Name: Madhamsetty Charitha

USN: 1RV18IS023

THEORY OF COMPUTATION

Part-A (Assignment)

Paper-1 (16IS52)

PART-A

1

1.1 Given language: $a^*b^*(ba)^*a^*$

Ans: bab

1.2 $(1^*01^*01^*01^*)^*$

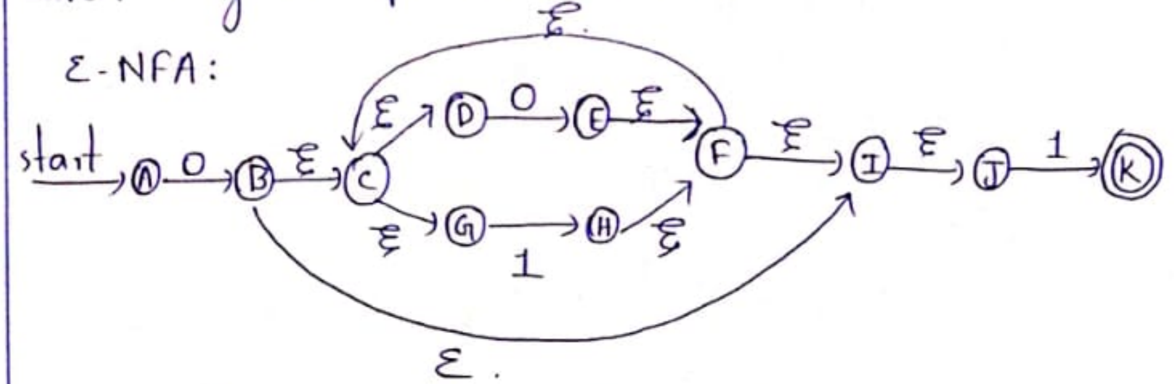
1.3 $L_2 = \{ww^R \mid w \in (a+b)^*\}$ is not regular.

$L_1 = \{ww \mid w \in (a+b)^*\}$ is not regular.

$L_4 = \{a^n b^m \mid m = n+5\}$ is not regular.

1.4 Given regular expression: $0(0+1)^*1$.

ε-NFA:



1.5 $S \rightarrow AB \mid aaB$ Take the string: aab.

$A \rightarrow a \mid Aa$

$B \rightarrow b$

LMD

$S \rightarrow AB$

$S \rightarrow AaB \ (A \rightarrow Aa)$

$S \rightarrow aaB \ (A \rightarrow a)$

$S \rightarrow aab \ (B \rightarrow b)$

LMD

$S \rightarrow aaB$

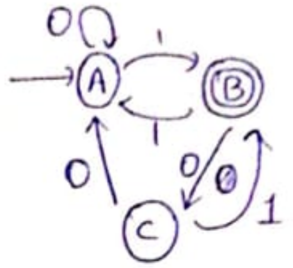
$S \rightarrow aab \ (B \rightarrow b)$

the grammar is ambiguous since there exists more than one

IMD for the same string.

1.6

	0	1
→ A	A	B
(B)	C	A
C	A	B.

DFA: 

Context Free Grammar: $A \rightarrow 0A \mid 1B$

$B \rightarrow 0C \mid 1A \mid \varepsilon$

$C \rightarrow 0A \mid 1B$.

~~DPDA = ({q}, {0,1}, J, q, z_0, δ) using empty stack.~~

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

$= (\{q\}, \{0,1\}, \{z_0\}, \delta, \{q_0\}, z_0, F)$ using empty stack.

$\delta(q, \varepsilon, z_0) = (q, Az_0), \{(q, 0Az_0), (q, 1Bz_0)\}$

$\delta(q, \varepsilon, A) = \{(q, 0Az_0), (q, 1Bz_0)\}$

$\delta(q, \varepsilon, B) = \{(q, 0Cz_0), (q, 1Az_0), (q, \varepsilon)\}$

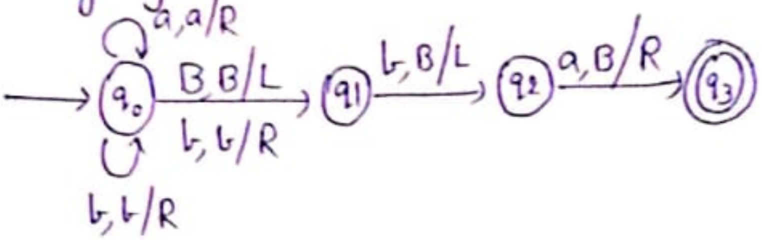
$\delta(q, \varepsilon, C) = \{(q, 0Az_0), (q, 1Bz_0)\}$

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

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

$\delta(q, \varepsilon, z_0) = (q, \varepsilon) \Rightarrow$ final state.

1.7 Given language. $L = \{(a+b)^n ab \mid n \geq 0\}$



1.8 Transition function δ for Turing machine with stay option is:

$\delta : Q \times \Gamma$ to $Q \times \Gamma \times \{$left, right, stay$\}$.

where Γ = set of tape symbols.

Q = set of finite states.

1.9 CFLs are closed under concatenation(uni; union and kleen closure.

1.10 Given grammar: $S \rightarrow aA|a|B|c$, $A \rightarrow aB|\varepsilon$, $B \rightarrow Aa$, $C \rightarrow cCD$, $D \rightarrow dd$.

~~1. Elimination of common prefix~~

~~- No common prefix.~~

~~2. Elimination of left recursion.~~

~~- No left recursion.~~

step1 Elimination of useless symbols and production.

Old Variable	New variable	Productions.
ϕ	S, A, D	$S \rightarrow a$ $A \rightarrow \varepsilon$ $D \rightarrow dd$
S, A, D	S, A, B, D	$S \rightarrow aA$ $B \rightarrow Aa$
S, A, B, D	S, A, B, D	$S \rightarrow B$ $A \rightarrow aB$.

