Finite state machine (FSM):

A FSM is defined as:

$$M = (S; I, O, F, G)$$

where

- S = finite set of states
- I = finite set of inputs
- O = finite set of outputs
- F = state rel^n or transition fxns
- G = output rel^n

- A state rel^n is of the form-
  (current state, input) → next state

- A output rel^n is of the form-
  (current state, input) → (next state, output)

eg: ON/OFF switch as a FSM.

$S = \{ON, OFF\}$

$I = \{press\}$

$O = \{fan\ ON, fan\ OFF\}$

F consists of:
(ON, press) → OFF
(OFF, press) → ON

G consists of:
(ON, press) → (OFF, FANOFF)
(OFF, press) → (ON, Fan ON)

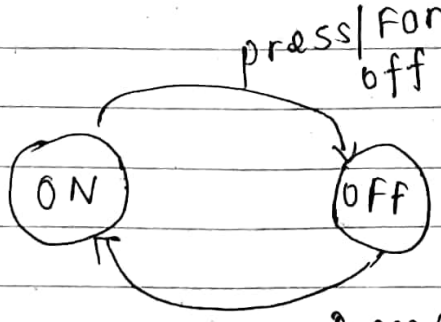
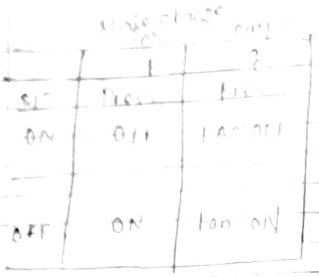


Fig: state diagram or transition diagram



	1	?
SL		
ON	OFF	
OFF	ON	

Finite state automata:

Alphabet:
- Collection of input symbols
- It is denoted by Σ
eg: binary alphabet, $\Sigma = \{0, 1\}$

String:
- combination or multiple occurence of input symbols.
- It is denoted by ω
- eg $\Sigma = \{0, 1\}$, $\omega = 1101\ 0$

Language
- collection of all possible string over some given alphabet.
- It is denoted by L.
eg: $\Sigma = \{0, 1\}$, $L = \{0, 1, 00, 01, 010, 1101, \ldots\}$

Finite state automata (FSM without output)

A finite state automata (FSA) is a mathematical model that is used to determine whether a particular string is valid or invalid. If a string is valid then it is said to be accepted, otherwise rejected. Therefore, a FSA is also called language reconizer.

Mathematically, a FSA is defined as:

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

where,

Q = finite set of states
Σ = finite sets of inputs
δ = transition fxn, that takes two argument in the form of state input and gives an argument in the form of state. A transition fxn is of the form

$$\boxed{Q \times \Sigma \rightarrow Q}$$

q_0 = initial or start state
F = set of final or accepting sites

eg: $Q = \{q_1, q_2, q_3\}$
$\Sigma = \{0, 1\}$
δ consists of

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

$q_0 = q_1$
$F = q_2$

Representation of FSA :
1. Transition diagram
2. Transition table

Transition diagram :
- directed, weighted graph.
- states are represented by vertices & transition from one state to another is represented by a directed edge from one vertex to another
- Inputs are represented by the weights provided to each edge.
- Initial state is represented by pointing an arrow towards it.
- Finite state is represented by double or concentric circle.

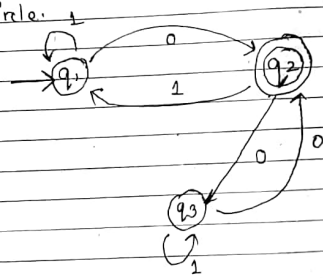


Fig : transition diagram

Transition table :-
- It is the tabular representation of FSA.
- Generally, states are arranged in rows and inputs are arranged in columns.
- Each cell of the table represents next state.
- Initial state is represented by pointing an arrow towards it.
- final state is represented by enclosing it in a circle or placing star (asterisk) sign infront of it.

Q \ Σ	0	1
→ q1	q2	q1
* q2	q3	q1
q3	q2	q3

Fig : Transition table

Processing of a string by FSA.

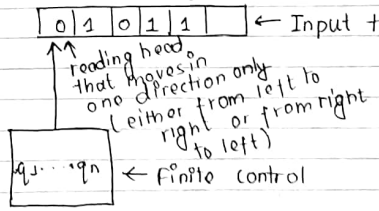


Fig : Block diagram of a FSA

A FSA consists of

1) Input tape
2) Reading head
3) Finite control

Input tape of a FSA is divided into number of square cells and each cell is responsible for holding the symbols of input string. Reading head is responsible for reading the content of each cell at a time in desired direction ie either from left to right or from right to left.

Finite control is used to control the processing of FSA ie the initial state, intermediate states must be the states desired in finite control. Initially, the reading head is placed at the leftmost cell of the input tape. Then, it reads each cell at a time from left to right. We know that anytime consuming a symbol, FSA regularly changes its states. During the process of consuming inputs symbols when an empty string is encountered i.e string terminates if finite control gives any of the designed final states as next state, the string is said to be accepted otherwise rejected.

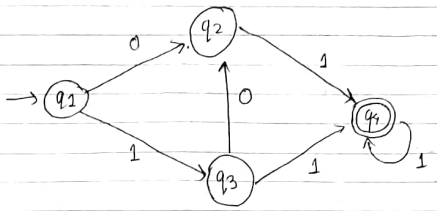


Fig: Finite automata

check whether the string $w = 100011$ is accepted by the above FA or not.

$\delta(q_1, 100011) \rightarrow \delta(q_3, 00011)$
$\rightarrow \delta(q_2, 0011)$
$\rightarrow \delta(q_2, 011)$
$\rightarrow \delta(q_2, 11)$
$\rightarrow \delta(q_4, 1)$
$\rightarrow \delta(q_4, \epsilon)$

Since, q_4 is the final state
$w = 100011$ is accepted.

eg: $w = 1000$

$\delta(q_1, 1000) \rightarrow \delta(q_3, 000)$
$\rightarrow \delta(q_2, 00)$
$\rightarrow \delta(q_2, 0)$
$\rightarrow \delta(q_2, \epsilon)$

since, q_2 is not a final state $w = 1000$ is rejected

Types of finite automata :

1. Deterministic finite automata (DFA)
2. Non-deterministic finite automata (NFA)

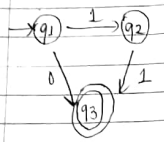


Fig : DFA

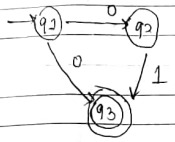
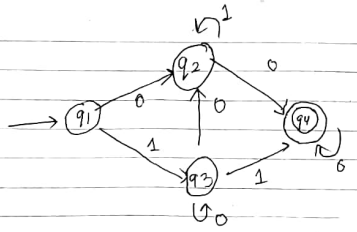


Fig : NFA



w = 10001

Conversion of NFA to DFA :

- To convert given NFA to its equivalent DFA, we use sub-set construction method

Steps :

1. Construct all the subsets by using all the states of given NFA.
2. Construct the transition table of the given NFA
3. Transition table gives next states as old states and new states.
4. Identify transitions for all the new states interms of given inputs.
5. If a new state generates further new state then identify transition for this new state as well
6. Continue the process until transitions for all new states are identified.
7. Finally, draw a transition diagram

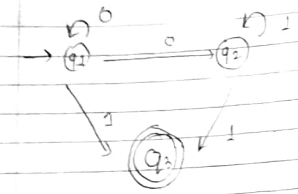


Fig 1 NFA

All possible sets are : $\{\phi\}, \{q_1\}, \{q_2\}, \{q_3\}, \{q_1, q_2\}, \{q_1, q_3\}, \{q_2, q_3\}, \{q_1, q_2, q_3\}$

The transition table of given NFA is

		0	1
→	q_1	$\{q_1, q_2\}$	$\{q_3\}$
*	q_3	q_3	ϕ
	q_2	ϕ	$\{q_2, q_3\}$

Let δ' be the transition fxn of the DFA being formed

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

$\delta'(\{q_3\}, 1) \rightarrow \delta(q_3, 1) \rightarrow \{q_3\}$ - old state

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

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

$\rightarrow \{q_1, q_2\}$

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

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

$\rightarrow \{q_2, q_3\}$ - new state

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

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

$\rightarrow \{q_3\}$

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

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

$\rightarrow \{q_2, q_3\}$

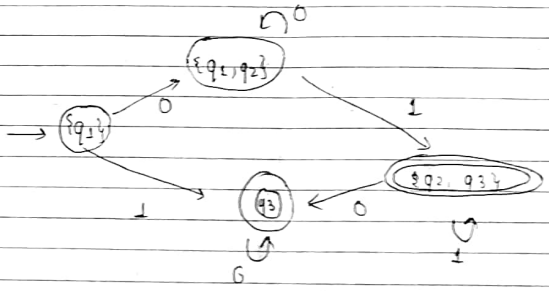


Fig 2 DFA

2013 Spring (Q) (E)

Convert the NFA $M=(\{q_0, q_1, q_2, q_3\}, \{0,1\}, \delta, q_0, \{q_2\})$ to its equivalent DFA δ is given by.

Q|	0	1
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\{q_2\}$	q_1
q_2	$\{q_3\}$	$\{q_3\}$
q_3	$\{\phi\}$	$\{q_2\}$

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

$\delta'(\{q_0\}, 0) = \delta\{q_0, 0\} \rightarrow \{q_0, q_1\}$ (New state) (II)

$\delta'(\{q_0\}, 1) \rightarrow \delta(q_0, 1) \rightarrow \{q_0\}$

$\delta'(\{q_0, q_1\}, 0) \rightarrow \delta'(q_0, 0) \cup \delta(q_1, 0)$

$\{q_0, q_1\} \cup \{q_2\}$

$\{q_0, q_1, q_2\}$ New state (III)

$\delta'(\{q_0, q_1\}, 1) \rightarrow \delta'(q_0, 1) \cup (q_1, 1)$

$\{q_0\} \cup \{q_1\}$

$\{q_0, q_1\}$

$\delta'(\{q_0, q_1, q_2\}, 0) \rightarrow \delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)$

$\{q_0, q_1\} \cup \{q_2\} \cup \{q_3\}$

$\{q_0, q_1, q_2, q_3\}$ New state (IV)

$\delta'\{q_0, q_1, q_2\}, 1) \rightarrow \delta'(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1\}$

$\Rightarrow \{q_0, q_2, q_3\}$ New state (V)

$\delta'(\{q_0, q_1, q_2, q_3\}, 0) \rightarrow \delta'(q_0, 0) \cup (q_1, 0), \{q_2, 0) (q_3, 0)$

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

$\delta'(\{q_0, q_1, q_2, q_3\}, 1) \rightarrow \delta'(q_0, 1) \cup (q_1, 1) \cup (q_2, 1) \cup (q_3, 1)$

$\Rightarrow \{q_0\} \cup \{q_1\} \cup \{q_3\} \cup \{q_2\}$

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

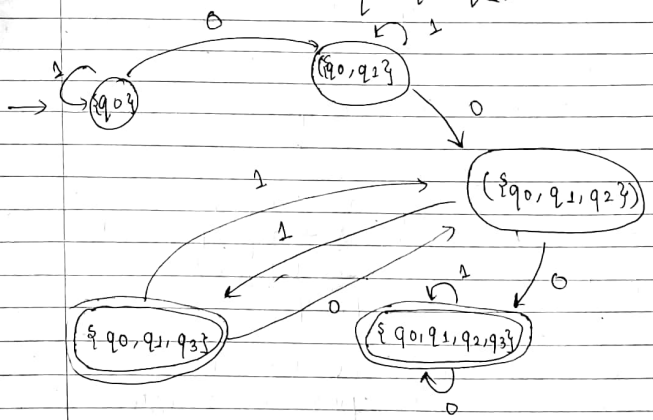
$\delta'\{q_0, q_1, q_3\}, 0\} \rightarrow \delta'(q_0, 0) \cup (q_1, 0) \cup (q_3, 0)$

$= \{q_0, q_1, q_2, \phi\}$

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

$\delta(\{q_0, q_1, q_3\}, 1\} \rightarrow \delta'(q_0, 1) \cup (q_1, 1) \cup (q_3, 1\}$

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



#

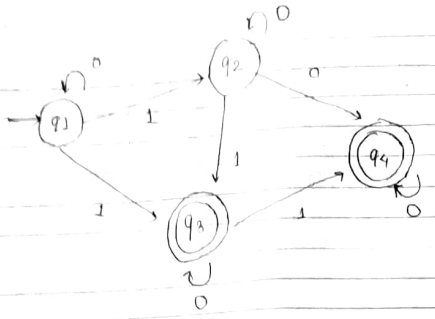


Fig: NFA

All possible sets are: $\{\phi\}, \{q_1\}, \{q_2\}, \{q_3\}, \{q_4\}$
$\{q_1, q_2\}, \{q_1, q_3\}, \{q_1, q_4\}, \{q_2, q_3\}, \{q_2, q_4\}$
$\{q_3, q_4\}, \{q_1, q_2, q_3\}$

