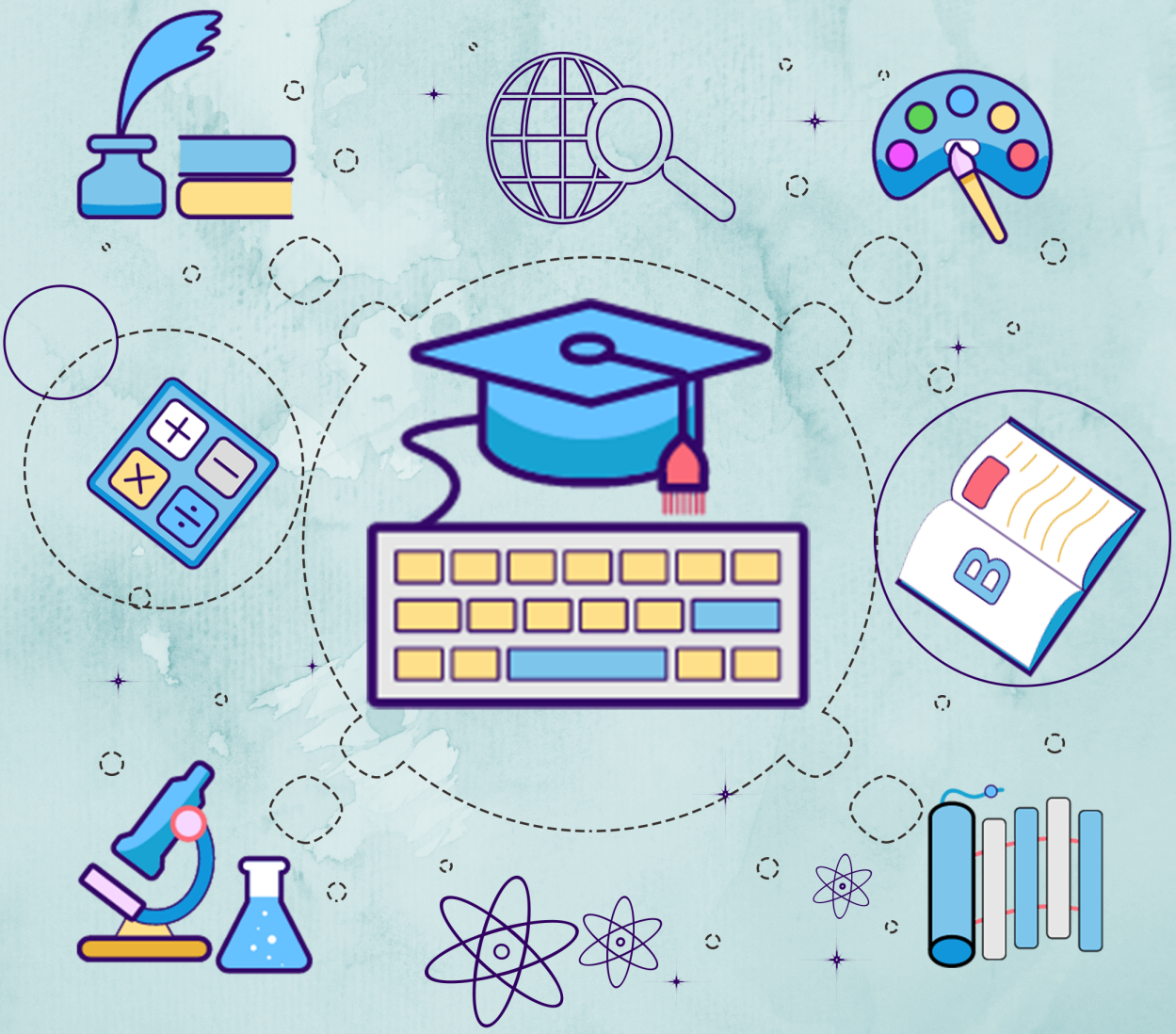
APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

KeralaNotes



SYLLABUS | STUDY MATERIALS | TEXTBOOK

PDF | SOLVED QUESTION PAPERS

www.keralanotes.com

KeralaNotes

KTU STUDY MATERIALS

FORMAL LANGUAGES AND AUTOMATA THEORY

CST 301

Module 3

Related Link :

- KTU S5 STUDY MATERIALS
- KTU S5 NOTES
- KTU S5 SYLLABUS
- KTU S5 TEXTBOOK PDF
- KTU S5 PREVIOUS YEAR SOLVED QUESTION PAPER

FORMAL LANGUAGES AND AUTOMATA THEORY

Module 3

Myhill-Nerode Relations and Context-Free Grammars:

Myhill-Nerode Relations (MNR)- MNR for regular languages, Myhill-Nerode Theorem (MNT) (No proof required), Applications of MNT.

Context-Free Grammar (CFG)- CFG representation of Context-Free Languages (proof of correctness is required), derivation trees and ambiguity, Normal forms for CFGs.

Prepared by:

Karishma PK

Assistant professor

Computer Science Department EKCTC

MODULE - III

Myhill - Nerode Relations (MNR)

Let $L \subseteq \Sigma^*$ be a regular language and $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA for L with no inaccessible states.

The automation M induces an equivalence relation R on Σ^* defined by.

$$xRy \Rightarrow \hat{\delta}(q_0, x) = \hat{\delta}(q_0, y)$$

This relation R is an equivalence relation as it is reflexive, symmetric and transitive.

(i) xRx – Reflexive

(ii) $xRy \Rightarrow yRx$ Symmetric

(iii) $xRy, yRz \Rightarrow xRz$ – Transitive

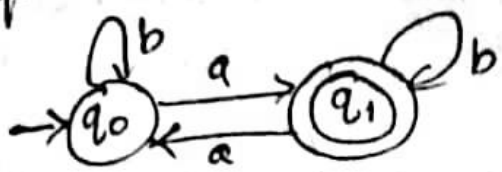
The equivalence relation R partitions the underlying set Σ^* into classes, which are called equivalence classes.

The no: of equivalence classes is the index of the equivalence relation.