S, A, B, D	S, A, B, D	—

step 2:

P'	T'	V'
—	—	S
$S \rightarrow a \mid aA \mid B$	a	S, A, B
$A \rightarrow aB \mid \varepsilon$	a, ε	S, A, B
$B \rightarrow Aa$	a	S, A, B

Final productions: $S \rightarrow a \mid aA \mid B$

$$A \rightarrow aB \mid \varepsilon$$

$$B \rightarrow Aa.$$

1.11 Recursively enumerable language: A language is recursively enumerable if some turing machine accepts it.

let L be a recursively enumerable language and M the turing machine that accepts it

for string w,

If $w \in L$, then M halts in a final state.

If $w \notin L$, then M halts in a non-final state and loops forever.

Recursive language:- A recursive language L is a formal language for which there exists a turing machine that will halt and accept an input string in L, and halt and reject otherwise.

1.12 <u>polynomial time reduction</u>:- when a problem A is polynomial time reducible to a problem B, it means that given an instance of A, there is an algorithm for transforming instances of A into instances of B. This is often done to derive hardness results: if there was a fast algorithm for some problem, there would also be a fast algorithm for some other problem.

1.13 $S \rightarrow aSbS \mid bSaS \mid \varepsilon$

Equal number of a's and b's.

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

1.14 A PDA is deterministic if there is never a choice of move in any situation.

PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ to be deterministic if the following conditions hold

1) $\delta(q, a, x)$ has atmost one member for any q in Q, a in Σ or $a = \varepsilon$ and x in Γ.
2) If $\delta(q, a, x)$ is non-empty for some a in Σ then $\delta(q, \varepsilon, x)$ must be empty.

PART-B

2.

a. <u>Pumping lemma for regular languages.</u>

<u>Statement</u>: If A is a regular language, then there exist a constant 'n' such that for every string $w \in A$ such that $|w| \geq n$ may be divided into three parts $w = xyz$ such that the following conditions must be true.

1) $y \neq \varepsilon$

2) $|xy| \leq n$

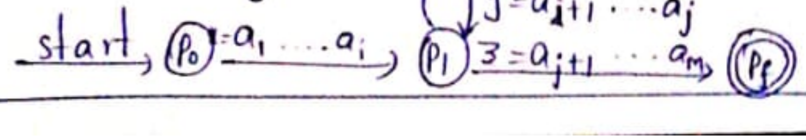
3) $xy^iz \in A$ for every $i \geq 0$.

Proof:- Suppose L is regular. Then $L = L(A)$ for some DFA A. Suppose A has n states. Now, consider only string w of length n ~~inpu~~ or more, say $w = a_1, a_2, \ldots, a_m$, where $m \geq n$ and each a_i is an input symbol. For $i = 0, 1, \ldots, n$, define state p_i to be $\delta(q_0, a_1 a_2, \ldots a_i)$ where δ is the transition function of A, and q_0 is the start state of A. This is, p_i is the state A is in after reading the first i symbols of w. Note that $p_0 = q_0$.

By the pigeonhole principle, it is not possible for the n+1 different p_i's for $i = 0, 1, \ldots, n$ to be distinct, since there are only n different states. Thus, we can find two different integers i and j, with $0 \leq i < j \leq n$, such that $p_i = p_j$. Now, we can break $w = xyz$ as follows

1. $x = a_1 a_2 \ldots a_n$
2. $y = a_{i+1} a_{i+2} \ldots a_m$
3. $z = a_{j+1} a_{j+2} \ldots a_m$.

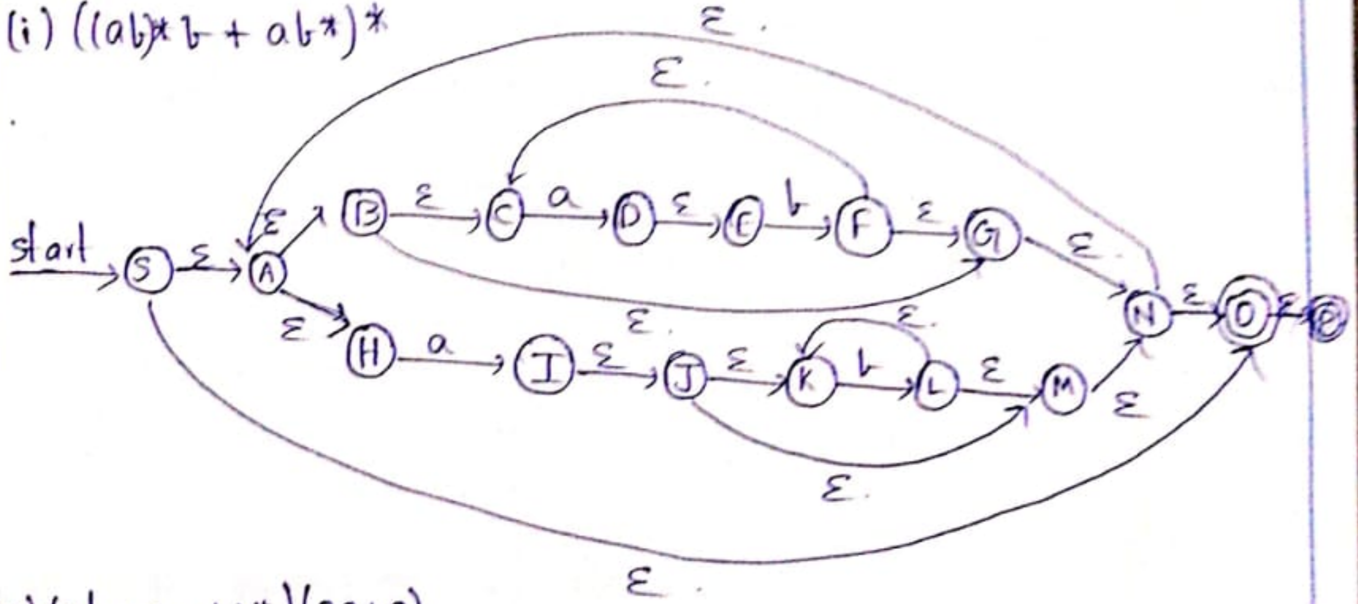
That w, x takes w to p_i once; y takes us from p_i back to p_i (since p_i is also p_j), and z is the balance of w. The relationships among the strings and states are suggested by the state diagram. Note that x may be empty, in the case that $i = 0$. ~~No~~ Also, z may be empty if $j = n = m$. However, y cannot be empty, since $i < j$



