Unit-II:

Topics: chapter-1:

1) Grammar – Def. & Representation
2) Types of Grammar.
3) Context Free Grammar (CFG)
4) Construction of CFG.
5) Difference b/w CFG & RL.
6) Derivation, Types of derivation
5) Derivation Tree (LMDT, RMDT)
6) Ambiguous grammar
7) Left recursion elimination

chapter: 2.

1) Pushdown Automata (PDA) – Def
2) Structural representation of PDA
3) construction of PDA.
4) CFG to PDA
5) PDA to CFG.

Grammar:

In automata, Grammar is defined as 4-tuples.

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

Where V or N → Set of non terminal symbols (Finite non-empty)

T → Finite set of terminal symbols

P → Production Rules

S → Start symbol.

Types of Grammar:

Grammar	Language	Automata	Production Rules
Type 3	Regular	Finite state Automata	$A \to \alpha$
Type 2	context-Free	Pushdown Automata (Nondeterministic)	$A \to \alpha\beta$
Type 1	context-Sensitive.	Linear bounded Automata.	$A \to Y$
Type 0	Recursively enumarated	Turing Machine	No restriction

Dr. K. Bhargavi

Types of Grammar

- On the Basis of Type of Production Rules
  - Type 0
  - Type 1
  - Type 2
  - Type 3.

  (Chomsky Hierarchy)
- On the Basis of No. of derivation Trees
  - Ambiguous Grammar
  - Unambiguous Grammar
- On the basis no. of Strings
  - Recursive Grammar
    - Left Recursive
    - Right Recursive
  - Non-Recursive Grammar.

Def:

A CFG is a formal grammar that consists of set of production rules used to generate strings in a Language

Purpose of CFG:

↳ To list all strings in a language using set of rules (i.e production rules).

↳ It extends the capabilities of regular expressions and finite automata.

Formal definition of CFG

A CFG is a set of 4 tuples.

$$G_1 = (V, T, P, S)$$

Where

↳ V is a finite set of variables (i.e non terminals — represents with uppercase letters)

↳ T is a set of Terminal / symbols (i.e represented by lowercase letters)

Symbols that appear in the final strings of the language cannot be replaced further.

Note: Left hand side only be a variable not Terminal.

Right hand side it can be variable or Terminal or both combination of variable and Terminal.

Variable → Varible / Terminal

→ P is a finite set of productions Rules, of the form $A \to \alpha$

where A is a variable

and $\alpha \in (V \cup T)^*$.
(V variable, T terminal with ∈)

Production rules represent recursive definition of the language.

Production symbol →

→ S is the start symbol represents the language.

Start symbol is always a non-terminal / variable.

Production rules represented as

LHS → RHS.

Single nonterminal / Variable → ∈ / string of terminals / nonterminals.

$A \rightarrow aB$.

State A upon input a gives state B.

$B \rightarrow \in$ (Followed by epsilon)

$B \rightarrow a$ (Followed by Terminal)

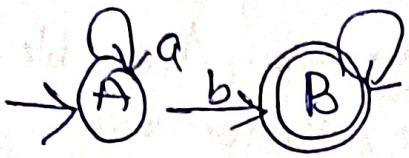
$C \rightarrow aA$ (Terminal followed by nonterminal).

$D \rightarrow Ab$ (nonterminal followed by Terminal/variable).

Regular Grammar
- Left Linear Grammar
- Right Linear Grammar

RG generates regular language.

① Construct Regular Grammar



$V = \{A, B\}$

$T = \{a, b\}$

Start symbol = A

$G_1 = \{V, T, P, S\}$

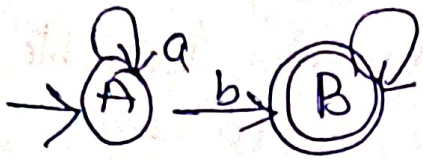
Right Linear Grammar

$P = \{ A \to aA,$
$A \to bB \text{ or } A \to b$ (∵ B is final state)
$B \to aB$
$B \to bB.$

$RG = (\{A, B\}, \{a, b\}, A, A)$

②

① Construct Regular Grammar



$V = \{A, B\}$

$T = \{a, b\}$

Start symbol = A

$G_1 = \{V, T, P, S\}$

Right Linear Grammar

$P = \{ A \to aA,$
$A \to bB \text{ or } A \to b$ (∵ B is final state)
$B \to aB$
$B \to bB.$

$RG_1 = (\{A, B\}, \{a, b\}, A, A)$

②

(1) Construct CFG for the language having any no. of a's over $\Sigma = \{a\}$

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

$RE = a^*$

Let S be the start symbol

$P = S \to aS$

$S \to \epsilon$

or $S \to aS / \epsilon$

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

$CFG = (\{S\}, \{a\}, P, S)$

Consider the string aaa & check the current aaa, by considering Production rule

$S \to aS$

$S \to aaS$

$S \to aaaS$

don't replace these S, in order satisfy the string aaa, represent $S \to \epsilon$

$S \to aaa\epsilon$

$S \to aaa$

2Q: Any no. of a's any no. of b's over $\Sigma = \{a, b\}$

Soln:

$$L = \{\in, a, b, aa, ab, ba, bb, aba, ---\}$$

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

Any no. of a's $\Rightarrow$ P : $S \to aS$

Any no. of b's P = $S \to bS$

0 occurance $P = S \to \in$

$\therefore$ $\boxed{S \to aS / bS / \in}$

consider the string aababab.

$S \to aS$

$S \to aaS$ ($\because S \to as$)

$S \to aabS$ (")

$S \to aababS$ (")

$S \to aababS$ ($S \to bS$)

This is given string

$S \to aababb \in$ ($S \to \in$)

$\boxed{S \to aababb}$

$\boxed{CFG = (\{S\}, \{a, b\}, P, S)}$

3) construct CFG for all strings having at least 2a's over $\Sigma = \{a, b\}$.

$L = \{aa, aba, abba, baaba, \ldots\}$

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

Let $T = (a+b)^*$ & write the PR.

$S \to TaTaT$

where $T \to$ any no. of a's & any no. of b's

$T \to aT$

$T \to bT$

$T \to \epsilon$

$\therefore T \to aT / bT / \epsilon$

$\therefore$ The production rules are

$S \to TaTaT$

$T \to aT / bT / \epsilon$

Let consider the string. babab & construct CFG.

$S = TaTaT$

$S \to bTaTaT \quad (T \to bT)$

$S \to b\epsilon aTaT \quad (T \to \epsilon)$

$S \to baTaT \quad (T \to bT)$

$S \to bab\epsilon aT \quad (T \to \epsilon)$

$S \to babaT \quad (T \to bT)$

$S \to babab\epsilon \quad (T \to \epsilon)$

$S \to$ babab.

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

$G = (\{S, T\}, \{a, b\}, P, S)$

4) Construct the CFG for $L = \{a^n b^n \mid n \geq 1\}$ over $\Sigma = \{a, b\}$

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

$L = \{ab, aabb, aaabbb, \ldots\}$

P :⇒ $S \to ab$

$S \to aSb$ (start with a, end with b)

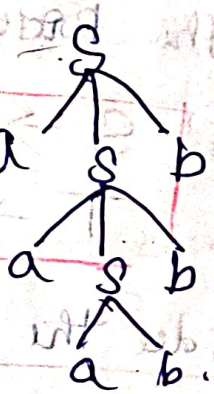
NOTE: No. of a's followed by no. of b's and both are equal. Not consisting ε.

