ATFL ID: 2100032517

Tutorial 1: DFA, NFA

pre tutorial

① Define NFA and DFA normally:

DFA:

for each input symbol one can determine the state to which the machine will move As it has a finite number of states the machine is deterministic finite Automata

DFA can be represented by 5-tuple

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

Q - finite set of states

Σ - input alphabet

δ - transition function

q_0 is initial state

f is non empty set of final state

NFA:

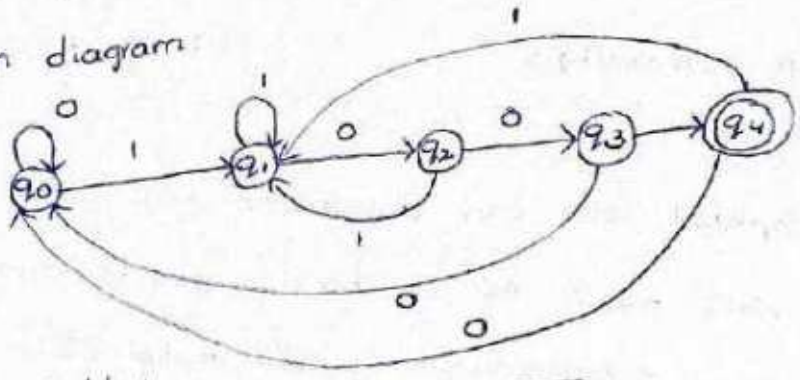
The finite automata are called NFA when there exist many paths for specific input from the current state to next state each NFA can be translated into DFA but every NFA is not DFA

The two exceptions are:

* it contains multiple next states
* It contains ε transitions

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

Transition diagram:



Transition table:

δ / Σ	0	1
→ 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

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

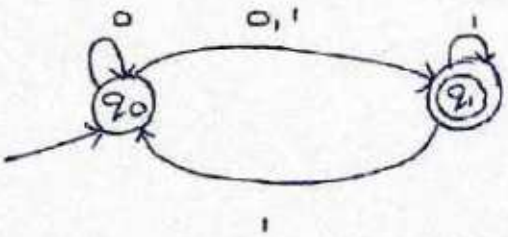
A. step①:- Initially $q' = \phi$

step②:- Add q_0 of NFA to q' Then find the transitions from this start state

step③: 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'

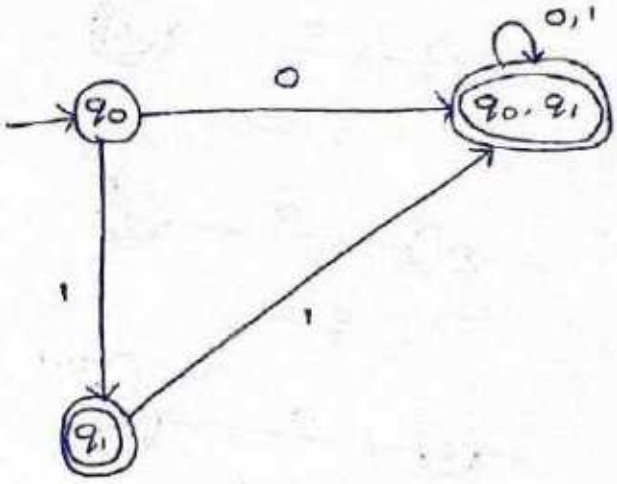
step④: In DFA, the final state will be all states which contain f (final states of NFA)

Eg:-



	0	1
q_0	$\{q_0, q_1\}$	$\{q_1\}$
* q_1	ϕ	$\{q_0, q_1\}$

	0	1
q_0	$\{q_0, q_1\}$	$\{q_1\}$
* q_1	ϕ	$\{q_0, q_1\}$
* $\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_1\}$



steps for converting DFA to NFA

step ①: Let's assume DFA D has state set

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

step ②: Now we build NFA N as follows

(i) start with DFA D

(ii) Add an additional accepting state for NFA N, such that N will have n+1 total no of states

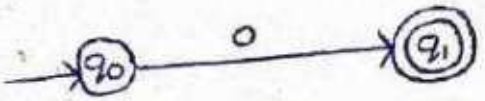
Lets call new accepting state q_{n+1}

(iii) Now, add an epsilon ϵ transition from all accepting state q_{n+1} and make all the original accepting states just normal states

Eg:- $\Sigma = \{0, 1$ starts with '0' $\}$

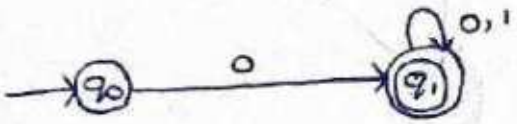
$L = \{ 0, 00, 01, 000, 001, 010, 0010, 0101, ----- \}$

