



TOC (Barsha Pokharel) - computer engineering notes

Computer engineering (Nepal Engineering College)



Scan to open on Studocu

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 reln or transition fxns

G = output reln

- A state reln is of the form -
(current state, input) \rightarrow next state

- A output reln is of the form -
(current state, input) \rightarrow (next, state, output)

eg : ON/OFF switch as a FSM.

$$S = \{ON, OFF\}$$

$$I = \{press\}$$

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

F consists of :

$$(ON, press) \rightarrow OFF$$

$$(OFF, press) \rightarrow ON$$

G consists of :

$$(ON, press) \rightarrow (OFF, FAN OFF)$$

$$(OFF, press) \rightarrow (ON, FAN ON)$$

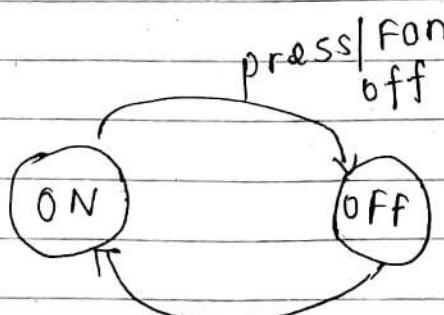


Fig : state diagram or transition diagram

state change		
S1	I	?
ON	OFF	ON OFF
OFF	ON	ON OFF

Finite state automata:

Alphabet:

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

String:

- combination or multiple occurrence of input symbols.
- It is denoted by w
- Eg: $\Sigma = \{0, 1\}$, $w = 1010$

Language

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

Finite state automata (FSA) 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 recognizer.

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 fn, that takes two argument in the form of state input and gives an argument in the form of state. A transition fn is of the form :

$$[Q \times \Sigma \rightarrow Q]$$

q_0 = initial or start state

F = set of final or accepting states

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

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

δ consists of

$$\begin{array}{l|l|l} \delta(q_1, 0) \rightarrow q_2 & \delta(q_2, 0) \rightarrow q_3 & \delta(q_3, 0) \rightarrow q_2 \\ \delta(q_1, 1) \rightarrow q_1 & \delta(q_2, 1) \rightarrow q_1 & \delta(q_3, 1) \rightarrow q_3 \\ q_0 = q_1 & & \\ F = q_2 & & \end{array}$$

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.
- final state is represented by double or concentric circle.

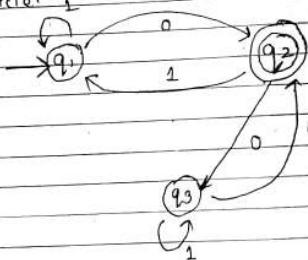


Fig : transition diagram

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 in front of it.

q _{1s}	0	1
→ q ₁	q ₂	q ₁
* q ₂	q ₃	q ₁
q ₃	q ₂	q ₃

Fig : Transition table

Processing of a string by FSA :-

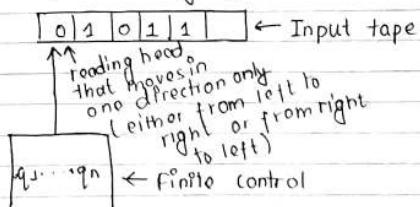


Fig : Block diagram of a FSA

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. A defined direction is either from left to right or from right to left.

Finite control is used to control the processing of FSA. In the initial state, intermediate states must be the states designed 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 after consuming a symbol, FSA regularly changes its states. During the process of consuming input symbols when an empty string is encountered in string terminator, if finite control gives any of the designed final states as next state, the string is said to be accepted otherwise rejected.

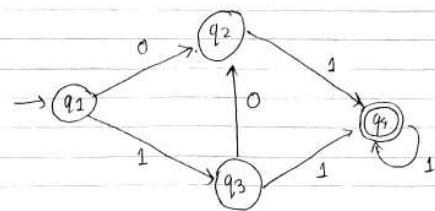


Fig: Finite automata

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

$$\begin{aligned}\delta(q_1, 100011) &\rightarrow \delta(q_3, 00011) \\ &\rightarrow \delta(q_2, 0011) \\ &\rightarrow \delta(q_2, 011) \\ &\rightarrow \delta(q_4, 11) \\ &\rightarrow \delta(q_4, 1) \\ &\rightarrow \delta(q_4, \epsilon)\end{aligned}$$

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

e.g. $w = 1000$

$$\begin{aligned}\delta(q_1, 1000) &\rightarrow \delta(q_3, 000) \\ &\rightarrow \delta(q_2, 00) \\ &\rightarrow \delta(q_2, 0) \\ &\rightarrow \delta(q_2, \epsilon)\end{aligned}$$

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)

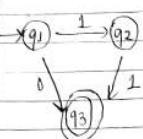


Fig : DFA

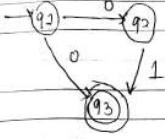
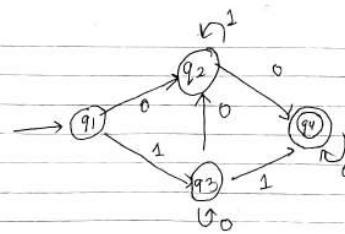


Fig : NFA



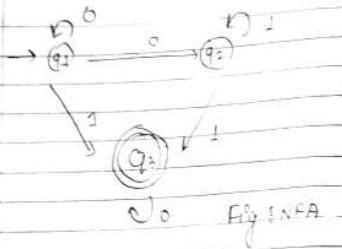
$w = 10001$

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 in terms 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.



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

The transition table of given NFA is:

$\delta' F$	0	1
$\rightarrow q_1$	$\{q_2, q_3\}$	$\{q_3\}$
$\rightarrow q_3$	$\{\emptyset\}$	$\{q_2, q_3\}$
q_2	$\{q_1\}$	$\{q_1, q_2\}$

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

$$\begin{aligned} \delta'(\emptyset, 0) &\rightarrow \delta(q_1, 0) \rightarrow \{q_1, q_2\} - \text{new state} \\ \delta'(\{q_1\}, 1) &\rightarrow \delta(q_1, 1) \rightarrow \{q_3\} - \text{old state} \\ \delta'(\{q_1, q_2\}, 0) &\rightarrow \delta(q_1, 0) \cup \{q_2, 0\} \\ &\rightarrow \{q_1, q_2\} \cup \{q_2\} \\ &\rightarrow \{q_1, q_2\} \end{aligned}$$

$$\begin{aligned} \delta'(\{q_1, q_2\}, 1) &\rightarrow \delta(q_1, 1) \cup \delta(q_2, 1) \\ &\rightarrow \{q_3\} \cup \{q_2, q_3\} \\ &\rightarrow \{q_2, q_3\} \rightarrow \text{new state} \end{aligned}$$

$$\begin{aligned} \delta'(\{q_2, q_3\}, 0) &\rightarrow \delta(q_2, 0) \cup \delta(q_3, 0) \\ &\rightarrow \{q_1\} \cup \{q_3\} \\ &\rightarrow \{q_1\} \end{aligned}$$

$$\begin{aligned} \delta'(\{q_2, q_3\}, 1) &\rightarrow \delta(q_2, 1) \cup \delta(q_3, 1) \\ &\rightarrow \{q_2, q_3\} \cup \{q_1\} \\ &\rightarrow \{q_2, q_3\} \end{aligned}$$

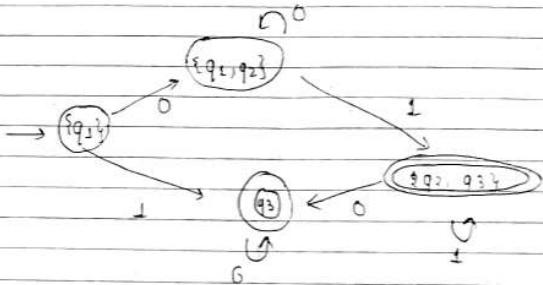


Fig 3 DFA

2013 Spring
Convert the NFA $M = \{q_0, q_1, q_2, q_3\}, \delta, \{q_0\}, \{q_0, q_1, q_2, q_3\}$ to its equivalent DFA δ' is given by

$\delta(\cdot)$	0	1
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\{q_2\}$	$\{q_2\}$
q_2	$\{q_3\}$	$\{q_3\}$
q_3	$\{q_3\}$	$\{q_2\}$

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

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

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

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

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

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

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

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

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

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

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

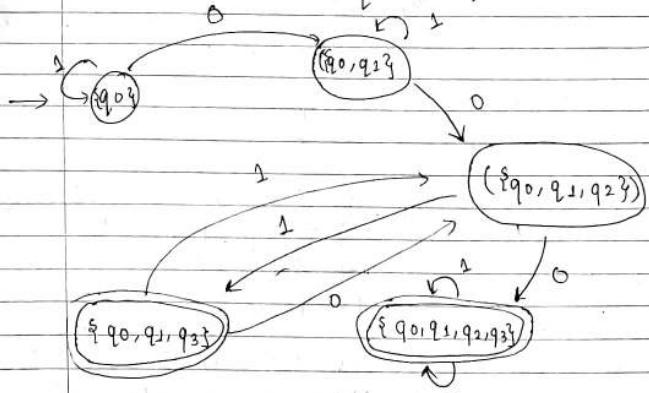
$$\delta'(\{q_0, q_1, q_2, q_3\}, 0) \rightarrow \delta'(q_0, 0) \cup (q_1, 0) \cup (q_2, 0) \cup (q_3, 0)$$

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

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

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

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



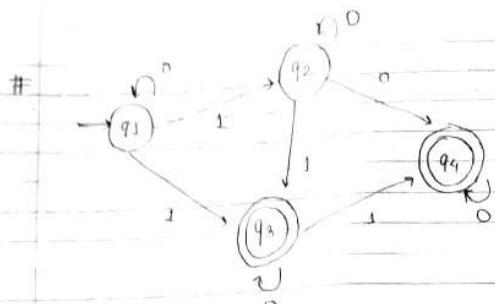


Fig. 9 NFA

All possible sets of	$\{\emptyset\}$	$\{q_1\}$	$\{q_2\}$	$\{q_3\}$	$\{q_4\}$
$\{q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_4\}$	$\{q_1, q_3\}$	$\{q_1, q_4\}$
$\{q_1, q_3\}$	$\{q_1, q_3\}$	$\{q_1, q_3, q_4\}$	$\{q_1, q_2, q_3\}$	$\{q_1, q_2\}$	$\{q_1, q_2, q_4\}$
$\{q_1, q_4\}$	$\{q_1, q_4\}$	$\{q_1, q_3, q_4\}$	$\{q_1, q_2, q_4\}$	$\{q_1, q_2\}$	$\{q_1, q_2, q_3\}$
$\{q_2, q_3\}$	$\{q_2, q_3\}$	$\{q_2, q_3, q_4\}$	$\{q_1, q_3\}$	$\{q_1, q_2\}$	$\{q_1, q_2, q_3\}$
$\{q_2, q_4\}$	$\{q_2, q_4\}$	$\{q_2, q_3, q_4\}$	$\{q_1, q_4\}$	$\{q_1, q_2\}$	$\{q_1, q_2, q_3\}$
$\{q_3, q_4\}$	$\{q_3, q_4\}$	$\{q_3, q_4\}$	$\{q_1, q_3\}$	$\{q_1, q_2\}$	$\{q_1, q_2, q_3\}$
$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_3, q_4\}$	$\{q_1, q_3, q_4\}$	$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_3, q_4\}$
$\{q_1, q_2, q_4\}$	$\{q_1, q_2, q_4\}$	$\{q_1, q_2, q_3, q_4\}$	$\{q_1, q_2, q_4\}$	$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_3, q_4\}$
$\{q_1, q_3, q_4\}$	$\{q_1, q_3, q_4\}$	$\{q_1, q_2, q_3, q_4\}$	$\{q_1, q_3, q_4\}$	$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_3, q_4\}$
$\{q_2, q_3, q_4\}$	$\{q_2, q_3, q_4\}$	$\{q_2, q_3, q_4\}$	$\{q_2, q_3, q_4\}$	$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_3, q_4\}$
$\{q_1, q_2, q_3, q_4\}$	$\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_3, q_4\}$			

