AIFL

Tutorial-1

R. Lohitha.
2100032325

Pre-Tutorial:-

① Define DFA and NFA normally?

Ans:- DFA:- For each input symbol, one can determine the state to which the machine will move. As it has a finite no. of states, the machine is called Deterministic Finite Machine / Deterministic Finite Automaton.

A DFA can be represented by a 5-tuple.

$(Q, \Sigma, \delta, q_0, F)$ where

Q - finite set of states.

Σ - finite set of symbols called alphabet.

δ - transition function where $\delta : Q \times \Sigma \to Q$

q_0 - Initial state ($q_0 \in Q$)

F - set of final state / states of Q ($F \subseteq Q$)

NFA:-

The finite automata are called NFA. when there exist many paths for specific input from the current state to next state. Each NFA can be translated into DFA but every NFA is Non DFA

(Here $\delta = Q \times \Sigma \to 2^Q$)

The two exceptions are

*) It contains mutiple next states.

*) It contains ϵ transitions.

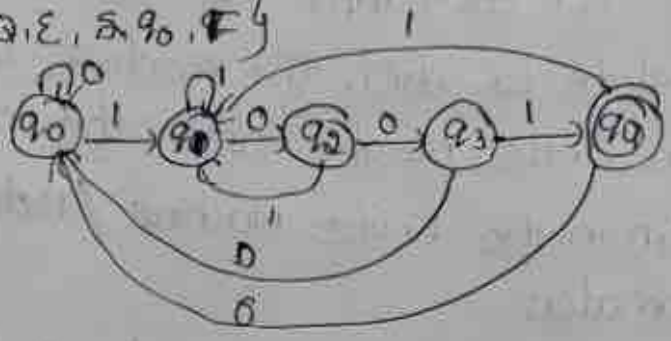
(2) construct a DFA that accepts the language $L = \{ w \in \{0,1\}^+ / w$ contains 1001 or 0110 $\}$

Sol:- $\Sigma = \{0,1\}$

$L = \{1001, 001001, 101001, \cdots\}$

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

Then $\Rightarrow$



	0	1
$\rightarrow q_0$	q_0	q_1
q_1	q_2	q_1
q_2	q_3	q_1
q_3	q_0	q_4
(q_4)	q_0	q_1

$\delta(q_0,0) = q_0$

$\delta(q_0,1) = q_1$

$\delta(q_1,0) = q_2$

$\delta(q_1,1) = q_1$

$\delta(q_2,0) = q_3$

$\delta(q_2,1) = q_1$

$\delta(q_3,1) = q_4$

$\delta(q_3,0) = q_0$

$\delta(q_4,0) = q_0$

$\delta(q_4,1) = q_1$

3) write the steps for converting ϵ-NFA to DFA and vice-versa with an example for each?

Ans:- steps for converting NFA to DFA:-

step 1:- Intially $Q' = \emptyset$

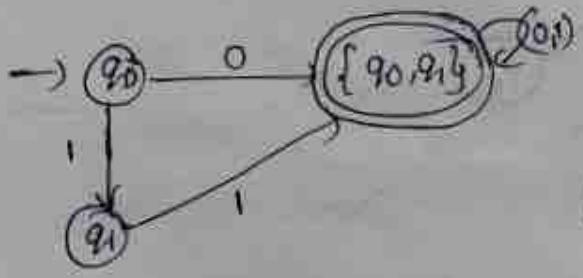
2) Add q_0 to NFA to Q'. Then find the transitions from this start state.

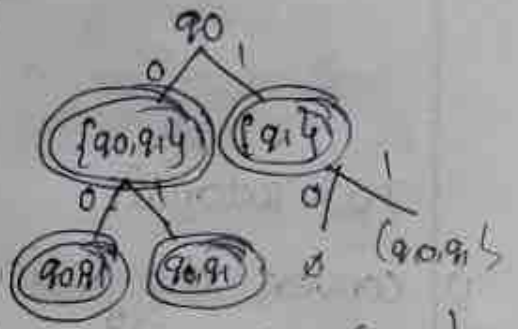
3) In Q', find possible set of states for each input symbol, if this set of states is not in Q'. then add it to Q'.

4) In DFA, the final state will be all states which contain (Final state of NFA)

ex:-

	0	1
→ q_0	$\{q_0, q_1\}$	$\{q_1\}$
q_1	$\emptyset$	$\{q_0, q_1\}$





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

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

$= \{q_0, q_1\}$

$\delta\{q_1, 0\} \cup \delta\{q_1, 1\}$

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

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

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

$= \{q_0, q_1\}$

steps for converting DFA to NFA:-

① Let's assume DFA has state set.