In the case of Myhill - Nerode Relation, the index is atmost the no: of states in the FA.

Eg:-

Consider the given DFA, M over $\Sigma = \{a, b\}$ and equivalence relation $xRy \Rightarrow \hat{\delta}(q_0, x) \Rightarrow \hat{\delta}(q_0, y)$



This DFA has 2 states q_0, & q_f. So the equivalence relation R on Σ^* partitions the Σ^* into 2 equivalence classes, c_1 & c_2 as follows:

c_1 = set of all strings which satisfies

$\hat{\delta}(q_0, x) = \hat{\delta}(q_0, y) = q_0$

$= \{b, bb, aba, abba, aa, bbb \cdots\}$

&

c_2 = set of all strings which satisfies

$\hat{\delta}(q_0, x) = \hat{\delta}(q_0, y) = q_f$

$\{ba, bba, babb, bbabb \cdots\}$

Hence the index of the equivalence relation is 2.

Properties of Myhill Nerode Relations (MNR)

Reflexive

$xR_m x \Rightarrow \hat{\delta}(q_0, x) = \hat{\delta}(q_0, x)$

Symmetric

If $\hat{\delta}(q_0, x) = \hat{\delta}(q_0, y)$, then $\hat{\delta}(q_0, y) = \hat{\delta}(q_0, x)$

$\Rightarrow yR_m x$

Properties of Myhill-Nerode Relations (MNR) continue

In addition to the properties reflexive, symmetric and transitive the Myhill Nerode relation satisfies other properties.

(i) It is a right congruence / right invariant

For any $x, y \in \Sigma^*$ and $a \in \Sigma$,

$xRy \Rightarrow xaRya$

To prove this, Assume that xRy, then

$$\hat{\delta}(q_0, xa) = \delta(\hat{\delta}(q_0, x), a)$$
$$= \delta(\hat{\delta}(q_0, y), a)$$
$$= \hat{\delta}(q_0, ya)$$

(ii) It refines L from Σ^*

For any $x, y \in \Sigma^*$

$xRy \Rightarrow (x \in L \Leftrightarrow y \in L)$

ie, If x is accepted by M, y is also accepted. If x is rejected by M, y is also rejected.

Transitive

If $\hat{\delta}(q_0, x) = \hat{\delta}(q_0, y)$ & $\hat{\delta}(q_0, y) = \hat{\delta}(q_0, z)$ then $\hat{\delta}(q_0, x) = \hat{\delta}(q_0, z)$

So, $xRy, yRz \Rightarrow xRz$ ∴ Transitive.

(iii) It is of finite order

The Myhill Nerode Relation has only finitely equivalence classes. Because there is exactly one equivalence class

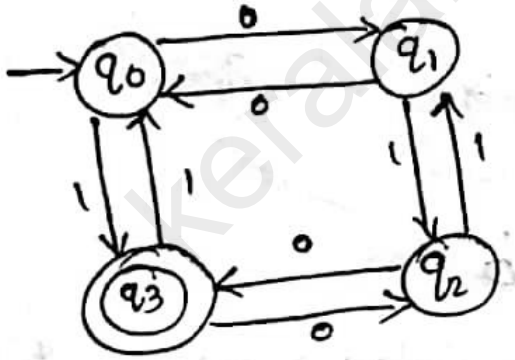
$$\{x \in \Sigma^* \mid \hat{\delta}(q_0, x) = q\}$$

corresponding to each state q of M.

CONVERSION OF DFA TO MNR

Q) Show the equivalence classes of the canonical Myhill-nerode relation for the language of binary strings with odd no: of 1's & even no: of 0's?

Soln:- DFA for the language L is equal to the set of all strings with odd no: of 1's & even no: of 0's.



This DFA has 4 states. So MNR partitions the Σ^* into 4 equivalence classes.

$$c_1 = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) = q_0\}$$

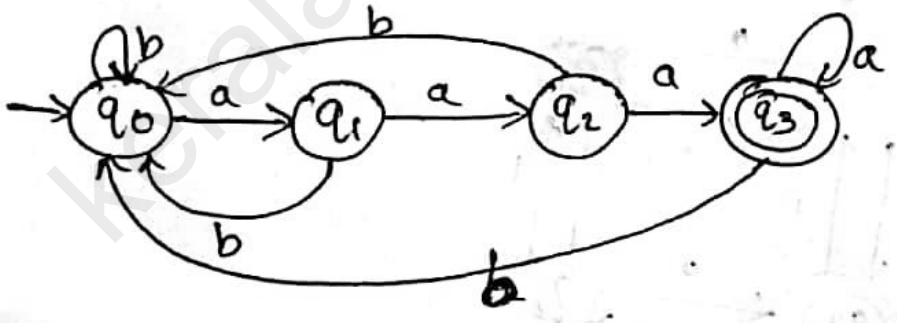
$$= \{11, 00, 0110, 1001, 1010, 0101 \ldots\}$$

$$C_2 = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) = q_1\}$$
$$= \{0, 000, 011, 01001, 101, \ldots\}$$
$$C_3 = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) = q_2\}$$
$$= \{10, 01, 1000, 0111, 0010, 1101 \ldots\}$$
$$C_4 = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) = q_3\}$$
$$= \{1, 111, 100, 10110, 010, 001 \ldots\}$$

(9) write the equivalence classes of the canonical Myhill Nerode Relation:

Soln:-

$L = \{x \in \Sigma^* \mid x$ ends with 3 consecutive a's



DFA has 4 states. so order of MNR is 4

MNR partitions the Σ^* into 4 equivalence classes.

$$C_1 = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) = q_0\}$$
$$\{b, ab, aab, abb \ldots\}$$

$C_2 = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) = q_1\}$

$= \{a, aba, ba, aaba, aaaba \cdots\}$

$C_3 = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) = q_2\}$

$= \{aa, baa, abaa, aabaa, \cdots\}$

$C_4 = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) = q_3\}$

$= \{aaa, baaa, abaaa \cdots\}$

Table Filling Method or Myhill-Nerode Theorem (MNT)

Myhill-Nerode Theorem states that the following 3 statements are equivalent.

