2.1 Context Free Grammars

1) They recognise a larger class of languages than the regular languages
2) They have applications in programming languages and compiler designs.

↳ Informal definition:

1) Terminal Symbols Similar to alphabet Symbols
2) Variable Symbols (Non terminals) Can be replaced with String of Variables and terminals.
3) Production rules dictate how variables get replaced
4) We have a start Variable / Starting point of Computation

↳ Process of Computation:

1) Write down the start Symbol as the Current String.
2) Pick a Variable in Current String and replace it with one of its production rules
3) follow above process till we get all terminals in the string.

eg: $\Sigma = \{0, 1\}$
$V = \{S\}$

Rules: $S \rightarrow OS1 \rightarrow I$
$S \rightarrow E \rightarrow II$
Start Symbol $\{S\}$

Procedure: $S \to 0SI \xrightarrow{I} 00SII \xrightarrow{I} 000SIII \xrightarrow{II} 000111$

General: It produces language of all terminal type 0^n1^n which is not regular (Pumping lemma) but CFG accept their Language.
Where $n \geq 0$

Example: 2

$S \to ASB/E$ — I
$A \to 0$ — II
$B \to 1$ — III

$S \to ASB \xrightarrow{II} 0SB \xrightarrow{I} 0ASBB \xrightarrow{I} 0ABB \xrightarrow{III} 0A1B$

$00111 \xrightarrow{II} 0A1111 \xleftarrow{III} 0A11B$

General:

Language accepted: $0^n1^{2n}, n \geq 0$

↳ Formal Definition of CFG:

A CFG can be defined using 4 tuple representation (V, Σ, P, S) or (V, T, P, S) where

(i) $V \to$ Set of Vertices / Variables
(ii) Σ or $T \to$ Set of terminals
(iii) $P \subseteq V \times \{V \cup \Sigma\}^*$ is a set of production rules
(iv) $S \in V$ is the start Symbol

For $U, V \in (V \cup \Sigma)^*$, we say that U yields V if

(i) $U = V$ or
(ii) $\exists$ a seq of strings $U_0, U_1, U_2 \ldots U_K$ such that $U_0 = U$ and $U_K = V$ and U_{i-1} yeild U_i in one step

Denoted as $\boxed{U \Rightarrow^* V}$

We define the language of G as

$$L(G) = \{ W \in \Sigma^* / S \Rightarrow^* W \}$$

A language L is said to be Context free language if $\exists$ a CFG (G) such that

$$\boxed{L = L(G)}$$

Types of production in CFG :

$\boxed{A \longrightarrow \alpha}$

$\boxed{\begin{array}{l} A \longrightarrow \text{Variable} \\ \alpha \longrightarrow (V \cup T)^* \end{array}}$

Example1: CFG for $L = \{ a^n b^m / n \geq 0, n \leq m \leq 2n \}$

$S \longrightarrow ASB / \epsilon$
$A \longrightarrow a$
$B \longrightarrow b / bb$

Example 2. $L = \{ W \in \{0,1\}^* /$ no of 0's = no of 1s $\}$
$\longrightarrow$ Start with 0
$\longrightarrow$ Start with 1

$S \rightarrow 0S1S / 1S0S / \epsilon$

Example.3 L. Language of Well balanced parenthesis

$\Sigma = \{(,)\}^*$

$S \rightarrow (S) / SS / \epsilon$

↳ Comparison to Regular :

Thus we can say that all regular languages are generated by Context free Grammar so Regular are Subset of Context free Grammars.

2.2 Parse tree / Derivation tree

CollateNotes

Formally it is defined as :

→ Let G be a CFG and w be a string $\in L(G)$ then a parse tree of a string w is a rooted ordered tree that represent the derivation of w with respect to G.

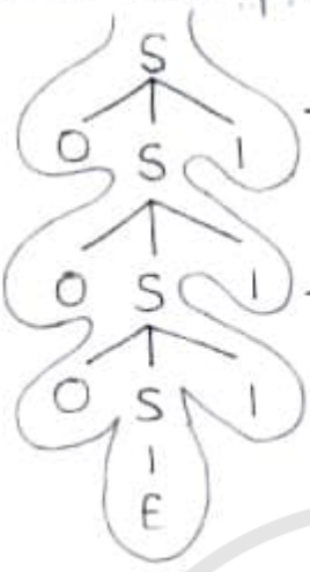
It is visual representation of derivation used in Construction syntax tree in Compiler Design.

Construct a parse tree :

$S \rightarrow 0S1 / \epsilon$

$w = 0^3 1^3$

Draw parse tree for w.?



→ Parsing proceeds from left to Right and top to bottom

→ Consider only terminals

↳ Properties of Parse/Derivation Tree:

(1) Every internal node is a Variable

(2) Every leaf node is either a terminal or ϵ If it is ϵ, then it is the only child of it's parent.

(3) Every internal node is expanded using production rules already specified

(4) Parse from left to right and top to bottom

(5) For unambiguoes grammars, we get only one (unique) parse tree.