$w = aaabbb$

$S \to aSb$

$\to aaSbb$ $(S \to aSb)$

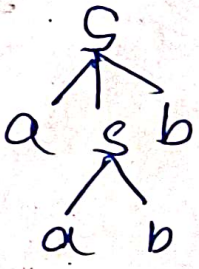
$\to aaaSbbb$ $(S \to ab)$



If $w = aabb$

$S \to aSb$ $(S \to ab)$

$S \Rightarrow aabb$



i.e aabb

$G = \{S, \{a, b\}, P, S\}$

Q5: construct the CFG which accepts the string for $L = \{a^n b^{2n} / n \geq 1\}$

Soln: $L = \{a^n b^{2n} / n \geq 1\}$

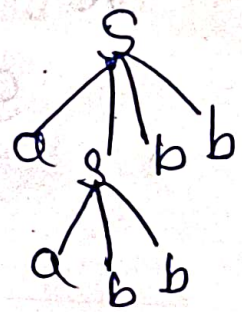
i.e one a followed by 2 b's

$L = \{abb, aabbbb, ---\}$

$S \rightarrow abb$
$S \rightarrow aSbb$

Let $\omega = aabbbb$

$\omega = aabbbb$
$S \rightarrow aSbb$
$S \rightarrow aabbbb$ ($S \rightarrow abb$)



aabbbb

$G_1 = \{S, \{a,b\}, P, S\}$

Q6: $L = \{\omega c \omega^T / \omega \in (a,b)^*\}$

Soln: $L = \{\omega c \omega^T / \omega \in (a,b)^*\}$

if $\omega = ab$ then $\{abcba\}$, $\omega = ba \Rightarrow bacab$

if $\omega = abb \Rightarrow abbcbba$, $\omega = bba \Rightarrow bbacabb$

start with a & end with a
start with b & end with b

$S \rightarrow aSa$
$S \rightarrow bSb$

if ab are 0 $\Rightarrow$ $w = c$, $w^T = c$

$\therefore S \to c$

The Production rules are

$$S \to c$$
$$S \to aSa$$
$$S \to bSb$$

$w = baCab$

$S \to bSb$

$S \to baSab$ $(\because S \to aSa)$

$S \to baCab$ $(S \to c)$

---

(10) At least one a over $\Sigma = \{a, b\}$

Soln: $L = \{a, ab, aa, bca, aab, abb, \ldots\}$

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

i.e Any no. of a's & Any no. of b's

$$S \to AaA$$
$$A \to aA \mid bA \mid \epsilon$$

$w = aab$

$S \to aA$

$S \to aaA$ $(\because A \to aA)$

$S \to aabA$ $(\because A \to bA)$

$S \to aab\epsilon$ $(\because A \to \epsilon)$

$G = (\{S, A\}, \{a, b\}, P, S)$

Derivation Trees / Parse Tree

Derivation Tree is a graphical representation for the derivation of the given production rules for a given CFG.

It is the simple way to show how derivation can be done to obtain some string from a given set of production rules.

The derivation tree is also called a Parse tree.

Derivation tree properties:

1. Root node represents start symbol / non Terminal
2. Interior node represents non terminals
3. Leaf nodes represents Terminals.
4. For each Non terminal apply the Production rule exist in the grammar.

The process of applying a sequence of Production rules (P) in order to obtain a string

8) Construct the DT for $S \to bSb / a / b$ for the string bbabb

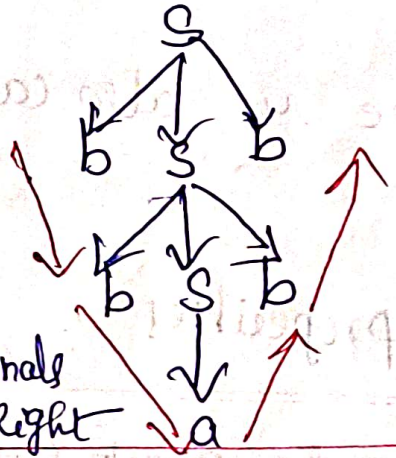
---

Given $S \to bSb$

$S \to a$

$S \to b$

Root symbol is start symbol. String start symbol is b; so consider initial state. P $S \to bSb$.



Read from left to right i.e Terminals.

$\Rightarrow$ bbabb.

Read terminals from left to Right

---

29) Given P as $S \to AB / \epsilon$,

$A \to aB$

$B \to Sb$

Write the DT for string aabbbb.

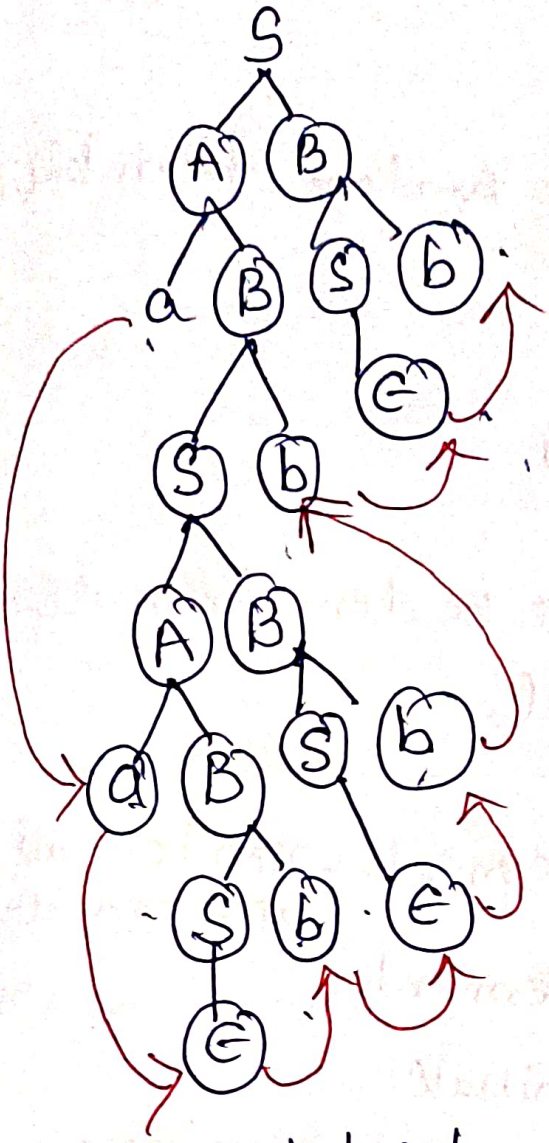
Given $S \to AB / \epsilon$

$A \to aB$

$B \to Sb$.

String starting with a, ~~[illegible]~~

ROOT node is start symbol i.e S



Read terminals. or leaf nodes from left to right

aaϵbϵbbϵb

⇒ aabbbb

Types of Derivation Trees:

2 Types of Derivation trees.

① Right most derivation tree. — Derivation from right most non terminal.

② Left most derivation tree. — Derivation from left most non terminal.

Left most derivation

Derivation will be done from left most non-terminal.

Right most derivation :

Derivation will be done from right most non-terminal.

Root node → start symbol — non terminal (uppercase letter)

Intermediate node → non terminals

leaf nodes → Terminals.

Q1) Given Grammar is $E \to E+E / E*E / id$ do left most and right most derivation for id + id * id