(i) The set $L \subseteq \Sigma^*$ is accepted by some FA

(ii) L is the union of some of the equivalence classes of a right-invariant equivalence relations of Finite index

(iii) Let equivalence relation R_L be defined by $(x, y \in \Sigma^*)$ $x R_L y$ iff for all z in Σ^*, xz is in L exactly when yz is in L. Then R_L is of finite index.

- It implies that there is a unique minimal DFA with minimum no. of states.

Minimization of DFA using Table filling Method or Myhill-Nerode Theorem (Appln of MNT)

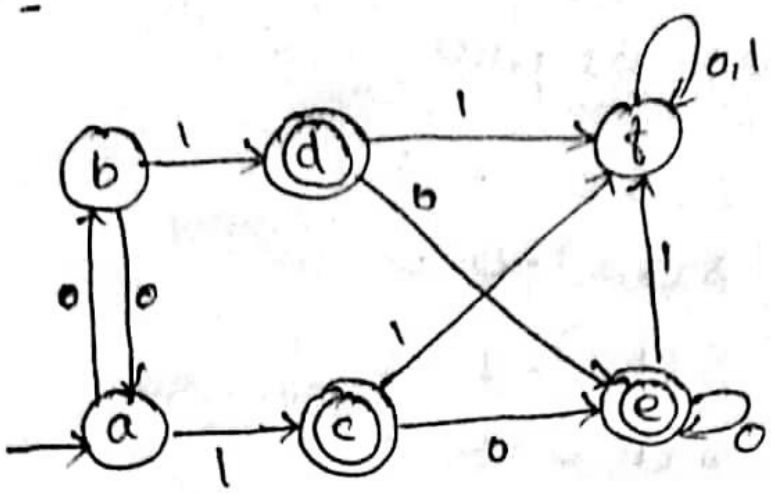
- DFA minimization stands for converting a given DFA to its equivalent DFA with minimum no: of states.

Input – DFA
output – Minimized DFA

STEPS

1) Draw a table for all pairs of states (P, Q)
2) Mark all pairs where $P \in F$ and $Q \notin F$.
3) If there are many unmarked pairs (P, Q) such that $[\delta(P,x), \delta(Q,x)]$ is marked, then mark [P, Q] (where x is an input symbol). Repeat this until no more markings can be made.
4) Combine all the unmarked pairs and make them a single state in the minimized DFA.

Eg:-



50
(1½)

STEP 1 :- Draw a table for all pairs of states (P,Q)

	A	B	C	D	E	F
A						
B						
C						
D						
E						
F						

: Take lower portion.

STEP 2: Mark all pairs where P∈F and Q∉F (one state should be final & other should be non final state)

B					
C	X	X			
D	X	X			
E	X	X			
F			X	X	X
	A	B	C	D	E

STEP 3:

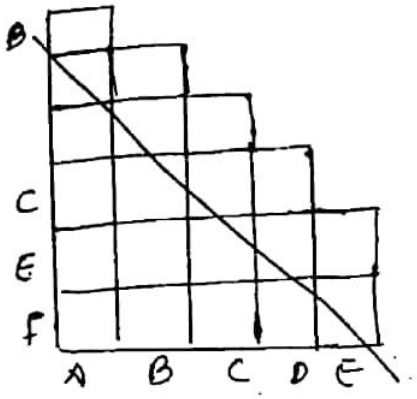
Take unmarked pairs:

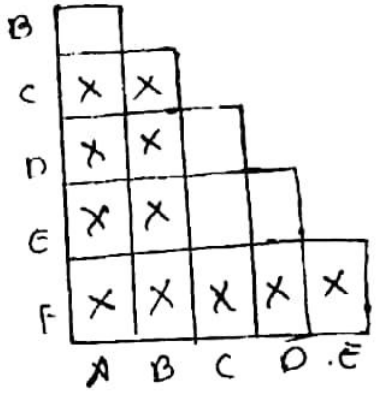
(BA) - $\delta(B,0) - A$, $\delta(A,0) - B$ } unmarked
$\delta(B,1) - D$, $\delta(A,1) - C$ } unmarked

(DC) → $\delta(D,0) - E$, $\delta(E,0) - E$ } no such column
$\delta(D,1) - F$, $\delta(E,1) - F$ } no such column

(EC) → $\delta(E,0) - E$, $\delta(E,0) - E$ } no such column
$\delta(E,1) - F$, $\delta(C,1) - F$ } no such column

(FA) → $\delta(F,0) - F$, $\delta(A,0) - B$ } unmarked
$\delta(F,1) - F$, $\delta(A,1) - C$ } marked, so we should mark FA





(F, B) → $\delta(F,0) - F$ } marked. so cell should mark FB also.
$\delta(B,0) - A$

$\delta(F,1) - F$ } already marked.
$\delta(B,1) - d$

STEP 4:
Combine the unmarked pairs
⇒ (A,B), (D,C), (E,C), (E,D)

Application of Myhill Nerode Theorem (MNT)

(1) It is used to prove that a certain language is regular or not.

(2) It is used to find the minimal number of states in a DFA. (Minimization of DFA)

Q) To prove that $L = \{a^n b^n \mid n \geq 0\}$ is not regular ? (Eg: for appln of MNT)

We can show that L has infinitely many equivalence classes by showing that a^k & a^i are distinguishable by L whenever $k \neq i$.

Let $x = a^k$ i.e, $a^k \neq a^i$

$y = a^i$

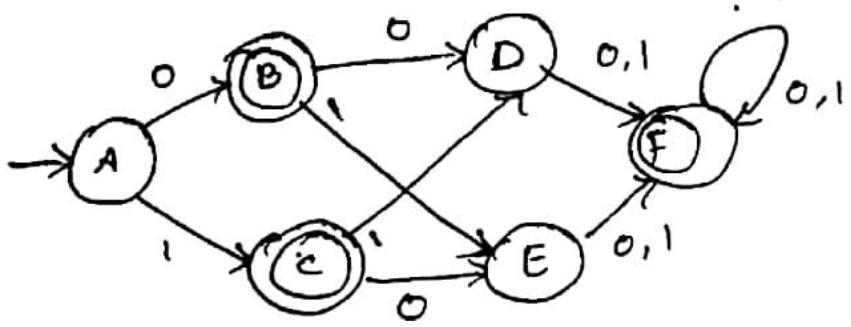
$z = b^k$

Then $xz = a^k b^k$ is in the language but $yz = a^i b^k$ is not in the language.