$Q = \{q_0, q_1 \dots q_n\}$

2) Now, we build NFA N as follows

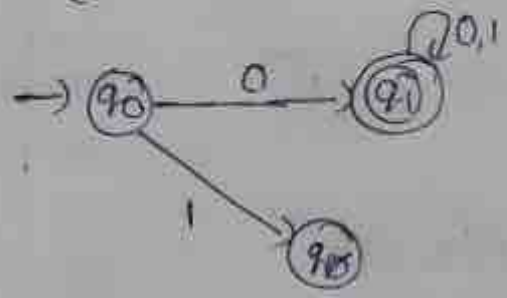
1) Start with DFA D
2) Add an additional accepting state for NFA N such that N will have n+1 total no of states.

Let call new accepting state q_{n+1}

3) Now add an exception state of epsilon ϵ transition for all accepting states to new accepting state q_{n+1}. and make all the original accepting states just normal states.

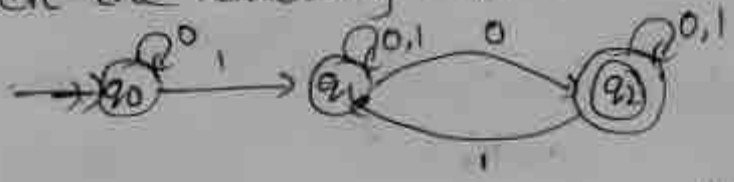
Ex:- ϵ = {0,1 starts with '0'}

L = {0, 00, 01, 000, 001, 010, 0010, 0101, ---}

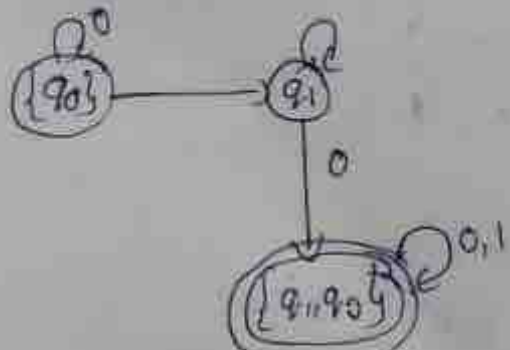


In - Tutorial:

1) convert the following NFA to DFA.



Ans:-



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

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

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

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

$= \{q_1, q_2\}$

② write the algorithm that converts NFA to DFA. explain your algorithm works using the below NFA?

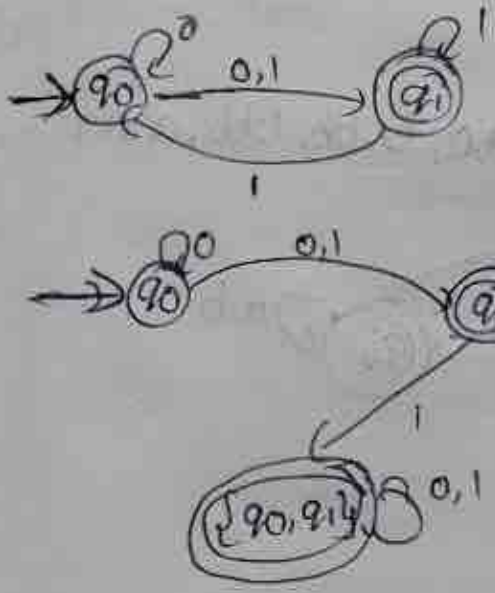
A) Algorithm:-

steps:- ① Intially $Q' = \emptyset$

② Add q_0 of NFA to Q' of DFA

③ In Q', find possible states for each input symbol. If this set of states is not in Q', then add it to Q'.

④ In DFA, the final state will be all states which contain F (final state of NFA)



	0	1
→q0	{q0,q1}	{q1}
q1	∅	{q0,q1}

$\delta([q_0,q_1],0)$
$= \delta(q_0,0) \cup \delta(q_1,0)$
$= \{q_0,q_1\} \cup \{\emptyset\}$
$= \{q_0,q_1\}$

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

Post-Tutorial:-

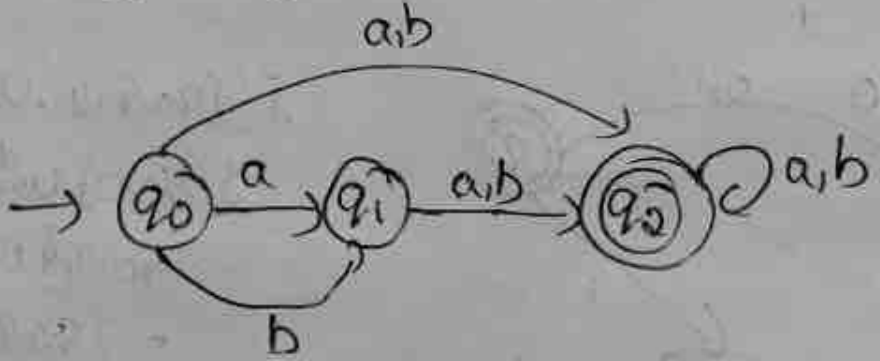
① Differentiate NFA and DFA

NFA	DFA
① $(Q, \Sigma, \delta, q_0, F)$	① $(Q, \Sigma, \delta, q_0, F)$
$\delta = Q \times \Sigma \rightarrow 2^Q$	$\delta = Q \times \Sigma \rightarrow Q$

