UNIT III

CONTEXT FREE GRAMMAR AND LANGUAGES

CONTENTS

CONTEXT FREE GRAMMAR (CFG)

- A CFG is a way of describing languages by recursive rules called productions.
- A CFG consists of a set of variables, a set of terminal symbols, and a start variable, as well as the productions.
- Each production consists of a start variable and a body consisting of a string of zero or more variables and /or terminals.

DEFINITION

A CFG is denoted as G = (V, T, P, S)

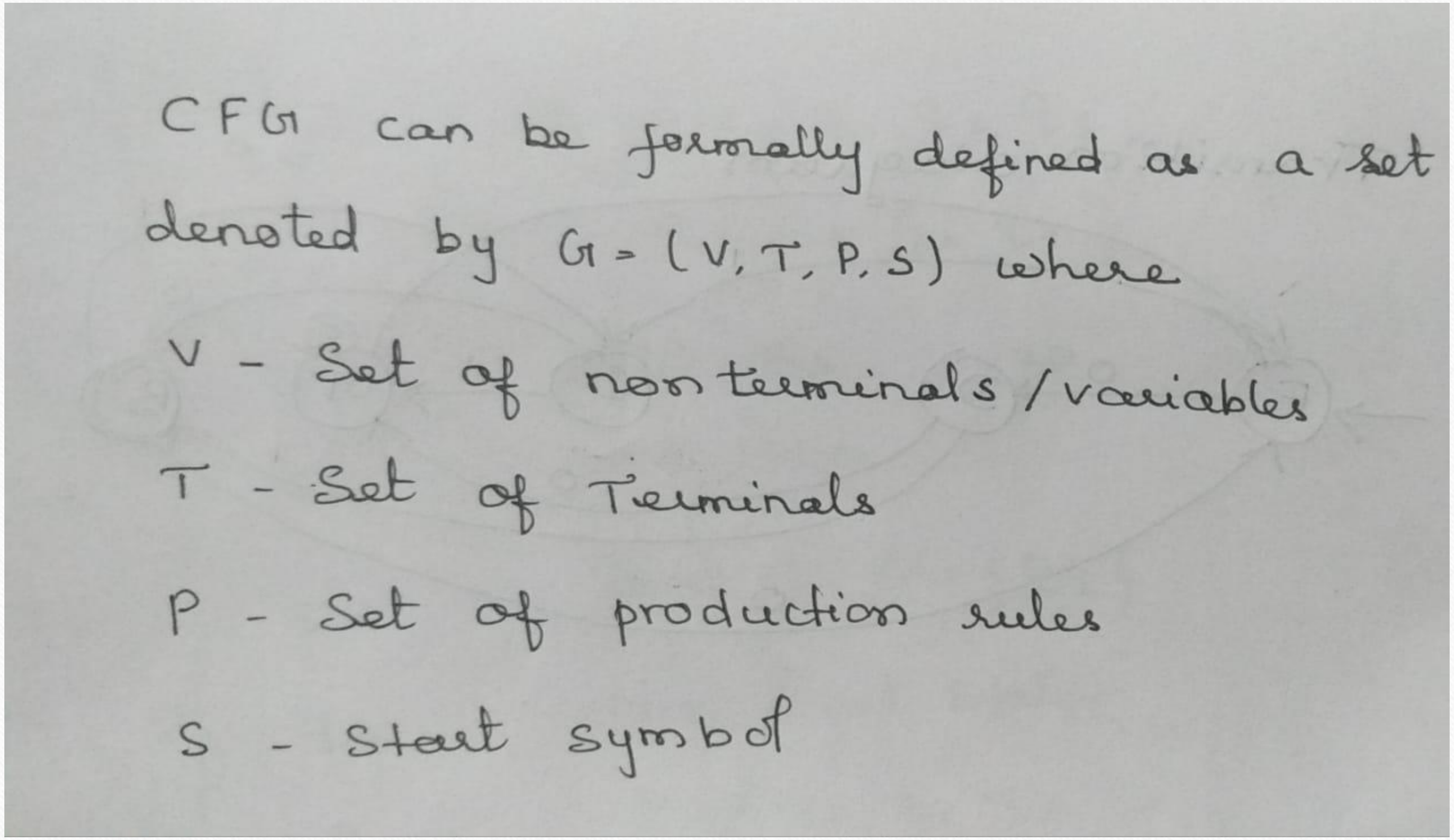
Where,

V →finite set of variables

T → finite set of terminals

P → finite set of productions

S → Start symbol



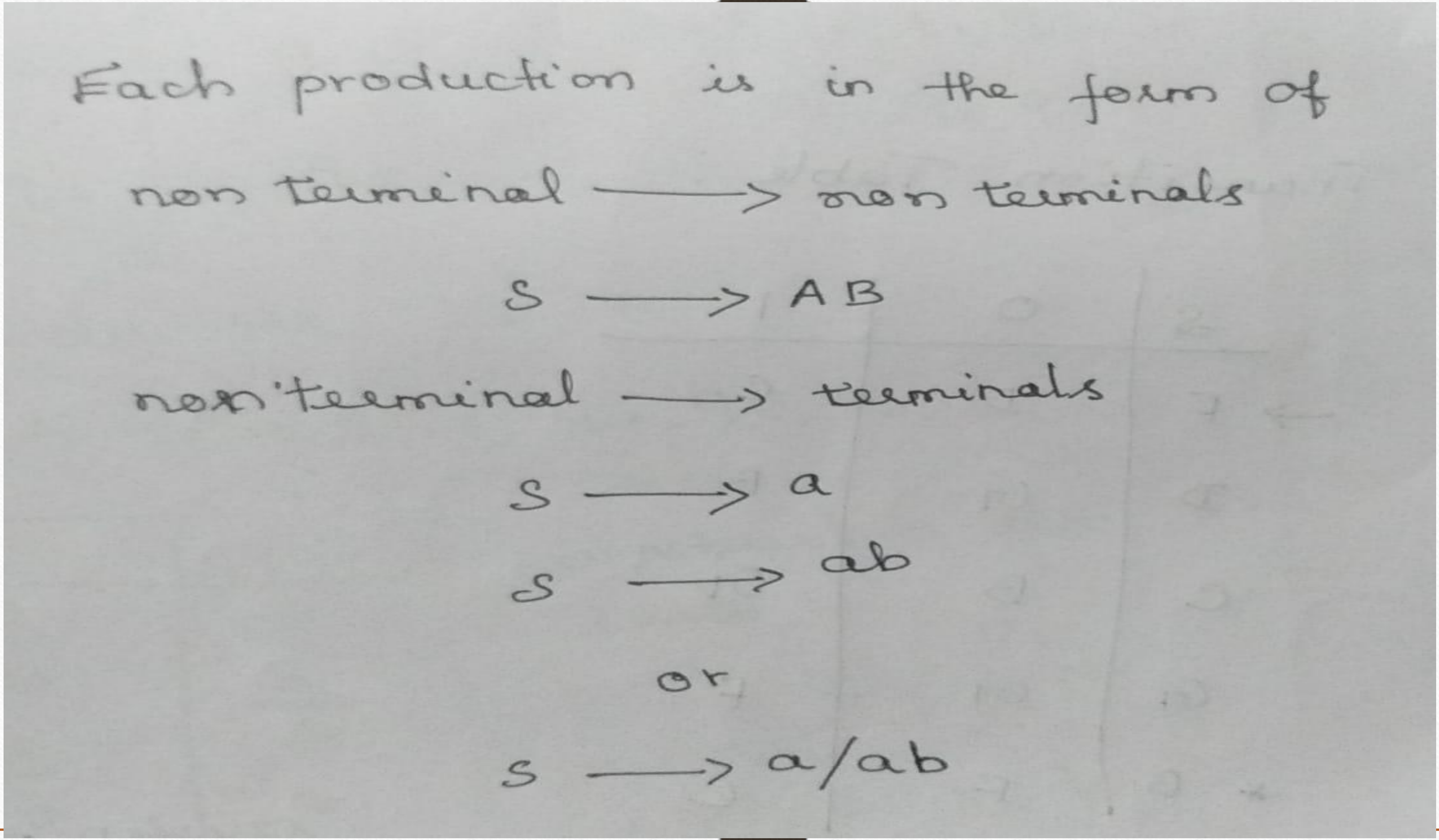
CFG can be formally defined as a set denoted by G = (V, T, P, S) where

V - Set of non terminals / variables

T - Set of Terminals

P - Set of production rules

S - Start symbol



Each production is in the form of

non terminal $\longrightarrow$ non terminals

$S \longrightarrow AB$

non terminal $\longrightarrow$ terminals

$S \longrightarrow a$

$S \longrightarrow ab$

or

$S \longrightarrow a/ab$

USES OF CFG LANGUAGE

- **Defining programming languages.**
- **Formalizing the notion of parsing.**
- **Translation of programming languages.**
- **String processing applications.**

USES OF CONTEXT FREE GRAMMARS

- Construction of compilers.
- Simplified the definition of programming languages.
- Describes the arithmetic expressions with arbitrary nesting of balanced parenthesis
- Describes block structure in programming languages.
- Model neural nets

LANGUAGE GENERATED BY CFG

- The language generated by Grammar G is L(G)

  L(G) = {w | w in T* and S *=>w }
- (1) The string contains only terminals.
- (2) The string can be derived from start symbol S.

Sentential Form

- Any step in a derivation is a string of terminal and / or variables.
- We call such a string a sentential form.

Left and right sentential form

Left Sentential Form:

- If the string a can be generated from the starting symbol by using left most derivation, such that S==> α is left sentential form.

Right Sentential Form:

- If the string a can be generated from the starting symbol by using rightmost derivation, such that S==> α is right sentential form.

Derivation

- Let G = (V, T, P, S) be the context free grammar.

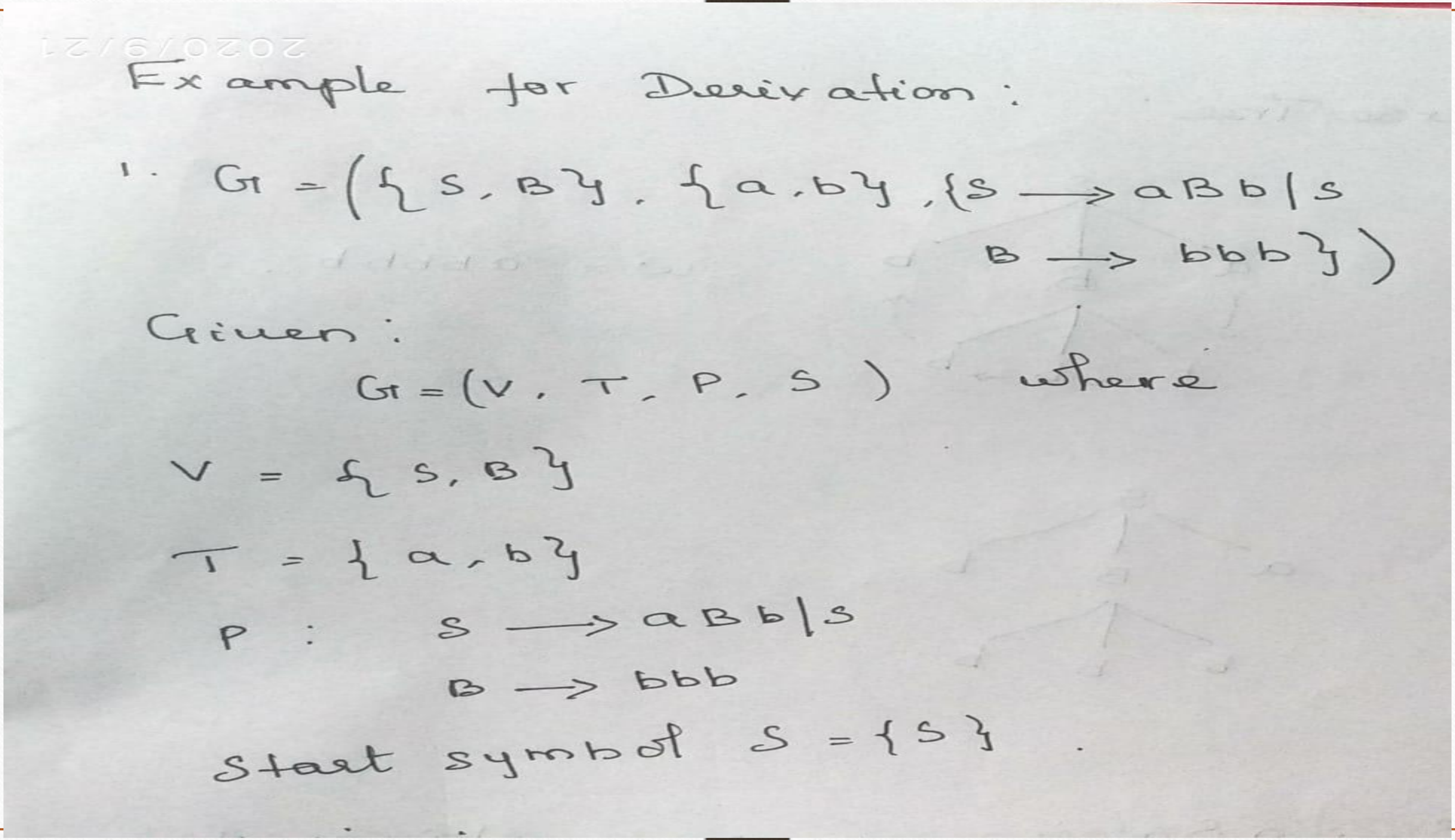
- Beginning with the start symbol, we derive terminal strings by repeatedly replacing a variable by the body of some production with that variable in the head.
- If $A \rightarrow \beta$ is a production of P and
- a and b are any strings in (VUT)* then
- $\alpha\ A\ \gamma \rightarrow \alpha\ \beta\ \gamma$

SUB TREE

- A subtree of a derivation tree is a particular vertex of the tree together with all its descendants, the edges connecting them and their labels.
- The label of the root may not be the start symbol of the grammar

PARSE TREES

- The strings that are derived from the CFG can be represented in a tree format known as Parse tree or derivation tree.
- Types:
  - Leftmost derivation Tree
  - Rightmost derivation Tree



Example for Derivation :

1. $G = (\{S, B\}, \{a, b\}, \{S \longrightarrow aBb \mid S$
$B \longrightarrow bbb\})$

Given :

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

$V = \{S, B\}$

$T = \{a, b\}$

$P : \quad S \longrightarrow aBb \mid S$
$\quad\quad B \longrightarrow bbb$

Start symbol $S = \{S\}$.