$\varepsilon = 0$

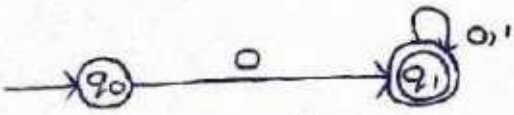


$\varepsilon = 00$

$\varepsilon = 01$

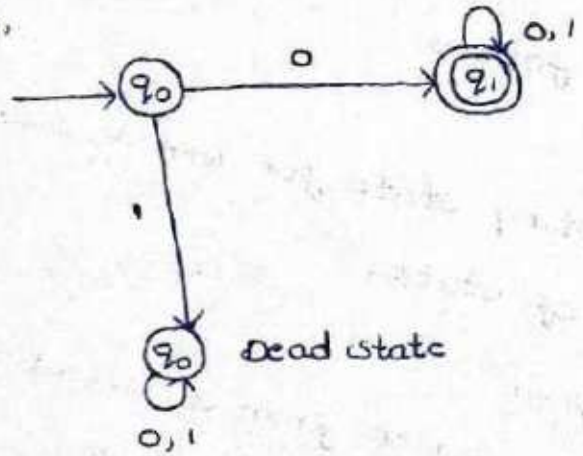


$\varepsilon = 000$



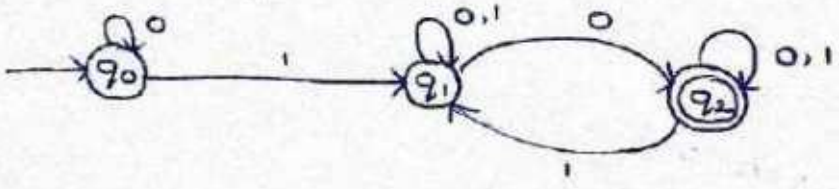
$\varepsilon = 001$ (we cannot draw)

So,



In tutorial

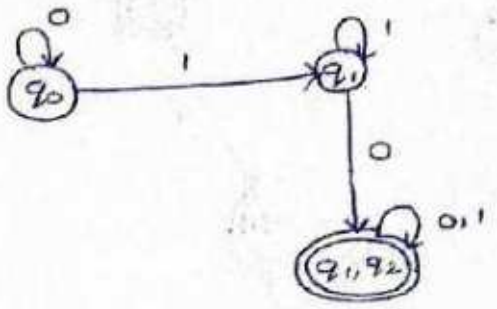
① convert the following NFA to DFA



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

Now,

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



(2) Write the algorithm that converts NFA to DFA. explain your algorithm works using the below NFA ?

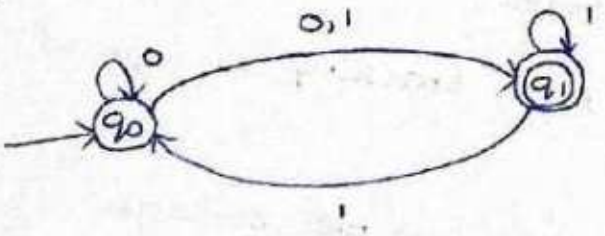
A- Algorithm :-

step ① : initially $Q' = \phi$

step ② : Add q_0 of NFA to Q' of DFA

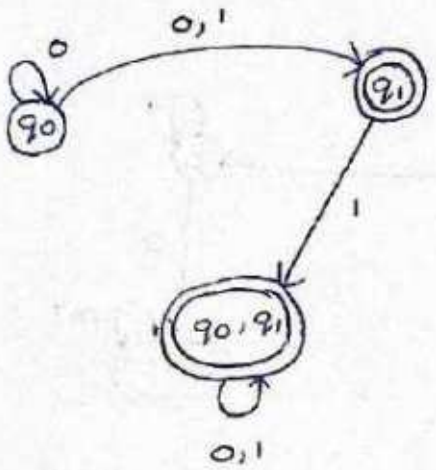
step ③ : In Q', find possible states for each input symbol if this set of states is not in Q', then add it to Q'

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



	0	1
q_0	$\{q_0, q_1\}$	$\{q_1\}$
q_1^*	ϕ	$\{q_0, q_1\}$

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



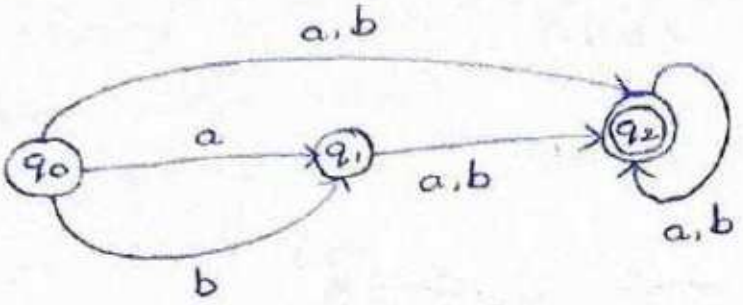
post tutorial :

① Differentiate NFA and DFA.