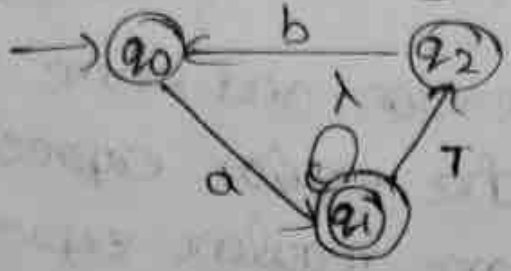
② $Q = \{q_0, q_1\}$, $\Sigma = \{0,1\}$ $q_0 \rightarrow 0$, $q_1 \rightarrow 1$	$Q = \{q_0, q_1\}$ $\Sigma = \{0,1\}$ $q_0 \xrightarrow{0,1}$ $q_1 \xrightarrow{0,1}$
③ Transition may leads to multiple states.	③ Transition leads to unique state.
④ Back track is not required	④ Back tracking is required.
⑤ practical implemention of DFA is feasible	⑤ Not feasible, convert NFA to DFA.

② construct NFA for language $L = \{ w \in \{0,1\}^* / a^* + b^* \}$

Sol:- language $= \{ a, aa, aba, b, bb, bbb, aaaa \ldots \}$

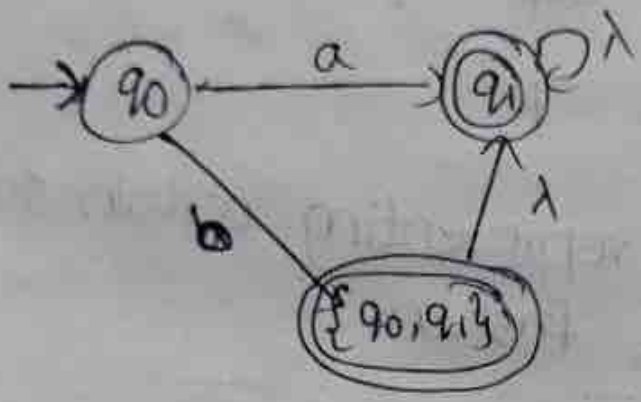


3) convert the following ε-NFA to DFA - 8100032325



	a	b	λ
→q0	q1	∅	∅
q1	∅	∅	{q1,q2}
q2	∅	q0	∅

Ans:-



$\delta(\{q_0,a\} \Rightarrow \{q_1\}$

$\delta\{q_0,b\} = \{\phi\}$

$\delta\{q_1,a\} = \{\phi\}$

$\{q_1,b\} = \{\phi\}$

$\delta(\{q_1,q_2\},a\}$

$= \delta\{q_1,a) \cup \delta(q_2,a)$

$= \phi \cup \phi = \phi$

$\delta(\{q_1,q_2\},b)$

$= \delta(q_1,b) \cup \delta(q_2,b)$

$\phi \cup q_0 = q_0$

$\delta(\{q_1,q_2\},\lambda\}$

$= \delta(q_1,\lambda) \cup \delta(q_2,\lambda)$

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

$= \{q_1,q_2\}$

$\delta(\{q_1,q_2\},b)$

Tutorial-2

2100032325

Pre-Tutorial:-

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

A) Regular expression:-

It is used for representing certain sets of strings in algebraic fashion.

Identity rules:-

$\emptyset + r = r$

$\emptyset \cdot r = r\emptyset = \emptyset$

$\epsilon \cdot r = r \cdot \epsilon = r$

$r + r = r$

$r^* \cdot r^* = r^*$

$r^* \cdot r = rr^* = r^+$

$(r^*)^* = r^*$

$RR^* = R^*R = R^+$

$\epsilon + r^*r = \epsilon + rr^* = r^*$

$(ab)^*a = a(ba)^*$

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

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

$(a+b)c = ac + bc$

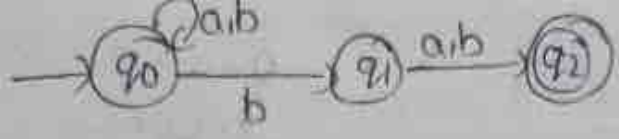
$\epsilon^* = \epsilon$

$\emptyset^* = \epsilon$

② Consider Language L given by regular expression $(a+b)^* b(a+b)$ over the alphabet $\{a, b\}$. Design a DFA that accepts L.

*) convert RE into NFA and the find out DFA from NFA.

