7-2-2025

MODULE-2

Context free grammar & language

CFG = (V, T, P, S)

V → finite set of non-terminals T → finite set of terminals

P → production rule finite set S → start symbol.

CFG, Production rule is of form $\alpha \longrightarrow \beta$

$|\alpha| = 1$ α is a single non-terminals.

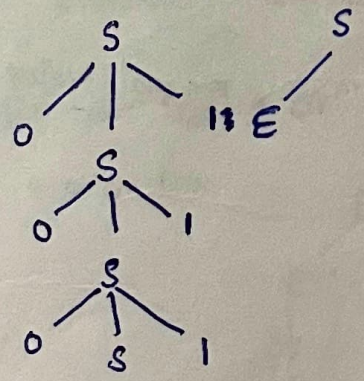
β is a set of terminals & non-terminals $(V \cup T)^*$

eg:
$$\begin{cases} S \longrightarrow 0S1 \\ S \longrightarrow \epsilon \end{cases} \qquad V = \{S\} \quad T = \{0, 1, \epsilon\} \quad S = \{S\}$$

grammar to lang.

P = ① $S \longrightarrow 0S1$

② $S \longrightarrow \epsilon$



L = { ε 01, 0011, 000111 . . . }

L = $\{0^n 1^n \quad n \geq 2\}$

CFL to CFG

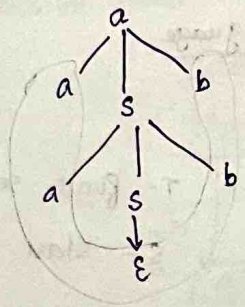
1) $a^n b^n$, $n \geq 0$

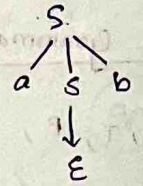
terminating condition is minimal string

$a^0 = b^0 = \epsilon$

So no string.

CFG: $S \longrightarrow aSb \mid \epsilon$





aabb

2) $a^n b^{n+2}$, $n \geq 0$

$S \rightarrow \underbrace{aSb}_{a^n b^n} \mid \underbrace{bb}_{b}$

3) $a^{2n} b^m$, $n \geq 0$

$S \rightarrow aaSb \mid \varepsilon$

4) $a^{2n} b^n$ $n \geq 1$

$S \rightarrow aaSb \mid aab$

5) $a^m b^n$, $m > n$, $n \geq 0$ $\Rightarrow$ $m > 0$ minimal value of m is 1 $\Rightarrow a$

and n is 0 $\Rightarrow b^0 = \varepsilon$

~~$S \rightarrow AS_1$~~

~~$S \rightarrow aSb \mid \varepsilon$~~

$S \rightarrow AS_1$

$S_1 \rightarrow a S_1 b \mid \varepsilon$

$A \rightarrow aA \mid a$

6) $a^m b^n$ $m \geq n$ $n \geq 0$

$S_1 \rightarrow a S_1 b \mid \varepsilon$

$A \rightarrow a A \mid \varepsilon$

$\Big\}$ $S \rightarrow AS_1 \mid a$

Derivation and parse tree / Derivation Tree

Derivation is a sequence of steps followed to generate a string from a given grammar.

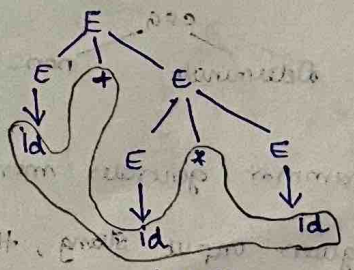
eg: $E \rightarrow E+E \mid E*E \mid id$

check whether the string id+id*id belongs to this grammar or not.

a) Derivation:

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

The pictorial representation of this derivation is called derivation tree or parse tree.



Derivations are of 2 types:

i) Left most derivation ii) Right most derivation.

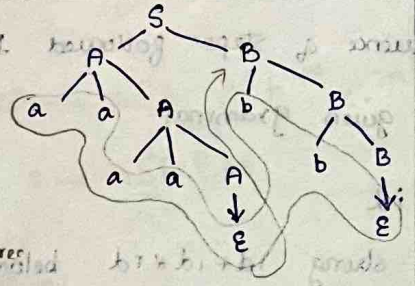
$S \rightarrow AB$

$A \rightarrow aaA \mid \varepsilon$

$B \rightarrow bB \mid \varepsilon$

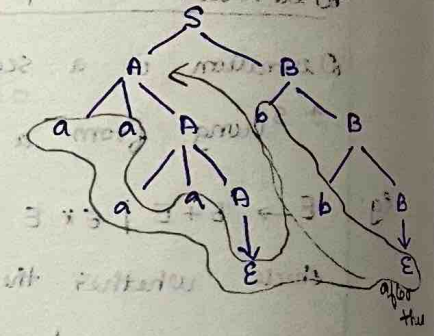
Check whether $w = aaaabb \in L(G)$

Left most derivation



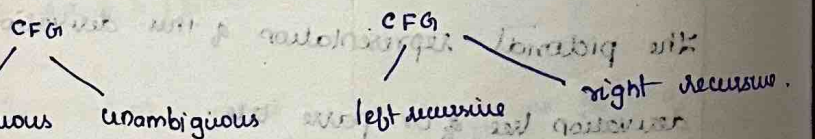
Each step of derivation, left most non terminal is replaced.

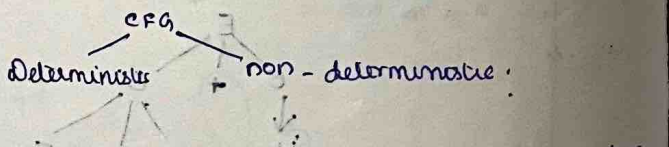
Right most derivation



Right most non-terminal is replaced.

CFGs are classified into 3 categories





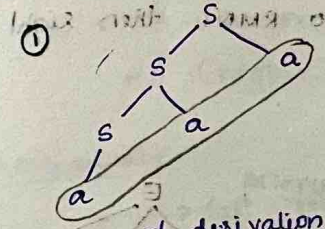
Ambiguous

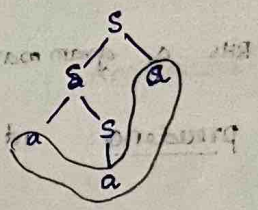
Suppose a grammar generates more than one derivation tree for the given input string, then the grammar is called ambiguous grammar.

Q. 1) Check whether given grammar is ambiguous grammar.

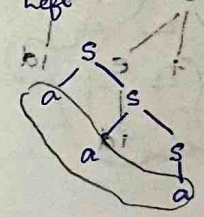
G: $S \rightarrow Sa \mid aS \mid a$

i/p → aaa

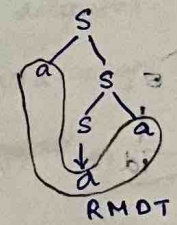
① 

② 

Left most derivation tree



RMDT



RMDT

LMDT

∴ given grammar is ambiguous since more than 1 derivation tree is possible.

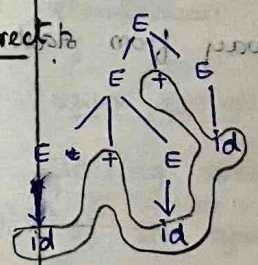
2) $E \rightarrow E + E \mid E * E \mid id$

i/p → id + id + id

conversion to unambiguous cannot be done directly. So should take associativity and precedence.

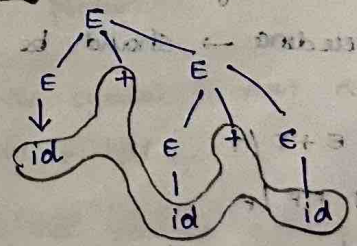
id + (id + id) Associativity

Correct



here id + id is associated first.

wrong



here only id is evaluated then id + id

+ operator is left associative.

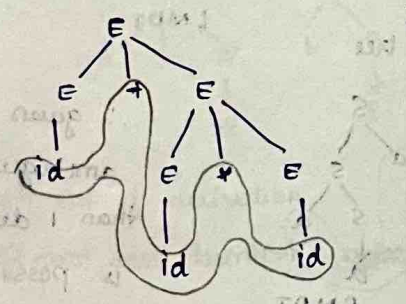
define recursive

$E \to E + id \mid id$ — LHS of grammar is equal to left most non-term. then its left recursive

$E \to id + E$ — RHS of grammar is equal to RMNT then right recursive (right association)

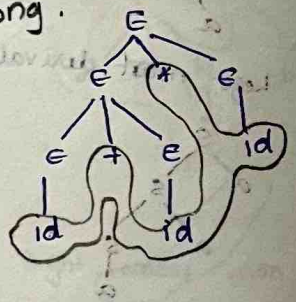
precedence: id + id * id

correct



id + (id * id)

wrong.



(id + id) * id
didn't check precedence.

→ define levels.

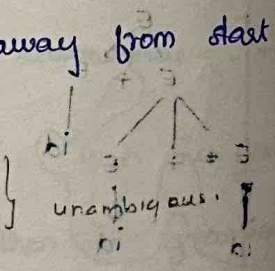
Operations with less precendence should be closer to start symbol. for

High precedence → should be far away from start symbol

2) $E \to E + T \mid T$ → left recursive

$T \to T * F \mid F$ → left recursive

$F \to id$



3) $E \to E * F \mid F + e \mid F$

$F \to F - F \mid id$.

precedence: + * − (operators)

* + (lower precedence)

\- (higher)

∴ Ambiguous.

Associativity

* → left recursive

\+ → right recursive.

\- → left and right associative

∴ Not ambiguous.

Simplification of CFGs

3 steps

1) Elimination of useless symbols.
2) Elimination of ε production
3) Elimination of unit production.

→ useless symbols - i) If the symbol is not derived from the starting symbol.

ii) If the symbol does not produce a set of terminal symbols.

eg: Simplify given grammar:

$S \rightarrow A \mid 0C1$

$A \rightarrow B \mid 01 \mid 10$

$C \rightarrow \epsilon \mid cD$ produces terminal symbols so not useless

$B \rightarrow d$

step 1: derived from starting symbols

→ but if there is $B \rightarrow d$, its useless since B is not derived from starting symbol

$S \rightarrow A \rightarrow 01$ ✓

$S \rightarrow 0C1 \rightarrow 0\epsilon1 \rightarrow 01$

✗ $A \rightarrow B$ (but B productions are not available so useless production)

✓ $A \rightarrow 01$

✓ $A \rightarrow 10$ (terminal symbols are generated so useful)

$C \rightarrow \epsilon$ (ϵ is also terminal) ✓

✗ $C \rightarrow cD$ (This is not going to generate any set of terminal symbols. C production not there). → no substitution for $C \rightarrow cD$, $C \rightarrow \epsilon$

After eliminating useless symbols / productions:

$S \rightarrow A \mid 0C1$

$A \rightarrow 01 \mid 10$

$C \rightarrow \epsilon$ ($C \rightarrow \epsilon$ is there)

step 2: $C \rightarrow \epsilon$, ϵ production, hence eliminated.

Whenever C comes its replaced with null ϵ.

If C is given ϵ $S \rightarrow A \mid 0C1 \mid 0\epsilon1$

$S \rightarrow A \mid 0C1 \longrightarrow A \mid 01 \Rightarrow S \rightarrow A \mid 0C1 \mid 01$

$A \rightarrow 01 \mid 10$

step 3: unit production: LHS contain only one non-terminal

Likewise RHS also contain only one non-terminal

$A \rightarrow B$, $B \rightarrow \alpha_1, \alpha_2, \ldots \alpha_n$

eg: $A \rightarrow B$, $B \rightarrow \alpha_1, \alpha_2, \ldots \alpha_n$

The unit production $A \rightarrow B$ can be eliminated by non-terminals in both sides

$A \rightarrow \alpha_1, \alpha_2, \ldots \alpha_n$.

$S \rightarrow A \mid 0C1 \mid 01$

so eliminate it.

Here in eg: $S \rightarrow A$ is the unit production.

We know $A \rightarrow 01 \mid 10$.

So we can eliminate $S \rightarrow A$ as and grammar becomes.

$S \rightarrow 01 \mid 10$

So simplified grammar after elimination is:

$S \rightarrow 01 \mid 10 \mid 0C1 \mid 01$ already there.

$\Rightarrow S \rightarrow 01 \mid 10 \mid 0C1$

Q.1) Eliminate useless symbols from given grammar.

$S \to AB \mid a$

$A \to a$.

Step: 1 S is producing AB, a. A can produce a but

B cant produce terminal symbol

∴ B is useless symbol.

∴ $S \to AB$ ✗ $S \to aB$

$S \to a$ ✓

=.

Step-2: 2) Eliminate ε productions from given grammar.

$S \to AB$

$A \to aAA \mid \varepsilon$

$B \to bBB \mid \varepsilon$

Here $A \to \varepsilon$ and $B \to \varepsilon$. Eliminate by substituting ε in place of A and B.

∴ final:

$S \to AB \mid B \mid A \mid \varepsilon$

$A \to aAA \mid aA \mid a$

$B \to bBB \mid bB \mid b$

=.

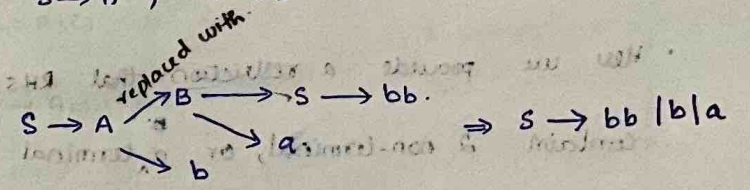
3) Eliminate unit production from given grammar.

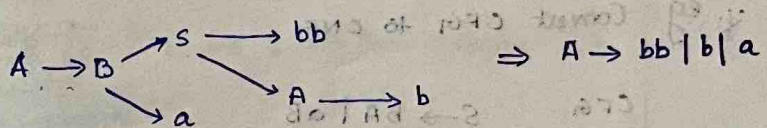
$S \to A \mid bb$

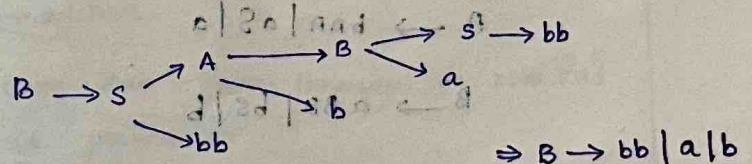
$A \to B \mid b$

$B \to S \mid a$.