Thus, each equivalence class of L can contain atmost one string of the form a^i so there must be infinitely many equivalence classes.

i.e, L is not regular

Q) consider the following DFA & construct a minimal DFA by using MNOT.?



Soln:-

B	X			
C	X			
D		X	X	
E		X	X	
F	X			X X
	A	B	C	D E

$\delta(C,0) = E$
$\delta(B,0) = D$
$\delta(C,1) = D$
$\delta(B,1) = E$

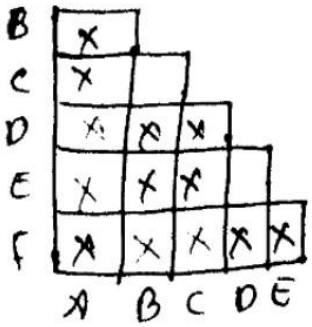
$\delta(D,0) = F$
$\delta(A,0) = B$
$\delta(D,1) = F$
$\delta(A,1) = C$

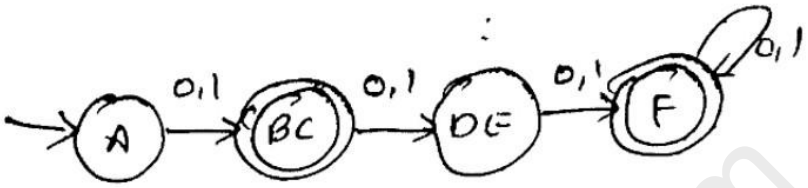
$\delta(E,0) = F$
$\delta(A,0) = B$
$\delta(E,1) = F$
$\delta(A,1) = C$

$\delta(E,0) = F$
$\delta(D,0) = F$
$\delta(E,1) = F$
$\delta(D,1) = F$

$\delta(F,0) = F$
$\delta(B,0) = D$
$\delta(F,1) = F$
$\delta(B,1) = E$
} marked

$\delta(F,0) = F$
$\delta(C,0) = E$
$\delta(F,1) = F$
$\delta(C,1) = D$
} marked





CONTEXT FREE GRAMMAR

A grammar is said to be context-free if every rule has a single non terminal on the LHS.

Formal definition

A Context-Free Grammar (CFG) is a four tuple grammar.

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

V → set of non terminals or variables

T → set of terminals

S → start symbol

P → set of production rules and it is of the form $A \rightarrow \beta$, where $A \in V$ & $\beta \in (V \cup T)^*$, $|A|$ is equal to 1

Eg:- Let $G = (\{S\}, \{0,1\}, P, S)$

$S \rightarrow 0S \mid 1S$

(a) Design a context free grammar for $L = \{a^n b^n / n>0\}$

Soln:-

$T = \{a, b\}$

$V = \{S\}$

$S = S$

$P = \{S \rightarrow aSb, S \rightarrow ab\}$

(b) $L = \{a^n b^n \mid n \geq 0\}$

$T = \{a, b\}$, $V = \{S\}$, $S = S$

$P = \{S \rightarrow \varepsilon, S \rightarrow aSb\}$

(c) $L = \{a^n b^{n+1} \mid n \geq 0\}$

$S \rightarrow b$

$S \rightarrow aSb$

(d) $L = \{a^n b^n \mid n \geq 1\}$

$\therefore P \Rightarrow$

$S \rightarrow ab$

$S \rightarrow aSb$

8) CFG which generates strength of balanced parenthesis?

$G = (\{S\}, \{(,)\}, P, S)$

P:

$S \to SS \mid (S) \mid \varepsilon$

Eg:- $S \to SS$
$\to (S)S$
$\to (S)\cdot$
$\to ()$

Eg:- $S \to SS$
$\to (S)(S)$
$\to ()()$

Eg:- $S \to SS$
$\to (S)S$
$\to ((S))S$
$\to (())$

9) CFG which generates palindromes for binary numbers?

$G = (\{S\}, \{0,1\}, P, S)$

P: $S \to 0S0 \mid 1S1 \mid 0 \mid 1 \mid \varepsilon$

Eg:- $S \to 0S0$
$\to 01S10$
$\to 0110$

CONTEXT FREE LANGUAGE (CFL)

- The language generated using Context Free Grammar is called as Context Free language.

Properties

- The class CFL are closed under union $(L_1 \cup L_2)^{\checkmark}$
- The CFL are closed under concatenation $(L_1 \cdot L_2)^{\checkmark}$
- The Context Free languages are closed under Kleen closure $(L_1^* \& L_2^*)^{\checkmark}$
- The CFL are not closed under intersection and complement. $(L_1 \cap L_2)^{\times}$, $(L_1' \& L_2')^{\times}$
- The family of regular language is a proper subset of the family of CFL.
- Each CFL is accepted by a pushdown automata (PDA)

DERIVATION

- Derivation is a sequence of production rules. It is used to get the input string through these production rules.
- At the time of derivation, we have to take a decisions. There are as follows.
  1) we have to decide the non-terminal which is to be replaced.

2) we have to decide the production rule by which the non terminal will be replaced.

- Two types of derivations
  1) Left Most Derivation
  2) Right-Most-Derivation

Leftmost Derivation

Find the leftmost derivation of the following grammar ?

$S \rightarrow AB \mid \varepsilon$

$A \rightarrow aB$

$B \rightarrow Sb$

, derive the string abb

Soln:-

$S \rightarrow AB$
$\rightarrow aBB$
$\rightarrow aSbB$
$\rightarrow abB$
$\rightarrow abSb$
$\rightarrow ab\varepsilon b$
$\rightarrow abb$

Right-most derivation

Find Right-most-derivation of the following Grammar:

$S \rightarrow AB \mid \varepsilon$

$A \rightarrow aB$

$B \rightarrow Sb$

derive the string abb.

