Regular expressions

Its symbolic representation of regular language

It is a set of character that represents the search pattern for any string in an algebraic form

Regular expression contains 3 main operators

1. union (+)
2. concatenation (.)
3. kleene closure / star $(a+b)^*$, $(a+b)^+$

a) ϕ is a regular expression which represent empty set

$L = \{\}$ — empty set

ε — $L = \{\epsilon\}$

ε - zero length string / empty string

a — $L = \{a\}$

All these 3 regular expression are called primitive regular expression

b) $r_1 + r_2$. $r_1 . r_2$. r_1^*

$L(r_1) \cup L(r_2)$ $L(r_1) . L(r_2)$ $(L(r_1))^*$ or $L(r_1^*)$

$L(R_1 \cup R_2)$ $L(r_1 . r_2)$

Using a and b any no of times is also a regular expression

Write the regular expression for the language defined on alphabet $\{a, b\}$

$L = \{$ set of all string whose length is exactly 2

ans

$a^* = \{ \in, a, aa, aaa - - - \}$

$(a+b)^* = \{ \in, a, b, aa, bb, ab, ba \}$

a or b

1st pos (a+b)	2nd pos a+b
a	a
a	b
b	a
b	b

$(a+b)\ (a+b)$

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

$\in$ not used in regular expression

- L_2: length is atleast 2

ans

$(a+b)\ (a+b)\ (a+b)^*$ or $(a+b)\ (a+b)^+$

atleast 2 length

- L_3: length is atmost 2

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

consider each as a 2 length string $\in \cdot \in \in$, $a \cdot a \in$, $b \in$

$(\in + a + b)\ (\in + a + b)$

- L_4: all even length string

ans

$(a+b)\ (a+b)$

2 length string

First ensure 2 position

$((a+b)\ (a+b))^*$

- L_5 : odd length string

~~$(a+b)^*$~~ ~~$(a+b)^*$~~

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

- L_6 : length of string divisible by 3

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

- L_7 : Length of string $\equiv 2 \mod 3$ length % 3 = 2

2, 5, 8, 11

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

- L : no of a's are exactly 2

* $(a)(a)(b)^*$ aab, aabb - - -
* $b^* a a$ bba, bbaa - - -
* $(b)^*(a)(b)^*(a)(b)^*$ babab, aba

- L : no of a's atleast 2

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

- L : no of a's atmost 2 a

$b^* (a+\varepsilon) b^* (a+\varepsilon) b^*$

- Number of a's all even — it will not take b, bb, bbb...

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

- Number of a's all odd

$(b^* a b^* a b^*)^* a$

- All string that ends with a

$(a+b)^* a$

- All string that start with a

$a\,(a+b)^*$

- All string that contains a

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

- All string starting and ending with different symbol

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

- All string starting and ending with same symbol

$a(a+b)^* a + b(a+b)^* b + a + b + \in$

↓

single a only

Q. L: set of all strings with no 2 a's come together

ans $(b^* a b^*)^*$ ~~bbb~~ ✗ $(b^* a b^*)^* + b^*$ } this will contain many states

