TUTORIAL SESSION 01:

NFA, DFA

Date of the Session:___/___/____ **Time of the Session _____ to_____**

Pre-Tutorial:

1. Define DFA and NFA formally.

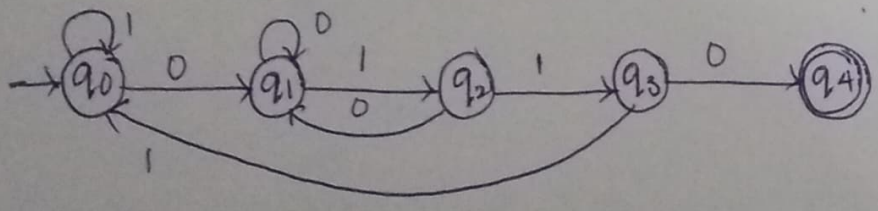
Solution:

DFA	NFA
1) Means deterministic Finite Automata	1) Means Non-deterministic Finite automata
2) It has 5 tuples $(Q, \Sigma, q_0, \delta, F)$	2) It also has 5-tuples $(Q, \Sigma, \delta, q_0, F)$

2. Construct a DFA that accepts the language

$$L = \{w \in \{0,1\}^* \mid w \text{ contains } 1001 \text{ or } 0110\}$$

Solution:



3. Write the Steps for converting $\epsilon - NFA$ to DFA and vice-versa withan example from each.

Solution:

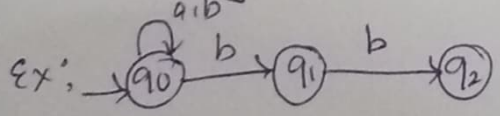
step 1:- let ϕ be a new set of states of DFA, D is null in the starting
→ let T be a new transition table of DFA

step 2:- Add start state of NFA to 1.
→ Add transition of the start state to transition table T.
→ If start state makes transition to multiple slists the same input alphabel. Then treat those multiple states

step3: If any new states is present in transition table T
→ Add new states in 'Q'
→ Add transition of that state in transition table

step4:- keep repeting

Ex:



Table

Q/Σ	a	b
→ q_0	q_0	q_0q_1
q_1	-	q_2
q_2	-	-

Step 1:- let take Q be a new set of DFA
let T be a new transition table of the DFA

step 2:- Add transition of start state q_0 the table

Q\ε	0	1
q_0	$[q_0 q_1]$	q_1

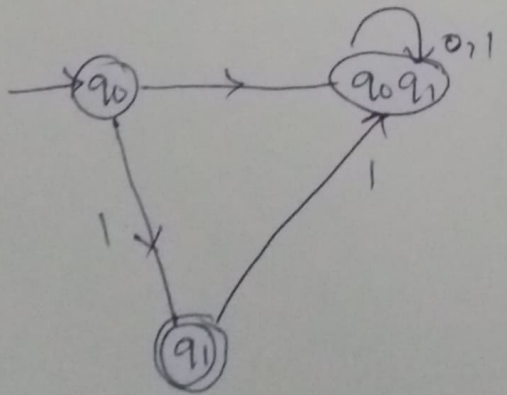
step 3: Add transition of state q_1 to the transition table new state is present instant a is (q_0, q_1)

Q\ε	0	1
q_0	$[q_0, q_1]$	q_1
q_1	ϕ	$[q_0, q_1]$

step 4:- since so new states are left to be added in the table T, so, we stop

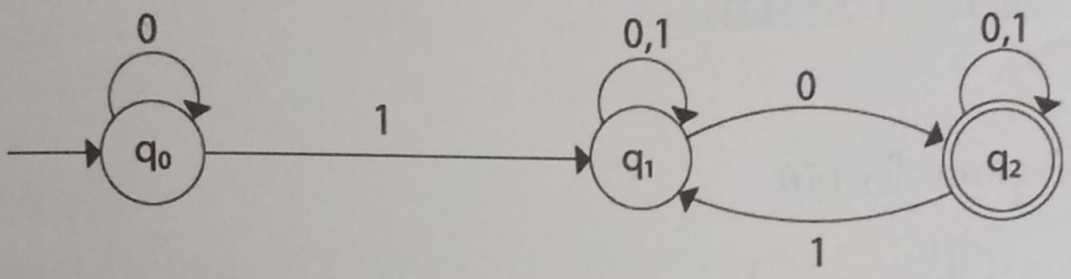
Finally, Transition table to DFA

Q\ε	0	1
q_0	$[q_0 q_1]$	q_1
q_1	ϕ	$[q_0, q_1]$
$q_0 q_1$	$[q_0 q_1]$	$[q_0 q_1]$



In-Tutorial:

1. Convert the following NFA to DFA.



Solution: Machine

$M = \{Q, \Sigma, q_0, \delta, F\}$

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

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

$\delta = Q \times \Sigma \to 2^Q$

$q_0 = q_0$

$F = q_2$

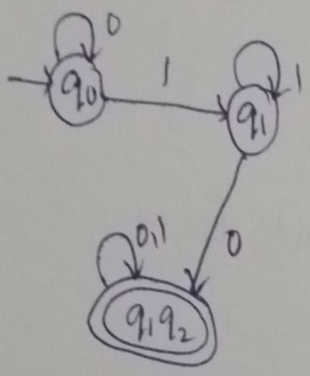
Transition table for NF

	0	1
→ q_0	q_0	q_1
q_1	q_1, q_2	q_1
q_2	q_2	q_2, q_1

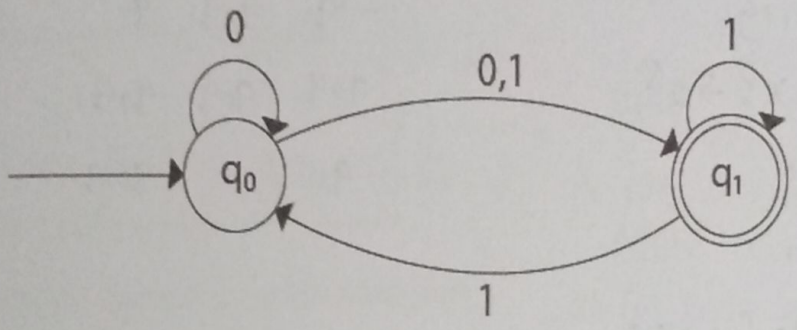
Transition table for DFA

	0	1
q_0	q_0	q_1
q_1	q_1q_2	q_1
q_1q_2	q_1q_2	q_1q_2

Diagram for DFA



2. Write the algorithm that converts NFA to DFA. Explain how your algorithm works using the below NFA.



Solution:

Algorithm for Conversion of NFA to DFA

I/P → NFA
O/P → DFA

Step-1 : Create the transition table for the given NFA

Step-2 : Create a blank Transition table under possible I/P alphabet for Equivalent DFA.

Step-3 : Mark start state of DFA same as NFA

Step-4 : Find out the Constraints of states $[q_0, q_1, q_2 \ldots q_n]$ on each possible I/P alphabets

Step-5 :- Each time we generate a new DFA state under the I/P alphabet columns. We have to apply step 4 again other wise go to step 6.