---

Left most derivation

$E \to E+E$

$E \to id+E$

$E \to id+E*E$

$E \to id+id*E$

$E \to id+id*E$

Right most derivation

$E \to E+E$

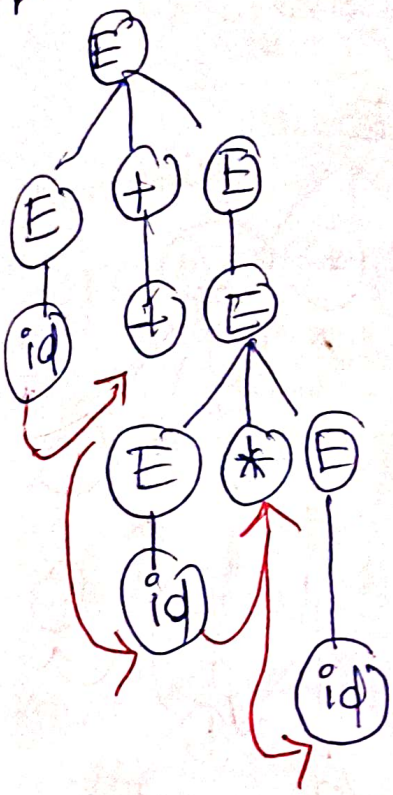
$E \to E+E*E$

$E \to E+E*id$

$E \to E+id*id$

$E \to id+id*id$

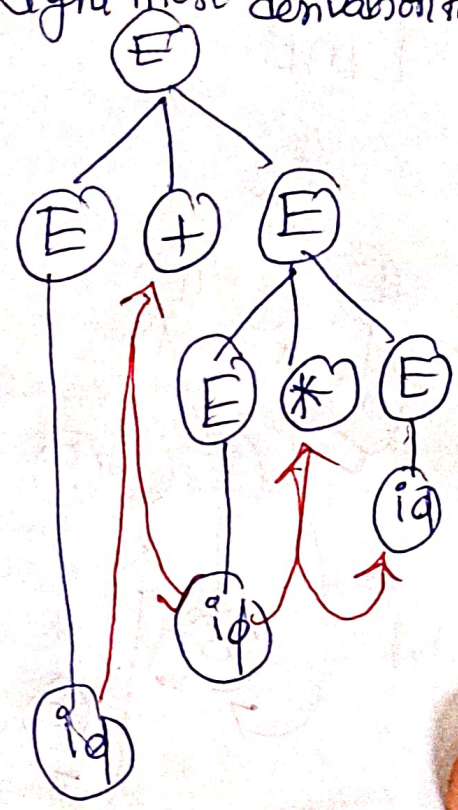
Left most derivation tree



Take terminals from left to right.

id + id * id.

Right most derivation tree



id + id * id

Q2. Given grammar is $E \rightarrow E+E \mid E*E \mid a \mid b \mid c$ derive the string $a+b*c$

Sol:

Left most Derivation

$E \rightarrow E+E$

$E \rightarrow a+E$

$E \rightarrow a+E*E$

$E \rightarrow a+b*E$

$E \rightarrow a+b*c$

Right most Derivation

$E \rightarrow E+E$

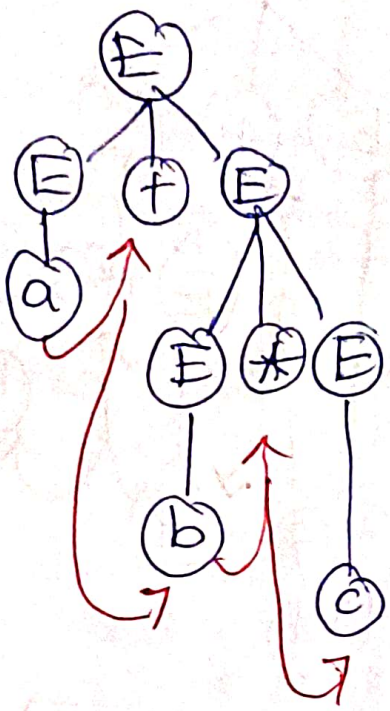
$E \rightarrow E+E*E$

$E \rightarrow E+E*c$

$E \rightarrow E+b*c$

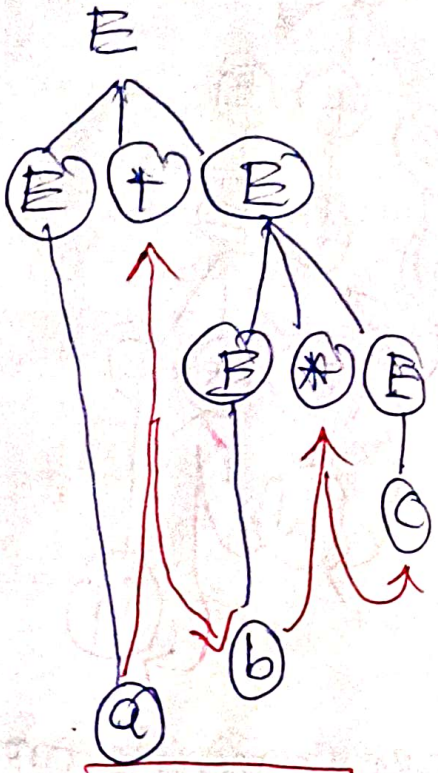
$E \rightarrow a+b*c$

LMD



a + b * c

RMD



a + b * c

Q3: S → TOOT

S → OT / 1T / ε , derive LMD, RMD for the string 10001

Ambiguous Grammar (AG)

Depending on number of derivation trees, CFG's are sub divided into 2 types.

1. Ambiguous Grammars.
2. Unambiguous Grammars.

A CFG is said to be ambiguous if there exist more than one derivation tree for the given input string (Left most or Right most)
(LMDT)
i.e More than one leftmost derivation Tree
& More than one Right most derivation Tree
(RMDT)

Ambiguous grammar cause inconsistency in parsing AG is not suitable for compiler construction.

Def: A CFG $G = (V, T, P, S)$ is ambiguous if there is a string in the terminal set T that can be produced more than one way, creating multiple parse trees.

Q1) Check whether the following grammar is ambiguous or not for the string "id + id * id"

$E \rightarrow E+E$

$E \rightarrow E*E$

$E \rightarrow id$.

Soln:

$P \Rightarrow E \rightarrow E+E$

$E \rightarrow E*E$

$E \rightarrow id$.

Right most DT (RMDT)

$E \rightarrow E+E$.

$E \rightarrow E+E*E$

$E \rightarrow E+E*id$

$E \rightarrow E+id*id$

$E \rightarrow id+id*id$.

RMDT:

$E \rightarrow E*E$

$E \rightarrow E*id$

$E \rightarrow E+E*id$

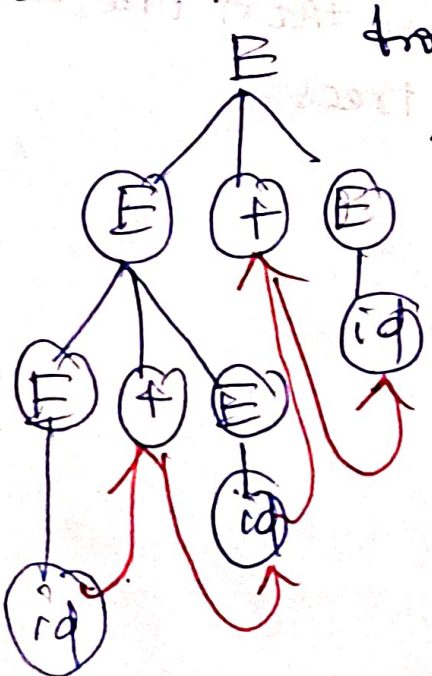
$E \rightarrow E+id*id$

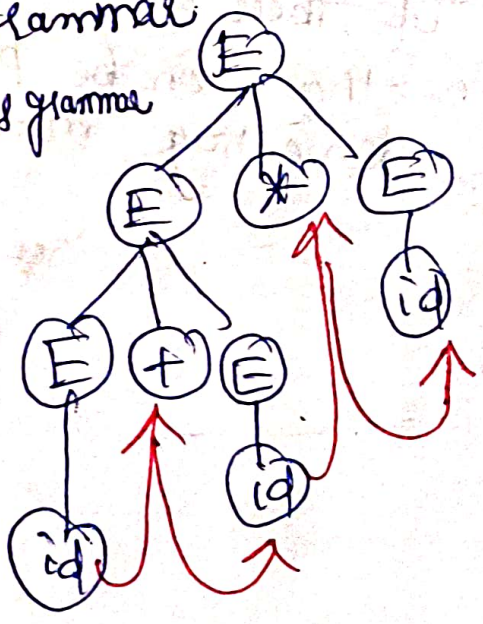
$E \rightarrow id+id*id$.

The above 2 Right most DT are generated.

Both parse trees are different, but derived from same grammar

∴ Ambiguous grammar





Q2: $S \to ABA$
$A \to aA/\epsilon$
$B \to bB/\epsilon$, derive the string aa from the above & check ambiguous or not.

Sol: Consider the left most derivation.
$w = aa$.