Solution :

Derive a String "abbbb."

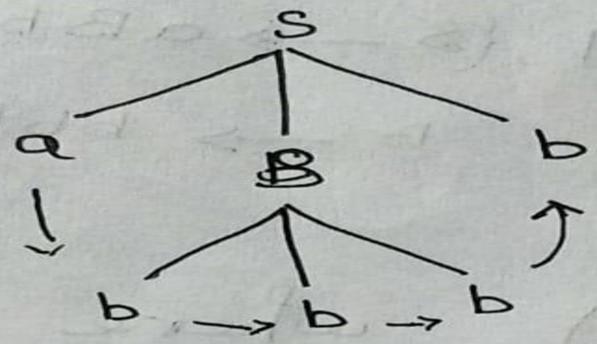
Start with production rule $S \rightarrow aBb$

$S \Rightarrow a\underline{B}b$ $\qquad\qquad S \rightarrow aBb$

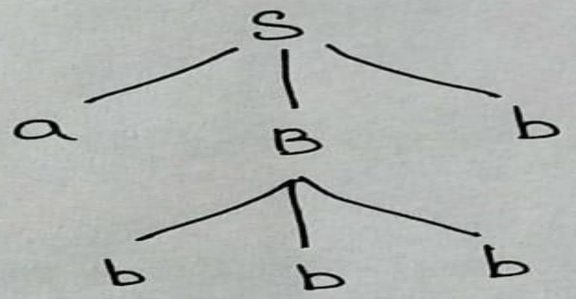
$\Rightarrow a\underline{bbb}\,b$ $\qquad\qquad B \rightarrow bbb$

$\Rightarrow abbbb$

Parse Tree :



$\omega = abbbbb$.



2. $S \rightarrow aSa$ Derive a string abcba.

$S \rightarrow bSb$ $w = abcba$.

$S \rightarrow c$

Given:

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

$V = \{S\}$, $T = \{a, b, c\}$, P: $S \rightarrow aSa \mid bSb$

$S \rightarrow c$.

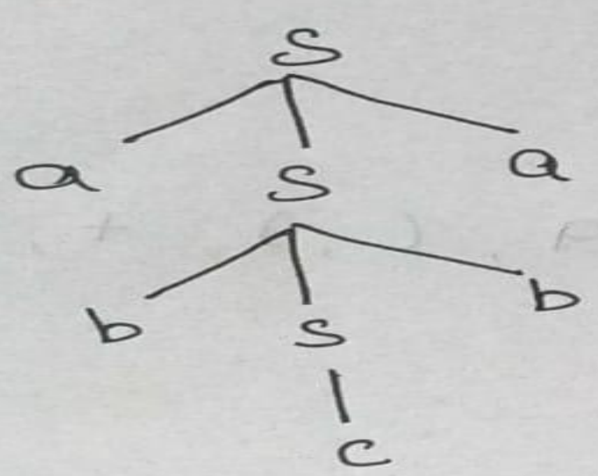
Derivation :

$S \Longrightarrow a\underline{S}a$ $\qquad$ $S \longrightarrow aSa$

$\Longrightarrow a\underline{bSb}a$ $\qquad$ $S \longrightarrow bSb$

$\Longrightarrow abcba$ $\qquad$ $S \longrightarrow c$

Parse tree :



LEFTMOST AND RIGHTMOST DERIVATION

LMD:

The leftmost derivation is a derivation in which the leftmost non terminal is replaced first from the sentential form.

RMD:

The rightmost derivation is a derivation in which ightmost non terminal is replaced first from

Example :

1. $E \longrightarrow D \mid (E) \mid E+E \mid E*E \mid E/E \mid E-E \mid E$

$D \longrightarrow 0 \mid 1 \mid 2 \mid 3 \ldots 9$

String to be derived $w = 1+2*3$.

Given :

$V = \{E, D\}$

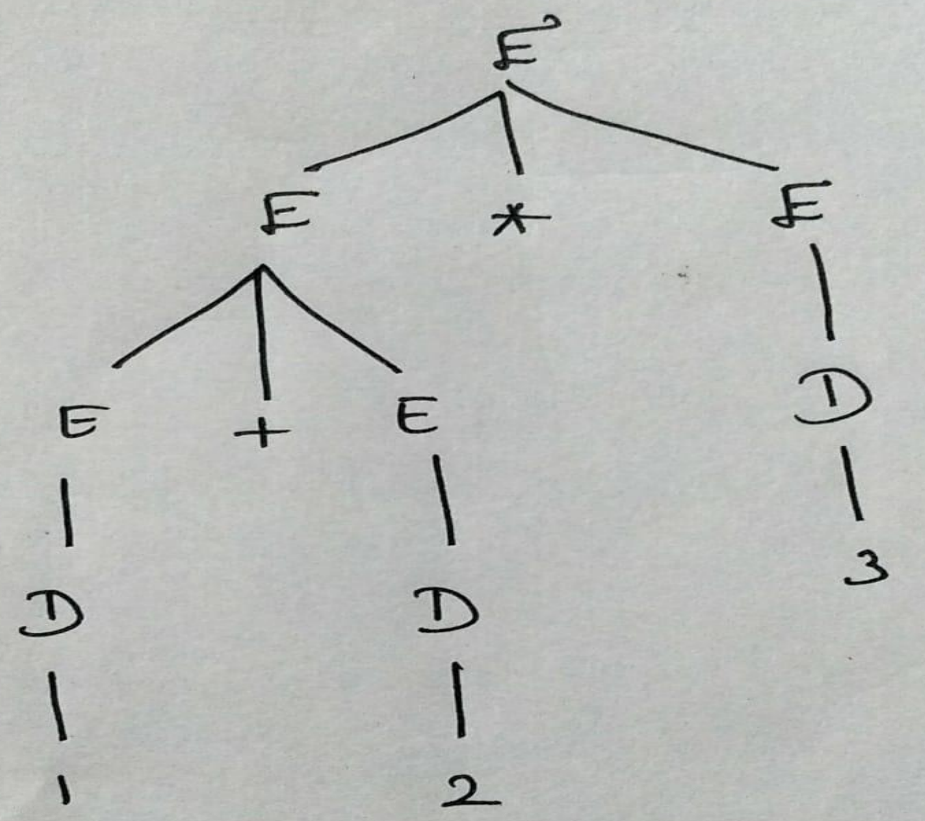
$T = \{0, 1, 2 \ldots 9, (,), +, *, /, -\}$

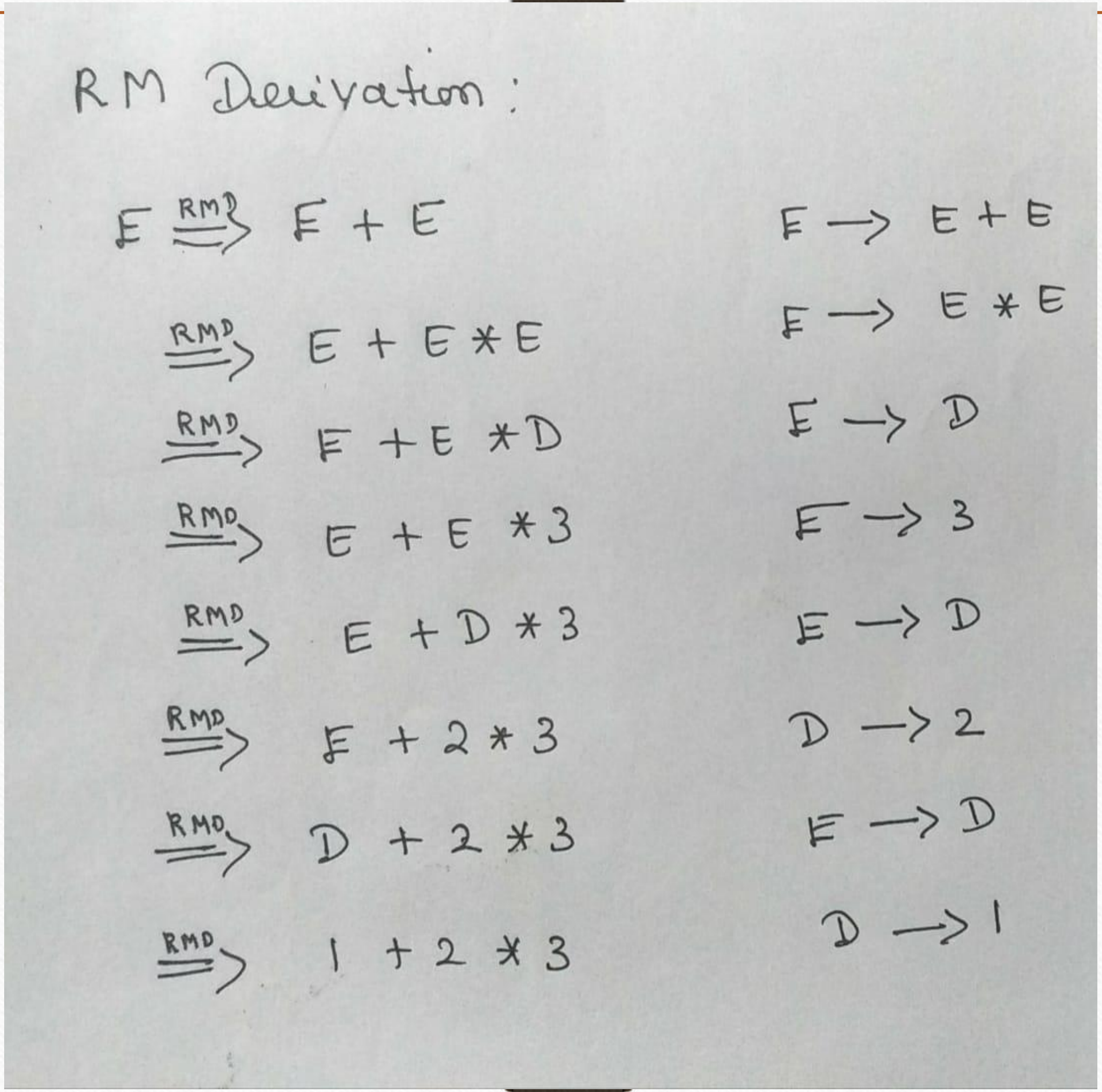
Solution :

LM Derivation :

$E \xRightarrow{LMD} \underline{E} * E$	$E \longrightarrow E * E$
$\xRightarrow{LMD} \underline{E + E} * E$	$E \longrightarrow E + E$
$\xRightarrow{LMD} \underline{D} + E * E$	$E \longrightarrow D$
$\xRightarrow{LMD} 1 + \underline{\underline{E}} * E$	$D \longrightarrow 1$
$\xRightarrow{LMD} 1 + \underline{\underline{D}} * E$	$E \longrightarrow D$
$\xRightarrow{LMD} 1 + \underline{\underline{2}} * \underline{E}$	$D \longrightarrow 2$
$\xRightarrow{LMD} 1 + 2 * \underline{\underline{D}}$	$E \longrightarrow D$
$\xRightarrow{LMD} 1 + 2 * \underline{\underline{3}}$	$D \longrightarrow 3$

LMD Parse Tree :



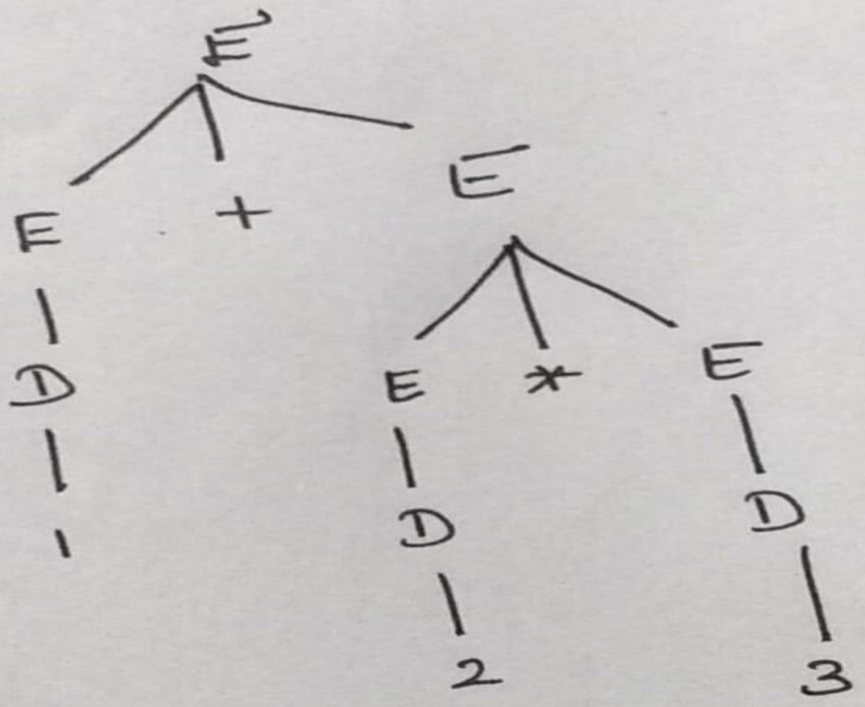


RM Derivation :