Q	0	1
→ q_1	$\{q_1\}$	$\{q_2, q_3\}$ $\{q_1, q_3, q_4\}$,
q_2	$\{q_2, q_4\}$	$\{q_3\}$ $\{q_1, q_2, q_4\}$
* q_3	$\{q_3\}$	$\{q_4\}$ $\{q_1, q_3, q_2, q_4\}$
* q_4	$\{q_4\}$	$\{\phi\}$ $\{q_2, q_3, q_4\}$

Let δ' be the transition function of DFA being formed

$\delta'(\{q_1\}, 0) = \delta(q_1, 0) \rightarrow \{q_1\}$
$\delta'(\{q_1\}, 1) = \delta(q_1, 1) \rightarrow \{q_2, q_3\}$ → New state
$\delta'(\{q_2, q_3, 0\} = \delta(q_2, 0) \cup (q_3, 0)$
$\{q_2, q_4, q_3\}$ → New state

$\delta'\{(q_2, q_3, 1)\} = \delta(q_2, 1) \cup \delta(q_3, 1)$
$= \{q_3, q_4\}$ → New state

$\delta'\{q_2, q_3, q_4, 0\} = \delta(q_2, 0) \cup \delta(q_3, 0) \cup \delta(q_4, 0)$
$= \{q_2, q_4\} \cup \{q_3\} \cup \{q_4\}$
$= \{q_2, q_3, q_4\}$

$\delta'\{q_2, q_3, q_4, 1\} = \delta(q_2, 1) \cup \delta(q_3, 1) \cup \delta(q_3, 1)$
$= \{q_3\} \cup \{q_4\} \cup \{\phi\}$
$= \{q_3, q_4\}$

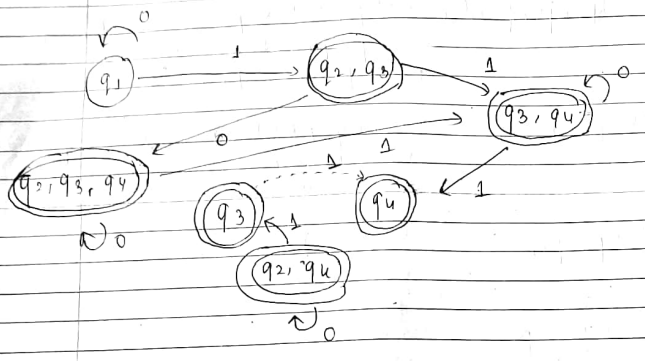
$\delta'\{(q_3, q_4, 0)\} = \delta(q_3, 0) \cup \delta(q_4, 0)$
$= \{q_3, q_4\}$

$\delta'\{(q_3, q_4, 1)\} = \delta(q_3, 1) \cup \delta(q_4, 1)$
$= \{q_4\} \cup \{\phi\}$
$= \{q_4\}$.

q_1 q_2, q_3 q_4 q_3, q_4 q_2, q_4, q_3

$\delta'\{(q_2, q_4), 0\} = \delta(q_2, 0) \cup \delta(q_4, 0)$
$= \{q_2, q_4\} \cup \{q_4\}$
$= \{q_2, q_4\}$

$\delta'\{(q_2, q_4), 1\} = \delta(q_2, 1) \cup \delta(q_4, 1)$
$= \{q_3\} \cup \{\phi\}$
$= \{q_3\}$



since $\{q_3\}$ and $\{q_2, q_4\}$ are not reachable from start state, so they are discarded and it gives

Epsilon-NFA (ϵ-NFA):

- A FA that changes its state by consuming empty string (ϵ) i.e without consuming any valid input is called epsilon-NFA.

eg :

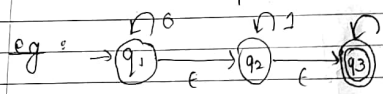


Fig :- ϵ-NFA

Epsilon closure

ϵ-closure $(q_1) = \{q_1, q_2, q_3\}$ → new
ϵ-closure $(q_2) = \{q_2, q_3\}$ → new
ϵ-closure $(q_3) = \{q_3\}$ → old.

Conversion of ϵ-NFA to DFA (elimination of empty moves):-

- To eliminate empty moves from a given ϵ-NFA, we need to calculate epsilon-closure of every state of the given ϵ-NFA.
- epsilon-closure of a particular state is defined as the continuous path that is labelled with ϵ.
- epsilon-closure of every state gives states in the form of old states and new states
- Then, we identify the transitions for all the new states.
- Transition is calculated as epsilon closure of the new state.
- After calculating transitions for all the new states, we draw a transition diagram, which is the required FA without empty moves.

eg :

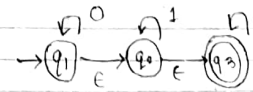


Fig :- ϵ-NFA

Here,
ϵ-Closure(q_1) → $\{q_1, q_2, q_3\}$
ϵ-closure(q_2) → $\{q_2, q_3\}$
ϵ-closure(q_3) → $\{q_3\}$

Let δ' be the transition function of the FA being formed.
Here, $\{q_1, q_2, q_3\}$ and $\{q_2, q_3\}$ are new states.

$\delta(\{q_1, q_2, q_3\}, 0) \rightarrow \epsilon$-closure($\delta(\{q_1, q_2, q_3\}, 0)$)
$\rightarrow \epsilon$-closure($\delta(q_1, 0) \cup \delta(q_2, 0) \cup \delta(q_3, 0)$)
$\rightarrow \epsilon$-closure($q_1 \cup \phi \cup q_3$)
$\rightarrow \epsilon$-closure($q_1 \cup q_3$)
$\rightarrow \epsilon$-closure(q_1) $\cup \epsilon$-closure(q_3)
$\rightarrow \{q_1, q_2, q_3\} \cup \{q_3\}$
$\rightarrow \{q_1, q_2, q_3\}$

$\delta'(\{q_1, q_2, q_3\}, 1) \rightarrow \epsilon$-closure($\delta\{q_1, q_2, q_3\}, 1$)
$\rightarrow \epsilon$-closure($\delta(q_1, 1) \cup \delta(q_2, 1) \cup \delta(q_3, 1)$)
$\rightarrow \epsilon$-closure($\phi \cup q_2 \cup \phi$)
$\rightarrow \epsilon$-closure(q_2)
$\rightarrow \{q_2, q_3\}$

$\delta'(\{q_2, q_3\}, 0) \rightarrow \epsilon$-closure($\delta\{q_2, q_3\}, 0$)
$\rightarrow \epsilon$-closure($\delta(q_2, 0) \cup \delta(q_3, 0)$)
$\rightarrow \epsilon$-closure($\phi \cup q_3$)
$\rightarrow \epsilon$-closure(q_3)
$\rightarrow \{q_3\}$

$\delta'(\{q_2, q_3\}, 1) \rightarrow \epsilon$-closure($\delta(q_2, q_3), 1$)
$\rightarrow \epsilon$-closure($\delta(q_2, 1) \cup \delta(q_3, 1)$)
$\rightarrow \epsilon$-closure($q_2 \cup \phi$)
$\rightarrow \{q_2, q_3\}$

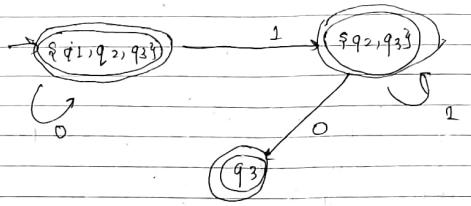
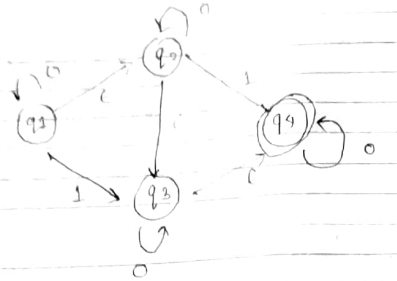


Fig: DFA



Epsilon closure

ε-closure(q_1) = ~~$\{q_1, q_3\}$~~ $\{q_2, q_3, q_4\}$

(q_2) = $\{q_3, q_4\}$

(q_3) = $\{q_4\}$

(q_4) = $\{q_4\}$

Let δ' be the transition function of the FA being formed.

$\delta'[\{q_1, q_2, q_3, q_4\}, 0) \rightarrow \epsilon\text{-closure}(\delta\{q_1, q_2, q_3, q_4\}, 0))$

$= \epsilon\text{-closure}(\delta(q_2, 0) \cup \delta(q_3, 0) \cup \delta(q_4, 0))$

$= \epsilon\text{-closure}(q_2 \cup q_3 \cup q_4)$

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

$\delta'(\{q_1, q_2, q_3, q_4\}, 1) \rightarrow \epsilon\text{-closure}(\delta(q_2, 1) \cup \delta(q_3, 1) \cup \delta(q_4, 1))$

$= \epsilon\text{-closure}(q_4, \phi, \phi)$

$= q_4$

$= \epsilon\text{-closure}(q_3 \cup q_4 \cup \phi \cup \phi)$

$= \{q_3, q_4\}$ now

$\delta'(\{q_3, q_4\}, 0) \rightarrow \epsilon\text{-closure}(\delta(q_3, 0) \cup \delta(q_4, 0))$

$= \epsilon\text{-closure}(q_3 \cup q_4)$

$= (q_3, q_4)$

$\delta'(\{q_3, q_4\}, 1) \rightarrow \epsilon\text{-closure}(\delta(q_3, 1) \cup \delta(q_4, 1))$

$= \epsilon\text{-closure}(\phi \cup \phi)$

$= \phi$

$\delta'(\{q_2, q_3, q_4\}, 0) \rightarrow \epsilon\text{-closure}(\delta(q_2, 0) \cup \delta(q_3, 0) \cup \delta(q_4, 0))$

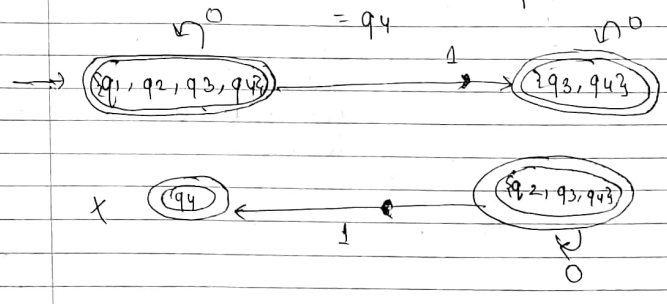
$= \epsilon\text{-closure}(q_2 \cup q_3 \cup q_4)$

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

$\delta'(\{q_2, q_3, q_4\}, 1) \rightarrow \epsilon\text{-closure}(\delta(q_2, 1) \cup \delta(q_3, 1) \cup \delta(q_4, 1))$

$= \epsilon\text{-closure}(q_4, \phi, \phi)$

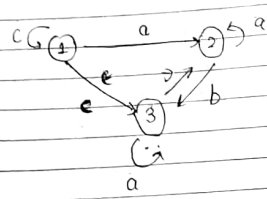
$= q_4$



2015 spring

Eliminate the epsilon transition from the ε-NFA $N = (\{1,2,3\}, \{a,b,c\}, \delta, 1, \{3\})$ where the transition function is-

state	a	b	c	ε
1	2	ϕ	{1,3}	3
2	2	3	ϕ	
3	3	ϕ	ϕ	2



ε-closure (1) → {1,3,2}
(2) → {2}
(3) → {2,3}

Let δ' be the transition fxn of the FA being formed.