Let δ' be the transition function of DFA being formed

$$\begin{aligned}\delta'(\{q_1\}, 0) &= \delta(q_1, 0) \rightarrow \{q_1\} \\ \delta'(\{q_1\}, 1) &= \delta(q_1, 1) \rightarrow \{q_2, q_3\} \rightarrow \text{New state} \\ \delta'(\{q_2, q_3\}, 0) &= \delta(q_2, 0) \cup \delta(q_3, 0) \\ &\quad \{q_2, q_4, q_3\} \rightarrow \text{New state}\end{aligned}$$

$$\begin{aligned}\delta'(\{q_2, q_3\}, 1) &= \delta(q_2, 1) \cup \delta(q_3, 1) \\ &= \{q_3, q_4\} \rightarrow \text{New state}\end{aligned}$$

$$\begin{aligned}\delta'(\{q_2, q_3, q_4\}, 0) &= \delta(q_2, 0) \cup \delta(q_3, 0) \cup \delta(q_4, 0) \\ &= \{q_2, q_4\} \cup \{q_3, q_4\} \cup \{q_4\} \\ &= \{q_2, q_3, q_4\}\end{aligned}$$

$$\begin{aligned}\delta'(\{q_2, q_3, q_4\}, 1) &= \delta(q_2, 1) \cup \delta(q_3, 1) \cup \delta(q_4, 1) \\ &= \{q_3, q_4\} \cup \{q_1\} \cup \{q_4\} \\ &= \{q_3, q_4\}\end{aligned}$$

$$\begin{aligned}\delta'(\{q_3, q_4\}, 0) &= \delta(q_3, 0) \cup \delta(q_4, 0) \\ &= \{q_3\}\end{aligned}$$

$$\begin{aligned}\delta'(\{q_3, q_4\}, 1) &= \delta(q_3, 1) \cup \delta(q_4, 1) \\ &= \{q_4\} \cup \{q_1\} \\ &= \{q_4\} \quad \text{not possible}\end{aligned}$$

~~q1~~

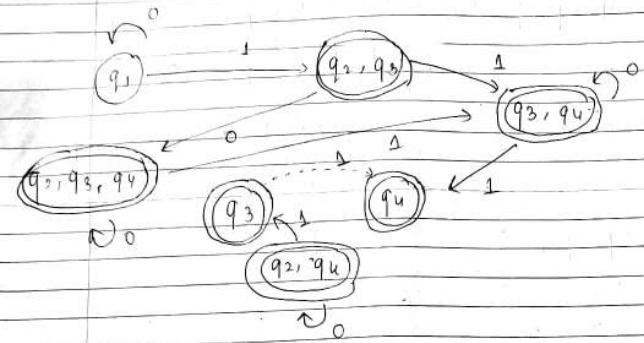
~~q3, q4~~

~~q2, q3~~

~~q4~~

$$\begin{aligned}\delta'(\{q_2, q_4\}, 0) &= \delta(q_2, 0) \cup \delta(q_4, 0) \\ &= \{q_2, q_4\} \cup \{q_4\} \\ &= \{q_2, q_4\}\end{aligned}$$

$$\begin{aligned}\delta'(\{q_2, q_4\}, 1) &= \delta(q_2, 1) \cup \delta(q_4, 1) \\ &= \{q_3\} \cup \{q_1\} \\ &= \{q_3\} \quad \text{not possible}\end{aligned}$$



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.



Fig :- ϵ -NFA

Epsilon closure

$$\epsilon\text{-closure } (q_1) = \{q_1, q_2, q_3\} \rightarrow \text{new}$$

$$\epsilon\text{-closure } (q_2) = \{q_2, q_3\} \rightarrow \text{new}$$

$$\epsilon\text{-closure } (q_3) = \{q_3\} \rightarrow \text{old.}$$

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.

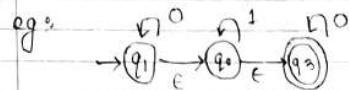


Fig :- DFA

Here,

$$\begin{aligned}\epsilon\text{-closure}(q_1) &= \{q_1, q_2, q_3\} \\ \epsilon\text{-closure}(q_2) &= \{q_2, q_3\} \\ \epsilon\text{-closure}(q_3) &= \{q_3\}\end{aligned}$$

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.

$$\begin{aligned}\delta'(\{q_1, q_2, q_3\}, 0) &\rightarrow \epsilon\text{-closure}(\delta(q_1, q_2, q_3), 0) \\ &\rightarrow \epsilon\text{-closure}(\delta(q_1, 0) \cup \delta(q_2, 0) \cup \delta(q_3, 0)) \\ &\rightarrow \epsilon\text{-closure}(q_1 \cup \emptyset \cup q_3) \\ &\rightarrow \epsilon\text{-closure}(q_1 \cup q_3) \\ &\rightarrow \epsilon\text{-closure}(q_1) \cup \epsilon\text{-closure}(q_3) \\ &\rightarrow \epsilon\text{-closure}(\{q_1, q_2, q_3\} \cup \{q_3\}) \\ &\rightarrow \{q_1, q_2, q_3\}\end{aligned}$$

$$\begin{aligned}\delta'(\{q_1, q_2, q_3\}, 1) &\rightarrow \epsilon\text{-closure}(\delta(q_1, q_2, q_3), 1) \\ &\rightarrow \epsilon\text{-closure}(\delta(q_1, 1) \cup \delta(q_2, 1) \cup \delta(q_3, 1)) \\ &\rightarrow \epsilon\text{-closure}(\emptyset \cup q_2 \cup \emptyset) \\ &\rightarrow \epsilon\text{-closure}(q_2) \\ &\rightarrow \{q_2\}\end{aligned}$$

$$\begin{aligned}\delta'(\{q_2, q_3\}, 0) &\rightarrow \epsilon\text{-closure}(\delta(q_2, q_3), 0) \\ &\rightarrow \epsilon\text{-closure}(\delta(q_2, 0) \cup \delta(q_3, 0)) \\ &\rightarrow \epsilon\text{-closure}(\emptyset \cup q_3) \\ &\rightarrow \epsilon\text{-closure}(q_3) \\ &\rightarrow \{q_3\}\end{aligned}$$

$$\begin{aligned}\delta'(\{q_2, q_3\}, 1) &\rightarrow \epsilon\text{-closure}(\delta(q_2, q_3), 1) \\ &\rightarrow \epsilon\text{-closure}(\delta(q_2, 1) \cup \delta(q_3, 1)) \\ &\rightarrow \epsilon\text{-closure}(q_2 \cup \emptyset) \\ &\rightarrow \{q_2\}\end{aligned}$$

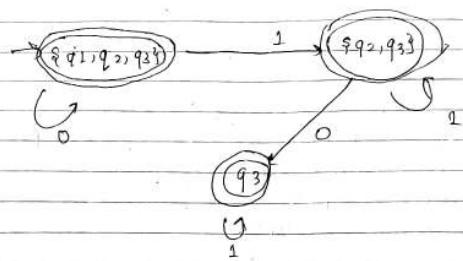
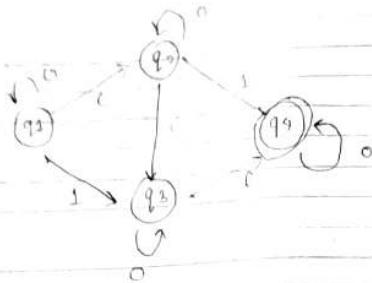


Fig 3 DFA A



epsilon closure

$$\begin{aligned}\text{-closure}(q_1) &= \{q_1, q_3\} \\ \text{-closure}(q_2) &= \{q_1, q_2, q_3, q_4\} \\ \text{-closure}(q_3) &= \{q_1, q_2, q_3\} \\ \text{-closure}(q_4) &= \{q_3\}\end{aligned}$$

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

$$\begin{aligned}\delta'(\{q_1, q_2, q_3, q_4\}, 0) &\rightarrow \text{-closure}(\delta(q_1, q_2, q_3, q_4), 0) \\ &= \text{-closure}(\delta(q_1, 0) \cup \delta(q_2, 0), \delta(q_3, 0) \cup \delta(q_4, 0)) \\ &= \text{-closure}(q_1 \cup q_2 \cup q_3 \cup q_4) \\ &= \{q_1, q_2, q_3, q_4\}\end{aligned}$$

$$\begin{aligned}\delta'(\{q_1, q_2, q_3, q_4\}, 1) &\rightarrow \text{-closure}(\delta(q_1, 1) \cup \delta(q_2, 1) \cup \delta(q_3, 1) \cup \delta(q_4, 1)) \\ &= \{q_1, q_2, q_3, q_4\}\end{aligned}$$

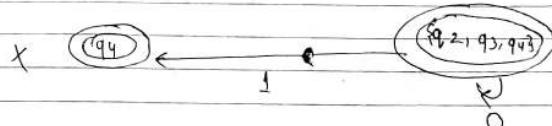
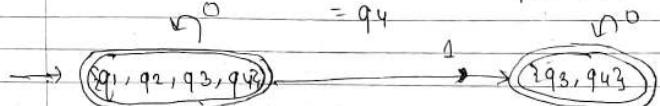
$$\begin{aligned}&= \text{-closure}(\{q_3\} \cup \{q_4\} \cup \emptyset \cup \emptyset) \\ &= \{q_3, q_4\} \quad \text{now}\end{aligned}$$

$$\begin{aligned}\delta'(\{q_3, q_4\}, 0) &\rightarrow \text{-closure}(\delta(q_3, 0) \cup \delta(q_4, 0)) \\ &= \text{-closure}(q_3 \cup q_4) \\ &= \{q_3, q_4\}\end{aligned}$$

$$\begin{aligned}\delta'(\{q_3, q_4\}, 1) &\rightarrow \text{-closure}(\delta(q_3, 1) \cup \delta(q_4, 1)) \\ &= \text{-closure}(\emptyset \cup \emptyset) \\ &= \emptyset\end{aligned}$$

$$\begin{aligned}\delta'(\{q_2, q_3, q_4\}, 0) &\rightarrow \text{-closure}(\delta(q_2, 0) \cup \delta(q_3, 0) \cup \delta(q_4, 0)) \\ &= \text{-closure}(q_2 \cup q_3 \cup q_4) \\ &= \{q_2, q_3, q_4\}\end{aligned}$$

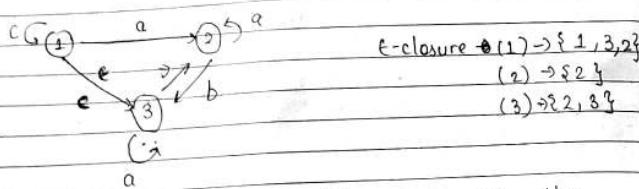
$$\begin{aligned}\delta'(\{q_2, q_3, q_4\}, 1) &\rightarrow \text{-closure}(\delta(q_2, 1) \cup \delta(q_3, 1) \cup \delta(q_4, 1)) \\ &= \text{-closure}(q_2 \cup q_3 \cup q_4) \\ &= \{q_2, q_3, q_4\}\end{aligned}$$



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	∅	∅	3
2	2	3	∅	∅
3	3	∅	∅	2



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

$$\begin{aligned} \delta'(\{1, 2, 3\}, a) &\rightarrow \delta(1, a) \cup \delta(2, a) \cup \delta(3, a) \\ &= \text{ε-closure } \{2 \cup 2 \cup 3\} \\ &= \text{ε-closure } \{2\} \cup \{3\} \\ &= \{2, 3\} \end{aligned}$$