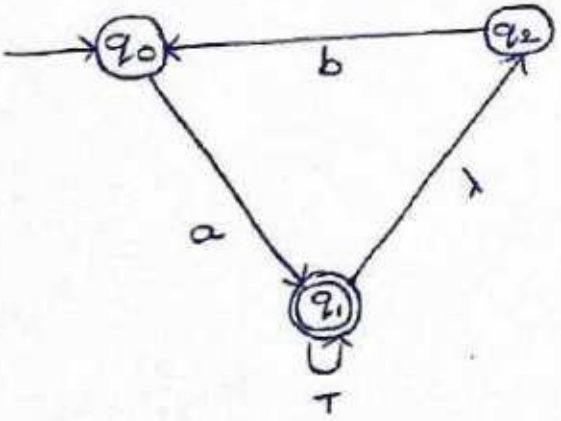
NFA	DFA
① $(Q, \Sigma, \delta, q_0, F)$ $\delta = Q \times \Sigma \to 2^Q$	① $(Q, \Sigma, \delta, q_0, F)$ $\delta = Q \times \Sigma \to Q$
② $Q = \{q_0, q_1\}$, $\Sigma = \{0, 1\}$ $q_0 \to 0$ $q_1 \to 1$	② $Q = \{q_0, q_1\}$, $\Sigma = \{0, 1\}$ 
③ Transition may leads to multiple states	③ Transition leads to unique state
④ Back Tracking is not required	④ Back tracking is required
⑤ practical implementation of DFA is feasible	⑤ Not feasible, convert NFA to DFA

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

$L = \{a, aa, aaa, b, bb, bbb, aaaa \cdots\}$

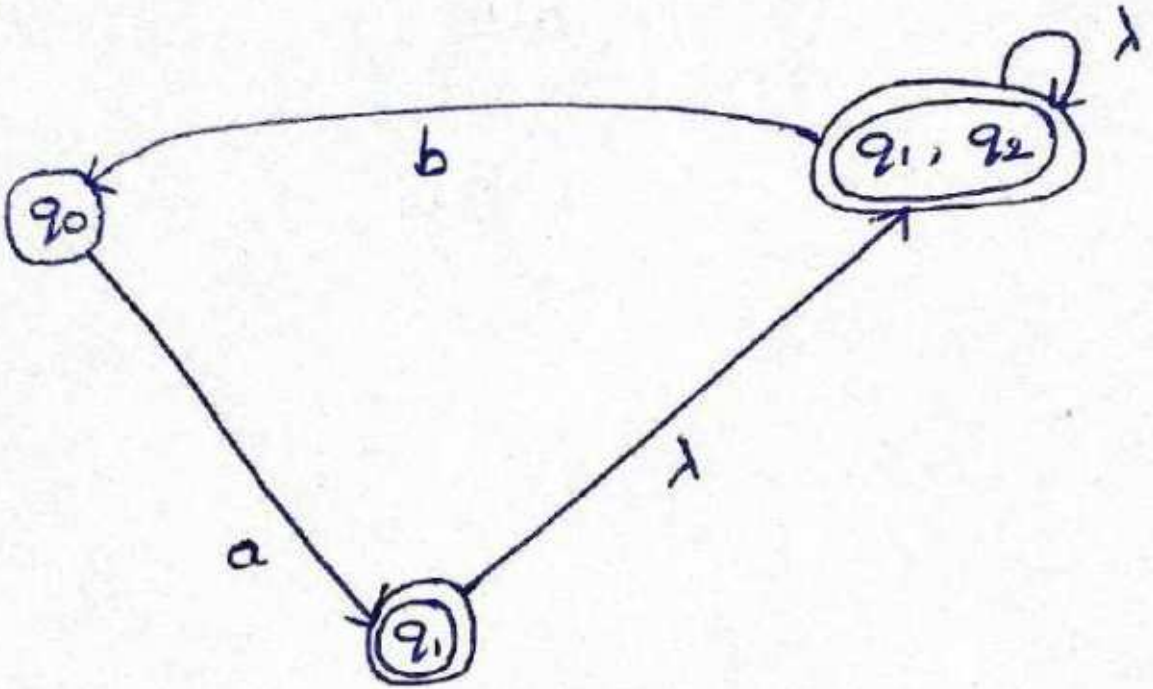


(3) convert the following ϵ-NFA to DFA



	a	b	λ
q_0	q_1	ϕ	ϕ
q_1	ϕ	ϕ	$\{q_1, q_2\}$
q_2^*	ϕ	q_0	ϕ

	a	b	λ
q_0	q_1	ϕ	ϕ
q_1	ϕ	ϕ	$\{q_1, q_2\}$
* $\{q_1, q_2\}$	ϕ	q_0	$\{q_1, q_2\}$



Tutorial 2: Regular expression

pre tutorial:

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

Regular expression:

it is used for representing certain sets of strings in algebraic fashion

identity rules:

$\phi + r = r$

$\phi \cdot r = r \cdot \phi = \phi$

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

$r + r = r$

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

$r^* r = r r^* = r^*$

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

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

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

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

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

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

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

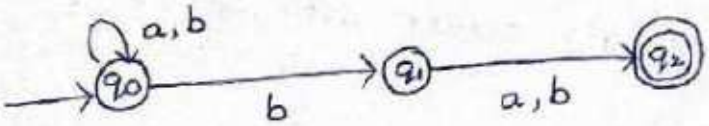
$\epsilon^* = \epsilon$

$\phi^* = \epsilon$

(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 find DFA from NFA

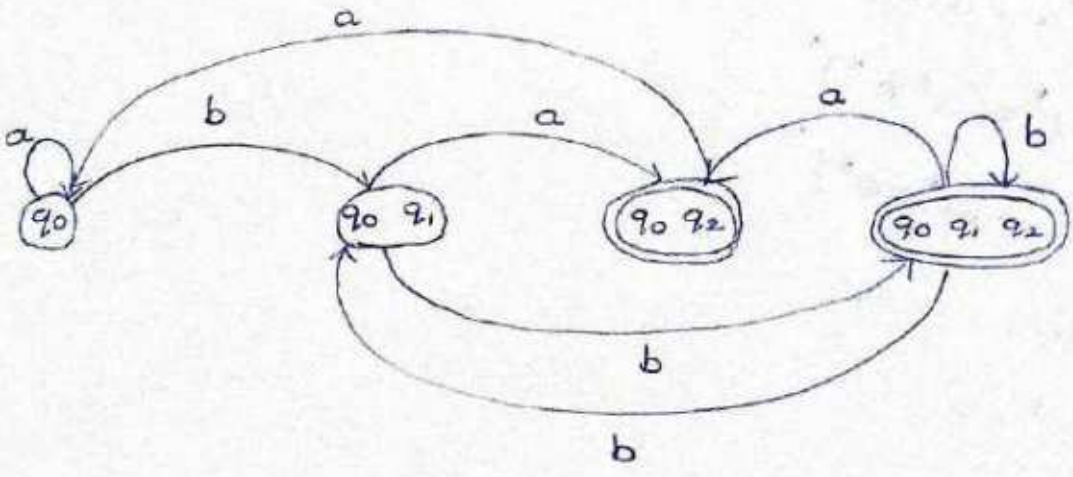
$RE = (a+b)^* b(a+b)$



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

DFA table from NFA

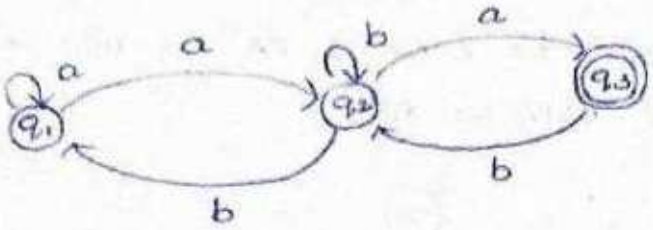
	a	b
q_0	q_0	$\{q_0, q_1\}$
$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$
* $\{q_0, q_2\}$	q_0	$\{q_0, q_1\}$
* $\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$



In - tutorial

① provide algorithm to convert NFA into RE

create a RE for following NFA



$q_1 = q_1a + q_2b$

$q_2 = q_2b + q_1a$

$q_3 = q_2a$

Now,

$q_1 = q_1a + q_2b + \epsilon$ — ①

$q_2 = q_1a + q_2b$ — ②

$q_3 = q_2a$ — ③

Sub ② in ①

$q_1 = q_2 + \epsilon$

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

$q_3 = q_1aa + q_2ab + q_3ab$ — ④

from eq ② $q_2 = q_1a + q_2b + q_3b$

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

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

$R \quad q \quad R \quad p$

$q_2 = (q_1a)(b+ab)^*$ — ⑤

from eq ③

$q_1 = \epsilon + q_1a + q_2b$

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

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

$R \quad q \quad R \quad p$

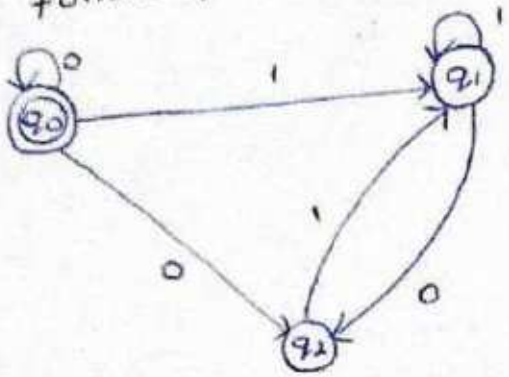
$q_1 = \epsilon((a + a(b+ab)^*)b)^*$ $\boxed{ER = R}$

$q_1 = (a + (a(b+ab)^*b)^*$ — ⑥

final state $q_3 \Rightarrow q_3 = q_2 a$

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

(2) explain arden's theorem to convert FA to RE use algorithm to convert following DFA to RE



Ardens theorem:-

If P and Q are 2 RE over Σ and if P doesn't contain ϵ then the following equation in R by $R = Q + RP$ has a unique solution

$$R = QP^*$$

steps

1) for each state $q_1, q_2, q_3 \cdots$ all exists that comes into state written in equation format
2) Add epsilon to initial state
3) calculate all equations
4) Result is value of final state

solution:

$q_0 = q_0 0 + \epsilon$ — (1)