$(ab)^*$: abab - - - - - 1/2 mark

{ ababab, abbbabb, bbbabb ✗ }

$(ab+b)^*$ - here string ending with a

$(ab+b)^*a$ - here all ending with a

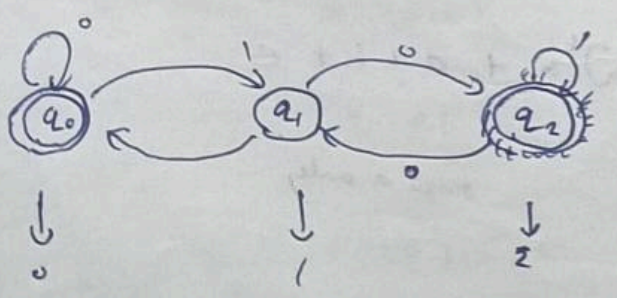
ans:- $(ab+b)^*(a+\varepsilon)$

- L_{1a} :- all string neither 2 a's nor 2 b's come together

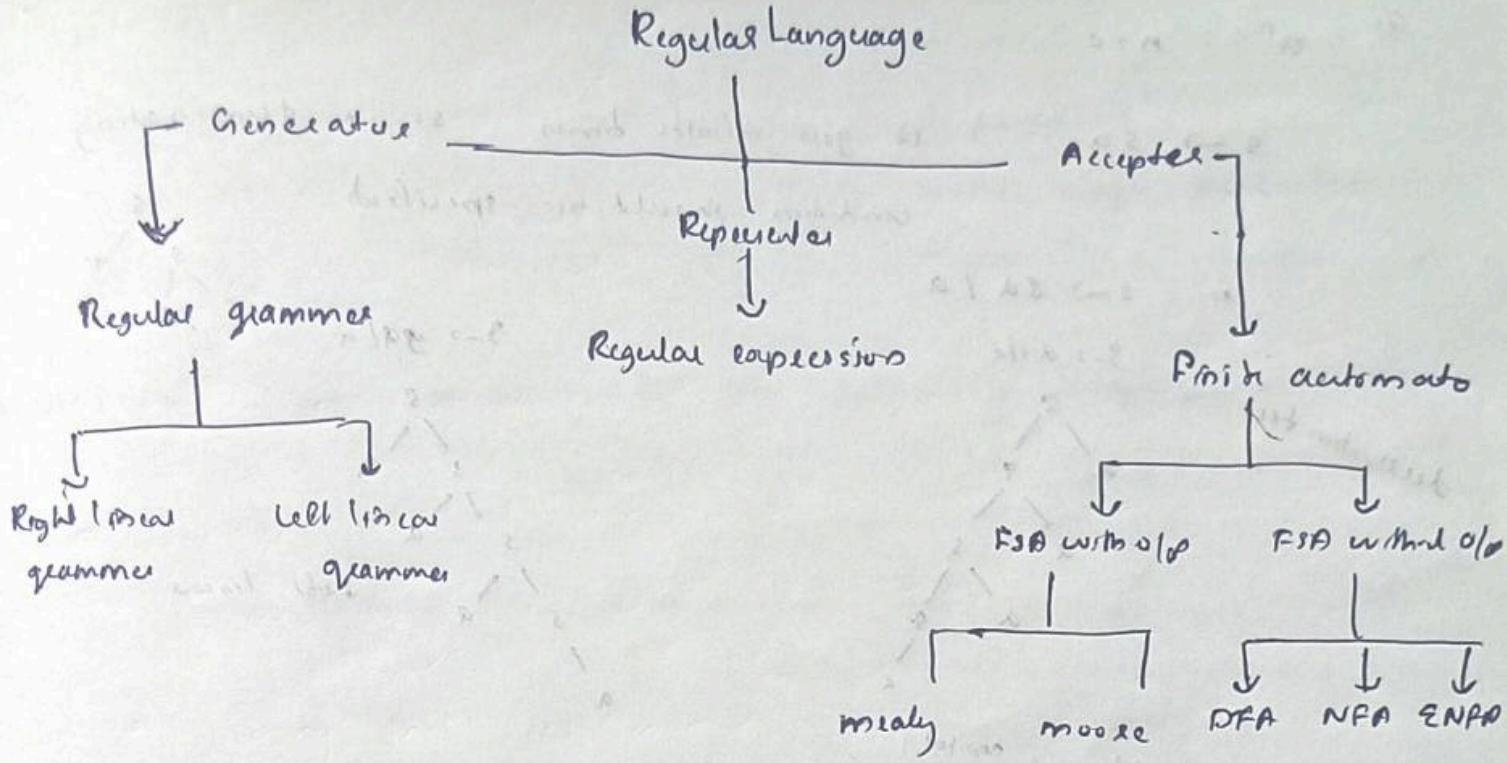
~~a+b+ε~~ $(ab)^* + (ba)^* + b + a + \varepsilon$

or

$(b+\varepsilon)(ab)^*(a+\varepsilon)$



} gives o/p when given all at start



- L = set of all strings over $\Sigma = \{a, b\}$, whose length = exactly 2

  $L = \{aa, ab, ba, bb\}$

  regular expression :- $(a+b)(a+b)$

  regular grammer :-

  $S \rightarrow aa / ab / ba / bb$

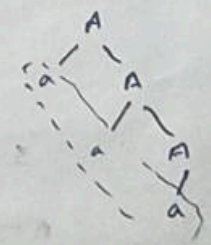
  it is not practically possible

  $S \rightarrow AA$

  $A \rightarrow a / b$

  $A \rightarrow aA / a$

  $A \rightarrow \wedge$



  If it is length 3

  $S \rightarrow AAA$

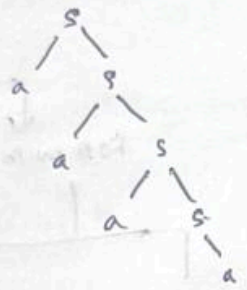
  $A \rightarrow a / b$

② a^n $n>0$

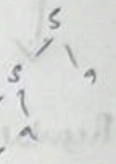
$S \to Sa$ it goes infinite times so a dominating condition should be specified

ie $S \to Sa \mid a$

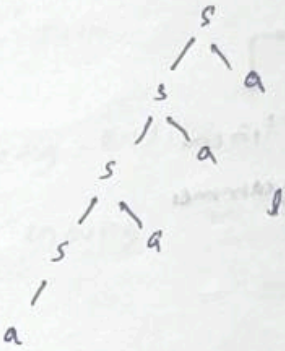
$S \to asla$



derivation tree



$S \to gal a$



left linear

S → variable (capital) (that are expandable)

if variable is at left then it is left linear $S \to Sa \mid a$

if variable is at right then it is right linear

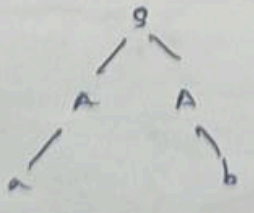
Uppercase → variable (expandable)

lowercase → non terminals

The tree is called derivation tree / Sparse tree

$S \to AA$

$A \to a \mid b$



it is not right linear or left linear. so it is not a regular grammer

Ida regular grammer is

$S \rightarrow Aa \mid Ab$ Aa is single production

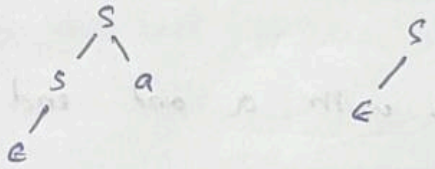
$A \rightarrow a \mid b$ Ab is another single production