Unit prd: $S \to A$, $A \to B$, $B \to S$







Normal forms of CFG

Used to give restriction to grammar.

P: $\alpha \to \beta$ $\alpha \to$ single non-terminal. β can have any no. of NT.

2 types of NF

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

Chomsky Normal Form (CNF)

$A \to BC$

(or) $A \to a$

(or) $A \to \varepsilon$ (if then A is the start symbol generating ε)

- Here we provide a restriction that RHS of production contain 2 non-terminal, or a terminal or ε

eg: Convert CFG to CNF

CFG: $S \to bA \mid aB$

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

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

a) 1) Step 1: Simplify the CFG:

2) Replace terminals in mixed product

eg: $X \to \alpha Y$ → mixture of terminal & non-terminal.

can be decomposed as: Here Replace terminal α to non terminal C_α.

$X \to C_\alpha Y$

$C_\alpha \to \alpha$ to balance

3) Reduce productions with more than 2 non-terminals

Consider a grammar $A \to A_1, A_2 \ldots\ldots A_m$ where $m \geq 3$, we can introduce new production.

$A_1 \to A_1 C_1$

$C_1 \to A_2 C_2$

$C_2 \to A_3 C_3$

$\vdots$

$C_{m-2} \to A_{m-1} C_{m-1}$

eg: $X \to XYZ$ → 3 non-term

$X \to XZ$ ↙ replace by 2

$Z \to YZ$ ↙ balance.

ans: 1) Simplify CFG:

→ $S \to A, B, \varepsilon$ A and B → non-terminal a, b

∴ No useless symbols.

No ε production

There are more than 1 non-terminal on ~~both~~ RHS.

∴ no unit production

2) Replace terminals in the mixed production

- $S \to bA$ (mixed) ⇒ $S \to bA$

$S \to C_b A$ ✓ chomsky.

$C_b \to b$

- $S \to aB$ ⇒ $S \to C_a B$ ✓

$C_a \to a$

- $A \to bAA$ ⇒ $A \to C_b AA$ ✓

$C_b \to b$

$A \to aS \Rightarrow A \to CaS$ ✓

$Ca \to a$

$B \to aBB \Rightarrow B \to CaBB$

$Ca \to a$

$B \to bS \to B \to CbS$ ✓

$Cb \to b$

Step 3: $CbAA$ and $CaBB$ not chomsky.

$A \to CbAA$

$A \to CbD_1$ ✓

$D_1 \to AA$

$B \to CaBB$

$B \to CaD_2$ ✓

$D_2 \to BB$

Final grammar: $S \to CbA$, $A \to CaS$, $Ca \to a$, $Cb \to b$, $B \to CaD_2$, $S \to CaB$, $D_2 \to BB$, $Ca \to a$, $B \to CbS$, $Cb \to b$, $A \to CbD_1$, $D_1 \to AA$

Greibach Normal Form (GNF)

A CFG is said to GNF if the production is in form:

$A \to a\alpha$ [OR] $A \to \varepsilon$ (start symbol generating ε)

A: non-terminal; a: terminal; α: zero or more non-terminals

→ Here no restriction in length of RHS. Restriction is given to the position in which terminal and non-terminal appears in the production

$A \to a$

or $A \to a$ — any no. of non-ter.

Conversion of CFG to GNF

Step 1: Simplify the grammar

2. Check whether the grammar is in Chomsky NF, if not convert it into CNF.

3. Replace non-terminals of the grammar with some Ai in ascending order of i

4. Alter the rules so that the non-terminals are in ascending order such that if the production in the form $Ai \to AjB$ then check 3 condition:

i) $i < j$ then leave the production as it is now, then do substitution after obtaining Aj value

ii) $i > j$, perform one substitution (let $A \to \beta\alpha$ and $\beta \to \beta_1 / B_2 \ldots B_n$ then $A \to \beta_1\alpha / \beta_2\alpha \ldots \beta_n\alpha$)

iii) $i = j$, grammar contains left recursion so eliminate

left recursion

$A \longrightarrow A\alpha \mid \beta$

↓ be replaced as

$A \longrightarrow \beta A'$

$A' \longrightarrow \alpha A' \mid \varepsilon$

Continue this process until all productions are in CNF form

Q. Convert the CFG to GNF

$S \rightarrow CA \mid BB$

$B \rightarrow b \mid SB$

$C \rightarrow b$

$A \rightarrow a$

Step 1. a) No useless value

No ε production.

No unit production.

The CFG is simplified

Step 2: The CFG is in CNF form

Step 3: S replace with A_1, C with A_2, A with A_3, B with A_4

$A_1 \longrightarrow A_2 A_3 \mid A_4 A_4$ —①

$A_4 \longrightarrow b \mid A_1 A_4$ —②

$A_2 \longrightarrow b$ —③

$A_3 \longrightarrow a$ —④

Step 4: i) $A_i \longrightarrow A_j \beta$ ⇒ $i < 2$ ie $(i<j)$ → leave production as it is and wait for substitution of A_2 after getting value of A_j

ii) $A_1 \longrightarrow A_2 A_4 A_4$

$1 < 4$ $(i<j)$

$\therefore A_1 \longrightarrow A_2 A_3 \mid A_4 A_4$ —①

b) $A_4 \longrightarrow b$ is already in GNF

$A_4 \longrightarrow A_1 A_4$ $(4>1)$ ie $(i>j)$, perform substitution

Here problem is created by A_1. So replace A_1

$A_4 \longrightarrow b \mid A_2 A_3 A_4 \mid A_4 A_4 A_4$ (GNF)

from ③, $A_4 \longrightarrow A_2 A_3 A_4$; $(i>j)$ A_2 is substituted by b.

$A_4 \longrightarrow b A_3 A_4$ — GNF

$A_4 \longrightarrow A_4 A_4 A_4$ $(i=j)$

~~$A_4 \longrightarrow b$ so substitute~~

remove left recursion

$A \rightarrow A\alpha \mid \beta$

$A \rightarrow \beta A'$

$A' \longrightarrow \alpha A' \mid \varepsilon$

$A_4 \longrightarrow b A_4' \mid b A_3 A_4 A_4'$

$A_4' \longrightarrow A_4 A_4 A_4' \mid \varepsilon$

Eliminate the ε production: (Replace A_4' by ε)

⇒ $A_4 \longrightarrow bA_4' \mid b \mid bA_3A_4A_4' \mid bA_3A_4$ → GNF —(F) Final.

$A_4' \longrightarrow A_4 A_4 A_4' \mid A_4 A_4$

$A_4 \rightarrow bA_4' \mid b \mid bA_3A_4A_4' \mid bA_3A_4$ — (F1) ← A_j

Now we got value of A_4 so substitute in (1) (one A_j value except it all are in GNF we got)

$A_1 \rightarrow A_2A_3 \mid bA_4'A_4 \mid bA_4 \mid bA_3A_4A_4'A_4 \mid bA_3A_4A_4$ $\quad A_1 \rightarrow A_jA_q$

Substitute A_2 with b

$A_1 \rightarrow bA_3 \mid bA_4'A_4 \mid bA_4 \mid bA_3A_4A_4'A_4 \mid bA_3A_4A_4$ } GNF — F_2

Substitute production of A_4 in A_4' ($A_4' \rightarrow A_4A_4A_4' \mid A_4A_4$)

$A_4' \rightarrow bA_4'A_4A_4' \mid bA_4A_4' \mid bA_3A_4A_4' \mid A_4A_4' \mid bA_3A_4 \mid A_4A_4'$

$bA_4'A_4 \mid bA_4 \mid bA_3A_4A_4'A_4 \mid bA_3A_4A_4$ — f_3

After converting GNF, the given grammar becomes:

$A_4 \rightarrow bA_4' \mid b \mid bA_3A_4A_4' \mid bA_3A_4$

$A_4' \rightarrow bA_4'A_4A_4' \mid bA_4A_4' \mid bA_3A_4A_4' \mid A_4A_4' \mid bA_3A_4 \mid A_4A_4' \mid$

$bA_4'A_4 \mid bA_4 \mid bA_3A_4A_4'A_4 \mid bA_3A_4A_4$

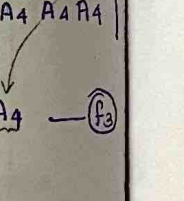
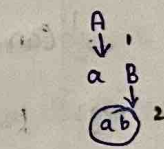
$A_1 \rightarrow bA_3 \mid bA_4'A_4 \mid bA_4 \mid bA_3A_4A_4'A_4 \mid bA_3A_4A_4$

$A_2 \rightarrow b$

$A_3 \rightarrow a$

(CNF)

Difference b/w Chomsky and GNF

	CNF	GNF
1)*	Syntax: $A \rightarrow BC$ (OR) $A \rightarrow a$ (OR) $A \rightarrow \alpha$ (A start symbol)	* syntax: $A \rightarrow aX$ $A \rightarrow \varepsilon$ (A start symbol)
2)	eg example	• example.
3 *	No. of steps required to generate a string of length n is $2\lvert n\rvert - 1$	* No. of steps required to generate a string of length n is n
	eg: $A \rightarrow AB$ $A \rightarrow a$ $B \rightarrow b$ 	eg: $A \rightarrow aB$ $B \rightarrow b$ 
	no. of steps $= 2\lvert n\rvert - 1 = 2 \times 2 - 1 = 3$	no. of steps $= n = 2$
4*	using membership algorithm.	* It is used to convert CFG to PDA
5 *	Derivation of parse tree obtained from CNF is always a binary tree	* Not always
6 *	Length of each production is restricted	* Not restricted bcz A can have

Push down Automata

Represented using 7 tuples.

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

τ → finite set of stack symbols

z_0 → stack start symbol. (by default, top of stack is z_0)

Limitation of FA

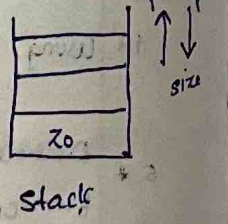
→ Can have only finite amount of memory only.

eg:- $L = \{0^n 1^n \mid n \geq 0\}$ ← Here there no limiting or comparison conditions.

→ Cannot produce complex languages.

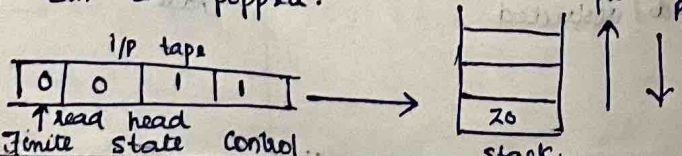
→ PDA = FA + stack.

Stack provides addition memory.



Stack

So in this case when 0's come, it can be pushed and 1's come, it can be popped.



stack

Design PDA

For DPDA, Deterministic: $\delta : Q \times (\Sigma \cup \epsilon) \times \tau \longrightarrow Q \times \tau^*$

For NPDA: $\delta : Q \times (\Sigma \cup \epsilon) \times \tau \longrightarrow 2^{(Q \times \tau^*)}$

→ δ takes argument as tuple $\delta(q, a, X)$ where,

q → state in Q a → either i/p symbol in Σ or $a = \epsilon$

X → stack symbol, that is a member of τ

→ O/p of δ is a finite set of pairs (P, τ) where

P → new state τ → string of stack symbols that replaces X at the top of stack.

$\delta(q, a, X) \longrightarrow (P, \tau)$

→ If $\tau = \epsilon$, then the stack is popped.

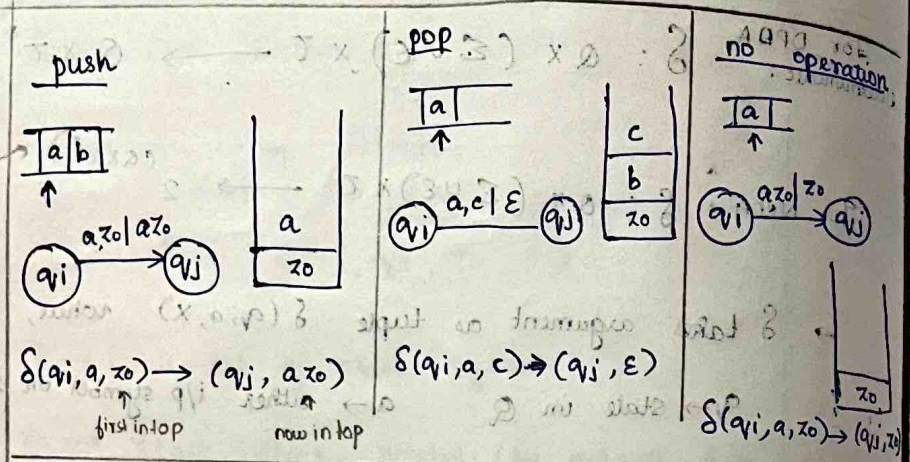
If $\tau = X$, then the stack is unchanged (no operation occured)

If $\tau = YZ$, then X is replaced with Z and Y. X is on top itself is pushed onto the stack.

3 operations of stack

i) push
ii) pop
iii) no operation

- Its not necessary that u should go to one state to another by doing these operation. If possible to stay in the same state.



