AUTOMATA THEORY & FORMAL LANGUAGES

Tutorial 1: DFA, NFA

Pre Tutorial

1) Define NFA and DFA normally?

A) 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 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 ~~finite~~ 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 $\delta = Q \times \Sigma \rightarrow 2^Q$)

The two exceptions are

* It contains multiple next states
* It contains ε transitions.

2) Construct a DFA that accepts the language

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

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

A) **Steps for converting NFA to DFA**

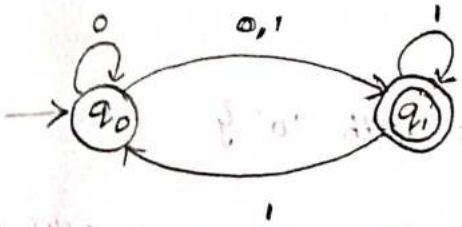
step1: Initially $Q' = \phi$

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'.

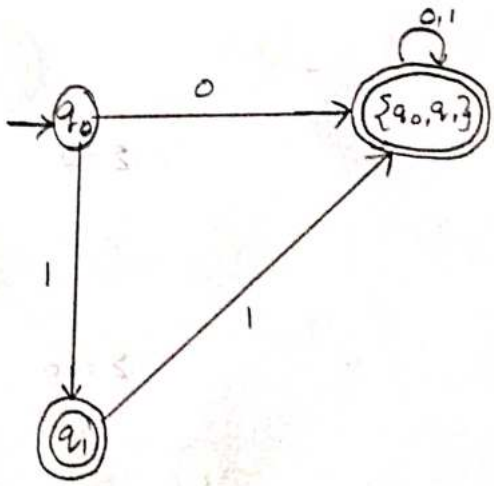
step4: In DFA, the final state will be all states which contain F (Final states of NFA)

Ex:-



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

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



Steps for converting DFA to NFA

step1: 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

(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

Lets call new accepting state q_{n+1}

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

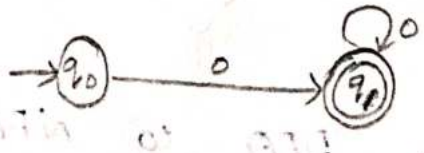
DONE

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

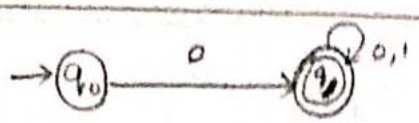
$L = \{0, 00, 01, 000, 001, 010, 0010, 0101, \ldots\}$

$\Sigma = 0$

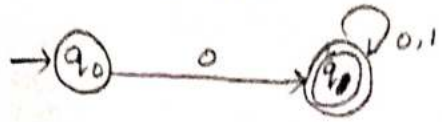
$\Sigma = 00$



$\Sigma = 01$

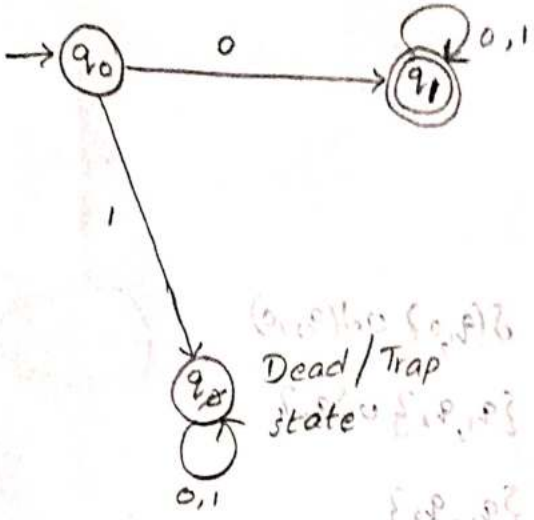


$\Sigma = 000$



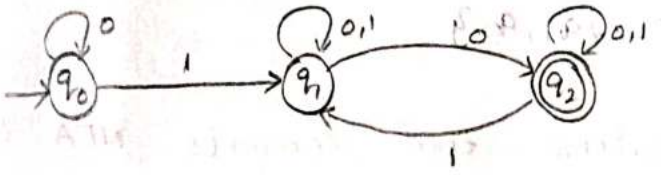
$\Sigma = 001$ (we cannot draw)