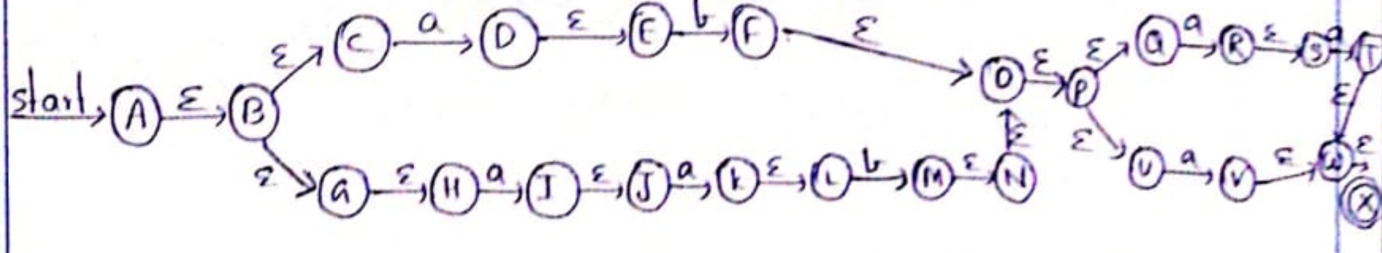
Now, consider what happens if the automaton A receives the input xy^kz for only $k \geq 0$. If $k=0$, then the automaton goes from the start state q_0 (which is also p_0) to p_i on input x. Since p_i is also p_j, it must be that A goes from p_i to the accepting state shown in the state diagram on input z. The A accepts xz.
If $k>0$, then A goes from q_0 to p_i on input x, circles from p_i to p_i k times on input y^k, and then goes to the accepting state on input z. Thus for any $k \geq 0$, xy^kz is also accepted by A; that is xy^kz is in L.

∴ Hence proved.

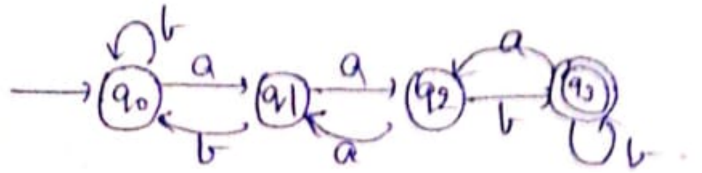
b. (i) $((ab)^*b + ab^*)^*$



(ii) $(ab+(aab)^*)(aa+a)$.

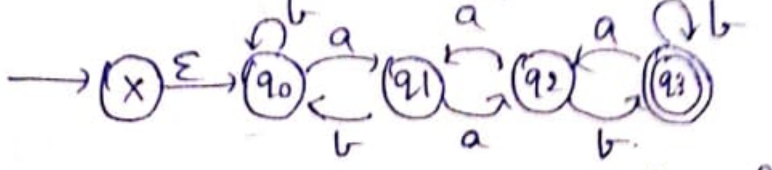


c. Given finite automata:

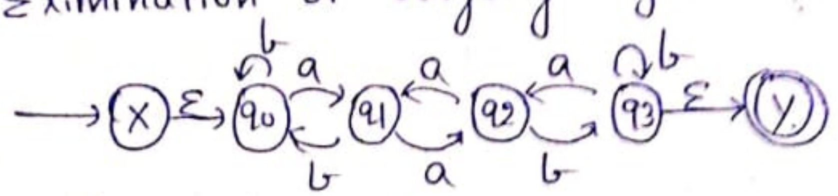


<u>steps</u>:

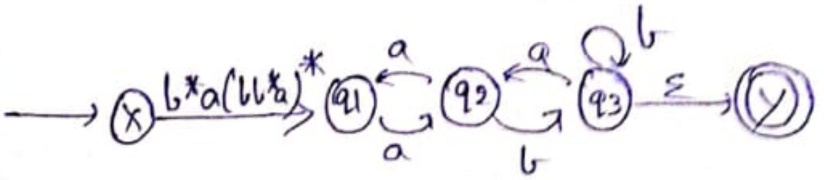
1. Elimination of incoming edge to the initial state



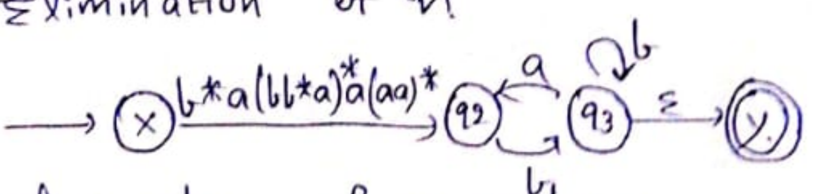
2. Elimination of outgoing edge from final state



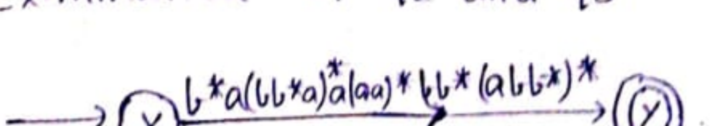
3. Elimination of q_0.



4. Elimination of q1



5. Elimination of q2 and q3.



Regular expression: $b^*a(bb^*a)^*a(aa)^*bb^*(abb^*)^*$.

3 a Given Grammar,

$S \to AaA|cA|BAB$
$A \to aaBa|CDA|aa|DC$
$B \to bB|bAB|bb|aS$
$C \to Ca|bC|D$
$D \to bD|\varepsilon$

step 1: Eliminating start symbol from RHS

$S_0 \to S$

step 2: Eliminating nullable variables.

Nullable variable = $\{D, C, A, S\}$.