$q_1 = q_0 1 + q_1 1$ — (2)

$q_2 = q_0 0 + q_1 0$ — (3)

from (2) and (3)

$q_1 = q_{01} + q_{11} + q_{01}$

$q_1 = q_{01} + q_1(1+01)$

R Q R P

$\boxed{q_1 = q_{01}(1+01)^*}$ — (4)

from (1) and (3)

$q_0 = q_{00} + q_{10} + \epsilon$ — (5)

from (4) and (5)

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

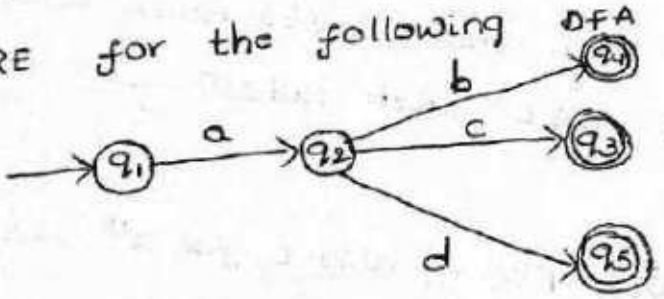
$q_0 = \epsilon + q_0(0+1(1+01)^*00)$

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

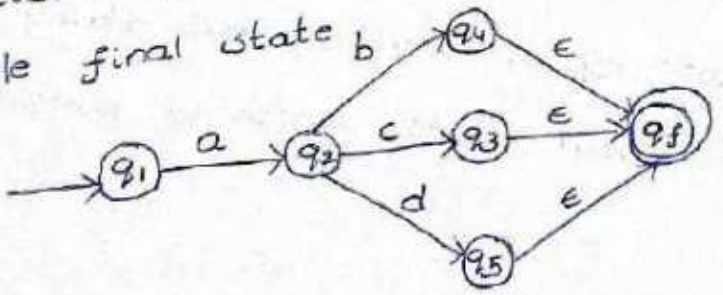
$\boxed{q_0 = 0+1(1+01)^*00}$

post tutorial

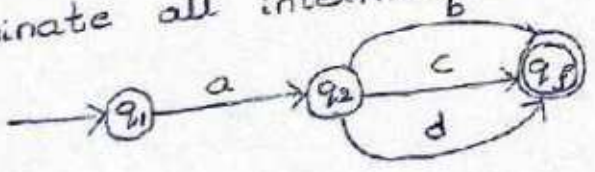
① find RE for the following DFA

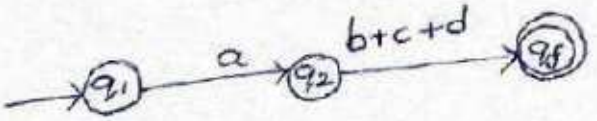


There exists multiple final states so we convert them into a single final state

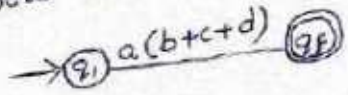


eliminate all intermediate stages q4, q3, q5





Lets eliminate q_2



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?

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

$L = \{a^2, a^5, \ldots\ldots ()\ b^{10}, b^{34}, - - - -\}$

pumping Lemma:

Let L be an infinite RL then there exists some positive integer m such that any $w \in L$ with $|w| \geq m$ can be decomposed as $w = xyz$

with $|xy| \leq m$ such that $w_i = xy^iz$ is also L for all $i = 0, 1, 2, \ldots$

∴ minimum pumping length should be 11, because string with length 10 ($w = b^{10}$) does not repeat anything, but string with length 11 (i.e, $w = b^{11}$) will repeat states length of pumping lemma is 24

Tutorial - 3:

pre tutorial:

① what is context free grammer. explain with an example?

context free grammar is a formal grammar which is used to generate all possible strings in a given formal language

it can be defined by four steps

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

where G = Grammar

V = set of non terminal symbols

T = finite set of terminal symbols

P = production rules

S = start symbol

Example:

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

production rules: $S \to aSa$

$S \to bSb$

$S \to c$

Now check the string abbcbba string can be derived from the given CFG

$S \to aSa$

$S \to abSba$

$S \to abbSbba$

$S \to abbcbba$

by applying production $S \to aSa$, $S \to bSb$ recursively and finally the production $S \to c$ we get string abbcbba

In tutorial:

① construct a CFG for a language

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

$L = \{aa, bb, abba, aabbaa, ababa, \}$

production rules

$S \to asa$

$S \to bsb$

$S \to c$

Now check the string abbcbba string can be derived from the given CFG

$S \to asa$

$S \to absba$

$S \to abbsbba$

$S \to abbcbba$

String = abbcbba

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

$S \to aB / bA$

$A \to a / as / bAA$

$B \to b / bs / aBB$

"aabbabba"