So,



In-tutorial

1) Convert the following NFA to DFA

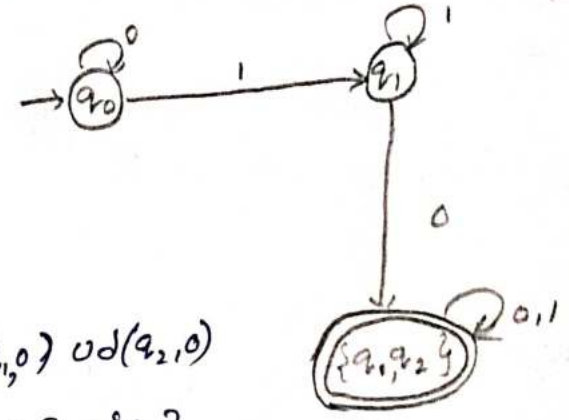


A)

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

Now,

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



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

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

$= \{q_1, q_2\}$

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

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

$= \{q_1, q_2\}$

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

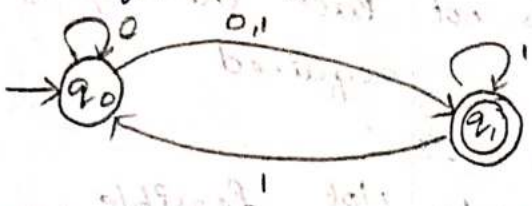
A) algorithm

step1: 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 states which contain F (final state of NFA)



	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_1\}$
$*q_1$	$\emptyset$	$\{q_0, q_1\}$

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

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

$= \delta(q_0, 0) \cup \delta(q_1, 0)$

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

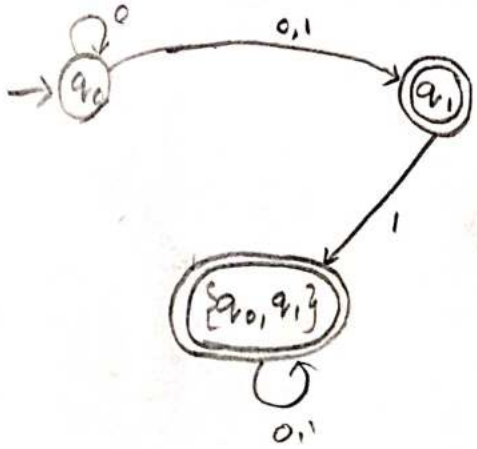
$= \{q_0, q_1\}$

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

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

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

$= \{q_0, q_1\}$



Post - Tutorial

1) Differentiate NFA and DFA

A)

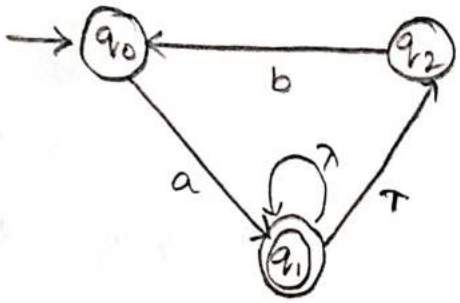
	NFA	DFA
1	$(Q, \Sigma, \delta, q_0, F)$ $\delta = Q \times \Sigma \rightarrow 2^Q$	$(Q, \Sigma, \delta, q_0, F)$ $\delta = Q \times \Sigma \rightarrow Q$
2	$Q = \{q_0, q_1\}$, $\Sigma = \{0, 1\}$ $q_0 \rightarrow 0$ $q_1 \rightarrow 1$	$Q = \{\{q_0, q_1\}$, $\Sigma = \{0, 1\}$ $q_0 \rightarrow 0, 1$ $q_1 \rightarrow 0, 1$

3	Transition may leads to multiple states	Transition leads to unique state
4	Back tracking is not required	Back tracking is required
5	Practical implementation of DFA is feasible	Not feasible, convert NFA to DFA

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

A)

3) Convert the following ϵ - NFA to DFA



A)