2.3 Ambiguity in Grammar

CollateNotes

(Important)

↳ **Left most derivation :** of W with respect to ϵ is a derivation where in each step the left most variable of a string gets replaced.

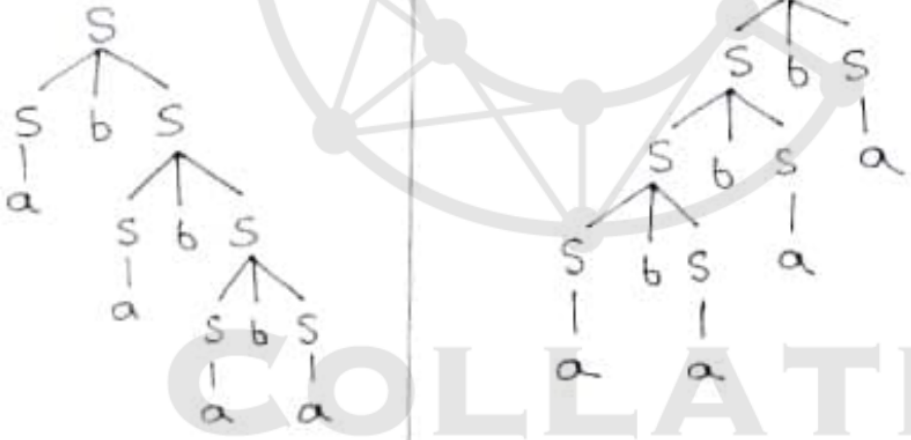
A string $W \in L(G)$ is ambiguous if it has 2 left most derivation for the same string.

→ A Grammar is ambiguous if it has any string which has more than one parse tree is left most derivation.

→ A context free language L is exherently ambiguous if all CFG has accepting L are ambiguous.

Example: $S \rightarrow SbS \mid a$

Deriving string : abababa



We Can Construct two parse tree thus the Grammar is ambiguous.

① $S \rightarrow SbS \rightarrow abS \rightarrow absbs \rightarrow ababs \rightarrow abababa$
$\rightarrow \leftarrow ababab s \leftarrow ababsbs$

② $S \rightarrow Sbs \rightarrow Sbsbs \rightarrow Sbsbsbs \rightarrow absbsbs$
$abababa \leftarrow abababs \leftarrow ababsbs$

Thus, 2 Left most derivation

2.4 Simplification of Context Free Grammars

A Simplified grammar is one such that it is free of any ϵo production, unit production and also useless symbols A Simplified grammar is usually preferred for parsing and also representation Simplified or reduced grammar is free of following things.

(i) ϵo productions
(ii) Unit productions (such that Non terminal → Non terminal)

Example: A → B type

(iii) Useless Symbols: they are such that either they don't derive any thing ie not deriving or they are not reachable.

Removing these in the aforesaid fashion is safe and they we will follow the above Rules in order to Simplify a grammar.

↳ Removing ϵ Production :

Steps : ① find Nullable set
② After Nullable Set write production with or Without Nullable Variable.
③ After the step only production Containing Null is S → ϵ (ie start Symbol) if start Symbol is in the Null set.

Example: S → AB
A → aAA / ϵ
B → bBB / ϵ

Null = { A, B, S } ∴ it is composed of both nullable variable S → ϵ will come

A → ϵ B → ϵ

Now write each production with and without Nullable Variable.

S → AB / B / A / ϵ	this is grammar without ϵ only start symbol can Contain ϵ
A → aAA / aA / a	
B → bBB / bB / b	

Example : S → Aba C
A → B C
B → b / ϵ
C → D / ϵ
D → d

Null = { B, C, A }

$$\left\{\begin{array}{l} S \to Aba\,C/ba\,C/Aba/ba \\ A \to BC/B/C \\ B \to b \\ C \to D \\ D \to d \end{array}\right\} \to$$ We don't replace both B and C and write $A \to \epsilon$ ∵ it is not a start symbol.

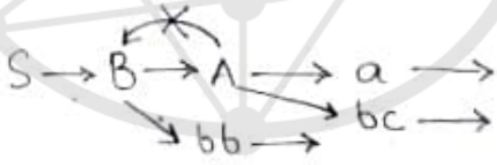
↳ Remaining Unit Productions:

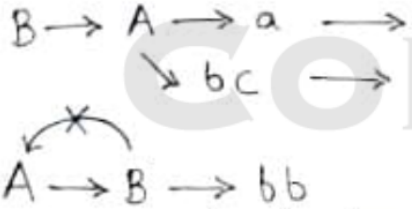
Steps: (i) follow path and find all chains
(ii) Add the Result
(iii) Delete the Unit production

Example: $S \to Aa/B$
$B \to A/bb$
$A \to a/bc/B$

Unit productions $\left\{\begin{array}{l} ① S \to B \\ ② B \to A \\ ③ A \to B \end{array}\right\}$

following Paths:





Remove Unit Production and add the Results

$$\left\{\begin{array}{l} S \to Aa/a/bc/bb \\ B \to a/bc/bb \\ A \to a/bc/bc \end{array}\right\}$$

Example: $S \rightarrow AB$
$A \rightarrow a$
$B \rightarrow c/b$
$C \rightarrow D$
$D \rightarrow \epsilon$
$\epsilon \rightarrow a$

Unit Productions: $\{B \rightarrow C, C \rightarrow D, D \rightarrow \epsilon\}$

$B \rightarrow C \rightarrow D \rightarrow \epsilon \rightarrow a$
$C \rightarrow D \rightarrow \epsilon \rightarrow a$
$D \rightarrow \epsilon \rightarrow a$

Delete Unit production and add the results

$S \rightarrow AB$
$A \rightarrow a$
$B \rightarrow b/a$
$C \rightarrow a$
$D \rightarrow a$
$\epsilon \rightarrow a$

↳ Removing Useless Symbols:

Steps: ① Create Useful Set ie
② Terminals are always useful & derivable
③ Variables formed by Symbols in useful set are also useful.
④ Above is an iterative process
⑤ Remove the not useful symbols
⑥ then check reachability.

Example:

$S \rightarrow AB$
$A \rightarrow a$
$B \rightarrow b/a$
$C \rightarrow a$
$D \rightarrow a$
$\epsilon \rightarrow a$

Useful = {a, C, D, ε, B, A, S, b}
Every thing is useful but {C, D, ε} are not reachable.
So remove

$\{ S \rightarrow AB,\ A \rightarrow a,\ B \rightarrow b/a \}$

Example 2:

$S \rightarrow AB/a$
$A \rightarrow Bc/b$
$B \rightarrow aB/C$
$C \rightarrow ac/B$

Useful: {a, b, A, S}

∴ B and C are not useful delete them.

$S \rightarrow a$
$A \rightarrow b$

{ B production removed }
{ Bc production also removed }

↓

here A is not reachable so delete.

$S \rightarrow a$ is reduced grammar.

2.5 Normal Forms

Context free grammars

Chomsky Normal form

Greibach Normal form

Production allowed

$A \rightarrow BC$

$A \rightarrow a$

&

$S \rightarrow \epsilon$

↓

Start symbol

Production allowed

$A \rightarrow a\alpha$

$a \rightarrow$ terminal

$\alpha \in V^*$

* Chomsky Normal form is useful in membership algorithm like CYK algorithm.

* Greibach normal form is useful in membership in Conversion of Grammar to Push down Automata

2.5.1 Chomsky Normal Form

Production allowed $\begin{cases} A \rightarrow BC \\ A \rightarrow a \\ S \rightarrow \epsilon \end{cases}$

Convert Context free grammar to chomsky Normal form

Steps: (1) If start Symbol present in RHS of any production Create new start production.

$S_0 \rightarrow S$

(2) Remove ϵ production
(3) Remove Unit Production
(4) Use observation skill to Convert to required form.

Example 1: Convert following grammar to CNF

$S \rightarrow ASB$
$A \rightarrow aAS \mid a \mid \epsilon$
$B \rightarrow SbS \mid A \mid bb$

(I) S appear on RHS so replace start symbol.

$S_0 \rightarrow S$
$S \rightarrow ASB$
$A \rightarrow aAS \mid a \mid \epsilon$
$B \rightarrow SbS \mid A \mid bb$

(II) Remove ϵ productions

Nullable = {A, B}

Remove ϵ productions

$S_0 \rightarrow S$
$S \rightarrow ASB \mid SB \mid AS$
$A \rightarrow aAS \mid a \mid aS$
$B \rightarrow SbS \mid A \mid bb$

{ $S_0 \rightarrow S$ (Redundant) not written }

(iii) Remove Unit productions

{ (1) $S_0 \rightarrow S$; (2) $B \rightarrow A$ }

$S_0 \rightarrow S \rightarrow ASB \mid SB \mid AS$
$B \rightarrow A \rightarrow aAS \mid a \mid as$

$S_0 \rightarrow As \mid AsB \mid sB$
$S \rightarrow As \mid AsB \mid sB$
$A \rightarrow aAs \mid as \mid a$
$B \rightarrow sbs \mid bb \mid aAS \mid as \mid a$

(iv) the observation skills and Convert to form

$S_0 \rightarrow As \mid PB \mid sB$	$P \rightarrow AS$
$S \rightarrow As \mid PB \mid sB$	$x \rightarrow a$
$A \rightarrow xP \mid xs \mid a$	$y \rightarrow b$
$B \rightarrow QS \mid YY \mid xP \mid xs \mid a$	$Q \rightarrow SY$

This grammar is is CNF

Example 2: $S \rightarrow ASB$
$A \rightarrow aASA \mid a \mid \epsilon$ (This is different example)
$B \rightarrow SbS \mid A \mid bb$

(1) Replace Start Symbol:
$S_0 \rightarrow S$
$S \rightarrow ASB$
$A \rightarrow aASA \mid a \mid \epsilon$
$B \rightarrow SbS \mid A \mid bb$