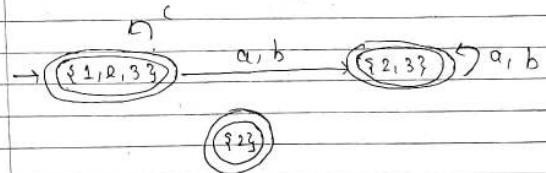
$$\begin{aligned} \delta'(\{1, 2, 3\}, b) &\rightarrow \text{ε-closure } \delta(1, b) \cup \delta(2, b) \cup \delta(3, b) \\ &= \text{ε-closure } \{ \emptyset \cup \emptyset \cup \{3\} \} \\ &= \{ \emptyset \cup \{3\} \} \\ &= \{ \emptyset, 3 \} \end{aligned}$$

$$\begin{aligned} \delta'(\{1, 2, 3\}, c) &\rightarrow \text{ε-closure } \delta(1, c) \cup \delta(2, c) \cup \delta(3, c) \\ &= \text{ε-closure } \{ \emptyset \cup \emptyset \cup \emptyset \} \\ &= \text{ε-closure } \{ \emptyset \} \cup \text{ε-closure } \{ \emptyset \} \\ &= \{ \emptyset, \emptyset \} \\ &= \{ \emptyset \} \end{aligned}$$

$$\begin{aligned} \delta'(\{2, 3\}, a) &\rightarrow \text{ε-closure } \delta(2, a) \cup \delta(3, a) \\ &= \{2, 3\} \end{aligned}$$

$$\begin{aligned} \delta'(\{2, 3\}, b) &\rightarrow \text{ε-closure } \delta(2, b) \cup \delta(3, b) \\ &= \text{ε-closure } \{ \emptyset \cup \emptyset \} \\ &= \{ \emptyset \} \end{aligned}$$

$$\begin{aligned} \delta'(\{2, 3\}, c) &\rightarrow \text{ε-closure } \delta(2, c) \cup \delta(3, c) \\ &\rightarrow \text{ε-closure } \{ \emptyset \cup \emptyset \} \\ &\rightarrow \{ \emptyset \} \end{aligned}$$



Regular Expressions

An expression used to generate string for FA is called regular expression or it is called language generation. A regular expression is recursively defined as:

- 1) empty string (ϵ) & every state (q_0) and symbol of input alphabet are regular expression.
- 2) If R_1 and R_2 are two regular expression then 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.
a) If R is a regular expression then closure of R denoted by R^* is also regular expression.
b) If R is regular expression then (R) is also regular expression.

e.g. $0 = (0+1)^*$

Q2

$$\begin{aligned} (0+1)^* &= \{ \epsilon, 0, 1, 00, 01, 10, 110, \dots \} \\ (0^* - 1^*) &= \{ \epsilon, 0, 00, 000, 0000, \dots \} \\ (01)^* &= \{ \epsilon, 01, 001, 0101, \dots \} \\ 0^* 1^* &= \{ \epsilon, 1, 10, 001, 00001, \dots \} \\ 1^* 0^* &= \{ 1, 11, 111, 1111, \dots \} \end{aligned}$$

Ques 3 Spring

Lespr of a FA that accepts a set of string such that every string end in 00 over alphabet {0, 1};
 $RE = (0+1)^* 00$

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.

Unions $S = \{0, 1\}$

$RE = 0+1$

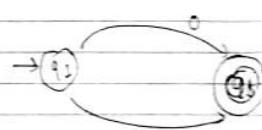
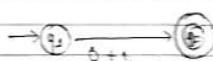


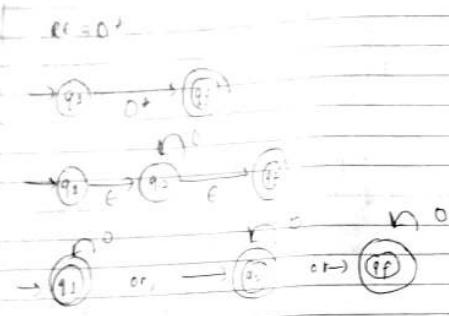
Fig 1 FA

Concatenation

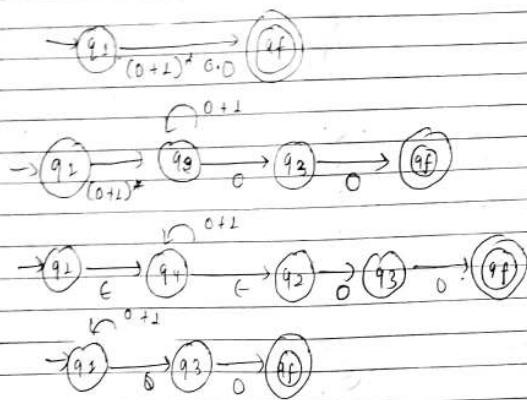
$RE = 0 \cdot 1$



Fig 2 FA



$$RE = (0+1)^* \cdot 0 \cdot 0$$



$q_1 \xrightarrow{0} q_2$

$q_2 \xrightarrow{0} q_3$

$q_3 \xrightarrow{0} q_f$

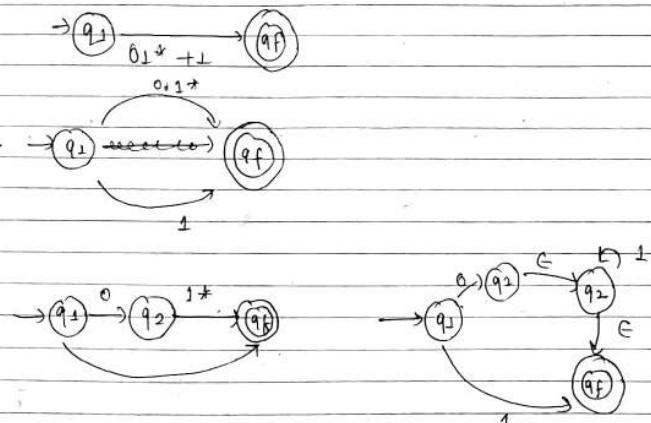
Fig : FA

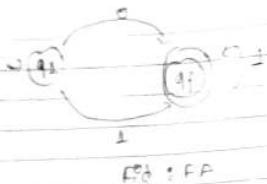
2014 Spring $RE = 01^* + 1$

2010 Spring $RE = a(a+b)^* bb$

2014 Fall $RE = (a+a(b+a)+b)^* a(b+a)^* a$

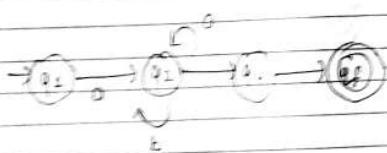
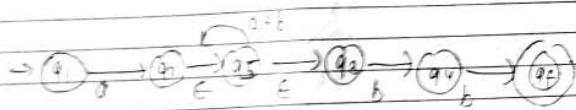
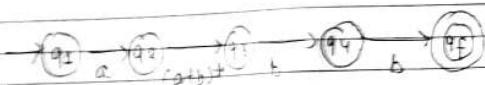
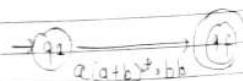
$RE = 01^* + 1$





FA : FP

RE : a(a+b)*bb



2014

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

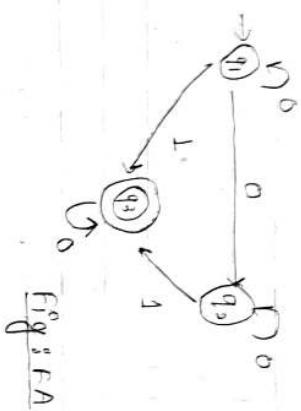
(ii) the regular expression :-

Auden's theorem :-
 Let P and Q be two regular expressions such that
 P does not contain empty string ϵ . Then, a regular
 expression $R = Q + RP$ of the form,

$$R = Q + RP$$

here, a unique solution

$$R = QP^*$$



$$\frac{q_1 = q \cdot 0 + e}{R} \rightarrow q_2 \rightarrow q_3$$

$$q_0 = q^{D+1} q_2 \cdot 0 \quad \text{--- (i)}$$

$$q_2 = q \cdot 1 + q_2 \cdot 1 + q_3 \cdot 0 \quad \text{--- (ii)}$$

From eqn ①

$$q_1 = q_1 \cdot 0 + \epsilon \\ R = Q \cdot P + S$$

Using theorem, we get

$$R = Q \cdot P^+ \\ i.e. q_2 = \epsilon \cdot 0^+ \\ q_2 = 0^+ - ④$$

From equation ② & ④, we get

$$q_2 = \underbrace{0^+ 0}_R + \underbrace{q_2 0}_P$$

Using Arden's theorem we get

$$q_2 = 0^+ 0 0^+ - ⑤$$

Using from equations ③, ④ & ⑤, we get,

$$q_2 = \underbrace{0^+ 1}_R + \underbrace{0^+ 0 0^+ 1}_Q + \underbrace{q_3 0}_P$$

Using Arden's theorem we get,

$$R = Q \cdot P^+ \\ q_3 = (0^+ 1 + 0^+ 0 0^+ 1) 0^+ \\ q_3 = 0^+ 1 0^+ + 0^+ 0 0^+ 1 0^+$$

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

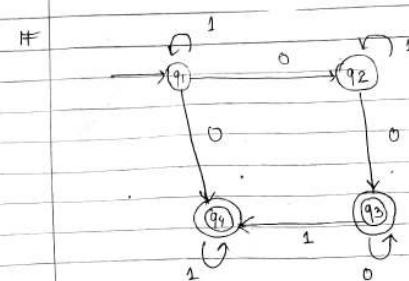


Fig: FA

$$\begin{aligned} q_1 &= q_1 \cdot 1 + \epsilon & ① \\ q_2 &= q_1 \cdot 0 + q_2 \cdot 1 & ② \\ q_3 &= q_2 \cdot 0 + q_3 \cdot 0 & ③ \\ q_4 &= q_2 \cdot 0 + q_3 \cdot 1 + q_4 \cdot 0 1 & ④ \end{aligned}$$

From eqn ①

$$q_1 = q_1 \cdot 1 + \epsilon \\ R = Q \cdot P + S$$

Using theorem, we get

$$i.e. q_1 = \epsilon \cdot 1^+ \\ q_1 = \epsilon \cdot 1^+ - ⑤$$

From eqn ② and ⑤ we get

$$q_2 = \underbrace{1^+ 0}_R + \underbrace{q_2 \cdot 1}_P$$

Using Arden's theorem:

$$f = QP^*$$

$$q_2 = 1^* 0 1^* - \textcircled{6}$$

From eqⁿ ②, ⑤, ⑥ we get

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

From eqⁿ ④, ⑥, ⑦ we get

Using Arden's theorem
$$q_3 = 1^* 0 \cdot 1^* 0 \cdot 0^* - \textcircled{7}$$

From eqⁿ 4, 5, 6, 7

$$q_4 = 1^* 0 + 1^* 0 \cdot 1^* 0 \cdot 0^* \cdot 1 + \underbrace{q_4 \cdot \frac{1}{P}}_{Q}$$

Using Arden's theorem

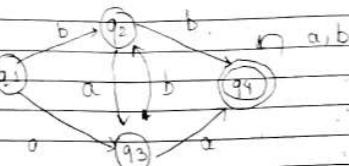
$$q_4 = (1^* 0 + 1^* 0 \cdot 1^* 0 \cdot 0^* 1)^{1^*}$$

$$q_4 = 1^* 0 \cdot 1^* + 1^* 0 \cdot 1^* 0 \cdot 0^* 1 1^*$$

Since q_2 and q_4 are final state, the required reg. expression is : $1^* 0, 1^* 0 \cdot 0^* + 1^* 0 \cdot 1^* + 1^* 0 \cdot 1^* 0 \cdot 0^* 1$

Ex 2.2.2 Fall

Find the regular expression for the following transition diagram



$$q_1 = \epsilon - \textcircled{1}$$

$$q_2 = q_1 \cdot b + q_3 \cdot b - \textcircled{1}$$

$$q_3 = 0 \cdot q_1 + 0 \cdot q_2 + \cancel{q_1 q_2} - \textcircled{1}, \textcircled{2}$$

$$q_4 = a q_1 + b q_4 + a q_3 + b q_2 - \textcircled{1}$$

$$q_1 = \epsilon \cdot q_2, b + q_3 \cdot b$$

~~$$q_2 = \epsilon \cdot b + q_3 \cdot b$$~~

$$q_2 = b + q_3 \cdot b$$

$$q_2 = b + q_1 a \cdot b + q_2 a b$$

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

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

$$R = Q P^*$$

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

$$q_2 = b(ab)^* + ab(ab)^* - \textcircled{1}$$

From (iii) - (i) & (v)

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

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

From (iii) + (i) & (v)

$$q_3 = q_1 \cdot a + (q_2 \cdot b + q_3 \cdot b) a$$

$$q_3 = b + q_2 \cdot ba + q_3 \cdot ba$$

R = QP*

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

$$q_3 = b(ba)^* + ba(ba)^* \quad - (v)$$

From (i), (ii), (iii), (iv)

$$\frac{q_4}{R} = \frac{(a+b)}{P} q_4 + a \left(b(ba)^* + ba(ba)^* \right) + b \left(b(ab)^* + (ab)^* \right)$$

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

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

This is not correct.