$S \to ABA$
$S \to aABA$ ($\because A \to aA$)
$S \to aaABA$ ($\because A \to aA$)
$S \to aa\epsilon BA$ ($\because A \to \epsilon$)
$S \to aa\epsilon\epsilon A$ ($\because B \to \epsilon$)
$S \to aa\epsilon\epsilon\epsilon$ ($\because A \to \epsilon$)
$S \to aa$.

$S \to ABA$
$S \to \epsilon BA$
$S \to \epsilon\epsilon A$
$S \to \epsilon\epsilon ABA$
$S \to \epsilon\epsilon aABA$
$S \to \epsilon\epsilon aaABA$
$S \to \epsilon\epsilon aa\epsilon BA$
$S \to \epsilon\epsilon aa\epsilon\epsilon A$
$S \to \epsilon\epsilon aa\epsilon\epsilon\epsilon$
$S \to aa$

deriving the same string aa with more than one left most derivation.

∴ The grammar is ambiguous

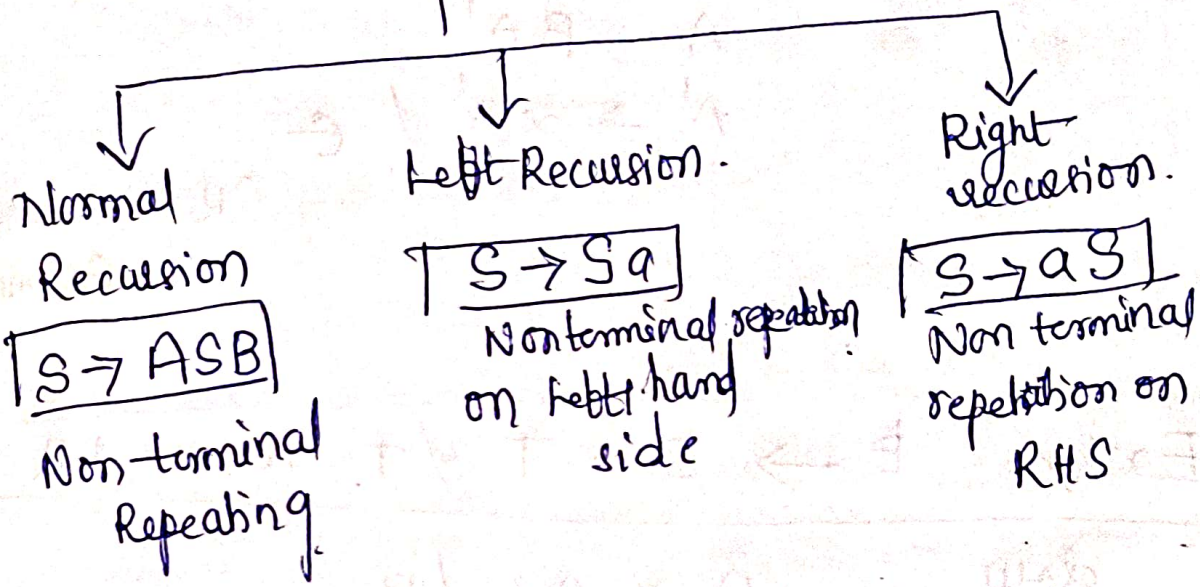
Q3: $S \to aSbS$
$S \to bSaS$
$S \to \epsilon$, derive the string abbaab & check grammar is ambiguous or not.

Regular grammar vs Context free gramma

Regular grammar (RG)	Context Free grammar (CFG)
1. RG is defined as $G_1 = (V, T, P, S)$ V – Non Terminals T – Terminals P – Production Rule – $S \to a$, $S \to aA$ S – Start symbol.	CG is derived as $G_1 = (V, T, P, S)$ $B \to a$ $S \to aAb$
2. RG is Used to construct DFA	CFG is used to construct PDA.
3. Not suitable for parsing.	Parsing is suitable
4. Type-3 Grammar	Type-2 Grammar
5. RG is a subset of CFG. Every RG can be a CFG.	Every CFG may not be RG.
6. Syntax of any programing language cannot be represented.	CFG can be represented by every Prog. Language.

Left Recursion Elimination or Removal of left recursion

Types of Recursion (R) — 3.



A Production of grammar is said to have left recursion, if left most variable of its RHS is same as variable of its LHS.

A grammar containing a production having left recursion is called as left recursive grammar

Ex: $S \rightarrow Sa / \in$

Left Recursion is problamatic situation for top down parsers.

∴ Left recursion has to be eliminated from the grammar

$A \to A\alpha / \beta$, where β does not begin with an A.

Then we can eliminate left recursion (LR) by replacing following

$$A \to \beta A'$$
$$A' \to \alpha A' / \epsilon$$

(Right recursion Grammar)

Ex:1 ✓ $E \to E + T / T$ remove LR

Soln: $A \to A\alpha / \beta$

$E \to E + T / T$

$A = E$, $\alpha = +T$, $\beta = T$

You: $E \to TE'$

$E' + TE' / \epsilon$.

---

2) $A \to AB\alpha / Aa / a$ remove LR.

You:

$A \to AB\alpha / Aa / a$

$A \to \beta A'$

$A' \to \alpha_1 A' / \alpha_2 A' / \epsilon$

$A \to ABx$
$A \to Aa$
$A \to a$

$A = A$
$\alpha_1 = Bx$
$\alpha_2 = a$, $\beta = a$.

$A \to AB\alpha / a$ $\alpha \quad \beta$	$A \to A a / a$ $\alpha \quad \beta$
Soln: $A \to \beta A'$ $\Rightarrow A \to aA'$ $A' \to \alpha A' / \epsilon$ $\Rightarrow A' \to B\alpha A' / \epsilon$	$A \to \beta A' \Rightarrow A \to aA'$ $A' \to \alpha A' / \epsilon \Rightarrow A' \to aA' / \epsilon$

The general form of left recursion is

$A \to A\alpha_1 + A\alpha_2 + \cdots | A\alpha_n | \beta_1 | \beta_2 | \cdots | \beta_n |$

can be replaced by

$A \to \beta_1 A' + \beta_2 A' | \cdots | \beta_n A'$

$A \to \alpha_1 A' | \alpha_2 A^2 | \cdots | \alpha_m A' / \epsilon$.

3Q) $A \to Ac/Aad/bd/c$, Remove the LR.

Soln:

$A \to Ac/Aad/bd/c$

$\alpha_1 \quad \alpha_2 \quad \beta_1 \quad \beta_2$

$A \to Ac/bd$	$A \to Aad/c$
$A \to bdA'$ $A' \to cA'/\epsilon$	$A \to cA'$ $A \to adA'/\epsilon$

4Q) Eliminate left recursion for

$A \to ABd/Aa/a$

$B \to Be/b$

$A \to A\alpha/\beta$
$A \to \beta A'$
$A' \to \alpha A'/\epsilon$

Soln:

$A \to ABd/Aa/a$

$A \to ABd/a \xrightarrow{RLR} A \to aA'$, $A' \to BdA'/\epsilon$

$A \to Aa/a \xrightarrow{RLR} A \to aA'$, $A' \to aA'/\epsilon$

$B \to Be/d \Rightarrow B \to dB'$, $B' \to eB'/\epsilon$

$\therefore$ $A \to aA'$
$A' \to BdA'/aA'/\epsilon$
$B \to dB'$, $B' \to eB'/\epsilon$

58) Remove the left recursion for the following grammar

$E \to E + T / T$

$T \to T * F / F$

$F \to id.$

Soln: (i) $E \to E + T / T$

$A \to A\alpha / \beta$

$A \to \beta A'$

$A' \to \alpha A' / \epsilon$

$A = E, \alpha = +T, \beta = T$

$A \to TA' \Rightarrow$ ~~[illegible]~~

$E \to TE'$

$E' \to +TE' / \epsilon$

$E \to TE'$

$E' \to +TE' / \epsilon$

(ii) $T \to T * F / F$

$A = T, \alpha = *F, \beta = F$

$T \to FT'$

$T' \to *FT' / \epsilon$

(iii) $F \to id.$

Unit - III . (Chapter-2)

Pushdown Automata

↳ Definition
↳ Structural representation of PDA.
↳ Construction of PDA
↳ CFG to PDA
↳ PDA to CFG

Unit - III
Chapter - II

Pushdown automata (PDA) & Structural representation:

A Pushdown automata is a way to implement a context free grammar. PDA is used to recognize context free languages (CFL) is

PDA has a stack, with this extra memory, the PDA can identify more languages.