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

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

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

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

$= \emptyset \cup \emptyset = \emptyset$

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

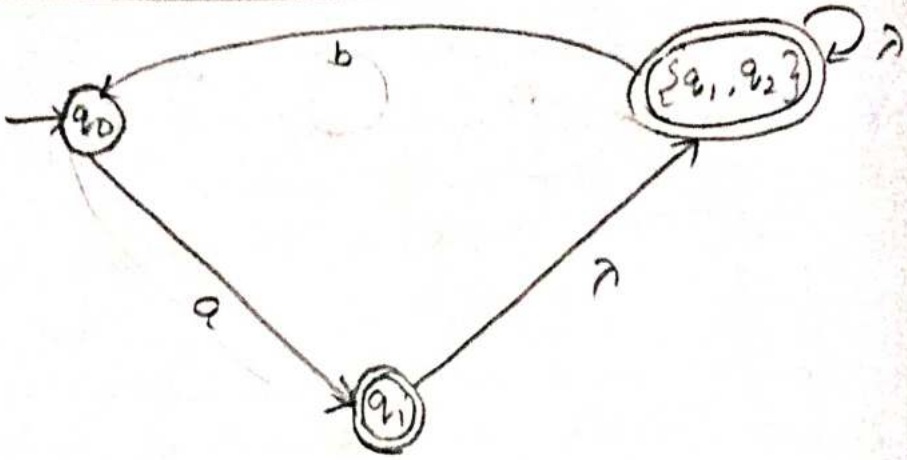
$= \delta(q_1, b) \cup \delta(q_2, b)$

$= \emptyset \cup q_0 = q_0$

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

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

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


b
λ
{q1, q2}
q0
a
λ
q1

Tutorial 2: Regular Expression

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:

$\phi + r = r$

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

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

$r + r = r$

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

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

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

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

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

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

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

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

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

$\epsilon^* = \epsilon$

$\phi^* = \epsilon$

1) Consider language L given by regular expression $(a+b)^* b(a+b)$ over the alphabet $\{a,b\}$

Design a DFA that accepts L.

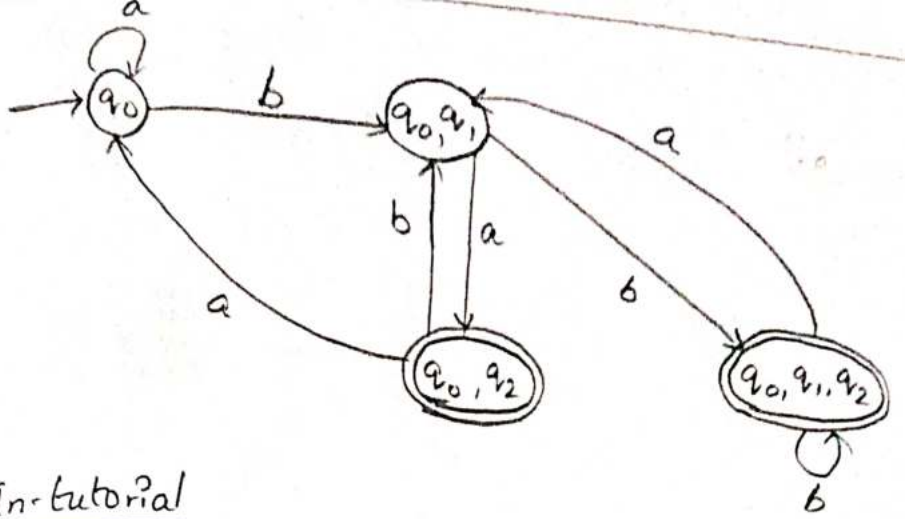
) * convert RE into NFA and the find out DFA from NFA

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

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

DFA table from NFA

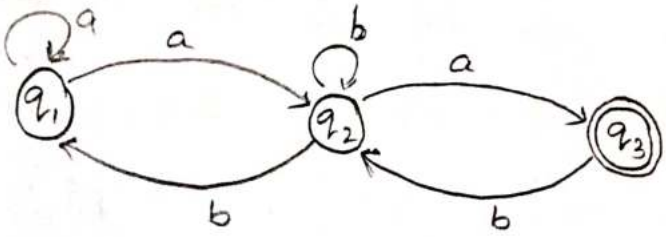
	a	b
$\to q_0$	q_0	$\{q_0, q_1\}$
$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$
$*\{q_0, q_2\}$	q_0	$\{q_0, q_1\}$
$*\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$