• Write the regular grammer for the following language

A) $a^n \quad n \geq 0$

~~$S \rightarrow Sa \mid a \mid \epsilon$~~

$S \rightarrow Sa \mid \epsilon$



2) $(a+b)^*$

~~$S \rightarrow S$~~ $S \rightarrow Sa \mid Sb \mid \epsilon$

3) atleast 2 symbols

$(a+b) \quad (a+b) \quad (a+b)^k$

$A \rightarrow a \mid b$

$A \rightarrow ab$

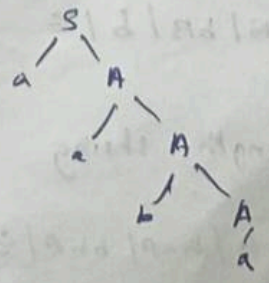
$B \rightarrow$ ~~$a+b+\epsilon$~~ $Ba \mid Bb \mid \epsilon$

$S \rightarrow AAB$
$A \rightarrow a \mid b$
$B \rightarrow Ba \mid Bb \mid \epsilon$
} not regular

or -

$S \rightarrow aA \mid bA$

$A \rightarrow a \mid b \mid aA \mid bA$



$S \rightarrow aS$ continuous a's

$S \rightarrow aS/bS$ continuous a's or b's

4) almost 2 symbols

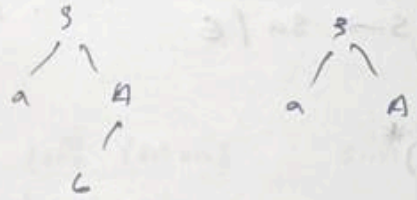
$S \rightarrow aA/bA/\varepsilon$

$A \rightarrow a/b/\varepsilon$

5) start with a and end with B

$S \rightarrow aA$

$A \rightarrow aA/bA/b$



6) starts and end with different symbol

or $S \rightarrow aA/bB$

$A \rightarrow aA/bA/B$

$B \rightarrow aB/bB/a$

7) starting and ending with same symbol

$S \rightarrow aA/bB$

$A \rightarrow aA/bA/a/\epsilon$

$B \rightarrow aB/bB/b/\epsilon$

a) All even length string

$S \rightarrow aaA/abA/baA/bbA/\varepsilon$

~~$A \rightarrow aA/bA/aa/bb/ab/ba$~~

$A \rightarrow aaA/abA/baA/bbA/\varepsilon$

9) All odd length strings

$S \rightarrow aA \mid bA$

$A \rightarrow aaA \mid abA \mid baA \mid bbA \mid \varepsilon$

10) no 2 a's come together

$S \rightarrow aA \mid bB \mid \varepsilon$

$A \rightarrow baA \mid bbA \mid \varepsilon$

$B \rightarrow aA \mid \varepsilon$

11) neither 2 a's nor 2 b's comes together

$S \rightarrow aA \mid bB \mid \varepsilon$

$A \rightarrow baA \mid b \mid \varepsilon$

$B \rightarrow abB \mid a \mid \varepsilon$

Identities of regular expressions

1. $\phi + R = R + \phi = R$
2. $\phi \cdot R = R \cdot \phi = \phi$
3. $\varepsilon \cdot R + R \cdot \varepsilon = R$
4. $\varepsilon^* = \varepsilon$
5. $\phi^* = \varepsilon$
6. $R + R = R$
7. $R^* R^* = R^*$
8. $(R^*)^* = R^*$
9. $R^* R = R R^* = R^+$

10. $\varepsilon + RR^* = R^*R + \varepsilon = R^*$

11. $(PQ)^*P = P(QP)^*$

12. $(P+Q)R = R(P+Q) = RP+RQ$

Closure properties of regular languages

A set is closed under an operation, if during the operation on the given set always produces the member of the same set.