$S \to aA|Aa|BaB|c/A$
$A \to aaBa|CD|CA|DA|aa|C/D$
$B \to bB|bAB|bb|a|aS$
$C \to Ca|a|b|bc|D$
$D \to bD|b|D$

step 3: Eliminating unit productions

unit productions: $S \to C$, $S \to A$
$A \to C$, $A \to D$
$C \to D$
$D \to D$.

Hence, $D \to bD/b$
$C \to Ca|a|b|bc|bD|b$
$B \to bB|bAB|bb|a|aS$.
$A \to aaBa|CD|CA|DA|aa|Ca|a|b|bc|bD|b$.
$S \to aA|Aa|BaB|aaBa|CD|CA|DA|aa|Ca|a|bc|bD|b$

step 4:

CNF: $S \to ZA|XB|WX|CD|CA|DA|W|CZ|a|VC|VD|b$
$Z \to a$
$X \to Ba|BX'$
$X' \to Z$
$W \to aa|ZZ$
$V \to b$
$A \to WX|CD|CA|DA|W|CZ|a|b|VC|VD|b$
$B \to VB|VV|a|ZS$
$C \to CZ|a|b|VC|VD|b$
$D \to VD|b$

b. Given grammar:

$$S \rightarrow bSSaaS \mid bSSaSb \mid bSb \mid a$$

<u>Left factoring:</u>

$$S \rightarrow bSS' \mid a$$
$$S' \rightarrow SaaS \mid SaSb \mid b$$
$$S' \rightarrow SaX$$
$$X \rightarrow aS \mid Sb$$

<u>CNF after left factoring:-</u>

$$S \rightarrow bSS' \mid a$$
$$S' \rightarrow SaX$$
$$X \rightarrow aS \mid Sb$$

c. (i) $L = \{uvwv^R : u, v, w \in \{a,b\}^+, |u| = |w| = 2\}$

CFG: $S \rightarrow AB$

$B \rightarrow aBa \mid bBb \mid aAa \mid bAb$

$A \rightarrow aa \mid ab \mid ba \mid bb$

(ii) $L = \{a^n b^m : n \leq m+3\}$

CFG: $S \rightarrow AAAB$

$A \rightarrow a \mid \varepsilon$

$B \rightarrow aBb \mid Bb \mid \varepsilon$

4. a. <u>Left Recursion:</u> A production of grammar is said to have left recursion if the leftmost production variable of its RHS is same as variable of its LHS. A grammar containing a production having left recursion is called as "Left Recursion."

Given grammar:

$E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow (E) \mid id$.

$A \rightarrow A\alpha \mid \beta$ $\left\{ \begin{array}{l} A \rightarrow \beta A' \\ A' \rightarrow \alpha A' \mid \varepsilon \end{array} \right.$

$E \rightarrow TE'$

$E' \rightarrow +TE' \mid \varepsilon$

Hence, $T \rightarrow FT'$

$T' \rightarrow FT' \mid \varepsilon$

$F \rightarrow (\varepsilon) \mid id$

b. Given grammar: $S \rightarrow aAS \mid a \mid SS$,

$A \rightarrow SbA \mid ba$.

Given string : aabbaa.

(i) Left most derivation.

$S \rightarrow aAS$

$S \rightarrow aSbAS$ $(A \rightarrow SbA)$.

$S \rightarrow aabAS$ $(S \rightarrow a)$

$S \rightarrow aabbas$ $(A \rightarrow ba)$

$S \rightarrow aabbaa$ $(S \rightarrow a)$.

(ii) Right most derivation.

$S \rightarrow aAS$

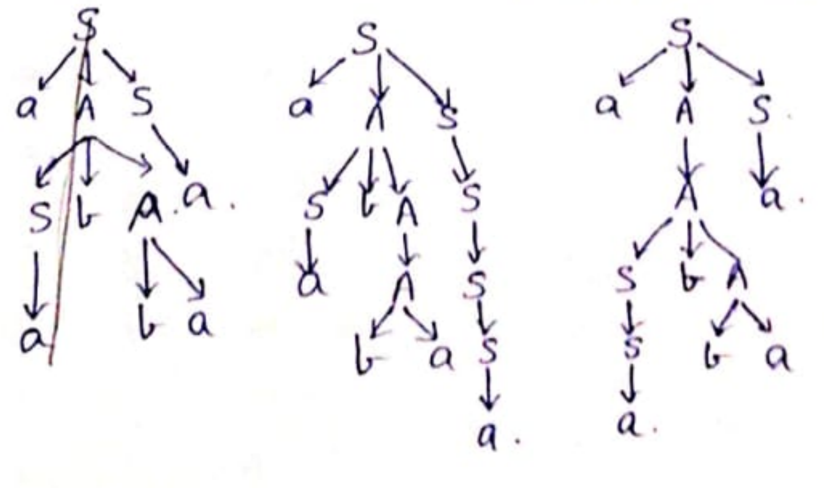
$S \rightarrow aAa$ $(S \rightarrow a)$

$S \rightarrow aSbAa$ $(A \rightarrow Sba)$

$S \rightarrow aabAa$ $(S \rightarrow a)$

$S \rightarrow aabbaa$ $(A \rightarrow ba)$

(iii) Derivation tree for LMD: Derivation tree for RMD:



c. Context free Grammar (CFG) is of great practical importance. It is used for following purposes-

- For defining programming languages.
- For parsing the program by constructing syntax tree
- For translation of programming languages.
- For describing arithmetic expressions.
- For construction of compilers.
- ~~For construction~~ Simplicity of proofs.

There are plenty of proofs around Context-free Grammar including reducability and equivalence of automata. Those are the simplest the most restricted set of grammars you have to deal with. Therefore, normal forms can be helpful here. As a concrete example, Greiback normal form is used to show (constructivity) that there is an ε-transition-free PDA for every CFL (that doesnot contain ε)

- They are used in an essential part of the Extensible Markup language (XML) called the Document Type Definition.