Step-6 :- The states which contain any of the final states of NFA or the final states of Equivalent DFA

Machine :-

$M = \{Q, \delta, \Sigma, q_0, F\}$

$Q = \{q_0, q_1\}$

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

$\delta = Q \times \Sigma \to 2^Q$

$q_0 = q_0$

$q_f = q_1$

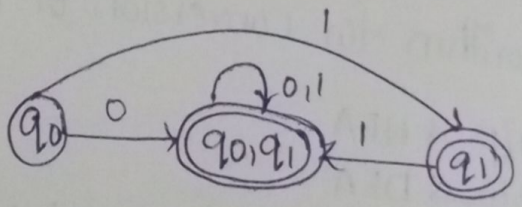
Transaction table for NFA

	0	1
$\to q_0$	$q_0 q_1$	q_1
q_1	—	$q_0 q_1$

Transition table for DFA

	0	1
$\to q_0$	$q_0 q_1$	q_1
$q_0 q_1$	$q_0 q_1$	$q_0 q_1$
q_1	-	$q_0 q_1$

Diagram for DFA



Post-Tutorial:

1. Differentiate NFA and DFA.

Solution:

NFA	DFA
1) It stands for non-deterministic finite automata	1) DFA stands for Deterministic finite automata
2) for each symbolic representation of alphabet there is no need to specify how it reacts	2) for Each symbolic representation of alphabet there is only one state transition in DFA
3) Every NFA is not DFA	3) Every DFA is NFA

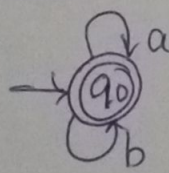
2. Construct an NFA for the language $L = \{w \in \{0,1\}^* \mid a^* + b^*\}$

Solution:

Given,

$L = \{w \in \{0,1\}^* \mid a^* + b^*\}$

diagram:-



$\Rightarrow a^* + b^*$

Machine

$M = \{Q, \Sigma, \delta, q_0, F\}$

$Q = \{q_0\}$

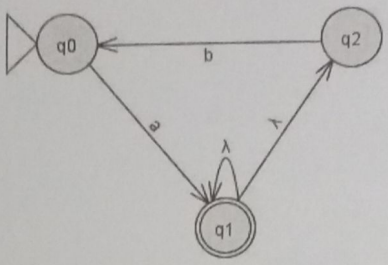
$\Sigma = \{a, b\}$

$\delta = Q \times \Sigma \to 2^Q$

$q_0 = q_0$

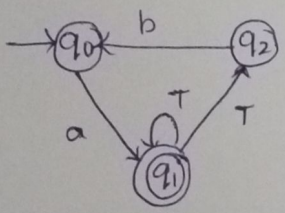
$F = q_0$

3. Convert the following $\epsilon - NFA$ to DFA.



Solution:

Given,



Transition Table

	a	b	λ
→q_0	q_1	-	-
q_2	-	q_0	-
q_1	-	-	q_1, q_2

Machine:-

$M = \{Q, \Sigma, q_0, \delta, F\}$

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

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

$\delta = Q \times \Sigma \to 2^Q$

$q_0 = q_0$

$F = q_1$

ϵ- closure finding:-

ϵ- closure $(q_0) = \{q_0, q_1, q_2\}$

ϵ- closure $(q_1) = \{q_0, q_1, q_2\}$

ϵ - closure $(q_2) = \{q_0, q_1, q_2\}$

TUTORIAL SESSION 02:

REGULAR EXPRESION

Date of the Session:___/___/____ **Time of the Session _____ to_____**

Pre-Tutorial:

1. Explain regular expression and name some of the identity rules for the regular expression? Assume a, b and c are regular expressions inthe identity rules.

Solution:

Regular Expression:-

The language accepted by finite automata are easily described by simple expressions called Regular Expression.

→ ϵ is also a Regular Expression

Simplication Rules:- If P and Q are regular Expressions

1) $Q+R=R$
2) $Q=RQ=Q$
3) $\epsilon R=R\epsilon=R$
4) $\epsilon^*=\epsilon \& Q^*=\epsilon$
5) $R+R=R$
6) $R^*+R^*=R$
7) $RR^*=R^*R=R^*$
8) $(R^*)^*=R^+$
9) $\epsilon+RR^*=R^*$
10) $(PQ)^*P=P(QP)^*$
11) $(P+Q)^*=(P^*Q^*)^*=(P^*+Q^*)^*$
12) $P(Q+R)=PQ+QR$
(or)
$(P+Q)R=PR+QR$

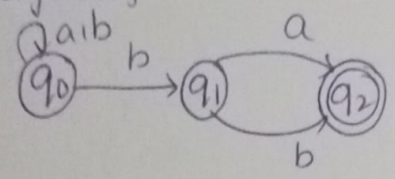
2. Consider the language L given by the regular expression

$$(a + b)^* b (a + b)$$

over the alphabet {a, b}. Design a DFA that accepts L.

Solution:

Diagram:-



Transition Table for NFA

	a	b
$\rightarrow q_0$	q_0	$\{q_0, q_1\}$
q_1	q_2	q_2
q_2	$\emptyset$	$\emptyset$

Machine:-

$M = \{Q, \Sigma, q_0, \delta, q_f\}$

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

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

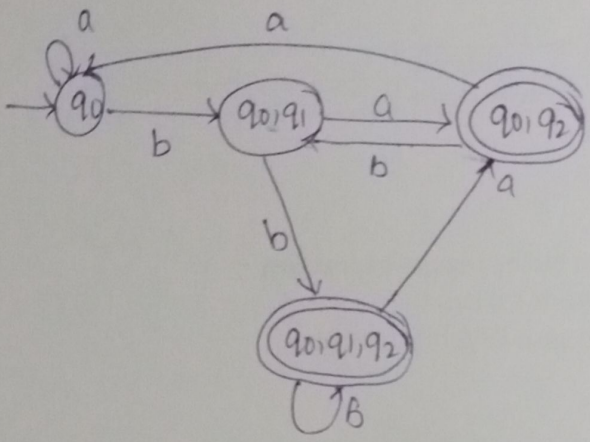
$\delta = Q \times \Sigma \rightarrow 2^Q$

$q_0 = q_0$

$F = q_2$

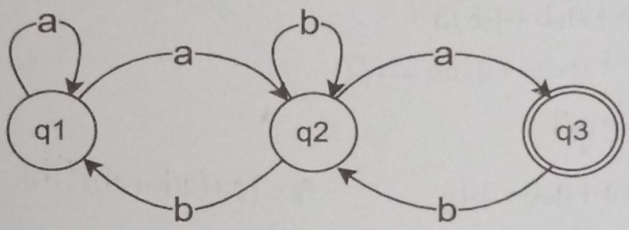
Transition Table for DFA :-

	a	b
$\rightarrow q_0$	q_0	$\{q_0, q_1\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$
$\{q_0, q_2\}$	q_0	$\{q_0, q_1\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$


a
a
q_0
b
q_0, q_1
a
b
q_0, q_2
a
b
q_0, q_1, q_2
B

In-Tutorial:

1. Provide the algorithm to convert NFA into regular expression. Create a regular expression for the following NFA.



Solution:

Algorithm:-

1) let, there be NFA with N states, our task is to convert NFA to RE
2) Writing the equactions
   1) For each state q_i in NFA, write an equation in the form

$$q_i = q_1a_{1i} + q_2a_{2i} + \cdots + q_na_{ni}$$

3) Applying Arden's theorem: $R = Q + RP$, then $R = QP^*$
4) Substituting state variables
5) Represent the final states to get the overall Regular Expression