$E \xRightarrow{RMD} E + E$	$E \rightarrow E + E$
$\xRightarrow{RMD} E + E * E$	$E \rightarrow E * E$
$\xRightarrow{RMD} E + E * D$	$E \rightarrow D$
$\xRightarrow{RMD} E + E * 3$	$E \rightarrow 3$
$\xRightarrow{RMD} E + D * 3$	$E \rightarrow D$
$\xRightarrow{RMD} E + 2 * 3$	$D \rightarrow 2$
$\xRightarrow{RMD} D + 2 * 3$	$E \rightarrow D$
$\xRightarrow{RMD} 1 + 2 * 3$	$D \rightarrow 1$

Rightmost Derivation

Parse Tree :



Example For Leftmost And Rightmost Derivations

- Derive the strings a*(a+b00) using leftmost and rightmost derivation for the following production.
- 1. E →I   2. E →E+E   3. E → E*E   4. E → (E)
- 5. I → a   6. I → b   7. I → Ia   8. I → Ib
- 9. I → I0   10. I → I1

AMBIGUITY IN GRAMMARS AND LANGUAGES

- A grammar is said to be ambiguous if it has more than one derivation trees for a sentence or in other words if it has more than one leftmost derivation or more than one rightmost derivation.

- A string w is derived ambiguously in context-free grammar G if it has two or more different leftmost derivations. Grammar G is ambiguous if it generates some string ambiguously.

- Sometimes an ambiguous grammar can be transformed into an unambiguous grammar for the same language.
- Some context-free grammars can be generated only by ambiguous grammars. These are known as inherently ambiguous languages.
- $L = \{ a^i b^j c^k \mid i = j \text{ or } j = k \}$
- A context free language L is said to be inherently ambiguous if all its grammars are ambiguous grammar.
- If even one grammar for L is unambiguous then L is an unambiguous language.
- Inherent ambiguous grammars are one for which unambiguous grammars do not exist.

Show that the grammar $S \rightarrow a \mid abSb \mid aAb$, $A \rightarrow bS \mid aAAb$ is ambiguous.

DEFINITION OF THE PUSHDOWN AUTOMATA

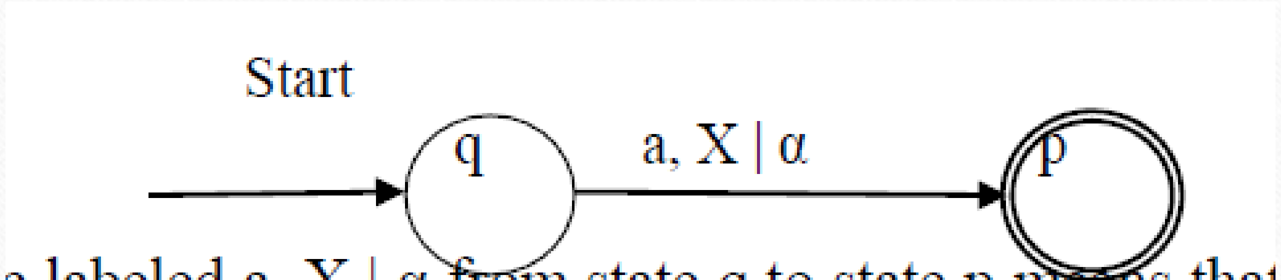
- A PDA is a nondeterministic finite automaton coupled with a stack that can be used to store a string of arbitrary length.
- The stack can be read and modified only at its top.

A PushDown Automata M is a system (Q, Σ, Γ, δ, q_0, Z_0, F)
Where,

- ✓ Q is a finite set of states.
- ✓ Σ is an alphabet called the input alphabet.
- ✓ Γ is an alphabet called stack alphabet.
- ✓ q_0 in Q is called initial state.
- ✓ Z_0 in Γ is start symbol in stack.
- ✓ F is the set of final states.
- ✓ δ is a mapping from Q X (Σ U {Є}) X Γ to finite subsets of Q X Γ*.

Transition Diagram and Transition Function of PDA

- **The transition diagram for PDA's in which**
- The nodes correspond to the states of the PDA
- An arrow labeled Start indicates the start states, and doubly circled states are accepting as for finite automata.
- **Transition Function: δ (q, a, X) = (p, α)**



c labeled a, X | α from state q to state p means that

Moves of Pushdown Automata

- A PDA chooses its next move based on its current state, the next input symbol, and the symbol at the top of its stack.
- It may also choose to make a move independent of the input symbol and without consuming that symbol from the input.

Types of moves in PDA

- The move dependent on the input symbol (a) scanned is:
- $\delta(q, a, Z)=\{ (p1, \gamma1), (p2,\gamma2),\ldots..(pm,\gamma m) \}$ Where
- q and p are states,
- a is in Σ,
- Z is a stack symbol and
- γ_i is in Γ^*.
- PDA is in state q, with input symbol a and Z the top symbol on state enter state pi and replace symbol Z by string γ_i

Types of moves in PDA

- The move independent on input symbol is **(Є-move):**
- δ(q,Є,Z) = { (p1,γ1), (p2,γ2),…………(pm,γm) }
- Is that PDA is in state q,
- independent of input symbol being scanned and with Z the top symbol on the stack
- enter a state pi and
- replace Z by γi.

Types of language acceptances by a PDA

- 1. Acceptance by Final State
- 2. Acceptance by Empty Stack

For a PDA M=(Q, Σ ,Γ ,δ ,q0 ,Z0 ,F) we define :

1. Acceptance by Final State:
L(P)={ w | (q0 , w , Z0) ├ (p, Є , γ) for some p in F and any stack string γ }.

2. Acceptance by Empty Stack:
N(P) = { w | (q0, w, Z0) ├ (p, Є, Є) for any state q}.

Is it true that the language accepted by a PDA by empty stack and final states are different languages?

No, because the languages accepted by PDA‘s by final state are exactly the languages accepted by PDA’s by empty stack.

Instantaneous Description (ID) in PDA

- ID represents the configuration of a PDA by a triple (q, w, γ),
- Where,
- □ q is the state,
- □ w is the remaining input, and
- □ γ is the stack contents.

- ID consisting of the state, remaining input, and stack contents to describe the “current condition” of a PDA.
- A transition function ├ between ID’s represents single moves of a PDA.

- If M = (Q, Σ, Ґ, δ, q0, Z0, F) be a PDA. If δ (q, a, X) contains (p, α).
- Then for all string w in Σ* and β in Γ*.
  - **(q, aw, X β) ├ (p, w, αβ)**
- This move reflects the idea that, by consuming ***a*** from the input and replacing X on top of the stack by ***α*,** we can go from state q to state p.

Significance of PDA

- Finite Automata is used to model regular expression and cannot be used to represent non regular languages.
- Thus to model a context free language, a Pushdown Automata is used.

String accepted by a PDA

The input string is accepted by the PDA if:

- **The final state is reached.**
- **The stack is empty.**

Examples of languages handled by PDA

- (1) L = $\{a^n b^n \mid n>=0\}$, here n is unbounded, hence counting cannot be done by finite memory. So we require a PDA, a machine that can count without limit.
- (2) L = $\{ww^R \mid w \in \{a,b\}^*\}$, to handle this language we need unlimited counting capability .

Is NPDA (Nondeterministic PDA) and DPDA (Deterministic PDA) equivalent?

- The languages accepted by NPDA and DPDA are not equivalent.
- Example:
- ww^R is accepted by NPDA and not by any DPDA.

State the equivalence of acceptance by final state and empty stack

- If L = L(M2) for some PDA M2 , then L = N(M1) for some PDA M1.
- If L = N(M1) for some PDA M1 , then L = L(M2) for some PDA M2.
- Where,
- L(M) = language accepted by PDA by reaching a final state.
- N(M) = language accepted by PDA by empty stack.

Problem : Design PDA to accept the language L={wcw^R / w={0,1}*}

5.1 Non Recursive Enumerable (RE) Languages

- A language is said to be recursive if there exists a turing machine that accepts every string of the language and every string is rejected if it is not belonging to that language.

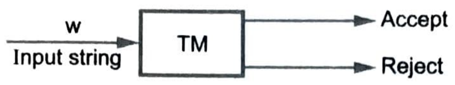


Fig. 5.1.1 Recursive languages

- A language is recursively enumerable if there exists a turing machine that accepts every string belonging to that language. And if the string does not belong to that language then it can cause a turing machine to enter in an infinite loop.

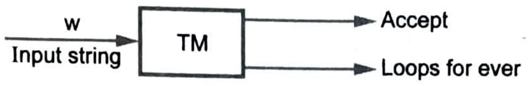


Fig. 5.1.2 Recursively enumerable languages

5.2 Properties of Recursive and Recursively Enumerable Languages

- The recursively enumerable (RE) languages are categorized into two classes :
  1. The class of languages that has turing machine. This turing machine decides whether the input string belongs to that language or not. Such a turing machine always halts, whether or not it reaches to accept state.
  2. The second class of languages consists of those RE languages that are not accepted by any turing machine with the guarantee of halting.
- A language is L actually denoted by L(M) called **recursive**. If it is accepted by some turing machine such that
  1. If string W is in L, turing machine M accepts it and then halts.
  2. If W is not in L, then M eventually halts although it never enters in accepting states.
- We can also call recursive languages as the **definite languages** or the languages which can be represented by some **algorithm** and such an algorithm helps in construction of turing machine for that language.

Decidable and Undecidable Languages :

- If a language is **recursive** then it is called **decidable languages** and if the language is **not recursive** then such a language is called **undecidable** language.
- Hence broadly there are three categories of the languages.
  1. Recursive language for which the algorithm exists.
  2. Recursively enumerable language for which it is not sure that on which input the TM will ever halt. Such languages are not recursive.

3. The non-recursively enumerable languages for which there is no turing machine at all.

- Fig. 5.2.1 shows the relationship between these languages.

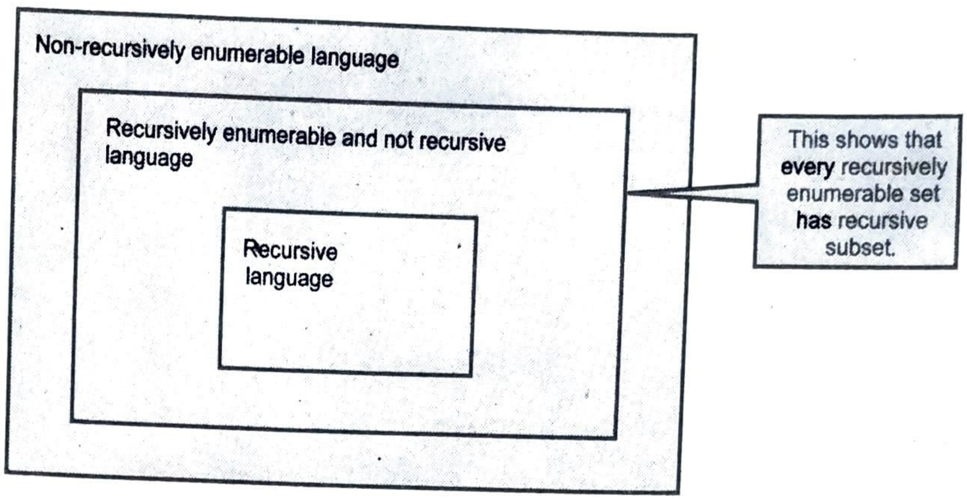

Non-recursively enumerable language
Recursively enumerable and not recursive language
Recursive language
This shows that every recursively enumerable set has recursive subset.

Fig. 5.2.1 Relationship between languages

- The important fact about recursive and recursively enumerable languages can be seen with the help of following theorem.

Theorem 1 : If L is recursive language then L' is also a recursive language.

Proof :

Let there will be some L that can be accepted by turing machine M. Hence we can denote language L by L(M). On acceptance of L(M) the machine M always halts. Now, we construct a TM M' such that L' = (M'). for construction of M' following steps are followed :

1. The accepting steps of M are made non-accepting states of M' and there is no transition from M'. That means we have created the states such that M' will halt without accepting.
2. Now create a new accepting state for M' say r and there is no transition from r.
3. In machine M, for each of the transition with combination of nonaccepting state and input tape symbol, make the same transition having the combination of accepting state and input tape symbol for machine M'.