$\delta'(\{1,2,3\}, a) \rightarrow \hat{\delta}(1,a) \cup \delta(2,a) \cup \delta(3,a)$
$= \epsilon\text{-closure}\{2 \cup 2 \cup 3\}$
$= \epsilon\text{-closure}\{2\} \cup \{3\}$
$= \{2,3\}$

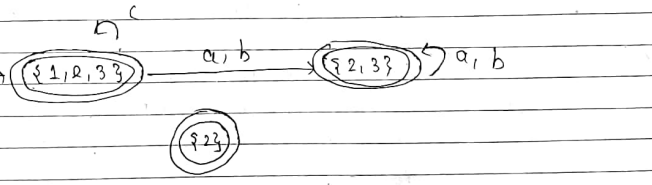
$\delta'(\{1,2,3\}, b) \rightarrow \epsilon\text{-closure}\ \delta(1,b) \cup \delta(2,b) \cup \delta(3,b)$
$= \epsilon\text{-closure}\{\phi \cup 3 \cup \phi\}$
$= \{2,3\}$

$\delta'(\{1,2,3\}, c) \rightarrow \epsilon\text{-closure}\ \delta(1,c) \cup (2,c) \cup \delta(2,c)$
$= \epsilon\text{-closure}\{1,3\} \cup \phi \cup \phi$
$= \epsilon\text{-closure}\{1\} \cup \epsilon\text{-closure}\{3\}$
$= \{1,2,3\}$

$\delta'(\{2,3\}, a) \rightarrow \epsilon\text{-closure}\ \delta(2,a) \cup \delta(3,a)$
$= \{2,3\}$

$\delta'(\{2,3\}, b) \rightarrow \epsilon\text{-closure}\{\delta(2,3), b\}$
$= \epsilon\text{-closure}\{\delta(2,b) \cup \delta(3,b)\}$
$= \{2,3\}$

$\delta'(\{2,3\}, c) \rightarrow \epsilon\text{-closure}(\delta(2,3), c)$
$\rightarrow \epsilon\text{-closure}\{\phi \cup \phi\}$
$\rightarrow \phi$



Regular expression:
An expression used to generate string for FA is called regular expression. It is called language generation. A regular expression is recursively defines as:

1. Empty string(ε), ∅, every state (∅) and symbols of input alphabet are regular expression.
2) If R_1 and R_2 are two regular expression the union of R_1 and R_2 denoted by $R_1 + R_2$ is also a regular expression.
3) If R_1 and R_2 are two regular expression then concatenation of R_1 & R_2 denoted by $R_1 R_2$ is also regular expression.
4) If R is a regular expression then closure of R denoted by R^* is also regular expression.
5) If R is regular expression, then (R) is also regular expression

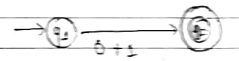
eg
$0 \cdot (0+1)^* \cdot 1 \cdot 1$

eg
$(0+1)^* = \{\varepsilon, 0, 1, 00, 01, 10, 101, 1101, \ldots\}$
$(0^* + 1^*) = \{\varepsilon, \varepsilon, 1, 00, 11, 000, 111, \ldots\}$
$(01)^* = \{\varepsilon, 01, 0101, 010101, \ldots\}$
$0^* 1^* = \{\varepsilon, 1, 11, 01, 001, 011, 0011 \ldots\}$
$1 \cdot 1^* = \{1, 11, 111, 1111, \ldots\}$

2018 Spring
Design a FA that accepts a set of string such that every string end in 00, over alphabet $\{0, 1\}$
RE = $(0+1)^* 0 \cdot 0$

Regular expression to FA:
- For every regular expression there exists a path from initial state to final state such that the path is labelled with regular expression.

Union: $\Sigma = \{0, 1\}$
RE = 0 + 1



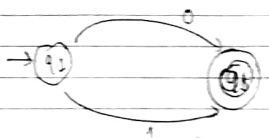
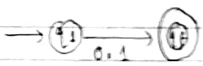


Fig 1 FA

Concatenation
RE = 0 · 1



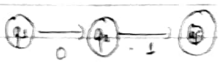


Fig 2 FA

RE = 0*

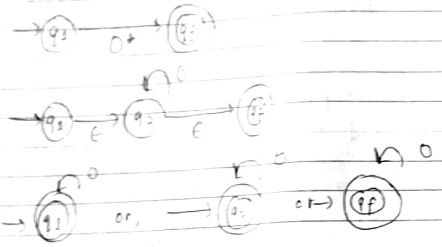
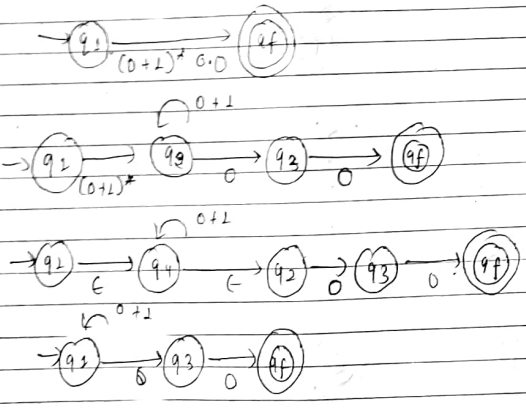


Fig : FA

RE = (0+1)* . 0 . 0



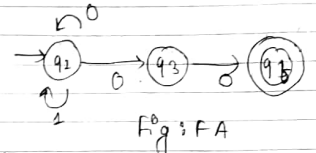


Fig : FA

2014spring

RE = 01* + 1

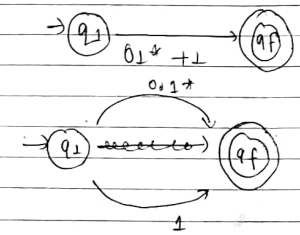
2010spring

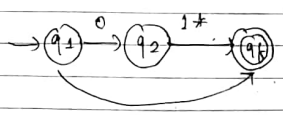
RE = a(a+b)* bb

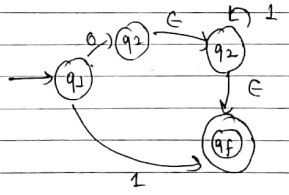
2015 Fall

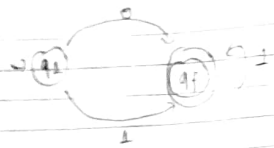
RE = (a + a(b+aa)* b)* a(b+aa)* a

RE : 01* + 1



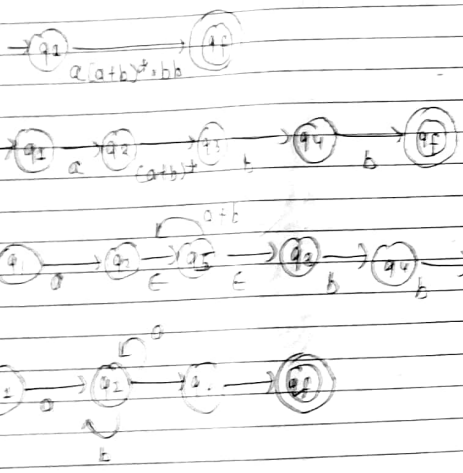






FA

RE : a(a+b)*bb



2014

RE = (a+a(b+aa)*b)*a(b+aa)*)a

FA to Regular expression :-

Arden's theorem:
Let P and Q be two regular expressions such that P does not contain empty string(ε). Then, a regular expression R of the form.

$$R = Q + RP$$

here, a unique solution

$$R = QP^*$$

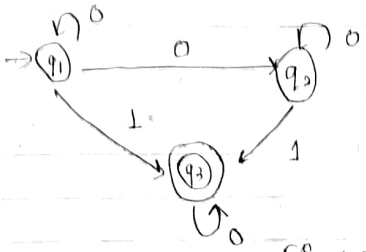


Fig : FA

$$q_1 = \underbrace{q_1 \cdot 0}_{R\ R \cdot P} + \underbrace{\epsilon}_{Q} \quad \text{— (i)}$$

$$q_2 = q_1 0 + q_2 \cdot 0 \quad \text{— (ii)}$$

$$q_3 = q_1 \cdot 1 + q_2 \cdot 1 + q_3 \cdot 0 \quad \text{— (iii)}$$

From eqn ①,

$q_1 = q_1 \cdot 0 + \epsilon$

$R = R \cdot P + Q$

Using theorem, we get

$R = QP^*$

i.e $q_1 = \epsilon \cdot 0^*$

$q_1 = 0^*$ — ④

From equation ② & ④, we get

$q_2 = \underbrace{0^* 0}_{Q} + q_2 \cdot \underbrace{0}_{P}$

$R \quad R \quad P$

Using Arden's theorem we get

$q_2 = 0^* 0 0^*$ — ⑤

Using

From equations ③, ④ & ⑤, we get,

$q_3 = \underbrace{0^* 1 + 0^* 0 \cdot 0^* 1}_{Q} + q_3 \underbrace{0}_{P}$

R

Using Arden's theorem we get,

$R = QP^*$

$q_3 = (0^* 1 + 0^* 0 0^* 1) 0^*$

$q_3 = 0^* 1 0^* + 0^* 0 0^* 1 0^*$

Since q_3 is the final state, required regular expression is : $0^* 1 0^* + 0^* 0 0^* 1 0^*$

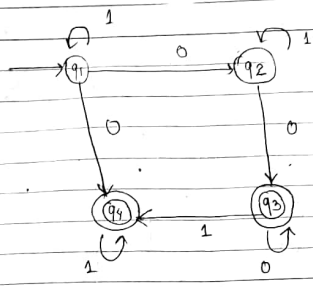


Fig: FA

$q_1 = q_1 \cdot 1 + \epsilon$ — ①

$q_2 = q_1 \cdot 0 + q_2 \cdot 1$ — ②

$q_3 = q_2 \cdot 0 + q_3 \cdot 0$ — ③

$q_4 = q_2 \cdot 0 + q_3 \cdot 1 + q_4 \cdot 1$ — ④

From eqn ①

$q_1 = q_1 \cdot 1 + \epsilon$

$R \quad R \cdot P \quad Q$

Using theorem, we get

$R = QP^*$

i.e $q_1 = \epsilon \cdot 1^*$

$q_1 = 1^*$ — ⑤

From eqn ② and 5 we get

$q_2 = \underbrace{1^* 0}_{Q} + q_2 \cdot 1$

$R \quad R \quad P$

Using Arden's theorem:

$R = QP^*$

$q_2 = 1^*01^*$ — (6)

From eqn (3), (5), (6) we get

$q_3 = \underbrace{1^*0.1^*.0}_{Q} + \underset{R}{q_3}.\underset{P}{0}$

From eqn (4), (5), (6)

Using Arden's theorem

$q_3 = 1^*0.1^*0.0^*$ — (7)

From eqn 4, 5, 6, 7

$q_4 = \underbrace{1^*0 + 1^*0.1^*0.0^*.1}_{Q} + \underset{R}{q_4}.\underset{P}{1}$

Using Arden's theorem

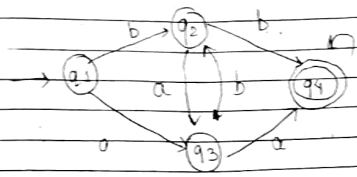
$q_4 = (1^*0 + 1^*0.1^*0.0^*1)1^*$

$q_4 = 1^*0.1^* + 1^*0.1^*0.0^*11^*$

Since q_3 and q_4 are final state, the required reg expression is : $1^*0.1^*0.0^* + 1^*0.1^* + 1^*0.1^*0.0^*11^*$

2013 Fall

Find the regular expression for the following transition diagram



$q_1 = \epsilon$ — (1)

$q_2 = q_1.b + q_2.b$ — (2)

$q_3 = q_1.a + q_2.a$ — (3)

$q_4 = aq_4 + bq_4 + aq_3 + bq_2$ — (4)

$q_2 = q_1.b + q_3.b$

$q_2 = \epsilon.b + q_3.b$

$q_2 = b + q_3.b$

$q_2 = b + q_1a.b + q_2.ab$

$q_2 = b + \epsilon ab + q_2.ab$

$\underset{R}{q_2} = \underbrace{b + ab}_{Q} + \underset{R}{q_2}.\underset{P}{ab}$

$R = QP^*$

$q_2 = (b + ab)(ab)^*$

$q_2 = b(ab)^* + ab(ab)^*$ — (5)

From (iii), (i) & (v)

$q_3 = \epsilon \cdot a + (b(ab)^* + ab(ab)^*) \cdot a$

$q_3 = a + b(ab)^* a + ab(ab)^* a$

From (iii), (i) & (v)

$q_3 = q_1 \cdot \epsilon + (q_1 \cdot b + q_3 \cdot b) a$

$\underset{R}{q_3} = \underset{}{a} + \underset{Q}{q_1 ba} + \underset{R\ P}{q_3 \cdot ba}$

$R = QP^*$

$q_3 = (a + ba)(ba)^*$

$q_3 = a(ba)^* + ba(ba)^* \quad -(vi)$

From (iv), (i), (ii) (vi)

$\underset{R}{q_4} = \underset{P}{(a+b)} \underset{R}{q_4} + \underbrace{a(a(ba)^* + ba(ba)^*) + b(b(ab)^* + (ab)^*)}_{Q}$

$q_4 = QP^*$

$= (aa(ba)^* + aba(ba)^* + bb(ab)^* + b(ab)^*)(a+b)^*$

$q_4 = ab(ba)^*(a+b)^* + aba(ba)^*(a+b)^* + bb(ab)^*(a+b)^* + b(ab)^*(a+b)^*$

This is not correct.

Closure properties of regular sets or language:-

The class of language of FA is closed under:

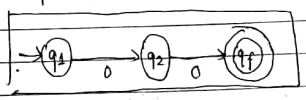
(i) Union
(ii) Concatenation
(iii) Kleene Closure

① Union:

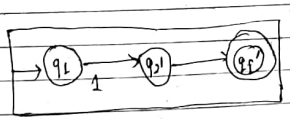
Let R_1 and R_2 be two regular expressions. Then, we have to show that the union of R_1 and R_2 i.e $R_1 + R_2$ is also a regular expression. We know that for every expression there exists an equivalent FA. Therefore, to show that $R_1 + R_2$ is regular, we need to design a FA that can process either the strings generated by R_1 or strings generated by R_2. Let $M_1 = (Q, \Sigma, \delta, q_1, F_1)$ and $M_2 = (Q_2, \Sigma_2, \delta_2, q_1', F_2)$ be the finite automata for R_1 and R_2 respectively.