According to Arden's theorem,

$q_1 = \epsilon + q_1a + q_2b \rightarrow (1)$

$q_2 = q_1a + q_2b + q_3b \rightarrow (2)$

$q_3 = q_2a \rightarrow (3)$

from (3), we get

$q_3 = q_2a$

$= (q_1a + q_2b + q_3b)a$

$= q_1aa + q_2ba + q_3ba \rightarrow (4)$

From (3), we get

$q_2 = q_1a + q_2b + q_3b$

$= q_1a + q_2b + (q_2a)b$

$q_2 = q_1a + q_2(b+ab)$

It is form of

$R = Q + RP$

$R = QP^*$

$q_2 = q_1a(b+ab)^* \rightarrow (5)$

$q_1 = \epsilon + q_1a + q_2b$

$q_1 = \epsilon + q_1a + (q_1a(b+ab)^*)b$

$q_1 = \epsilon + q_1(a + (a(b+ab)^*b$

$R = Q + RP$

$q_1 = \epsilon(a + (a(b+ab)^*)b)^*$

$q_1 = (a + (a(b+ab)^*)b)$

Now,

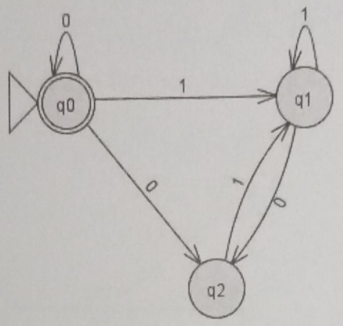
$q_3 = q_2a$

$q_3 = q_1a(b+ab)^*a$

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

$a(b+ab)^*a$

2. Explain Arden's theorem to convert FA to RE. Use the algorithm to convert the following DFA to RE.



Solution:

Steps to convert Finite Automata to RE :-

1) For a Equaction, for each state considering the transitions which comes towards the intial state Add ϵ in the eq^n of intial state.

2) Bring final state into $R = Q + RP$ form to get the Required FA.

Machine :-

$M = \{Q, \Sigma, \delta, q_0, q_f\}$

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

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

$\delta = Q \times \Sigma \to Q$

$q_0 = q_0$

$F = q_0$

Transition Table

	0	1
$\to q_0$	$\{q_0, q_2\}$	q_1
q_1	q_2	q_1
q_2	–	q_1

According to Arden's theorem,

$q_0 = q_{00} + F \to (1)$

$q_1 = q_{01} + q_{11} + q_{21} \to (2)$

$q_2 = q_{00} + q_{10} \to (3)$

Here, we need to get the $R = Q + PR$ eq (1) because it is final state

$R = q_0$ $\quad q_0 = \epsilon + q_0 0$

$Q = \epsilon$ $\quad \Downarrow$

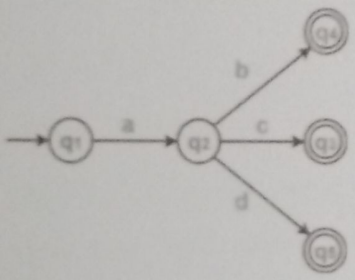
$P = 0$ $\quad$ Here it is in form of $R = Q + RP$, then $R = QP^*$

$R = \epsilon 0^*$

$R = 0^*$

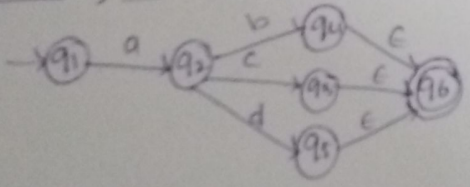
Post-Tutorial:

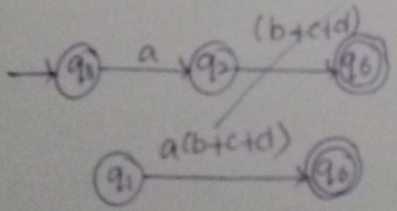
1. Find the regular expression for the following DFA.



Solution:

Converting the given DFA to RE :-





RE = a(b+c+d)

1. For $\Sigma = \{a, b\}$, let us consider the regular language

$$L = \{x \mid x = a^{2+3k} \text{ or } x = b^{10+12k}, k \geq 0\}$$

What could be the minimum pumping length (the constant guaranteed by the pumping lemma) for L?

Solution:

Given,

$L = \{x | x = a^{2+3k}$ or $x = b^{10+12k}, k \geq 0\}$

$L = \{a^2, a^5, a^8, a^{11} \text{----} b^{10}, b^{22}, b^{34} \ldots\}$

By using pumping lemma :-

Let, L be an infinite Regular lang, then there Exists some +ve integer such that any $w \in L$ with $w \geq m$

$w = xyz$

with $|xy| \leq m$ such that $wi = xy^i z$

$i = 0, 1, 2, \text{----}$

There are 3 rules

$xy^i z \in w$

$|y| \geq 0$

$|xy| \leq p$

let assume $p = 10$,

$w = b^{10}$ has length ≥ 10

So, b^{10} must be pumped, but in b^{10}, if we take any non-Empty substring. and we remove that then resulted string doesnot belong to L.

So, 10 can't be pumped.

Now, let us assume $p = 12$

Take $w = a^{14}$

Then the non-empty substring y with in the first i.e; 12 symbols can be taken,

$x = aaa \rightarrow$ if we extend it belongs to L.

similarly, for

a^{2+3k}, $k \geq 4$, x can be choosen

$x = aaa \rightarrow$ extends from 0 (or) above no change in L.

Take $b = 2$

↓

$p = 12$

↓

so, 12 b's to taken when we Repeat the Result belongs to 'L' only

similarly for any b^{10+12k}, $k \geq 1$

x can be 12 b's → same not change in language L

So, Minimum pumping length for any language L is 12 &

any numbers ≥ 12 is a pumping length for L.

TUTORIAL SESSION 03:

CONTEXT FREE GRAMMAR, LEFTMOST ANDRIGHTMOST DEVIATION

Date of the Session:___/___/____ **Time of the Session** _____ **to**_____

Pre-Tutorial:

1. What is context free grammar. Explain with an example?

Solution:

It is a finite set of variables each of which represented a language

1) The language represented by variables are desired recursively in terms of each other and primitive symbol are called terminal.

2) The rule relating the variable are called as the production

3) The language that Require one Memory element is known as content free Grammar

4) without, Memory element is known as the regular language

Ex:- $S \rightarrow ABl\epsilon$

$A \rightarrow aB$

$B \rightarrow Sb$

In-Tutorial:

1. Construct a CFG for a language $L = \{wcw^R \mid w \in (a,b)^*\}$.

Solution:

Given, language

$L = \{wcw^R \mid w \in (a|b)^*\}$

The context free grammar is

$S \rightarrow asa \rightarrow (1)$

$S \rightarrow bsb \rightarrow (2)$

$S \rightarrow c \rightarrow (3)$

Now if we want to derive string abbcbba,

$S \rightarrow asa$

$S \rightarrow absba$ (from (2))

$S \rightarrow abbsbba$ (from (2))

$S \rightarrow abbcbba$

Hence derived ||

2. Derive the string "*aabbabba*" for left most derivation and right most derivation using a CFG.

Solution:

Given string is :

aabbabba

let CFG be :-

S → aB|bA

A → a|as|bAA

B → b|bs|aBB

left Most derivation (LMD) :-

S → aB
→ aaBB
→ aabSB
→ aabbAB
→ aabaaB
→ aabbabs
→ aabbabbA
→ aabbabba

Right most derivation (RMD) :-

S → aB
→ aaBB
→ aaBbs
→ aaBbbA
→ aaBbbA
→ aaBbba

Post-Tutorial:

1. Generate CFG for the language $L = \{0^i 1^j 0^k \mid j > i + k\}$

Solution:

Given,

$L = \{0^i 1^j 0^k \mid j > 1+k\}$

$S \rightarrow xyz$

$X \rightarrow 0x1 \mid 01$

$Y \rightarrow 1y1 \mid 1$

$Z \rightarrow 1z0 \mid 10$

This is the CFG.