Regular languages are closed under following

i) union

Regular languages are closed under union

L_1 and L_2 are regular, then $L_1 \cup L_2$ is also regular

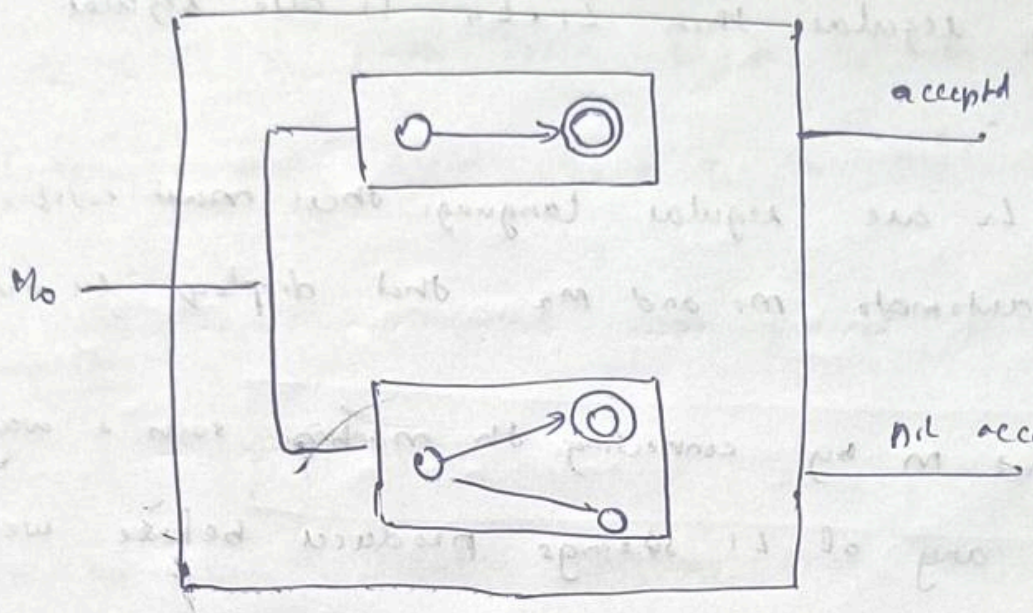
Proof

Since L_1 and L_2 are regular languages there exist a finite state automata ~~that~~ M_1 and M_2 that displays L_1 and L_2 respectively. We can make a new machine M for the language L_1 union L_2 from M_1 and M_2 in the following way. M will accept the string w if either M_1 or M_2 ~~with~~ accepts ~~that~~

Since we can make a finite state automata M for $L_1 \cup L_2$ $L_1 \cup L_2$ is regular. So regular languages are

closed under union

ii) ~~intersection~~



Intersection

Regular languages are closed under intersection

L_1 and L_2 are regular, then $L_1 \cup L_2$ is also regular

Proof

Since L_1 and L_2 are regular languages there exist a finite state automata M_1 and M_2 that displays L_1 and L_2 respectively

We can construct M by parallely running M_1 and M_2 and M will be accepting ~~the Lang~~ only when both M_1 and M_2 is accepting

Concatenation

Regular languages are closed under concatenation

L_1 and L_2 are regular then $L_1 \cdot L_2$ is also regular

Proof

Since L_1 and L_2 are regular languages there must exist finite state automata M_1 and M_2 that display L_1 and L_2 respectively

We can construct M by connecting the machine such a way that all or any of L_1 strings produced before we get to the machine that produces L_2. So

Kleen star

Regular languages are closed under kleen star

Kleen star is repeated concatenation and we know that concatenation is closed.

So Regular languages are closed under kleen star

Reverse operation

Regular languages are closed under reverse operation

We can make an automata from reverse of L. If L is regular then L^R is also regular

If there is a path from q_0 to q_f in M then there will be a path from q_f to q_0 in M^R

Complement

Since L_1 is a regular language there exist a finite state automata M. change all accepting states of M as non accepting states and vice versa. The new machine will accept the language L'

Difference

Regular languages are closed under difference

L_1 and L_2 are regular then $L_1 - L_2$ is also regular

Proof $L_1 - L_2$ or $L_1 \cap \overline{L_2}$

Proof

Since L_1 and L_2 are regular language there exist a FSA M_1 and M_2 that displays L_1 and L_2 respectively

Construct M such that FSA of M will be the pair where M_1 state is final and M_2 state is not final

Homomorphism

Converting 1 set to another while leaving its property

$$h(L) = \{ h(w) \mid w \in L \} \quad \text{where } h: \Sigma \to \Gamma^*$$

Γ - one
$\Sigma = \{0,1\}$
$\Gamma = \{a,b\}$

eg: $h(0) = aa \qquad h(1) = bbb$

if $L = 00, 10$ |

$h(L) = \{aaaa, bbbaabbb\}$

Inverse homomorphism

$h^{-1}(aa) = 0 \qquad h^{-1}(bbb) = 1$

- Using homomorphism prove that $L = a^n b^n c^{2n}$ is not regular

given that $a^n b^n \quad n \geq 0$ is not regular

// here we have to remember how many a's are there eg if there are 200 a we have to remember 200 states

$L = a^n b^n c^{2n}$

Let $h(a) = a$

$h(b) = \varepsilon$

$h(c) = b$

homomorphism

$\longrightarrow \quad h(L) = a^n a^n b^{2n}$

$= a^{2n} b^{2n}$

⇓

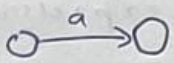
It is in the form $a^n b^n$

∴ it is not regular

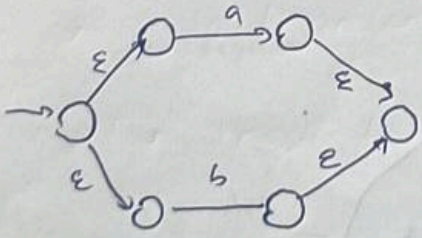
Equivalance of regular expression and ENFA

Every regular expression can be converted to equivalent ε NFA using the following constraints