Soln:-

$S \rightarrow AB$
$\rightarrow ASb$
$\rightarrow Ab$
$\rightarrow aBb$
$\rightarrow aSbb$
$\rightarrow a\varepsilon bb$
$\rightarrow abb$

DERIVATION TREE / PARSE TREE

- Derivation tree is a graphical representation for the derivation of the given production rules for a given CFG.
- It is simple way to show how the derivation can be done to obtain some string from a given set of production rules.
- The derivation tree is also called a parse tree

Parse tree contains the following properties

- The root node is always a node indicating start symbols.
- The derivation is read from left-to right.
- The leaf node is always terminal nodes.
- The interior nodes are always the non-terminal nodes.

Eg:- production rules

$E \rightarrow E+E$

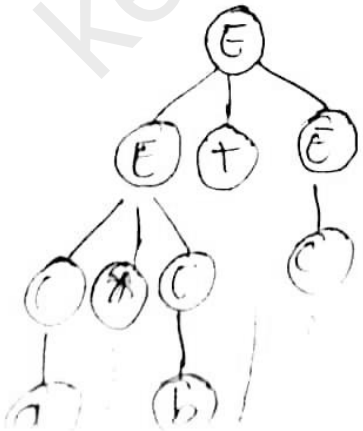
$E \rightarrow E*E$

$E \rightarrow a|b|c$

input

a*b+c

soln:-

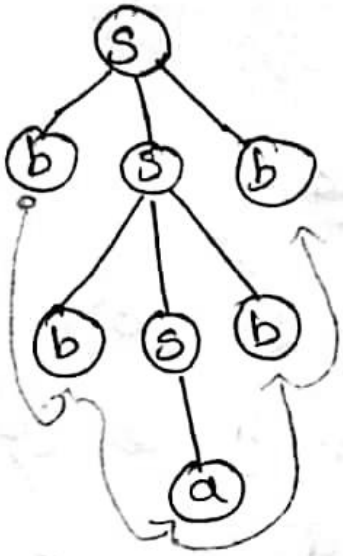


$\Rightarrow a*b+c$

Draw a derivation tree for the string "bbabb" from the CFG given by

$S \rightarrow bSb \mid a \mid b$.

Soln:-



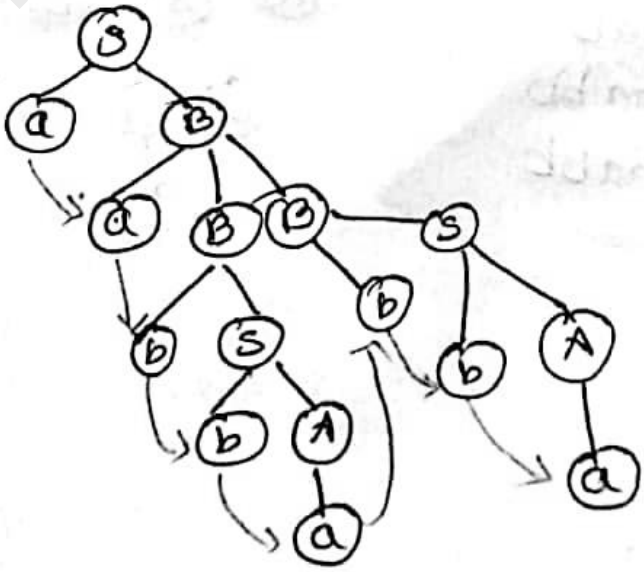
$\Rightarrow$ bbabb

Construct a derivation tree for the string aabbabba for the CFG given by,

$S \rightarrow aB \mid bA$

$A \rightarrow a \mid aS \mid bAA$

$B \rightarrow b \mid bS \mid aBB$



= aabbabba

Q) Derive the string 'aaabb' from the given grammar

$S \rightarrow AB \mid BA$

$B \rightarrow bB \mid b$

$A \rightarrow aA \mid a$

Right-most derivation Tree

$S \rightarrow AB$

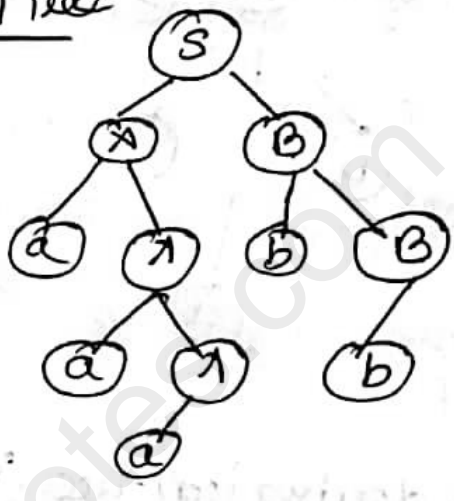
$\rightarrow AbB$

$\rightarrow Abb$

$\rightarrow aAbb$

$\rightarrow aaAbb$

$\rightarrow aaabb$



Leftmost derivation tree

$S \rightarrow AB$

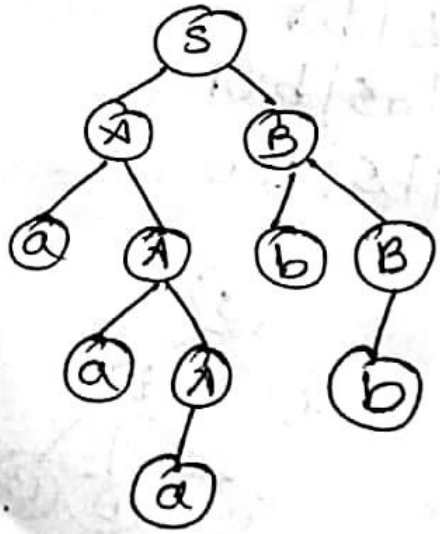
$\rightarrow aAB$

$\rightarrow aaAB$

$\rightarrow aaaB$

$\rightarrow aaabB$

$\rightarrow aaabb$



AMBIGUITY IN GRAMMAR

- A grammar is said to be ambiguous if there exists
  - More than one left-most derivation or
  - More than one right-most derivation
  - More than one parse tree for the given input string
- If the grammar is not ambiguous, then it is called unambiguous.
- If the grammar has ambiguity, then it is not good for compiler construction.
- no method can automatically detect and remove the ambiguity, but we can remove ambiguity by rewriting the whole grammar without ambiguity.