In-tutorial

1) Provide algorithm to convert NFA into RE

Create a RE for following NFA



A)

$$q_1 = q_1 a + q_2 b$$

$$q_2 = q_2 b + q_1 a$$

$$q_3 = q_2 a$$

Now,

$$q_1 = q_1 a + q_2 b + \epsilon \quad \rightarrow (1)$$

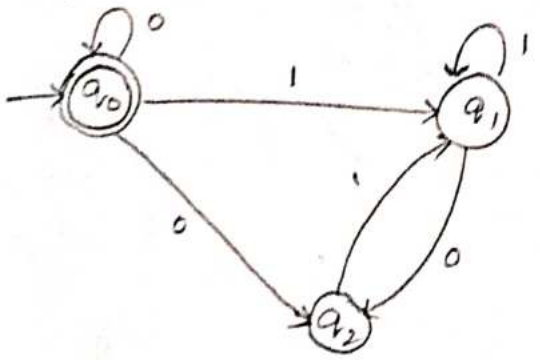
$$q_2 = q_1 a + q_2 b \quad \rightarrow (2)$$

$$q_3 = q_2 a \quad \rightarrow (3)$$

Substitute (2) in (1)

$$q_1 = q_2 + \epsilon$$

3) Explain Arden's theorem to convert FA to RE. Use algorithm to convert following DFA to RE



A) Arden's 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 i.e., $\boxed{R = QP^*}$

Steps:-

1) For each state $q_1, q_2, q_3, \ldots$ 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 for example

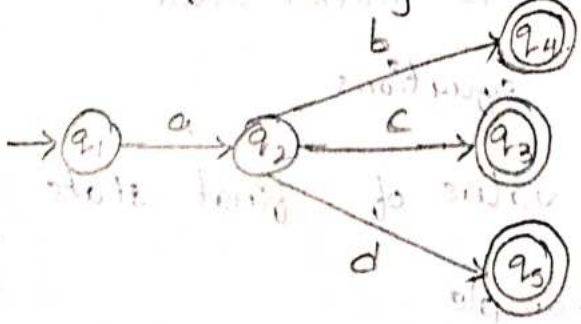
$$q_0 = q_0 0 + \epsilon \rightarrow (1)$$

$$q_1 = q_0 1 + q_1 1 \rightarrow (2)$$

$$q_2 = q_0 0 + q_1 0 \rightarrow (3)$$

Post Tutorial

1) Find RE for following DFA



A)

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 $x = b^{10+12k}, k \geq 0\}$

$L = \{a^2, a^8, a^{14}, a^{20} \ldots\ldots\} \cup \{b^{10}, b^{22}, b^{34}, \ldots\}$

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 (i.e., $\omega = b^{11}$) will repeat states

length of pumping lema is 24

Tutorial - 3

Context Free Grammar, left most and Right most deviations.

Pre Tutorial

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

A) Context Free Grammar:

It is a formal grammar which is used to generate all the possible patterns of strings in a given formal language.

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

G → grammar, which consists of set of production rules

T → lower case letter, final set of terminal symbols

V → capital letter, final set of non-terminal symbols

P → set of production rules, used for replacing non terminal symbols in (with ϵ) a string with other terminals

S → start symbol used to derive string

Ex:- Construct CFG of language having any 2 number of a's over set $\Sigma = \{a, b\}$

Sol:- $RE = (a^* + b)^* \, a \, (a+b)^* \, a \, (a+b)^*$

$S \rightarrow TaTaT$

$T \rightarrow aT / bT / \epsilon$

Now, let's take $V = \{S, T\}$

$T = \{a, b\}$

$S = \{S\}$

$S \Rightarrow TaTaT$

$\downarrow$

$bTaTaT$

$\downarrow$

$b\epsilon aTaT$

$baTaT$

$\downarrow$

$babTaT$

$\downarrow$

$babaT$

$\downarrow$

$bababT$

$\downarrow$

$babab\epsilon$

$bababa$

In Tutorial

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

A) $L = \{ \omega c \omega^R / \omega \in (a,b)^* \}$

$S \to a\,Sa$ rule 1

$S \to b\,Sb$ rule 2

$S \to c$ rule 3

$S =$ abbcbba

$S \to aSa$

$S \to abSba$ from rule 2

$S \to abbSbba$ from rule 2

$S \to abbcbba$ from rule 3

2) Derive string "aabbabba" for left most derivation and right most derivation using a CFG