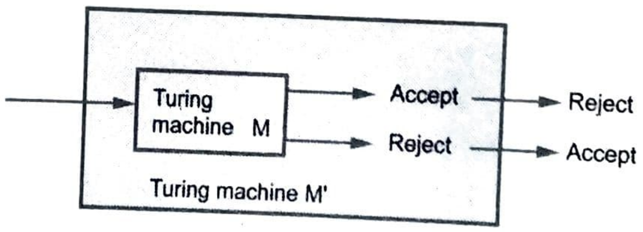


Fig. 5.2.2 Construction of M ' accepting L'

Since M is guaranteed to halt M' is also guaranteed to halt. In fact, M' accepts exactly those strings that M does not accept. Thus we can say that M' accepts L'.

Theorem 2 : If a language L and its complement L' both are RE then L is a recursive language.

Proof :

Consider a turing machine M made up of two turing machines M1 and M2. The machine M2 is complement of machine M1. We can also denote that L(M) = L(M1) and L(M2). Both M1 and M2 are simulated in parallel by machine M. Machine M is a two tape TM, which can be made one tape TM for the ease of simulation. This One tape then will consist of tape of machine M1 and machine M2. The states of M consists of all the states of machine M1 and all the states of machine M2. The machine M made up of M1 and M2 is as shown in Fig. 5.2.3.

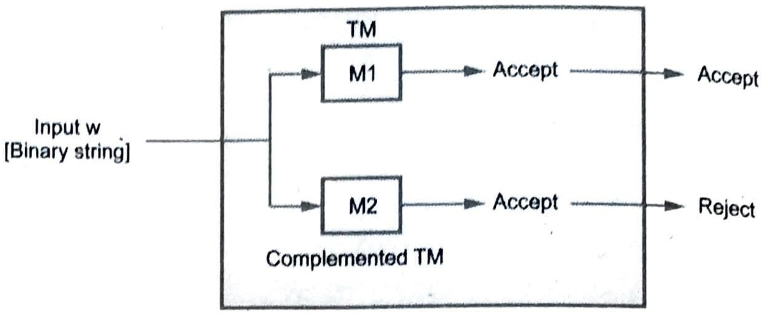


Fig. 5.2.3

If the input W of language L is given to M then M1 will eventually accept and therefore M will accept L and halt. If w is not in L, then it is in L'. So M2 will accept and therefore M will half without accepting. Thus on all inputs, M halts. Thus L(M) is exactly L. Since M always halts we can conclude that L(M) mean L is a recursive language. Thus a recursive language can be recursively enumerable but a recursively enumerable language is not necessarily be recursive.

Theorem 3 : Show that if L_1 and L_2 are recursive languages then $L_1 \cup L_2$ and $L_1 \cap L_2$ also recursive.

Proof : Let,

L_1 is a recursive language.

L_2 is a recursive language.

As L_1 and L_2 are recursive languages there exists a machine M_1 that accepts L_1 as well as machine M_1 that accepts L_2. Now, we have to simulate a machine (algorithm) M that accepts the language L such that

$L = L_1 \cup L_2$. Then construct machine M which accepts if M_1 accepts. The construction of M is as shown in following Fig. 5.2.4.

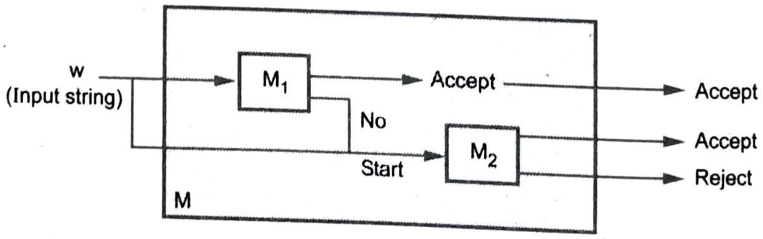
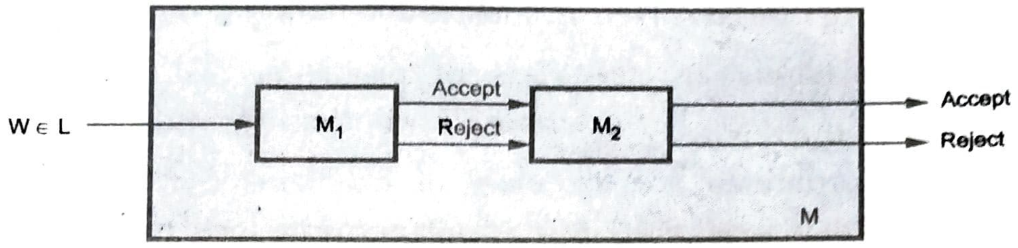


Fig. 5.2.4

If machine M_1 does not accept then M_2 simulates M. That means if M_2 accepts then M accepts, if M_2 rejects M also rejects. Thus M accepts the language $L = L_1 \cup L_2$ which is recursive.

To prove $L_1 \cap L_2$ as a recursive language consider machine M that accepts the language L such that $L = L_1 \cap L_2$.

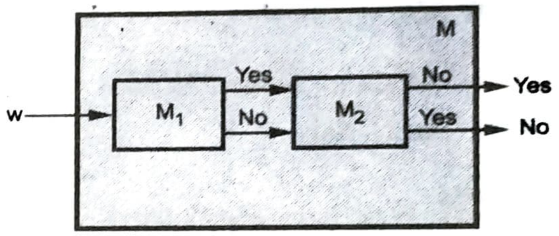


M_1 is a TM which accepts L_1 and M_2 is a TM which accepts L_2. The TM M is simulated which halts on accepting $w \in L$ such that $L = L_1 \cap L_2$. Hence intersection of two recursive languages is also recursive.

Theorem 4 : If L1 and L2 are two recursive languages and if L is defined as : L = {w | w is in L1 and not in L2 and not in L1}. Prove or disprove that L is recursive.

Proof :

- Let L1 is a recursive language which is accepted by M1 and L2 is a recursive language which is accepted by M2. Languages L1 and L2 are such that if $w \in L1$ then $w \notin L2$. Similarly if $w \notin L1$ then $w \in L2$.
- Find $L = L1 \cup L2$. We can then design a TM M which accepts the language L. Such a TM can be drawn as follows :



- As the language L is accepted by TM M, we can say that the language L is recursive language.

Theorem 5 : Show that the set of languages L over {0, 1} so that neither L nor L' is recursively enumerable in uncountable.

Proof :

- If there is any language L which is recursively enumerable then there exists a TM which semidecides it (either accepts and halt or loops for ever).
- Every TM has description of finite length. Hence number of TM and number of recursively enumerable languages is countably. infinite, because power set of every countable set is countably infinite. Now, for the language L its complement L' is RE. Then for L' also there exists TM which semidecides it. But there are uncountable number of languages. Hence there may be language L and L' which are not recursively enumerable. And there may be uncountable number of such languages. Hence neither L nor L' is RE in uncountable.

5.3 Tractable and Intractable Problems **AU : Dec.-15, 16, May-17, Marks 16**

- The class of solvable problems is known as **decidable problems**. That means decidable problems can be solved in measurable amount of time or space.
- The **tractable problems** are the class of problems that can be solved within reasonable time and space. For example - Sorting a list, multiplication of integers.
- The **intractable problems** are the class of problems that can be solved within polynomial time. For example - List of all permutations of n numbers, Tower of Hanoi. This has lead to two classes of solving problems - **P and NP class** problems.

UNIT-V

HALTING PROBLEM

HALTING PROBLEM

~~[illegible]~~ O/P of TM:
i) Halt ii) No halt.

Halt:
M/c will halt after a finite no. of sts

No Halt:
M/c never reaches a halt state, no matter how long it runs.

Halting Problem:
Given any func. matrix, i/p data tape & initial configuration, then is it possible to determine whether the process will ever halt?
This is called halting problem.

Is there any procedure which enable us to solve the halting problem for every pair (machine, tape).
the answer is "no".

That is the halting problem is unsolvable.

Proof :

why ~~T~~ Halting problem is unsolvable ?

Turing m/c M_1 $\Downarrow$ decides whether any computation by a TM T will ever halt

$\Downarrow$

i/p - (t, d_T) is given

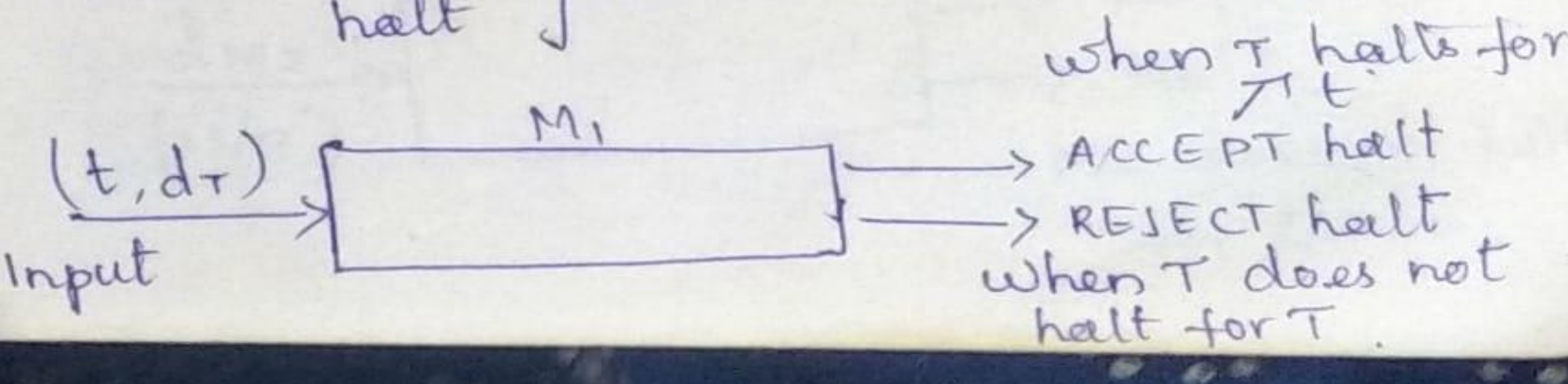
d_T - description of T (i/p to m/c M_1)

t - tape of T (tape)

* for the i/p (t, d_T) to M_1

=> if T halts => M_1 also halts => Accept halt

=> If T does not halt } => M_1 also halts => Reject halt



* Another TM M_2.

$$\Uparrow$$

I/p - d_T.

* first * Copies d_T,
  * duplicates d_T on its tape
* Duplicated tape information is given as i/p to m/c M_1.
* M/c M_1 is a modified m/c with the modification.

$$\Downarrow$$

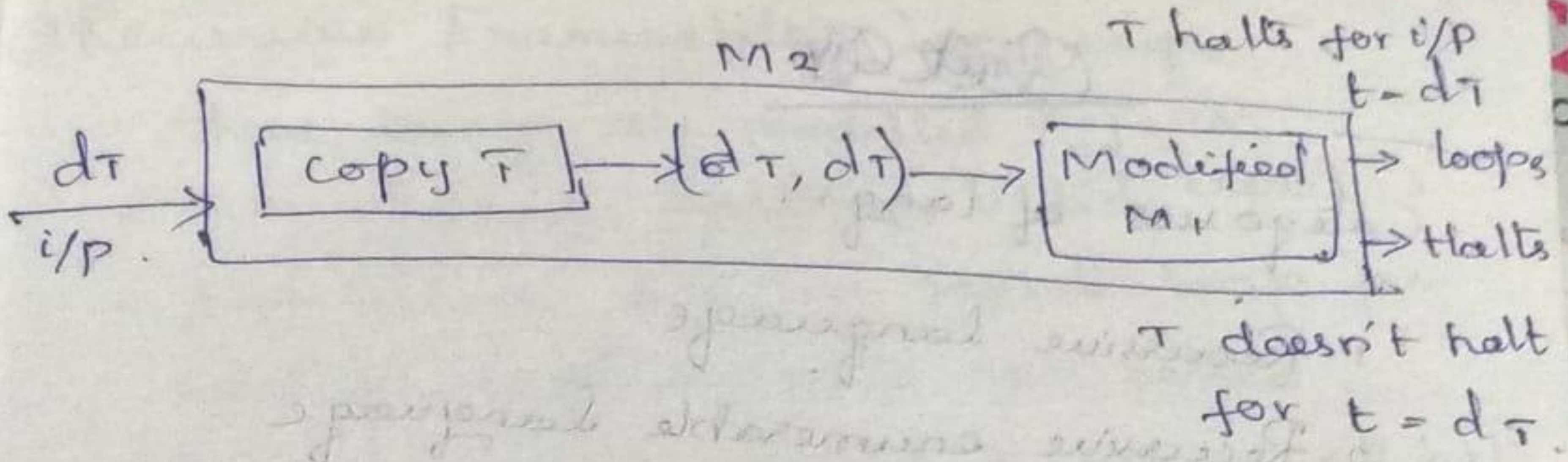
whenever M_1 is supposed to reach an accept halt, M_2 loops forever.

* It loops if T halts for i/p $t = d_T$.
* It halts if T does not halt for $t = d_T$.