Left most derivation

S

aB $S \to aB$

aaBB $B \to aBB$

aabB $B \to b$

aabbS $B \to bS$

aabbaB $S \to aB$

aabbabs B→bs

aabbabbA S→bA

aabbabba A→a

Right most derivation

S

aB S→aB

aaBB B→aBB

aaBbs B→bS

aaBbbA S→bA

aaBbba A→a

aabsbba B→bS

aabbAbba S→bA

aabbabba A→a

post tutorial

① Generate CFG for the language

$L = \{0^i, {}^j 0^k / j > i+k\}$

$L = \{0^i 1^j 0^k / j > i+k / i, k \geq 1\}$

0 1 1 1 0

S → XYZ

X → 0X1 | 01

Y → 1Y | 1

Z → 1Z0 | 10

S → XYZ

→ 0X1YZ X → 0X1

→ 0011YZ X → 01

→ 00111YZ Y → 1Y

→ 0011111Z Y → 1

→ 00111110 Z → 10

i=2 j=5 k=1

5 > 3

j > i+k

Tutorial 4 - parse tree, ambiguity CFG

pre tutorial

① Differentiate ambiguous and unambiguous grammar

ambiguous	unambiguous
① The leftmost and rightmost derivations are not same	① The leftmost and rightmost derivations are same
② Amount of non-terminals in ambiguous grammar is less than unambiguous grammar	② Amount of non-terminals in unambiguous grammar is greater than ambiguous grammar
③ length of parse tree is short	③ length of parse tree is large
④ it generates more than one parse tree	④ it generates only one parse tree
⑤ it contains ambiguity	⑤ it doesnot contain ambiguity

In tutorial

① consider the following grammar

$S \rightarrow ASB|c$

$A \rightarrow \epsilon | aA$

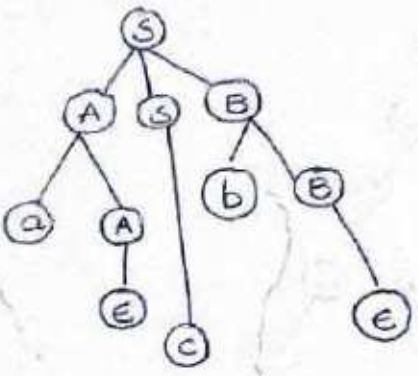
$B \rightarrow \epsilon | bB$

Derive the string acb using leftmost and rightmost derivation show the parse tree for your derivation

LMD:

S	
ASB	S → ASB
aASB	A → aA
aεSB	A → ε
aεcB	S → c
aεcbB	B → bB
aεcbε	B → ε

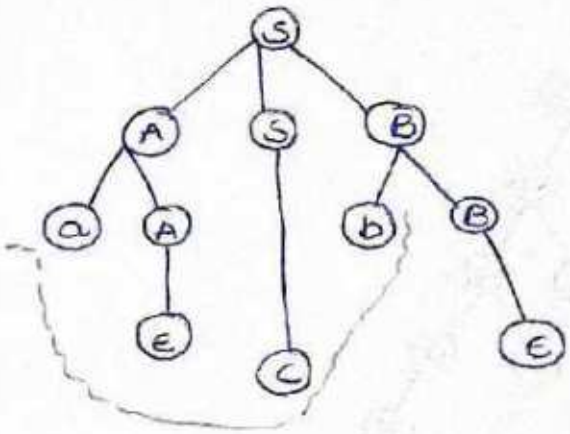
acb



RMD

S	
ASB	S → ASB
ASbB	B → bB
ASbε	B → ε
Acbε	S → c
aAcbε	A → aA
aεcbε	A → ε

acb



(2) consider the following grammar

S → as / ε

The language generated by this grammar

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

i) find the LMD and RMD

ii) Also, prove All the strings generated from this grammar have their LMD and RMD exactly same draw the parse of the same

Let us consider string w = aaa

i) LMD

$S \to aS/\epsilon$

S	
aS	$S \to aS$
aaS	$S \to aS$
aaaS	$S \to aS$
aaaϵ	$S \to \epsilon$

aaa

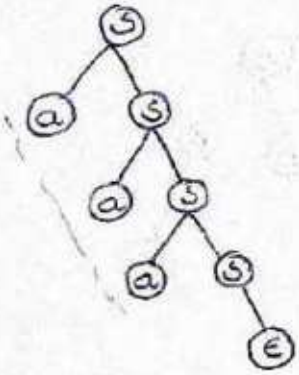
RMD

$S \to aS/\epsilon$

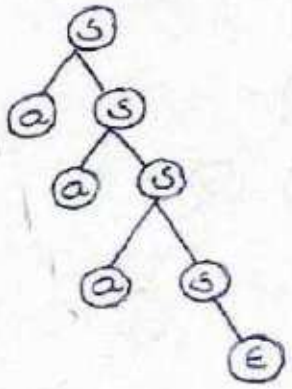
S	
aS	$(S \to aS)$
aaS	$S \to aS$
aaaS	$S \to aS$
aaaϵ	$S \to \epsilon$

aaa

ii)



aaa



aaa

clearly

LMD parse tree = RMD parse tree

past tutorial:

① consider the following grammar

$S \to SaS/b$

it is an ambiguous grammar ? generate the string babab from this grammar to prove your point ?