Closure properties of regular sets or language :-

The class of language of FA is closed under:-

(i) Union

(ii) Concatenation

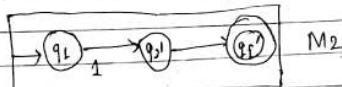
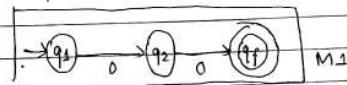
(iii) Kleene Closure

(i) 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 $M_1 = (Q_1, \Sigma, \delta, q_1, F_1)$ and $M_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2)$ 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:-



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

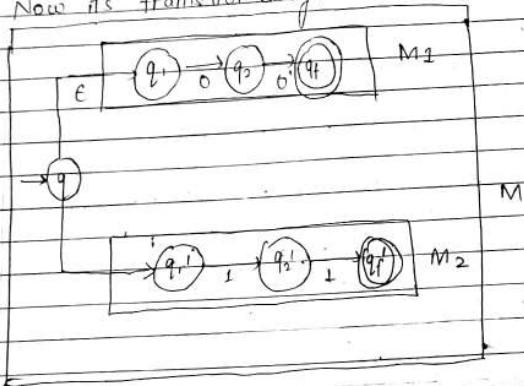
Its operational characteristics are defined as

$$\begin{aligned} Q &= Q_1 \cup Q_2 \cup q \\ \Sigma &= \Sigma_1 \cup \Sigma_2 \end{aligned}$$

$$\delta \text{ consists of } \delta_1 \cup \delta_2 \cup \delta(q, \epsilon) \rightarrow q_1, \delta(q, \epsilon) \rightarrow q_1'$$

$$\begin{aligned} q_0 &= q \\ F &= F_1 \cup F_2 \end{aligned}$$

Now its transition diagram is:



Here, q_0 is the initial state of M . The transition $\delta(q, \epsilon) \rightarrow q_1$ and $\delta(q, \epsilon) \rightarrow q_1'$ 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 $\delta(q, \epsilon) \rightarrow q_1$, it transits to initial state of M_1 , uses transition δ_1 and processes all strings generated by R_1 . Similarly, when M uses transition $\delta(q, \epsilon) \rightarrow q_1'$, it transits to initial state of M_2 , uses transition 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, $R_1 \cup R_2$ is also regular. Thus, the class language of FA is closed under union.

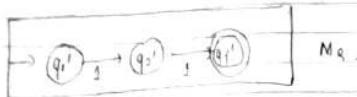
(ii) Concatenation:

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 processes the strings generated by R_1 and R_2 . Let $M_1 = (\mathcal{Q}_1, \Sigma_1, \delta_1, q_1, F_1)$ be the automata for R_1 and $M_2 = (\mathcal{Q}_2, \Sigma_2, \delta_2, q_2, F_2)$ be the finite automata for R_2 respectively.

Again, let $R_1 = \emptyset$ and $R_2 = \emptyset$

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 characteristics in such a way that it processes the strings of R_1 and followed by strings of R_2 .

i.e., its operational characteristics are defined as.

$$\delta = \delta_1 \cup \delta_2 \cup \delta_3$$

$$\Sigma = \Sigma_1 \cup \Sigma_2$$

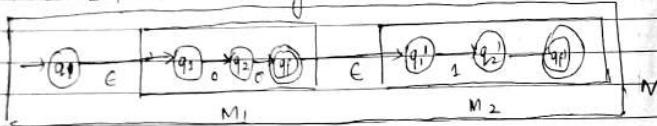
δ_1 consists of

$$\delta_1: \Sigma_1 \rightarrow Q \times \{q_1\} \quad \delta(q_1, \epsilon) \rightarrow q_1, \delta(q_1, \epsilon) \rightarrow q_1, q$$

$$q_0 = q_1$$

$$F_1 = F_2$$

Now its transition diagram δ_2 :



Here, q_1 is the initial state of M . The transition $\delta(q_1, \epsilon) \rightarrow q_1$ and $\delta(q_1, \epsilon) \rightarrow q_1'$ tells that M can transit to initial state of M_1 and $\delta(q_1, \epsilon) \rightarrow q_1'$ 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 .

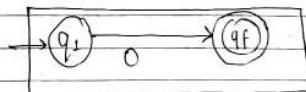
by consuming ϵ . After transitioning to initial state of M_2 , it uses transition δ_2 and processes all the strings generated by R_2 . Then it uses transition $\delta(q_1, \epsilon) \rightarrow q_1'$ and transits to initial state of M_2 . After that, it uses transition functions δ_2 and process all the 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.

b) closure

Let R be the regular expression. Then we have to show that the closure of R 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 $M = (Q, \Sigma, \delta, q_0, F)$ be the finite automata of R .

Again, $R = \emptyset$

Now, FA for R will be



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

$$Q = Q_1 \cup Q_2$$

$$\Sigma = \Sigma_1$$

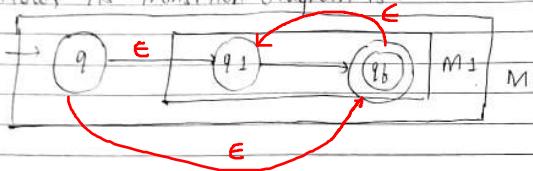
S consists of

$$\delta_1: \Sigma_1 \delta(q, \epsilon) \rightarrow Q_1, \quad \delta(q, \epsilon) \rightarrow q_f, \quad \delta(Q_f, \epsilon) \rightarrow Q_2$$

$$q_0 = q$$

$$f = F_1$$

Now, its transition diagram is:



Here, q is the initial state of M . The transition function $\delta(q, \epsilon) \rightarrow q_1$ tells that the initial state M can transits the initial state of M_1 uses transition function δ_1 and process all the strings generated by R . Similarly when M uses transition $\delta(q, \epsilon) \rightarrow q_f$ tells that it can process empty string and $\delta(q_f, \epsilon) \rightarrow q_2$ tells that P_1 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 $w \in L$ such that $|w| = n$. Let m be the total number of states and $m \leq n$. Then, w can be decomposed into three sub-string x, y and z such that:

$$1) y \neq \epsilon$$

$$2) |xy| \leq n$$

$$3) xy^iz \in L \text{ for all } i \geq 0$$

Proof:

Let L be regular language and $w \in L$. Let $m = m_1, m_2, \dots, m_m$ be the total no. of states.

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

$$x = a_1 a_2 \dots a_i$$

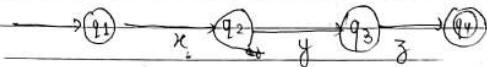
$$y = a_{i+1} a_{i+2} \dots a_j$$

$$z = a_{j+1} a_{j+2} \dots a_n$$

We know that

$$w = xyz \in L$$

Now, FA becomes P_1 ,

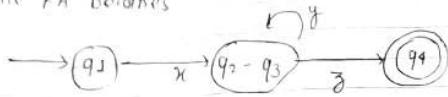


Here, $m - q \leq n = 3$

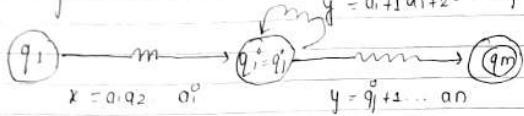
But, $m \leq n$. Therefore, according to the pigeon hole principle any two state must coincide. Let q_2 and q_3 be such states.

Now,

the FA becomes



It's generalize FA is



from the above FA, we can write

$$1 \leq i < j \leq m$$

$$1 \leq i < j \leq n$$

$$|xyz| \leq n$$

When

$$i = 0$$

$$x^0 = xz$$

when x is inputted to q_i , it transit to state $q_i (= q^0)$ and after consuming z it transit to state q_m (final state). Therefore, xz is accepted

$$\text{i.e. } xz \in L$$

When $i = 1$

$$w = xyz \in L$$

when $i = 2$

$$xy^2 \in L$$

similarly,

$$xy^i \in L \text{ for all } i \geq 0$$

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

\rightarrow Let $L = \{a^r b^r : r \geq 0\}$ is regular

and $w = a^p b^p \in L$

According to pumping lemma, w can be decomposed into three sub-string x, y, z as:

$$x = a^{\frac{p}{3}}$$

$$y = a^{\frac{p}{3}}$$

$$z = a^{\frac{p}{3}} b^p$$

$$\text{Now, } xy^2z = a^{\frac{p}{3}} a^{\frac{p}{3}} a^{\frac{p}{3}} b^p = a^{\frac{p}{3} + \frac{p}{3} + \frac{p}{3}} b^p = a^p b^p$$

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

ie $a^{p+p} b^p$ is not of the form of $a^p b^p$

$$1 \cdot 2 \cdot 2 \cdot 2 \notin L$$

Hence, L is not regular.

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

\rightarrow Let $L = \{1^r 2^r : r \geq 0\}$ is regular

and $w = 1^p 2^p \in L$

According to pumping lemma, w can be decomposed into three sub-string x, y, z as:

$$x = 1^{\frac{p}{3}}$$

$$y = 1^{\frac{p}{3}}$$

$$z = 1^{\frac{p}{3}}$$

$$\begin{aligned} \text{Now, } \\ xy^2z &= 1^{\frac{p}{3}} 1^{\frac{p}{3}} 1^{\frac{p}{3}} 2^p = 1^p 2^p \\ &= 1^p 2^p = 2^p + p^2 - p^2 - p^2 \\ &= 1^p 2^p \end{aligned}$$

Since, $p^2 > 0$, $p^2 + p^2 > p^2$

i.e. $1^p 2^p - p^2$ is not of the form $1^p 2^p$

$$1^p 2^p - p^2 \notin L$$

Hence, L is not regular.

A reverse

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

\rightarrow Let $L = \{xx^* : x \in \{0, 1\}^*\}$

$x \in \{0, 1\}^*$

i.e. $L = \{0^r 1^s 0^t 1^u : r, s, t, u \geq 0\}$ is regular

According to pumping lemma, w can be decomposed into three sub-string x, y, z as:

$$x = 0^r$$

$$y = 0^{r+1} 1^s 1^u 0^t$$

$$\text{Now, } xy^2z = 0^r 0^{2r} 0^p - 1^s 1^p 0^p$$

$$= 0^{r+2r} 0^p - 1^s 1^p 0^p$$

$$= 0^{r+1} 1^s 1^p 0^p$$

Since $t > 0$, $p+t > p$

i.e. L^P is not of the form $O^P 1 P 1^P O^P$
Hence, L^P is not regular.

H Grammar

A grammar is defined as:

$$U = (\mathbb{V}, \mathcal{E}_U, R_U, S)$$

where V = set of non-terminals

Σ set of terminals

R - set of production rules

s = start symbol

e.g.: for the regular expression $R = a(a^* + b^*)b$,
equivalent grammar is -

equivalent grammar
(\equiv , \sim , β , δ)

$G = \{v_1, v_2, R, S\}$

$\lambda = \delta \in M(A, B)$

E consists of

basis of

$A_1 \rightarrow A_1 B$

$$M \rightarrow A/B$$

$$B \rightarrow e/bB$$

$$\begin{array}{l}
 \text{Derivation} \\
 \left. \begin{array}{l}
 \text{S} \rightarrow a M b \\
 \rightarrow a B b \quad [M \rightarrow B] \\
 \rightarrow a B B B \quad [\vdash B \rightarrow B B] \\
 \rightarrow a b b B b \quad [\vdash B \rightarrow B B] \\
 \rightarrow a b b B b b \quad [\vdash B \rightarrow B B] \\
 \rightarrow a b b b b \quad [\vdash B \rightarrow E] \\
 \rightarrow a b b b b b
 \end{array} \right\} \text{on } a
 \end{array}$$

Construct an equivalent grammar of the following regular expression.