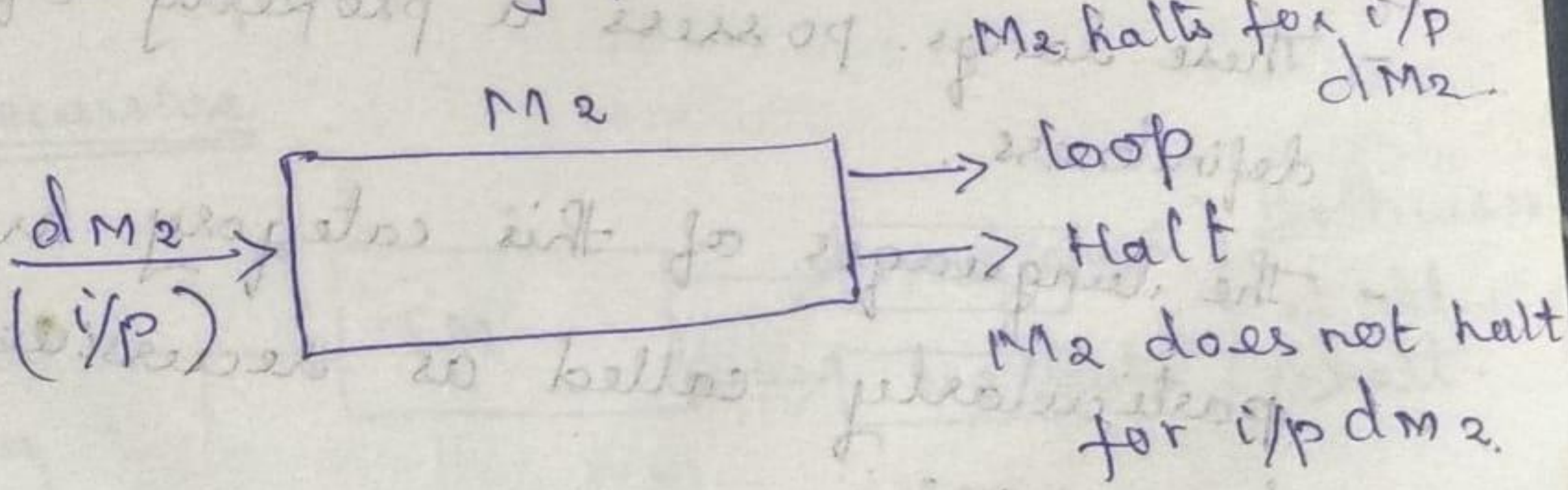
T is any orbitary TM.

d

i/



* $M_2 = T$
* Replace T by M_2
* M_2 halts for i/p d_{M_2} if M_2 does not halt for i/p d_{M_2}.
* This is a contradiction.
* M/c M_1 can tell whether any other TM will halt on particular i/p does not exist.
* Hence halting problem is unsolvable.



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

Solution :

1) Assume that given language

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

2) Find the no. of states of L.

no. of states = ? $\underset{\downarrow_1}{\underline{a}}^n \ \underset{\downarrow_1}{\underline{b}}^n$.

$n + n = 2n$

no. of states = 2n.

3) Take one string 'w'.

$w = a^n b^n$

• Calculate the length of the string 'w'.

$|w| = n + n = 2n$.

4) If $|w| \geq n$,

• $2n \geq n$ → true

Break the string w into 3 substrings

x, y, z.

x =

y =

z =

i i

$w = a^n b^n$

$w = a^i b^i$

$xy = a^m$ | $a^i = a^m . a^{i-m}$

$y = a^j$ | $a^7 = a^3 . a^{7-3}$

$z = a^{i-m} . b^i$ | $a^7 = a^3 . a^4$

| $a^7 = a^7$

Substitute $xyz = \underline{xy} . \underline{z}$

$= a^m . a^{i-m} . b^i$

$= a^{\cancel{m}+i-\cancel{m}} . b^i$

$\boxed{xyz = a^i . b^i}$

It is in the form of $a^n b^n$.

So our assumption is correct.

5. Check the conditions.

i) $|xy| \leq n$

$|a^m| \leq n$. Put $xy = a^m$

$m \leq n$. - true.

ii) $y \neq \varepsilon$ $|y| \geq 1$

$a^j \neq \varepsilon \Rightarrow$ true $|a^j| \geq 1$

$j \geq 1 \Rightarrow$ true

Conditions i) and ii) are true.

for $k \geq 0$, the string xy^kz is in L.

6. find xy^kz.

Put $k = 0, 1, 2$.

$xy^kz = xy \cdot y^{k-1} \cdot z$

$xy^kz = a^m \cdot (a^j)^{k-1} \cdot a^{i-m} \cdot b^i$

$$xy^kz = a^m.(a^j)^{k-1}. a^{i-m}. b^i$$

When $k=0$.

$$xy^0z = a^m.\underline{(a^j)^{0-1}}_{1}. a^{i-m}. b^i$$

$$= a^m. \underline{a^0.a^{-j}}_{1}.a^{i-m}. b^i$$

$$= a^{m-j} a.a^{i-m}. b^i$$

$$= a^{\cancel{m}+i-\cancel{m}+j}. b^i$$

$$\boxed{xy^0z = a^{i-j}. b^i} \neq a^i b^i \text{ is not in } L.$$

When $k=1$

$$xy^1z = a^m.(a^j)^{k-1}. a^{i-m}. b^i$$

$$= a^m. a^{jk}. a^{-j}. a^{i-m}. b^i$$

$$= a^m. a^j. a^{-j}. a^{i-m}. b^i$$

$$= a^m. \cancel{a^j}. \cancel{a^{-j}}. a^{i-m}. b^i$$

$$= a^m. a^{i-m} b^i$$

$$= a^{m+i-m} \cdot b^{i}$$

$$= a^{\cancel{m}+i-\cancel{m}} \cdot b^{i}$$

$\boxed{xy~~z~~ = a^{i}.b^{i}}$ $= a^i b^i$ is in L

When K = 2

$$xy^2z = a^m \cdot (a^j)^{K-1} \cdot a^{i-m} \cdot b^i$$

$$= a^m \cdot (a^j)^{2-1} \cdot a^{i-m} \cdot b^i$$

$$= a^m \cdot a^j \cdot a^{i-m} \cdot b^i$$

$$= a^{\cancel{m}+j+i-\cancel{m}} \cdot b^i$$

$\boxed{xy^2z = a^{i+j} \cdot b^i}$ $\neq a^i b^i$

is not in L.

Since for K = 0, 2, we have strings that do not belong to L, so $L = \{a^n b^n / n \geq 1\}$ is not regular

UNIT -V

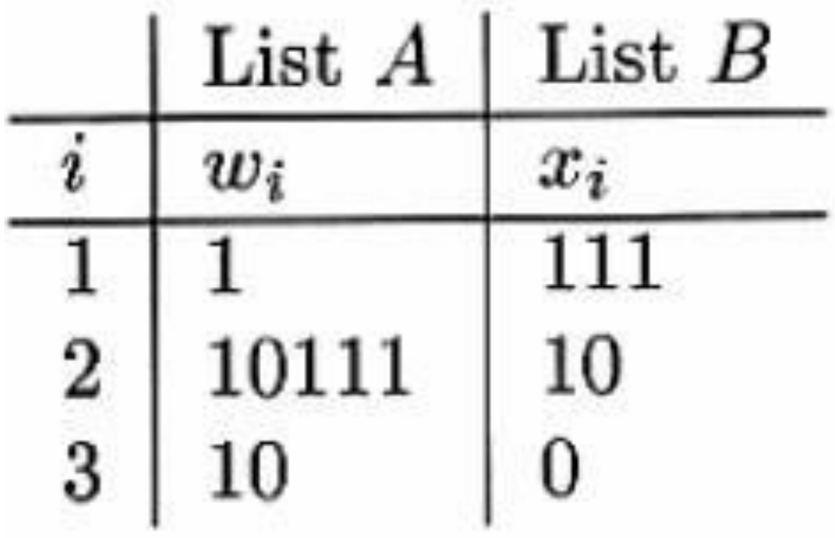
POST CORRESPONDANCE PROBLEM (PCP)

DEFINITION OF POST'S CORRESPONDENCE PROBLEM

- **An instance of Post's Correspondence Problem (PCP) consists of two lists of strings over some alphabet Σ.**
- **The two lists must be of equal length.**
- **We generally refer to the A and B lists, and write A=w1, w2…. wk and B=x1, x2…. xk for some integer k.**
- **For each i, the pair (wi , xi) is said to be a corresponding pair.**
- **We say this instance of PCP has a solution, if there is a sequence of one or more integers i1 ,i2,…. im that, when interpreted as indexes for strings in the A and B lists, yield the same string.**
- **That is, wi1, wi2….. wim, = xi1, xi2,….. xim.**
- **We say the sequence i1 ,i2,…. im is a solution to this instance of PCP.**

EXAMPLE 1:

Let Σ = {0,1}, and let the A and B lists be as defined in the following Figure. Given an instance of PCP, tell whether this instance has a solution.



	List A	List B
i	w_i	x_i
1	1	111
2	10111	10
3	10	0

EXAMPLE 1:

- First we take the strings from the list A and B which starts from same symbol.
- W1 and x1 & w2 and x2 starts with same symbol 1.
- String 3 w3 and x3 starts with different symbol.
- So let start with string 2.
- Take w2 and x2.

i	2			
wi	10111			
xi	10			

EXAMPLE 1:

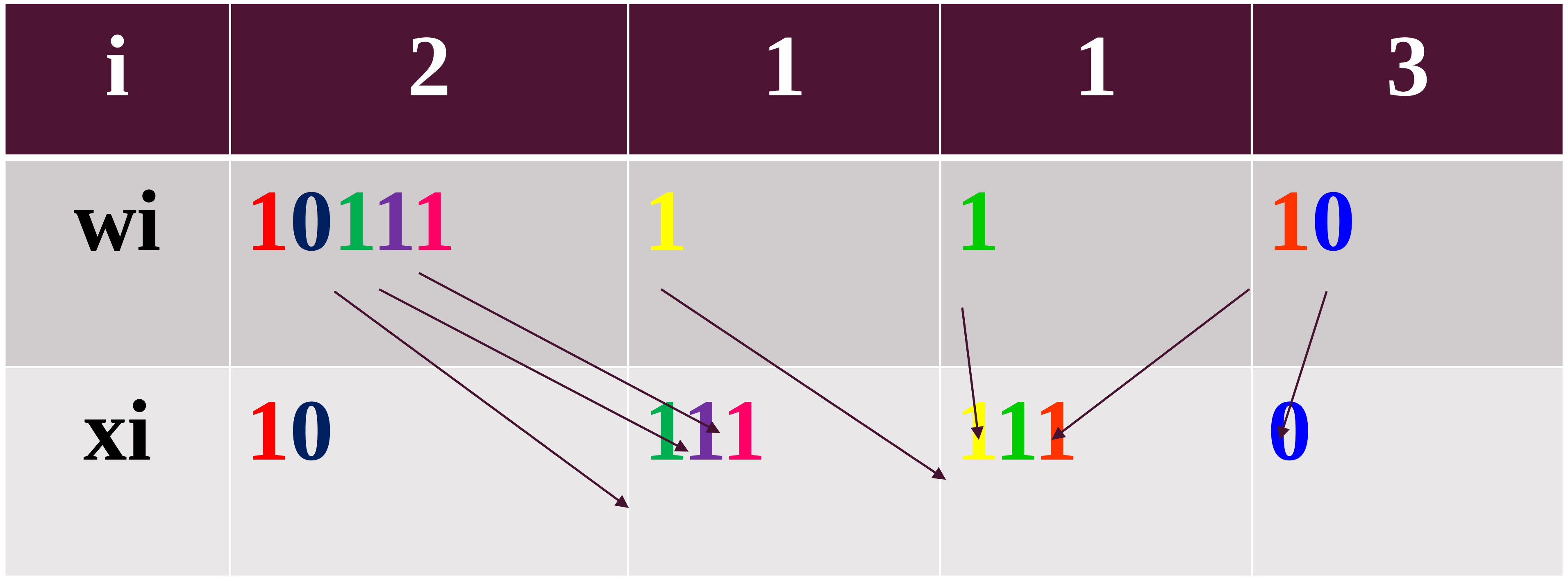
- Next choose the string starts with 1. Again we can choose either 1 or 2.
- Here w2 has more symbols. So we will choose 1.

i	2	1			
wi	10111	1			
xi	10	111			

- Again in wi, l is remaining. So we have to choose string which starts with l from xi.
- Let choose again l.

i	2	1	l		
wi	10111	1	1		
xi	10	111	111		

- Now in xi, 1 is remaining. So we have to choose string which starts with 1 from wi.
- Let choose string 3. i.e: w3 and x3.



i	2	1	1	3
wi	10111	1	1	10
xi	10	111	111	0

That is, w2. w1. w1. w3 = x2. x1. x1. x3 = 101111110

- For instance, let m = 4, $i_1 = 2$, $i_2 = 1$, $i_3 = 1$, and $i_4 = 3$
- (i.e.,) the solution is the list 2, 1, 1, 3

PCP has solution (i1,i2,i3,i4) = (2,1,1,3)

- Note this solution is not unique.
- For instance, 2, 1, 1, 3, 2, 1, 1, 3 is another solution.

EXAMPLE 2:

Let $\Sigma = \{0,1\}$, A= {1,0,010,11} B={ 10,10,01,1}. Given an instance of PCP, tell whether this instance has a solution.