(II) Remove ϵ Production

Nullable = $\{A, B\}$

So $\rightarrow$ ϵ

S $\rightarrow$ ASB | SB | AS

A $\rightarrow$ aASA | a | aSA | asA | as

B $\rightarrow$ SbS | bb | A

(III) Remove Unit Production

So $\rightarrow$ ϵ $\rightarrow$ ASB | SB | AS

B $\rightarrow$ A $\rightarrow$ aASA | a | aSA | aAS | as

So $\rightarrow$ ASB | SB | AS

S $\rightarrow$ ASB | SB | AS

A $\rightarrow$ aASA | aSA | aAS | as | a

B $\rightarrow$ SbS | bb | aASA | aSA | aAS | as | a

(IV) Use observation skills

So $\rightarrow$ U_1B | SB | AS

S $\rightarrow$ U_1B | SB | AS

A $\rightarrow$ U_2A | U_3A | V_1U_1 | V_1S | a

B $\rightarrow$ U_4S | V_2V_2 | U_2A | U_3A | V_1U_1 | V_1S | a

$U_1 \rightarrow$ AS

$U_2 \rightarrow V_1U_1$

$U_3 \rightarrow V_1$S

$U_4 \rightarrow$ SV_2

$V_1 \rightarrow$ a

$V_2 \rightarrow$ b

This is required grammar.

2.5.2 Greibach Normal Form

Production Accepted :- $A \rightarrow a\alpha$
$S \rightarrow \epsilon$ (S → start symbol)

Where $\{a \in \text{terminal}\}$
$\{\alpha \in V^*\}$

It is useful to Convert CFG to PDA

Steps :- (I) Remove ϵ production and unit production
(II) Convert Grammar to CNF
(An Exam usually above steps are done already)

(III) Rename all the Symbols eg

$S \rightarrow AA$

$\boxed{A_1 \rightarrow A_2 A_2}$

rename
ie $\{S \rightarrow A_1\}$
$\{A \rightarrow A_2\}$

(IV) If $A_i \rightarrow A_j r$ where $(J > i) \rightarrow$ ok
but for
$A_i \rightarrow A_j r$ where $J < i$
Expand terminal A_j

(V) Eliminate left Recursion
ie $A_k \rightarrow A_k r$
Rule : $A \rightarrow A\alpha_1 / A\alpha_2 \ldots B_1 / B_2$
Introduce new terminal Z

$A \rightarrow B_1 / B_2 \ldots$
$A \rightarrow B_1 Z / B_2 Z \ldots$
$Z \rightarrow \alpha_1 / \alpha_2$
$Z \rightarrow \alpha_1 Z / \alpha_2 Z \ldots$

(VI) Finally Expand remaining not in form terminals including 2

Example: $S \to AA/a$, $A \to SS/b$ } Convert to GNF

(1) Eliminate null / unit production
(2) Convert to CNF
(3) Rename

$A_1 \to A_2 A_2 / a$
$A_2 \to A_1 A_1 / b$

(4)

$A_1 \to a$
$A_1 \to A_2 A_2$
$A_2 \to b$
$A_2 \to A_1 A_1$ × $(J<i)$

Expand

$A_2 \to A_2 A_2 A_1 / a A_1$

(5) Eliminate Left Recursion

$A_1 \to A_2 A_2 / a$
$A_2 \to A_2 A_2 A_1 / a A_1 / b$

(α_1: $A_2 A_1$; β_1: $a A_1$; β_2: b)

Left Recursion (new terminal)

$A_1 \to A_2 A_2 / a$
$A_2 \to a A_1 / b$
$A_2 \to a A_1 Z / b Z$
$Z \to A_2 A_1$
$Z \to A_2 A_1 Z$

$A_1 \rightarrow A_2A_2 \mid a$ → Put value of A_2

$A_2 \rightarrow aA_1 \mid b \mid aA_1z \mid bz$
$A_1 \rightarrow a \mid aA_1A_2 \mid bA_2 \mid aA_1zA_2 \mid bzA_2$

$Z \rightarrow A_2A_1$
$Z \rightarrow A_2A_1Z$ → Put value of A_2

$Z \rightarrow aA_1A_1 \mid bA_1 \mid aA_1ZA \mid bzA_1$
$Z \rightarrow aA_1A_1Z \mid bA_1Z \mid aA_1ZA_1Z \mid bzA_1Z$

→ GNF productions

2.6 Pumping lemma for Context free languages

Theorem: Let L be a CFL Then there exists a constant n such that if z is any string in L such that $|z|$ is at least n, then we can write $z = uvwxy$, subject to following conditions.

1) $|vwx| \leq n$ That is, the middle portion is not too long.
2) $vx \neq \epsilon$
3) for all $i \geq 0$, uv^iwx^iy is also in L.

Example: Apply pumping lemma on
$L = \{0^n 1^n 2^n \mid n \geq 1\}$

Let $z = 0^n 1^n 2^n$
Suppose the string breaks in $z = uvwxy$ where $|uwx| \leq n$ & v and x are not both ϵ. Then vwx cant have both 0's & & 2's since the last 0 and first 2 are seperated by $(n+1)$ positions.