Construct a PDA

Acceptance of PDA

A language is accepted by PDA is two ways:

i) Acceptance by final state

ii) Acceptance by empty stack

Q. Construct a PDA for the language

$L = \{ a^n b^n \mid n > 0 \}$

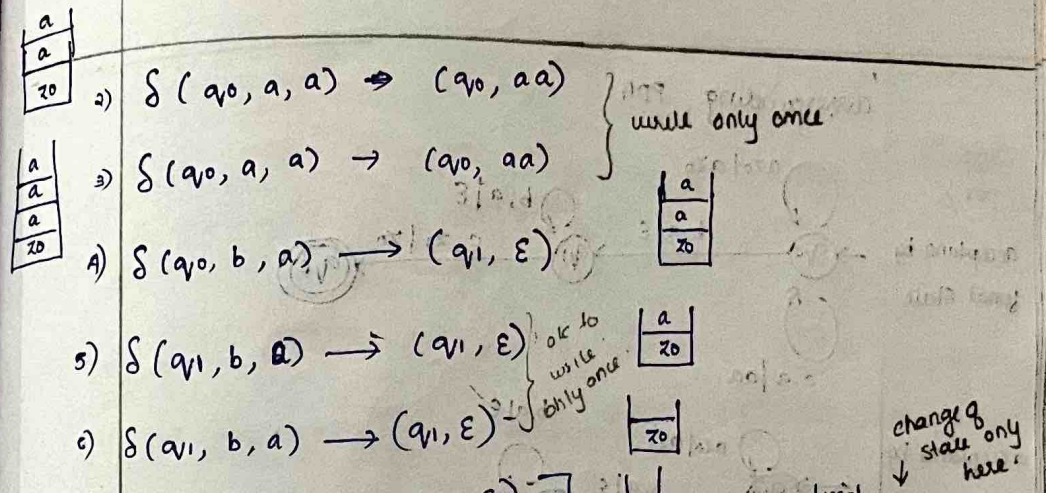
PDA = $\{ Q, \Sigma, \tau, \delta, q_0, z_0, F \}$

- when a comes push, when first b comes change state as well as pop and for rest just pop only.

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

Consider i/p symbol aaabbbε

1) $\delta(q_0, a, z_0) \rightarrow (q_0, a z_0)$ (a / z0) (ii) every symbols dont write → just only write 2 symbols in top



- Can represent configuration of PDA using instantaneous description also called ID

Instantaneous description (ID) can be defined using a tuple

(Q, w, γ)

Q → current state w → Remaining i/p string to be processed

γ → current stack content. Here will full content of stack.

ID of language:

eg: $(q_0, aaabbb, z_0) \vdash (q_0, aabbb, a z_0) \vdash (q_0, abbb, aaz_0)$ (fallen T)

$\vdash (q_1, bb, a z_0) \vdash (q_1, b, z_0) \vdash (q_1, \varepsilon, \varepsilon)$ (pop)

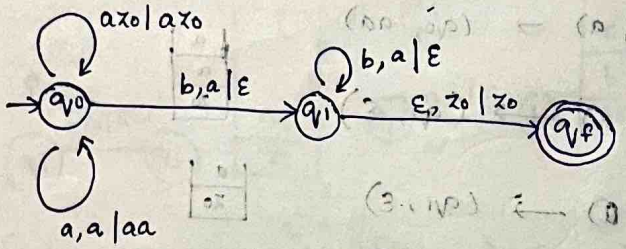
$(q_1, \varepsilon, z_0) \vdash (q_F, \varepsilon, z_0)$ acceptance by final state

acceptance by empty stack $\vdash (q_1, \varepsilon, \varepsilon)$

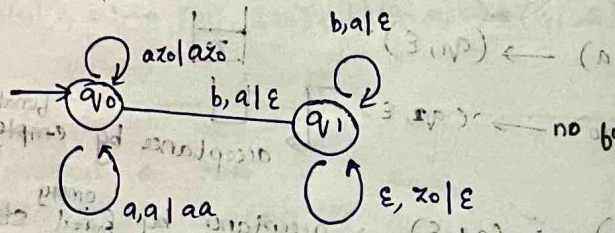
draw as per δ

acceptance by final state:

corresponding PDA:



acceptance by empty stack:



acceptance by final state

$(q_0, w, z_0) \vdash^* (p, \varepsilon, \gamma)$ $\gamma \in \Gamma^*$

p: final state; $p \in F$

I/P string completely processed

acceptance by empty state

$(q_0, w, z_0) \vdash^* (p, \varepsilon, \gamma)$ $\gamma \in \varepsilon$

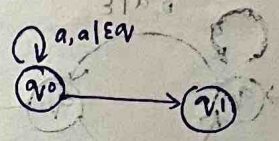
any state in PDA

I/P string is completely processed

for 2b pop1 a)

Deterministic PDA (DPDA)

Can move maximum to one state only (corresponding to I/P symbol and stack topmost symbol)



CFG → generates → CFL

CFL → accepted by → PDA

DCFL ⊂ CFL (DCFL for some)

CFL (for all)

PDA → DPDA, NDPDA

P(DPDA) ⊂ P(NDPDA)

examples of DCFL (Det context free lang)

- This language is accepted by PDA.
- Power of DFA = power NFA, but
- Power of DPDA < P(NDPDA)

example of DCFL / languages accepted by DPDA

1. $L = \{ a^n b^n \mid n \geq 1 \}$
2. $L = \{ a^n b^{2n} \mid n \geq 1 \}$
3. $L = \{ w c w^R \mid w \in (a+b)^* \}$
4. $L = \{ w \mid n_a(w) = n_b(w) \}$

Q. Design a DPDA for $L = \{ a^n b^{2n} \mid n \geq 1 \}$

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

read a → push a

first b → don't do pop operation

second b → perform pop operation.

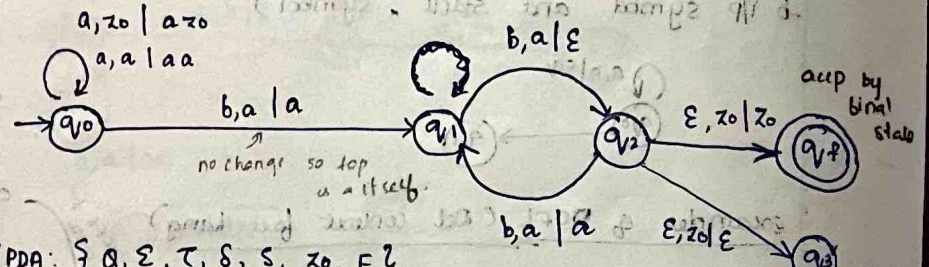
if there is comparison, can be accepted by FA, then we PDA

only elements that are pushed to stack are stack symbols. for odd no of bs q_1
even no of b - q_2

Third b → dont perform any pop operation

fourth b → perform pop operation.

i/p aabbbb



wk, PDA: { Q, Σ, τ, δ, s, z0, F }

corresponding PDA is dont need to draw both. only 1 method is needed

PDA: { {q0, q1, q2 qf} {a, b}, {a, z0}, δ, q0, z0, qf }

c is a seperator for w.

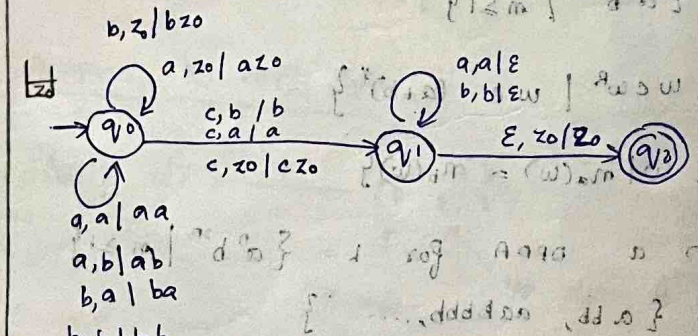
R - reverse

c is middle of str

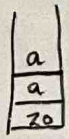
Q) For the language $wcw^R \mid w \in (a+b)^*$. Design a PDA

a) L = { c, aca, bcb, aacaa, abacaba, ababcbaba }

unlike c - push at c state change after c - pop.



till c push to the stack (a or b). when c comes it's popped.



PDA = { {q0, q1, q2}, {a, b}, {a, b, z0}, δ, q0, z0, q2 }

2) Construct PDA for lang $L = \{ w \mid n_a(w) = n_b(w) \}$

a) L = { ab, ba, aabb, aaabbb, bbaba, ababab ..... }

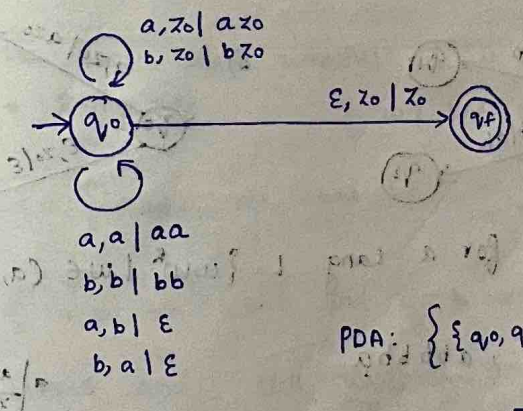
i) Stack empty, i/p → a or b } push

ii) if TOS = a & i/p → a } push 'a' into stack.

∴ if TOS = a & i/p is b, pop

iii) if TOS = b & i/p = b, push

if TOS = b & i/p = a, pop



PDA: { {q0, qf}, {a, b}, {a, b, z0}, δ, q0, z0, qf }

3) Design a PDA for a $L = \{ a^{2n} b^n \mid n \geq 1 \}$

L = { aab, aaaabb, aaaaaabbb ... }

even 1st a comes → push

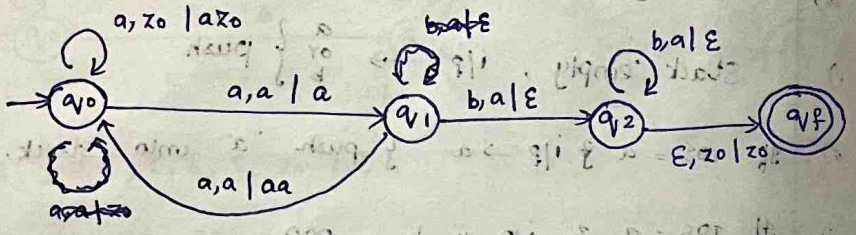
and a comes → ignore (no operation) but change state

if particular string given in Que and asked if acceptable, write instantaneous desc also.

3rd a → push 4th a → ignore

(odd as → push, even as → ignore, & change state)

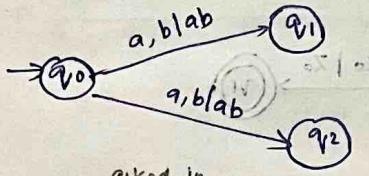
whenever b comes pop. (1st b → change state)



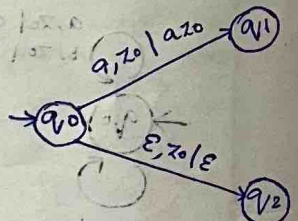
PDA = { { q_0, q_1, q_2, q_f }, { a, b }, { a, z_0 }, δ, q_0, z_0, q_f }

NDPDA (Non-Deterministic PDA)

1)



2)



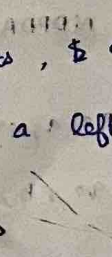
asked in que

eg. Construct NDPDA for a Lang $L = \{ ww^R \mid w \in (a,b)^* \}$

don't know where's middle of the str.

$L = \{$ abbba, abbbba $\}$

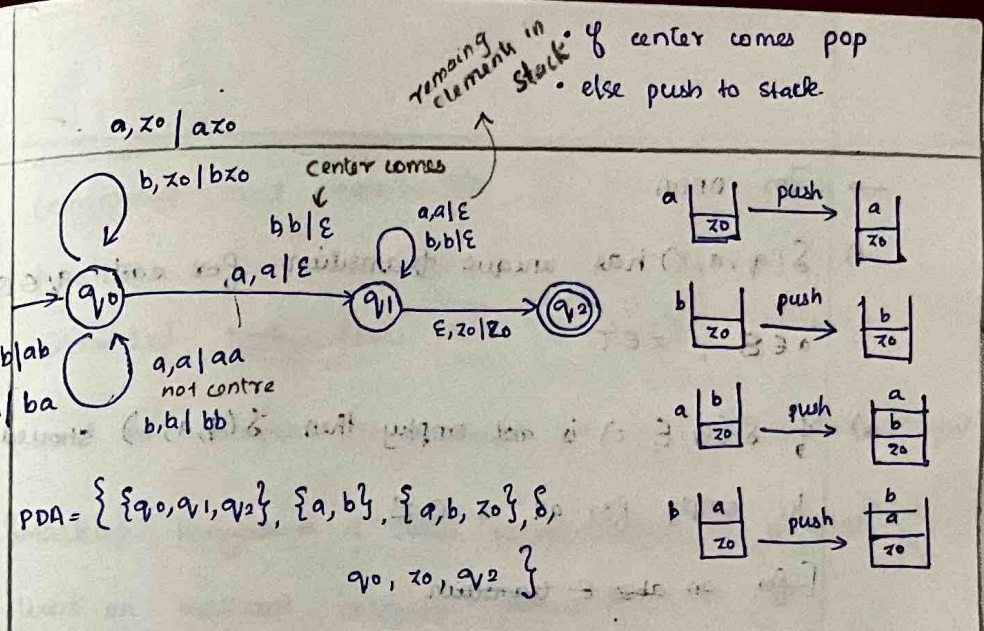
w, w^R ; w, w^R



1st element → push

and elem → may be central elem
→ may not be central elem } 2 possibilities.

this case only if two same symbols are in consecutive position as



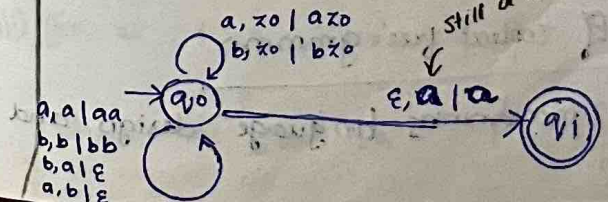
PDA = { { q_0, q_1, q_2 }, { a, b }, { a, b, z_0 }, δ, q_0, z_0, q_2 }

- Its called NDPDA bcz corresponding to same i/p symbol and same stack symbol (eg: a,a|aa), its moving to two states q_0 & q_1 at the same time.

Q. How can check whether Design a PDA to accept a language $L = \{ w \mid w \in (a,b)^* \text{ and } n_a(w) > n_b(w) \}$

a) $L = \{$ aaab, aa, abaa, ... $\}$

q_b if a_λ is in stack and if b comes, & a and b is popped and still there is an a left inside stack, it means $n_a(w) > n_b(w)$



→ For DPDA

i) $\delta(q, a, z)$ has unique transition for each $q \in Q$, $a \in \Sigma$, $z \in \Gamma$

ii) If $\delta(q, \varepsilon, z)$ is not empty then $\delta(q, a, z)$ should be empty for all $a \in \Sigma$

[~~for an abso ε transition~~

If both satisfied — DPDA

If one is not — NDPDA

- If one of the transition is ε transition, then for that particular stack symbol (z) there should not be a transition for other input symbol (a)

→ In this case, 1st condition is satisfied but 2nd is not.