- **First we take the strings from the list A and B which starts from same symbol.**
- **W1 and x1 & w3 and x3 & w4 and x4 starts with same symbol 1 and 0.**
- **String 2 w2 and x2 starts with different symbol.**
- **So let start with string 1.**
- **Take w1 and x1.**

i	1			
wi	1			
xi	10			

- **Next choose the string starts with 0 in w.**
- **Either 2nd string or 3rd string.**
- **So now we can choose string 2.**

i	1	2		
wi	1	0		
xi	10	10		

- Now the string in x remains. 01.
- Now choose string in w starts with 1.
- Again we can choose either 1 or 4.
- So we will choose 1.

i	1	2	1	
wi	1	0	1	
xi	10	10	10	

- Now the string in x remains. 010
- Now choose string in w starts with 0.
- Again we can choose either 2 or 3.
- So we will choose 3.

i	1	2	1	3
wi	1	0	1	010
xi	10	10	10	01

- Now the string in x remains. 01
- Now choose string in w starts with 0.
- Again we can choose either 2 or 3.
- So we will choose 3.

i	1	2	1	3	3	
wi	1	0	1	010	010	
xi	10	10	10	01	01	

- Now the string in x remains. 1
- Now choose string in w starts with 1.
- Again we can choose either 1 or 4.
- So we will choose 3.

i	1	2	1	3	3	4
wi	1	0	1	010	010	11
xi	10		10	10	01	01	1

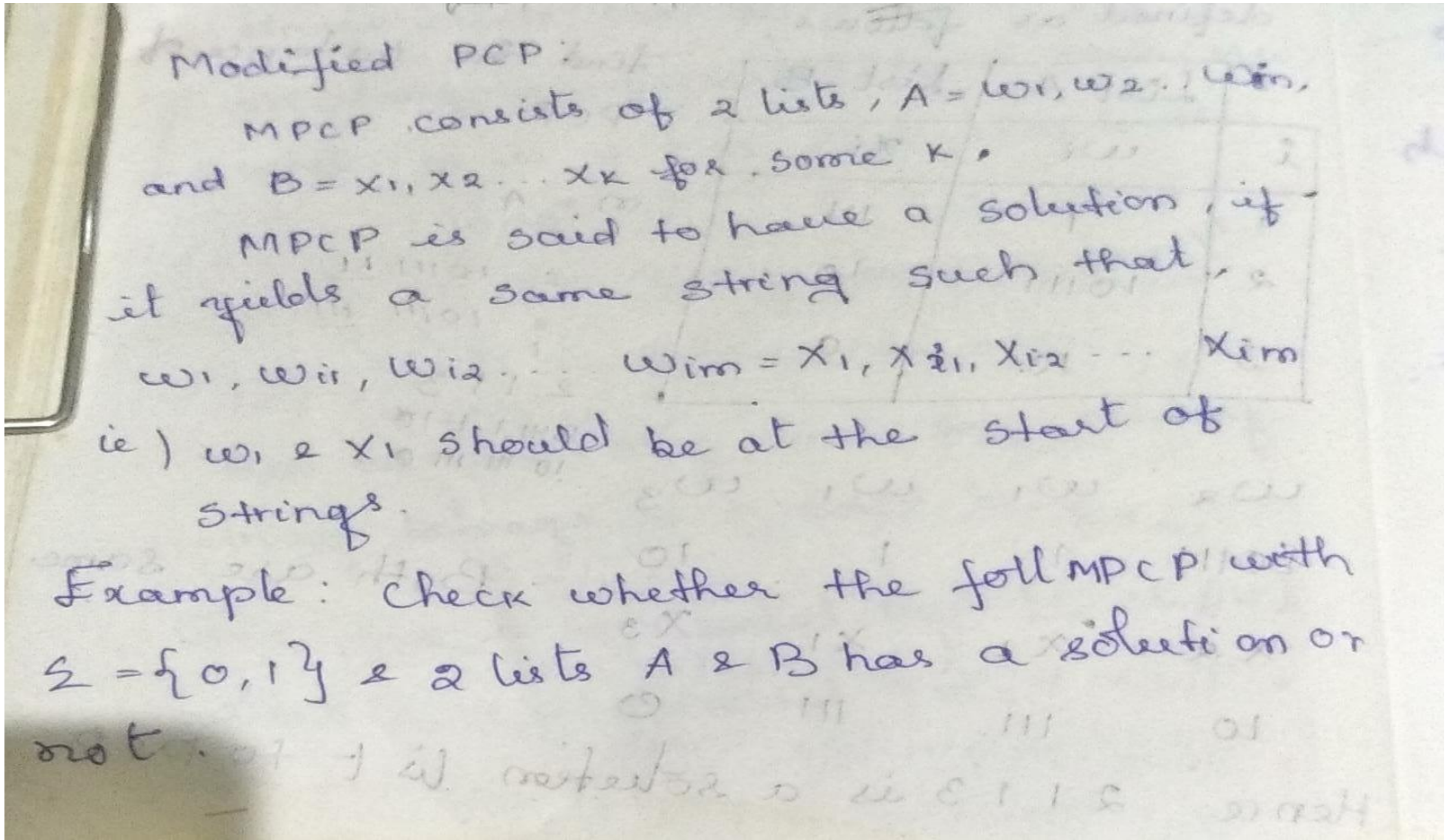
That is, w1. w2. w1. w3. w3. w4 = x1. x2. x1. x3. x3. x4 = 10101001011

- For instance, let m = 6,
- $i_1 = 1, i_2 = 2, i_3 = 1, i_4 = 3, i_5 = 3, i_6 = 4$

- (i.e.,) the solution is the list 1,2,1,3,3,4

PCP has solution (i1,i2,i3,i4,i5,i6) = (1,2,1,3,3,4)

MODIFIED PCP



Modified PCP:

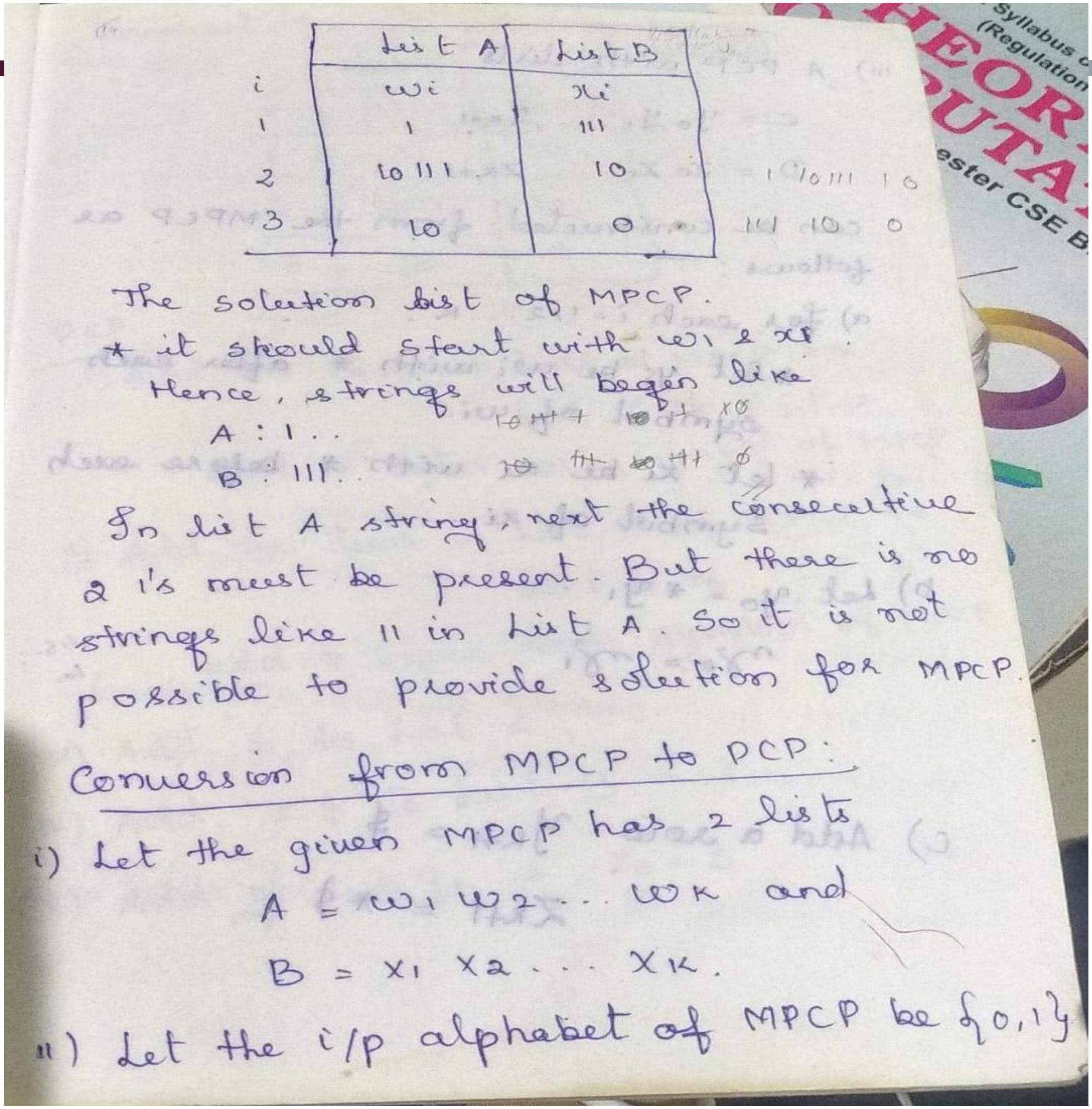
MPCP consists of 2 lists, A = $w_1, w_2 \ldots w_k$, and B = $x_1, x_2 \ldots x_k$ for some k.

MPCP is said to have a solution, if it yields a same string such that,

$w_1, w_{i_1}, w_{i_2} \ldots w_{i_m} = x_1, x_{i_1}, x_{i_2} \ldots x_{i_m}$

ie) w_1 & x_1 should be at the start of strings.

Example: Check whether the foll MPCP with $\Sigma = \{0,1\}$ & 2 lists A & B has a solution or not.



i	List A	List B
	wi	xi
1	1	111
2	10111	10
3	10	0

1 10111 1 0

111 10 0

The solution list of MPCP.

* it should start with w_1 & x_1.

Hence, strings will begin like

A : 1 . .

B : 111 . .

In list A string, next the consecutive 2 1's must be present. But there is no strings like 11 in List A. So it is not possible to provide solution for MPCP.

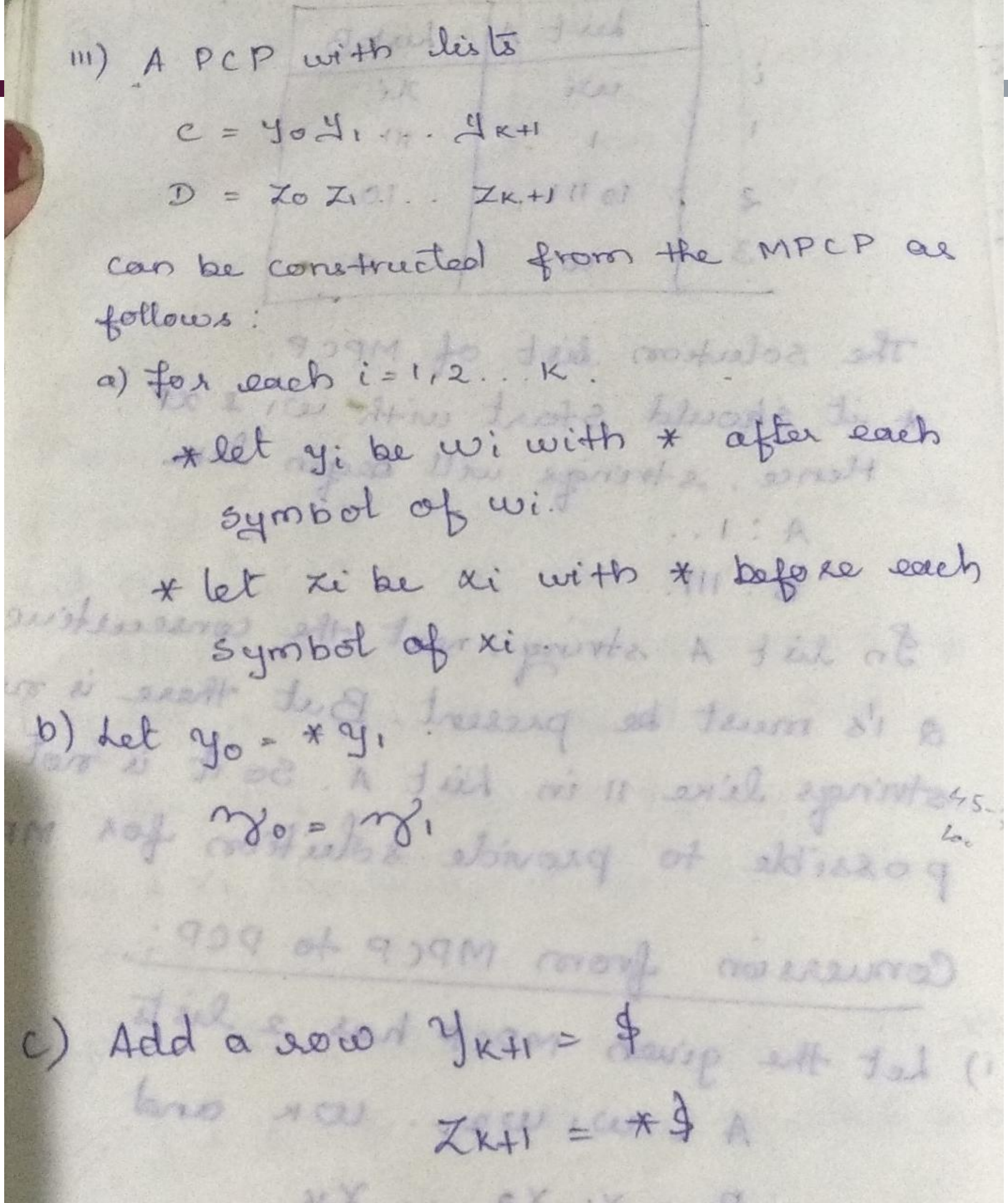
Conversion from MPCP to PCP:

i) Let the given MPCP has 2 lists

$A = w_1 w_2 \dots w_k$ and

$B = x_1 x_2 \dots x_k$.

ii) Let the i/p alphabet of MPCP be $\{0,1\}$



iii) A PCP with lists

$C = y_0 y_1 \ldots y_{k+1}$

$D = z_0 z_1 \ldots z_{k+1}$

can be constructed from the MPCP as follows :

a) for each $i = 1, 2 \ldots k$.