Eg:- Let us consider a grammar G with the production rule.

$E \rightarrow I$

$E \rightarrow E+E$

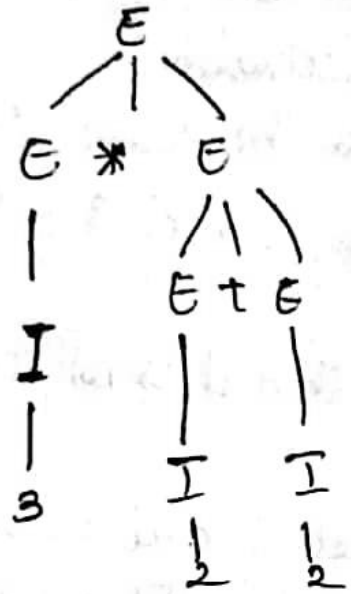
$E \rightarrow E*E$

$E \rightarrow (E)$

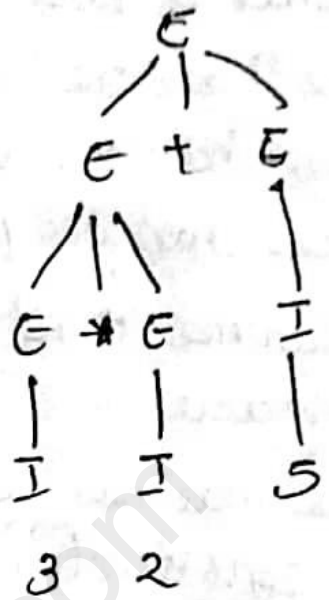
$I \rightarrow \epsilon \mid 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid \ldots \mid 9$

consider the string 3*2+5.

- For the string '3*2+5' the above grammar can generate 2 parse trees by left-most derivation.



$\Rightarrow 3 * 2 + 2$



$\Rightarrow 3 * 2 + 2$

- Since there are 2 parse trees for a single string "3*2+5", the grammar G is ambiguous.

Eg:- Consider the grammar (V, T, E, P)

$V = \{E, I\}$

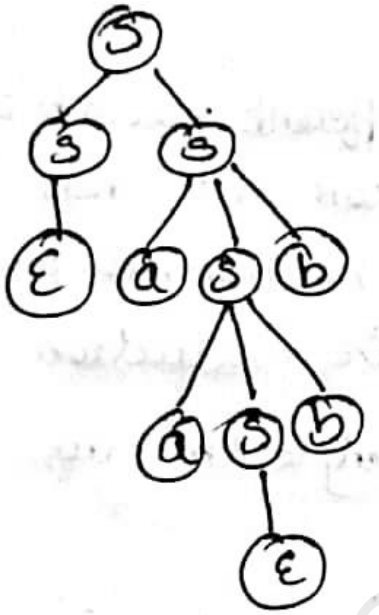
$T = \{a, b, c, +, *, (,)\}$

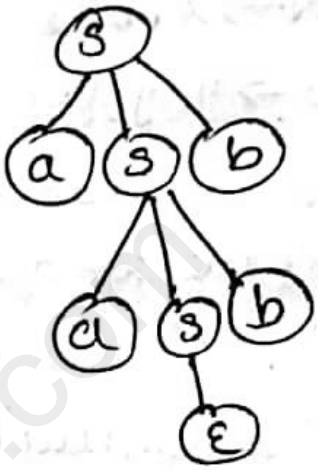
$P \Rightarrow E \rightarrow I$

Eg:- check whether the given grammar G is ambiguous or not for the given string "aabb"?

$S \rightarrow aSb \mid SS$

$S \rightarrow \varepsilon$





∴ G. There are 2 parse tree for a single string "aabb", the grammar G is ambiguous.

Simplification of CFG

1) Substitution Rule

Let $G = (V, T, S, P)$ be a CFG suppose that P contains a production of the form

$A \rightarrow x_1 B x_2$

$B \rightarrow y_1 | y_2 | \cdots y_n$

Assume that A & B are different variables. If the set of all production in P, then let G', $G' = (V, T, P', S)$ be the grammar in which P' is constructed by deleting production.

$A \rightarrow x_1 B x_2$ from P & adding to the other production

$\therefore P' \Rightarrow x_1 y_1 x_2 | x_1 y_2 x_2 | \cdots x_1 y_n x_2$

Eg:- $S \rightarrow aBa$

$B \rightarrow bcd$

$\therefore S \rightarrow abcda$

2) Removing useless production

- Let G = (V, T, P, S) be a CFG a variable A ∈ V is said to be as if we can't reached to the production from starting state.

Eg:- S → A
A → aA | ε
B → bA → useless production.

- There are 2 reason to become a variable useless either because it can't be reached from start symbol or it can't derive a terminal string.

Removing ε production

Any production of a CFG of the form A → ε is a ε production.

eg:- Remove ε production from the grammar.

S → aS, b
S, → aS, b | ε

soln:-

S → aS, b
a ε b
a b

S, → aS, b
→ a ε b
→ ab

Then production after removing ε becomes

$S \rightarrow aS_1b \mid ab$

$S_1 \rightarrow aS_1b \mid ab$

(9) Remove ε production from the following CFG by preserving the meaning of it.