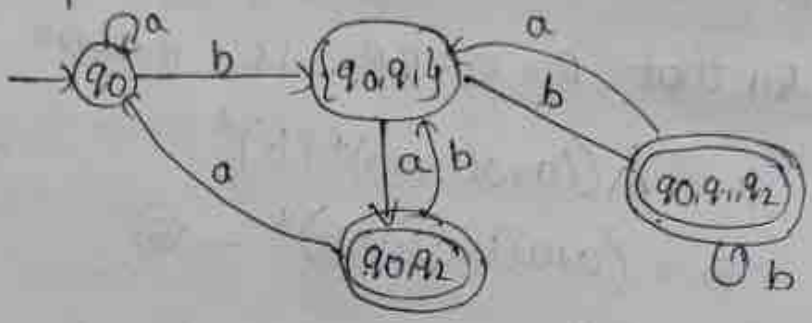
$RE = (a+b)^* b(a+b)$



	a	b
→q0	q0	{q0,q1}
q1	q2	q2
q2	∅	∅

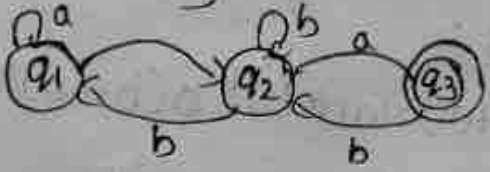
DFA table From NFA.

	a	b
→ q0	q0	{q0,q1}
{q0,q1}	{q0,q1}	{q0,q1,q2}
* {q0,q2}	q0	{q0,q1}
* {q0,q1,q2}	{q0,q2}	{q0,q1,q2}



In tutorial:-

1) provide algorithm to convert NFA into RE create a RE for following NFA.



Ans:

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

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

$q_3 = q_2a \rightarrow (3)$

Now, substitute (2) in (3)

$q_3 = (q_1a + q_2b + q_3b)a$

$q_3 = q_1aa + q_2ab + q_3ab \rightarrow (4)$

subst (1) in (2) from eq's (2)

$q_2 = q_1a + q_2b + q_3b$ [sub's value q_3 in eq]

$q_2 = q_1a + q_2b + (q_2a)b$

$q_2 = q_1a + q_2b + q_2ab$

$q_2 = q_1a + q_2(b+ab)$ $\quad R = Q+RP$ $\quad R = QP^*$

$R = Q + RP$

$q_2 = (q_1a)(b+ab)^*$ —⑤

From eq ①

$q_1 = q_1a + q_2b + \lambda$ [sub eq I in eq ①]

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

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

By that $R = Q+RP$ i.e., $R = QP^*$

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

$q_1 = (a+a(b+ab)^*b)^*$ —⑥

Final state (q_3)

$q_3 = q_2a$ [sub ⑤ in ③]

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

subst eq'6 :-

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

The required regular expression.

Algorithm

① create a new state for the start state of the NFA.

② create a new state for the accept state of the NFA.

③ For each state of transition in the NFA, create a new transition in the new automaton, using rules to determine regular expression for the transition.

④ Add an epsilon transition for the new start state to the old start state.

⑤ This algorithm will convert any NFA into regular expression.

② Explain

use al

Ans:- Arden's

If p a

contain

$R = Q$

steps:-

① For

into st

② Add

③ calc

④ Res

solutio

$R =$

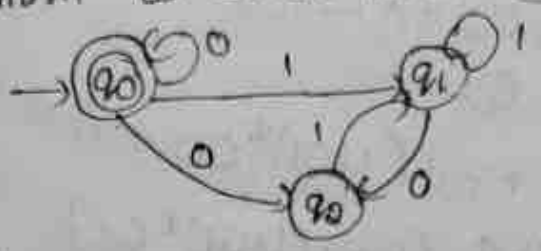
from

$q_1 =$

q_1

R

(2) Explain Arden's theorem to convert FA to RE. use algorithm to convert following DFA to RE.



Ans:- Arden's Theorem:-

If P and Q are 2 RE over Σ and if P doesn't contain ϵ then the following eq is R by

$R = Q + RP$ has a unique solution ie; $R = QP^*$

steps:-

① For each state $q_1, q_2, q_3, \ldots$ all exists that comes into state written in eq format.

② Add epsilon to initial state.

③ Calculate all equations.

④ Result is value of final state.

solution for example:-

$$q_0 = q_00 + \epsilon \rightarrow ①$$
$$q_1 = q_01 + q_11 \rightarrow ②$$
$$q_2 = q_00 + q_10 \rightarrow ③$$

$R = QP \rightarrow ①$ $\Rightarrow Q + QP^*P$ $\quad (R = QP^*)$

$\Rightarrow Q[\epsilon + P^*P] \Rightarrow Q[\epsilon + P^*P]$

$R = QP^*$ → proved.

from ② & ③

$q_1 = q_01 + q_11 + q_101$

$q_1 = q_01 + q_1(1+01)$

$R = Q + RP$ by Arden's theorem.