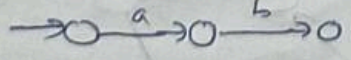
a



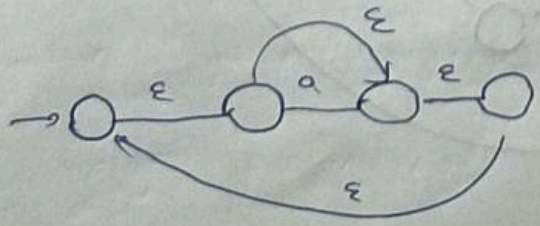
a|b
a+b



a.b



a^*



Steps for converting

1) Create a single start state for the automata and mark it as the initial state

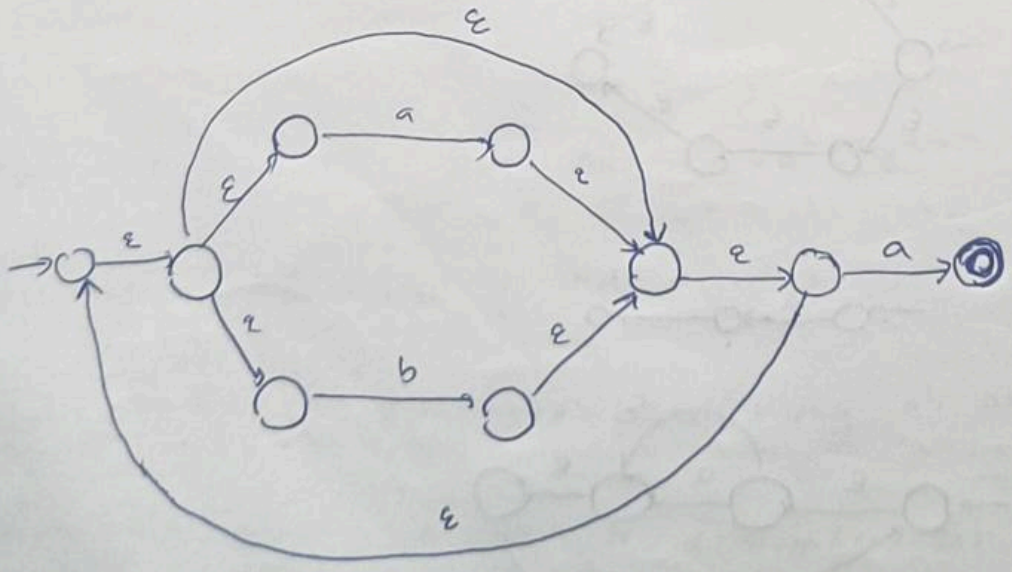
2) For each character in the regular expression create a new state and add an edge b/w previous state and new state with character as label

3) For each operator in the regular expression create a new states and add the appropriate edges to represent the operator (according to these rules... Prev page Pic)

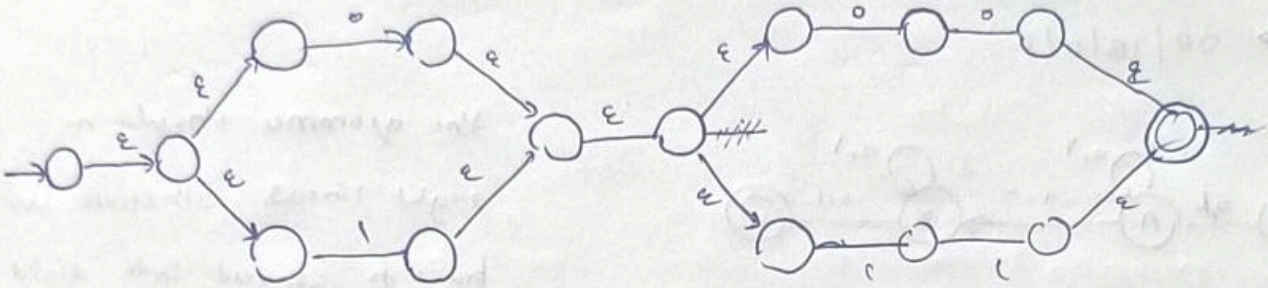
4) Mark the final state as the accepting state which is the state that is reached when the regular expression is fully reached

Q) Creat ENFA for the regular expression

$(a+b)^* a$



- $(0+1)^*(00+11)$



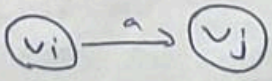
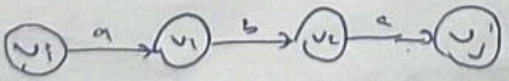
Equivalence of regular grammar and FSA

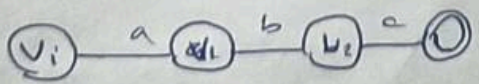
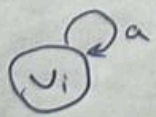
Regular grammar and FSA are equivalent since we can convert a right linear or left linear regular grammar to FSA and vice versa

eg:-

$S \rightarrow abS$ regular

$S \rightarrow ABa$ not regular

- $V_i \rightarrow aV_j$ in FSA is 
- $V_i \rightarrow wV_j \Rightarrow$ 

  $w = ab$
- $V_i \rightarrow w \Rightarrow$ 
- $V_i \rightarrow aV_i \Rightarrow$ 

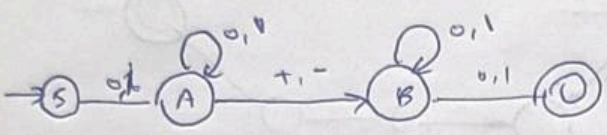
• Find equivalent FSA for following regular grammar