TUTORIAL SESSION 04:

PARSE TREE, AMBIGUITY IN CFG

Date of the Session:___/___/____ **Time of the Session** _____ **to**_____

Pre-Tutorial:

1. Differentiate ambiguous and unambiguous grammar.

Solution:

Ambiguous Grammar	Unambiguous Grammar
1) In ambiguous grammar, the leftmost and right most derivation are not same	1) In unambiguous grammar, the leftmost and right and most derivations are same.
2) Amount of non-terminals in ambiguous grammar is less than in an ambiguous grammar	2) Amount of non-terminal unambiguous grammar is more than in ambiguous grammar.
3) Ambiguous grammar generates more than one parse tree	3) Unambiguous grammars generate only one parse tree.
4) It contains ambiguity	4) It doesnot contain any ambiguity
5) length of parse tree in ambiguous grammar is comparatievely short	5) length of parse tree in unambiguous grammar is comparitively large.

In-Tutorial:

1. Consider the following grammar

$$S \rightarrow ASB|c$$

$$A \rightarrow \epsilon|aA$$

$$B \rightarrow \epsilon|bB$$

Derive the string *acb* using leftmost and rightmost derivation. Show the parse trees for your derivation.

Solution:

Given Grammar,

$S \rightarrow ASB|C$

$A \rightarrow \epsilon|aA$

$B \rightarrow \epsilon|bB$

left most derivation

$S \rightarrow aASB$

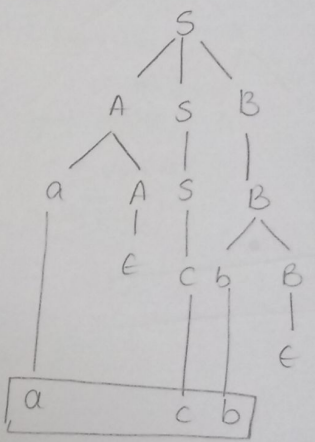
$S \rightarrow a\epsilon SB$

$S \rightarrow aSB$

$S \rightarrow acB$

$S \rightarrow acDB \Rightarrow S \rightarrow acb$

derived



Right most derivation

$S \rightarrow ASB$

$S \rightarrow ASbB$

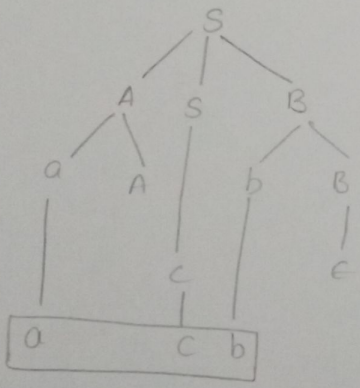
$S \rightarrow ASb\epsilon$

$S \rightarrow Asb$

$S \rightarrow aAsb$

$\boxed{S \rightarrow acb}$

Tree :-



2. Consider the following grammar

$$S \rightarrow aS \mid \epsilon$$

The language generated by this grammar is $L = \{ a^n, n >= 0 \}$ or a^*

i) Find the Leftmost Derivation and Rightmost Derivation

ii) Also, Prove All the strings generated from this grammar have their leftmost derivation and rightmost derivation exactly same.Draw the Parse tree for the same.

Solution:

Given $L = \{ a^n, n>=0 \}$ or a^*

$S \rightarrow as|\epsilon$

i) Left most derivation:-

String "aaa"

$S \rightarrow as$

$\rightarrow aas$

$\rightarrow aaas$

$\rightarrow aaa$

Right most derivation:-

String "aaa"

$S \rightarrow as$

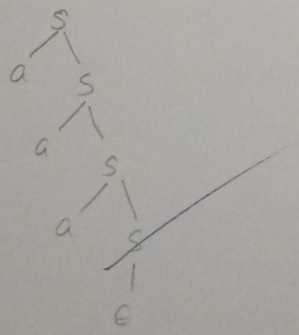
$\rightarrow asa$

$\rightarrow aasa$

$\rightarrow aa\epsilon a$

$\rightarrow \boxed{aaa}$

ii)



in parse tree, each non-terminal expands into its respective production; terminals are leaves of tree

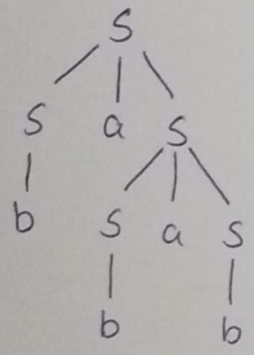
The tree represents both LMP and RMD its same, for both due to the nature of the grammar

Post-Tutorial:

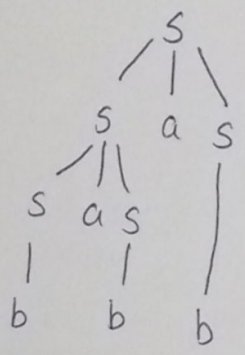
1. Consider the following grammar-
   $S \rightarrow SaS \mid b$
   Is it an ambiguous grammar? Generate the string *babab* from this grammar to prove your point.

Solution: Given, grammar

$S \rightarrow SaS|b$



babab



babab

Here the given grammar has more than one parse tree so, it in an ambiguous Grammar.

TUTORIAL SESSION 05:

SIMPLIFICATION OF CFG, NORMAL FORMS

Date of the Session:___/___/____ **Time of the Session** _____ **to**_____

Pre-Tutorial:

1. Explain simplification of grammar? Mention its use? Elaborate the steps that are followed in simplification process?

Solution: Simplification of Grammar

1) All the grammar are not always optimized that means the grammar may consists of extra symbols

2) Having extra symbols else the length of grammar

3) The properties of Reduced Grammar are,

   1) Each variable and each terminal of a appers in the derivation of same word in L

   2) There should not be any production as $x \to y$ (where x and y are non-terminals

   3) If ϵ is not in the language then there not be production to $x \to \epsilon$

2. Find a reduced grammar equivalent to the grammar G, having production rules:

$$S \rightarrow AC \mid B$$
$$A \rightarrow a$$
$$C \rightarrow c \mid BC$$
$$E \rightarrow aA \mid e$$

Solution:

phase 1:- $T = \{a, c, e\}$

$W_1 = \{A, C, E\}$ from rules $A \rightarrow a$, $C \rightarrow c$, $C \rightarrow aA$