5a. Given language,

$$L = \{a^{2n}b^n \mid n \geq 0\}$$

Given string: aaaabb, ~~PDA P = {~~

Moves made by PDA:

$(q_0, aaaabb, z_0)$

$\vdash (q_0, aaabb, az_0)$

$\vdash (q_0, aabb, aaz_0)$

$\vdash (q_0, abb, aaaz_0)$

$\vdash (q_0, bb, aaaaz_0)$

$\vdash (q_1, b, aaaz_0)$

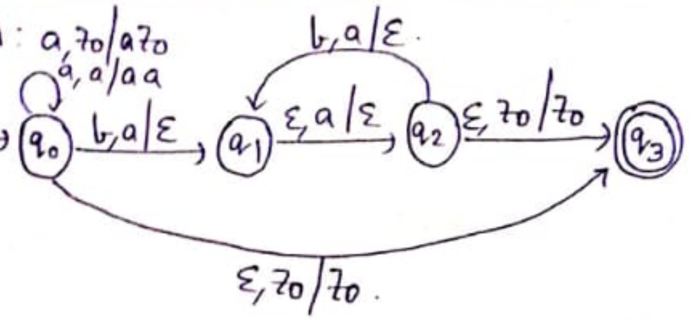
$\vdash (q_2, b, aaz_0)$

$\vdash (q_1, \varepsilon, az_0)$

$\vdash (q_2, \varepsilon, z_0)$

$\vdash (q_2, \varepsilon, z_0)$ (final state)

PDA :



b. <u>statement</u>: CFls are not closed under intersection.

<u>proof</u>: If L_1 and if L_2 are two context free languages, their intersection $L_1 \cap L_2$ need not be context free

for example,

$L_1 = \{a^n b^n c^m \mid n >= 0 \text{ and } m \geq 0\}$

and $L_2 = \{a^m b^n c^n \mid n \geq 0 \text{ and } m \geq 0\}$.

$L_3 = L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 0\}$ need not be context free.

L_1 says number of a's should be equal to number of b's and L_2 says number of b's should be equal to number of c's. Their intersection says both conditions need to be true, but pushdown automata can compare only two. So it cannot be accepted by pushdown automata, hence not context free.

So, CFL are not closed under Intersection.

c. Given PDA

$P = \{\{q_0, q_1, q_f\}, \{a, b\}, \{Z_0, A\}, \delta, q_0, Z_0, q_f\}$.

$\delta(q_0, a, Z_0) = (q_0, AZ_0)$

$\delta(q_0, a, A) = (q_0, AA)$

$\delta(q_0, b, A) = (q_1, \varepsilon)$

$\delta(q_1, b, A) = (q_1, \varepsilon)$

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

step 1 :- Generation of S production

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

step 2 :- Take the production move,

$$\delta(q_0, a, Z_0) = (q_0, AZ_0)$$

It is a non-erasing move.

The productions are:

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

step 3 :- Take the production move,

$$\delta(q_0, a, A) = (q_0, AA)$$

It is a non-erasing move.

The ~~move is~~ productions are:

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

$[q_0, A, q_f] \longrightarrow a[q_0, A, q_f][q_f, A, q_f]$

Step 4 :- Take the production move,

$\delta(q_0, b, A) = (q_0, AA)$

$\delta(q_0, b, A) = (q_1, \varepsilon)$

It is an erasing move.

production: $[q_0, A, q_1] \longrightarrow b$.

Step 5 :- Take the move,

$\delta(q_q, b, A) = (q_1, \varepsilon)$

It is also an erasing move.

production: $[q_1, A, q_1] \longrightarrow b$.

Step 6 :- Take the move,

$\delta(q_1, \varepsilon, z_0) = (q_f, z_0)$.

Acceptance by empty stack.

$[q_1, z_0, q_f] \longrightarrow \varepsilon [q_1, z_0, q_f]$.

6.Q. (a) Languages accepted by PDA

(1) Acceptance by final state.

Let, $P = (Q, \varepsilon, \Gamma, \delta, q_0, z_0, F)$ be a PDA.

Language accepted by PDA, Then $L(P) = \{w \mid (q_0, w, z_0) \vdash_P (q, \varepsilon, \alpha)\}$ For some state $q \in F$, that is starting in the initial ID w waiting on the input. PDA P consumes w from the input and enters an accepting state. The contents of the stack at that time is irrelevant.

(2) Acceptance by empty stack.

$P = (Q, \varepsilon, \Gamma, \delta, q_0, z_0, F)$ be a PDA

$N(P) = \{w \mid (q_0, w, z_0) \vdash^* (q, \varepsilon, \varepsilon)\}$ is the language accepted by PDA.

Here $N(P)$ is the set of inputs w that P can consume and at the same time empty stack. Here, the set of accepting states is irrelevant

The stack shouldn't even contain Z_0.

b. <u>Algorithm to convert CFG to PDA</u>.

Let $G = (V, T, Q, S)$ be a Context Free Grammar, Construct the PDA P that accepts $L(G)$ by empty stack as follows

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

where δ is defined by

1) For each variable A,

$$\delta(q, \varepsilon, A) = \{(q, \beta) \mid A \rightarrow \beta \text{ is a production of } P\}.$$

$$G = (V, T, Q, S) \quad P = (\underbrace{\{q\}}_{Q}, T, \underbrace{V \cup T}_{\Gamma}, \delta, q, \underbrace{S}_{Z_0})$$

2) For each terminal a,

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

3) when there is no input symbol,

For start variable, $\delta(q, \varepsilon, Z_0) = (q, SZ_0)$.