∴ Its NDPDA (so write corresponding transitions)

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

$\delta_2(q_0, b, a) = (q_0, \varepsilon)$

Applications of context free grammar

i) Its used for programming language design and implementation.

ii) Commonly used CFL are for writing syntax:

Backus–Naur Form (BNF)

Extended Backus–Naur Form (EBNF)

2) Used in compiler construction, parsing and syntax analysis

3) Markup languages & data representation, it is used

4) Used in natural language processing

5) Used in theoretical computer scienc

Pumping lemma

- By using CFG we can compare only two strings at a time (like $a^n b^n$, not $a^n b^n c^n$)

Theorem

Let L be a CFL and n be an integer constant choose any string z in L where $|z| \geq n$, then split the string into 5 parts $z = uvwxy$ such that

i) $|vwx| \leq n$ ii) $|vx| \geq 1$

iii) for all $i \geq 0$, $uv^i wx^i y \in L$

eg: Show that $L = \{a^n b^n c^n \mid n \geq 1\}$ is not a CFL

$L = \{abc, aabbcc, aaabbbccc, \ldots\}$

Let L be a CFL

Let $n = 3$ and choose $z = aaabbbccc$

$\therefore |z| \geq n \Rightarrow q \geq 3$

divide z into 5 parts $\underbrace{a\,a}_{u}\,\underbrace{a}_{v}\,\underbrace{b}_{w}\,\underbrace{b\,b}_{x}\,\underbrace{c\,c\,c}_{y}$

i) $|vwx| \leq n$ ii) $|vx| > 1$ iii) for $i \geq 0$ $uv^i wx^i y \in L$

$|abb| = 3 \leq 3$ $|ab| = 2 > 1$ for $i = 0$, $uwy = aabbccc \notin L$

$\therefore$ given language is not CFL

Closure properties of Context Free language

	closed under CFL	not closed under CFL
1)	Union	1) Intersection
2)	Concatenation	2) complementation
3)	Kleen closure	3) Set difference.
4)	Reversed	
5)	homomorphism	

1) If L_1 & L_2 are a CFL, $L_1 \cup L_2$ is also CFL, then CFL are closed under union

$L_1 \cup L_2 = \{$ string from L_1 or string from $L_2\}$

L_1 $S_1 \to aA$, $A \to b$ $\qquad$ L_2 : $S_2 \to bBa$, $B \to aB \mid \varepsilon$

$L_1 \cup L_2$: $S \to S_1 \mid S_2$, $S_1 \to aA$, $A \to b$, $S_2 \to bBa$, $B \to aB \mid \varepsilon$

2) $L_1 \cdot L_2$

$S \to S_1 \cdot S_2$, $S_1 \to aA$, $A \to b$, $S_2 \to bBa$, $B \to aB \mid \varepsilon$

3) $(L_1)^*$

$S \to S_1 S \mid \varepsilon$

4) let $L_1 = 0^n 1^n$, $n \geq 1$ $\longrightarrow$ $L_1^R = 1^n 0^n$

$S \to 0S1 \mid 01$ $\qquad$ $S \to 1S0 \mid 10$

5) $L = \{0^n 1^m \mid m \geq 1\}$

$h(0) = ab$ $\quad h(1) = \varepsilon$

$S \to abS \mid ab$

6) $L_1 = \{0^n 1^m 2^m, n \geq 0, m \geq 0\}$ $\qquad$ $L_2 = \{0^n 1^m 2^n, n \geq 0, m \geq 0\}$

$L_1 \cap L_2 = \{0^k 1^k 2^k, k \geq 0\}$ $\longrightarrow$ not CFL

2) $L_1 \cap L_2 = \overline{L_1 \cup L_2}$ ∴ not closed

(not closed ↑, closed ↑)

3) $L_1 \cap L_2 = L_1 - (L_1 - L_2)$

↓ not closed — if assumed it closed, but is actually not else, so contrary.

Decision Properties of CFL

1) Emptyness problem
2) Finiteness problem
3) Membership problem

} decidable - able to find a solution or can find an algorithm to solve problem

4) Equivalence problem } - undecidable - cannot find sol or algo.

Emptyness problem

- It is used to check if the given grammar generated an empty language / not
- If start symbol produce useless symbols then the language is empty. So on removing useless symbols from start symbol we find a solution to emptyness problem.

eg: $S \rightarrow AB \mid BC$ (No A, B or C production) → useless

∴ this is empty language.

Q. check whether the given grammar generates empty lang. or if is empty grammar.

$S \rightarrow AB$

$A \rightarrow AA \mid a$

$B \rightarrow BB$

a) B does not produce terminal symbol → useless.

So productions containing B should be removed.

So $S \rightarrow AB$ and $B \rightarrow BB$ removed.

Here S itself is removed ∴ grammar generates empty lang.

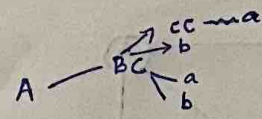
2) $S \rightarrow AB$

$A \rightarrow BC$

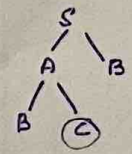
$B \rightarrow cc \mid b$

$c \rightarrow a \mid b$

a) A does not



A is useful.



C is not reachable from start symbol

∴ given grammar is not empty

Finiteness problem

If the given grammar generates finite gno language.

Steps to identify the given grammar is finite

Step: 1 Check whether given CFG is in chomsky NF, if not convert it into CNF.

2. Draw a graph for this grammar. Check if graph contains any loop or cycle. If loop exists, its a infinite lang, otherwise its finite

eg: $S \to AB$ ✓

$A \to BC$ ✓

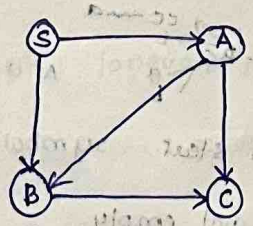
$B \to \check{C}c \mid b$ ✓

$c \to \check{a} \mid b$ ✓

chomsky, $A \to BC$, $A \to a$, $A \to \varepsilon$

1) ∴ given grammar is already in CNF

2)



(consider only non-terminals)

Here there is no loop. Hence its a ~~finite~~ generates a finite language. so we can find rank of non-terminals.

Rank of non-terminals = length of longest path from that variable (check outgoing edges)

Rank (C) = 0

Rank (A) = 2 (A → c, A → B → c)

Rank (B) = 1 (B → c)

Rank (S) = 3 (S → A → B → c , S → B → c)

Membership problem

- Used to check if the given string is a member of CFG or not.
- For finding membership we use CYK (cocke Younger Kasami) algorithm. It acceptable only when grammars in CNF

eg: $S \to AC \mid AB$

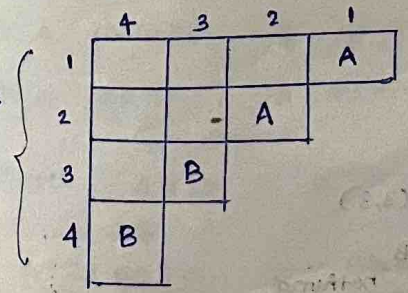
$C \to SB$

$A \to a$

$B \to b$

check the string aabb is a member of this grammar or not.

a) no. of substrings → $\frac{n(n+1)}{2} = \frac{4(4+1)}{2} = \frac{20}{2} = 10$ cells



(A → a) (B → b)

a a b b

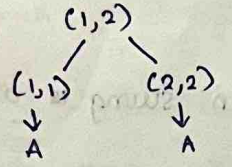
(1,1) (2,2) (3,3) (4,4)

1 length string

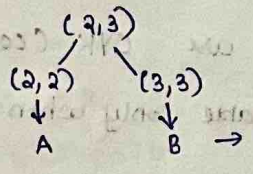
1) Take diagonal elements first

(1,1) → A
(2,2) → A ⋯→ (non-terminal determine the terminal symbol a)
(3,3) → B
(4,4) → B

2 length string

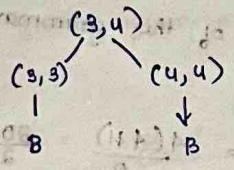


→ AA (no production determines AA so put '–')



→ AB is produced by S ie, S → AB of (2,3) with S

	4	3	2	1
1	S	–	–	A
2	C	S	A	
3	–	A	B	
4	B			



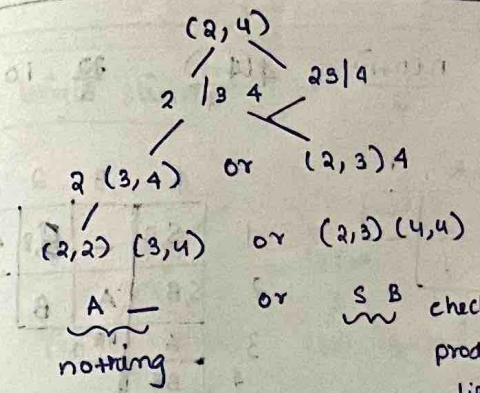
→ BB (no production)

3 length string

(1,3)
1|23 or 12|3
(1,1) (2,3) or (1,2) (3,3)
↓ or
A S — B
nothing

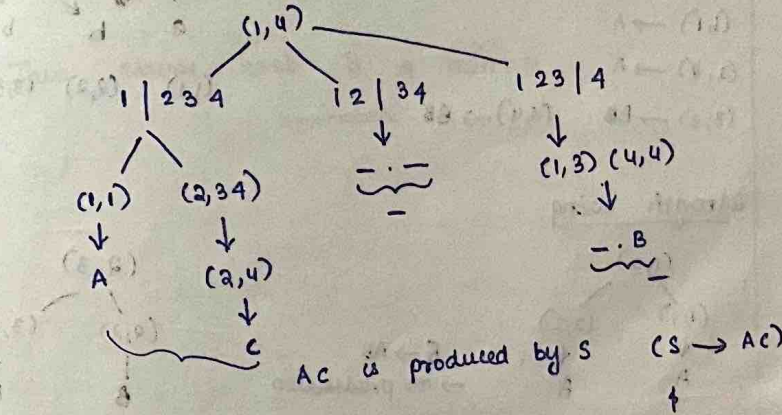
no production generating AS

	1	2	3	4
1	A			
2		A	–	
3			B	
4				



check SB is produced by any production. Here C → SB fill (2,4) with C

4 length string



AC is produced by S (S → AC)

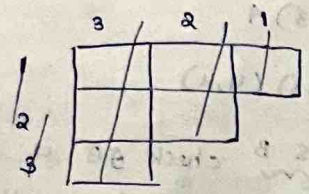
If start symbol is present in position (1,4) or 4 length string, then the given str is member of the given grammar or if not S then not a member of given string.

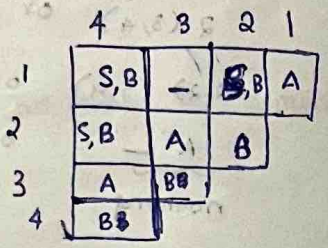
Thus aabb is a member of given string.

HW 1) Check 'abbb' is a member of given grammar.

S → AB
A → BB | a , BB → AB | b

no. of substring = $\frac{n(n+1)}{2} = \frac{4(4+1)}{2} = \frac{20}{2} = 10$ cells.





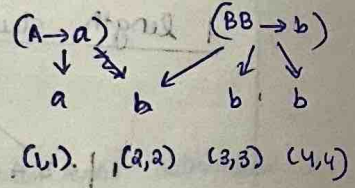
1 length string

diagonal elements

(1,1) → A

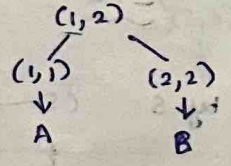
(2,2) → A

(3,3) → BB (4,4) → BB

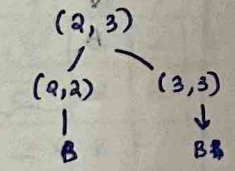


(1,1) (2,2) (3,3) (4,4)

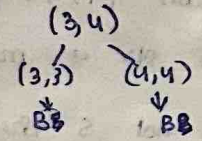
2 length string.



B → AB
→ no production generating AA

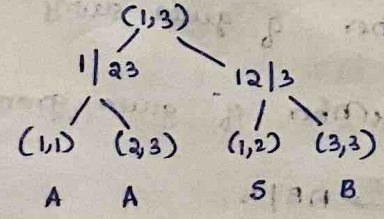


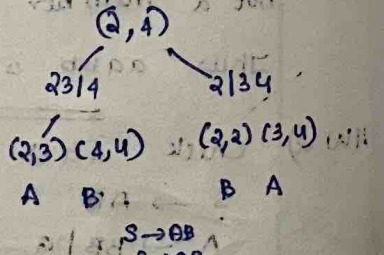
A → BB



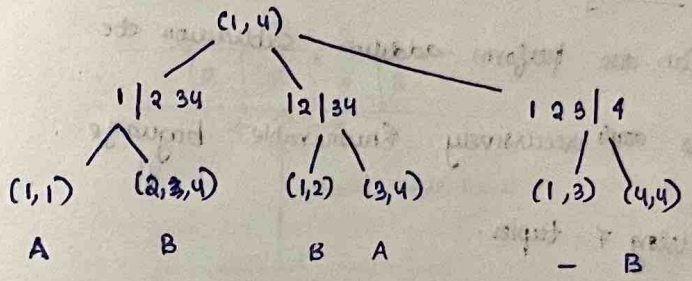
A → BB

3- length string





4- length string



S→AB B→AB

Thus string abbb is a member

11-8-2025

MODULE-3

Turing Machine

- Can also perform addition, subtraction etc.
- Its accepts recursively Enumerable language.

represented Using 7 tuples.

$TM = (Q, \Sigma, \tau, \delta, q_0, B, F)$

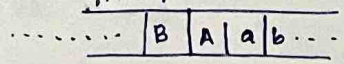
Q - finite set of states Σ - input alphabet ($\Sigma \in \tau$)

$\Sigma \subseteq \tau$, $B \subseteq \tau$ — τ - i/p tape alphabet B - blank symbol (also a part of tape alphabet τ)
(complete set of tape symbols)

F - set of final state.

q_0 - initial state δ - transition fn.