A) $S \to aB \mid bA$

$A \to a \mid aS \mid bAA$

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

Leftmost derivation $\rightleftharpoons$ Right most Derivation

S
↓
aB
↓
aaBB
↓
aaBbS
↓
aaBbbA
↓

~~aaB~~

S
↓
aB
↓
aaBB
↓
aabB
↓

aa Bbba
↓
aabSbba
↓
aabbAbba
↓
aabbabba

aabbS
↓
aabbaB
↓
aabbabS
↓
aabbabbA
↓
aabbabba

Post Tutorial

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

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

let $i=1, k=1$
$j=3$

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

$\Rightarrow 0\ 1\ 1\ 1\ 0$

$S \to XYZ$

$X \to 0X1 \mid 01$

$Y \to 1Y \mid 1$

$Z \to 1Z0 \mid 10$

(or)

$X \to 0X1 \mid \epsilon$

$Z \to 1Z0 \mid \epsilon$

also we can use it as it is

$S \to XYZ$
↓
$0XYZ$
↓
$00111YZ$
↓
$001111Z$
↓
00111110

00111110
$i=2\ \ j=5\ \ k=1$

$j > i+k$

Tutorial 4 : Parse Tree, Ambiguity In CFG

Pre Tutorial

1) Differentiate ambiguous and unambiguous grammar?

A)

SNO	Ambiguous Grammar	Unambiguous Grammar
1	It generates more than one-parse tree	It generates exactly one parse-tree
2	The leftmost & Right most derivation represents different parse tree	The leftmost & Right most derivations represents same parse tree
3	Contains a smaller number of non-terminals	Contains a greater number of non-terminals
4	length is less	length is large
5	Example $S \rightarrow S+S \mid S \times S \mid id$	Example $S \rightarrow S+\epsilon \mid \epsilon$ $\epsilon \rightarrow \epsilon \times F \mid F$ $F \rightarrow id$

Intutorial

1) Consider following grammar?

$S \rightarrow A\,S\,B \mid C$

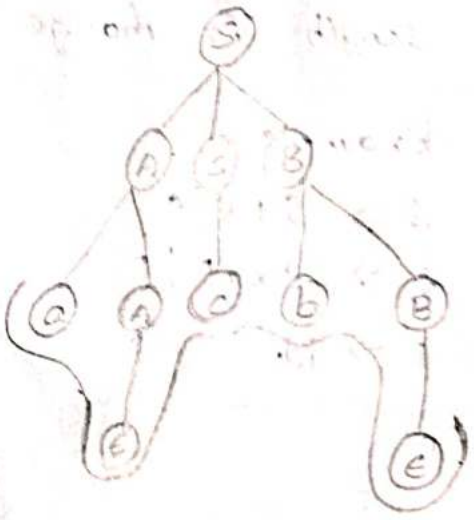
$A \rightarrow \epsilon \mid aA$

$B \rightarrow \epsilon \mid bB$

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

A)

leftmost derivation	Right most derivation
S	S
↓	↓
ASB	ASB
↓	↓
aASB	ASbB
↓	↓
aϵSB	ASbϵ
↓	↓
aSB	Asb
↓	↓
acB	Acb
↓	↓
acbB	aAcb
↓	↓
acbϵ	aϵcb
↓	↓
→ acb	acb



1) Consider the following grammar

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

The language generated by this grammar

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

i) Find leftmost derivation and Right most derivation

ii) Also, prove all the strings generated from this grammar have their leftmost and rightmost derivation exactly same. Draw the parse tree for the same

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

$L = \{\epsilon, a, aa, aaa, aaaa, \ldots\}$

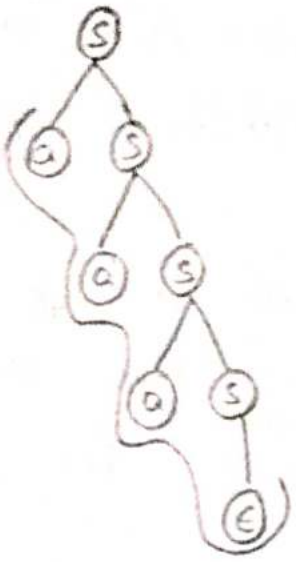
$S \rightarrow aS/\epsilon$

Leftmost Derivation	Rightmost Derivation
$S \rightarrow aS$ $\downarrow$ aaS $\downarrow$ $aaaS$ $\downarrow$ $aaa\epsilon$ aaa	$S \rightarrow aS$ $\downarrow$ aaS $\downarrow$ $aaaS$ $\downarrow$ $aaa\epsilon$ aaa

Leftmost derivation = Rightmost derivation

Similar is the case for all other strings

Parse Tree is



- Post - Tutorial

1) Consider the following grammar -

$$S \rightarrow SaS \mid b$$

Is it ambiguous grammar? Generate the string "bábáb" from this grammar to prove your point.