Pushdown datomata consist of an
- ↳ Input tape
- ↳ A stack
- ↳ Finite set of states.

The PDA transitions between states based on input symbols and stack operations and validate strings generated by context-free grammars (CFG's).

Type 2	–	CFG – CFL	– Infinite memory	– PDA
Type 3	–	RG – RL	– Finite memory	– FSM.

Dr. K. Bhargavi

2

Stack has 2 operations → push & pop.

Finite memory is not sufficient, so extra memory is added in order to implement context free grammar.

PDA = FSM + A Stack

PDA = Finite state + A stack
memory

Stack follows LIFO

Push — New symbol is added at the top

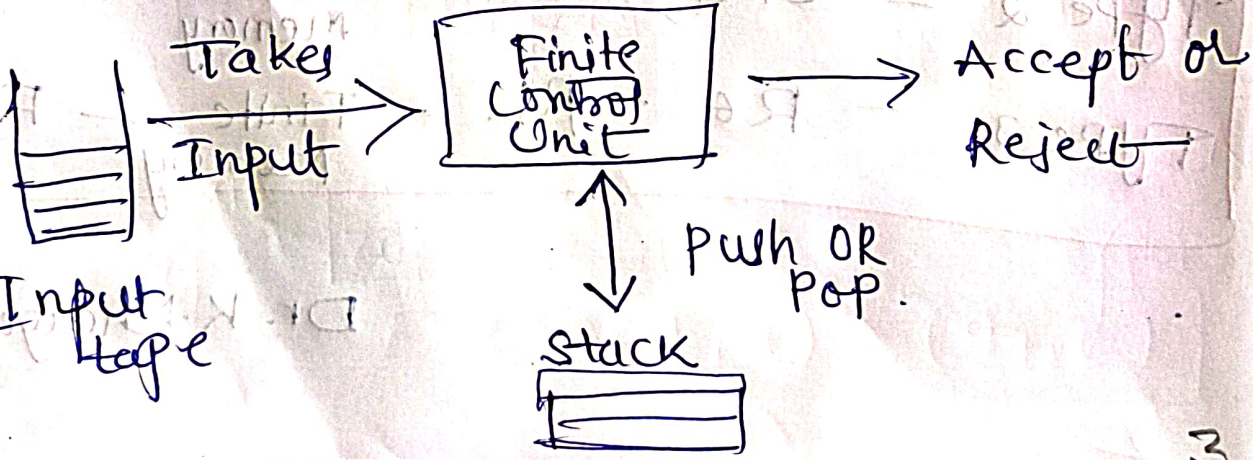
Pop — Reading & removing symbol from top.

Structural representation of PDA:

PDA has three (3) components

1. An input tape
2. A control Unit and
3. A Stack with infinite size.

PDA uses the concept of LIFO



Each transition

→ Is based on current i/p symbol and top of the stack.

→ Initially stack holds a special symbol Z_0, that is on the bottom of the stack. → Top.

→ Pops the top of stack symol.

→ Push new symbol onto the Stacks.

PDA = FSM + A Stack

PDA = 5 tuples + 2 tuples

PDA = 7 tuples.

PDA = M = $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

Where Q = set of finite states

Σ = set of finite alphabets/ symbols

Γ = Finite set of stack symbols

δ = Transition function.

q_0 = Initial state

Z_0 = Initial stack top symbol ($Z_0 \in \Gamma$)

F = Finite state (s)

δ - Transition Function where

$\delta : Q \times (\Sigma \cup \{\in\}) \times \Gamma \to Q \times \Gamma^*$

Steps for construction of PDA :

1. Write the language & consider the string.
2. Logic for construction of PDA
3. Transition function (δ).
4. Transition diagram.
5. PDA = $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$
6. ID of PDA.

Q) Design the PDA for accepting the Language

$L = \{a^n b^n / n \geq 1\}$

& $L = \{a^n b^m / n = m \geq 1\}$ &

check the string acceptance for aaabbb.

Solution:

$L = \{a^n b^n / n \geq 1\}$.

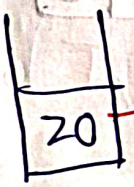
$L = \{ab, aabb, aaabbb, ---\}$.

Logic for construction of PDA:

1. Initially Push all the a's into the stack.
2. On reading every single 'b', Pop one 'a' from the stack.
3. If the input string reached end and stack is empty then string / Language is accepted by PDA.

Initially the stack is Z0 element is top of the stack.

i.e



Stack.

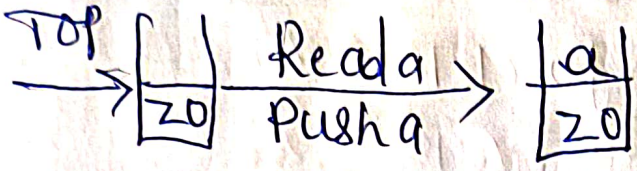
Let $w = aabb$. & Let $q_0 = q_0$

I/p string ⇒ | a | a | b | b | ∈ |

Step 1:

| a | a | b | b | ∈ |

↑ Read: a (1st)

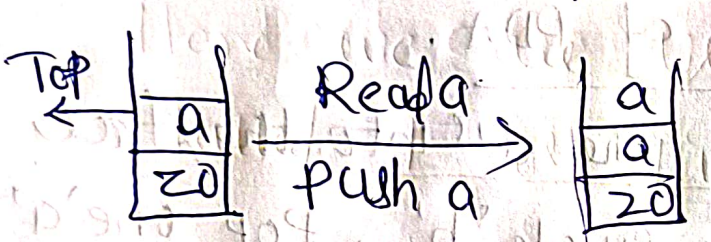


$\delta(q_0, a, Z_0) = (q_0, aZ_0)$

Step 2:

| a | a | b | b | ∈ |

↑ Read 2nd a.

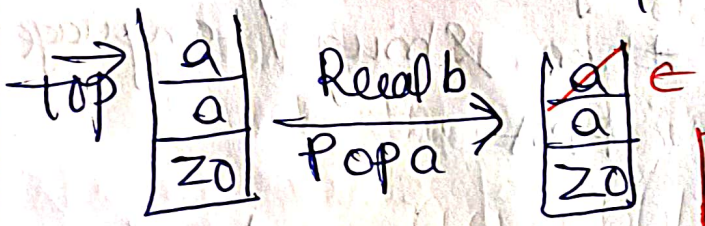


$\delta(q_0, a, a) = (q_0, aaZ_0)$

Step 3:

| a | a | b | b | ∈ |

↑ Read 1st b & (Pop a)

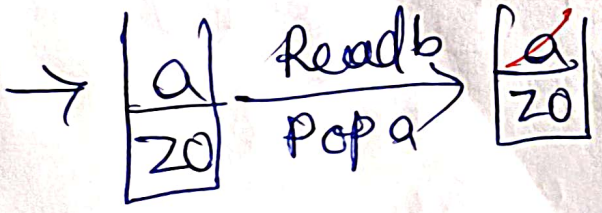


$\delta(q_0, b, a) = (q_1, \in)$

Step 4:

| a | a | b | b | ∈ |

↑ Read 2nd b & Pop a.



$\delta(q_1, b, a) = (q_1, \in)$

Steps:

a | a | b | b | ∈

Top [Z0] → Read ∈ → [Z0]

$\delta(q_1, \epsilon, Z_0) = (q_2, Z_0)$

The stack is empty & the string aabb is accepted.