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

I use this rule

$$\begin{cases}
 S \rightarrow A/B \\
 A \rightarrow 0C01D \\
 D \rightarrow D11E \\
 C \rightarrow E/1C \\
 D \rightarrow E/0D \\
 E \rightarrow E/1E/1F
 \end{cases}$$

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 -

$$w_1 \alpha w_2 \rightarrow w_1 \beta w_2$$

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 -

$$\alpha \rightarrow \beta$$

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

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

non-terminal \rightarrow exactly one terminal

or

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

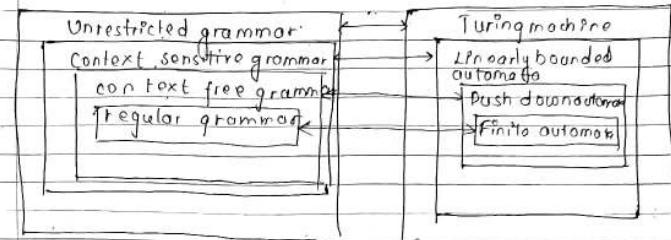


fig : types of grammar with their respective mathematical models.

Derivations:-

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

e.g. $(w_1, w_2 \dots w_n)$

$$\boxed{w_1 \xrightarrow{R} w_2 \xrightarrow{R} \dots \xrightarrow{R} w_n}$$

- There are two types of derivations

① Leftmost derivation.

② Rightmost derivation

e.g.:

$$S \rightarrow AB$$

$$A \rightarrow aA \mid b$$

$$B \rightarrow bB \mid a$$

$$w = abba$$

Leftmost derivation:-

$$\begin{aligned} S &\rightarrow AB \\ &\rightarrow aAB [A \rightarrow aA] \\ &\rightarrow aaAB [A \rightarrow aA] \\ &\rightarrow aabB [A \rightarrow b] \\ &\rightarrow aabbB [A \rightarrow bB] \\ &\rightarrow aabbba [A \rightarrow bB] \\ &\rightarrow aabbba [A \rightarrow b] \end{aligned}$$

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

Leftmost derivation:

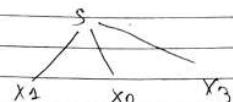
$$\begin{aligned} S &\rightarrow aAS / aS / b \\ &\rightarrow aAS [A \rightarrow bB] \\ &\rightarrow abBS [B \rightarrow aA] [A \rightarrow bB] \\ &\rightarrow abaAS [B \rightarrow aA] [B \rightarrow aA] \\ &\rightarrow abaaAS [B \rightarrow aA] [B \rightarrow aA] \\ &\rightarrow abaaaAS [B \rightarrow aA] \\ &\rightarrow abaaaaAS [B \rightarrow aA] \\ &\rightarrow aaaaaaaaa [A \rightarrow a] \\ &\rightarrow obaaaaab [S \rightarrow b] \end{aligned}$$

Rightmost derivation:-

$$\begin{aligned} S &\rightarrow AB \\ &\rightarrow ABB [A \rightarrow bB] \\ &\rightarrow AbbB [A \rightarrow bB] \\ &\rightarrow Abba [A \rightarrow a] \\ &\rightarrow aAbba [A \rightarrow aA] \\ &\rightarrow aaAbba [A \rightarrow aA] \\ &\rightarrow aabbba [A \rightarrow b] \end{aligned}$$

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.
e.g. if $S \rightarrow X_1 X_2 X_3$ be the production rule, then its derivation tree is -



Rightmost derivation:

$$\begin{aligned} S &\rightarrow aAS \\ &\rightarrow aA [S \rightarrow aS] \\ &\rightarrow aAa [S \rightarrow aS] \\ &\rightarrow aAAAa [S \rightarrow aS] \\ &\rightarrow aAAAAa [S \rightarrow aS] \\ &\rightarrow aAAAAAA [S \rightarrow aS] \\ &\rightarrow aAaaaaa [S \rightarrow aS] \\ &\rightarrow aAaaaaab [S \rightarrow b] \\ &\rightarrow abBaaaaab [A \rightarrow bB] \\ &\rightarrow abAaaaaab [B \rightarrow aA] \\ &\rightarrow abaaaaab [A \rightarrow a] \end{aligned}$$

Eg:

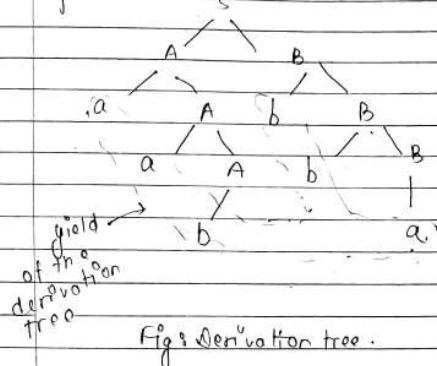
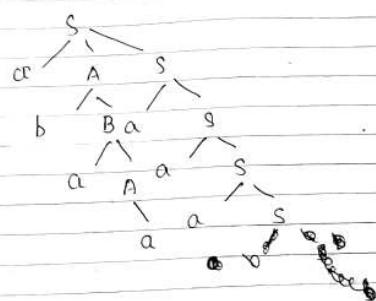
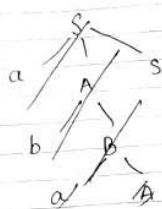


Fig: Derivation tree.



Ambiguity in CFG:

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

e.g.:

$$\begin{array}{l} S \rightarrow aAs/b \\ A \rightarrow bB/ba \\ B \rightarrow a \end{array}$$

is an ambiguous CFG

$$w = abab$$

LMD :-

$$S \rightarrow aAs$$

$\rightarrow abab$ [A \rightarrow b]

$$S \rightarrow aAS$$

$\rightarrow abBb$ [A \rightarrow B]

$$S \rightarrow aAS$$

$\rightarrow abab$ [A \rightarrow a]

$$S \rightarrow aAS$$

$\rightarrow abab$ [A \rightarrow b]

LMD :-

$$S \rightarrow aAs$$

$\rightarrow abab$ [A \rightarrow b]

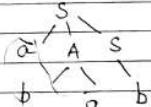
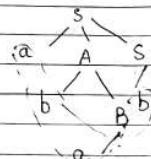


Fig: Derivation tree

Show that the following CFG is ambiguous.

$$E \rightarrow E + F | F * E | a | b$$

$$w = a + b * a + b$$

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

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

$$E \rightarrow E + F$$

$$\rightarrow F + E * E$$
 ($E \rightarrow E + E$)

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

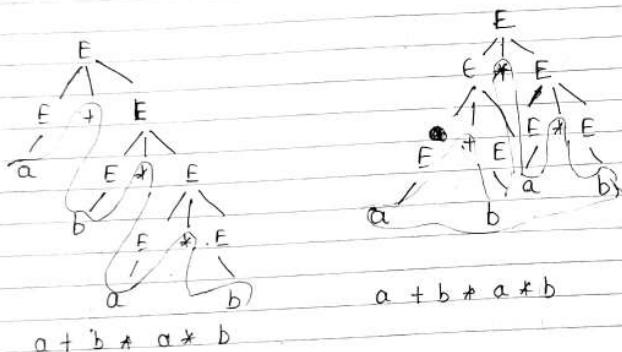
$\rightarrow E + b + a + b$ ($E \rightarrow b$)

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

$\rightarrow E + E$ [$E \rightarrow E + E$]
 $\rightarrow a + E$ [$E \rightarrow a$]
 $\rightarrow a + E * E$ [$E \rightarrow E * E$]
 $\rightarrow a + b * E$ [$E \rightarrow b$]
 $\rightarrow a + b * E * E$ [$E \rightarrow E * E$]
 $\rightarrow a + b * a * E$ [$E \rightarrow a$]
 $\rightarrow a + b * a * b$ [$E \rightarrow b$]

LMD

$\rightarrow E * E$
 $\rightarrow E * E + E$ [$E \rightarrow E + E$]
 $\rightarrow a + E * E$ [$E \rightarrow a$]
 $\rightarrow a + b * E$ [$E \rightarrow b$]
 $\rightarrow a + b * E * E$ [$E \rightarrow E * E$]
 $\rightarrow a + b * a * E$ [$E \rightarrow a$]
 $\rightarrow a + b * a * b$ [$E \rightarrow b$]



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

2014 Fall

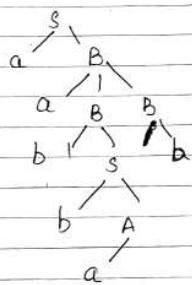
Check ambiguity of the grammar

$S \rightarrow aB | bA$
 $A \rightarrow aS | bAA | a$
 $B \rightarrow abbB | bs | b$

$S \rightarrow aB$
 $\rightarrow aabbB$ [$B \rightarrow aBB$]
 $\rightarrow aabbbs$ [$B \rightarrow BS$]
 $\rightarrow aabb$
 $\rightarrow aabbB$ [$B \rightarrow bBB$]
 $\Rightarrow aabsB$ [$B \rightarrow bs$]
 $\rightarrow aabbabb$ [$B \rightarrow b$]
 $\rightarrow aabbabb$ [$B \rightarrow b$]
 $\rightarrow aabbabb$ [$B \rightarrow b$]

$S \rightarrow aB$
 $\rightarrow aabbB$ [$B \rightarrow aBB$]
 $\rightarrow aabbbs$ [$B \rightarrow bs$]
 $\rightarrow aabbAB$ [$S \rightarrow bA$]
 $\rightarrow aabb$

LMD

$$\begin{aligned} S &\rightarrow aB \quad [\because S \rightarrow aB] \\ \rightarrow a &BB \quad [B \rightarrow aBB] \\ \rightarrow a &aBS \quad [B \rightarrow BS] \\ \rightarrow a &abbAB \quad [B \xrightarrow{S} bA] \\ \rightarrow a &abbAB \quad [A \rightarrow a] \\ \rightarrow a &abbab \quad [B \rightarrow b] \end{aligned}$$


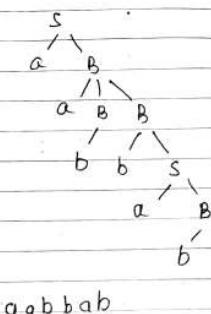
aabbab

Since, there exists 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)

LMD

$$\begin{aligned} S &\rightarrow aB \\ \rightarrow a &aBB \quad [B \rightarrow aBB] \\ \rightarrow a &abB \quad [B \rightarrow b] \\ \rightarrow a &abBS \quad [B \rightarrow Bs] \\ \rightarrow a &abbAB \quad [B \rightarrow aB] \\ \rightarrow a &abbAB \quad [S \rightarrow aB] \\ \rightarrow a &abbab \quad [B \rightarrow b] \end{aligned}$$


aabbab

Chomsky Normal Form (CNF):

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

- Non-terminal \rightarrow exactly one terminal
- or
- Non-terminal \rightarrow exactly two non-terminals.

e.g. $S \xrightarrow{[S \rightarrow a, S \rightarrow AB]}$, etc.