ie; $R = QP^*$

$q_1 = q_01 (1+01)^* \rightarrow ④$

from ① & ③

$q_0 = q_{00} + q_{100} + \lambda \rightarrow ⑤$

from ④ & ⑤

$q_0 = q_{00} + q_{01}(1+01)^*00 + \lambda$

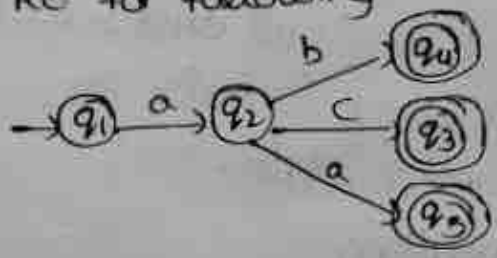
$q_0 = \lambda + q_0(0+1(1+01)^*00)$

$R = Q + RP \Rightarrow R = QP^*$

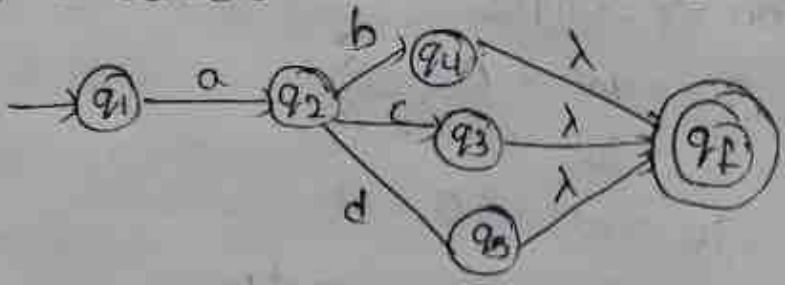
$q_0 = \lambda(0+1(1+01)^*00) \Rightarrow \boxed{q_0 = (0+1(1+01)^*00)}$

Post Tutorial:-

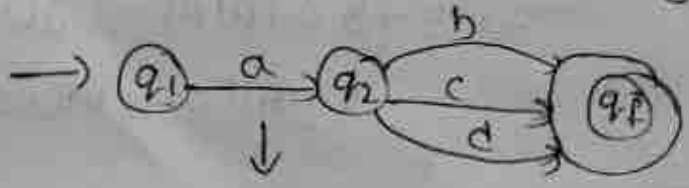
① Find RE for following DFA



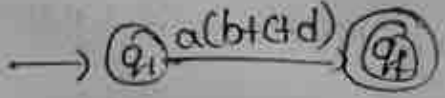
Ans:- there exist multiple final states.
so, we convert them into a single final state



eliminate all intermediate stage $q_4, q_3 q_5$



let's eliminate q_2



$\boxed{\text{Regular expression} = a(b+c+d)}$

(2) For $\Sigma = \{a,b\}$ Let us consider the Regular language $L = \{x / x = a^{2+3k}$ or $x = b^{10+12k}, k \geq 0\}$ what could be the minimum pumping length the constant guaranteed by the pumping lemma for L?

Ans: $L = \{x / x = a^{2+3k}$ (or) $x = b^{10+12k}, k \geq 0\}$

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

pumping lemma:

Let L be an infinite RL. Then there exists some positive integer m such that any $\omega \in L$ with $|\omega| \geq m$ can be decomposed as

$\omega = xyz$

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

is also L for all $i = 0, 1, 2, \ldots$

∴ Minimum pumping length should be 11, because string with length 10 ($\omega = b^{10}$) does not repeat anything, but string with length 11 (ie; $\omega = b^{11}$) will repeat states length of pumping lemma is 24.

2100032385

Tutorial-3

Pre Tutorial

(1) what is the context free grammer. Explain with an example?

Sol:- Context free grammer is a formal grammar which is used to generate all possible strings in a given formal language.

Context free grammar G can be defined by four types as:

$$G = (V, T, P, S)$$

where G : describes the grammar.

V : describes set of non-terminal symbols

T : Finite set of Terminal symbols

P : production rules

S : start symbol.

Ex:- $L = \{ wcw^R \mid w \in (a,b)^* \}$

production rules:- $S \rightarrow aSa$

$S \rightarrow bSb$

$S \rightarrow c$

Now, check the string abbcbba derived from given.

$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abbSbba \Rightarrow abbcbba$

By applying the productions $S \rightarrow aSa$, $S \rightarrow bSb$, $S \rightarrow$ we get string abbcbba.

In-Tutorial: 2100032301

① construct a CFG for a language

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

sol: $L = \{aa, bb, abba, aabbaa, abaaba --- \}$

production rules:- S → aSa
S → bSb
S → c

Now, check the strings ~~are not~~ are reverse (or) not

For example:- abbcbba.

S ⇒ aSa ⇒ abSba ⇒ abbSbba ⇒ abbcbba

It convert and get a string.