4) Final transition

$$\delta(q, \varepsilon, Z_0) = (q, \varepsilon).$$

Given CFG:

$S \rightarrow aABB \mid aAA$

$A \rightarrow aBB \mid a$

$B \rightarrow bBB \mid A$

$C \rightarrow a$

$$PDA, P = (\{q\}, \{a, b\}, \{a, b, A, B, Z_0\}, q, Z_0, \delta, \phi\}$$

$\delta(q, \varepsilon, z_0) = (q, Sz_0)$

$= (q, aABBz_0), (q, aAA)$

$\delta(q, \varepsilon, A) = (q, aBBz_0), (q, az_0)$

$\delta(q, \varepsilon, B) = (q, bBBz_0), (q, Az_0)$

$= (q, bBBz_0), (q, aBBz_0), (q, az_0)$

$\delta(q, a, a) = (q, \varepsilon)$

$\delta(q, b, b) = (q, \varepsilon)$

c. Given language:

$L = \{a^n b^n c^n \mid n \geq 0\}$.

Let us assume that L is Context free, then by Pumping Lemma, the following rules hold good for an integer n such that for all $x \in L$ with $|x| \geq n$, there exists $u, v, w, x, y \in \Sigma^*$ such that $x = uvwxy$, and

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

For L, if (i) and (ii) hold, then $x = a^n b^n c^n$, $uvwxy$ with $|vwx| \leq n$ and $|vx| \geq 1$

(i) tells that vwx doesnot contain both a and c. Thus either vwx has no a's or vwx has no c's. Remaining two more cases,

Suppose vwx has no a's By (ii) vx contains a 'b' or a 'c'. Thus uwy has n a's and uwy either has less than 'n' b's or has less than n c's.

But (iii) tells that $uwy = uv^0 wx^0 y \in L$

So, uwy has an equal number of a's, b's and c's gives a contradiction. The case where vwx has no c's is similar and also gives a contradiction

Thus, L is not Context free language.

7.a. $f(x) = \begin{cases} x/2, & x \text{ is even} \\ \frac{x+1}{2}, & x \text{ is odd} \end{cases}$

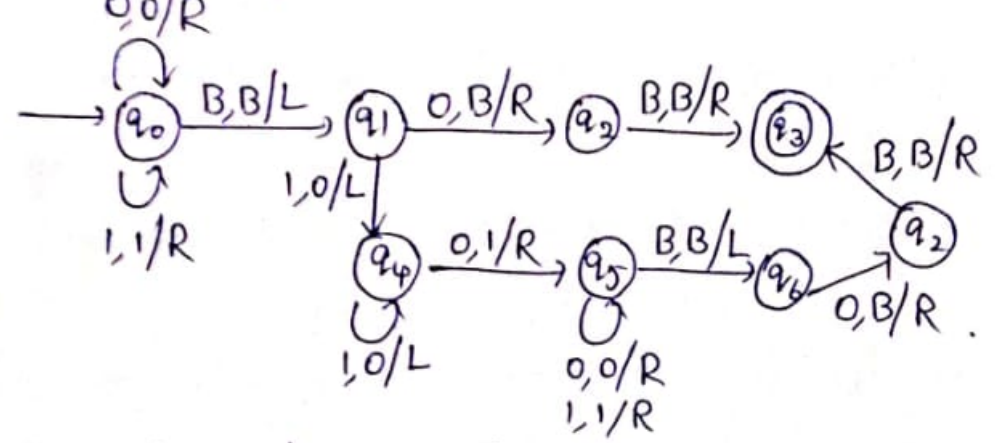
Turing machine, $T = (Q, \Sigma, J, \delta, q_0, B, F\}$ is given by

$Q = \{q_1, q_2, q_3, q_4, q_5, q_6, q_7\}$

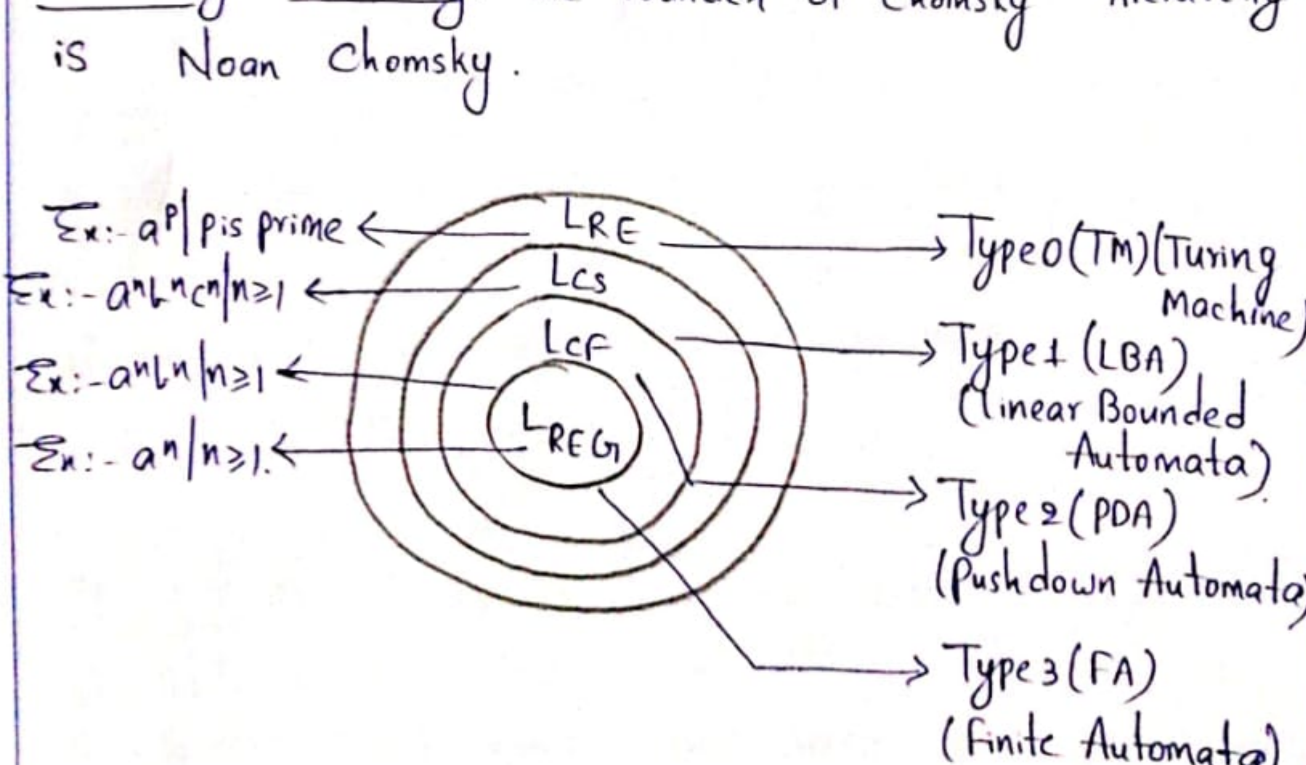
$\Sigma = \{0, 1\}$

$Z = \{B, 0, 1\}$

$F = \{q_3\}$



b. Chomsky Hierarchy:- The founder of chomsky Hierarchy is Noan Chomsky.