Again, Let $R_1 = 00$ and $R_2 = 11$

Now, FA for R_1 and R_2 will be as:



M_1



M_2

Let $M(Q, \Sigma, \delta, q_0, F)$ be a FA such that we need to define its operational characteristics in such a way that it process the strings of R_1 or R_2.

Its operational characteristics are defined as

$Q = Q_1 \cup Q_2 \cup q$

$\Sigma = \Sigma_1 \cup \Sigma_2$

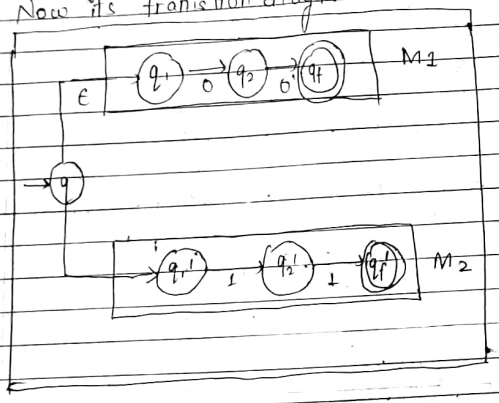
δ consists of

$\delta_1 \cup \delta_2 \cup \{\delta(q, \epsilon) \to q_1, \delta(q, \epsilon) \to q_1'\}$

$q_0 = q$

$F = F_1 \cup F_2$

Now its transition diagram is:



Here, q is the initial state of M. The transition fxns $\delta(q, \epsilon) \to q_1$ and $\delta(q, \epsilon) \to q_1'$ tells that M can transit to either the initial state of M_1 or initial state of M_2 by consuming ϵ non-deterministically. When M uses transition fxn $\delta(q, \epsilon) \to q_1$, it transits of initial state of M_1, uses transition fxn δ_1 and processes all strings generated by R_1. Similarly, when M uses transition fxn $\delta(q, \epsilon) \to q_1'$, it transits to initial state of M_2, uses transition fxns δ_2 and processes all the strings generated by R_2 i.e M copies the processing of either M_1 or M_2. Hence we say that M is such a FA that can process either the strings generated by R_1 or strings generated by R_2. Therefore, $R_1 + R_2$ is also regular. Thus, the class language of FA is closed under union.

(ii) Concatenations:

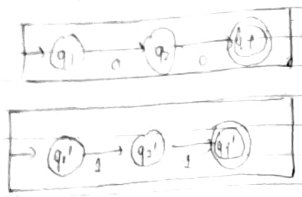
Let R_1 and R_2 be two regular expression. Then we have to show that the concatenation of R_1 and R_2 i.e R_1R_2 is also regular expression. We know that for every expression there exists an equivalent FA. Therefore, to show R_1R_2 is regular, we need to design a FA that process the strings generated by R_1 and R_2. Let strings followed by strings generated

$M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ automata for R_1 and M_2

$M_2 = (Q_2, \Sigma, \delta_2, q_1', F_2)$ be the finite automata for R_1 and R_2 respectively.

Again, let $R_1 = 00$ and $R_2 = 11$

The FA for R_1 and R_2 will be as:



Let $M = (Q, \Sigma, \delta, q_0, F)$ be a FA such that we need to define its operation characherstics in such a way that it is process the strings of R_1 and followed by strings of R_2

It's operational characherstics are defined as.

$Q = Q_1 \cup Q_2 \cup q$

$\Sigma = \Sigma_1 \cup \Sigma_2$

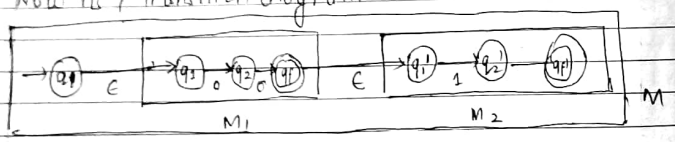
δ consists of

$\delta_1 \cup \delta_2 \cup \{\delta(q, \epsilon) \to q_1, \delta(q_f, \epsilon) \to q_1'\}$

$q_0 = q$

$F_0 = F_2$

Now its, transition diagram is:



Here, q is the initial state of M. The transition fxn $\delta(q, \epsilon) \to q_1$ ~~and $\delta(q_f, \epsilon) \to q_1'$~~ tells that M can transist to initial state of M_1 and fxn $\delta(q_f, \epsilon) \to q_1'$ tells that final state of M_1 consumes ϵ and goes to initial state of M_2 and by using δ_2 it processes R_2.

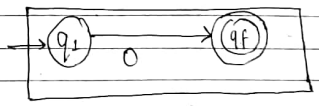
by consuming ϵ. After transiting to initial state of M_1, it uses transition fxn δ_1 and processes all the strings generated by R_1. Then it uses transition fxn $\delta(q_f, \epsilon) \to q_1'$ and transits to initial state of M_2. After that, it uses transition functions δ_2 and process all the strings generated by R_2 i.e M can process all the strings of M_1 followed by all the strings of M_2. Thus, we say that M can process the concatenation of R_1 and R_2. Hence R_1R_2 is also regular.

3) closure

Let R_1 be the regular expression. Then we have to show that the closure of R_1 is also a regular expression. We know that for every expression there exists an equivalent FA. Therefore, to show that R^* is a regular, we need to design a FA that can process the string generated by R. Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1)$ be the finite automata of R.

Again, $R = 0$

Now, FA for R will be



Let M (Q, Σ, δ, q0, f) be FA such that we need to define its operational characteristics in such a way that it can process R.

It's operation characteristics are defined as

Q = Q1 ∪ q1

Σ = Σ1

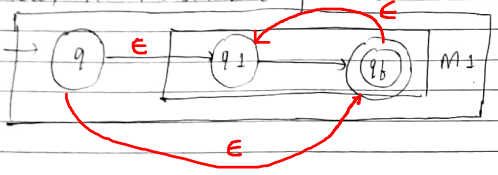
δ consists of

δ1 ∪ {δ(q, ε) → q1, δ(q, ε) → qf, δ(qf, ε) → q1}

q0 = q

f = f1.

Now, its transition diagram is:



Here, q is the initial state of M. The transition function δ(q, ε) → q1 tells that the initial state M can transits the initial state of M1 uses transition function δ1 and process all the strings generated by R. Similarly when M uses transition fxn δ(q, ε) → qf tells that it can process empty string and δ(qf, ε) → q1 tells that M can process any no. of zero or empty. Since, M is a finite automata that R is also a regular expression.

* Pumping lemma for regular language or regular sets:

Let L be a regular language and w ∈ L such that |w| = n. Let m be the total number of states and m ≤ n. Then, w can be decomposed into three sub-string x, y and z such that:

1) y ≠ ε
2) |xy| ≤ n
3) $xy^i z \in L$ for all $i \geq 0$

Proof:

Let L be regular language and w ∈ L. Let w = a1, a2 ... an be the string & q1, q2, ... qm be the total no. of states.

Now, according to the pumping lemma, we can decompose into three - sub-string as:

x = a1 a2 ... ai

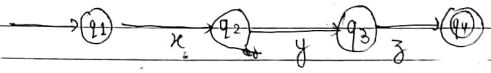
y = ai+1 ai+2 ... aj

z = aj+1 aj+2 ... an

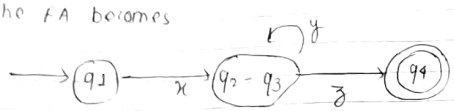
We know that

w = xyz ∈ L

Now, FA becomes is,



Here, $m = 4$ & $n = 3$
But, $m \leq n$. Therefore, according to the pigeon hole principle any two state must coincide. Let q_2 and q_3 be such states
Now,
the FA becomes



It's generalize FA is

from the above FA, we can write

$$1 \leq i < j \leq m$$
$$1 \leq i < j \leq n$$
$$|xy| \leq n$$

When
$i = 0$
$w = xz$
When x is inputted to q_i, it transist to state $q_i (= q_j)$ and after consuming z it transit to state q_m (final state). Therefore, xz is accepted
i.e $xz \in L$

When $i = 1$
$w = xyz \in L$

When $i = 2$
$xy^2z \in L$

similarly,

$xy^iz \in L$ for all $i \geq 0$ //

Show that the language $L = \{a^n b^n : n > 0\}$ is not regular.

→ Let $L = \{a^n b^r ; n > 0\}$ is regular
and $w = a^p b^p \in L$
According to pumping lemma, w can be decomposed into three sub-string x, y and z as:

$x = a^q$
$y = a^r ; r > 0$
$z = a^{p-(q+r)} b^p$

Now, $xy^2z = a^q a^{2r} a^{p-q-r} b^p$
$= a^{q+2r+p-q-r} b^p$
$= a^{p+r} b^p$

Since $r > 0$, $p+r > p$

i.e $a^{p+r} b^p$ is not of the form of $a^p b^p$
$\therefore xy^2z \notin L$
Hence, L is not regular.

Show that the language $L = \{1^{n^2} : n > 0\}$ is not regular.

→ Let $L = \{1^{n^2} : n > 0\}$ is regular
and $w = 1^{p^2} \in L$
According to pumping lemma, w can be decomposed into three sub-string x, y, z as:

$x = 1^{q^2}$
$y = 1^{q^2+r}$
$z = 1^{p^2-q^2-r}$ $; r > 0$

Now, $xy^2z = 1^{q^2} (1^{q^2+r})^2 1^{p^2-q^2-r}$
$= 1^{q^2+2r^2+p^2-q^2-r^2}$
$= 1^{p^2+r^2}$

Since, $r^2 > 0$, $p^2+r^2 > p^2$
i.e $1^{p^2+r^2}$ is not of the form 1^{p^2}
$\therefore xy^2z \notin L$
Hence, L is not regular.

Show that the language $L = \{xx^R : x \in \{0,1\}^*\}$ is not regular. (x^R = reverse)

→ Let $x = 0^n 1^n$
$x^R = 1^n 0^n$
$\therefore L = \{0^n 1^n 1^n 0^n ; n > 0\}$ is regular
and $w = 0^p 1^p 1^p 0^p \in L$
According to pumping lemma, w can be decomposed into three sub-string x, y, z as

$x = 0^q$
$y = 0^r$
$z = 0^{p-(q+r)} 1^p 1^p 0^p$ $, r > 0$
Now, $xy^2z = 0^q 0^{2r} 0^{p-q-r} 1^p 1^p 0^p$
$= 0^{q+2r+p-q-r} 1^p 1^p 0^p$
$= 0^{p+r} 1^p 1^p 0^p$

since, $i > 0$, $p+i > p$

i.e $0^{p+i}1^p1^p0^l$ is not of the form $0^l1^p1^p0^p$

Hence, L is not regular

Grammar

A grammar is defined as:-

$G = (V, \Sigma, R, S)$

where, V = set of non terminals

Σ = set of terminals

R = set of production rules

S = start symbol

eg: For the regular expression $R = a(a^* + b^*)b$,

Equivalent grammar is-

$G = (V, \Sigma, R, S)$

where,

$V = \{S, M, A, B\}$

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

R consists of

$S \rightarrow a\ M\ b$

$M \rightarrow A/B$

$A \rightarrow \epsilon / aA$

$B \rightarrow \epsilon / bB$

S = start symbol

$w = abbbb$

Derivation:

$S \rightarrow a\ M\ b$

$\rightarrow a\ B\ b$ [$\because M \rightarrow B$]

$\rightarrow a\ b\ B\ b$ [$\because B \rightarrow bB$]

$\rightarrow a\ b\ b\ B\ b$ [$\because B \rightarrow bB$]

$\rightarrow a\ b\ b\ b\ B\ b$ [$\because B \rightarrow bB$]

$\rightarrow a\ b\ b\ b\ b$ [$\because B \rightarrow \epsilon$]

$\rightarrow abbbb$

Construct an equivalent grammar of the following regular expression.

$RE = 01^*010^* | 0^*11(0+1)^*$

use this production rule

$S \rightarrow A/B$

$A \rightarrow 0C01D$

$D \rightarrow D11\epsilon$

$C \rightarrow \epsilon / 1C$

$D \rightarrow \epsilon / 0D$

$E \rightarrow \epsilon / 1E / 1E$

Types of Grammar (Chomsky hierarchy of grammar):-

1. Unrestricted grammar (Type 0)
2. Context sensitive grammar (Type 1)
3. Context free grammar (Type 2).
4. Regular grammar (Type 3)

- If no restriction is applied to production rules of a grammar then it is called unrestricted grammar.
- A grammar is said to be context sensitive grammar if its production rules are of the form -

$$w_1 \alpha w_2 \longrightarrow w_1 \beta w_2$$

Here, $\alpha, \beta \in V$ and w_1, w_2 are called contexts of α and β.