So, if vwx has no 2's then 0's and 1's can be pumped which will create unequal 2's thus pumping lemma failed and language is not context free.

2.7 Closure properties of CFG

Collate Notes

① Union :- CFG are closed under Union.
If L_1 and L_2 be 2 CFG then $L_1 \cup L_2$ is also CFG.

$L_1 : \{V_1, T_1, P_1, S_1\}$ $L_2 : \{V_2, T_2, P_2, S_2\}$

$L = L_1 \cup L_2$

$L : \begin{cases} V_1 \cup V_2 \\ T_1 \cup T_2 \\ P_1 \cup P_2 \\ S \rightarrow S_1 / S_2 \end{cases}$

② Concatenation :- If L_1 and L_2 be two CFG then,
$L: L_1 L_2$ will also be CFG

Proof :- $L_1 = \{V, T, P, S\}$ $L_2 = \{V_2, T_2, P_2, S_2\}$

$$L = \begin{Bmatrix} V_1 \cup V_2 \\ T_1 \cup T_2 \\ P_1 \cup P_2 \\ S \to S_1 \; S_2 \end{Bmatrix}$$

(iii) Reversal :- If L be CFG then $(L)^R$ is also CFG where
$(L)^R$ = Reverse of L
$L : \{ S \to A_1 A_2 \quad A_{K-1} A_K \}$

$$\boxed{(L)^R : \{ S \to A_K A_{K-1} \quad A_2 A_1 \}}$$

(iv) Intersection :- If L_1 and L_2 be two CFG then $L_1 \cap L_2$ may or may not be CFG

Proof :- $L_1 : \{ 0^n 1^n 2^i \; ; n>,1; i>,1 \}$

$L_2 : \{ 0^i 1^n 2^n \; ; n>,1; i>,1 \}$

$L_1 \cap L_2 = \{ 0^n 1^n 2^n \; ; n>,1 ; i>,1 \}$

↓

not CFG by pumping lemma

(v) Complementation :- not closed under complementation

Proof by Contradiction- Let L be closed Under Complement
So, $\bar{L}$ is also CFG.

$L_1 \cap L_2 = \overline{\bar{L}_1 \cup \bar{L}_2}$

If above statement is true then $L_1 \cap L_2$ is also closed we know that $L_1 \cap L_2$ is not closed so our assumption is not valid.
Thus L is not closed under Complement.

(vi) Closure :- closed under Kleen closure

2.8 Push Down Automata

CollateNotes

These machine act as acceptor of context free grammar or Context free languages.
A push down automata is basically a

: finite Automata + stack

It is represented by 7 tuples :

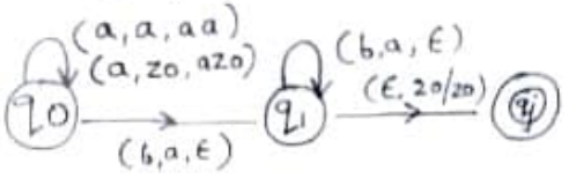
$$(Q , \Sigma , S , q_0 , Z_0 , F , \tau)$$

- Q → all States
- Σ → input alphabet
- S → Transition Symbol
- q_0 → initial State
- Z_0 → Bottom of stack
- F → Final stack
- τ → Top of stack

Deterministic PDA	NDPDA
(i) Deterministic push down Automata.	(i) Non deterministic push down automata
(ii) δ : $Q \times (\Sigma \cup \epsilon) \times T \to Q \times T^*$	(ii) δ : $Q \times (\Sigma \cup \epsilon) \times T \to_2 Q \times P^*$
(iii) States have deterministic moves or no moves	(iii) Non deterministic moves ie Same Condition going to different states.
(iv) Less powerful	(iv) More powerful
(v) It does not create multiple instances of itself.	(v) Creates multiple instances of itself while parsing.

2.9 Deterministic PDA

(i) $a^n b^n / n > 1$



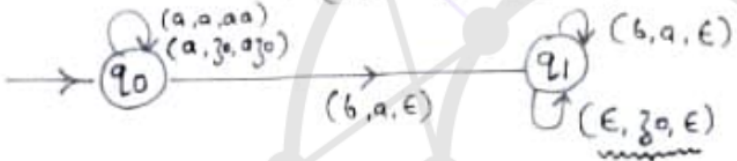
{ keep pushing a's for each b pop a when input is ε if top is z0 then accept }

Accepted by final state

transition

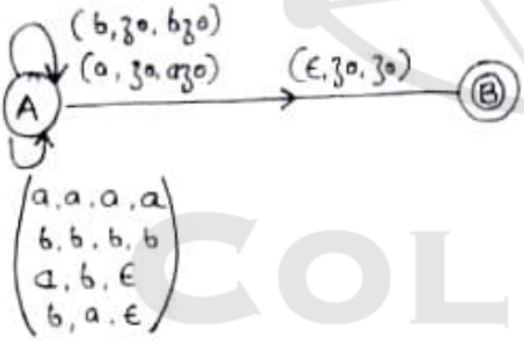
$\delta(q_0, a, z_0) = (q_0, az_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, a) = (q_1, \epsilon)$
$\delta(q, b, a) = (q_1, \epsilon)$