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

sol:- S → aB/bA aabbabba
A → a/aS/bAA
B → b/bS/aBB

Left most derivation:-

S → aB ⇒ aaBB ⇒ aabB ⇒ aabbS ⇒
⇐ aabbabbA ⇐ aabbabS ⇐ aabbaB
↳ aabbabba

Right most derivation:-

S → aB ⇒ aaBB ⇒ aaBbS ⇒ aaBbbA ⇒
aabbabba ⇐ aabbAbba ⇐ aabSbba ⇐ aaBbba

2100032325

Post Tutorial:-

(1) Generate CFG for the language

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

Sol:- Given $L = \{ 0^i 1^j 0^k \mid j > i+k \mid i, k \geq 1 \}$

$\Rightarrow$ 0 1 · 1 10 let $i=1, k=1$

$j = 3$

$L = \{ 0^1 1^3 0^1, \dots \}$

$S \Rightarrow XYZ$

$X \Rightarrow 0X1 / 01$

$Y \Rightarrow 1Y / 1$

$Z \Rightarrow 1Z0 / 10$

$S \rightarrow XYZ \Rightarrow 0X1YZ \Rightarrow 0011YZ \Rightarrow 00111YZ$

$\Rightarrow 00111 1Z \Rightarrow 00111110$

$i=2 \quad j=5, k=1$

$5 > 3$

$j > i+k$

Tutorial-4 2100032325

PARSE TREE, AMBIGUITY IN CFG:-

Pre-Tutorial:-

① Differentiate ambiguous and unambiguous grammar.

Ambiguous	unambiguous
*) The leftmost and Right most derivations are not same.	*) the left most and rightmost derivations are same.
*) Amount of non-terminal in ambiguous grammar is less than unambiguous grammar.	*) Amount of non-terminals In unambiguous grammar is greater than in ambiguous grammar. Length of parse tree is large.
*) Length of parse tree is short.	
*) It generate more than one parse tree	*) It generates only one parse tree.
*) It contains ambiguity.	*) It does not contain any ambiguity.

In-Tutorial:-

① consider the following grammar

$S \rightarrow ABB / C$

$A \rightarrow \epsilon / aA$

$B \rightarrow \epsilon / bB$

Derive the string acb using left most and rightmost derivation. show the parse trees in your derivation.

Ans: Left most derivation

S

ASB S→ASB

aASB A→aA

aεSB A→ε

aεCB S→C

aεCbB B→bB

aεCbε B→ε

abC

Right most derivation

S

ASB S→ASB

ASbB B→bB

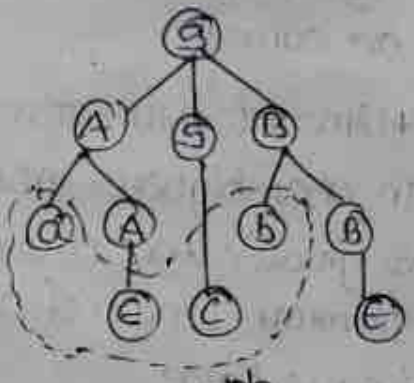
ASbε B→ε

ACbε S→C

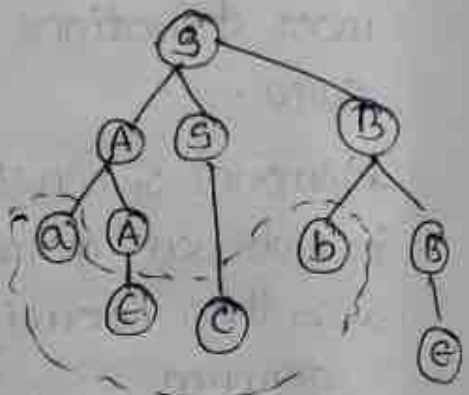
aACbε A→aA

aεCbε A→ε

abC



abb



acb

② Consider the following grammar:

S→ aS/ε

The language generated by this grammar.

L = {aⁿ, 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 right most derivation exactly same draw the parse of the same.

Sol: i)

Let us consider string ω = aaa

Leftmost derivation

$S \rightarrow aS/\epsilon$

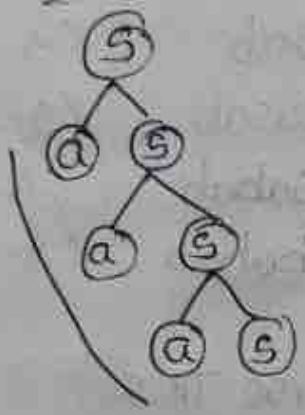
S
aS (S→aS)
aaS (S→aS)
aaaS (S→aS)
aaaϵ (S→ϵ)
aaa

Rightmost derivation

$S \rightarrow aS/\epsilon$
aS (S→aS)
aaS (S→aS)
aaaS (S→aS)
aaaϵ (S→ϵ)
aaa

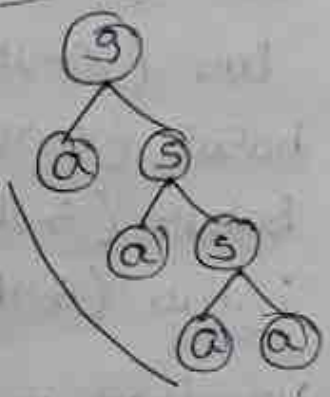
ii) parse Tree for the given

Leftmost Tree



aaa

Right most Tree



aaa

clearly, the leftmost derivation = The Rightmost derivation parse Tree parse tree

POST-Tutorial:-

① consider the following grammar.