The given grammar

$S \rightarrow SAS / b$

$w = babab$

LMD

S	
SAS	$S \rightarrow SAS$
SASAS	$S \rightarrow SAS$
bASAS	$S \rightarrow b$
bABAS	$S \rightarrow b$
bABAB	$S \rightarrow b$

RMD

S	
SAS	$(S \rightarrow SAS)$
SASAS	$S \rightarrow SAS$
SASAB	$S \rightarrow b$
SABAB	$S \rightarrow b$
BABAB	$S \rightarrow b$

LMD

S	
SAS	$S \rightarrow SAS$
bAS	$S \rightarrow b$
bASAS	$S \rightarrow SAS$
bABAS	$S \rightarrow b$
bABAB	$S \rightarrow b$

RMD

S	
SAS	$S \rightarrow SAS$
SAB	$S \rightarrow b$
SASAB	$S \rightarrow SAS$
SABAB	$S \rightarrow b$
BABAB	$S \rightarrow b$

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: Simplification of CFG, normal forms

pre tutorial

① Explain simplification of grammar? mention its use? elaborate the steps that are followed in simplification process?

Simplification of grammar

It means reduction of grammar by removing useless symbols each variable and each terminal of G appears in the derivation of some word in L

* There should not be any production as $x \to y$ where x and y are non-terminal
* If ϵ is not in language L there not be production $x \to \epsilon$

Reduced grammar

→ Removal of useless grammar
→ elimination of ϵ production
→ Removal of unit production

Removal of useless grammar:

A variable can be useless if it does not take part in derivation of any string take part in derivation

eg: $T \to aaB / abA / aaT$

$A \to aA$

$B \to ab / d$

$C \to ad$

here $C \to ad$ is useless, $A \to aA$ is useless, $A \to aA$ is useless

To remove $A \to aA$ we will find first all variables which will never lead to a terminal string such as variable 'A' Then we will remove all productions in which the variable 'B' occurs

Elimination of ϵ production

$S \to \epsilon$ are called ϵ productions

step① find out all nullable non-terminal variable which derives ϵ

step② : for each production $A \to a$ construct all production $A \to x$ where x is obtained from a by removing one or more non-terminal from step ①

step③ Now combine the result of step 2 with original production and remove ϵ productions

Eg $S \to xyx$

$x \to 0x \mid \epsilon$

$y \to 1y \mid \epsilon$

Let us take

$S \to xyx$

$\downarrow$

ϵyx

yx

$S \to xyx$

$\downarrow$

$x \epsilon x$

xx

$S \to xyx$

$\downarrow$

$xy\epsilon$

xy

if both x are ϵ

$S \to y$

if y and x are ϵ $S \to x$

Now,

$S \to xy \mid yx \mid xx \mid x \mid y$

consider $x \to 0x$

replace ϵ at RHS for x then

$x \to 0$

$x \to 0x|10$

similarly $y \to 1y|1$

Rewrite the CFG as

$S \to xy|yx|xy|x|y$

$x \to 0x$

$y \to 1y|1$

Removing unit productions

There are productions in which one non-terminal given another non terminal

step① : To remove $x \to y$ add production $x \to a$ to grammar rule whenever $y \to a$ occurs in this grammar

step② : Now, delete $x \to y$ from grammar

step③ : Repeat step① and step② until all unit productions are removed

Eg $S \to 0A|1B|C$

$A \to 0S|00$

$B \to 1|A$

$C \to 01$

$S \to C$ is unit production, by removing $S \to C$ add a rule to S

$S \to 0A|1B|01$

$B \to A$ is also a unit production

$B \to 1|0S|00$

Rewrite CFG

$S \to 0A \mid 1B \mid 01$

$A \to 0S \mid 00$

$B \to 1 \mid 0S \mid 00$

$C \to 01$

(2) find a reduced grammar equivalent to the grammar G_1 having production rules

$S \to AC \mid B$

$A \to a$

$C \to c \mid BC$

$E \to aA \mid \epsilon$

phase 1:-

$T = \{a, c, e\}$

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

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

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

since $W_2 = W_3$ we can derive G_1' as

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

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

phase 2:

$Y_1 = \{S\}$

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

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

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

since $Y_3 = Y_4$ we can derive G_1''

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

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

In tutorial:-

① Remove unit production from following grammar

$S \to AC$

$A \to a$

$C \to x \mid b$

$x \to y$

$y \to z$

$z \to a$

There are 3 unit productions in the grammar

$C \to x$, $x \to y$ and $y \to z$

At first remove $y \to z$

As $z \to a$, we add $y \to a$ and $y \to z$ is removed

So

$S \to AC$, $A \to a$, $C \to x \mid b$, $x \to y$, $y \to a$, $x \to A$

Now remove $x \to y$

As $y \to a$ we add $x \to a$ and $x \to y$ is removed

So

$b \to ac$, $A \to a$, $C \to x \mid b$, $x \to a$, $y \to a$, $z \to a$

Now remove $C \to x$

As $x \to a$ we add $C \to a$ and $C \to y$ is removed

$S \to AC$. $A \to a$, $C \to a \mid b$. $x \to a$, $y \to a$, $z \to a$

Now x, y, z are unreachable hence we can remove those

The final CFG is

$S \to AC$, $A \to a$, $C \to a \mid b$

② A grammar G_1 is defined with rules $S \to xA \mid BB$, $B \to b \mid SB$, $x \to b$, $A \to a$ write productions obtained after normalized GNF of G_1