Transition diagram:

$\delta(q_0, a, Z_0) = (q_0, aZ_0)$

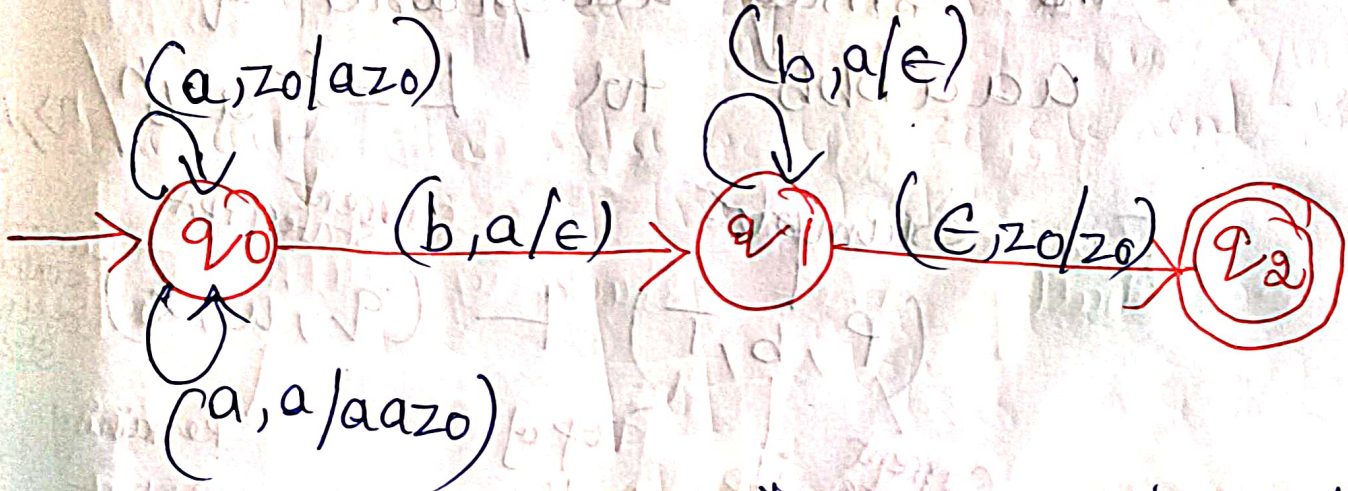
$\delta(q_0, a, a) = (q_0, aaZ_0)$

$\delta(q_0, b, a) = (q_1, \epsilon)$

$\delta(q_1, b, a) = (q_1, \epsilon)$

$\delta(q_1, \epsilon, Z_0) = (q_2, Z_0)$

draw the transition diagram based on the above transitions functions.



q_0 → is initial & q_2 is Final state

Stack symbols (Γ) = {a, Z0}

PDA = M = ({q0, q1, q2}, {a, b}, {a, Z0}, δ, q0, Z0, q2).

Instantaneous description of PDA or ID of PDA :-

String acceptance can be done by using ID.

Here we are using the symbol called tailed ⊢ or Turnstile notation (⊢).

1) ⊢ represents single move
2) ⊢* represents multiple moves / sequence of moves.

Check the string acceptance of aaabbb for L = {a^n b^n / n ≥ 1}

For example:

(p, b, T) ⊢ (q, w, α)

- p : current state
- b : i/p symbol
- T : top of the stack
- q : current state of PDA
- w : Remaining input
- α : Current content of the stack

$\delta(q_0, aaabbb, z_0) \vdash (q_0, aabbb, az_0)$

$\delta(q_0, aabbb, az_0) \vdash (q_0, abbb, aaz_0)$

$\delta(q_0, abbb, aaz_0) \vdash (q_0, bbb, aaaz_0)$

$\delta(q_0, bbb, aaaz_0) \vdash (q_1, bb, aaz_0)$

$\delta(q_1, bb, aaz_0) \vdash (q_1, b, az_0)$

$\delta(q_1, b, az_0) \vdash (q_1, \epsilon, z_0)$

$\delta(q_1, \epsilon, z_0) \vdash (q_2, \epsilon)$

String accepted & Empty stack.

$q_0 \rightarrow$ initial state

$q_2 \rightarrow$ Final state.

PDA = $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$

$M = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, z_0\}, \delta, q_0, z_0, q_2)$

Q2) Design a PDA for Language

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

Soln:

$L = \{ c, aca, bcb, abcba, abbcbba, \_\_\_ \}$.

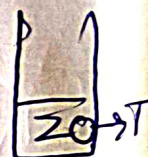
Logic for PDA construction:

1. Push all the 'a's & 'b's until reach the Symbol c
2. Read c and don't do any operation (No change)
3. (i) If input is b and top of the stack is b then pop it. (Remove b).

(ii) If input is a and top of the stack is a then pop it.

Let $w = abcba$.

Initially the stack is with z_0



Dr. K. Bhargavi

Input string

a	b	c	b	a	∈

Step1:

a	b	c	b	a	∈
↑					

Read 'a' & push a.

Top → | z0 | —Read a, Push a→ | a | z0 |

$$\delta(q_0, a, z_0) = (q_0, az_0)$$

Step2:

a	b	c	b	a	∈
	↑				

Read b & push b

Top → | a | z0 | —Read b, push b→ | b | a | z0 |

$$\delta(q_0, b, a) = (q_0, ba)$$

Step3:

a	b	c	b	a	∈
		↑			

Read c & No operation

$$\delta(q_0, c, b) = (q_0, ba)$$

Step4:

a	b	c	b	a	∈
			↑		

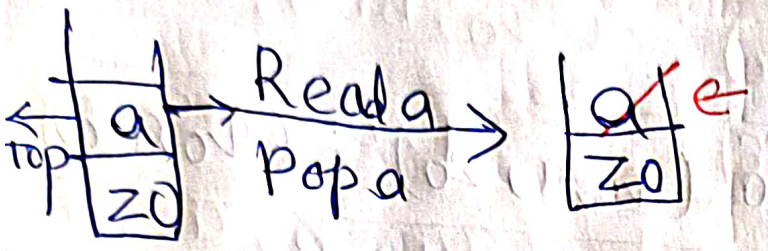
Read b & Top b then pop b

Top → | b | a | z0 | —Read b, pop b→ | b | a | z0 | ∈

$$\delta(q_0, b, b) = (q_0, \in)$$

Step 5: | a | b | c | b | a | ∈ |

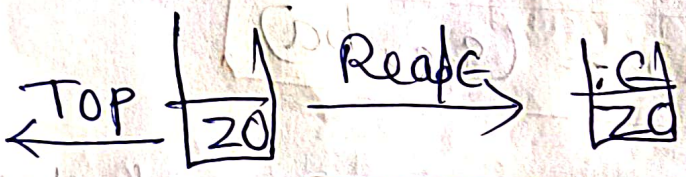
Read a & Top a, then pop a.



$$\delta(q_1, a, a) = (q_1, \in)$$

Step 6: | a | b | c | b | a | ∈ |

Read ∈.



$$\delta(q_1, \in, Z_0) = (q_2, Z_0)$$

Stack is empty with initial symbol Z_0

∴ String is accepted.