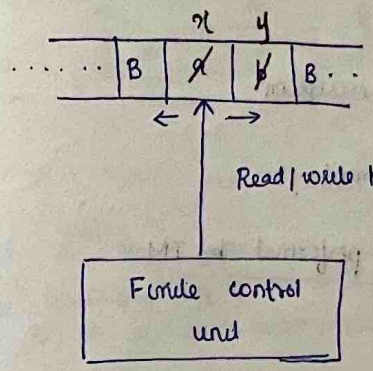
- Power of TM is equivalent to that of a computer.
- Input tape is unbounded (infinite) in both side.



(infinite)/
ie, input tape gives extra m/y in TM

- Can store any no. of i/p.
- each cell can store one input symbol at a time.

Components of TM



- ab is first symbol. Hence it points to a.
- By reading a, its replaced with x and b replaced by y
- R/w head can move in left as well as ~~write~~ right direction
- R/w performs read & write
- FCU decides which state we can go by controlling R/w head
- ~~$\delta : Q \times \tau$~~

τ = { a, b, B, X, Y } (B: current symbol; X, Y: additional symbols)

- τ contains I/P symbol, current symbol, addition symbol for logic.

$\delta : Q \times \tau \longrightarrow Q \times \tau \times \{L, R\}$ (Left or right)

$TM \rightarrow \delta : \{ (q_0, a) \longrightarrow (q_1, X, R) \}$

In general: $\delta : (q, x) \longrightarrow (p, Y, D)$

- q: current state
- p: next state
- Y: x replaced by y which a symbol in τ
- D: direction of R/w head after operation

TM can be represented using 3 notations.

i) transition table

ii) Instantaneous Description

iii) Transition diagram.

Various actions performed by TM

- changing state
- changing the i/p
- R/w head can move in left or right direction

Acceptance of a Language by TM

TM can perform 3 operations:

a) Halt and accept by entering into a final state

b) Halt and reject → δ(q,a) is not defined when coming to a state, if there is no transition for the symbol.

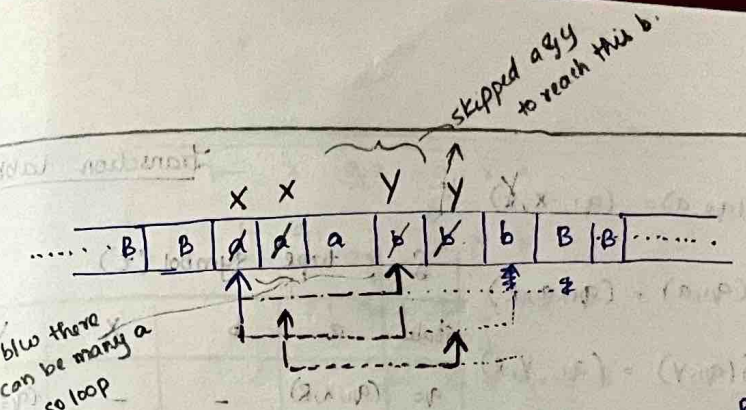
c) Turing M never halt and enters into an infinite loop

Q. Design a TM for a language $L = \{a^n b^n \mid n \geq 1\}$

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

Consider input str aaabbb inside the i/p tape.

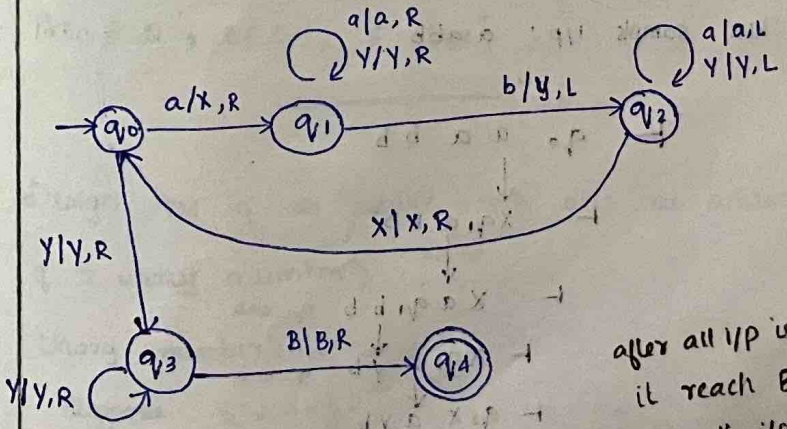
- Initially R/w head points to first i/p symbol a.



- Whenever a comes, its replaced with x and R/w head moves right until a first b space is found (it skips a in b/w)
- So when before B, its b and hence its b is replaced with y
- Now R/w is moved Left until it reaches X once replaced. replace x with x and is reached, moved right to next a and replace it with x.
- Again R/w head moves right until it reaches y. next b (in this step we skip as y & on to reach next b) reaching y take the b in the left and replace with y.
- This process is continues for all i/p

blank space logic is for palindrome

Skipping means replace with itself



after all i/p is reached, it reach B, replace it with itself and hault

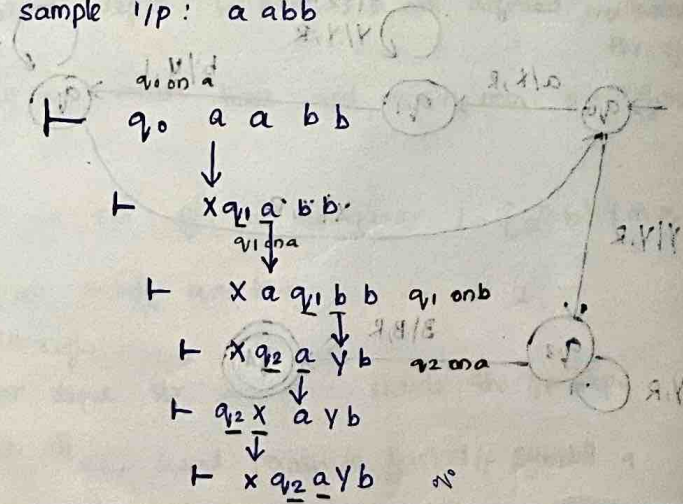
1. $\delta(q_0, a) = (q_1, X, R)$
2. $\delta(q_1, a) = (q_1, a, R)$
3) $\delta(q_1, Y) = (q_1, Y, R)$
4. $\delta(q_1, b) = (q_2, Y, L)$
5. $\delta(q_2, Y) = (q_2, Y, L)$
6. $\delta(q_2, a) = (q_2, a, L)$
7. $\delta(q_2, X) = (q_0, X, R)$
8. $\delta(q_0, Y) = (q_3, Y, R)$
9. $\delta(q_3, Y) = (q_3, Y, R)$
10. $\delta(q_3, B) = (q_4, B, R)$

Transition table

δ	tape symbol (τ)				
state	a	b	X	Y	B
q_0	(q_1, X, R)	–	–	(q_3, Y, R)	–
q_1	(q_1, a, R)	(q_2, Y, L)	–	(q_1, Y, R)	–
q_2	(q_2, a, L)	–	(q_0, X, R)	(q_2, Y, L)	–
q_3	–	–	–	(q_3, Y, R)	(q_4, B, R)
q_4	–	–	–	–	–

Instantaneous description represented by taking a sample:

here sample i/p: a abb



$\vdash q_0\ a\ a\ b\ b$

$\vdash X q_1\ a\ b\ b$

$\vdash X\ a\ q_1\ b\ b$

$\vdash X\ q_2\ a\ Y\ b$

$\vdash q_2\ X\ a\ Y\ b$

$\vdash X\ q_2\ a\ Y\ b$

$\vdash X\ a\ q_0\ Y\ b$

$\vdash X\ X\ q_1\ Y\ b$

$\vdash X\ X\ Y\ q_1\ b$

$\vdash X\ X\ q_2\ Y\ Y$

$\vdash X\ q_2\ X\ Y\ Y$

$\vdash X\ X\ q_0\ Y\ Y$

$\vdash X\ X\ Y\ q_3\ Y$

$\vdash X\ X\ Y\ Y\ q_3\ B$

$\vdash X\ X\ Y\ Y\ B\ q_4$ (final)

Accept

TM to accept the language

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

$M = \{Q, \Sigma, \tau, \delta, q_0, B, F\}$

here $Q = \{q_0, q_1, q_2, q_3, q_4\}$,

$\Sigma = \{a, b\}$, $\tau = \{a, b, X, Y, B\}$, δ, (write all)

$q_0 \in Q$, $B \in \tau$, $F = \{q_4\}$

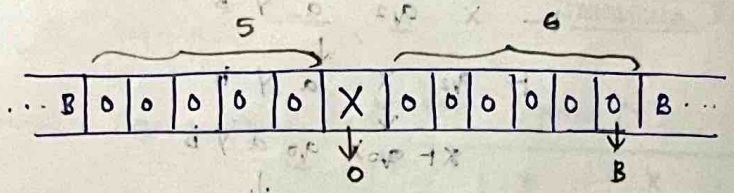
Q. Design TM ~~q~~ ~~adder~~ which acts as adder (addition of 2 unary numbers)

a) Unary number: 0 or 1

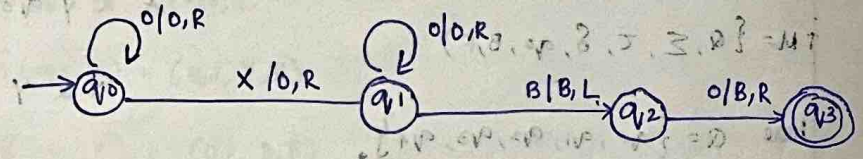
Suppose $\Sigma = \{0\}$

here $X = 5 = 00000$ $Y = 6 \rightarrow 000000$ } To seperate x and y seperator X is used.

6+
5
11



* To get ans 11, the seperator X is replace with 0 and last 0 is replaced with B → The R/w head is moved until X and is replaced with 0 and then R/w head moves until B then B is replaced with B and moved left and the last 0 is replaced with B.



HW Design a TM as a subtractor using unary numbers (m,n)

proper subtraction

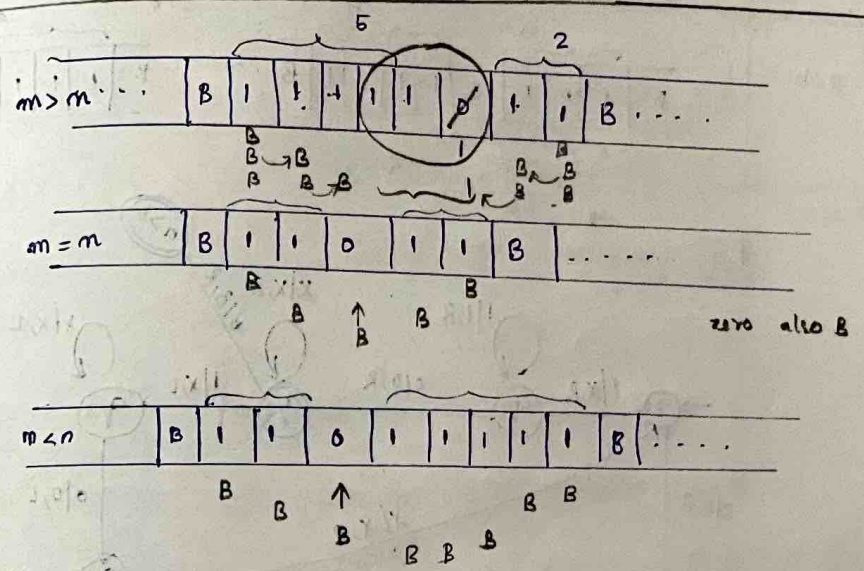
$$f(m,n) = \begin{cases} m-n & \text{if } m>n \\ 0 & \text{if } m \le n \text{ full blank.} \end{cases}$$

m=5 n=2 3

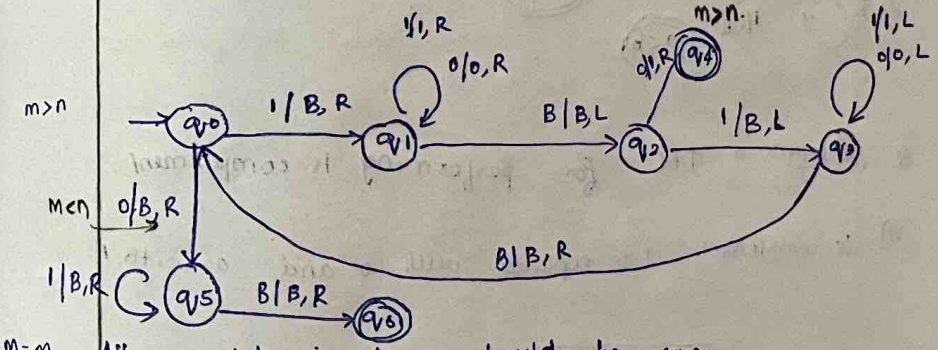
a) Suppose $\Sigma = \{1\}$

m=5 → 1 1 1 1 1 n=2 → 1 1 after sub 1 1 1

using 0 as seperator here.



whenever we see 1 make it blank and 0 as 1 at end

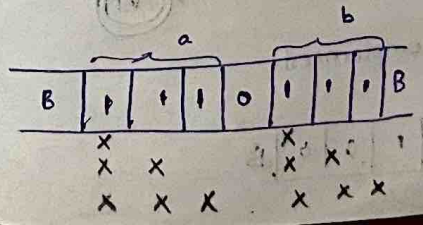


All symbols in tape should be zero.