$\delta(q_1, \epsilon, z_0) = (q_f, z_0)$

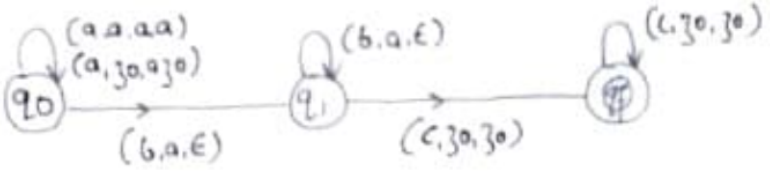


Accepted by empty stack

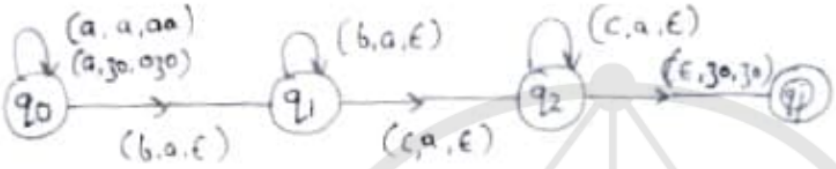
(ii) $W / n_a(W) = n_b(W)$



(iii) $a^n b^n c^m \mid n, m \geq 1$ (Important)



(iv) $a^{m+n} b^m c^n \mid m, n \geq 1$

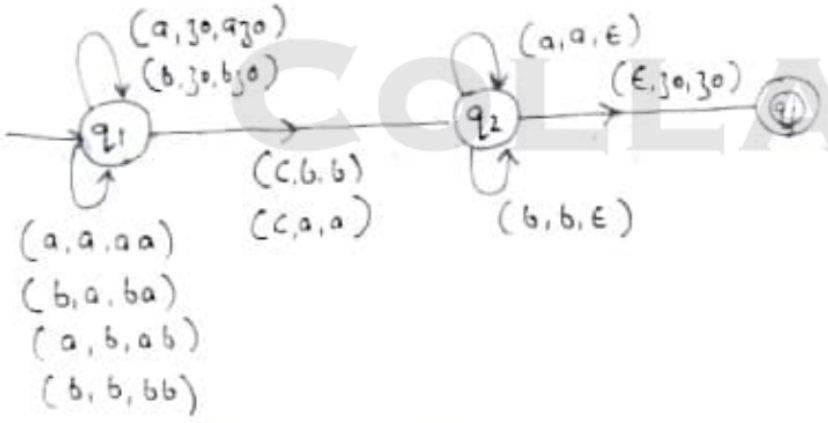


(v) $a^n b^{2n} \mid n \geq 1$

for every a push 2 a's

(vi) $WC W^R \mid W \in (a, b)^+$ (Important)

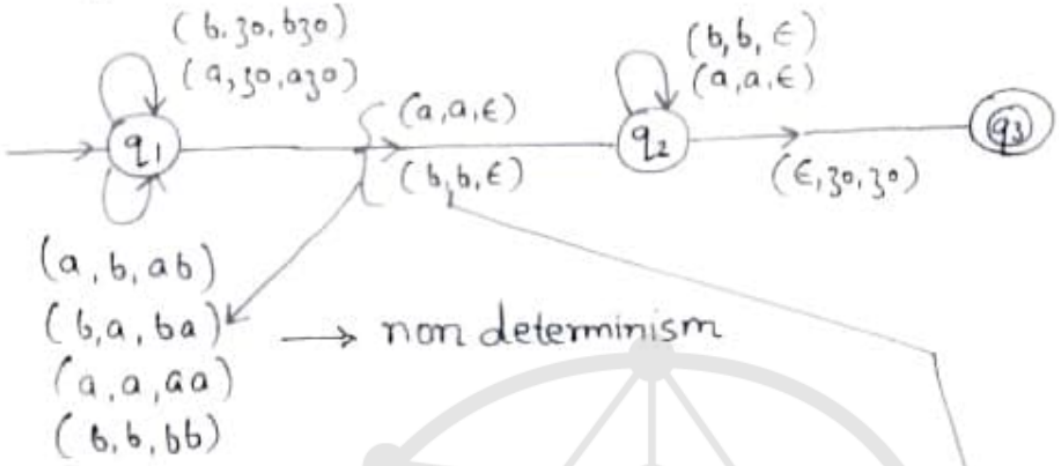
Center Known



2.10 Non deterministic PDA

Collate Notes

Example: $WW^R / W \in (a,b)^+$



① EK tarika manta hai ki center nahi aaya.

② Doosra kehta hai Center aa gya start popping.

