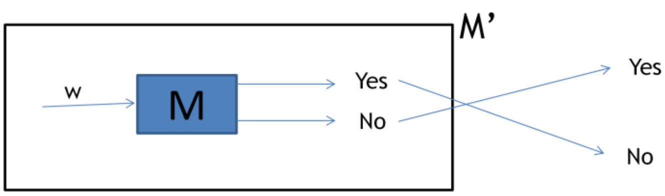
The Complement of recursive language is recursive.

Proof :

Let L be a recursive language. Then there exists a TM M that halts on every string on L.

$$L = \sum^* - L$$

Since L is recursive there is an "algorithm" (TM M) to accept L. Now construct an "algorithm" (TM M') for L is as follows.



If M halts without accepting the string, then M' halts accepting that string and if M halts on accepting it, M' enters into the final state without accepting it.

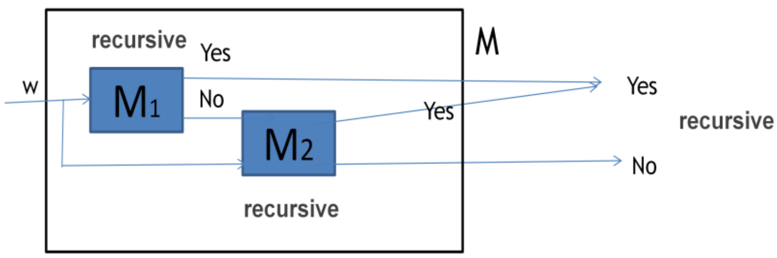
Clearly L(M') is the complement of L and thus L is a recursive language.

Theorem :

If L_1 and L_2 are two recursive languages then L_1 U L_2 is also a recursive language.

Proof :

- ✓ Let L_1 and L_2 be recursive languages accepted by the TMs M_1 and M_2 respectively.
- ✓ Construct a new TM M which first simulates M_1. If M_1 accepts, then M accepts. If M_1 reject, the simulates M_2 and accepts if and only if M_2 accepts.
- ✓ Thus M has both accepting and rejecting criterion. So, M accepts L_1 U L_2.



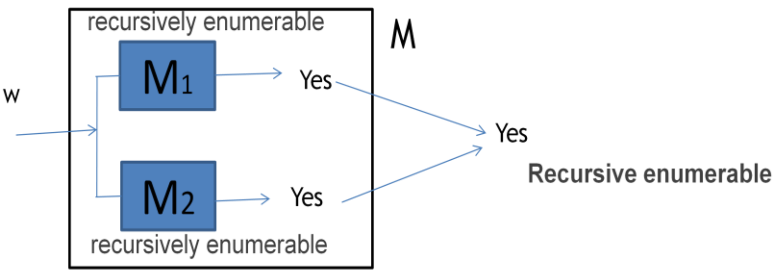
Dr. D. Jagadeesan, B.E., M.Tech., Ph.D., MISTE
Professor in CSE,

Theorem :

If L_1 and L_2 are two recursively enumerable languages then $L_1 \cup L_2$ is also a recursively enumerable language.

Proof :

- ✓ Let L_1 and L_2 be recursively enumerable languages accepted by the TMs M_1 and M_2 respectively.
- ✓ Construct a new TM M which simultaneously simulates M_1 and M_2 on different tapes.
- ✓ If M_1 or M_2 accepts, the M accepts.

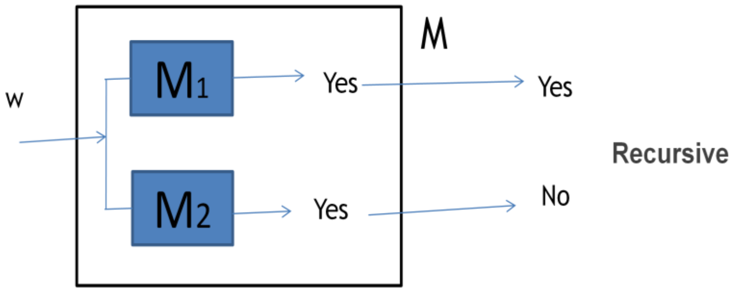


Theorem :

L if L and complement of L (L) are recursively enumerable is recursive.

Proof :

- ✓ Let M**1** and M2 be the TMs designed for the languages L and L respectively.
- ✓ Construct a new TM M which simulates M1 and M2 simultaneously.
- ✓ If M accepts w if M1 accepts w, M rejects w if M2 accepts w.



Post Correspondence Problem (PCP)

The Post Correspondence Problem (PCP), introduced by Emil Post in 1946, is an undecidable decision problem. The PCP problem over an alphabet ∑ is stated as follows −

Given the following two lists, **M** and **N** of non-empty strings over ∑ −

$M = (x_1, x_2, x_3, \ldots\ldots\ldots, x_n)$

$N = (y_1, y_2, y_3, \ldots\ldots\ldots, y_n)$

We can say that there is a Post Correspondence Solution, if for some $i_1, i_2, \ldots\ldots\ldots\ldots\ i_k$, where $1 \leq i_j \leq n$, the condition $x_{i1} \ldots\ldots x_{ik} = y_{i1} \ldots\ldots y_{ik}$ satisfies.

Example:

Find whether the lists M = (abb, aa, aaa) and N = (bba, aaa, aa) have a Post Correspondence Solution?

Solution

	x_1	x_2	x_3
M	Abb	aa	aaa
N	Bba	aaa	aa

Here,

$x_2x_1x_3$ = 'aaabbaaa' and **$y_2y_1y_3$ = 'aaabbaaa'**

We can see that

$x_2x_1x_3 = y_2y_1y_3$

Hence, the solution is **i = 2, j = 1, and k = 3.**

Modified Post Correspondence Problem

- ✓ We have seen an undecidable problem, that is, given a Turing machine M and an input w, determine whether M will accept w (universal language problem).
- ✓ We will study another undecidable problem that is not related to Turing machine directly.
- ✓ Given two lists A and B:

  $A = w_1, w_2, \ldots, w_k \quad B = x_1, x_2, \ldots, x_k$

  The problem is to determine if there is a sequence of one or more integers $i_1, i_2, \ldots, i_m$ such that:

  $w_1w_{i1}w_{i2}\ldots w_{im} = x_1x_{i1}x_{i2}\ldots x_{im}$

  (w_i, x_i) is called a corresponding pair.

- ✓ Example

	A	B
i	w_i	x_i
1	11	1
2	1	111
3	0111	10
4	10	0

This MPCP instance has a solution: 3, 2, 2, 4:

$w_1w_3w_2w_2w_4 = x_1x_3x_2x_2x_4 = 1101111110$