$S \rightarrow ABAC$

$A \rightarrow aA \mid \varepsilon$

$B \rightarrow bB \mid \varepsilon$

$C \rightarrow c$

Soln:- The production with ε are

$A \rightarrow \varepsilon$, $B \rightarrow \varepsilon$

- Replace A with ε. consider the first production

$S \rightarrow ABAC$

$S \rightarrow BAC$, $S \rightarrow ABC$, $S \rightarrow BC$

- consider the second production

$A \rightarrow aA$

$A \rightarrow a$

- now replace B with ε.

consider the production $S \rightarrow ABAC \mid BAC \mid ABC \mid BC$

~~$S \rightarrow ABAC$~~ ~~$S \rightarrow AAC$~~ $S \rightarrow AAC \mid AC \mid C$

- Consider the next production $B \rightarrow bB$.

$B \rightarrow b$.

∴ Final answer after removing the ε prod is

S → ABAC | BAC | ABC | BC | AAC | AC | C

A → aA | a

B → bB | b

Removing unit production

Any production of a CFG of the form A → B where A, B ∈ V is called unit production.

We can remove unit production by drawing a dependency graph with an edge C, D, whenever the grammar as a unit production C → D

imp:

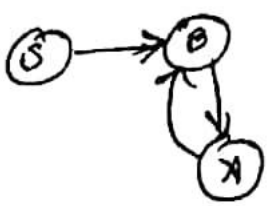
The dependency diagram is generated between unit production only.

Eg:-

S → Aa | B

B → A | bb

A → a | bc | B



→ dependency diagram with unit production only

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

$S \to Aa$

$B \to bb$

$A \to a \mid bc$

~~$S \to bb$~~ $S \to Aa \mid bb \mid a \mid bc$

$B \to a \mid bc \mid bb$ ⟶ useless production

$A \to a \mid bc \mid bb$

$\therefore S \to Aa \mid bb \mid a \mid bc$

$A \to a \mid bc \mid bb.$

∴ we remove unit production from the grammar. note that removal of unit production has made B & A associated production useless.

Q) Remove the unit production from the following CFG by preserving the meaning of it.

$S \to XY$

$X \to a$

$Y \to Z \mid b$

$Z \to M$

$M \to N$

$N \to a$

Soln: —

unit productions are

$Y \to Z$

$Z \to M$

$M \to N$

consider $M \to N$
$M \to a$

consider $Z \to M$
$Z \to a$

consider $Y \to Z$
$Y \to a$

$\therefore$ $S \to XY$
$X \to a$
$Y \to a|b$
$Z \to a$ — useless production
$M \to a$ — useless production
$N \to a$ — useless production.

$\therefore$ the final answer is

$S \to XY$

$X \to a$

$Y \to a|b$

NORMAL FORMS

1) Chomsky Normal Form (CNF)
2) Greibach Normal Form (GNF)

CHOMSKY NORMAL FORM

In chomsky normal form (CNF) we have a restriction on the length of RHS; which is, elements in RHS should either be 2 variables or a terminal.

A context Free Grammar is in chomsky Normal Form if the productions are in the following forms:

$A \to a$

$A \to BC$,

where A, B, & C are non terminals & a is a terminal.

- And also start symbol can generate ε i.e., $S \to \varepsilon$

(Q) $S \to AS \mid BAS$

$S \to SA \mid aa$

It is doesn't follow CNF.

$S \to XS \mid XAS$ → 3 variables in RHS so, violates the rule

$S \to SX \mid aa$ → 2 terminals in RHS so, it violates the rules.

Steps for converting CFG to CNF

1) Eliminate start symbol from the RHS. If the start symbol 'S' is at the right-hand side of any production, create a new production as

$S_1 \rightarrow S$,

where S_1 is the new start symbol.

2) In the grammar, remove the null, unit and useless productions.

3) Eliminate terminals from the RHS of the production if they exist with other non-terminals

Eg:- $S \rightarrow aA$

$\Rightarrow S \rightarrow XA$

$X \rightarrow a$

4) Eliminate RHS with more than 2 non-terminals

Eg:- $S \rightarrow ASB$ can be decomposed as.

$\Rightarrow S \rightarrow RB$

$R \rightarrow AS$

Q) Convert the given grammar to CNF ?

$S \rightarrow ABa$

$A \rightarrow aab$

$B \rightarrow Ac$

soln :-

i) $S \rightarrow ABX$

$A \rightarrow XXbY$

$B \rightarrow Ac$

$X \rightarrow a$

$Y \rightarrow b$

ii) $S \rightarrow NX$

$A \rightarrow RY$

$B \rightarrow AC$

$X \rightarrow a$

$Y \rightarrow b$

$N \rightarrow AB$

$R \rightarrow XX$

5) Convert CFG to CNF

$S \rightarrow ASA \mid aB$

$A \rightarrow B \mid S$

$B \rightarrow b \mid \varepsilon$

• Since S appears in RHS, we add a new state S' & $S' \to S$ is added to the production.

ie,

$S' \to S$

$S \to ASA \mid aB$

$A \to B \mid S$

$B \to b \mid \varepsilon$

• Remove null productions

$B \to \varepsilon$

$A \to \varepsilon$

$(A \to B \mid S)$

$(A \to \varepsilon \mid S)$

$S' \to S$

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

$A \to B \mid S$

$B \to b$

• Remove the unit production.

After removing $S \to S$

$S' \to S$

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

$A \to B \mid S$

$B \to b$

After removing $S' \to S$

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

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

$A \to B \mid S$

$B \to b$

After Removing $A \to B$

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

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

$A \to b \mid S$

$B \to b$.

After removing $A \to S$

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

$A \to b \mid ASA \mid aB \mid AS \mid SA \mid \epsilon$, $B \to b$

3) now find out the productions that has more than 2 variables in RHS.

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

$\Rightarrow$ Take SA as X

ie,

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

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

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

$B \to b$

$X \to SA$

⇒ now change the production.

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

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

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

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

$B \to b$

$X \to SA$

$Y \to a$.

Greibach Normal Form (GNF)

Here we put restrictions not on the length of the right-sides of a production - but on the positions in which terminals & variables can appear. A CFG is said to be in GNF if all production have the form

$A \to a\alpha$

where $a \in T$ & $\alpha \in V^*$

ie, $A \to a$

$A \to aBCW$

$\in V^*$

- And starting symbol can generate ϵ production only.