- A grammar is said to be context free grammar if its production rules are of the form.

$$\alpha \longrightarrow \beta$$

where, $\alpha \in V$ and $\beta \in (V \cup \Sigma)^*$

- A grammar is said to be regular grammar if its production rules are of the form

non-terminal $\longrightarrow$ exactly one terminal

or

non-terminal $\rightarrow$ one terminal followed by one non-terminal.

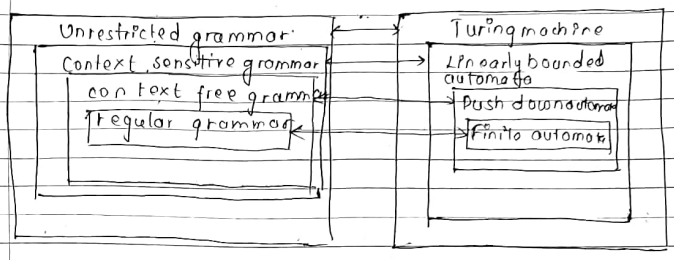


fig: Types of grammar with their respective mathematical models.

Derivations:-

- The sequence of production rules used to generate a string is called derivation.

eg: $w_1, w_2 \ldots w_n$

$$w_1 \xrightarrow{*} w_n$$

- There are two types of derivations
  1. Leftmost derivation.
  2. Right most derivation

eg:

$S \rightarrow AB$

$A \rightarrow aA \mid b$

$B \rightarrow bB \mid a$

$w = aabbba$

Leftmost derivation :-

$S \rightarrow AB$
$\rightarrow aAB$ [∵ A → aA]
$\rightarrow aaAB$ [∵ A → aA]
$\rightarrow aabB$ [∵ A → b]
$\rightarrow aabbB$ [∵ B → bB]
$\rightarrow aabbbB$ [∵ B → bB]
$\rightarrow aabbba$ [∵ B → a]

Rightmost derivation :-

$S \rightarrow AB$
$\rightarrow AbB$ [∵ B → bB]
$\rightarrow AbbB$ [∵ B → bB]
$\rightarrow Abba$ [∵ B → a]
$\rightarrow aAbba$ [A → aA]
$\rightarrow aaAbbba$ [A → aA]
$\rightarrow aabbba$ [∵ A → b]

$S \rightarrow aAS|aS|b$
$A \rightarrow bB|a$
$B \rightarrow aA|b$
$w = abaaaaab$

Leftmost derivation:

$S \rightarrow aAS|aS|b$
$\rightarrow aAS$ [A → bB]
$\rightarrow abBS$ [B → aA] [A → bB]
$\rightarrow abaAS$ [B → aA] [B → aA]
$\rightarrow abaaAS$ [B → bA] [B → aA]
$\rightarrow abaaaAS$ [B → aA]
$\rightarrow abaaaaAS$ [B → aA]
$\rightarrow abaaaaaS$ [A → a]
$\rightarrow abaaaaab$ [S → b]

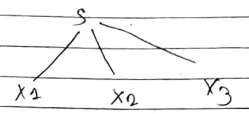
Rightmost derivation:

$\rightarrow aAS$
$\rightarrow aA$ [S → aS]
$\rightarrow aAaS$ [S → aS]
$\rightarrow aAaaS$ [S → aS]
$\rightarrow aAaaaS$ [S → aS]
$\rightarrow aAaaaaS$ [S → aS]
$\rightarrow aAaaaab$ [S → b]
$\rightarrow abBaaaab$ [A → bB]
$\rightarrow abaAaaaab$ [B → aA]
$\rightarrow abaaaaab$ [A → a]

Derivation tree / production tree :

- It is the hierarchical representation of the derivation
- The start symbol is the root node of the derivation tree and the string in the right side of start symbol is divided into multiple number of branches
- This process is continued at every level of the tree.

eg : $S \rightarrow X_1 X_2 X_3$ be the production rule, then its derivation tree is -



Eg :

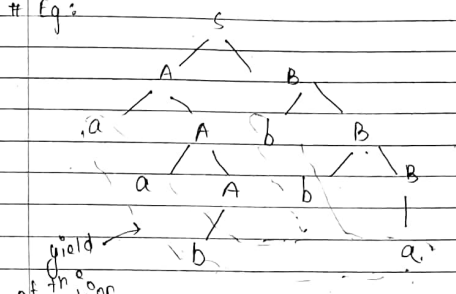
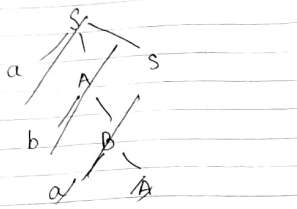
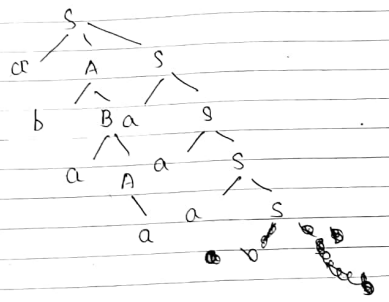


Fig : Derivation tree.





Ambiguity in CFG:

A CFG is said to be ambiguous if there exists more than one leftmost derivation or more than one rightmost derivation for the same string.

eg:

$S \rightarrow aAs \mid b$
$A \rightarrow bB \mid ba$
$B \rightarrow a$
is an ambiguous CFG

$w = abab$

LMD:-
$S \rightarrow aAS$
$\rightarrow abBs$ [$\because A \rightarrow bB$]
$\rightarrow abas$ [$\because B \rightarrow a$]
$\rightarrow abab$ [$\because S \rightarrow b$]

RMD:-
$S \rightarrow aAs$
$\rightarrow abas$ [$\because A \rightarrow ba$]
$\rightarrow abab$ [$\because S \rightarrow b$]

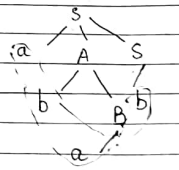


Fig: Derivation tree

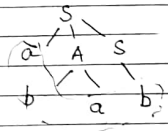


Fig: Derivation tree

Show that the following CFG is ambiguous.

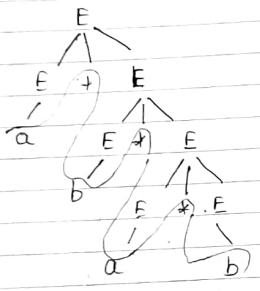
$E \rightarrow E+E \mid E*E \mid a \mid b$

$w = a+b*a*b$

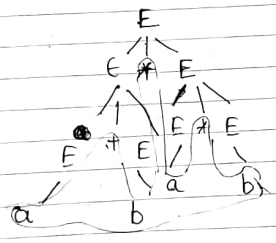
$E \rightarrow E+E$
$\rightarrow E+E*E$ ($E \rightarrow E*E$)
$\rightarrow E+E*E*E$ ($E \rightarrow E*E$)
$\rightarrow E+E*E*b$ ($E \rightarrow b$)
$\rightarrow E+E*a*b$ ($E \rightarrow a$)
$\rightarrow E+b*a*b$ ($E \rightarrow b$)
$\rightarrow a+b*a*b$ ($E \rightarrow a$)

→ E + E [E → E + E]
→ a + E [E → a]
→ a + E * E [E → E * E]
→ a + b * E [E → b]
→ a + b * E * E [E → E * E]
→ a + b * a * E [E → a]
→ a + b * a * b [E → b]

LMD
→ E * E
→ E + E * E [E → E + E]
→ a + E * E [E → a]
→ a + b * E [E → b]
→ a + b * E * E [E → E * E]
→ a + b * a * E [E → a]
→ a + b * a * b [E → b]



a + b * a * b



a + b * a * b

∴ Since, there exists more than one LMD, so, CFG is ambiguous.

2017 FALL

Check ambiguity of the grammar
S → aB | bA
A → aS | bAA | a
B → aBB | bS | b

S → aB
→ aaBB [B → aBB]
→ aabS [B → bS]
→ aaba
→ aaBB [B → aBB]
→ aabS B [B → bS]
→ aabaBB [B → b]
→ aababB [B → b]
→ aababb [B → b]

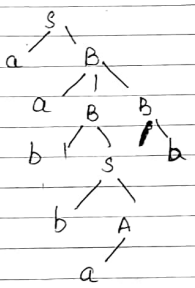
S → aB
→ aaBB [B → aBB]
→ aabS B [B → bS]
→ aabbA B [S → bA]
→ aabbbA [S → bA]
→ aabbbA [A → a]

S → aB
→ aaBB [B → aBB]
→ aabS B [B → bS]
→ aabbA B [S → bA]
→ aabb [A → bA]

S → aB
→ aaBB [a → aBB]
→ aa

LMD

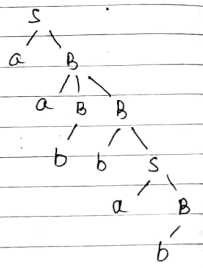
$S \rightarrow aB$ [$\because S \rightarrow aB$]
$\rightarrow aaBB$ [$B \rightarrow aBB$]
$\rightarrow aabSB$ [$B \rightarrow bS$]
$\rightarrow aabbAB$ [$S \rightarrow bA$]
$\rightarrow aabbaB$ [$A \rightarrow a$]
$\rightarrow aabbab$ [$B \rightarrow b$]



aabbab

LMD

$S \rightarrow AB$
$\rightarrow aaBB$ [$B \rightarrow aBB$]
$\rightarrow aabB$ [$B \rightarrow b$]
$\rightarrow aabbS$ [$B \rightarrow bS$]
$\rightarrow aabbaB$ [$S \rightarrow aB$]
$\rightarrow aabbab$ [$B \rightarrow b$]



aabbab

Since, there exits more than one LMD, so grammar is ambiguous.

Normal form of CFG
- If certain restriction is applied to the production rules of CFG then it is said to be in normal form.
- There are two types of normal forms -
  (i) Chomsky Normal form (CNF)
  (ii) Greibach normal form (GNF)

Chomsky Normal form (CNF):

A CNG is said to be in CNF if its production rules are of the forms-

Non-terminal → exactly one terminal

or

Non-terminal → exactly two non-terminals.

eg: $S \rightarrow a$, $S \rightarrow AB$, etc.

imp. # Conversion of CFG to CNF:

To convert a given CFG to its equivalent CNF, we need to convert all the production rules of the given CFG to the form-

Non-terminal → exactly one terminal

or

Non-terminal → exactly two non-terminal

Note:

To accomplish this task, we use substitution method. First, we replace all terminal symbols in a production rule by non-terminal symbols. But this substitution is possible only when the non-terminals that are substituted generates the respective terminal symbols. Now, add such production rule to the given CFG.

eg: $S \rightarrow aA$ be the production rule. To convert it

into its equivalent CNF, we replace 'a' by a nonterminal, say C_7 so that production rule becomes $S \to C_1A$. But this is possible only when a production rule $C_1 \to a$ exists. So, we add $C_1 \to a$ to the given grammar. We continue this process until all the production rules are converted to CNF.

eg: Convert the following CFG to its equivalent CNF.

$S \to aAS \mid as$

$A \to aBb$

$B \to bB \mid bAa$

1. $S \to aAS$ gives $S \to C_1AS$ where $C_1 \to a$
   $S \to C_1AS$ gives $S \to C_1C_2$ where $C_2 \to AS$
2. $S \to as$ gives $S \to C_1S$ where $C_1 \to a$
3. $A \to aBb$ gives $A \to C_1BC_3$ where $C_1 \to a$, $C_3 \to b$

   $A \to C_1BC_3$ gives $A \to C_1C_4$ where $C_4 \to BC_3$
4. $B \to bB$ gives $B \to C_3B$ where $C_3 \to b$
5. $B \to bAa$ gives $B \to C_3AC_1$ where $C_3 \to b$, $C_1 \to a$

   $B \to C_3AC_1$ gives $B \to C_5C_1$ where $C_5 \to C_3A$.

Let $G' = (V', \Sigma', R', S')$ be the equivalent CNF where,

$V' = \{S, A, B, C_1, C_2, C_3, C_4, C_5\}$

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

R' consists of -

$\{S \to C_1C_2 \mid C_1S, A \to C_1C_4, B \to C_3B \mid C_5C_1,$
$C_2 \to AS, C_4 \to BC_3, C_5 \to C_3A, C_1 \to A,$
$C_3 \to b\}$

$S' = S$

2016 Spring

Consider the grammar $G = (V, \Sigma, P, S)$ where,

$V = \{S, A, B\}$

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

and the productions P are:

$S \to bA \mid aB$

$A \to bAA \mid as \mid a$

$B \to aBB \mid bS \mid b$

Find an equivalent grammar in CNF.