$S \rightarrow SaS/b$

It is an ambiguous grammar? Generate the string babab from this grammar to prove your point.

Sol: the given grammar

$S \rightarrow SaS/b$ $\omega = babab$

Leftmost derivation

S
SaS (S→SaS)
SaSaS (S→SaS)
baSaS (S→b)
babaS (S→b)
babab ε

Rightmost Derivation

S
SaS (S→SaS)
SaSaS (S→SaS)
SaSab (S→b)
SabAb (S→b)
babab

(or.)

→ S
SaS (S→SaS)
baS (S→b)
baSaS (S→SaS)
babaS (S→b)
babab (S→b)

→ S
SaS (S→SaS)
Sab (S→b)
SaSab (S→SaS)
SabAb (S→b)
babab (S→b)

→ More than one left most parse Tree
→ More than one right most parse Tree
→ More than one parse Tree

∴ The given grammar is ambiguous.

Tutorial-5

2100032325

(1) Explain the simplication grammar? Mention its use? Elaborate the steps that are followed In simplication process?

A) simplication of grammer means reduction of grammar by removing useless symbals.

It use Includes facilitating laguage learning making test more accessible to individuals with language difficulties, and improving reading comprehensions for language learning.

The steps followed by in automated simplification productions:

(1) Inputting the original text into the software
(2) Running algorithms to identify compler sentence (or) vocabulary and structures.
(3) Modifying the text according to predefined rules or guidelines for grammar simplication.
(4) Evaluting the output for accuracy and clarity.
(5) Making further adjustments as necessary to ensure that the simplified text conveys the same meaning as the original.

(2) Find a reduced grammer equivalent to the grammar G, having, production rules.

$S \rightarrow AC/B$

$A \rightarrow a$

$C \rightarrow c/BC$

$E \rightarrow aA/e$

Sol:- Phase 1:- $Ter = \{a, C, e\}$

$\omega_1 = \{A, C, E\}$

$\omega_2 = \{A, C, E, S\}$

$\omega_3 = \{A, C, E, S\}$

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

P: $S \rightarrow AC$

$A \rightarrow a$

$C \rightarrow c$

$E \rightarrow aA/e$

Phase 2:- $Y_1 = \{S\}$

$Y_2 = \{S, A, C\}$

$Y_3 = \{S, A, C, a, c\}$

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

$G'' = \{(A, C, S), (a, c), P(S)\}$

P: $S \rightarrow AC$, $A \rightarrow a$, $C \rightarrow X$

In Tutorial:-

(1) Remove unit productions from the following grammar.

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

Sol:- procedure for Removal:-

step 1:- To remove $A \rightarrow B$, add production.

$A \to x$ to the grammar rule whenever $B \to x$ occurs in the grammer.

step 2:- Delete $A \to B$ from the grammar.

step 3:- Repeat from step 1 until all unit productions are removed.

the given grammar is

$P: S \to AC, A \to a, C \to X/b, X \to Y, Y \to Z, Z \to a$

productions:-

$C \to X, X \to Y, Y \to Z$

$Y \to Z, Z \to a$ from step 1 add $Y \to a$

1) $P: S \to AC, A \to a, C \to X/b, X \to Y, Y \to a, Z \to a$

2) Since $Y \to a$, we add $Z \to a$

$P: S \to AC, A \to a, C \to X/b, Z \to a, Y \to a, Z \to a$

3) Since $X \to a$, we add $C \to a$

$P: S \to AC, A \to a, C \to a/b, Z \to a, Y \to a, Z \to a$

Remove the unreachable symbols

$\therefore P \Rightarrow S \to AC, A \to a, C \to a/b$

(2) A grammar G is defined with rules $S \to XA/BB$, $B \to b/SB$, $X \to b$, $A \to a$. write the productions obtained after normalized GNF of G.

sol: steps to convert a given CFG to GNF:

step 1: check if the given CFG has any unit

productions (or) NULL productions and remove of there are any.

step 2:- check whether the CFG is already in chomsky Normal Form (CNF) and convert it to CNF if it is not

step 3:- change the names of the Non-Teriminal symbols into the same A_i in ascending order of i.

The given grammar with rules.