A)

TUTORIAL-5 : Simplification of CFG, Normal forms

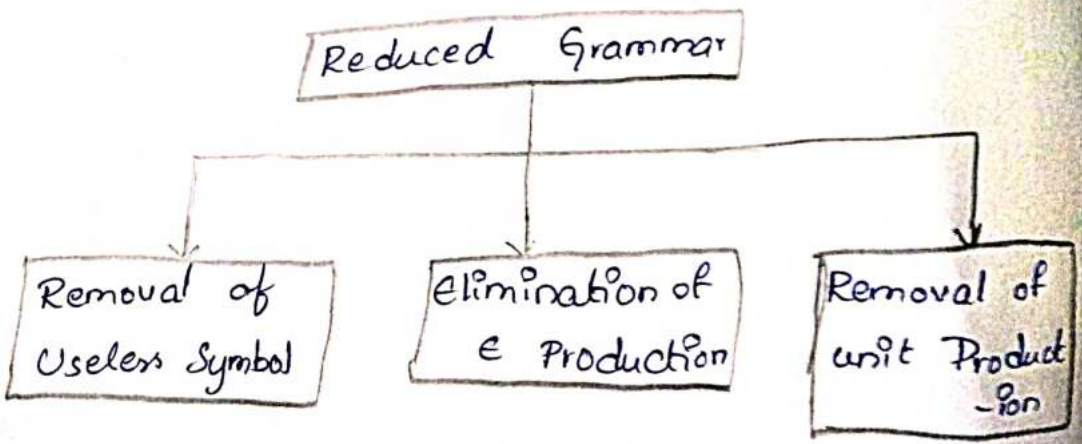
Pre Tutorial

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

A) **Simplication 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 there need not be production $X \to \epsilon$



Removal of Useless Symbol:

A variable can be useless, if it does not

take part in derivation of any string.

Eg: $T \rightarrow aaB \mid abA \mid aaT$

$A \rightarrow aA$

$B \rightarrow ab \mid d$

$C \rightarrow ad$

Here $C \rightarrow ad$ is usless, $A \rightarrow aA$ is useless

To remove $A \rightarrow aA$, we will first find 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:

$S \rightarrow \epsilon$ are called ϵ productions

step1: Find out all nullable non-terminal variable which derives ϵ

step 2: For each production $A \rightarrow a$ construct all production $A \rightarrow x$, where x is obtained from a by removing one or more non-terminal from step 1

step 3: Now combine the result of step 2 with original production and remove ϵ productions

Ex:- $S \rightarrow XYX$

$X \rightarrow 0X \mid \epsilon$

$Y \rightarrow 1Y \mid \epsilon$

$S \Rightarrow xyx$

$\downarrow$

ϵyx

yx

$\downarrow$

ϵx

$S \Rightarrow xyx$

$\downarrow$

$xy\epsilon$

xy

Sol:

Let us take

$S \rightarrow xyx$

$\downarrow$

ϵyx

yx

$S \rightarrow xyx$

$\downarrow$

$x\epsilon x$

xx

If both x are ϵ

$S \rightarrow y$

$S \rightarrow xyx$

$\downarrow$

$xy\epsilon$

xy

If y and x are ϵ

$S \rightarrow x$

Now,

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

Consider $x \rightarrow 0x$

replace ϵ at for RHS x then

$x \rightarrow 0$

$x \rightarrow 0x \mid 0$

Similarly $y \rightarrow 1y \mid 1$

Rewrite the CFG as

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

$X \rightarrow 0x$

$Y \rightarrow 1y \mid 1$

Removing unit Productions:

These are productions in which one non-terminal gives another non-terminal.