2.11 Equivalence of PDA And CFG

Collate Notes

Equivalence means that every PDA can be converted to CFG and vice versa

2.11.1 From CFG (Context Free Grammar) to PDA (Push Down Automata)

We need to define only transition rules and may not Construct an automaton.

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

(i) For each variable or non terminal (Push)
$\delta(q, \epsilon, A) = \{q, \alpha\}$ where $A \rightarrow \alpha$

(ii) For each terminal (Pop)
$\delta(q, a, a) = (q, \epsilon) \mid a \in \Sigma$

Example 1: $S \rightarrow OBB$
$B \rightarrow OS / 1S / 0$

$V = \{S, B\}$, $T = \{0, 1\}$

$\delta = (q, \epsilon, S) = (q, OBB)$

$\delta = (q, \epsilon, B) = \{(q, OS), (q, 1S), (q, 0)\}$

$\delta = (q, 0, 0) = (q, \epsilon)$

$\delta = (q, 1, 1) = (q, \epsilon)$

PDA: $(\{q\}, \{0, 1\}, \{S, B, 0, 1\}, \delta, q, S, \phi))$

Γ (stack symbol)

final state ie accepted by empty stack

Example 2: $S \rightarrow 0S1 | A$
$A \rightarrow 1A0 | S | \epsilon$

$\delta(q, \epsilon, S) = \{(q, 0S1), (q, A)\}$
$\delta(q, \epsilon, A) = \{(q, 1A0), (q, S), (q, \epsilon)\}$
$\delta(q, 1, 1) = (q, \epsilon)$
$\delta(q, 0, 0) = (q, \epsilon)$

This defined equivalent PDA

$\{\{q\}, \{0,1\}, \{S, A, 0, 1\}, \delta, q, S, \$\}$ → bottom of stack

2.11.2 From PDA (Push Down Automata) to CFG (Context Free Grammar)

The productions in P are induced by moves of PDA as follows:

① S productions are given by $S \rightarrow [q_0 Z_0 q)$ for every $q \in Q$

② Each erasing move $\delta(q, a, Z) = (q', \epsilon)$ induces $(q Z q') \rightarrow a$

③ Each non erasing move $\delta(q, a, Z) = (q_1, Y_1, Y_2)$ in induces.

$[q Z q'] \rightarrow a [q, Z_1 q_2] [q_2 Z_2 q_3] \ldots [q_m Z_m q')$

where $q', q^2 \ldots q^m \in Q$

PDA:

$A = \{q_0, q_1\}, \{a, b\}, \{Z_0, Z\}, \delta, q_0, Z_0, \phi$

(Z_0: stack alphabet; q_0: initial state; ϕ: Bottom of stack)

$\delta(q_0, b, Z_0) = (q_0, ZZ_0)$ → non erasing
$\delta(q_0, \epsilon, Z_0) = (q_0, \epsilon)$
$\delta(q_0, b, Z) = (q_0, ZZ)$
$\delta(q_0, a, Z) = (q_1, Z)$
$\delta(q_1, b, Z) = (q_1, \epsilon)$
$\delta(q, a, Z_0) = (q_0, Z_0)$

$S \rightarrow [q_0 Z_0 q_0]$
$[q_0 Z_0 q_1]$ } Rule (1)