$W_2 = \{A, C, E\} \cup \{S\}$ from rule $S \rightarrow AC$

$W_3 = \{A, C, E, S\} \cup \emptyset$

Since $W_2 = W_3$, we can derive 'G' as

$G' = \{\{A, C, E, S\}, \{a, c, e\}, P, \{S\}\}$

where P: $S \rightarrow AC$, $A \rightarrow a$, $C \rightarrow c$, $E \rightarrow aA \mid e$

phase 2: $Y_1 = \{S\}$

$Y_2 = \{S, A, C\}$ from rule $S \rightarrow AC$

$Y_3 = \{S, A, C, a, c\}$ from rules $A \rightarrow a$ and $C \rightarrow c$

$Y_4 = \{S, A, C, a, c\}$

Since $Y_3 = Y_4$, are can derive G_1'' as—

$G_1'' = \{\{A, C, S\}, \{a, c\}, P\{S\}\}$

where P: $S \rightarrow AC$, $A \rightarrow a$, $C \rightarrow c$

In-Tutorial:

1. Remove unit productions from the following grammar

$$S \rightarrow AC, A \rightarrow a, C \rightarrow X|b, X \rightarrow Y, Y \rightarrow Z, Z \rightarrow a$$

Solution: Given

$S \rightarrow AC$

$A \rightarrow a$

$C \rightarrow X|b$

$X \rightarrow Y$

$Y \rightarrow Z$

$Z \rightarrow a$

After Removing unit productions we get

$S \rightarrow AC$

$A \rightarrow a$

$C \rightarrow a|b$ (∵ $Z \Rightarrow a$, $C \rightarrow X \Rightarrow C \rightarrow a$

$Y \rightarrow Z \Rightarrow Y \rightarrow a$

$X \rightarrow Z \Rightarrow X \rightarrow a$)

$X \rightarrow a$

$Y \rightarrow a$

$Z \rightarrow a$

2. A grammar G is defined with rules $S \rightarrow XA \mid BB, B \rightarrow b \mid SB, X \rightarrow b, A \rightarrow a$. Write the productions obtained after normalized GNF of G.

Solution: Given grammar

$S \rightarrow XA \mid BB$

$B \rightarrow b \mid SB$

$X \rightarrow b$

$A \rightarrow a$

The given grammar is CNG we get Now converting to GNF, we get

$A_1 \rightarrow S$, $A_2 \rightarrow X$, $A_3 \rightarrow A$, $A_4 \rightarrow B$

$A_1 \rightarrow A_2A_3 / A_4A_4$

$A_4 \rightarrow b / A_1A_4$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

Now,

After 2 rules so that non-terminals are in Ascending order such that $A_i \rightarrow A_j$ x $(i<j)$ & never be $(i>=j)$

from a above

$A_1 \rightarrow A_2A_3 / A_4A_4 \rightarrow (1)$

$A_4 \rightarrow b / A_1 A_4 \rightarrow$ ②

$A_2 \rightarrow b \rightarrow$ ③

$A_3 \rightarrow a \rightarrow$ ④

From 1st production

$A_1 \rightarrow A_2 A_3 / A_4 A_4$

the production has i<j

From 2nd production,

$A_4 \rightarrow b / A_1 A_4$

$A_4 \rightarrow b$

$A_4 \rightarrow A_1 A_4$

Now,

$A_4 \rightarrow A_1 A_4$

$A_4 \rightarrow A_2 A_3 / A_4 A_4 A_4$

Sub $A_2 \rightarrow b$)

$A_4 \rightarrow bA_3 A_4 / A_4 A_4 A_4$

sub $A_2 \rightarrow b_1$

Now our 2nd production,

$A_4 \rightarrow b / A_1 A_4$ Converted to

$A_4 \rightarrow b / bA_3 A_4 / A_4 A_4 A_4$

① & ② and rule already in GNF

$A_4 \rightarrow A_4 A_4 A_4$

Now,

Remove left Recursion

for

So, introduce new variable

$Z \rightarrow A_4 A_4 Z / A_4 A_4$

$A_4 \rightarrow A_4 A_4 A_4 A_4$

Post-Tutorial:

1. Convert the following

a) CFG into CNF

$S \rightarrow ASA \mid aB, A \rightarrow B \mid S, B \rightarrow b \mid \varepsilon$

Solution: Given

$S \rightarrow ASA \mid aB$

$A \rightarrow B \mid S$

$B \rightarrow b \mid \epsilon$

step 1:- there, the starting symbols appear on the Right hand side of production New production

$S \rightarrow S$

$S \rightarrow ASA \mid aB$

$A \rightarrow B \mid S$

$B \rightarrow b \mid \epsilon$

step 2:- No useless symbols in production

step 3:- Remove 'e' symbols

$S' \rightarrow S$

$S \rightarrow ASA \mid aB \mid a$

$A \rightarrow B \mid S \mid \epsilon$

$B \rightarrow b$

While Removing $B \to \epsilon$, we get $A \to \epsilon$

$S' \to S$

$S \to ASA|aB|a|SA|SA|S$

$A \to B|S|\epsilon$

$B \to b$

step 4:- Remove unit production

$S' \to S$, $S \to S$, $A \to B$, $A \to S$

$S' \to ASA|aB|a|SA|AS|SA$

$S' \to AX|YBa|a|AS|SA$

$S \to AX|YB|a|AS|SA$

$A \to b|AX|YB|a|AS|SA$

$B \to b$

$X \to SA$

$Y \to a$

2. Write the steps for removing null productions and unreachable symbols? Explain with an example of your own.

Solution:

step 1:- find out all nullable (ϵ) non-terminals variables which derive ϵ

step 2:- for each production $A \to a$ construct all productions, $A \to x$ where x is obtained from removing one (or) more non-terminals from step 1

step 4:- Now combine, the Result of step 2 with original production & Remove ϵ production

ex:- $S \to XYX$

$X \to 0X|\epsilon$

$Y \to 1Y|\epsilon$

1> Removing useless symbols

No useless symbols

Removal of ϵ-productions

$S \to XYX$

$S \to XYX|YX|XX|XY|X|Y$

and productions:

$X \to 0X$

$X \to 0$

$X \to 0X|0$

3rd production :-

$Y \rightarrow 1Y$

$Y \rightarrow 1$

$Y \rightarrow 1Y|1$

∴ The production rules after Removing ϵ are:

$S \rightarrow XYX|YX|XX|XY|X|Y$

$X \rightarrow 0X|0$

$Y \rightarrow 1Y|1$

TUTORIAL SESSION 06:

PDA, N-PDA

Date of the Session:___/___/____ **Time of the Session** _____ **to**_____

Pre-Tutorial:

1. Differentiate between PDA and FA.

Solution:

Pushdown Automata :-

A pushdown automata is finite state machine with an added stack storage. Additional stack is used in making the division for transition apart from input.

Symbols :-

7 tuples

Q : finite set of states
Σ : ste of input symbols
Γ = stack alphabet
q_0 = intial state
F = set of final states

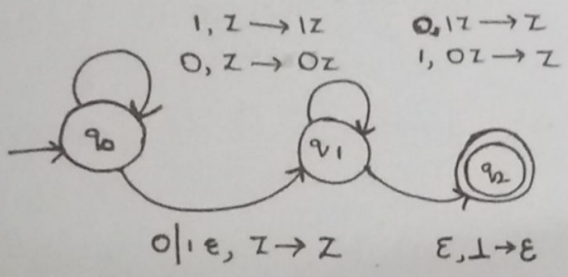
T. Transition function

Finite Automata :-

A finite automata is mathematical model of any machine by which we can calculate transition of states on any input symbol.