$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

$Q = \{q_0, q_1, q_2\}$, $\Sigma = \{a, b, c\}$, $F = q_2$

$\Gamma = \{a, b, Z_0\}$, $q_0 = q_0$, $Z_0 = Z_0$

$$M = (\{q_0, q_1, q_2\}, \{a, b, c\}, \{a, b, Z_0\}, \delta, q_0, Z_0, q_2)$$

apart from step ① & step ② the other possible transition functions are

$\delta(q_0, b, Z_0) = (q_0, bZ_0)$

$\delta(q_0, b, b) = (q_0, bb)$

$\delta(q_0, a, b) = (q_0, ab)$

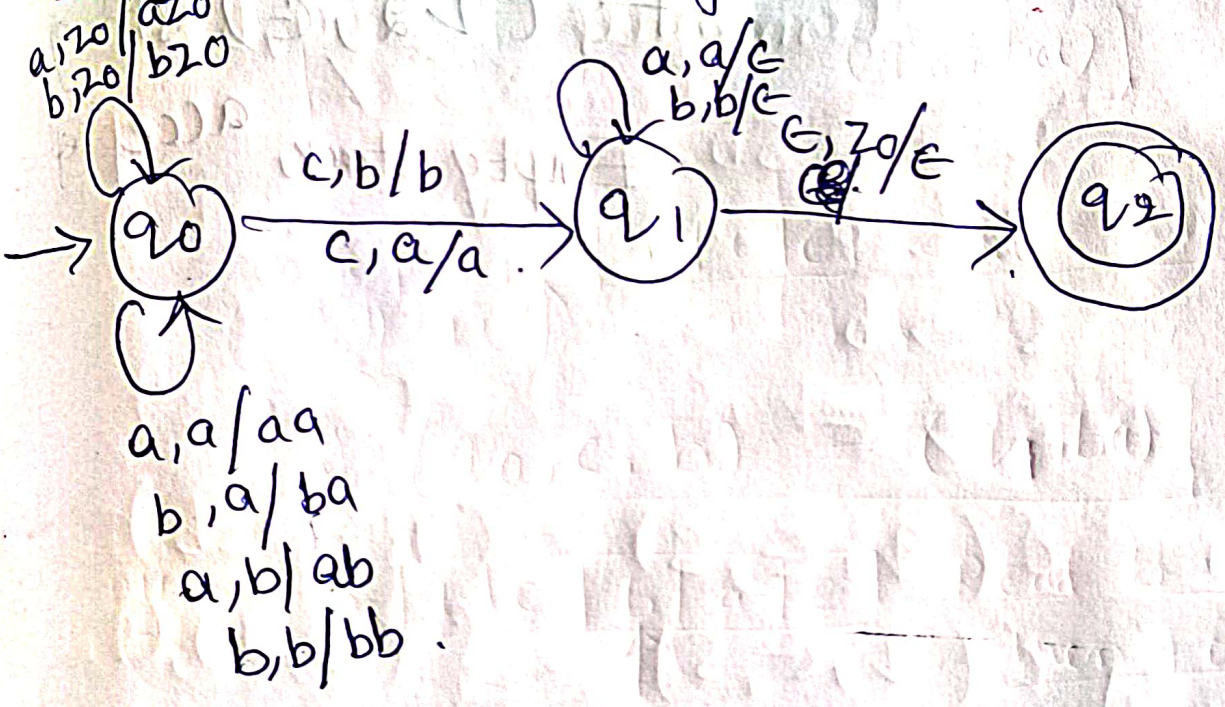
$\delta(q_0, a, a) = (q_0, aa)$

From step ③.

$\delta(q_0, c, Z_0) = (q_1, Z_0)$

$\delta(q_0, c, a) = (q_1, a)$

The transition diagram is.



Dr. K. Bhargavi

Id of PDA for $w = abbcbba$

$(q_0, abbcbba, z_0) \vdash (q_0, bbcbba, az_0)$

$\vdash (q_0, bcbba, bbaz_0)$

$\vdash (q_0, cbba, bbaz_0)$

$\vdash (q_0, bba, bbaz_0)$

$\vdash (q_0, ba, baz_0)$

$\vdash (q_0, a, az_0)$

$\vdash (q_1, \epsilon, z_0)$

$\vdash (q_2, \epsilon)$ string accepted

Empty stack

convertion of CFG to PDA

steps for convertion of CFG to PDA

1. convert CFG productions into GNF (Greibach Normal Form)
2. The PDA will only have one state {q}
3. The initial symbol of CFG will be the initial symbol of PDA.
4. Include the following Rule for non-terminal

$$\boxed{\delta(q, \in, A) = (q, \alpha)}$$

Where production rule is $A \to \alpha$

5. Add the following rule for each terminal symbol:

$$\boxed{\delta(q, a, a) = (q, \in)}$$

GNF: A CFG is in GNF if PR satisfy the

1. start symbol generating $\in$ ex: $S \to \in$
2. A non-terminal generating terminal

Ex: $S \to a$

3. A non-terminal generating terminal, which is followed by any no. of terminals & non terminal. Ex: $S \to aASB$

Q1) Convert the following CFG to PDA

$S \to aSb$

$S \to a/b/\in$ & check string acceptance for aaabb

Soln: The given Productions are in GNF

non terminals → Apply Rule 1 & Rule 2 ← terminals

Rule 1: $\delta(q, \in, A) = (q, \alpha)$ (when $A \to \alpha$)

Rule 2: $\delta(q, a, a) = (q, \in)$

Given $\Sigma = \{a, b\}$

Non terminals $= S$

terminals $= \{a, b\}$

$S \to aSb$

$S \to a$

$S \to b$

$S \to \in$

Rule 1 (Non Terminals): $\delta(q, \in, S) = (q, aSb)$ — R1

$\delta(q, \in, S) = \{(q, a), (q, b), (q, \in)\}$ — R2

Rule 2 (Terminals): $\delta(q, a, a) = (q, \in)$ — R3

$\delta(q, b, b) = (q, \in)$ — R4

$\delta(q, \in, Z_0) = (q, \in)$ — R5

String acceptance for aaabb.

$\delta(q, aaabb, S) \vdash (q, aaabb, aSb)$ (R1)

$\vdash (q, aabb, Sb)$ (R1)

$\vdash (q, aabb, aSbb)$ (R1)

$\vdash (q, abb, Sbb)$ (R2)

$\vdash (q, abb, abb)$ (R2)

$\vdash (q, bb, bb)$ (R4)

$\vdash (q, b, b)$ (R4)

$\vdash (q, \in, Z_0)$ (R5)

$\vdash (q, \in)$

Empty string,

String is accepted.

Q2) Construct PDA for the given CFG & Test Whether 0104 & 010000 accepted by PDA or not. $S \to OBB$, $B \to OS/1S/0$

Soln:

$PDA = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$.

$\Sigma = \{0, 1\}$, Let $q_0 = q_0$, $Z_0 = S$.

$S \to OBB$; $S \to \epsilon$

$B \to 0S/1S/0$.

Non terminals $= \{S, B\}$

Terminals $= \{0, 1\}$ Apply

Rule 1 for Non terminals

Rule 2 for Terminals.

Rule 1 : $\delta(q, \epsilon, A) = (q, \alpha)$

$(\because A \to \alpha)$

R1 : $\delta(q, \epsilon, S) = (q, OBB)$

R2 : $\delta(q, \epsilon, B) = \{(q, 0S), (q, 1S), (q, 0)\}$

Rule 2 : $\delta(q, a, a) = (q, \epsilon)$.

R3 : $\delta(q, 0, 0) = (q, \epsilon)$

R4 : $\delta(q, 1, 1) = (q, \epsilon)$

string acceptance 01^40 or 010000

$\delta(q, 010000, S) \vdash \delta(q, 010000, S)$

$\vdash \delta(q, 010000, 0BB)$ R1

$\vdash \delta(q, 10000, BB)$ R3

$\vdash \delta(q, 10000, 1SB)$ — R2

$\vdash \delta(q, 0000, SB)$ — R4

$\vdash \delta(q, 0000, 0BBB)$ — R1

$\vdash \delta(q, 000, BBB)$ — R3

$\vdash \delta(q, 000, 0BB)$ — R2

$\vdash \delta(q, 00, BB)$ — R3

$\vdash \delta(q, 00, 0B)$ — R2

$\vdash \delta(q, 0, B)$ — R3

$\vdash \delta(q, 0, 0)$ — R2

$\vdash \delta(q, \in)$ — R3

empty stack & String is accepted

PDA = $M = (\{q\}, \{0,1\}, \{S, B, 0, 1\}, \delta, q_0, Z_0, q_2)$.