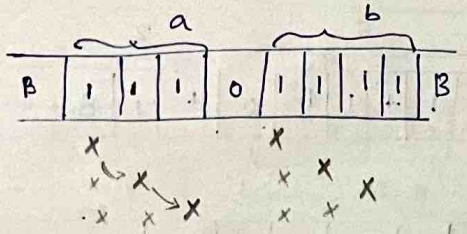
Q. Turing machine as Comparator

a = b, a < b, a > b

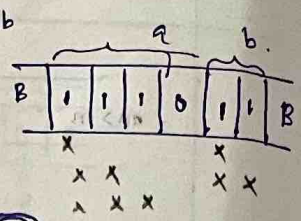
a=b

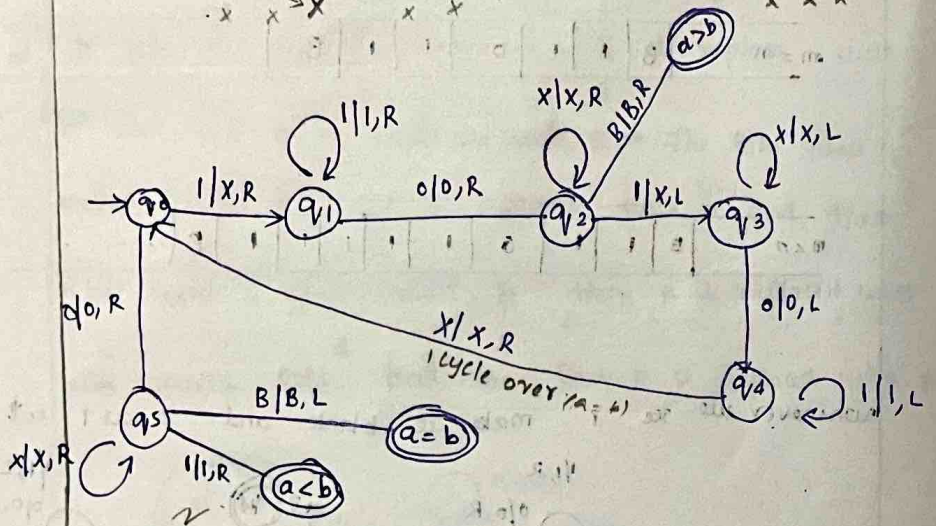


a<b



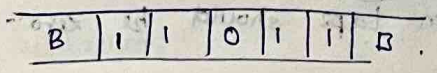
a>b

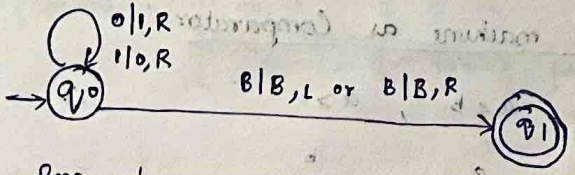




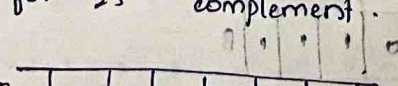
Q. Design a TM for performing 1s complement.

a) 1s complement : 1 is replaced with 0 and 0 with 1.



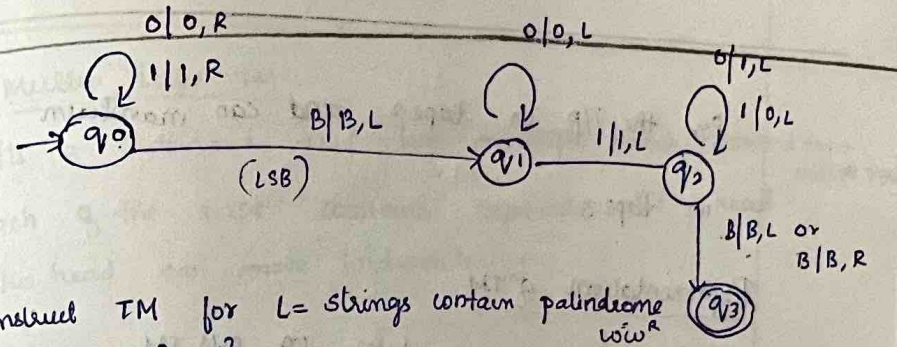


2) TM for 2's complement.



LSB ending
keep as it
then 0 as 1
1 as 0

0 1 1 1 0 0



HW 3 Construct TM for L= strings contain palindrome $w\bar{w}w^R$ over $\Sigma = \{0,1\}$

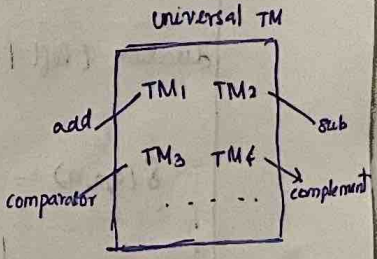
Universal Turing Machine

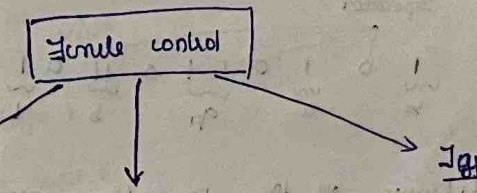
- UTM can do any operation that computer can do.

Structure of UTM:

→ Have finite control unit

→ Three tapes unit





Tape 1 used for representation of TM. All the TM that are needed are placed on 1 (for add, sub, comp..)

Tape 2 used to save the i/p

Tape 3 used to represent the state of the TM

With the help of FC we can execute any TM in Tape 1

for the i/p in tape 2 and can maintain state in tape 3

Representation of TM

Assume we have 3 states in our TM

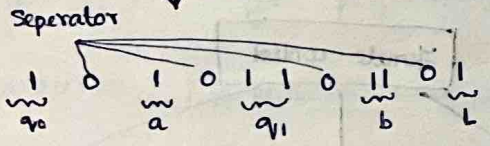
q_0, q_1, q_2 encoded as → 1, 11, 111

The i/p symbols are a, b encoded as → 1, 11

direction (left / right) encoded as → 1, 11

$\delta(q_0, a) = [q_1, b, L]$

↓ encoded as

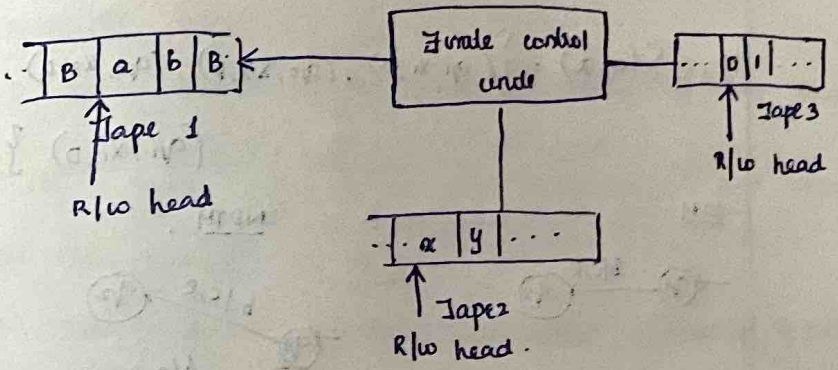


- Likewise all the transition are converted and placed within tape 1. This is how it is stored inside computer
- UTM structure is similiar to multiple tape TM.
- UTM is similar to a compiler.

Types of TM

1) Multip tape TM

- Its a standard TM with multiple tapes. (Standard TM + multiple tape)
- Each of the tape contains seperate R/W head.
- R/W head can move independently.



no. of tapes in TM = 3

eg: $\delta(q, a, b, c) = (p, \alpha, y, z, L, R, S)$

(symbol on tape 1, tape 2, tape 3; replaced symbols; next state)

L - R/W head of tape 1

R - R/W head of tape 2

S - R/W head of tape 3 (stationary not left nor right)

- Power of multi-tape TM = P (single tape TM)

ie whatever language accepted by single tape TM can be accepted by multitape TM and vice-versa.

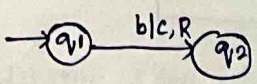
2) Non-deterministic TM.

Is same as DTM. Only difference is in transition fn.

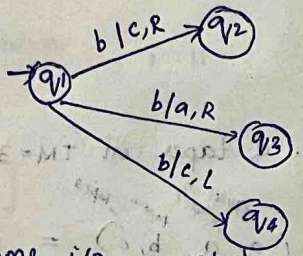
$\delta : Q \times \Sigma$ to $2^{Q \times \Gamma \times \{L,R\}}$

ie., $\delta(q_i, x) = \{(q_1, x_1, D), (q_2, x_2, D), (q_3, x_3, D), \ldots (q_i, x_i, D)\}$

DTM.

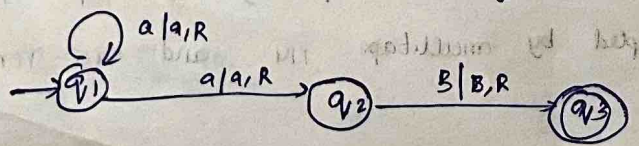


NDTM.



- NDTM corresponding to same i/p symbol we are going to different state and is replaced by different symbol
- P(DTM) = P(NDTM)
- DTM is a type/subset of NDTM

eg: Design a NDTM that accept strings end with a over $\Sigma = \{a\}$



Turing machine languages

2 types:

i) Recursive Lang ii) Recursively enumerable lang

Recursive Lang

A lang. L is said to be recursive if there exist a TM which will accept all strings in L and reject all the strings not in L.

- ie, TM will always ~~hold~~ halt either it accept & halt or reject & halt.
- Recursive languages are called Turing decidable language.

ii) Recursively Enumerable lang.

- A language L is said to REL, if there exist a TM which will accept (:. halt) for all the input strings which are in Lang. But may or may not halt for all input string which are not in L.
- REL is called Turing recogonisable ~~grammar~~ language.

Decidability & Undecidability

- A problem is said to decidable if we can always construct an algorithm or TM to solve the problem.
- A lang is decidable if there is a TM that accept & halt on every i/p strings with a solution either yes (accept ~~reject~~) or No (reject)
- decidable problems are also called Turing decidable problem. because it will always be in halt (always there is a result either (accept or reject))
- A language L is decidable if it is a recursive lang.

∴ All decidable languages are the recursive languages and vice versa.

Partially decidable problems

PDP are those for which a TM halt on the i/p accepted by it. But it can either halt or ~~hold~~ loop forever on the i/p which is rejected by the TM.

- A lang L is semi decidable or partially decidable if L is recursive enumerable language.

Undecidable problems

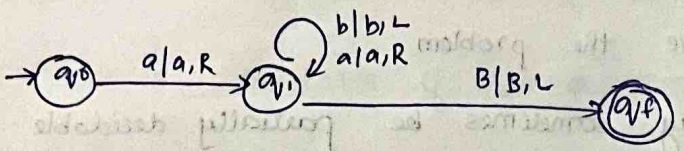
- A problem is undesirable if there is no algo/TM to solve the problem.
- UP may sometimes be partially decidable but not decidable.
- If a lang is not even partially decided then there exist no TM for that Lang.

Recursive language	TM will always Halt (Turing decidable)
Recursive enumerable language	TM will halt sometimes & may not halt sometimes (Turing recogonisable)
Decidable language	Recursive Lang.
Partially decidable language	Recursively enumerable language.
undecidable problems	No TM for that lang.

eg:
i) Halting problem of a TM
ii) Post correspondance problem.
iii) Fermat's last ~~th~~ theorem : $a^n + b^n = c^n \quad \forall \; n \geq 2$

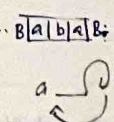
Halting problem of TM

$\Sigma = \{a, b\}$ $\quad L = aa^+$ (i/p)



This TM will accept aba.

Suppose if i/p is baa, then TM will reject string.



If i/p is aba → ~~accept~~ never halting (always on loop b/w states q0 and q1)

→ For all accepted inputs, the TM halts. But for rejected i/p, TM may or may not halt. ie, whether the given machine will halt or not, there is no algorithm for it. This is Halting problem of TM.

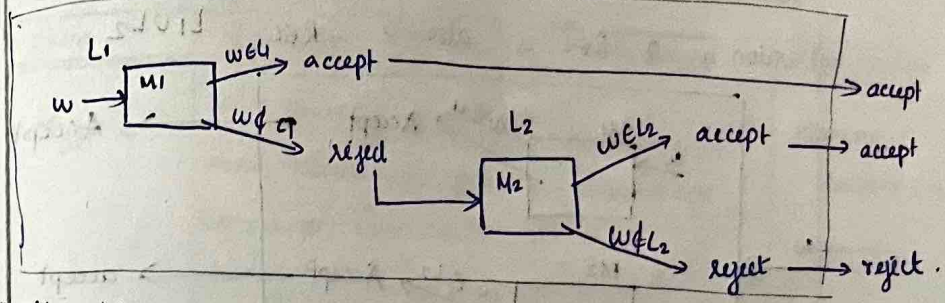
Properties of recursive lang & recursively enumerable lang

i) Properties of recursive lang (Union, intersection, complement)

i) Union of 2 recur. Lang is recursive.

L_1 – palindrom string. L_2 – all inputs that starts with a.

eg: $L_1 \cup L_2$

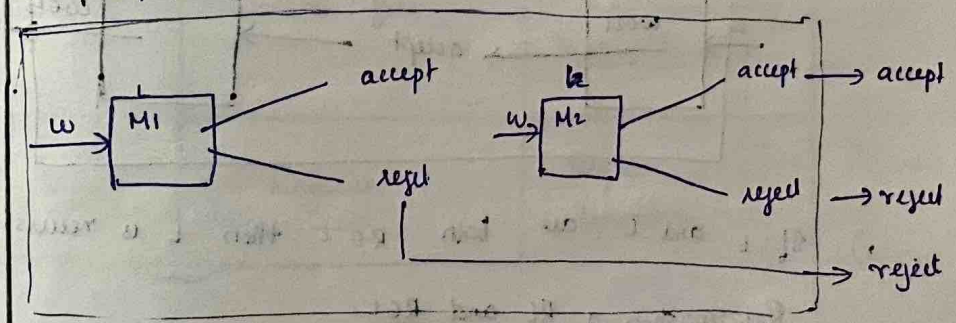


If the i/p is rejected by L_1 & L_2 and i/p is accepted either by L_1 or L_2.

ii) Intersection of 2 recursive lang is also a recursive lang.