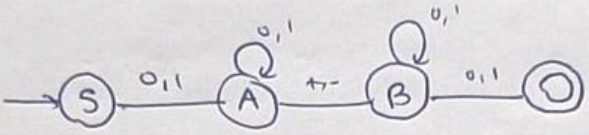
$S \rightarrow 0A \mid 1A$

$A \rightarrow 0A \mid 1A \mid +B \mid -B$ $+, -$ are terminals symbols

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

are





the grammar should be right linear, otherwise we have to convert into right linear

• Using Myhill-Nerode theorem PT $a^n b^n$ is not regular

$x = a^2$ $y = b^2$

Necessary conditions for a language to be regular

Pumping Lemma is the necessary condition for a language to be regular

Q: some a's followed by same number of b

$w = a^n b^n$

- we assume that there exists a FSA with 'n' states to accept L
- Take a string w from the language

  $w = a^n b^n$ such that $|w| \geq n$ | so there will be loops
- split w

  $w = xyz$ such that $|xy| \leq n$ and $|y| \geq 1$

  length of y atleast 1
- Now take $xy^kz \in L$ For every $k \geq 0$ then L is regular

eg:-

$w = a^n b^n$ $|w| = 2n > n$

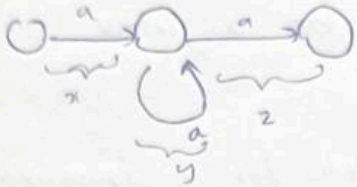
suppose y contains i no of a's

$\underbrace{a^{n-i}}_{x} \underbrace{a^i}_{y} \underbrace{b^n}_{z}$ $i \geq 1$ here we are pumping a^i

pump y 0 time $a^{n-i} b^n \notin L$ // here no of a's is less than no of b's

2 times xy^2z $a^{n-i} a^{2i} b^n$ $a^{n+i} b^n \notin L$

xy^3z $a^{n-i} a^{3i} b^n$ $a^{n+2i} b^n \notin L$



the have 3 states

$|w| \geq n$

here when we pump y 2 times it loops 2 times and reach final state

Q. Using pumping Lemma Prove that the Language ww regular or not

$L = \{ xx \mid x \in \{0,1\}^* \}$

ans

Assume there exist a FSA with n states that accepts L

Take a string w

$w = 0^n 1^n 0^n 1^n$

$|w| = 4n \geq n$

Split w into 3 string

$w = xyz$

0^n 1^n $0^n 1^n$
x y z

pump y 0 times

$\Rightarrow 0^n 0^n 1^n \notin L$

$0^n 1^n$ $0^n 1^n$
x y z

pump y 0 times

$\Rightarrow 0^n 0^n 1^n \notin L$

$0^n 1^{n-i}$ 1^i $0^n 1^n$
x y z

pump y 0 time $0^n 1^{n-i} 0^n 1^n \notin L$

Q. $L = \{ x \mid x = x^R \ \& \ x \in \{0,1\}^+ \}$

ans

Let $w = 0^n 1 0^n$

$|w| = 2n+1 > n$

divide w into xyz such $|xy| \leq n$ and $|y| \geq 1$ ($xy \neq \varepsilon$)

$$xyz = \underbrace{0^{n-i}}_{x} \underbrace{0^i}_{y} 1 \underbrace{0^n}_{z}$$

pump y 0 times:

$0^{n-i} 1 0^n \notin L$

$xy^k z$ for $k=0$ is $xz = 0^{n-1} 1 0^n$

∴ It is not regular

Q. $L = \{ x \mid x = wcw^R \ \& \ \varepsilon \{0,1\}^+ \}$

Let $x = \underbrace{0^n 1}_{w} \underbrace{1 0^n}_{w^R}$

$|x| = 2n+2 \geq n$

divide it into 3 parts such that $|xy| \leq n$ and $|y| \geq 1$

$$\underbrace{0^{n-i}}_{x} \underbrace{0^i}_{y} \underbrace{11 0^n}_{z}$$

pump y 0 time $\Rightarrow$ $0^{n-i} 1 1 0^n \notin L$ ∴ not regular

Q. ~~$L = \{0^{i^2} \mid$~~

$L = \{0^{i^2} \mid i \geq 1\}$ // length of string is perfect square

ans:- let $w = 0^{n^2}$ $|w| = n^2 > n$

$|xyz| = n^2$

xy^2z : $|x| + |y| + |y| + |z|$ $\quad 1 \leq y \leq n$

n^2 $\quad$ n (max value)