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 \rightarrow exactly one terminal

or

Non-terminal \rightarrow exactly two non-terminals

Method

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.

e.g. $S \rightarrow a$ be the production rule. To convert it

into its equivalent CNF, we replace 'a' by a nonterminal, say c_1 , so that production rule becomes $s \rightarrow c_1 a$. But this is possible only when a production rule $c_1 \rightarrow a$ exists. So, we add $c_1 \rightarrow a$ 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.

$$S \rightarrow aAs|as$$

$$A \rightarrow abB$$

$$B \rightarrow bB|baa$$

$$1. S \rightarrow aAs \text{ gives } S \rightarrow c_1 aS \text{ where } c_1 \rightarrow a$$

$$S \rightarrow c_1 aS \text{ gives } S \rightarrow c_1 c_2 S \text{ where } c_2 \rightarrow aS$$

$$2. S \rightarrow aS \text{ gives } S \rightarrow c_1 S \text{ where } c_1 \rightarrow a$$

$$3. A \rightarrow abB \text{ gives } A \rightarrow c_1 B c_2 \text{ where } c_1 \rightarrow a, c_2 \rightarrow b$$

$$A \rightarrow c_1 B c_2 \text{ gives } A \rightarrow c_1 c_3 c_4 \text{ where } c_3 \rightarrow Bc_2$$

$$4. B \rightarrow bB \text{ gives } B \rightarrow c_3 B \text{ where } c_3 \rightarrow b$$

$$5. B \rightarrow baA \text{ gives } B \rightarrow c_3 A c_4 \text{ where } c_3 \rightarrow b, c_4 \rightarrow a$$

$$B \rightarrow c_3 A c_4 \text{ gives } B \rightarrow c_5 c_6 c_7 \text{ where } c_5 \rightarrow c_3 A, c_6 \rightarrow c_4$$

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 -

$$S \rightarrow C_1 C_2 | C_1 S, A \rightarrow C_1 C_4, B \rightarrow C_3 B | C_5 C_1,$$

$$C_2 \rightarrow AS, C_4 \rightarrow BC_3, C_5 \rightarrow C_3 A, (1) A, C_3 \rightarrow b\}$$

$$S' = S$$

2016 Spring

Consider the grammar $G = (V, \Sigma, P, S)$ where,
 $V = \{S, A, B\}$

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

and the productions P are:

$$S \rightarrow bA | ab$$

$$A \rightarrow bAA | as | a$$

$$B \rightarrow aBB | bB | b$$

Find an equivalent grammar in CNF.

$$1. S \rightarrow bA \text{ gives } S \rightarrow A \text{ where, } B \rightarrow b$$

$$2. S \rightarrow ab \text{ gives } S \rightarrow A \text{ where, } A \rightarrow a$$

$$3. A \rightarrow bAA \text{ gives } A \rightarrow BA A \text{ where } A \rightarrow B \rightarrow B \rightarrow b$$

$$A \rightarrow B AA \text{ gives } A \rightarrow C_1 A \text{ where } B \rightarrow C_1 \rightarrow BA$$

$$4. A \rightarrow as \text{ gives } A \rightarrow AS \text{ where } A \rightarrow a$$

$$5. B \rightarrow aBB \text{ gives } B \rightarrow ABB \text{ where } A \rightarrow a$$

$$B \rightarrow AB B \text{ gives } C_2 \rightarrow AP \text{ where } C_2 \rightarrow AB$$

$$6. B \rightarrow bB \text{ gives } B \rightarrow B \text{ where } B \rightarrow b, B \rightarrow b$$

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

$$\{ S \rightarrow AB | ABA | BAA, A \rightarrow AS | GA | AB \rightarrow C_2 B | BB | B, C_1 \rightarrow BA, C_2 \rightarrow AB \}$$

ii Greibach normal form (GNF):

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

$$\alpha \rightarrow \alpha\beta^*$$

where, $\alpha, \beta \in V$ and $\alpha \in \Sigma$.

Non-terminal \rightarrow exactly one terminal

or

Non-terminal \rightarrow one terminal followed by any number of non-terminals.

e.g.: $S \rightarrow a, S \rightarrow \alpha A, S \rightarrow \alpha A B$, etc.

iii Simplification of CFG:

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

i. Removal of unit productions:-

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

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

e.g.: 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:

$$\begin{aligned} A &\rightarrow B \\ B &\rightarrow C \\ C &\rightarrow D \\ D &\rightarrow E \end{aligned}$$

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 $D \rightarrow \epsilon$.

Similarly, we $C \rightarrow a$, $B \rightarrow a$ and $A \rightarrow a$ to the given CFG and eliminate unit productions $C \rightarrow D$, $B \rightarrow C$ and $A \rightarrow B$.

Hence, after eliminating unit productions, the simplified CFG is

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow a \\ B &\rightarrow a \\ C &\rightarrow a \\ D &\rightarrow a \\ \epsilon &\rightarrow a \end{aligned}$$

2. Removal of null productions:

A production rule of the form
non-terminal \rightarrow empty string
is called null production.

e.g. $S \rightarrow \epsilon$, $A \rightarrow \epsilon$, etc.

Let $\alpha \rightarrow \epsilon$ be the null production. To eliminate $\alpha \rightarrow \epsilon$ from the given CFG, we search for all the occurrence 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 $\alpha \rightarrow \epsilon$. This process is continued until all the null production

are eliminated.

e.g. simply the following grammar by eliminating null productions.

$$\begin{aligned} S &\rightarrow aABs \mid AB \\ A &\rightarrow aA \mid \epsilon \\ B &\rightarrow bB \mid \epsilon \end{aligned}$$

→ Soln :-

Here, the null productions are :-

$$\begin{aligned} A &\rightarrow \epsilon \\ B &\rightarrow \epsilon \end{aligned}$$

To eliminate $A \rightarrow \epsilon$, we search for the occurrence of A in the production rules of given CFG.

$$\begin{aligned} C &\rightarrow aABs \text{ gives } S \rightarrow aBs \\ S &\rightarrow AB \text{ gives } S \rightarrow B \\ A &\rightarrow aA \text{ gives } A \rightarrow a \end{aligned}$$

After eliminate $A \rightarrow \epsilon$, the CFG becomes -

$$\begin{aligned} S &\rightarrow aABs \mid AB \mid B \\ A &\rightarrow aA \mid a \\ B &\rightarrow bB \mid \epsilon \end{aligned}$$

To eliminate $B \rightarrow \epsilon$, we search for occurrences of B in the production rules of new CFG.

Now,

$$\begin{aligned} S &\rightarrow aABs \text{ gives } S \rightarrow aBs \\ S &\rightarrow AB \text{ gives } S \rightarrow A \end{aligned}$$

$s \rightarrow aBS$ gives $s \rightarrow aS$
 $s \rightarrow B$ gives $s \rightarrow \epsilon$
 $s \rightarrow bB$ gives $s \rightarrow b$

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

$S \rightarrow aABS | AB | aBS | B | aAs | AaS | b | \epsilon$
 $A \rightarrow aAa$
 $B \rightarrow bBb$

To eliminate $S \rightarrow \epsilon$, we search for occurrences of S in the production rule of new CFG.

Now,
 $S \rightarrow aABS$ gives $S \rightarrow aAB$
 $S \rightarrow aAB$ gives $S \rightarrow aA$
 $S \rightarrow aBS$ gives $S \rightarrow aB$
 $\epsilon \rightarrow a$ gives $S \rightarrow a$.

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

$S \rightarrow aAB | aA | aB | a | AB | B | a | b$
 $A \rightarrow aAa$
 $B \rightarrow bBb$

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 production rules, so we eliminate them. Among the generating production rules, some of the production rules may not be reachable from start symbol. To identify such production rules and symbols, we construct a graph called dependency 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 indirect path from start symbol, are the symbols that are not reachable from start symbol. Hence, they are also eliminated.

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

$S \rightarrow aAS | b$
 $A \rightarrow bA | AB | a$
 $B \rightarrow bA | B | aB$
 $D \rightarrow ab$

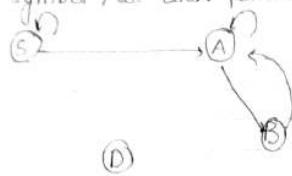
→ soln:

The generating production rules are:-

$S \rightarrow aNS|b$
 $A \rightarrow bA|AB|a$
 $B \rightarrow bA|b$
 $D \rightarrow ab$

The production rule
 $B \rightarrow ab$ 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:
 $S \rightarrow aNS|b$
 $A \rightarrow bB|AB|a$
 $B \rightarrow bA|b$

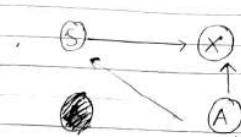
$S \rightarrow aB|bX$
 $A \rightarrow Ba|b|bsX|a$
 $B \rightarrow aSB|bBX$
 $X \rightarrow SB|aBx|ad$.

→ The generating production rules are.

$S \rightarrow aB|bX$
 $A \rightarrow ba|b|a$
 $B \rightarrow$
 $X \rightarrow ad$

The production rule
 $S \rightarrow aB$, $A \rightarrow Ba$, $B \rightarrow aSB|bBX$,
 $X \rightarrow SB|aBx|ad$ 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:
 $S \rightarrow bX$
 $X \rightarrow ad$

Closure properties of context free g. 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 Q . Let $G = (V, E, R, S)$ be a CFG. Now, we have to define operational characteristics of G in such a way that G can generate all the strings of Q_1 or all the strings of Q_2 .

Now, its operational characteristics are defined as:

$$V = V_1 \cup V_2 \cup U$$

$$\Sigma = \Sigma_1 \cup \Sigma_2$$

R consists of -

$$R_1 \cup R_2 \cup S \rightarrow S_1 \cup S_2$$

S = start symbol

Here, S is the start symbol of Q . The production rules $S \rightarrow S_1$, $S \rightarrow S_2$ tell that S can be replaced by either the start symbol of Q_1 or the start symbol of Q_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 Q_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 Q_2 ; so G can generate either the strings of Q_1 or the string of Q_2 .

Hence, we say that G is a CFG that generates all the strings of Q_1 or Q_2 i.e. it can process $L_1 \cup L_2$. Thus, $L_1 \cup L_2$ is also a context free language.

⑪ Concatenation

Let $, Q_1 = (V_1, \Sigma_1, R_1, S_1)$ and $Q_2 = (V_2, \Sigma_2, R_2, S_2)$ be two context free languages such that $L_1 \in L(Q_1)$ and $L_2 \in L(Q_2)$. 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 all the strings of Q_1 followed by the strings of Q_2 . Let $G = (V, E, R, S)$ be a CFG.

Now, we have to define operational characteristics of G in such a way that G can generate the strings of Q_1 followed by the strings of Q_2 .

Now, its operational characteristics are defined as:

$$V = V_1 \cup V_2 \cup U$$

$$\Sigma = \Sigma_1 \cup \Sigma_2$$

R consists of -

$$R_1 \cup R_2 \cup U \rightarrow S_1 S_2$$

S = start symbol

Here, S is the start symbol of Q . 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 Q_1 followed by strings of Q_2 . First it uses production rules R_1 and generates all strings of Q_1 and immediately it uses R_2 and generates all the strings of Q_2 i.e. G can process $L_1 L_2$.

(ii) closure

Let $Q_1 = (V_1, \Sigma_1, R_1, S_1)$ and $Q_2 = (V_2, \Sigma_2, R_2, S_2)$ be two context free grammar and $L_1 \in \text{CFL}$, 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^* so it can generate the string of Q_2 .

Let $Q = (V, \Sigma, R, S)$ be the CFG. Now, we have to define the operational characteristics of Q in such a way that Q can generate the string of Q_2 . It's operation characteristics are:

$V = V_1 \cup V_2$

$\Sigma = \Sigma_1$

R consists of

$R_1 \cup S_1 \rightarrow \underline{\underline{SS_1}}$

$S = \text{start symbol}$

Here, S is the start symbol of Q . 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_1 and generates all strings of Q_2 number of times.