1. $S \to bA$ gives $S \to C_bA$ where, $C_b \to b$
2. $S \to aB$ gives $S \to C_aB$ where, $C_a \to a$
3. $A \to bAA$ gives $A \to C_bAA$ where $C_b \to b$
   $A \to C_bAA$ gives $A \to C_1A$ where $C_1 \to C_bA$
4. $A \to as$ gives $A \to AS$ where $A \to a$
5. $B \to aBB$ gives $B \to ABB$ where $A \to a$
   $B \to ABB$ gives $B \to C_2B$ where $C_2 \to AB$
6. $B \to bS$ gives $B \to C_bS$ where $B \to b$, $C_b \to b$

Let $G' = (V', \Sigma', P', S')$ be the equivalent CNF where,

$V' = \{S, A, B, C_1, C_2\}$

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

P' consists of

$\{S \to AB \mid C_1A \mid BA$, $A \to AS \mid C_aB \mid a$, $C_a \to a$, $C_2B \mid BB \mid b$,
$C_1 \to BA$, $C_2 \to AB$

Greibach normal form (GNF):

A CFG is said to be in GNF if its production rules are of the form:-

$$\boxed{A \rightarrow a\beta^*}$$

Where, $A, \beta \in V$ and $a \in \Sigma$.

Non-terminal → exactly one terminal
or
Non-terminal → one terminal followed by any number of non-terminals.

eg: $S \rightarrow a$, $S \rightarrow aA$, $S \rightarrow aAB$, etc.

Simplication of CFG:

1. Removal of unit productions.
2. Removal of null productions.
3. Removal of useless productions and symbols.

1. Removal of unit productions:-

A production rule of the form-
Non-terminal → non-terminal.
is called unit-production.
eg: $S \rightarrow A$, $A \rightarrow B$, etc.

Let $A \rightarrow B$ be the unit production, To eliminate $A \rightarrow B$ from the given CFG, we search for a production rule of the form $B \rightarrow w$, where w is the terminal or terminal string (generated directly or indirectly). Then, we replace B in $A \rightarrow B$ by w so that a new production rule $A \rightarrow w$ is obtained. We add $A \rightarrow w$ to the given CFG and eliminate $A \rightarrow B$. This process is continued until all the unit productions are eliminated.

eg: Simplify the following CFG by eliminating unit productions:
$S \rightarrow AB$, $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow D$, $D \rightarrow E$, $E \rightarrow a$.

→ Soln:
Here, the unit productions are:
$A \rightarrow B$
$B \rightarrow C$
$C \rightarrow D$
$D \rightarrow E$

To Eliminate these unit productions, we search such a production rule for B, C, D and E, that generates a terminal or terminal string directly or indirectly.

Here, $E \rightarrow a$ exists.
Now, replace E in $D \rightarrow E$ by 'a' so that a new production rule $D \rightarrow a$ is obtained. We add $D \rightarrow a$

to the given CFG and eliminate $D \to \epsilon$.
Similarly, we $C \to a$, $B \to a$ and $A \to a$ to the given CFG and eliminate unit productions $C \to D$, $B \to C$ and $A \to B$.
Hence, after eliminating unit productions, the simplified CFG is

$S \to AB$
$A \to a$
$B \to a$
$C \to a$
$D \to a$
$E \to a$

2. # Removal of null productions:

A production rule of the form
non-terminal $\Rightarrow$ empty string
is called null productions.
eg: $S \to \epsilon$, $A \to \epsilon$, etc.

Let $\alpha \to \epsilon$ be the null production. To eliminate $\alpha \to \epsilon$ from the given CFG, we search for all the occurence of α in the production rules of given CFG. Then, we replace α by ϵ in all the production rules where α occurs so that new production rules are obtained. We add these production rules to the given CFG and eliminate $\alpha \to \epsilon$. This process is continued until all the null productions are eliminated.

eg Simply the following grammar by eliminating null productions.

$S \to aABs \mid AB$
$A \to aA \mid \epsilon$
$B \to bB \mid \epsilon$

→ Soln:
Here, the null productions are:-
$A \to \epsilon$
$B \to \epsilon$
To eliminate $A \to \epsilon$, we search for the occurence of A in the production rules of given CFG.
$S \to aABS$ gives $S \to aBS$
$S \to AB$ gives $S \to B$
$A \to aA$ gives $A \to a$

After eliminate $A \to \epsilon$, the CFG becomes -
$S \to aABS \mid AB \mid aBS \mid B$
$A \to aA \mid a$
$B \to bB \mid \epsilon$

To eliminate $B \to \epsilon$, we search for occurences of B in the production rules of new CFG.
Now,
$S \to aABs$ gives $S \to aAs$
$S \to AB$ gives $S \to A$

$S \to aBS$ gives $S \to aS$
$S \to B$ gives $S \to \epsilon$
$S \to bB$ gives $S \to b$

After eliminating $B \to \epsilon$, the CFG becomes-

$S \to aABS \mid AB \mid aBS \mid B \mid aAS \mid A \mid aS \mid b \mid \epsilon$
$A \to aA \mid a$
$B \to bB \mid b$

To eliminate $S \to \epsilon$, we search for occurences of S in the production rule of new CFG.
Now,

$S \to aABS$ gives $S \to aAB$
$S \to aAB$ gives $S \to aA$
$S \to aBS$ gives $S \to aB$
$S \to a$ gives $S \to a$.

After eliminating $S \to \epsilon$, the CFG becomes

$S \to aAB \mid aA \mid aB \mid a \mid AB \mid B \mid A \mid b$
$A \to aA \mid b$
$B \to bB \mid b$

3. Removal of useless productions and symbols :-

To eliminate useless productions and symbols from the given CFG, we first identify all the generating production rules of the given CFG. Generating production rules are the rules that can generate terminal or terminal strings directly or indirectly. The remaining production rules are non-generating productions rules, so we eliminate them. Among the generating production rules, some of the production rules may not be ~~redu~~ reachable from start symbol. To identify such production rules and symbols, we construct a graph called depending graph. The non-terminal symbols of the given CFG are represented by vertices of the dependency graph. The vertices that remains disconnected or there exists no direct or indirected path from start symbol, are the symbols that are not reachable from start symbol. Hence, they are also eliminated.

eg: Simplify the following CFG by eliminating useless production and symbols.

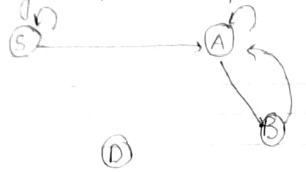
$S \to aAs \mid b$
$A \to bA \mid AB \mid a$
$B \to bA \mid b \mid a \mid b$
$D \to ab$

→ soln :
The generating production rules are :-

S → aAs | b
A → bA | AB | a
B → bA | b
D → ab

The production rule B → acb is non-generating, so it is eliminated

To identify the symbols that are not reachable from start symbol, we draw following dependency graph



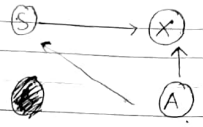
here, D remains disconnected. So it is discarded.
Hence, the simplified CFG is:
S → aAs | b
A → bB | AB | b
B → bA | b

S → aB | bx
A → Bad | bsX | a
B → asB | bBX
X → sBD | aBx | ad.

→ The generating production rules are.
S → aB | bx
A → bsX | a
B →
X → ad

The production rule S → aB, A → Bad, B → asB | bBX, X → sBD | aBx are non-generating so, it is eliminated.

To identify the symbols that are not reachable from start symbols, we draw following dependency graph.



There is no direct/indirect path from S to A.
Here, A remains disconnected. So, it is discarded
Hence, the simplified CFG is:
S → bx
X → ad

Closure properties of context free language:

The class of language of context free grammar is closed under:-
(i) Union
(ii) Concatenation
(iii) kleene closure.

Union:

Let $G_1 = (V_1, \Sigma_1, R_1, S_1)$ and $G_2 = (V_2, \Sigma_2, R_2, S_2)$ be two context free languages such that $L_1 \in L(G_1)$ and $L_2 \in L(G_2)$. We have to show that the union of L_1 and L_2 $L_1 + L_2$ is also a CFL. To show this, we have to design a CFG that can process $L_1 + L_2$ i.e it can generate either the strings of G_1 or

the strings of G_2. Let $G = (V, \Sigma, R, S)$ be a CFG. Now, we have to define operational characteristics of G in such a way that G can generate all the string of G_1 or all the strings of G_2.
Now, its operational characteristics are defined as:

$V = V_1 \cup V_2 \cup S$
$\Sigma = \Sigma_1 \cup \Sigma_2$
R consists of -
$R_1 \cup R_2 \cup S \rightarrow S_1 | S_2$
S = start symbol

Here, S is the start symbol of G. The production rules $S \rightarrow S_1 | S_2$ tell that S can be replaced by either the start symbol of G_1 or the start symbol of G_2. When G uses $S \rightarrow S_1$, S is replaced by S_1 and then uses production rules R_1 and generates all the strings of G_1. Similarly, when G uses production rule, $S \rightarrow S_2$, then S is replaced by S_2, uses production rules R_2 and generates all the strings of G_2 i.e can generate either the strings of G_1 or the string of G_2.

Hence, we say that G is a CFG that generate all the strings of G_1 or G_2 i.e it can process $L_1 + L_2$. Thus, $L_1 + L_2$ is also a context free language.

(11) Concatenation

Let, $G_1 = (V_1, \Sigma_1, R_1, S_1)$ and $G_2 = (V_2, \Sigma_2, R_2, S_2)$ be two context free languages such that $L_1 \in L(G_1)$ and $L_2 \in L(G_2)$. We have to show the concatenation of L_1 and L_2 i.e $L_1 L_2$ is also CFL. To show this we have to design a CFG that can process $L_1 L_2$ i.e it can generate of G_1 followed by the strings of G_2. Let $G = (V, \Sigma, R, S)$ be a CFG.
Now, we have to define operational characteristics of G in such a way that G can generate the strings of G_1 followed by the string of G_2.
Now, its operational characteristics are defined as:

$V = V_1 \cup V_2 \cup S$
$\Sigma = \Sigma_1 \cup \Sigma_2$
R consists of -
$R_1 \cup R_2 \cup S \rightarrow S_1 S_2$
S = start Symbol

Here, S is the start symbol of G. The production rules $S \rightarrow S_1 S_2$ tells that the S can be replaced by $S_1 S_2$ i.e by using $S \rightarrow S_1 S_2$, it can be generate the strings of G_1 followed by strings of G_2. First it uses production rules R_1 and generates all strings of G_1 and immediately it uses R_2 and generates all the strings of G_2 i.e G can process $L_1 L_2$.

(iii) closure

Let $G_1 = (V_1, \Sigma_1, R_1, S_1)$ ~~and $G_2 = (V_2, \Sigma_2, R_2, S_2)$~~ be ~~this~~ context free grammar and $L_1 \in L(G_1)$, ~~$L_2 \in L(G_2)$~~ we have to show that the ~~concatenation~~ ~~of~~ closure denoted by L_1^* is also CFL.

To show that we have to design a CFG that can process L_1^* i.e it can generate the string of G_1.

Let $G = (V, \Sigma, R, S)$ be the CFG. Now, we have to define the operational characteristics of G in such a way that G can generate the string of G_1.

It's operation characteristics are:

$V = V_1 \cup S$

$\Sigma = \Sigma_1$

R consists of

$R_1 \cup S_1 \rightarrow$ ~~ϵ / SS_1~~ / $S \rightarrow \epsilon / SS_1$

S = start symbol

Here, S is the start symbol of G. The production rule $S \rightarrow \epsilon / SS_1$ tells S can be replaced by ϵ or SS_1 i.e by using $S \rightarrow \epsilon$, it can generate the empty string and $S \rightarrow SS_1$, it can ~~generate~~ replace S by SS_1. First it uses production rule R, and generates all string of G_1 number of times.

(1) Design a CFG that generates all the palindrome over $\Sigma = \{0, 1\}$.

(2) Design a CFG for the language $L = \{a^n b^n : n > 0\}$

$S \neq \epsilon$

→ $S \rightarrow 0S0 / 1S1 / 0 / 1 / \epsilon$

$G = (V, \Sigma, R, S)$

where,

$V = \{S\}$

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

R consist of -

$S \rightarrow 0S0 / 1S1 / 0 / 1 / \epsilon$

S: start symbol

$S \rightarrow 0S0 / 1S1 / \epsilon$

even →

odd

$S \rightarrow 0S0 / 1S1 / 0 / 1$

$w = 10101$	$w = 010$
$S \rightarrow 1S1$	$S \rightarrow 0S0$
$\rightarrow 10S01$	$\rightarrow 010$
$\rightarrow 101S101$	
$\rightarrow 10101$	

2.