$|xy^2z| \leq n^2 + n$

$n^2 \leq |xy^2z| \leq n^2 + n$

$n^2 < |xy^2z| < n^2 + 2n + 1$

$n^2 < |xy^2z| < (n+1)^2$

n and nth are 2 consecutive numbers. It is not possible to have a perfect square b/w 2 consecutive perfect square

Q. $L = \{a^p \mid p$ is a prime number$\}$

ans Assume a FSA with n state that accept L

~~let us~~ let $w = a^n$ where n is a prime number

Assume that $|xy^iz| = |az| = m$ where m is prime no

take a string xy^mz

$|xy^m z| = |xz| + m \times |y|$

$= m + m \times |y|$

$= (1 + |y|)\, m$ $\quad$ [$|y|$ is atleast 1

$(1+|y|)m$ is not a prime number

$\therefore$ not regular

Q. $L = \{ a^{n!} \mid n \geq 1 \}$

1, 2, 6, 24, 120 ...

ans $w = a^{n!}$ $\quad |w| = n! \geq n$

Assume that $|xy^2z| = n!$

Let us assume that $|y| = i$ where $2 \leq i \leq n$

Let us consider $|xy^0z| = n! - i$

$n! - i > (n-1)!$

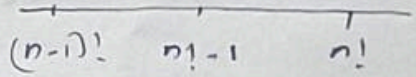
$(n-1)! \times n = n!$

$n! - i$ is always b/w $(n-1)!$ and $n!$

$xy^1z = n!$

$xy^0z = (n-1)!$

b/w 2 consecutive number there is no factorial



Q $L = \{ w \in \{a,b\}^* \mid |w|_a \geq |w|_b \}$

ans Let $w = a^{n+1} b^n$

$|w| = 2n+1$

split $w = xyz$ such that $|xy| \leq n$ and $|y| \geq 1$

a^{n+1-i} a^i b^n

x y z

pump y 2 times

a^{n+1-i} a^{2i} b^n

a^{n+1}

a^{n+1} b^i b^{n-i}

x y z

pump y 3 times

a^{n+1} b^{n+3i} b^{n-i} $\quad a^{n+1}$ b^{n-i+3i}

a^{n+1} b^{n+2i} $\quad i \geq 1$

$w = a^{n+1-i}$ a^{i-1} b^n

x y z

pump y 0 times

a^{n+1-i} b^n

$n+1-i < n \Rightarrow xy^2z \notin L$

$\therefore$ L is not regular

~~$n+1 > n+2i$~~ $\quad n+1 < n+2i$

s.no. of a's is less than no of b's,

$\therefore$ not regular

Context free languages

Context free languages are generated by context grammer

regular grammar

$A \to Bx$ or $A \to xB$ $\quad A, B$ are variable $\quad \{A, B\} \in V$

x is terminal $\quad x \in T^*$

$S \to aaaBc$ ✗ not grammer

$S \to aAcab$ not regular

Write grammer for $a^n b^n$

$S \to$ ~~Aa / Bb~~ / a $\quad aA / bB /$

$A \to$ ~~aA~~ / bB / aB / bA

|

- For regular grammar
  - There must be single variable at LHS
  - There must be single variable at RHS which is either at left side or right side

- Write grammar for $a^n b^n$

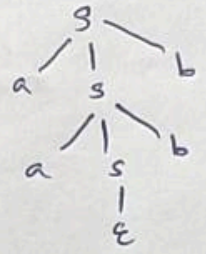
  $a^n b^n$ is not a regular language

Context Free grammar

Here there is no restrictions as in regular grammar. Here we can have multiple variable at right hand side

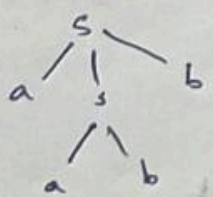
$L = a^n b^n \quad n \geq 0$

$S \to aA \quad A \to$ ~~$S \to aA$~~ $\quad S \to asb / \varepsilon$



$L = a^n b^n \quad n \geq 1$

$S \to asb / ab$



Format of context free grammer

$A \longrightarrow \alpha$

$A \in V$

$x \in (V \cup T)^*$ any number of variables or terminals

Every regular grammar is a context free grammer but not vice versa

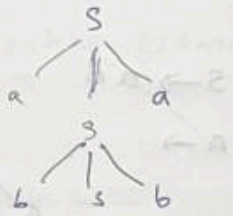
$A \to Bc$ it is both regular and context free grammer