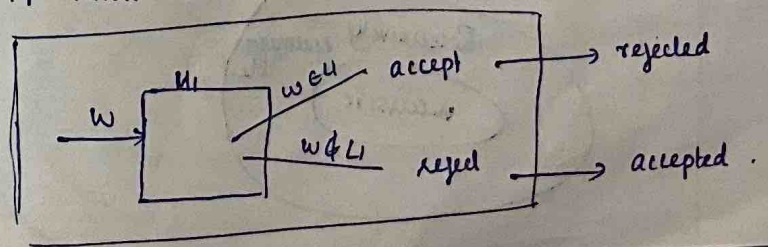
If either of one machine reject, entire $L_1 \cap L_2$ rejects language.

To accept, both machine must accept.



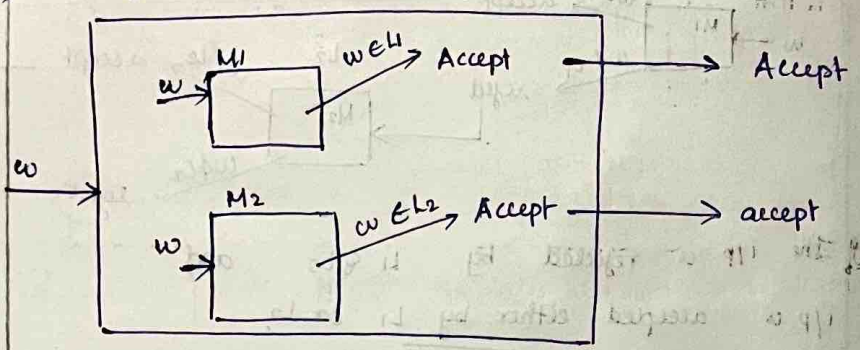
iii) Complement of 2 recur. L is also a recursive lang.

L_1 – i/p start with a $\quad \overline{L_1}$ – not start with a

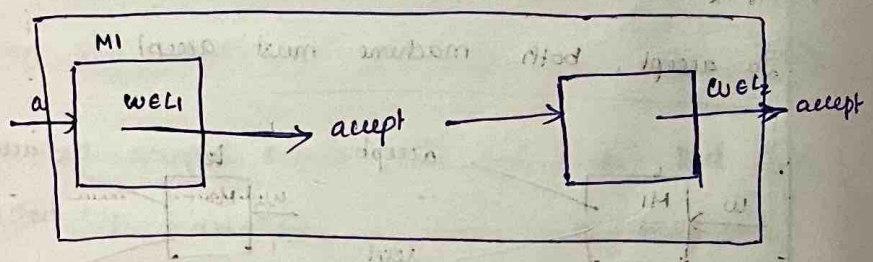


Properties of Recursively enumerable language

1) Union of 2 REL is also a REL. $L_1 \cup L_2$



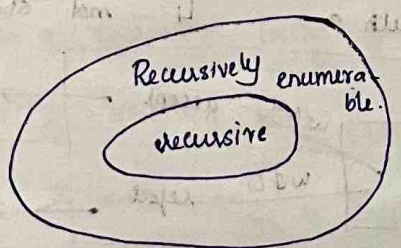
2) $L_1 \cap L_2$. Intersection of 2 REL is also REL



3) If L and $\bar{L}$ are both REL then L is recursive.

Relationship of RL and REL

Recursive lang. is a subset of Recursively EL.

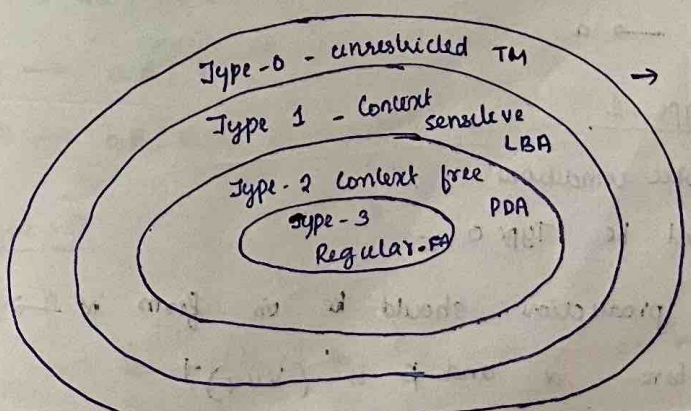


25-8-24.

MODULE-4

Chomsky Heirarchy

class	Grammar (generator)	Languages	Automaton (acceptor)
Type 0	unrestricted grammar / phase structured grammar / Recursive enumerable grammar.	Recursive enumerable	Turing Machine (TM)
Type 1	Context sensitive / length increasing / non-contracting grammar	Context-sensitive language.	Linear Bounded Automata (LBA)
Type 2	Context-free	Context-free language	Push down Automata (PDA)
Type 3	Regular	Regular language	Finite Automata (FA)



→ Relationship b/w Languages

Type 0

If a grammer $G = (V, T, P, S)$ is said to be type 0 if all the production are of the form

$\alpha \longrightarrow B$

α is $(V \cup T)^+$ β is $(V \cup T)^*$

↳ any string of non terminal and terminals with atleast 1 non terminal.

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

V → non terminal
T → terminals
P – production rules
S – start symbol

REG
↓
REL
↓
TM

eg: $S \longrightarrow aAb \mid \varepsilon$

$aA \longrightarrow bAA$

$bA \longrightarrow a$

- It is unrestricted: no restriction in length.

Type 1

- Three conditions:

i) Must be Type 0

ii) all production should be in form $\alpha \longrightarrow \beta$ where α and β is $(V \cup T)^+$

iii) $|\alpha| \leq |\beta|$ (restriction)

- It is an ε-free grammar.
- Exception is that start symbol may contain / can generate ε provided if the start symbol not appears on RHS of any production.

eg: $S \longrightarrow aAb$

$aA \longrightarrow bAA$

$bA \longrightarrow aa$

Type 2

i) Should be Type 1

ii) If production are of form $A \longrightarrow \alpha$ where $\alpha \in (V \cup T)^*$, A is a single non-terminal. ie $|A| = 1$ (restriction)

eg: $S \longrightarrow aB \mid bA \mid \varepsilon$

$A \longrightarrow aA \mid b$

$B \longrightarrow bB \mid A \mid \varepsilon$

Type - 3

i) Type 2

ii) It can be either be left linear or right linear.

Left linear

$A \rightarrow B\alpha \mid \beta$

$A, B \in V$

$|A| = |B| = 1$

$\alpha, \beta \in T^*$

eg: $S \rightarrow Baa \mid Abb \mid \varepsilon$

$A \rightarrow Aa \mid b$

$B \rightarrow Bb \mid a \mid \varepsilon$

Right linear

$A \rightarrow \alpha B \mid \beta$

$A, B \in V$

$|A| = |B| = 1$

$\alpha, \beta \in T^*$

eg: $S \rightarrow aaB \mid bbA \mid \varepsilon$

$A \rightarrow aA \mid b$

$B \rightarrow bB \mid a \mid \varepsilon$

- Combination of left and right linear grammar is called linear grammar ← it's not type 3
- The automata which accepts context sensitive grammar is called linear Bounded Automata.

Linear Bounded Automata (LBA)

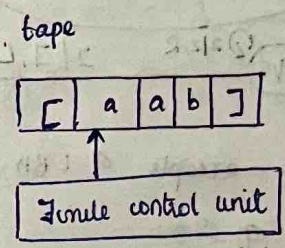
There are 7 tuples.

$(Q, \Sigma, \tau, \delta, q_0, F, M)$

M – marker : of two types → left end marker, right end marker

τ – tape symbol

- Like TM there is i/p tape and finite control unit.
- In LBA, the tape is not infinite. It is bounded end markers.



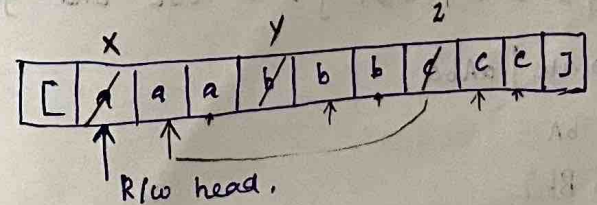
→ Size of i/p tape = linear function of i/p. That is why its called LBA.

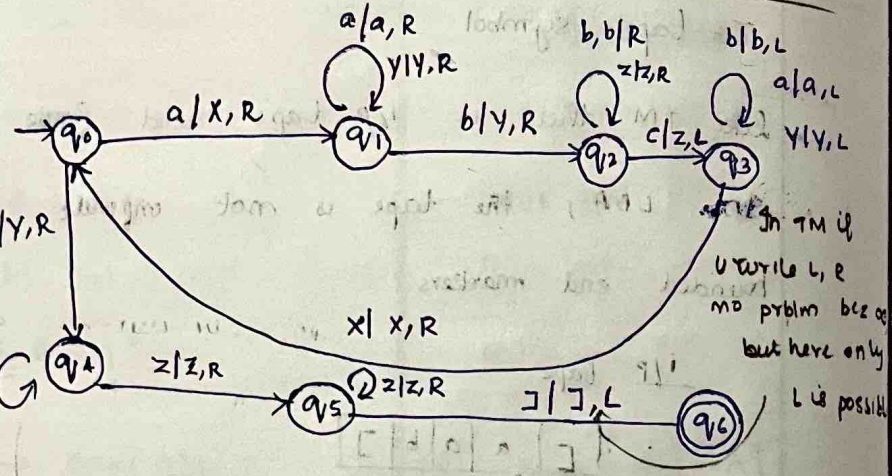
eg: aabb = 4x1 = 4

Q Design a LBA for the language $L = \{a^n b^n c^n \mid n \geq 1\}$

Q. $n = 1$: abc $n = 2$, aabbcc

$L = \{abc, aabbcc, aaabbbccc\ldots\}$.



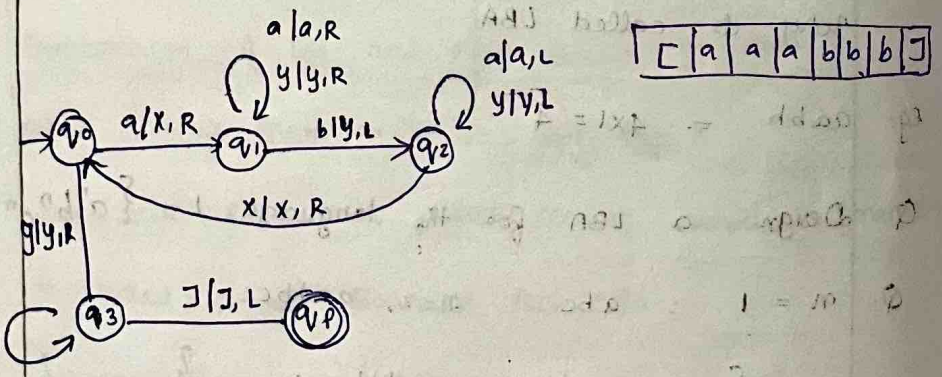


In TM if
U write L, R
no prblm bcz of
but here only
L is possible

2) Design a LBA which accepts a LBA which accepts

$L = \{a^m b^m \text{ where } m \geq 1\}$

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



[| a | a | b | b | b |]

3. $L = \{a^n b^n c^n \mid n \geq 1\}$ is a CSL and grammar is

$S \longrightarrow abc \mid aAbc$

$Ab \longrightarrow bA$

$Ac \longrightarrow Bbcc$

$bB \longrightarrow Bb$

$aB \longrightarrow aa \mid aaA$

Test whether aabbcc is in L(G1)

ans) Check for substitutions that can give aabbcc

$S \longrightarrow aAbc$ (Ab → bA)

$S \longrightarrow abAc$ (Ac → Bbcc) not only non-terminals, consider its context also (nearest symbol of non-terminal)

$S \longrightarrow abBbcc$ (bB → Bb)

$S \longrightarrow aBbbcc$ (aB → aa)

$S \longrightarrow aabbcc \in L(G1)$

Equivalence of Regular Grammar & FA (RG → FA, FA → RG)

Here 2 conversions are required:

i) Conversion of RG into FA

a) Conversion of Right regular grammar into FA

$G = (V, T, P, S)$ be RG (V: non-term)

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

States = non terminals (V) of RG

no. of states in automata = no. of non-terminals + 1

(additional state added will be the final state)

Σ - T of RG

$\delta \rightarrow$ P or RG

q_0 - S of RG

F - additional state added

3 Rules

1) If the productions is of the form $A_i \rightarrow aA_j$ then the transition fn is $\delta(A_i, a) = A_j$

2) If production is of form, $A_i \rightarrow a$, then introduce a new state called final state. Corresponding transition fn, $\delta(A_i, a) = A_j$ where A_j is the newly added state (final state)

3) If the production is of form $A \rightarrow \varepsilon$ Corresponding transition fn, $\delta(A, \varepsilon) = A$ A will be final state.

Q. Convert the following right RG to FA

$S \rightarrow 0A \mid 1A$

$A \rightarrow 0A \mid 1A \mid +B \mid -B$

$B \rightarrow 0B \mid 1B \mid 0 \mid 1$

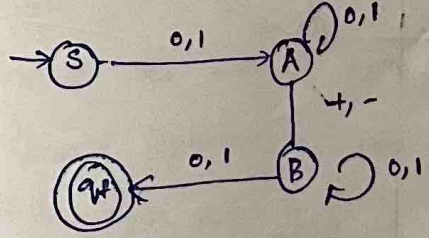
non-terminals

FA : $(Q, \Sigma, \delta, q_0, F)$

$Q = \{S, A, B\}$ $\Sigma = \{0, 1, +, -\}$ $q_0 = \{S\}$ $F = \{$

F = Added new state.