Dr. K. Bhargavi

Procedure to conversion of PDA to CFG

Let PDA = $(Q, \Sigma, \Gamma, \delta, q_0, Z_0)$

Let PDA accepted by empty stack is given then construct the CFG = (V, T, P, S)

Rules for variable:

V = variables / non terminals /

1. Special symbol S (start symbol)
2. All the other variables can be written by in the form.

$[P X q]$

where P, q are states in Q

X = stack symbol. $(X \in \Gamma)$

Terminals:

T = Terminals / lower case letters /

Set of Alphabets / symbols.

Dr. K. Bhargavi

Production (P):

1. For all states p

$$S \rightarrow [q_0 Z_0 p]$$

q_0 — start state; Z_0 — initial symbol in stack; p — all possible combinations

Push operation:

2. Let $\delta(q, a, X) = (r, Y_1, Y_2 \cdots Y_k)$

q — current state; a — input symbol; X — top of stack

$$[q X r_k] \rightarrow a [r Y_1 r_1] [r_1 Y_2 r_2] \cdots [r_{k-1} Y_k r_k]$$

q — current state; X — top of stack

For all states $r_1, r_2 \cdots r_k$.

Pop operation:

3. $\delta(q, a, x) = (r, \epsilon)$

$$(q, x, r) \Rightarrow a$$

4. $\delta(q, \epsilon, x) = (r, \epsilon)$ // ϵ in input & Pop

$$(q, x, r) \rightarrow \epsilon$$

2Q: Convert the following PDA into CFG

$P = (\{q_0, q_1\}, \{0,1\}, \{X, Z\}, \delta, q_0, Z_0)$

$\delta(q_0, 1, Z_0) = (q_0, XZ_0)$ $\quad \delta(q_0, 0, X) = (q_1, X)$

$\delta(q_0, 1, X) = (q_0, XX)$ $\quad \delta(q_1, 1, X) = (q_1, \epsilon)$

$\delta(q_0, \epsilon, X) = (q_0, \epsilon)$ $\quad \delta(q_0, 0, Z_0) = (q_1,$

Solution:

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

$Q = \{q_0, q_1\}$, $\Sigma = \{0,1\}$, $\Gamma = (X,$

V = Variables / Non Terminals.

$V = \{ S, [q_0 X q_0] [q_1 X q_1] [q_0 X q_1] [q_1 X$

$[q_0 Z q_0] [q_1 Z q_1] [q_0 Z q_1] [q_1 Z$

$\}$

Start state (S)

$S \to [q_0 Z_0 q_1]$

$S \to [q_0 Z_0 q_0]$

Terminals (T) $= \{0, 1\}$

Productions (P):

Consider all the transitions and write the Productions

(i) $\delta(q_0, 1, Z_0) = (q_0, XZ_0)$ — 2 symbols, $2^2 = 4$ productions

$[q_0 Z_0 q_0] \Rightarrow 1\,[q_0 X q_0]\,[q_0 Z_0 q_0]$

$[q_0 Z_0 q_0] \rightarrow 1\,[q_0 X q_1]\,[q_1 Z_0 q_0]$

$[q_0 Z_0 q_1] \rightarrow 1\,[q_0 X q_0]\,[q_0 Z_0 q_1]$

$[q_0 Z_0 q_1] \rightarrow 1\,[q_0 X q_1]\,[q_1 Z_0 q_1]$

(ii) $\delta(q_0, 1, X) = (q_0, XX)$ — $2^2 = 4$

$[q_0 X q_0] \rightarrow 1\,[q_0 X q_0]\,[q_0 X q_0]$

$[q_0 X q_0] \rightarrow 1\,[q_0 X q_1]\,[q_1 X q_0]$

$[q_0 X q_1] \rightarrow 1\,[q_0 X q_0]\,[q_0 X q_1]$

$[q_0 X q_1] \rightarrow 1\,[q_0 X q_1]\,[q_1 X q_1]$

(iii) $\delta(q_0, \epsilon, X) = (q_0, \epsilon)$ — Pop operation

$[q_0 X q_0] \rightarrow \epsilon$

Dr. K. Bhargavi

(iv) $\delta(q_0, 0, x) = (q_1, x)$

$2^1 = 2$ Products one with q_0 & other q_1

$[q_0 x q_0] \to 0\ [q_1 x q_0]$

$[q_0 x q_1] \to 0\ [q_1 x q_1]$

(v) $\delta(q_1, 1, x) = (q_1, \in)$ Pop

$[q_1 x q_1] \to 1$

(vi) $\delta(q_0, 0, z_0) = (q_1, z_0)$

$[q_0 z_0 q_0] \to 0\ [q_1 z_0 q_0]$

$[q_0 z_0 q_1] \to 0\ [q_1 z_0 q_1]$

Convert the PDA into CFG.

Q1: $M = (\{q_0, q_1\}, \{a, b\}, \{a, z_0\}, \delta, q_0, z_0, \phi)$

$\delta(q_0, a, z_0) = (q_0, az_0)$

$\delta(q_0, a, a) = (q_0, aa)$

$\delta(q_0, b, a) = (q_1, a)$

$\delta(q_1, b, a) = (q_1, a)$

$\delta(q_1, a, a) = (q_1, \in)$, $\delta(q_1, \in, z_0) = (q_1, \in)$

Soln: $G_1 = (V, T, P, S)$

V = Set of variables / non terminals. (Q in Γ)

$= \{S, [q_0 a q_0], [q_0 a q_1], [q_1 a q_0], [q_1 a q_1], [q_0 z_0 q_0], [q_0 z_0 q_1], [q_1 z_0 q_0], [q_1 z_0 q_1]\}$

T = Set of Terminals / Lower case letters.

$T = \{a, b\}$

$S \to [q_0 z_0 q_0]$ } start symbol

$S \to [q_0 z_0 q_1]$

Productions:

(i) $\delta(q_0, a, z_0) = (q_0, az_0)$ — 2 symbols. $2^2 = 4$ productions we will get

$[q_0 z_0 q_0] \to a [q_0 a q_0] [q_0 z_0 q_0]$

$[q_0 z_0 q_0] \to a [q_0 a q_1] [q_1 z_0 q_0]$

$[q_0 z_0 q_1] \to a [q_0 a q_0] [q_0 z_0 q_1]$

$[q_0 z_0 q_1] \to a [q_0 a q_1] [q_1 z_0 q_1]$

$\{q_0, q_1\}$ three q_0 & q_1.

(ii) $\delta(q_0, a, a) = (q_0, aa)$ — 2 symbols, $2^2 = 4$

$[q_0\ a\ q_0] \to a[q_0\ a\ q_0]\ [q_0\ a\ q_0]$

$[q_0\ a\ q_0] \to a[q_0\ a\ q_1]\ [q_1\ a\ q_0]$

$[q_0\ a\ q_1] \to a[q_0\ a\ q_0]\ [q_0\ a\ q_1]$

$[q_0\ a\ q_1] \to a[q_0 a\ q_1]\ [q_1\ a\ q_1]$

(iii) $\delta(q_0, b, a) = (q_1, a)$ — one symbol, $2^1 = 2$ Productions

$[q_0\ a\ q_0] \to b[q_1\ a\ q_0]$

$[q_0\ a\ q_1] \to b[q_1\ a\ q_1]$

(iv) $\delta(q_1, b, a) = (q_1, a)$ — // Pushing 2 Producing

$[q_1\ a\ q_0] \to b[q_1\ a\ q_0]$

$[q_1\ a\ q_1] \to b[q_1\ a\ q_1]$

(v) $\delta(q_1, a, a) = (q_1, \in)$ — // Pop

$[q_1\ a\ q_1] \to a$

(vi) $\delta(q_1, \in, z_0) = (q_1, \in)$

$[q_1\ z_0\ q_1] \to \in$

Construct the PDA for the following.

1. $L = \{ a^n b^{2n} / n \geq 1 \}$
2. $L = \{ a^n b^m c^n / m, n \geq 1 \}$
3. $L = \{ a^n b^m c^m d^n / n \geq 1 \}$
4. $L = \{ w w^R / w \in (a, b)^* \}$