→ $G = \{V, \Sigma, R, S\}$

where,

$V = \{S, A, B\}$

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

R consists of

$S \rightarrow AB / ab$

$A \rightarrow aA / a$

$B \rightarrow bB / b$

S: start symbol

$S \rightarrow asb$
$\rightarrow aaabb$

$S \rightarrow asb$
$\rightarrow aaSbb$
$\rightarrow aaabbb$

$\omega = 0^n1^n$
$\omega^R = 1^n0^n$
$S \rightarrow 0S1 | 01$

$\omega.\omega^R = 0^n1^n1^n0^n$
$S \rightarrow AB$
$A \rightarrow 0A1 \mid 01$
$B \rightarrow 1B0 \mid 10$

$L = \{a^n b^{2n} : n > 0\}$

$S \rightarrow asbb \mid abb$

→ Soln

$G = (V, \Sigma, R, S)$

where,

$V = \{S\}$

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

R consists of -

$S \rightarrow asbb|abb$

Example: aaabbbbbb

$S \rightarrow asbb$
$\rightarrow aasbbbb \quad [S \rightarrow asbb]$
$\rightarrow aaabbbbbb \quad [S \rightarrow abb]$

Pushdown automata:

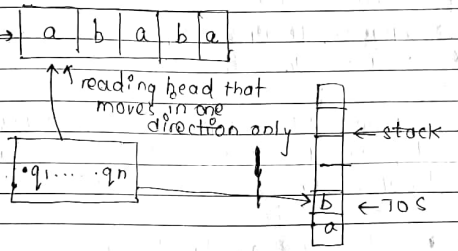


Fig: Block diagram of a PDA

Mathematical a PDA is defined as:

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

Where,

Q = finite set of states
Σ = finite set of inputs
δ = transition function.
q_0 = initial state
F = set of final state
Γ = stack symbols.

Move of a PDA

- The instant given by a transition function of a PDA is called its move.
- A move of a PDA is defined as:

$$\delta(q, X, Y) \rightarrow (p, z) \Rightarrow Q \times \Sigma \times \Gamma \rightarrow Q \times \Gamma$$

1.2	.3849	.3869	.3888	.3907	.3925	.3944	.3962	.3980	.3997	.4015
1.3	.4032	.4049	.4066	.4082	.4099	.4115	.4131	.4147	.4162	.4177
1.4	.4192	.4207	.4222	.4236	.4251	.4265	.4279	.4292	.4306	.4319
1.5	.4332	.4345	.4357	.4370	.4382	.4394	.4406	.4418	.4429	.4441
1.6	.4452	.4463	.4474	.4484	.4495	.4505	.4515	.4525	.4535	.4545
1.7	.4554	.4564	.4573	.4582	.4591	.4599	.4608	.4616		
1.8	.4641	.4649	.4656	.4664	.4671	.4678				
1.9	.4713	.4719								

where,

q = current state
x = symbol of input tape
Y = current top of stack
p = next state
z = symbol that replaces the current top of the stack and becomes new top of stack.

eg: push 'a' to the empty stack.

$\delta(q, a, \epsilon) \rightarrow (p, a)$
$\delta(q, a, a) \rightarrow (p, a, a)$

Pop 'a' from the stack

$\delta(q, a, a) \rightarrow (p, \epsilon)$

eg. Design a PDA for the language $L = \{a^n b^n : n > 0\}$

w = aaabbb

① $\delta(q, a, \epsilon) \rightarrow (f, a)$
② $\delta(f, a, a) \rightarrow (f, a, a)$
③ $\delta(f, b, a) \rightarrow (f, \epsilon)$

Let $M = (Q, \Sigma, \delta, q_0, F, \Gamma)$ be the PDA.

where,

$Q = \{q, f\}$
$\Sigma = \{a, b\}$
δ consists of

② $\delta(q, a, \epsilon) \rightarrow (f, a)$
③ $\delta(f, a, a) \rightarrow (f, aa)$
④ $\delta(f, b, a) \rightarrow (f, \epsilon)$

$q_0 = q$
$F = f$
$\Gamma = \{a\}$
w = aaabbb

State	Unread input	Stack Content	Transition used
q	aaabbb	empty	–
f	aabbb	a	①
f	abbb	aa	②
f	bbb	aaa	②
f	bb	aa	③
f	b	a	③
f	ϵ	ϵ	③

Since, the string & stack both becomes empty the string is accepted.

1.2		.3665	.3686	.3708						
1.3	.3849	.3869	.3888	.3907	.3729	.3749	.3770	.3790	.3599	.3621
1.4	.4032	.4049	.4066	.4082	.3925	.3944	.3962	.3980	.3997	.4015
	.4192	.4207	.4222	.4236	.4099	.4115	.4131	.4147	.4162	.4177
1.5	.4332	.4345	.4357	.4370	.4251	.4265	.4279	.4292	.4306	.4319
1.6	.4452	.4463	.4474	.4484	.4382	.4394	.4406	.4418	.4429	.4441
1.7	.4554	.4564	.4573	.4582	.4495	.4505	.4515	.4525	.4535	.4545
1.8	.4641	.4649			.4591	.4599	.4608	.4616	.4625	
1.9	.4713					.4678				

Acceptance of a string by PDA:

i) Acceptance by final state
ii) Acceptance by empty stack
↓
A string is said to be accepted if the stack becomes empty, when the string terminates.

Q. Design a PDA for the language L = {set of all the string over $\Sigma = \{a, b\}$ that contains equal number of a's and b's}

→ Let $M = (Q, \Sigma, \delta, q_0, F, \Gamma)$ be the PDA.
where,

$Q = \{q, f\}$ $\quad w = abaabb$
$\Sigma = \{a, b\}$
δ consists of

① $a \rightarrow \delta(q, a, \epsilon) \rightarrow (f, a)$
② $aa \rightarrow \delta(f, a, a) \rightarrow (f, aa)$
③ $b \rightarrow \delta(q, b, \epsilon) \rightarrow (f, b)$
④ $bb \rightarrow \delta(f, b, b) \rightarrow (f, bb)$
⑤ $ab \rightarrow \delta(f, b, a) \rightarrow (f, \epsilon)$
⑥ $ba \rightarrow \delta(f, a, b) \rightarrow (f, \epsilon)$

$q_0 = q$
$F = f$
$\Gamma = \{a\}$

state	unread input	stack content	Transition used
q	abaabb	empty	–
f	baabb	a	①
q	aabb	ϵ	③ ⑤
f	abb	a	①
f	bb	aa	②
f	b	a	⑤
f	ϵ	ϵ	⑤

Q. Design a PDA for the language $L = \{a^n b^{2n} ; n > 0\}$
→ Let $M = (Q, \Sigma, \delta, q_0, F, \Gamma)$ be the PDA.
where,

$Q = \{q, f\}$ $\quad w = aabbbb$
$\Sigma = \{a, b\}$
δ consists of

① $a \rightarrow \delta(q, a, \epsilon) \rightarrow (f, a)$ $\quad bb \rightarrow$
2 $b \rightarrow \delta(q, b, \epsilon) \rightarrow (f, b)$
3 $ab \rightarrow \delta(f, b, a) \rightarrow (f, \epsilon)$
4 $aa \rightarrow \delta(f, a, a) \rightarrow (f, aa)$
$q_0 = q$, $F = f$, $\Gamma = \{a\}$

state	unread input	stack content	transition
q	aabbbb	empty	–
f	abbbb	a	①
f	bbbb	aa	②
f	bbb	a	③
f	bb	ϵ	④

1.0	.3413	.3438	.3461	.3485	.3508	.3531	.3554	.3577	.3599	.3621
1.1	.3643	.3665	.3686	.3708	.3729	.3749	.3770	.3790	.3810	.3830
1.2	.3849	.3869	.3888	.3907	.3925	.3944	.3962	.3980	.3997	.4015
1.3	.4032	.4049	.4066	.4082	.4099	.4115	.4131	.4147	.4162	.4177
1.4	.4192	.4207	.4222	.4236	.4251	.4265	.4279	.4292	.4306	.4319
1.5	.4332	.4345	.4357	.4370	.4382	.4394	.4406	.4418	.4429	.4441
1.6	.4452	.4463	.4474	.4484	.4495	.4505	.4515	.4525	.4535	.4545
1.7	.4554	.4564	.4573	.4582	.4591	.4599	.4608	.4616	.4625	
1.8	.4641	.4649	.4656	.4664		.4678				
1.9	.4713									

① $\delta(q, a, \epsilon) \rightarrow (f, aa)$
② $\delta(f, a, a) \rightarrow (f, aaa)$
③ $\delta(f, b, a) \rightarrow (f, \epsilon)$

State	Unread input	Stack Content	Transition used
q	aabbbb	empty	—
f	abbbb	aa	①
f	bbbb	aaaa	②
f	bbb	aaa	③
f	bb	aa	③
f	b	a	③
f	empty	empty	③

CFG to PDA conversion:

① Push the start symbol to the empty stack.
② Then, push the right side string of the start symbol to the stack.
③ If the top of stack is a non-terminal then push the right side of the string of non-terminal to the stack.
④ If the top of stack is a terminal then read the symbol from the input string and pop the terminal from the stack.
⑤ Continue the process until all the symbols of input string are scanned.

eg:

$S \rightarrow aAS \mid aS \mid b$
$A \rightarrow aB \mid b$
$B \rightarrow bA \mid a$

$w = aabbab$
$S \rightarrow aAS$
$\rightarrow aaBS$
$\rightarrow aabAS$
$\rightarrow aabbS$
$\rightarrow aabbaS$
$\rightarrow aabbab$

① $\delta(q, \epsilon, \epsilon) \rightarrow (f, S)$
② $\delta(f, \epsilon, S) \rightarrow ((f, aAS), (f, aS), (f, b))$
③ $\delta(f, \epsilon, A) \rightarrow ((f, aB), (f, b))$
④ $\delta(f, \epsilon, B) \rightarrow ((f, bA), (f, a))$
⑤ $\delta(f, a, a) \rightarrow (f, \epsilon)$
⑥ $\delta(f, b, b) \rightarrow (f, \epsilon)$

Let $M = (Q, \Sigma, \delta, q_0, F, \Gamma)$ be a PDA where,

$Q = \{q, f\}$
$\Sigma = \{a, b\}$
δ consists of -

$q_0 = q$
$F = f$
$\Gamma = \{S, A, B, a, b\}$

1.1	.3643	.3665	.3686	.3485	.3508				.3365	.3389
1.2	.3849	.3869	.3888	.3708	.3729	.3749	.3770	.3790	.3599	.3621
1.3	.4032	.4049	.4066	.3907	.3925	.3944	.3962	.3980	.3997	.4015
1.4	.4192	.4207	.4222	.4082	.4099	.4115	.4131	.4147	.4162	.4177
1.5	.4332	.4345	.4357	.4236	.4251	.4265	.4279	.4292	.4306	.4319
1.6	.4452	.4463	.4474	.4370	.4382	.4394	.4406	.4418	.4429	.4441
1.7	.4554	.4564	.4573	.4484	.4495	.4505	.4515	.4525	.4535	.4545
1.8	.4641	.4649	.4656	.4582	.4591	.4599	.4608			
1.9	.4713	.4719		.4664	.4671	.4678				

state	unread input	stack content	transition used
q	aabbab	empty	—
f	aabbab	S	①
f	aabbab	aAs	②
f	abbab	As	⑤
f	abbab	aBs	③
f	bbab	Bs	⑤
f	bbab	bAs	④
f	bab	As	⑤
f	bab	bBs	③
f	ab	Bs	④
f	ab	as	①
f	b	s	⑤
f	b	b	①
f	ε	ε	⑥

Construct the equivalent PDA of the following CFG

S → 0BB/1S

B → 0A/1

A → 1B/0

Check whether the string ω = 100111 ∈ L(G)

ω = 1

→ Let M = (Q, Σ, δ, q0, f, Γ) be PDA where,

Q = {q, f}

Σ = {0, 1}

S → 0BB 1S

→ 10BB

→ 100BBB

→ 100111.

consists of -

① δ(q, t, ε) → (f, S)

② δ(f, t, S) → ((f, 0BB), (f, 1S))

③ δ(f, t, A) → ((f, 1B), (f, 0))

④ δ(f, t, B) → ((f, 0A), (f, 1))

⑤ δ(f, 1a, 1a) → (f, t)

⑥ δ(f, 0, 0) → (f, t).

state	unread input	stack content	transition use
q	100111	empty	—
f	100111	S	①
f	100111	1S	②
f	00111	S	⑤
f	00111	0BB	②
f	0111	BB	⑤
f	0111	0BBB	②
f	111	BBB	⑥
f	111	1BB	④
f	11	BB	⑥
f	11	1B	④
f	1	B	⑤
f	1	1	④
f	ε	ε	⑤

1.2	.3849	.3869	.3888	.3907	.3925	.3944	.3962	.3980	.3997	.4015
1.3	.4032	.4049	.4066	.4082	.4099	.4115	.4131	.4147	.4162	.4177
1.4	.4192	.4207	.4222	.4236	.4251	.4265	.4279	.4292	.4306	.4319
1.5	.4332	.4345	.4357	.4370	.4382	.4394	.4406	.4418	.4429	.4441
1.6	.4452	.4463	.4474	.4484	.4495	.4505	.4515	.4525	.4535	.4545
1.7	.4554	.4564	.4573	.4582	.4591	.4599	.4608			
1.8	.4641	.4649	.4656	.4664	.4671	.4678				
1.9	.4713	.4719								

Pumping lemma for CFL

Let L be a context free language and $z \in L$ such that $|z| \geq n$. Then, z can be decomposed into sub-string u, v, w, x and y such that -

1. $|vx| \geq 1$
2. $|vwx| \leq n$
3. $uv^k wx^k y \in L$ for all $k \geq 0$.