Productions	Transitions	
$S \rightarrow 0A$	$\delta(S,0) = A$	$A_i \rightarrow aA_j$
$S \rightarrow 1A$	$\delta(S,1) = A$	
$A \rightarrow 0A$	$\delta(A,0) = A$	
$A \rightarrow 1A$	$\delta(A,1) = A$	
$A \rightarrow +B$	$\delta(A,+) = B$	
$A \rightarrow -B$	$\delta(A,-) = B$	
$B \rightarrow 0B$	$\delta(B,0) = B$	
$B \rightarrow 1B$	$\delta(B,1) = B$	
$B \rightarrow 0$	$\delta(B,0) = q_f$	$A_i \rightarrow a$
$B \rightarrow 1$	$\delta(B,1) = q_f$	



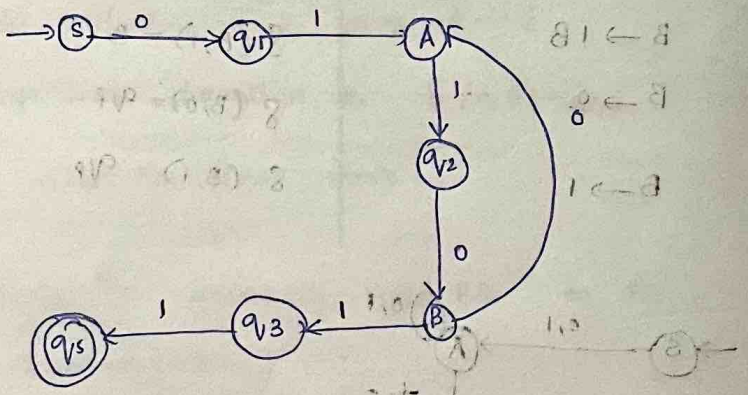
2)

$S \rightarrow 01A$

$A \rightarrow 10B$

$B \rightarrow 0A \mid 11$

Productions	Transition (δ)
$S \rightarrow 01A$	$\delta(S, 01) = A$
$A \rightarrow 10B$	$\delta(A, 10) = B$
$B \rightarrow 0A$	$\delta(B, 0) = A$
$B \rightarrow 11$	$\delta(B, 11) = q_f$



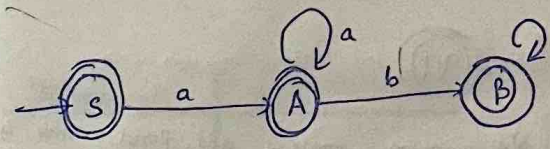
3)

$S \rightarrow aA \mid \varepsilon$

$A \rightarrow aA \mid bB \mid \varepsilon$

$B \rightarrow bB \mid \varepsilon$

Production	transition
$S \rightarrow aA$	$\delta(S, a) = A$
$S \rightarrow \varepsilon$	S is the final state
$A \rightarrow aA$	$\delta(A, a) = A$
$A \rightarrow bB$	$\delta(A, b) = B$
$A \rightarrow \varepsilon$	A is the final state
$B \rightarrow bB$	$\delta(B, b) = B$
$B \rightarrow \varepsilon$	B is the final state



FA, $M = \{Q, \Sigma, \delta, q_0, F\}$ $\quad Q = \{S, A, B\}$

$\Sigma = \{a, b\}$, $q_0 = \{S\}$, $F = \{S, A, B\}$

Conversion of Left Regular grammar to FA

Step 1) Reverse the RHS of every production.

2) construct the NFA

3) Interchange initial & final state

4) Change the direction of edges.

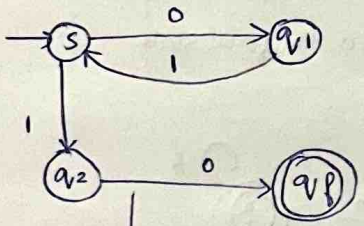
eg $S \longrightarrow S10 \mid 01$

1) Reverse RHs of every production

$S \longrightarrow 01S \mid 10$

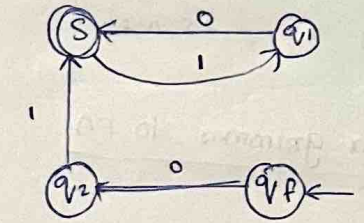
2) Construct the NFA

Production	Transition
$S \to 01S$	$\delta(S, 01) = S$
$S \to 10$	$\delta(S, 10) = q_f$



3) Interchange initial and final state & change.

4) direction of edge.



Convert FA to ~~Right~~ RG

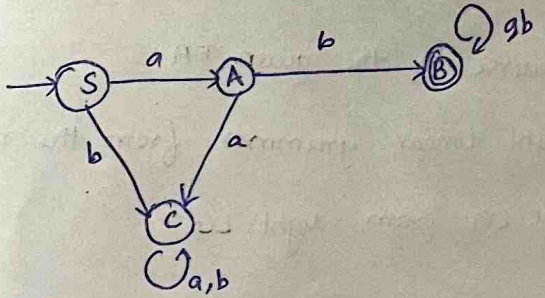
a) Convert FA to right RG.

Step 1 For each transition of the form $\delta(Ai, a) = Aj$ then corresponding production will be,

$Ai \longrightarrow aAj$

2) If $A \in F$, ie, A is the final state in FA, then introduce the production $A \to \varepsilon$

eg:



Transition	production
$\delta(S, a) \Rightarrow A$	$S \to aA$
$\delta(S, b) \to C$	$S \to bC$
$\delta(A, a) \to C$	$A \to aC$
$\delta(A, b) \to B$	$A \to bB$
$\delta(B, a) \to B$	$B \to aB$
$\delta(B, b) \to B$	$B \to bB$
$\delta(C, a) \to C$	$C \to aC$
$\delta(C, b) \to C$	$C \to bC$
B is the final state	$B \to \varepsilon$

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

$V = \{S, A, B, C\}$, $T = \{a, b\}$, $S = \{S\}$ $P = \{ \ldots \}$.

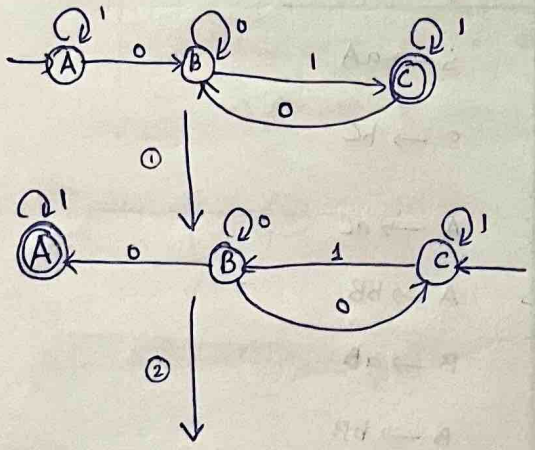
Convert the FA to Left RG / left Linear G

Step 1: Obtain the reverse of the given FA.

2. Obtain the right linear grammar from the reversed FA

3. Obtain the left LG from right LG

eg: Obtain the left LG from the given FA.



$C \longrightarrow 1C \mid 1B$

$B \longrightarrow 0B \mid 0A \mid 0C$

$A \longrightarrow 1A \mid \varepsilon$

④ ⟶

$C \longrightarrow C_1 \mid B_1$

$B \longrightarrow B_0 \mid A_0 \mid C_0$

$A \longrightarrow A_1 \mid \varepsilon$

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

$V = \{A, B, C\}$, $T = \{0, 1\}$, $P = \{\ \}$, $S = \{C\}$.

2) Obtain right LG for the language $L = \{a^n b^m \mid n \geq 2, m \geq 3\}$

$n \geq 2$ so 1 more a & 3 or more b

a) $S \longrightarrow aaA$ remaining a and b in A

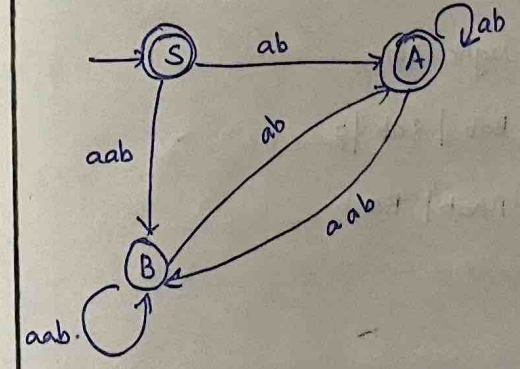
$A \longrightarrow aA \mid bbbB$

$B \longrightarrow bB \mid \varepsilon$

3) Obtain right LG for Regular expression $(aab)^* ab)^*$

$(aab)^* ab)^*$ → 0 or more → start final state

first construct corresponding FA



RLG

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

$V = \{S, A, B\}$

$T = \{a, b\}$

$P = \{ S \longrightarrow abA \mid aabB \mid \varepsilon$

$A \longrightarrow abA \mid aabB \mid \varepsilon$

$B \longrightarrow abA \mid aabB \}$

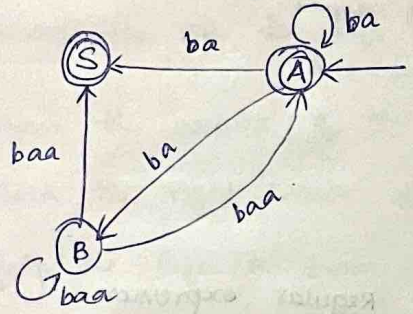
$S = \{S\}$

binary tree

4) Obtain Left LG for RE $(aab)^* ab)^*$

a) 1) construct FA corresponding to RE.

2) Reverse the FA.



3) Obtain RLG for reversed FA.

$A \longrightarrow baA \mid baB \mid baS \mid \varepsilon$

$B \longrightarrow baaS \mid baaA \mid baaB$

$S \longrightarrow \varepsilon$

4) Reverse the RHS of right LG,

$A \longrightarrow Aab \mid Bab \mid Sab \mid \varepsilon$

$B \longrightarrow Saab \mid Aaab \mid Baab$

$S \longrightarrow \varepsilon$

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

$V = \{A, B, S\}$ $T = \{a, b\}$ $P = \{$ $\}$

$S = \{A\}$

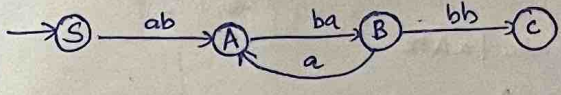
5) Obtain the Left LG for the righ LG. given.

$S \longrightarrow abA$

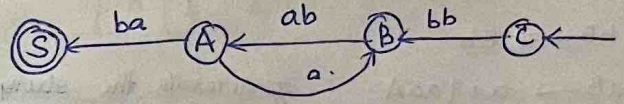
$A \longrightarrow baB$

$B \longrightarrow aA \mid bb$.

a) 1) Construct FA for given right LG.



2) Reverse the FA



3) Obtain RLG for the reversed FA

$C \longrightarrow bbB$

$B \longrightarrow abA \}$

$A \longrightarrow ba\$ \mid aB$ c is merely considered state on taking reverse S not considered

4) Reverse RHS of every production of RLG

$C \longrightarrow Bbb$

$B \longrightarrow Aba$

$A \longrightarrow ab \mid Ba$

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

$V = \{C, B, A\}$ $T = \{a, b\}$, $P = \{$ $\}$

$S = \{C\}$

Equivalence of LBA and CSL

Language accepted by LBA is context sensitive lang.

Theorem

If L is a CSL then it is accepted by some LBA.

Consider CSG,

$S \longrightarrow abc \mid aAbc$

$Ab \longrightarrow bA$

$Ac \longrightarrow Bbcc$

$bB \longrightarrow Bb$

$aB \longrightarrow aa \mid aaA$ generates the string aabbcc.

Derivation process of CSG to derive the string aabbcc,

$S \longrightarrow aAbc$ $(Ab \rightarrow bA)$

$S \longrightarrow abAbc$ $(Ac \rightarrow Bbcc)$

$S \longrightarrow abBbcc$ $(bB \rightarrow Bb)$

$S \longrightarrow aBbbcc$ $(aB \rightarrow aa)$

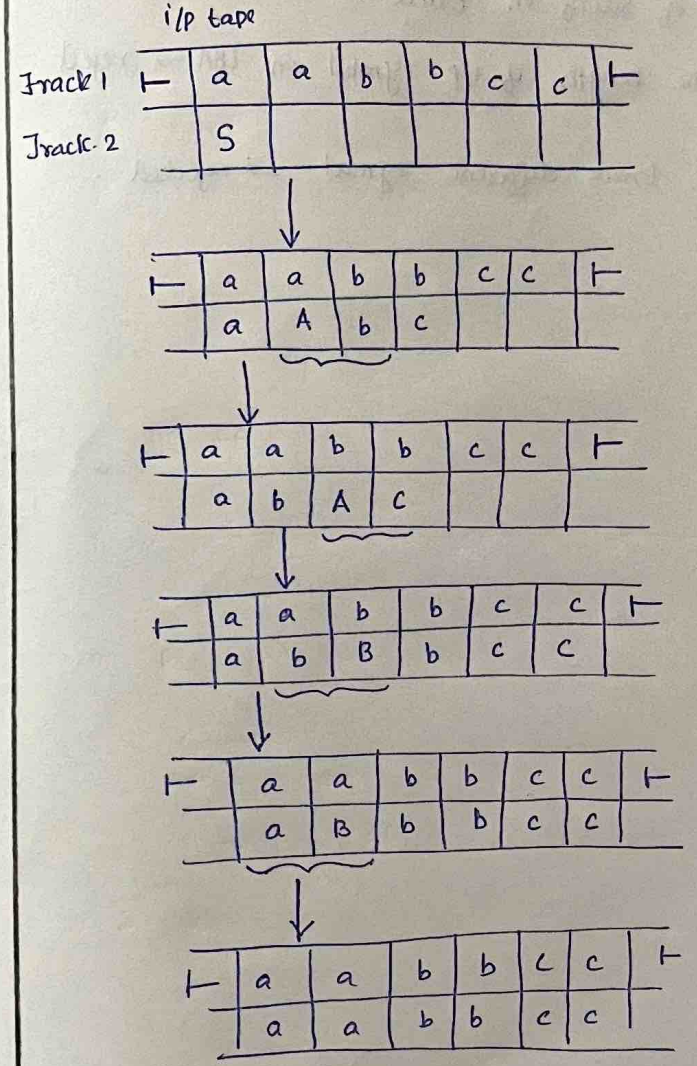
$S \longrightarrow aabbcc$

If the same process can be implemented by a LBA, both are equivalent (LBA & CSL)

For LBA, construct 2 tapes tracks.

track - 1 i/p symbol

track - 2 computation



If track 1 and track 2 have same i/p symbol then the LBA will accept the language.

The length cannot be more than the LBA during derivation. If length is more, it gets rejected.

- Length of string in track 2 cannot be more than the length of I/P symbol in LBA → reject.
- track-1, track different symbol → rejected.