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

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

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

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

where,

$V = \{S\}$

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

R consists of -

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

S - start symbol

$$w = 101101$$

$$S \rightarrow 1S1$$

$$\rightarrow 101S01$$

$$\rightarrow 10110101$$

$$\rightarrow 101101$$

$$w = 010$$

$$S \rightarrow 0S0$$

$$\rightarrow 010$$

②

$\rightarrow Q = \{V, \Sigma, R, S\}$

where,

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

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

R consists of

$$S \rightarrow AB \mid a \mid ab$$

$$A \rightarrow 2A \mid a$$

$$B \rightarrow bB \mid b$$

S - start symbol

$s \rightarrow asb$
 $\rightarrow abb$

$s \rightarrow asb$
 $\rightarrow aasbb$
 $\rightarrow aaabbba$

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

$$\begin{array}{l} aasbb \\ \downarrow \\ s \rightarrow aasbb \end{array}$$

$$w.wR = a^n b^n | a^n$$

$$\begin{array}{l} s \rightarrow AB \\ A \rightarrow aAa \\ B \rightarrow bBb \end{array}$$

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

$$s \rightarrow \cancel{a} \cancel{b} \rightarrow asbb | abb$$

$$\Rightarrow s \in L$$

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

where,

$$V = \{S\}$$

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

R consists of -

$$s \rightarrow asbb | abb$$

Example: $\cdot a a a b b b b b b b$

$$s \rightarrow asbb$$

$$\rightarrow aasbbb [s \rightarrow asbb]$$

$$\rightarrow a a a b b b b b [s \rightarrow abb]$$

Pushdown automata:

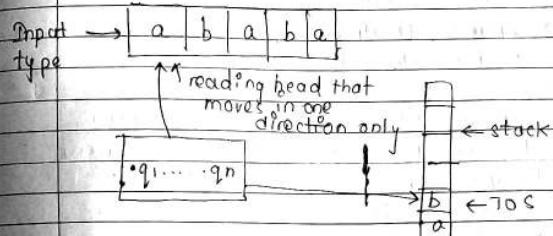


Fig: Block diagram of a PDA

Mathematical a PDA is defined as:

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

where,

Q = finite set of states

Sigma = finite set of inputs

delta = transition function.

q₀ = initial state

F = set of final state

Gamma = stack symbols.

Move of a PDA

- The instant given by a function transition function of a PDA is called its move.

- A move of a PDA is defined as:-

$$\delta(q, x, Y) \rightarrow (p, z) \Rightarrow (q \times x \times T \rightarrow p \times z)$$

1.2	.3849	.3869	.3665	.3686	.3708	.3729	.3749	.3769	.3770	.3799	.3819
1.3	.4032	.4049	.4066	.4082	.4099	.4115	.4131	.4147	.4162	.4177	.4191
1.4	.4192	.4207	.4222	.4236	.4251	.4265	.4279	.4292	.4306	.4319	.4331
1.5	.4332	.4345	.4357	.4370	.4382	.4394	.4406	.4418	.4429	.4441	.4453
1.6	.4452	.4463	.4474	.4484	.4495	.4505	.4515	.4525	.4535	.4545	.4555
1.7	.4554	.4564	.4573	.4582	.4591	.4599	.4608	.4616	.4624	.4632	.4640
1.8	.4641	.4649	.4656	.4664	.4671	.4678	.4685	.4692	.4700	.4707	.4715
1.9	.4713	.4719	.4724	.4730	.4736	.4742	.4748	.4754	.4760	.4766	.4772

where,

q = current state

x = symbol of input tape

y = current top of stack

p = next state

α = symbol that replaces the current top of the stack and becomes new top of stack.

e.g. push ' a ' to the empty stack.

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

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

or Pop ' a ' from the stack

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

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

$$w = aaaa bbbb$$

$$\textcircled{1} \quad \delta(q, a, \epsilon) \rightarrow (f, a)$$

$$\textcircled{2} \quad \delta(f, a, a) \rightarrow (f, a, a)$$

$$\textcircled{3} \quad \delta(f, b, a) \rightarrow (f, \epsilon)$$

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

where,

$$Q = \{q, f\}$$

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

δ consists of:

$$\textcircled{1} \quad \delta(q, a, \epsilon) \rightarrow (f, a)$$

$$\textcircled{2} \quad \delta(f, a, a) \rightarrow (f, a, a)$$

$$\textcircled{3} \quad \delta(f, b, a) \rightarrow (f, \epsilon)$$

$$q_0 = q$$

$$F = f$$

$$\Gamma = \{a\}$$

$$w = aaaa bbbb$$

Stride	Unread input	Stack Content	Transition label
q	aaaa	empty	-
f	aaaa	a	\textcircled{1}
f	aaa	aa	\textcircled{2}
f	aa	aaa	\textcircled{2}
f	a	aa	\textcircled{3}
f	b	a	\textcircled{3}
f	bb	ab	\textcircled{3}
f	bbb	abb	\textcircled{2}
f	bbb	abbb	\textcircled{2}
f	bbb	abbb	\textcircled{1}

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

1.2	.3849	.3869	.3686	.3708	.3729	.3749	.3770	.3790	.3799	.3810	.3821
1.3	.4032	.4049	.3888	.3907	.3925	.3944	.3962	.3980	.3997	.4015	.4020
1.4	.4192	.4207	.4066	.4082	.4099	.4115	.4131	.4147	.4162	.4177	.4182
1.5	.4332	.4345	.4357	.4370	.4382	.4394	.4406	.4418	.4429	.4441	.4453
1.6	.4452	.4463	.4474	.4484	.4495	.4505	.4515	.4525	.4535	.4545	.4555
1.7	.4554	.4564	.4573	.4582	.4591	.4599	.4608	.4616	.4625	.4635	.4645
1.8	.4641	.4649	.4654	.4662	.4671	.4678	.4686	.4694	.4702	.4710	.4718
1.9	.4711										

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 $L = \{ \text{set of all the strings over } \Sigma = \{a, b\} \text{ that contains equal number of } a's \text{ and } b's \}$

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

Where,

$$Q = \{q, f\}$$

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

S consists of

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

$$q_0 = q$$

$$F = f$$

$$\delta = \{q\}$$

state	unreading input	stack content	transition used
q	ababb	empty	-
f	babbb	a	①
q	pabb	ε	② ⑤
f	abb	a	①
f	bb	aa	③
f	b	a	⑤
f	ε	ε	⑤

Q. Design a PDA for the language $L = \{ a^n b^n : n \geq 0 \}$
 \Rightarrow Let $M = (Q, \Sigma, S, q_0, F, \delta)$ be the PDA.
 Where,

$$Q = \{q, f\}$$

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

S consists of

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

$$q_0 = q, f = f, T = 2q$$

state	unreading input	stack content	transition
q	abbbb	empty	-
f	abbbb	a	①
f	bbb	aa	④
f	bb	a	③
f	ε	ε	⑤

	.3140	.3212	.3267	.2995	.3023	.3764	.2794	.3211	.3249
1.0	.3413	.3438	.3461	.3485	.3508	.3531	.3554	.3577	.3592
1.1	.3643	.3665	.3688	.3708	.3729	.3749	.3770	.3790	.3810
1.2	.3849	.3869	.3888	.3907	.3925	.3944	.3962	.3980	.3997
1.3	.4032	.4049	.4066	.4082	.4099	.4115	.4131	.4147	.4162
1.4	.4192	.4207	.4222	.4236	.4251	.4265	.4279	.4292	.4306
1.5	.4322	.4345	.4357	.4370	.4382	.4394	.4406	.4418	.4429
1.6	.4452	.4463	.4474	.4484	.4495	.4505	.4515	.4525	.4441
1.7	.4554	.4564	.4573	.4582	.4591	.4599	.4608	.4616	.4545
1.8	.4641	.4649	.4656	.4664	.4671	.4678	.4686	.4694	.4699
1.9	.4713	.4710							

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

State	Unread Input	Stack Content	Transition w/s
q	aabbba	empty	
f	abbbb	aa	(1)
f	bbb	aaa	(2)
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 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:
 $S \rightarrow aAs | as | b$

w = aabbdb
 $S \rightarrow aAs$
 $A \rightarrow AB | b$
 $B \rightarrow BA | a$
 $\rightarrow aAbA | s$
 $\rightarrow aabb | as$
 $\rightarrow aabbab$

- (1) $S(q, \epsilon, \epsilon) \rightarrow (f, s)$
(2) $S(f, \epsilon, \epsilon) \rightarrow (f, (aAs), (f, as), (f, b))$
(3) $S(f, \epsilon, A) \rightarrow ((f, aB), (f, b))$
(4) $S(f, \epsilon, B) \rightarrow ((f, BA), (f, a))$
(5) $S(f, a, a) \rightarrow (f, \epsilon)$
(6) $S(f, b, b) \rightarrow (f, \epsilon)$

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

where,

$$Q = \{q_0, f\}$$

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

δ consists of -

$$q_0 = q$$

$$F = f$$

$$\Gamma = \{S, A, B, a, b\}$$

1.1	.3643	.3665	.3686	.3708	.3729	.3749	.3770	.3790	.3810	.3830	.3850	.3865	.3881	.3897	.3915	.3931	.3944	.3962	.3980	.3997	.4015
1.2	.3849	.3869	.3888	.3907	.3925	.3944	.3962	.3980	.3997	.4015	.4032	.4049	.4066	.4082	.4099	.4115	.4131	.4147	.4162	.4177	.4191
1.3	.4032	.4049	.4066	.4082	.4099	.4115	.4131	.4147	.4162	.4177	.4192	.4207	.4222	.4236	.4251	.4265	.4279	.4292	.4306	.4319	.4332
1.4	.4207	.4222	.4236	.4251	.4265	.4279	.4292	.4306	.4319	.4332	.4345	.4357	.4370	.4382	.4394	.4406	.4418	.4429	.4441	.4454	.4461
1.5	.4332	.4345	.4357	.4370	.4382	.4394	.4406	.4418	.4429	.4441	.4454	.4461	.4469	.4466	.4471	.44784473
1.6	.4452	.4463	.4474	.4484	.4495	.4505	.4515	.4525	.4535	.4545	.4554	.4564	.4573	.4582	.4591	.4599	.4608	.4614	.4625	.4635	.4641
1.7	.4554	.4564	.4573	.4582	.4591	.4599	.4608	.4614	.4625	.4635	.4641	.4713	.4719	.4776	.4776
1.8	.4641	.4649	.4656	.4664	.4671	.4678	
1.9	.4713	.4719	.4776	

state	unreadInput	stackContent	transition used
q	aabbab	empty	-
f	aabbab	s	①
f	aabbab	as	②
f	abbab	as	⑤
f	abbab	abs	③
f	bab	bs	⑤
f	bab	bas	④
f	bab	bas	⑤
f	ab	abs	④
f	ab	abs	①
f	b	s	⑤
f	b	b	①
f	E	E	⑥

δ consists of -

- ① $\delta(q, \epsilon, \epsilon) \rightarrow (f, s)$
- ② $\delta(f, \epsilon, \epsilon) \rightarrow ((f, 0BB), (f, 1s))$
- ③ $\delta(f, \epsilon, A) \rightarrow ((f, 1B), (f, 0))$
- ④ $\delta(f, \epsilon, B) \rightarrow ((f, 0A), (f, 1))$
- ⑤ $\delta(f, 1a, 1a) \rightarrow (f, t)$
- ⑥ $\delta(f, 0, 0) \rightarrow (f, t)$

state	unreadInput	stackContent	transition used
q	100111	empty	-
f	100111	s	①
f	100111	1s	②
f	00111	s	⑤
f	00111	0BB	②
f	0111	BB	⑥
f	0111	0BBB	②
f	111	BBB	⑥
f	111	1BB	④
f	11	BB	⑥
f	11	1B	⑦
f	1	B	⑤
f	1	1	④
f	E	E	⑤

Construct the equivalent PDA of the following CFG

$S \rightarrow 0BB|1s$

$B \rightarrow 0A|1$

$A \rightarrow 1B|0$

Check whether the string $w = 100111 \in L(C_4)$

$w = 1$

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

$$Q = \{q, f\}$$

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

$$S \rightarrow \cancel{0}BB|1s$$

$$\rightarrow 10BB$$

$$\rightarrow 100BBB$$

$$\rightarrow 100111.$$

	.3665	.3686	.3708	.3729	.3749	.3770	.3790	.3799	.3821
1.2	.3849	.3868	.3889	.3907	.3925	.3944	.3962	.3980	.3997
1.3	.4032	.4049	.4066	.4082	.4099	.4115	.4131	.4147	.4162
1.4	.4192	.4207	.4222	.4236	.4251	.4265	.4279	.4292	.4306
1.5	.4332	.4345	.4357	.4370	.4382	.4394	.4406	.4418	.4429
1.6	.4452	.4463	.4474	.4484	.4495	.4505	.4515	.4523	.4533
1.7	.4554	.4564	.4573	.4582	.4591	.4599	.4608		
1.8	.4641	.4649	.4656	.4664	.4671	.4678			
1.9	.4713	.4719	.4726						

4 Pumping lemma for CFL

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

- ① $1 \vee x_1 > 1$
 - ② $1 \vee w_1 x_1 \leq n$
 - ③ $w^k \vee x^k \vee y \in L$ for all $k > 0$.