$S \rightarrow XA / BB$
$B \rightarrow b / SB$
$X \rightarrow b$
$A \rightarrow a$

Replace: S with A_1
X with A_2
A with A_3
B with A_4

we get

$A_1 \rightarrow A_2A_3 / A_4A_4$
$A_4 \rightarrow b / A_1A_4$
$A_2 \rightarrow b$
$A_3 \rightarrow a$.

step 4:- After the rules so that the Non-Terminals are in ascending order, such that,

If the production is of the form $A_i \rightarrow A_j X$, then $i<j$ and should never be $i \geq j$

$A_4 \rightarrow b / A_1A_4$
$A_4 \rightarrow b / A_2A_3A_4 / A_4A_4A_4$
$A_4 \rightarrow b / A_3A_4 / A_4A_4A_4 \rightarrow$ Left Recursion.

(5) Remove Left recursion.

$Z \rightarrow A_4A_4Z / A_4A_4$

$A_4 \rightarrow bA_3A_4 / bZ / bA_3A_4Z$

Now the grammar is:

$A_1 \rightarrow A_2A_3 / A_4A_4$

$A_4 \rightarrow b / bA_3A_4 / bZ / bA_3A_4Z$

$Z \rightarrow A_4A_4 / A_4A_4Z$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

In GNF we are not allow to have variable In the beginning we need to modify A_1

$A_1 \rightarrow bA_3 / bA_4 / bA_3A_4A_4 / bZA_4 / bA_3A_4ZA_4$

$A_4 \rightarrow b / bA_3A_4A_4 / bZA_4 / bA_3A_4ZA_4 / bA_4Z / bA_3A_4A_4Z / bZA_4Z / bA_3A_4ZA_4Z$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

Post-Tutorial:-

① convert the following

a) CFG into CNF

$S \rightarrow ASA / aB$, $A \rightarrow B/S$, $B \rightarrow b/\epsilon$.

Sol: step① - Since S appears in RHS, we add a new state, S and $S' \rightarrow S$ is added to the production

P: $S' \rightarrow S$, $S \rightarrow ASA / aB$, $A \rightarrow B/S$, $B \rightarrow b/\epsilon$

② Remove the Null productions:

$B \rightarrow \epsilon$ and $A \rightarrow \epsilon$;

After removing $B \rightarrow \epsilon$: P: $S' \rightarrow S$, $S \rightarrow ASA / aB / a$, $A \rightarrow B/S/\epsilon$, $B \rightarrow b$

After removing $A \rightarrow \epsilon$: P: $S' \rightarrow S$, $S \rightarrow ASA / aB / a / AS / SA / S$, $A \rightarrow B/S$, $B \rightarrow b$

(3) Remove the unit productions
$S \to S$, $S' \to S$, $A \to B$ and $A \to S$.

After removing $S \to S$:-

P: $S' \to S$, $S \to ASA/aB/a/AS/SA$,
$A \to B/S$, $B \to b$

After removing $S' \to S$:-

P: $S' \to ASA/aB/a/AS/SA$,
$S \to ASA/aB/a/AS/SA$,
$A \to B/S$, $B \to b$

After removing $A \to B$:-

P: $S' \to ASA/aB/a/AS/SA$,
$S \to ASA/aB/a/AS/SA$,
$A \to b/S$, $B \to b$

After removing $A \to S$:-

P: $S' \to ASA/aB/a/AS/SA$,
$S \to ASA/aB/a/AS/SA$,
$A \to b/ASA/aB/a/AS/SA$,
$B \to b$

(4) Now change the productions
$S' \to aB$, $S \to aB$, and $A \to aB$

Finally we get

P: $S' \to AX/YB/a/AS/SA$,
$S \to AX/YB/a/AS/SA$,
$A \to b/AX/YB/a/AS/SA$,
$B \to b$
$X \to SA$
$Y \to a$.

which is required chomsky Normal form for the given CFG.

(2) write the steps for removing null productions and unreachable symbols? Explain with an ex of your own?

Sol:- (1) Remove null productions:-

A production is considered null if its right hand side is empty.

For ex, consider a CFG with the following production : $A \rightarrow \epsilon$, This production can be removed.

(2) Remove unreachable symbols:-

*) A symbol is considered unreachable if it can never appear in any string generated by the CFG.

. To identify unreachable symbols, start form the start symbol and make all symbols reachable.

Then remove all the symbols that were not marked as reachable.

Ex:- consider the following CFG:

$G = (N, T, P, S)$

when $N = \{S, A, B\}$

$T = \{a, b\}$

$P = \{S \rightarrow \epsilon, S \rightarrow A, A \rightarrow B, B \rightarrow a\}$

S = S

① Remove the null production:-

The null production: $(S \to \epsilon)$ can be remove

New grammer:

$G = (N, T, P, S)$

where $N = \{S, A, B\}$

$T = \{a, b\}$

$P = \{S \to A, A \to B, B \to a\}$

$S = S$

② Remove unreachable symbols.

starting from the start symbols "S", we can reach "A", "B" and "a". so, "A", "B", and "a" are reachable symbols.

The unreachable symbols can be removed.

New grammar.

$G = (N, T, P, S)$

where $N = \{S, B\}$

$T = \{a, b\}$

$P = \{S \to A, B, a\}$

$S = S$.