Eg:- $S \to aXB \mid aB$,
$X \to aX \mid a$
$B \to bB \mid b$

Eg:- $S \to aXB \mid aB$
$X \to aX \mid \varepsilon$
$B \to bB \mid \varepsilon$

} not in CNF, because, only start symbol can generate ε.

Steps for converting CFG into CNF

1) Convert the grammar into CNF
2) If the grammar exist left recursion, eliminate it.

Elimination of left Recursion

Left recursion form

$A \to A\alpha$ (RHS starts with the same letter in LHS)

It can be removed by

$A \to \beta A'$
$A' \to \alpha A' \mid \varepsilon$

eg:- $A \rightarrow Aa \mid b$
$A \rightarrow A\alpha \mid \beta$ (pattern)

~~$A \rightarrow Aa \mid b$~~

$A \rightarrow bA'$
$A' \rightarrow aA' \mid \varepsilon$

3) In the grammar, convert the given production rule into CNF :

5) convert the given grammar into CNF :

$S \rightarrow XB \mid AA$
$A \rightarrow a \mid SA$
$B \rightarrow b$
$X \rightarrow a$

a) 1st check it's in CNF or not. If it's not in CNF then convert the grammar to CNF.

- check left recursion. (there is no left recursion & it's in CNF).
- The production rule $A \rightarrow SA$ is not in CNF, so we substitute $S \rightarrow XB \mid AA$ in the production rule $A \rightarrow SA$ as:

ie,

$S \rightarrow XB \mid AA$

$A \rightarrow a \mid XBA \mid AAA$

The production rule $S \rightarrow XB$ & $B \rightarrow XBA$ is not in CNF, so we substitute $X \rightarrow a$ in the production rule $S \rightarrow XB$ and $B \rightarrow XBA$ as:

$S \rightarrow aB \mid AA$

$A \rightarrow a \mid aBA \mid AAA$

$B \rightarrow b$

$X \rightarrow a$

Now we will remove left recursion

$A \rightarrow AAA$, we get.

$A \rightarrow AAA \mid a$

$\Rightarrow A \rightarrow aA'$

$A' \rightarrow AAA' \mid \epsilon$

ie, $S \rightarrow aB \mid AA$

$A \rightarrow aA' \mid aBAA'$

$A' \rightarrow AAA' \mid \epsilon$

$B \rightarrow b$

$X \rightarrow a$

Now we will remove ∈ production

$S \to aB \mid AA$

$A \to aA' \mid aBAA' \mid a \mid aBA$

$A' \to AAA' \mid AA$

$B \to b$

$X \to a$

- The production rule $S \to AA$ is not in CNF. So substitute $A \to aA' \mid aBAA' \mid a \mid aBA$ in production rule $S \to AA$ as:

$S \to aB \mid aA'A \mid aBAA'A \mid aA \mid aBAA$

$A \to aA' \mid aBAA' \mid a \mid aBA$

$A' \to AAA'$

$A' \to aA'AA' \mid aBAA'AA' \mid aAA' \mid aBAAA'$

$B \to b$

$X \to a$

- $A' \to AAA'$ is not in CNF. So we substitute $A \to aA' \mid aBAA' \mid a \mid aBA$ in production rule $A' \to AAA'$

$\therefore S \to aB \mid aA'A \mid aBAA'A \mid aA \mid aBAA$

$A \to aA' \mid aBAA' \mid a \mid aBA$

$A' \to aA'AA' \mid aBAA'AA' \mid aAA' \mid aBAAA'$

$A' \to aA'A \mid aBAA'A \mid aA \mid aBAA$

$B \to b$

$X \to a$

Proving the correctness of CFG

How to show that the constructed grammar does indeed generate some required language?
If L is some language, and G is a CFG, then to prove that L = L(G) you have to prove 2 things:

- Every string w generated by G is in L. Typically, this a proof by induction on the number of steps in the derivation of w.
- Every string w in L can be generated by G. Typically this is a proof by induction on the length, or some other parameter, of w.

Eg:- Let $L = \{0^i 1^j \mid 2i \le j \le 3i\}$. In the class

$$S \to 0S11 \mid 0S111 \mid \varepsilon$$

now, lets prove that G is indeed the grammar that generates L ie, L = L(G).

Claim 1: If a string w is generated by G, then $w \in L$

Proof: Induction on the no: of steps in the derivation of w

- Base case: Derivation with 1 step. The only derivation with 1 step is $S \Rightarrow \varepsilon$, thus $w = \varepsilon$ & so $w \in L$.
- Inductive hypothesis: Assume that for every derivation $S \Rightarrow w$ with $n \ge 1$ steps, w is in

- Inductive step: prove that every derivation with $n+1$ steps generates a string in L.

Let $S \overset{*}{\Rightarrow} w$ be a derivation with $n+1$ steps. Since $n+1 \geq 1$, the first step in this derivation can't be $S \Rightarrow \varepsilon$. Then the derivation starts with either $S \Rightarrow 0S11$ or $S \Rightarrow 0S111$.

In the 1st case, w must be of the form $0w_1 11$, where w_1 is a string derived from the nonterminal S in the remaining n steps. By the inductive hypothesis, $w_1 \in L$ ie, $w_1 = 0^i 1^j$ with $2i \leq j \leq 3i$.

$\therefore w = 00^i 1^j 11 = 0^{i+1} 1^{j+2} \in L$. because if $2i \leq j \leq 3i$, then $2i+2 \leq j+2 \leq 3i+2$, & so $2(i+1) \leq j+2 \leq 3(i+1)$.

Similarly, in the second case, w must be of the form $0w_1 111$, where w_1 is a string derived from the nonterminal S in the remaining n steps. By the inductive hypothesis $w_1 \in L$ ie,

$w_1 = 0^i 1^j$, with $2i \leq j \leq 3i$

$\therefore w = 00^i 1^j 111 = 0^{i+1} 1^{j+3} \in L$, because if $2i \leq j \leq 3i$, then $2i+3 \leq j+3 \leq 3i+3$ ie,

$2(i+1) < j+2 \leq 3(i+1)$

This completes the proof of the claim.