Prnat:

Let us consider a CFG -

$S \rightarrow A\beta$

$$A \rightarrow aB/a$$

$$B \rightarrow b A) b$$

Let ω be a CFL and $g \in \omega$.

Now, $g = ababb \in L$

then, according to pumping lemma we can decompose w into substring uv^iwk and y as -

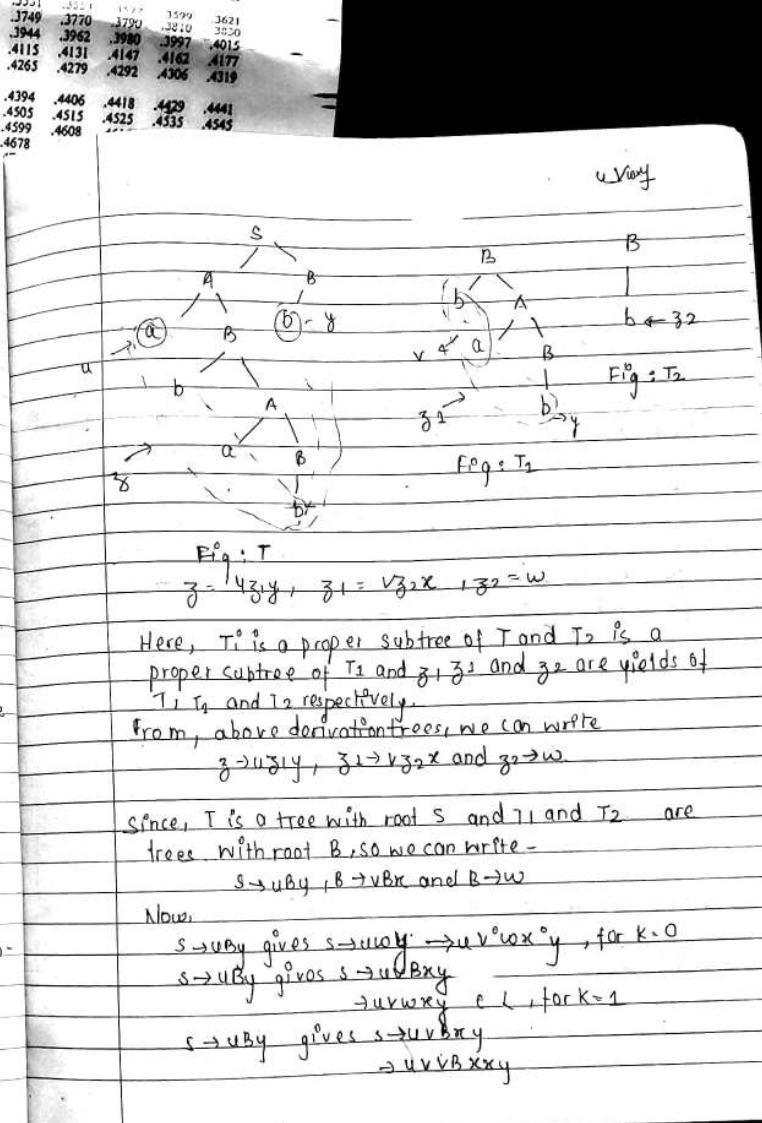
$$y = a$$

v = ba

$\omega = b$

$$x = t$$

Let us construct a derivation tree for $3 = ababbab$.



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	.4617	.4626	.4635
1.8	.4641	.4649	.4656	.46644678
1.9	.4713	.4710

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

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

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

Let L be context free language i.e. $L = \{a^n b^n c^n | n > 0\}$ and $w = a^p b^q c^r \in L$

According to the pumping lemma, w can be decomposed into three five substrings, such that u, v, w, x and y such that:

$$u = a^s$$

$$v = a^t, t > 0$$

$$w = a^b$$

$$x = \epsilon$$

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

$$\begin{aligned} uv^2wz^2y &= a^q a^{2r} a^s a^{p-(q+r+s)} b^p c^p \\ &= a^{p+r} b^p c^p \end{aligned}$$

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

i.e. $a^{p+r} b^p c^p$ is not of the form

$$a^p b^p c^p$$

$uv^2wz^2y \notin L$

Turning machine

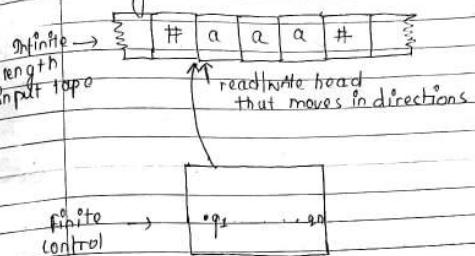


Fig : Block diagram of TM

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 storing the symbols permanently to the input tape.

1.3	.3869	.3888	.3907	.3925	.3944	.3962	.3980	.3997	.4015	.4032	.4049	.4066	.4082	.4099	.4115	.4131	.4147	.4162	.4177	
1.4	.4192	.4207	.4222	.4236	.4251	.4265	.4279	.4292	.4306	.4319	.4332	.4345	.4357	.4370	.4382	.4394	.4406	.4418	.4429	.4441
1.5	.4452	.4463	.4474	.4484	.4495	.4505	.4513	.4525	.4535	.4545	.4554	.4564	.4573	.4582	.4591	.4599	.4608	.4617	.4626	.4635
1.6	.4641	.4649	.4657	.4665	.4673	.4681	.4688	.4695	.4702	.4709	.4717	.4725	.4733	.4741	.4749	.4757	.4765	.4773	.4781	.4789
1.7	.4797	.4805	.4813	.4821	.4829	.4837	.4845	.4853	.4861	.4869	.4877	.4885	.4893	.4901	.4909	.4917	.4925	.4933	.4941	.4949
1.8	.4957	.4965	.4973	.4981	.4989	.4997	.5005	.5013	.5021	.5029	.5037	.5045	.5053	.5061	.5069	.5077	.5085	.5093	.5101	.5109
1.9	.5117	.5125	.5133	.5141	.5149	.5157	.5165	.5173	.5181	.5189	.5197	.5205	.5213	.5221	.5229	.5237	.5245	.5253	.5261	.5269

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.

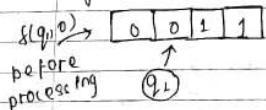
x = symbol of input tape or symbol under read/write head.

p = next state.

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

d = direction of the next symbol to be read (L/R or N).

e.g.: $\delta(q_0, 0) \rightarrow (q_0, 1, R)$



• Instantaneous description (ID)

$0q_011$

$(q_0, 1, R) \rightarrow [0 \ 1 \ 1 \ 1]$
 after processing $q_0 \uparrow$
 $01q_011$

$0q_011 \rightarrow 01q_011$

$\delta(q, x^i) \rightarrow (p, y, L)$

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

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

before processing $q \uparrow$
 $x_1 x_2 \dots x_{i-1} q x_i x_{i+1} \dots x_n$

after processing $x_1 x_2 \dots p x_{i-1} y x_{i+1} \dots x_n$
 $x_1 x_2 \dots x_{i-1} q x_i x_{i+1} \dots x_n \rightarrow x_1 x_2 \dots p x_{i-1} y x_{i+1} \dots x_n$

Design of a TM:

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

- Halt is a special state that is used to determine

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	.4617	.4626	.4635
1.8	.4641	.4649	.4656	.4663	.4670	.4678	.4686	.4694	.4702	.4710
1.9	.4711									

the acceptance of a string.

- The difference between final state and halt state is that TM stops processing after transitioning to halt state.

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

w = abbaab

- 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:

$q_1 a b a a b \rightarrow \# q_2 b a a b$
 $\rightarrow \# \# q_2 a a b$
 $\rightarrow \# \# \# q_2 a b$
 $\rightarrow \# \# \# \# q_2 \#$
 $\rightarrow \# \# \# \# h$

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

~~for example~~

- 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:

1.6	.4332	.4345	.4357	.4370	.4382	.4394	.4406	.4418	.4429	.4441	.4453	.4464
1.7	.4452	.4463	.4474	.4484	.4495	.4505	.4515	.4525	.4535	.4545		
1.8	.4554	.4564	.4573	.4582	.4591	.4599	.4608					
1.9	.4641	.4649	.4656	.4664	.4671	.4678						

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 -

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

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

$$\begin{aligned} \#q_1 I \# &\rightarrow \# I q_2 \# \\ &\rightarrow \# I I q_2 \# \\ &\rightarrow \# II h \end{aligned}$$

$$① \delta(q_1, I) \rightarrow (q_1, I, R)$$

$$② \delta(q_1, \#) \rightarrow (q_2, I, R)$$

$$③ \delta(q_2, \#) \rightarrow (h, \#, N)$$

When $n=2$,

$$\begin{aligned} \# q_1 T I \# &\rightarrow \# I q_1 \# \\ &\rightarrow \# I I q_2 \# \\ &\rightarrow \# II h \end{aligned}$$

When $n=3$,

$$\begin{aligned} \# q_1 T T I \# &\rightarrow \# I q_1 I \# \\ &\rightarrow \# I I q_2 \# \\ &\rightarrow \# II I q_3 \# \\ &\rightarrow \# III h \end{aligned}$$

Similarly,

$$\# q_1 T^n \# \rightarrow \dots \rightarrow \# I^{n+1} h$$

$$\# f(n) = n+2, n > 0$$

When $n=1$,

- ① $\delta(q_1, I) \rightarrow (q_1, I, R)$
- ② $\delta(q_1, \#) \rightarrow (q_2, I, R)$
- ③ $\delta(q_2, \#) \rightarrow (q_3, I, R)$
- ④ $\delta(q_3, \#) \rightarrow (h, \#, N)$

When $n=2$,

$$\begin{aligned} \# q_1 T I \# &\rightarrow \# I q_1 \# \\ &\rightarrow \# I I q_2 \# \\ &\rightarrow \# II I q_3 \# \\ &\rightarrow \# III h \end{aligned}$$

When $n=3$,

$$\begin{aligned} \# q_1 T T I \# &\rightarrow \# I q_1 I \# \\ &\rightarrow \# I I q_2 \# \\ &\rightarrow \# II I q_3 \# \\ &\rightarrow \# III h \end{aligned}$$

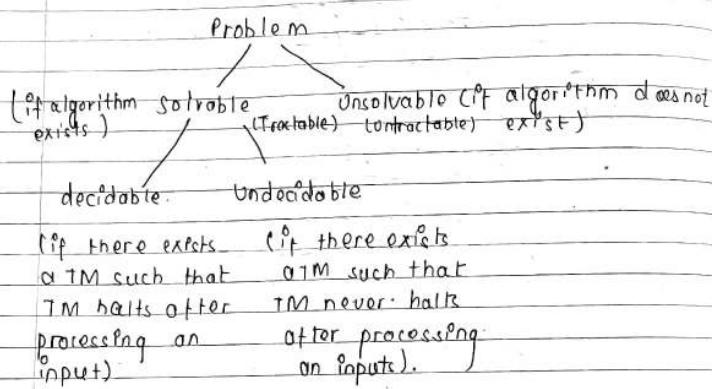
Similarly,

1.7	.4463	.4474	.4484	.4495	.4505	.4515	.4525	.4535	.4545
1.8	.4554	.4564	.4573	.4582	.4591	.4599	.4608	.4617	.4626
1.9	.4641	.4649	.4657	.4665	.4673	.4681	.4689	.4697	.4705

Language of TM

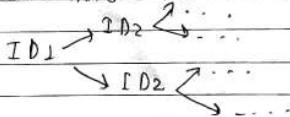
1. Recursive language (decidable problems)
2. Recursively enumerable language (decidable + Undecidable problems).

Decidability and Undecidability



Types of TM:-

- ① Deterministic TM $\rightarrow T D_1 \rightarrow T D_2 \rightarrow \dots \rightarrow T D_n$
- 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 the 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.