- * let y_i be w_i with * after each symbol of w_i.
- * let z_i be x_i with * before each symbol of x_i

b) Let $y_0 = * y_1$

$z_0 = z_1$

c) Add a row $y_{k+1} = \$$

$z_{k+1} = * \$$

	List A (C)	List B (D)
i	w_i	x_i
1	11	111
2	100	001
3	111	11

	List A	List B
i	w_i	x_i
1	11	111
2	100	001
3	111	11

	List C	List D
i	y_i	z_i
1	1*1*	*1*1*1
2	1*0*0*	*0*0*1
3	1*1*1*	*1*1

Add row as y0 = *y1 , z0 = z1

	List A	List B
i	w_i	x_i
1	1*1*	*1*1*1
2	1*0*0*	*0*0*1
3	1*1*1*	*1*1

	List C	List D
i	y_i	z_i
0	*1*1*	*1*1*1
1	1*1*	*1*1*1
2	1*0*0*	*0*0*1
3	1*1*1*	*1*1

Add row as y_{k+1} = \$ and z_{k+1} = *\$

	List A	List B
i	w_i	x_i
0	*1*1*	*1*1*1
1	1*1*	*1*1*1
2	1*0*0*	*0*0*1
3	1*1*1*	*1*1

	List C	List D
i	y_i	z_i
0	*1*1*	*1*1*1
1	1*1*	*1*1*1
2	1*0*0*	*0*0*1
3	1*1*1*	*1*1
4	$	*$

MODIFIED PCP

- While considering y0,y2,y3 = z0,z2,z3, both are equal.

y_i = z_i

***1*1*1*0*0*1*1*1*$ = *1*1*1*0*0*1*1*1*$**

1. If G is the grammar $S \rightarrow S+S \mid S*S \mid a$, Show that G is ambiguous.

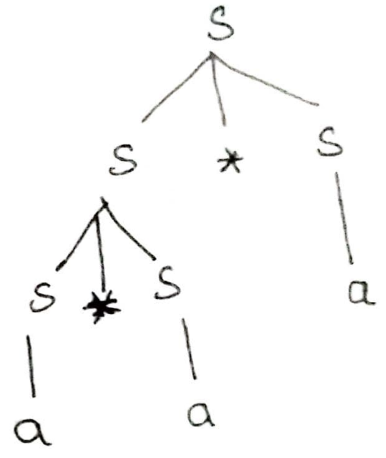
Solution :

Consider the string $w = a * a * a$.

<u>Leftmost Derivation 1</u>:

$S \Rightarrow \underline{S} * S$	$S \rightarrow S * S$
$\Rightarrow \underline{S} * S * S$	$S \rightarrow S * S$
$\Rightarrow a * \underline{S} * S$	$S \rightarrow a$
$\Rightarrow a * a * S$	$S \rightarrow a$
$\Rightarrow a * a * a$	$S \rightarrow a$

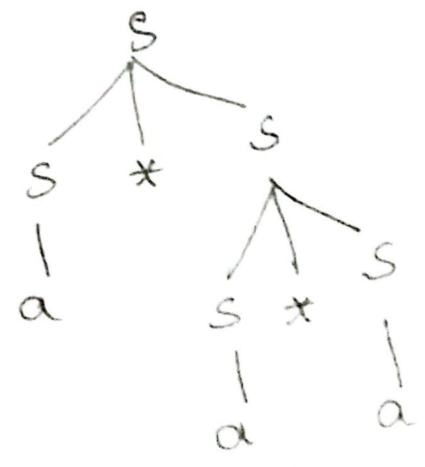
<u>LMD Tree 1</u>



<u>Leftmost Derivation 2</u>:

$S \Rightarrow \underline{S} * S$	$S \rightarrow S*S$
$\Rightarrow a * \underline{S}$	$S \rightarrow a$
$\Rightarrow a * \underline{S} * S$	$S \rightarrow S*S$
$\Rightarrow a * a * \underline{S}$	$S \rightarrow a$
$\Rightarrow a * a * a$	$S \rightarrow a$

<u>LMD Tree 2</u>



$\therefore$ Since there are two LMD trees, it is ambiguous grammar.

2. Show that $E \rightarrow E+E \mid E*E \mid (E) \mid id$ is ambiguous

Solution   string: id + id * id

LMD1

$E \Rightarrow E+E$ $E \rightarrow E+E$

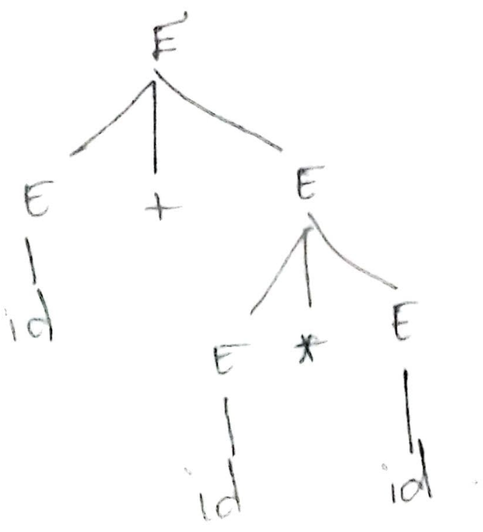
$\Rightarrow id+E$ $E \rightarrow id$

$\Rightarrow id+E*E$ $E \rightarrow E*E$

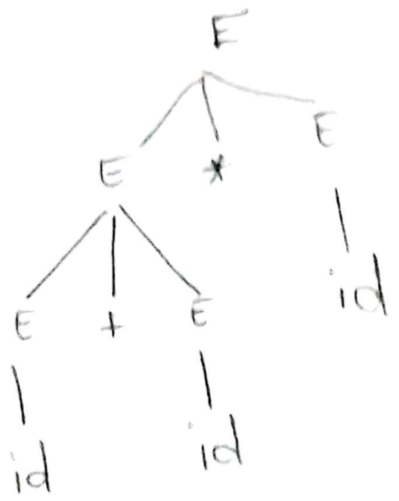
$\Rightarrow id+id*E$ $E \rightarrow id$

$\Rightarrow id+id*id$ $E \rightarrow id$

Tree1



Tree 2



LMD 2

$E \Rightarrow E*E$ $E \rightarrow E*E$

$\Rightarrow E+E*E$ $E \rightarrow E+E$

$\Rightarrow id+E*E$ $E \rightarrow id$

$\Rightarrow id+id*E$ $E \rightarrow id$

$\Rightarrow id+id*id$ $E \rightarrow id$

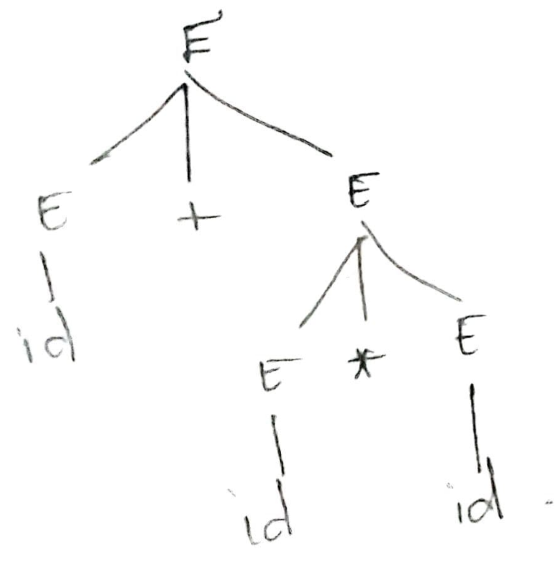
2. Show that $E \to E+E \mid E*E \mid (E) \mid id$ is ambiguous.

Solution: string = $id + id * id$.

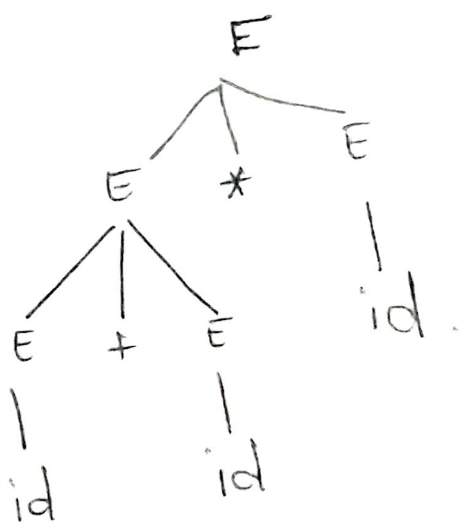
LMD1

$E \Rightarrow E+E$	$E \to E+E$
$\Rightarrow id+E$	$E \to id$
$\Rightarrow id+E*E$	$E \to E*E$
$\Rightarrow id+id*E$	$E \to id$
$\Rightarrow id+id*id$	$E \to id$

Tree1



Tree 2



LMD 2

$E \Rightarrow E*E$	$E \to E*E$
$\Rightarrow E+E*E$	$E \to E+E$
$\Rightarrow id+E*E$	$E \to id$
$\Rightarrow id+id*E$	$E \to id$
$\Rightarrow id+id*id$	$E \to id$

2. Show that $E \rightarrow E+E \mid E*E \mid (E) \mid id$ is ambiguous.

Solution:

string = id + id * id

LMD1

$E \Rightarrow E+E$ $\quad E \rightarrow E+E$

$\Rightarrow id+E$ $\quad E \rightarrow id$

$\Rightarrow id+E*E$ $\quad E \rightarrow E*E$

$\Rightarrow id+id*E$ $\quad E \rightarrow id$

$\Rightarrow id+id*id$ $\quad E \rightarrow id$

Tree1 | Tree 2

2. Show that $E \rightarrow E+E \mid E*E \mid (E) \mid id$ i

ambiguous.

Solution : string = id + id * id.

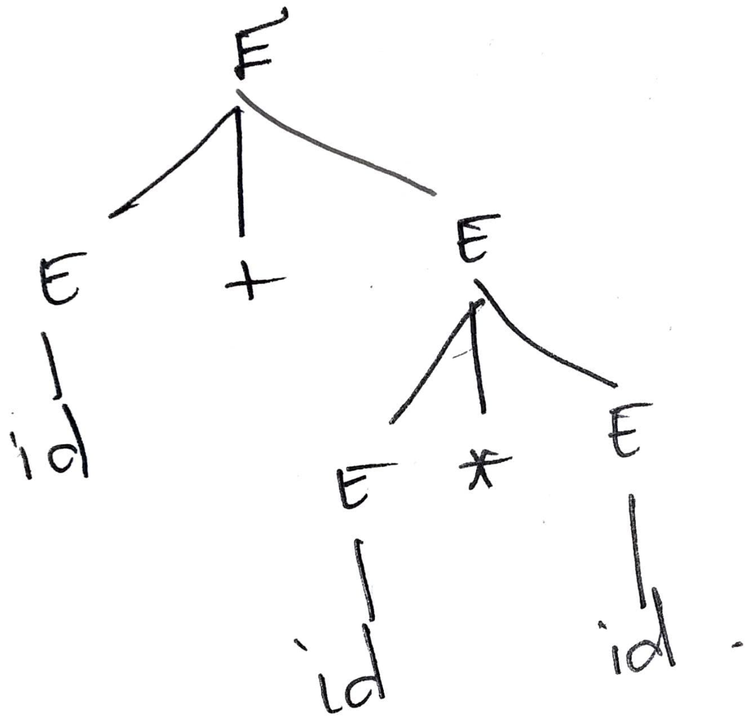
<u>LMD)</u>

$E \Rightarrow E+E$	$E \rightarrow E+E$
$\Rightarrow id+E$	$E \rightarrow id$
$\Rightarrow id+E*E$	$E \rightarrow E*E$
$\Rightarrow id+id*E$	$E \rightarrow id$
$\Rightarrow id+id*id$	$E \rightarrow id.$

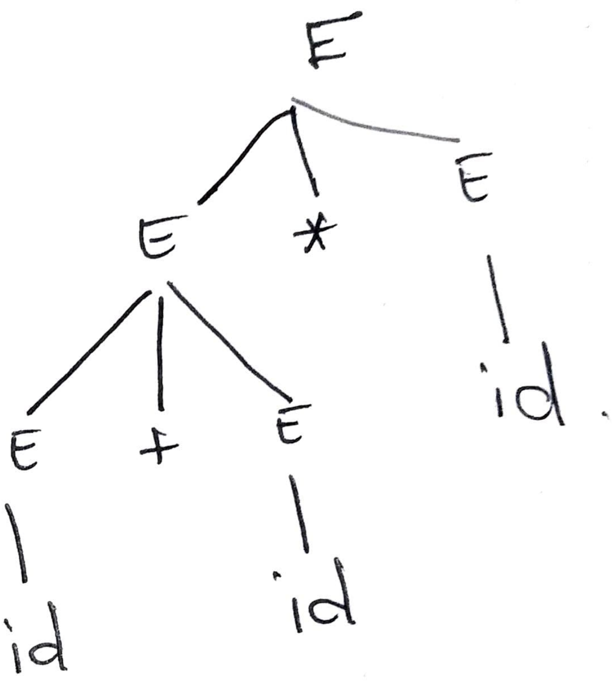
<u>Tree 2</u>.