$\delta(q_0, b, Z_0) = (q, Z, Z_0)$ → not erasing

$[q_0 Z_0 q_0] \rightarrow b\,[q_0 Z\, q_0]\,[q_0 Z_0 q_0]$

$[q_0 Z_0 q_0] \rightarrow b\,[q_0 Z q_1]\,[q_1 Z_0 q_0]$

$[q_0 Z_0 q_1] \rightarrow b\,[q_0 Z q_0]\,[q_0 Z_0 q_1]$

$[q_0 Z_0 q_1] \rightarrow b\,[q_0 Z q_0]\,[q_0 Z_0 q_1]$

$\delta(q_0, \epsilon, Z_0) \rightarrow (q_0, \epsilon)$ → Erasing

$[q_0 Z_0 q_0] \rightarrow \epsilon$

$\delta(q_0, b, Z) = (q_0, ZZ)$

$[q_0 z q_0) = b(q_0 z q_0)\ [q_0 z q_0]$

$[q_0 z q_0) = b[q_0 z q_1]\ [q_1 z q)$

$[q_0 z q_1] = b[q_0 z q_0]\ [q_0 z q_1]$

$b[q_0 z q_1]\ [q_1 z q_1]$

$\delta(q_0, a, z) = (q, z) \rightarrow$ non erasing

$[q_0 z q_0] \rightarrow a[q_1 z q_0)$

$[q_0 z q_1] \rightarrow a[q_1 z q_1]$ }

$\delta(q_1, b, z) = (q, \epsilon)$

$[q_1 z q_1] \rightarrow b$ }

$\delta(q, a, z_0) = (q_0, z_0)$

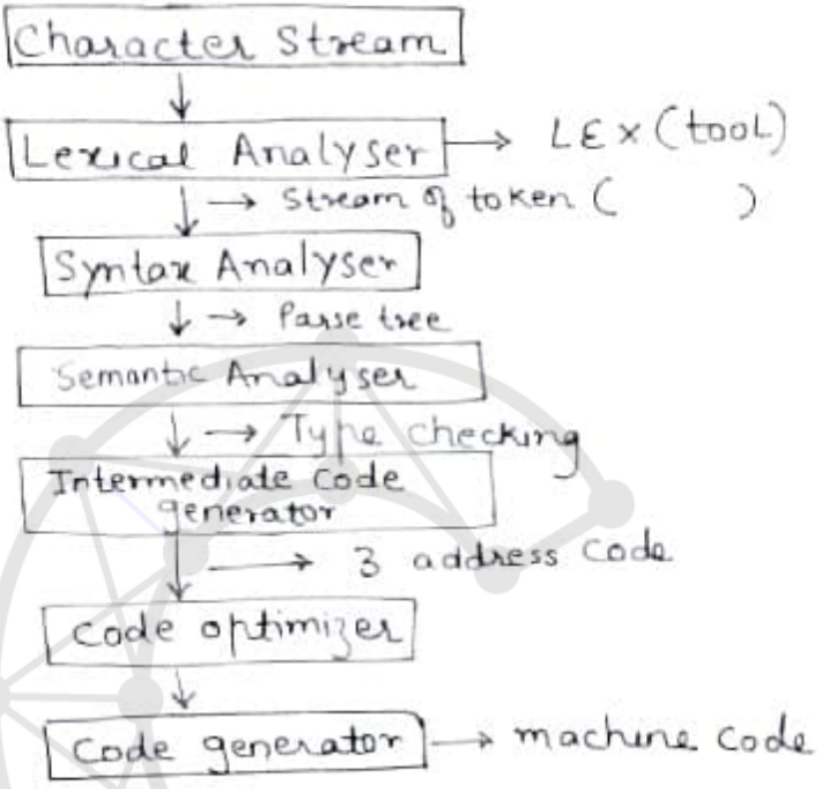
$[q_0 z_0 q_0] = a[q_0 z_0 q_0]$

$[q_0 z_0 q_1] = a[q_0 z_0 q_1]$ }

Blue Pen → CFG

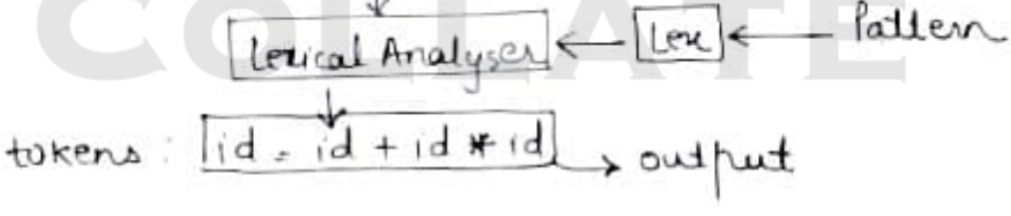
2.12 Overview of LEX and YACC

Language processing



Lesk and Johnson published paper on lex and YACC in 1975 which Simplified Compiler Writing

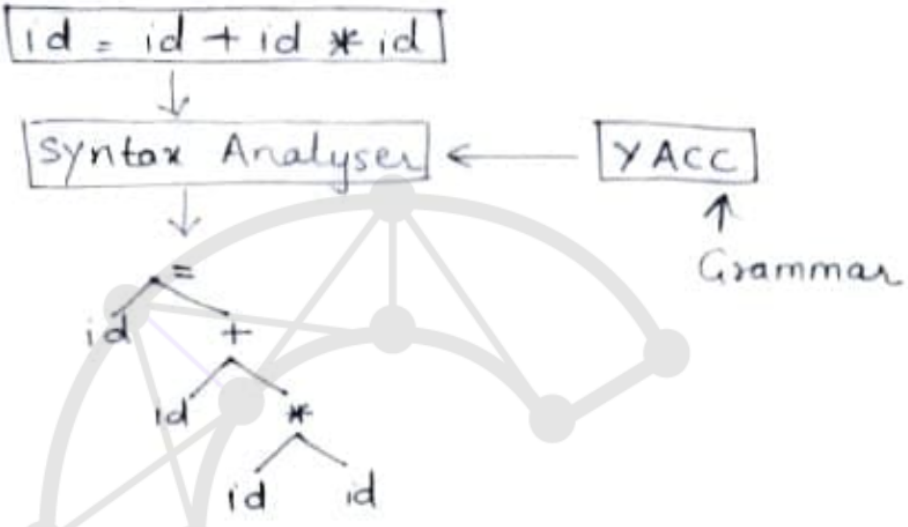
Source code : a = b + c * d



↳ The lex is provided a pattern according to which it creates token they may be regular expression abc

Pattern → Regular expression

↳ YACC: is parsing tool which takes input the grammar and create a parse tree or derivation tree It operates upon stream of tokens provided by LEX



Both Lex and YACC form front end of complete design.