It contains the following 5 tuples.

2. Consider the NPDA Q={q0,q1,q2}, Σ = {0,1} Γ={0,1, $\perp$},δ,q0, $\perp$,F={q2}, where (as per usual convention) Q is the set of states, Σ is the input alphabet , Γ is the stack alphabet , δ is the state transition function , q0 is the initial state , $\perp$ is the initial stack symbol, and F is the set of accepting states. The state transition is as follows. What sequence of the string 101100 must follow so that the overall string is accepted by the automaton?



Solution:

At state q0, input is 1, push 1

At state q0, input is 0, push 0

At state q0, input is 1, push 1

At state q0, input is 1, push 1

At state q0, input is 0, push 0

At state q0, input is 1, stack top 0, pop

At state q1, input is 0, stack top 1, pop

At state q1, input is 0, stack top 1, pop

At state q1, input is 1, stack top 0, pop

At state q1, input is 0, stack top 1, pop

At state q1, intial is null, stack top null, accepted

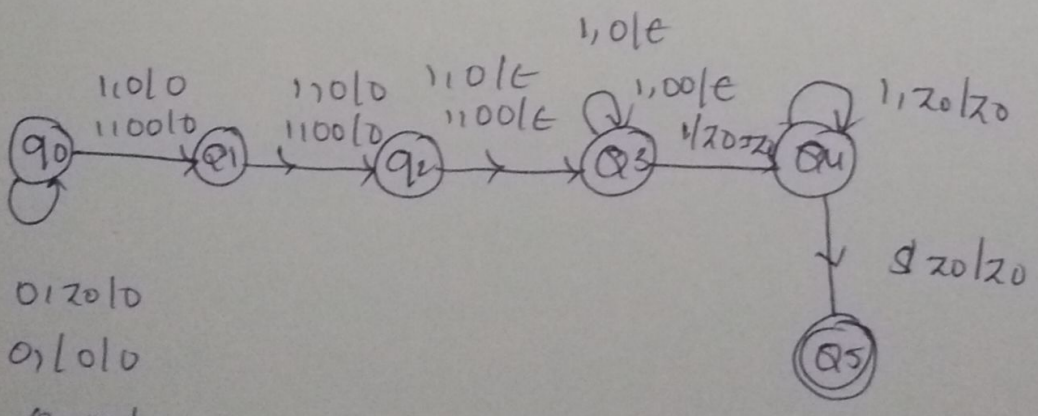
In-Tutorial:

1. Construct a PDA for language $L = \{0^n1^m \mid n >= 1, m >= 1, m > n+2\}$

Solution:

Frist 0s are pushed into stack. When 0s are finished, 2 1's are ignored. therfore for every 1 as input a 0 is popped out of stack. when stack is empty & still some 1s are left them all of them are ignored.

1) on receiving 0 push it into stack. on receiving 1, ignore it & got next state.
2) on receiving 1, ignore it and goto next state
3) on receiving 1, pop a 0 from top of stack & go to next state
4) on receiving 1, pop 0 from top of stack if stack is empty on receiving 1 ignore it & goto next state.
5) on receiving 1 ignore it, if input is finished then goto last state



2. Design NPDA for accepting the language $L = \{a^{2m}b^{3m} \mid m \geq 1\}$

Solution:

$L = \{aabbb, aaaabbbbbb, aaaaaabbbbbbbbb, \ldots\}$

For every 2 'a' there is 3 'b'.

$\delta(q_0, a, z) \vdash (q_1, z)$

$\delta(q_0, a, z) \vdash (q_2, aaaz)$

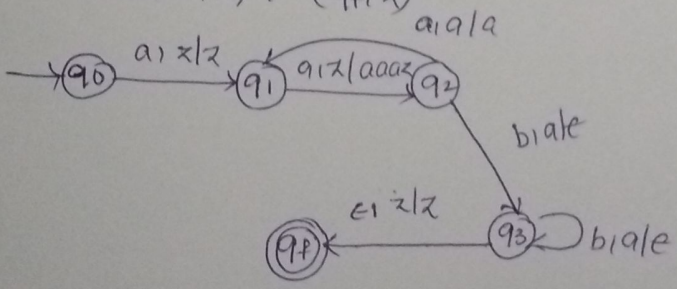
$\delta(q_2, a, aa, az) \vdash (q_1, aaaz)$

$\delta(q_1, a, aaaz) \vdash (q_2, aaaz)$

$\delta(q_2, b, a) \vdash (q_3, \epsilon)$

$\delta(q_3, b, a) \vdash (q_3, \epsilon)$

$\delta(q_3, \epsilon, z) \vdash (q_f, z)$



Post-Tutorial:

1. Why NPDA is more powerful than DPDA?

Solution:

NPDA is more powerful than DPDA because we can add more transitions to it. It is possible for every language to add a transition. For some language, we can construct DPDA there exist an NPDA but there are some language that are accepted by NPDA but are not by DPDA.

2. Draw a diagram to explain the basic structure of PDA and describe 7 tuple(Q, Σ, S, δ, q0, I, F)->

Solution:

A pushdown automata is a way to implement of context free grammar in a similar way we design DFA for a regular grammar. A DFA can remember a finite amount of information, but a PDA can remember an infinite amount of information.

Basically a pushdown automata is–

"finite state machine" + "a stack".

- Q is the finite no. of states
- Σ is a input alphabet
- S is stack symbols
- δ is the transition function : $Q \times (\Sigma \cup \{\varepsilon\}) \times S \times Q \times S^*$
- q0 is the intial state ($q_0 \in Q$)
- I is the intial stack top symbol ($I \in S$)
- F is a set of accepting states ($F \in Q$)

TUTORIAL SESSION 07:

PDA ACCEPTANCY, PUMPING LEMMA

Date of the Session:___/___/____ **Time of the Session** _____ **to**_____

Pre-Tutorial:

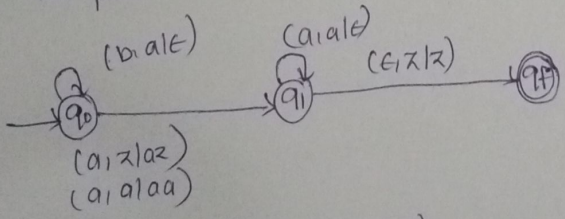
1. Design a PDA that accepts strings of the form $a^n b^n$ (where $n \geq 1$)

Solution:

$L = \{ ab, aabb, aaabbb, \ldots \}$

This can be achieved by pushing a's in STACK and then we will pop a's whenever "b" comes.

Finally at the end of the strings if nothing is left in the STACK then we can declare than language is accepted in the PDA



$\delta(q_0, a, z) \vdash (q_0, az)$

$\delta(q_0, a, a) \vdash (q_0, aa)$

$\delta(q_0, b, a) \vdash (q_1, \epsilon)$

$\delta(q_0, b, a) \vdash (q_1, \epsilon)$

$\delta(q_1, \epsilon, z) \vdash (q_f, z)$

2. Find out whether $L=\{x^n y^n z^n \mid n>=1\}$ is context free or not

Solution:

let L is context free then, L must satisfy pumping lemma At first, choose a number of the pumping lemma, Then, take z as $0^n, 1^n, 2^n$.