$S \to xA \mid BB$

$B \to b \mid SB$

$x \to b$

$A \to a$

step① convert grammar into CNF

step② if grammar exists left recursion, eliminate it

step③ convert production rule into GNF form in the grammar

step① and step ② already exist in question so skipping it

$S \to xA \mid BB$

$B \to b \mid xAB \mid BBB$

$A \to a$

$x \to b$

$S \to xA$ and $A \to xAB$ is not in GNF so substitute $x \to b$ in production rule $S \to xA$ and $A \to xAB$ as

$S \to bA \mid BB$

$B \to b \mid bAB \mid BBB$

$A \to a$

$x \to a$

Now remove left recursion ($B \to BBB$)

$S \to bA \mid BB$

$B \to bc \mid bABC$

$C \to BBC \mid \epsilon$

$A \to a$

$x \to b$

Now remove null production $C \to \epsilon$

$S \to bA \mid BB$

$B \to bc \mid bABC \mid b \mid bAB$

$C \to BBC \mid BB$

$A \to a$

$x \to b$

$S \to BB$ is not in GNF, substitute $B \to bc \mid bABC \mid b \mid bAB$ in production rule $S \to BB$

$S \to bA \mid bcB \mid bABCB \mid bA \mid bABB$

$B \to bc \mid bABC \mid b \mid bAB$

$C \to BBC$

$C \to bcB \mid bABCB \mid bB \mid bABB$

$A \to a$

$x \to b$

$C \to BBC$ is not in GNF, substitute $B \to bc \mid bABC \mid b \mid bAB$ in production rule

$C \to BBC$ as

$S \to bA \mid bcB \mid bABCB \mid bB \mid bABB$

$B \to bc \mid bABC \mid b \mid bAB$

$C \to bcB \mid bABCB \mid bB \mid bABB$

$A \to a$

$x \to b$

Hence, this is GNF form for grammar G

post tutorial:

① convert the following

a) CFG into CNF

$S1 \to S$
$S \to ASA \mid aB$

$A \to B \mid S$

$B \to b \mid \epsilon$

Now ϵ is removed

$S1 \to S$

$S \to A.SA \mid aB$

$A \to B \mid S$

$B \to b$

Remove unit productions $S \to S$, $S' \to S$ and $A \to B$, $A \to S$:

after removing $S \to S$:

P: $S' \to S$, $S \to ASA \mid aB$

$A \to B \mid S$

$B \to b$

After removing $S' \to S$:

P $S' \to ASA \mid aB \mid a \mid AS \mid SA$,

$S \to ASA \mid aB \mid a \mid AS \mid SA$,

$A \to B \mid S$, $B \to b$

After removing $A \to B$:

P: $S' \to ASA \mid aB \mid a \mid AS \mid SA$,

$S \to ASA \mid aB \mid a \mid AS \mid SA$,

$A \to b \mid S$, $B \to b$

After removing $A \to S$:

P: $S' \to ASA \mid aB \mid a \mid AS \mid SA$,

$S \to ASA \mid aB \mid a \mid AS \mid SA$,

$A \to b \mid ASA \mid aB \mid a \mid AS \mid SA$,

$B \to b$

Now find out the productions that has more than two variables in RHS

$S' \to ASA$, $S \to ASA$ and $A \to ASA$

P: $S' \to AX \mid aB \mid a \mid AS \mid SA$,

$S \to AX \mid aB \mid a \mid AS \mid SA$,

$A \to b \mid AX \mid aB \mid a \mid AS \mid SA$,

$B \to b$,

$X \to SA$

change the productions

$S' \to aB$, $S \to aB$ and $A \to aB$

finally we get

P: $S' \to AX \mid YB \mid a \mid AS \mid SA$,

$S \to AX \mid YB \mid a \mid AS \mid SA$,

$A \to b \mid AX \mid YB \mid a \mid AS \mid SA$,

$B \to b$,

$X \to SA$,

$Y \to a$

which is the required chomsky normal form for the given CFG

② write the steps for removing null productions and unreachable symbols: example with your own:

1) Remove null productions

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

→ for example consider a cfG with the following production

A → ε the production can be removed

Eg the null production (S → ε) can be removed

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

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

$T = \{a, b\}$

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

$S = S$

2) Remove unreachable symbols:

→ A symbol is considered unreachable if it can never appear in any thing generated by the cfG

→ To identify unreachable symbols, start from start symbol and mark all symbols reachable from it

Eg - the unreachable symbols can be removed new grammar

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

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

$T = \{a, b\}$

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

$S = S$