Step 1 : To remove $X \rightarrow Y$ add production $X \rightarrow a$ to grammar rule whenever $Y \rightarrow a$ occurs in this grammar

step 2 : Now delete $X \rightarrow Y$ from grammar

step 3 : Repeat step 1 and step 2 until all unit productions are removed

Ex : $S \rightarrow 0A \mid 1B \mid C$

$A \rightarrow 0S \mid 00$

$B \rightarrow 1 \mid A$

$C \rightarrow 01$

Sol :

$S \rightarrow C$ is unit production, By removing $S \rightarrow C$, add a rules to S

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

$B \rightarrow A$ is also a unit production

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

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, having production rules

A) $S \to AC \mid B$,

$A \to a$

$C \to c \mid BC$

$E \to aA \mid e$

A) 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' as-

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

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

Phase 2 –

$Y_1 = \{S\}$

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

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

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

Since $Y_3 = Y_4$, we can derive G"

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

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

In Tutorial

1) Remove unit production from following grammar

$S \to AC$

$A \to a$

$C \to X \mid b$

$X \to Y$

$Y \to Z$

$Z \to a$

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 $Y \to a$ & $Y \to Z$ is removed

So,

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

Now remove $X \to Y$

As $Y \to a$ we add $X \to a$ and $X \to Y$ is removed

So,

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

Now remove $C \to \epsilon$

As $X \to a$ we add $C \to a$ and $C \to \epsilon$ is removed

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

Now, X, Y, Z are unreachable, hence we can remove those

The final CFG is unit production free

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

3) A grammar G is defined with rules $S \to XA | BB$, $B \to b | SB$, $X \to b$, $A \to a$. Write productions obtained after normalized GNF of G.

A) $S \rightarrow XA \mid BB$

$B \rightarrow b \mid SB$

$X \rightarrow b$

$A \rightarrow a$

step1: Convert grammar into CNF

step2: If grammar exists left recursion, eliminate it

step3: convert production rule into GNF form in the grammar

Sol: step1 and step2 already exist in question, so skipping it.

step3:

$B \rightarrow SB$ is not in GNF, so we substitute $S \rightarrow XA \mid BB$ in production rule $B \rightarrow SB$ as

$S \rightarrow XA \mid BB$

$B \rightarrow b \mid XAB \mid BBB$

$A \rightarrow a$

$X \rightarrow b$

$S \rightarrow XA$ and $A \rightarrow XAB$ is not in GNF

so substitute $X \rightarrow b$ in production rule $S \rightarrow XA$ and $A \rightarrow XBAB$ as:

$S \rightarrow bA \mid BB$

$B \rightarrow b \mid bAB \mid BBB$

$A \rightarrow a$

$X \rightarrow a$

Now, remove left recursion ($B \rightarrow BBB$)

$S \rightarrow bA \mid BB$

$B \rightarrow bC \mid bABC$

$C \rightarrow BBC \mid \epsilon$

$A \rightarrow a$

$X \rightarrow b$

Now remove null production $C \rightarrow \epsilon$

$S \rightarrow bA \mid BB$

$B \rightarrow bC \mid bABC \mid b \mid bAB$

$C \rightarrow BBC \mid BB$

$A \rightarrow a$

$X \rightarrow b$

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

$S \rightarrow bA \mid bCB \mid bABCB \mid bA \mid bABB$

$B \rightarrow bC \mid bABC \mid b \mid bAB$

$C \rightarrow BBC$

$C \rightarrow bCB \mid bABCB \mid bB \mid bABB$

$A \rightarrow a$

$X \rightarrow b$

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

$C \rightarrow BBC$ as:

$S \rightarrow bA \mid bCB \mid bABCB \mid bB \mid bABB$

$B \rightarrow bC \mid bABC \mid b \mid bAB$

$C \rightarrow bCB \mid bABCB \mid bB \mid bABB$

$A \rightarrow a$

$X \rightarrow b$

Hence, this is GNF form for grammar G.

Post Tutorial

1) Convert the following CFG into CNF

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

A) $S1 \rightarrow S$

$S \rightarrow ASA \mid aB$

$A \rightarrow B \mid S$

$B \rightarrow b \mid \epsilon$

Now, ϵ is removed

$S1 \to S$

$S \to ASA \mid aB$

$A \to B \mid S$

$B \to b$

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

A) Same answer as 1st Question of Pre Tutorial