Proof:

Let us consider a CFG -

$S \to AB$

$A \to aB \mid a$

$B \to bA \mid b$

Let L be a CFL and $z \in L$.

Now, $z = ababb \in L$

Then, according to pumping lemma, we can decompose z into substring u, v, w, x and y as -

$u = a$

$v = ba$

$w = b$

$x = \epsilon$

$y = b$

Now, we construct a derivation tree for $z = ababbab$ -

uvwxy

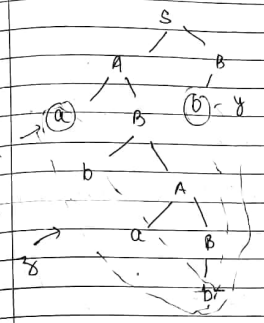


Fig: T

$z = uzy$, $z_1 = vz_2x$, $z_2 = w$

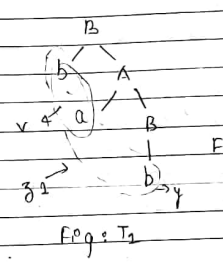


Fig: T_1

Fig: T_2

Here, T_1 is a proper subtree of T and T_2 is a proper subtree of T_1 and z_1, z_2 and z_2 are yields of T_1, T_1 and T_2 respectively.

From, above derivation trees, we can write

$z \to uzy$, $z_1 \to vz_2x$ and $z_2 \to w$.

Since, T is a tree with root S and T_1 and T_2 are trees with root B, so we can write -

$S \to uBy$, $B \to vBx$ and $B \to w$

Now,

$S \to uBy$ gives $S \to uvBxy \to uv^0wx^0y$, for $k = 0$

$S \to uBy$ gives $S \to uvBxy$

$\to uvwxy \in L$, for $k = 1$

$S \to uBy$ gives $S \to uvBxy$

$\to uvvBxxy$

1.5	.4332	.4345	.4357	.4370	.4382	.4394	.4406	.4418	.4429	.4441
1.6	.4452	.4463	.4474	.4484	.4495	.4505	.4515	.4525	.4535	.4545
1.7	.4554	.4564	.4573	.4582	.4591	.4599	.4608			
1.8	.4641	.4649	.4656	.4664		.4678				
1.9	.4713	.4719								

→ $uvwxy$
→ $uv^2wx^2y \in L$ for $k = 2$

Similarly,
$uv^kwx^ky \in L$, for $k \geq 0$

Eg. Show that the language $L = \{a^n b^n c^n : n > 0\}$ is not context free.

→ Let L be context free language ie $L = \{a^n b^n c^n : n \geq 0\}$ and $w = a^p b^p c^p \in L$
According to the pumping lemma, w can be decomposed into ~~three~~ five substring. ~~such that~~ u, v, w, x and y such that.

$u = a^q$
$v = a^r, r > 0$
$w = a^s$
$x = \epsilon$
$y = a^{p-(q+r+s)} b^p c^p$

$uv^2wx^2y = a^q a^{2r} a^s a^{p-(q+r+s)} b^p c^p$
$= a^{p+r} b^p c^p$

since $q, r \geq 0$, $p + r > p$
i.e $a^{p+r} b^p c^p$ is not of the form
$a^p b^p c^p$
$\therefore uv^2wxy \notin L$

Turning machine

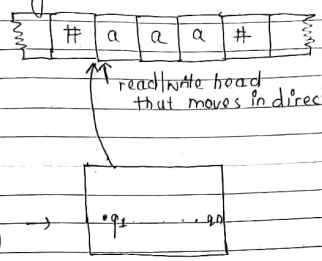


Fig: Block diagram of TM

A TM is defined as-
$M = (Q, \Sigma, \delta, q_0, F, \Gamma, \#)$
where,
Q = finite set of states
Σ = finite set of inputs
δ = transition function
q_0 = initial state
F = set of final states
Γ = symbols of input tape
$\#$ = blank symbol

The head of a TM can read from the input tape and can write the symbol back to the input tape. The write feature of a TM provides the facility of the storing the symbols permanently to the input tape.

1.3	.4032	.4049	.4066	.4082	.4099	.4115	.4131	.4147	.4162	.4177
1.4	.4192	.4207	.4222	.4236	.4251	.4265	.4279	.4292	.4306	.4319
1.5	.4332	.4345	.4357	.4370	.4382	.4394	.4406	.4418	.4429	.4441
1.6	.4452	.4463	.4474	.4484	.4495	.4505	.4515	.4525	.4535	.4545
1.7	.4554	.4564	.4573	.4582	.4591	.4599	.4608			
1.8	.4641	.4649								
1.9										

further, it can read in both directions i.e. a TM can re-read the content of any cell at any time so that the result of previous calculations can be used for future processing. Due to these features a TM is said to be functionally stronger than PDA and FA.

Move of a TM:-

A move of TM is of the form.

$\delta(q, X) \rightarrow (P, Y, D)$

Where,

q = current state

X = symbol of input tape or symbol under read/write head.

P = next state

Y = symbol that replaces the symbol of the input tape.

D = Direction of the next symbol to be read (L or R or N).

eg: $\delta(q_1, 0) \rightarrow (q_2, 1, R)$

$\delta(q_1, 0)$ → | 0 | 0 | 1 | 1 |

before processing ↑ q_1

- Instantaneous description (ID)

$0q_1011$

$(q_2, 1, R)$ → | 0 | 1 | 1 | 1 |

after processing ↑ q_2

$01q_211$

$0q_1011 \rightarrow 01q_211$

eg

$\delta(q, x_i) \rightarrow (p, y, L)$

$w = x_1 x_2 \ldots x_{i-1} x_i x_{i+1} \ldots x_n$

$x_1 x_2 \ldots x_{i-1} x_i x_{i+1} \ldots x_n$

↑ q_a

before processing → $x_1 x_2 \ldots x_{i-1} q_a x_i x_{i+1} \ldots x_n$

after processing → $x_1 x_2 \ldots p x_{i-1} y x_{i+1} \ldots x_n$

$x_1 x_2 \ldots x_{i-1} q x_i x_{i+2} \ldots x_n \rightarrow x_1 x_2 \ldots p x_{i-1} y x_{i+1} \ldots x_n$

Design of a TM:

1) Acceptance by final state
2) Acceptance by halt stat

- Halt is a special state that is used to determine

				.4236	.4251			.4147	.4162	.4177
1.5	.4332	.4345	.4357			.4265	.4279	.4292	.4306	.4319
1.6	.4452	.4463	.4474	.4370	.4382	.4394	.4406	.4418	.4429	.4441
1.7	.4554	.4564	.4573	.4484	.4495	.4505	.4515	.4525	.4535	.4545
1.8	.4641	.4649	.4656	.4582	.4591	.4599	.4608			
1.9	.4713					.4678				

the acceptance of a string.
- The difference between final state and halt state is that TM stops processing after transiting to halt state.

Design a TM that erases all the symbols of the strings over $\Sigma = \{a, b\}$.

w = dbaab
① $\delta(q_1, a) \rightarrow (q_2, \#, R)$
② $\delta(q_1, b) \rightarrow (q_2, \#, R)$
③ $\delta(q_2, a) \rightarrow (q_2, \#, R)$
④ $\delta(q_2, b) \rightarrow (q_2, \#, R)$
⑤ $\delta(q_2, \#) \rightarrow (h, \#, N)$

eg:
$q_1 abaab \rightarrow \#q_2 baab$
$\rightarrow \#\#q_2 aab$
$\rightarrow \#\#\#q_2 ab$
$\rightarrow \#\#\#\#q_2 b$
$\rightarrow \#\#\#\#\#q_2 \#$
$\rightarrow \#\#\#\#\#h$

Design a TM that accepts all the strings of even length over $\Sigma = \{a, b\}$.

~~$\delta(q_1, a) \rightarrow (q_2, \#, R)$~~
~~$\delta(q_1, b) \rightarrow (q_2, \#, R)$~~

① $\delta(q_1, a) \rightarrow (q_2, a, R)$
$\delta(q_1, b) \rightarrow (q_2, b, R)$
$\delta(q_2, a) \rightarrow (q_1, a, R)$
$\delta(q_2, b) \rightarrow (q_1, b, R)$
$\delta(q_1, \#) \rightarrow (h, \#, N)$

eg:

1.5	.4332	.4345	.4357	.4370	.4382	.4394	.4406	.4418	.4429	.4441
1.6	.4452	.4463	.4474	.4484	.4495	.4505	.4515	.4525	.4535	.4545
1.7	.4554	.4564	.4573	.4582	.4591	.4599	.4608			
1.8	.4641	.4649	.4656	.4664	.4671	.4678				
1.9	.4713	.4719	.4726							

Turing machine for computing functions:

Let $f(w) = u$ be a function. Then f is said to be Turing computable if there exists a TM such that -

$$\delta(q, \# w \#) \to (h, u)$$

When $n = 1$, $\#(f(n) = n+1), n > 0)$

$\#q_1 I \# \longrightarrow \# I q_2 \#$
$\to \# I I q_2 \#$
$\to \# I I h$

① $\delta(q_1, I) \to (q_1, I, R)$
② $\delta(q_1, \#) \to (q_2, I, R)$
③ $\delta(q_2, \#) \to (h, \#, N)$

When $n = 1$,
$\#q_1 I \# \to \# I q_1 \#$
$\to \# I I q_2 \#$
$\to \# I I h$

When $n = 2$,
$\#q_1 I I \# \to \# I q_1 I \#$
$\to \# I I q_1 \#$
$\to \# I I I q_2 \#$
$\to \# I I I h$

Similarly,
$\#q_1 I^n \to \cdots \to \# I^{n+1} h$

$f(n) = n+2, n \geq 0$

When $n = 1$,
(i) $\delta(q_1, I) \to (q_1, I, R)$
(ii) $\delta(q_1, \#) \to (q_2, I, R)$
(iii) $\delta(q_2, \#) \to (q_3, I, R)$
(iv) $\delta(q_3, \#) \to (h, \#, N)$

When $n = 1$,
$\#q_1 I \# \to \# I q_1 \#$
$\to \# I I q_2 \#$
$\to \# I I I q_3 \#$
$\to \# I I I h$

When $n = 2$,
$\#q_1 I I \# \to \# I q_1 I \#$
$\to \# I I q_1 \#$
$\to \# I I I q_2 \#$
$\to \# I I I I q_3 \#$
$\to \# I I I I h$

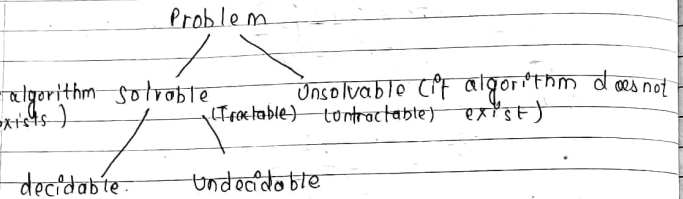
similary,

1.7	.4554	.4564	.4573	.4582	.4591	.4599	.4608			
1.8	.4641	.4649				.4678				
1.9										

Language of TM

1. Recursive language (decidable problems)
2. Recursively enumerable language (decidable + undecidable problems).

Decidability and undecidability

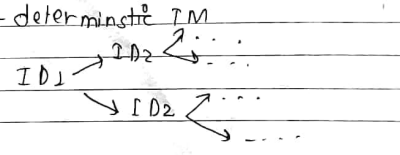


(if there exists a TM such that TM halts after processing an input)

(if there exists a TM such that TM never halts after processing on inputs).

Types of TM:-

1. Deterministic TM → ID1 → ID2 → . . . . → IDn
2. Non-deterministic TM



class P, class NP problems:-

- A problem is said to be in class P if there exists a deterministic TM such that it can solve the problem in polynomial time.
  eg Kruskal's algorithm.

- A problem is said to be in class NP if there exists a non-deterministic TM such that it can solve problem in polynomial time.
  eg Travelling salesman problem.

Recursive function theory:-

1. Partial function
2. Total function
3. Initial function
   - zero functions
   - successor functions
   - projection functions
4. Recursion and composition
5. Primitive recursive functions.