$L_{RE} > L_{REC} > L_{CS} > L_{CF} > L_{DCF} > L_{REG}$

Type 0 Grammar: A Grammar $G=(V,T,P,S)$ is said to be Type 0 Grammar or unrestricted or phase structured grammar if all productions are of the form
$\alpha \rightarrow \beta$ where $\alpha \in (V \cup T)^+$ and $\beta \in (V \cup T)^*$

Ex:- $S \rightarrow aAb \mid \varepsilon$
$aA \rightarrow bAA$
$bA \rightarrow a$

Type 1 Grammar: A Grammar $G=(V,T,P,S)$ is said to be type 1 or Context Sensitive Grammar or Epsilon free Grammar if all the productions are of the form $\alpha \rightarrow \beta$. Here there is a restriction on the length of β i.e the length of β must be atleast as much as the length of α ($|\beta| \geq |\alpha|$) and $\beta \in (V \cup T)^+$

Ex:- $S \rightarrow aAb$
$aA \rightarrow bAA$
$bA \rightarrow aa$.

Type 2 Grammar: A grammar $G=(V,T,P,S)$ is said to be type 2 grammar if all the productions are of the form $A \rightarrow \alpha$ where $\alpha \in (V \cup T)^*$

Ex:- $S \rightarrow aB \mid bA \mid \varepsilon$
$A \rightarrow aA \mid b$
$B \rightarrow bB \mid a \mid \epsilon$

Type 3 Grammar/Regular Grammar: The grammar $G=(V,T,P,S)$ is said to be of type 3 or regular if grammar is either left linear or right linear
→ A grammar is said to be left linear if all the

production are of the form $A \to Bw$ or $A \to w$

→ A Grammar is said to be right linear if all the productions are of the form. $A \to wB$ or $A \to w$.

Ex:- $S \to aA$ ——→ Right linear.

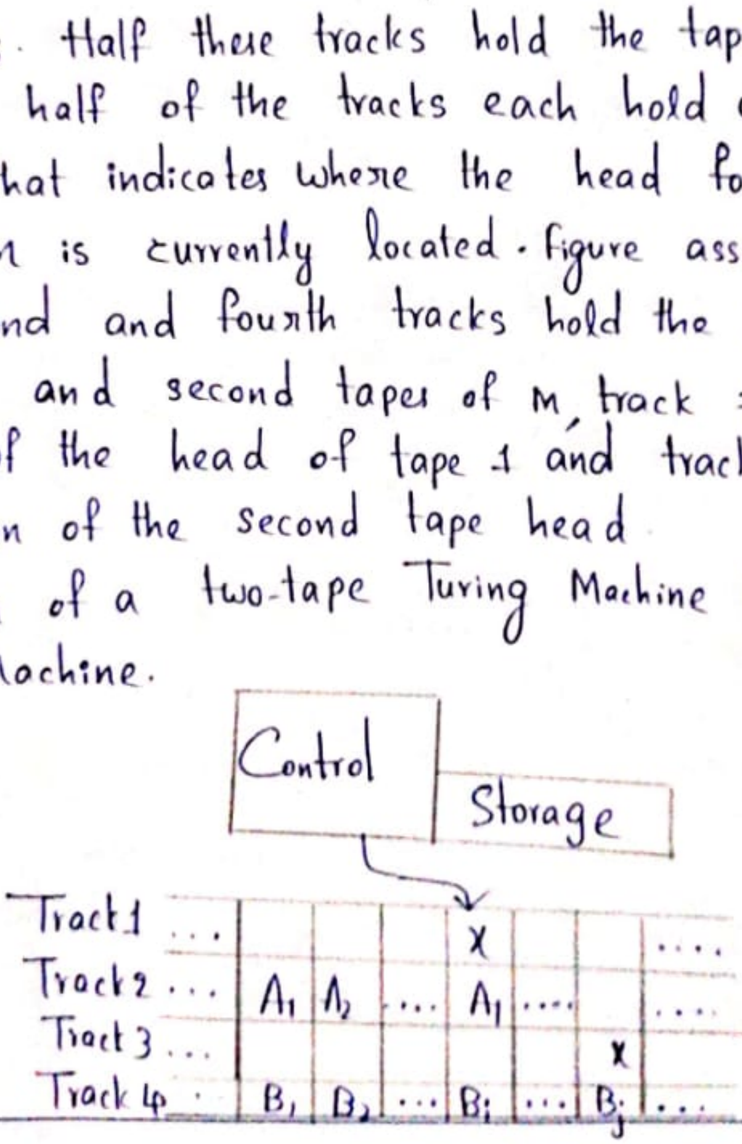
$A \to aB | b$ ——→ Right linear

$B \to Ab | a$ ——→ left linear.

c. Every language accepted by a multi-tape Turing Machine is recursively enumerable.

PROOF:- Suppose language L is accepted by a k-tape Turing Machine M we simulate M with a one-tape Turing Machine N whose tape we think of as having 2k tracks. Half these tracks hold the tapes of M, and the other half of the tracks each hold only a single marker that indicates where the head for the corresponding tape of M is currently located. figure assumes k-2. The second and fourth tracks hold the contents of the first and second tapes of M, track 1 holds the position of the head of tape 1 and track 3 holds the position of the second tape head.