Break z into uvwxy, where

$|vwx| \leq n$ & $vx \neq \epsilon$

Hence vwx cannot involve both 0s & 2s, since the last 0 and the first 2 are at least $(n+1)$ position apart. There are 2 cases.

case 1:- vwx has no 2s. Then vx has only 0s & 1s. Then uwy, which would have to be in L, has n 2s, but fewer than n 0s or 1s

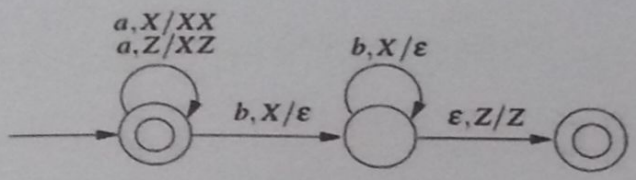
case 2:- vwx has no 0s

Here contradiction occurs.

Hence, L is not a context free Language

In-Tutorial:

1. Consider the transition diagram of a PDA given below with input alphabet Σ = {a, b}and stack alphabet Γ = {X, Z}. Z is the initial stack symbol. What is the language accepted by the following PDA.



Solution:

In this PDA, intial state is also the final state

Transition of given PDA are:

1) $\delta(q_1, a, Z) \vdash (q_1, XZ)$
2) $\delta(q_1, a, X) \vdash (q_1, XX)$
3) $\delta(q_1, b, X) \vdash (q_2, \epsilon)$
4) $\delta(q_2, b, X) \vdash (q_2, \epsilon)$
5) $\delta(q_2, \epsilon, Z) \vdash (q_3, Z)$

2. Let L= { $a^n b^n c^n$ | n>=0 }. By using pumping lemma show that L is not context free language.

Solution:

$L = \{abc, aabbcc, aaabbbccc, \ldots\}$

L $|z| \geq n$

$n=1$, $z = abc$, $\Rightarrow$ can't be spleled

if $n=4$, $z = aaaabbbbcccc$

$z = uvwxy$ then

$z = aaaabbbbcccc \Rightarrow$ split into 5 parts

$u = aaa$, $v = a$, $w = bb$, $x = b$, $y = bccc$

$\therefore |vwx| \leq 4$

$|vx| \geq 1$

$z = uxwxy$ $i \geq a$

if $i = 2$, $z = uv^2wx^2y$

$= aaaaabbbbbccccc \notin L$

if $i = 3$, $z = uv^3wx^3y$ $i \geq 0$

$= aaaaaabbbbbbccc \notin L$

Therefore ux^2wx^2y and w^3wx^3y is not "L"

which is contradiction of taking the assumption initially in CFL.

→ Concluding here the given language is not is CFL

Post-Tutorial:

1. Let L be the language $\{ 0^n1^n2^n \mid n \geq 1 \}$. Show that this language is not a CFL.

Solution:

let L is context-free. Then, L Must satisfy pumping lemma

At first, choose a number n of the pumping lemma Then, take z as $0^n1^n2^n$

Break z into uvwxy, where

$|vwx| \leq n$ and $vx \neq \epsilon$

Hence vwx cannot involve both 0s and 2s, since the last 0 and the first 2 are at least (n+1) positions apart there are 2 cases.

case 1 :- vwx has no 2s. Then vx has only 0s and 1s. then uwx, which would have to be in L, has n 2s, but fewer, than n 0s or 1s.

case 2 :-

vwx has no 0s

Here contradiction occurs.

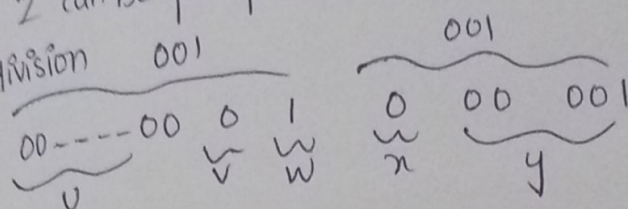
Hence, L is not a context-free languge

2. Let L be the language {ww | w ∈ {0,1}* }. Show that this language is not a CFL.

Solution:

Suppose E is Context-free let p be the pumping length

* consider z_1 $0^n10^n1 \in L$
* z can be pumped, if we make the following division



since $|z| > p$, there are u, w, v, x, y, such that $z = uvwxy$, $|vwx| \leq p$, $|vx| > 0$ and $uv^2wx^2y \in L$ for all ≥ 0

* vwx must straddle the midpoint of z

→ suppose vwx is only in the first half. then in uv^2wx^2y the second half stands wt 1 Thus it is not of the form wow.

→ case when vwx is only in the 2nd half then in uv^2wx^2y the first half ends is a 0. Thus, it is not of the form ww

– suppose, vwx straddels the middle then $vwwxy$ must be of the form $0^i1^i0^ip$, where either i, j is not p. Thus, $uv^2wx^2y \notin$

TUTORIAL SESSION 08:

TURING MACHINES. TM FOR COMPUTING

Date of the Session:___/___/____ **Time of the Session** _____ **to**_____

Pre-Tutorial:

1. Turing machines cannot be considered equivalent to general purpose digital computers. Why? Explain the solution to this objection.

Solution:

Computing Machinery and inteligence is a seminal paper, written by Alan turning on the topic of Artificial intelligence, the paper) published in 1920 in mid, was the first to introduce his concept of what is how known as the turning text to the general public.

Turing is paper consider the question "can machines think ?" Turning says that sinu the defined we should "replace the which is closely related to it and is expressed in relatively, umambiguous idea to replace the word " thick". second he must first find a simple and unambiguous idea to replace the work.

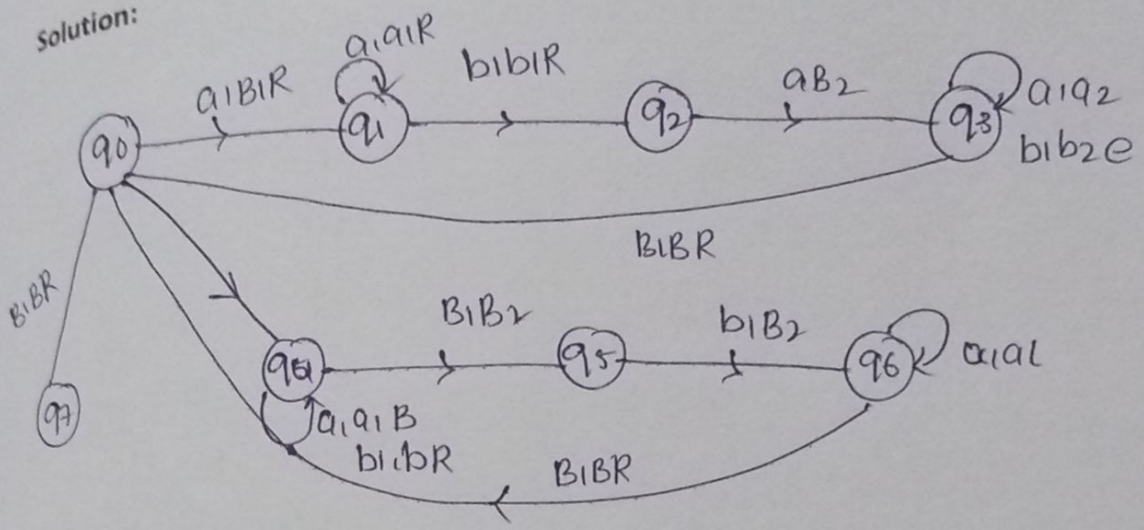
In-Tutorial:

1. Design a Turing machine that computes the following function

$f(x)=x-2$ if $x>2$

$=0$ if $x<=2$

Solution:



2. Construct a Turing Machine for checking the palindrome of the stringof even length?

Solution:

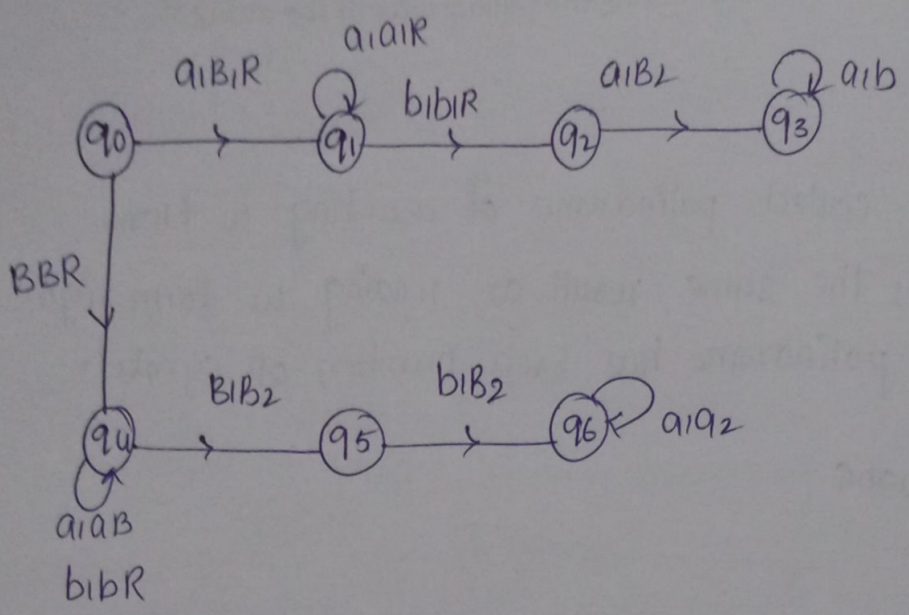
A string let is called palindrome of reacting w from left to right gives the same result as reading w form right to left an even palindrome has Even number of symbol.

Ex:-

input : abaaba
o/p : yes
o/p : abba
o/p : yes
I/P : abbaba
o/p : NO
I/p : Empty string of c
o/p : yes

Start of Computation:-

→ The tape contains the input string w, the tape head is on the left most symbol of w, and the turning machine as in the start state q_0


a1a1R
a1B1R
b1b1R
a1B2
a1b
q0
q1
q2
q3
BBR
B1B2
b1B2
a1a2
q4
q5
q6
a1aB
b1bR

Asst Tutorial:

1. **Find the Difference between Turing machine and Universal Turing machine**

solution:

Turing Machine :-

It was first described by Al an turning it was primarily invented to investigate the computability of a given problem it accepts type-0 grammar which is Recursively enumable language the turning machine has a type of infinite length where can perform read and write operations the infinite cells of the turning machine can contain input symbols and tapes

universal Turning machine :-

Simulates a turing machine universal turing machine can be considered as a subject of all the turning machines, it can match or surpass other turning machines it can match or surpass other turning machines including it self. universal turning machine is like a single turning machine that has a solution to all problems that is computable.

TUTORIAL SESSION 09:

RECURSIVE LANGUAGE

Date of the Session:__/__/____ **Time of the Session** _____ **to**_____

Pre-Tutorial:

1. List out the differences between Recursive languages and recursively enumerable languages with suitable examples each.

Solution:

Recursive languages:-

* the turing machine accepts all valial strings thal are part of language & reject all the strings thal are not part of language & halt

Ex:- $L = \{a^n b^n c^n / n >= 1\}$

Recursively Enumerable languages:-

* This acepts all valid string thal are part of language & reject all the string bul do not halt & starts an infinite loop

Ex:- The set of halting turing machine

In-Tutorial:

1. Let X be a recursive language and Y be a recursively enumerable butnot recursive language. Let W and Z be two languages such that Y' reduces to W, and Z reduces to X' (reduction means the standard many-one reduction). What can be said about the languages W and Z.

Solution:

Hence X is a recursive language & $\bar{X}$ is complement of X. As complement of a recursive language is also recursive, so $\bar{X}$ is also recursive.

According to rule i$\bar{x}$ is recursive so Z is also recursive.

2. Check whether the following problems are undecidable for recursive language(L)?

(i) G is a CFG. L(G)=Ø?
(ii) G is a CFG. L(G) = Σ^*
(iii) M is a Turing Machine. L(M) is regular'
(iv) M is a DFA and N is a NFA. L(M) = L(N)

Solution:

i) CFL is deabable
So, L(G)= ∅ is decidable

ii) L(G)=~~E~~ Σ^* is un-decidable

iii) L(M) regular is un-decidable

iv) L(A) = L(N) is decidable

Post-Tutorial:

1. List out all the properties of Recursive Languages in detail along with examples.

Solution:

Union :- $L_1 \cup L_2$

→ Concentranation = $L_3 = L_1 L_2$

→ kleene closure operator

→ substitution

→ Homo morphism inverse homomorphism

→ Reversal.

kleene closure :-

Ex:- $L_1 = \{a^n b^n c^n \mid n >= 0\}$

$L_1^n = \{a^n b^n c^n \mid n >= 0\}$ is also recursive

intersection & complement :-

Ex:- $L_1 = \{a^n b^n c^n d^m \mid n >= 0 \& m >= 0\}$

$L_2 = \{a^n b^n c^n d^n \mid n >= 0 \& m >= 0\}$

$L_3 = \{n L_2 = \{a^n b^n c^n d^n \mid n >= 0\}$

will be recursive

TUTORIAL SESSION 10:

UNIVERSAL TUNING MACHINE, CHOMSKY HIERARCHYLANGUAGES

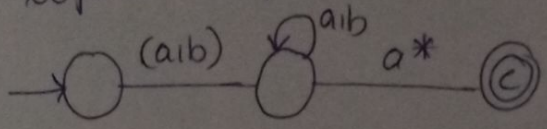
Date of the Session:__/__/____ **Time of the Session** _____ **to**_____

Pre-Tutorial:

1. Classify and explain the different types of Chomsky hierarchy languages with a neat diagram.

Solution:

→ Regular language :-



content free grammar :-

These language are recognised by push elosure automata.

Recursive Enumerable language :-

There language are recognied by turing machine.

In-Tutorial:

1. Explain the types, Grammar, languages and Automation of the Chomsky hierarchy in a tabular form and elaborate in detail.

Solution:

level	Grammar type	language type	Automation type
type1	content sensitive Grammar	content-sensitive	LBA
type 2	content free Grammars	content-free	PDA
type3	Regular grammar	Regular	FA

2. Explain briefly about the Universal Turing Machines with the help of diagrams?

Solution:

A V.T.M is a theoritical Construct in Computer Science and automata theory

components:-

tape

Read | write head

control unit

state register