<u>Tree1.</u>

Tree1.



Tree 2



LMD2

$$\begin{aligned} E &\Rightarrow \underline{E} * E & \quad E &\rightarrow E * E \\ &\Rightarrow E + E * E & E &\rightarrow E + E \\ &\Rightarrow id + E * E & E &\rightarrow id \\ &\Rightarrow id + id * E & E &\rightarrow id \\ &\Rightarrow id + id * id & E &\rightarrow id. \end{aligned}$$

CFG TO PDA

* Let $G = (V, T, P, S)$ be a CFG.
* Construct the PDA P that accepts $L(G)$ by empty stack as follows.

$$\begin{tikzcd}
P = (\{q\}, \arrow[d] & T, \arrow[d] & V \cup T, \arrow[d] & \delta, \arrow[d] & q, \arrow[d] & S) \arrow[d] \\
P = (Q, & \Sigma, & \Gamma, & \delta, & q_0, & Z_0)
\end{tikzcd}$$

where

$Q = \{q\}$, $\Sigma = \{T\}$, $\Gamma = \{V \cup T\}$,

$q_0 = q$, $Z_0 = S$.

δ is defined by

1. for each variable A,

$\delta(q, \varepsilon, A) = (q, \beta)$ where $A \to \beta$ is a production

2. for every terminal symbol $\underline{a}$,

$\delta(q, \underline{a}, a) = (q, \varepsilon)$

(1.) Convert the grammar CFG to a PDA

$E \to E + E$, $E \to id$

Solution:

Let G be a CFG $G(V, T, P, S)$

where $V = \{E\}$, $T = \{+, id\}$, $S = \{E\}$

* The equivalent PDA is given by

$P = (\{q\}, T, V \cup T, \delta, q, S)$

$P = (\{q\}, \{+, id\}, \{+, id, E\}, \delta, q, E)$

Find δ :

For terminals : $+, id$

'+' : $\delta(q, a, a) \longrightarrow (q, \varepsilon)$

$\boxed{\delta(q, +, +) \longrightarrow (q, \varepsilon)}$

'id' : $\boxed{\delta(q, id, id) \longrightarrow (q, \varepsilon)}$

For Non Terminals : E

General form :

If $A \longrightarrow \beta$ then $\delta(q, \varepsilon, A) = (q, \beta)$

i) If $E \longrightarrow E+E$ then $\boxed{\delta(q, \varepsilon, E) = (q, E+E)}$

ii) If $E \longrightarrow id$ then $\boxed{\delta(q, \varepsilon, E) = (q, id)}$

* Check the i/p id+id+id is in N(P) :

$w = id + id + id$

$(q, id+id+id, E) \vdash (q, id+id+d, E+E)$

(state, i/p string w, start symbol)

$\vdash (q, \underline{id}+id+id, \underline{id}+E)$

$\vdash (q, \underline{+}id+id, \underline{+}E)$

$\vdash (q, id+id, E)$

$\vdash (q, id+id, E+E)$

$\vdash (q, \underline{id}+id, \underline{id}+E)$

$\vdash (q, \underline{+}id, \underline{+}E)$

$\vdash (q, id, E)$

$\vdash (q, \underline{id}, \underline{id})$

$\vdash (q, \epsilon, \epsilon)$

∴ Since i/p symbols & stack are empty, the id+id+id is accepted by empty stack.

(2) Convert the grammar $S \to 0S1 | A$, $A \to 1A0 | S | \epsilon$ into a PDA that accepts the same language by empty stack. Check whether 0101 belongs to N(P).

Solution:

Let G be a CFG G(V, T, P, S)

where $V = \{S, A\}$, $T = \{0, 1\}$, $S = \{S\}$.

* The equivalent PDA is given by

$$P = (\underbrace{\{q\}}_{Q}, \underbrace{\{0,1\}}_{T/\Sigma}, \underbrace{\{S, A, 0, 1\}}_{V \cup T / \Gamma}, \underbrace{\delta}_{\delta}, \underbrace{q}_{q_0}, \underbrace{S}_{Z_0})$$

Find δ :

* For terminals : 0 , 1

* for terminal '0' :

$$\delta(q, a, a) \rightarrow (q, \epsilon)$$

$$\boxed{\delta(q, 0, 0) \rightarrow (q, \epsilon)}$$

* for terminal '1' :

$$\boxed{\delta(q, 1, 1) \rightarrow (q, \epsilon)}$$

* for Non Terminals : S, A

* ~~If~~ $A \rightarrow \beta$ then $\delta(q, \epsilon, A) = (q, \beta)$

for Non Terminal S :

* If $S \rightarrow 0S1 | A$ then $\delta(q, \epsilon, S) = (q, 0S1), (q, A)$

for Non Terminal A :

* If $A \rightarrow 1A0 | S | \epsilon$ then

$$\delta(q, \epsilon, A) = (q, 1A0), (q, S), (q, \epsilon)$$

Checking : $w = 0101$

$(q, 0101, S) \vdash (q, 0101, S)$

(state, w, start var)

$\vdash (q, 0101, 0S1)$

$\vdash (q, 101, S1)$

$\vdash (q, 101, 1A01)$

$\vdash (q, 01, A01)$

$\vdash (q, 01, \varepsilon 01)$

$\vdash (q, 01, 01)$

$\vdash (q, 1, 1)$

$\vdash (q, \varepsilon, \varepsilon)$

$\therefore$ $w = 0101 \in N(P)$.

PDA TO CFG

* Let PDA be $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$
* Let CFG be $G = (V, T, P, S)$

where $T = \{\Sigma\}$, $S = S$.

* Variable is made up of $[Q \; \Gamma \; Q]$

$V = [Q \; \Gamma \; Q]$

[State Stackalphabet State]

And all possible combinations of states in Q.

To write production :

① $\delta(q_0, a, Z_0) = (q_0, aZ_0)$

$$[q_0 \; Z_0 \; \underline{q_0}] \rightarrow a[q_0 \; a \; \underline{q_1}] \; [\underline{q_1} \; Z_0 \; \underline{q_0}]$$

② $\delta(q_0, a, Z_0) = (q_1, axx)$

$$[q_0 \; Z_0 \; \_] \rightarrow a[q_1 \; a \; \_] \; [\_ \; x \; \_] \; [\_ \; x \; \_]$$

Eg: To find V = ?

* $Q = \{q_0, q_1\}$, [illegible] $\Gamma = \{a\}$
* [state stack alphabet state]

  $[q_0 \ a \ q_0]$, $[q_0 \ a \ q_1]$

  $[q_1 \ a \ q_0]$, $[q_1 \ a \ q_1]$

* for $Q = \{q_0, q_1\}$, the possible combinations of states : $[q_0, q_0]$ $[q_0, q_1]$ $[q_1, q_0]$ $[q_1, q_1]$

Problem 1 : Convert a given PDA into CFG.

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

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

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

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

v) $\delta(q_1, \varepsilon, Z_0) = (q_1, \varepsilon)$

Solution :

* Let PDA be $(\{q_0, q_1\}, \{a, b\}, \{a, Z_0\}, \delta, q_0, Z_0, \phi)$.

$Q = \{q_0, q_1\}$ $T = \{a, b\}$, $\Gamma = \{a, Z_0\}$, [illegible]

Let CFG be $G(V, T, P, S)$ where

$V = \{[q_0\ a\ q_0]$, $[q_0\ Z_0\ q_0]$
$[q_0\ a\ q_1]$, $[q_0\ Z_0\ q_1]$
$[q_1\ a\ q_0]$, $[q_1\ Z_0\ q_0]$
$[q_1\ a\ q_1]$, $[q_1\ Z_0\ q_1]$, $S\}$

Note:
Here $Q = \{q_0, q_1\}$, $\Gamma = \{a, Z_0\}$

[state stackalp. state]

$[q_0\ \_\ q_0]$, $[q_0\ \_\ q_0]$,
$[q_0\ \_\ q_1]$, $[q_0\ \_\ q_1]$,
$[q_1\ \_\ q_0]$, $[q_1\ \_\ q_0]$,
$[q_1\ \_\ q_1]$, $[q_1\ \_\ q_1]$,

$[q_0\ a\ q_0]$, $[q_0\ Z_0\ q_0]$,
$[q_0\ a\ q_1]$, $[q_0\ Z_0\ q_1]$,
$[q_1\ a\ q_0]$, $[q_1\ Z_0\ q_0]$,
$[q_1\ a\ q_1]$, $[q_1\ Z_0\ q_1]$, S

$\boxed{a}$ $\boxed{Z_0}$

To find Productions :

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

$[q_0 Z_0 \_] \rightarrow a [q_0\ a\ \_][\_\ Z_0\ \_]$

$[q_0 Z_0 \_] \rightarrow a [q_0\ a\ \_][\_\ Z_0\ \_]$

$[q_0 Z_0 \_] \rightarrow a [q_0\ a\ \_][\_\ Z_0\ \_]$

$[q_0 Z_0 \_] \rightarrow a [q_0\ a\ \_][\_\ Z_0\ \_]$

$\Downarrow$

$[q_0 Z_0 q_0] \rightarrow a [q_0\ a\ q_0][\ \ Z_0\ q_0]$

$[q_0 Z_0 q_0] \rightarrow a [q_0\ a\ q_1][\ \ Z_0\ \ ]$

$[q_0 Z_0 q_1] \rightarrow a [q_0\ a\ q_0][\ \ Z_0\ \ ]$

$[q_0 Z_0 q_1] \rightarrow a [q_0\ a\ q_1][\ \ Z_0\ \ ]$

$\Downarrow$

* $[q_0 Z_0 q_0] \rightarrow a [q_0\ a\ q_0][q_0\ Z_0\ q_0]$
* $[q_0 Z_0 q_0] \rightarrow a [q_0\ a\ q_1][q_1\ Z_0\ q_0]$
* $[q_0 Z_0 q_1] \rightarrow a [q_0\ a\ q_0][q_0\ Z_0\ q_1]$
* $[q_0 Z_0 q_1] \rightarrow a [q_0\ a\ q_1][q_1\ Z_0\ q_1]$

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

$$[q_0\ a\ \overset{q_0\ q_1}{\_}] \rightarrow a\ [q_0\ a\ \_]\ [\_\ a\ \_]$$

$\Downarrow$

$[q_0\ a\ q_0] \rightarrow a\ [q_0\ a\ q_0]\ [q_0\ a\ q_0]$

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

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

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

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

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

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

$[q_1\ a\ q_1] \rightarrow b$

5) $\delta(q_1, \varepsilon, Z_0) = (q_1, \varepsilon)$

$[q_1\ Z_0\ q_1] \rightarrow \varepsilon$

for starting variable, production be like.

start var $\longrightarrow$ [st, stack, st]

$S \longrightarrow [q_0, Z_0, q_0]$

$S \longrightarrow [q_0, Z_0, q_1]$

The resultant CFG is G(V, T, P, S) where

$V = \{[q_0\, a\, q_0], [q_0\, Z_0\, q_0], [q_0\, a\, q_1], [q_0 Z_0 q_1]$
$[q_1\, a\, q_0]\ [q_1\, Z_0\, q_0]\ [q_1\, a\, q_1]\ [q_1 Z_0 q_1], S\}$

$T = \{a, b\}$ S - start symbol

P :

$[q_0\, Z_0\, q_0] \rightarrow a\, [q_0\, a\, q_0][q_0\, Z_0\, q_0]$
$[q_0\, Z_0\, q_0] \rightarrow a\, [q_0\, a\, q_1][q_1\, Z_0\, q_0]$
$[q_0\, Z_0\, q_1] \rightarrow a\, [q_0\, a\, q_0][q_0\, Z_0\, q_1]$
$[q_0\, Z_0\, q_1] \rightarrow a\, [q_0\, a\, q_1][q_1\, Z_0\, q_1]$

$[q_0\, a\, q_0] \rightarrow a\, [q_0\, a\, q_0][q_0\, a\, q_0]$
$[q_0\, a\, q_0] \rightarrow a\, [q_0\, a\, q_1][q_1\, a\, q_0]$
$[q_0\, a\, q_1] \rightarrow a\, [q_0\, a\, q_1][q_0\, a\, q_1]$
$[q_0\, a\, q_1] \rightarrow a\, [q_0\, a\, q_1][q_1\, a\, q_1]$

$[q_0\, a\, q_1] \rightarrow b$

$[q_1\, a\, q_1] \rightarrow b$

$[q_1\, Z_0\, q_1] \rightarrow \varepsilon$.

$S \rightarrow [q_0\, Z_0\, q_0]$
$S \rightarrow [q_0\, Z_0\, q_1]$