Simulation of a two-tape Turing Machine by a one-tape Turing Machine.



To simulate a move of M, N's head must visit the k head markers. So that N not get lost, it must remember how many head markers are to its left at all times; that count is stored as a component of N's finite control. After visiting each head marker and storing the scanned symbol is a component of its finite control, N knows what tape symbols are being scanned by each of M's heads. N also knows the state of M, which it stores in N's own finite symbol. Thus, N knows what move M will make.

N now revisits each of the head markers ~~of the~~ on its tape, changes the symbol in the track representing the corresponding tapes of M and moves the head markers left or right, of necessary. Finally, N changes the state of M as recorded in its own finite control. At this point, N has simulated one move of M.

We select as N's accepting state all those states that record M's state as one of the accepting states of M. Thus, whenever the simulated M accepts, N also accepts and M does not accept otherwise.

8.a 3-SAT is polynomial time reducible to CLIQUE.

PROOF:

Let $\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \ldots \wedge (a_k \vee b_k \vee c_k)$

We reduce this boolean formula into an undirected graph G. This is done by grouping the nodes in G into k groups of three nodes, each called a triple. $t_1, \ldots, t_k$

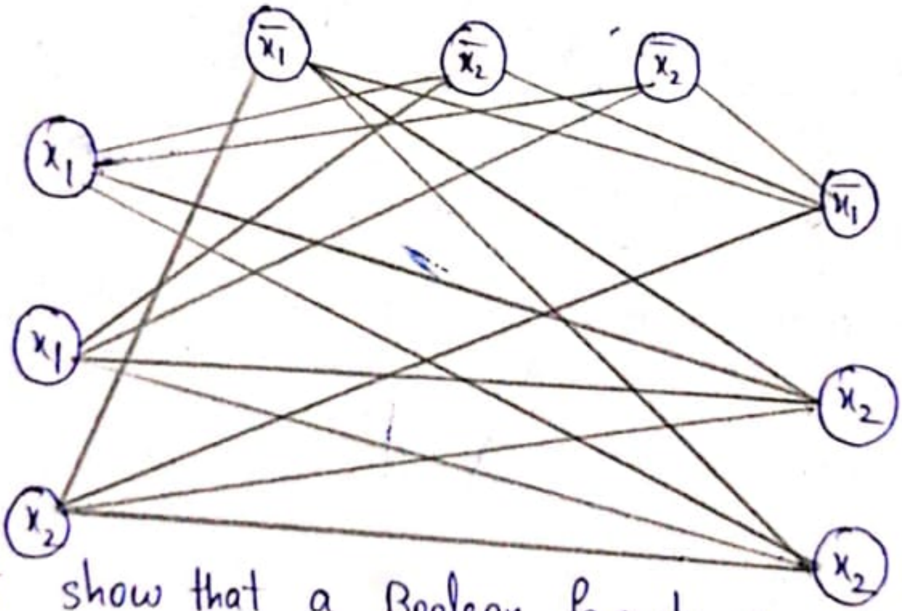
Each of these triplets corresponds to one of the clauses in the formula. Each individual node within a triple corresponds to a literal is the corresponding clause. In the resulting graph G, all nodes are connected by an edg except:

- between nodes in the same triple.
- between complimentary nodes.

Boolean formula,

$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2 \vee x_2).$$

Converted to graph,



We must show that a Boolean formula is satisfiable iff G has a k-clique. Suppose the Boolean formula is satisfiable. In the satisfying assignment atleast one literal in each clause is the true, so we select that node corresponding node in each triple of the graph G.

- If more than one literal is true, pick one arbitrarily.

All nodes selected from a k-clique since we choose one

From each of the k-triples. Each pair of selected nodes is:
• Joined by an edge, becouse no pair fits one of the exceptions previously mentioned.
• Not from the same triple, becouse only one ~~was~~ node was selected from each triple.
• Not conflicting labels, becouse the associated literals were both true in the assignment
Therefore, G contains a k-clique.

(b) Primitive Recursive Functions:

They define a set of functions that contain only computable function, using only basic operations (ex: the operation "add") and basic ways of putting functions together (ex: composition). The model is as simple as possible and guarantees that all functions generated as computable.

A function $f(x_1, ..., x_n)$ is primitive recursive if either:

1. f is the function that is always 0, i.e $f(x_1, .., x_n) = 0$; This is denoted by Z when the number of arguments is understood. This rule is for deriving a primitive recursive function is called the zero rule.

2. f is the successor function, i.e $f(x_1, ..., x_n) = x_i + 1$; this rule for deriving a primitive recursive function is called the successor rule

3. f is the projection function, i.e $f(x_1, ..., x_n) = x_i$; This is denoted by Π, when the number of arguments is understood. This rule for deriving a primitive recursive function is called the Projection Rule.

4. f is defined by the composition of primitive functions,

i.e if $g_1(x_1, \ldots, x_n), g_2(x_1, \ldots, x_n) \ldots, g_k(x_1, \ldots, x_n)$ are primitive recursive and $h(x_1, \ldots, x_k)$ is primitive recursive, then

$$f(x_1, \ldots, x_n) = h(g_1(x_1, \ldots, x_n), \ldots, g_k(x_1, \ldots, x_n))$$

is primitive recursive. This rule for deriving a primitive recursive function is called the composition rule.

5. f is defined by recursion of two primitive recursive functions, i.e, if $g(x_1, \ldots, x_{n-1})$ and $h(x_1, \ldots, x_{n+1})$ are primitive recursive then the following function is also primitive recursive.

$$f(x_1, \ldots, x_{n-1}, 0) = g(x_1, \ldots, x_{n-1})$$

$$f(x_1, \ldots, x_{m-1}, m+1) = h(x_1, \ldots, x_{n-1}, m, f(x_1, \ldots, x_{n-1}, m))$$

This rule for deriving a primitive recursive function is called the Recursion rule. It is very powerful rule and is why these functions are called 'primitive ~~funct~~ recursive functions.'