Q. CFG for $\{w : w^R \mid w \in \{a,b\}^*\}$

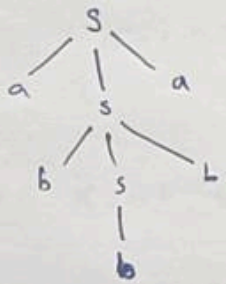
ans sample strings : aba, abba, aabbaa

A/B S

$S \to aSa \mid bSb \mid \varepsilon$



this give even length palindrome



$S \to aSa \mid bSb \mid a \mid b \mid \varepsilon$

Q. $L = a^i b^j c^i$ where $i, j \geq 1$

ans $S \to aAc$ $B \to Bb \mid b$

~~$A \to bAS \mid \varepsilon$~~

$S \to aSc \mid aBc$

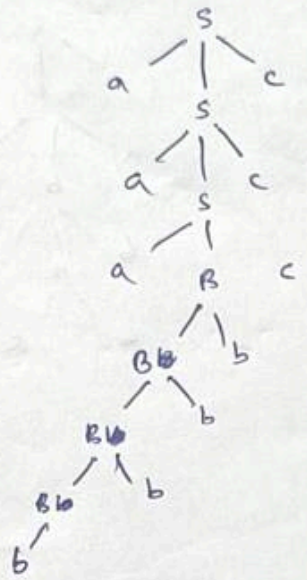
$B \to Bb \mid b$

$i, j > 0$

$S \to aSc \mid aBc \mid \varepsilon$

$B \to Bb \mid b$

eg:- aaabbbbccc



Q. $L = a^i b^i c^j \mid i,j \geq 1$

~~$S \rightarrow aSb \mid \epsilon$~~ ~~$aSbC$~~

$C \rightarrow$ ~~$\epsilon c \mid c$~~

$S \rightarrow A\ C$

$A \rightarrow aAb \mid ab$

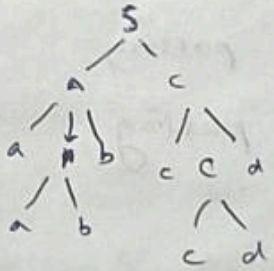
$C \rightarrow cC \mid c$

Q) $a^i b^i c^j d^j$

$S \rightarrow A\ C$

$A \rightarrow aAb \mid ab$

$C \rightarrow cCd \mid cd$



Q) $a^i b^j c^j d^i$

~~$S \rightarrow aBd$~~

$S \rightarrow aSd \mid aBd$

$B \rightarrow bBc \mid bc$

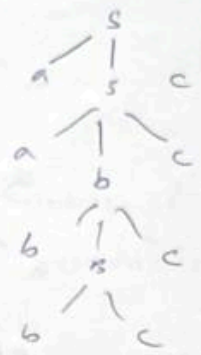
a) $a^m b^n c^{n+m}$ $m,n \geq 1$

$a^m b^n c^n c^m$

$m = 0$ included bc

$S \rightarrow aSc \mid aBc$

$B \rightarrow bBc \mid bc$ $\mid \varepsilon$



$a^m b^n c^{n+m}$ $m,n \geq 0$

$S \rightarrow aSc \mid B$

$B \rightarrow bBc \mid \varepsilon$

a) $a^n b^m \mid n \neq m, n,m \geq 0$

$S \rightarrow aAB \mid a \mid b \mid B \mid \varepsilon$

$A \rightarrow aA \mid \varepsilon$

$B \rightarrow bB \mid \varepsilon$

$S \rightarrow aSb \mid aA \mid bB \mid \varepsilon$

$A \rightarrow aA \mid \varepsilon$

$B \rightarrow bB \mid \varepsilon$

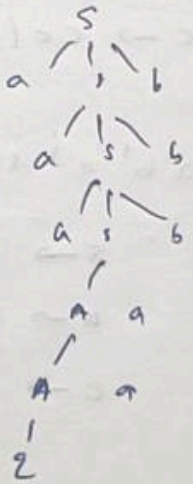
Application of context grammar

- top down parsing
- bottom up parsing

Parse tree / derivation tree

- Root of tree is always start symbol
- Intermediate nodes of tree are variables
- leaves are terminals

aaaaabbb



$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow w_3 \ldots \Rightarrow w_n = w$ $\quad w \in T^*$ only terminals

$w_1, w_2 \ldots \overset{w_{n-1}}{\in} (N \cup T)^*$

Each intermediate form of derivation is called ~~sentential form~~ sentencial form

~~if~~ ~~in~~

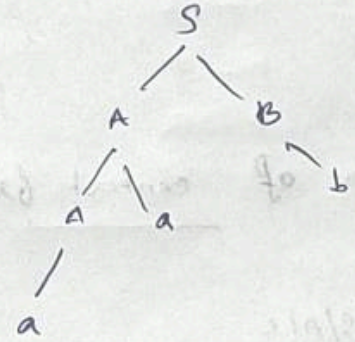
$S \overset{*}{\underset{G}{\Rightarrow}} w$

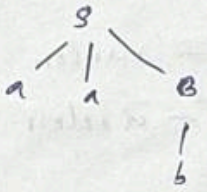
Q) $S \to AB / aaB$

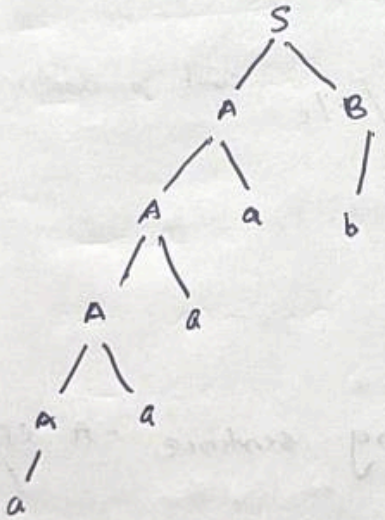
$A \to a / Aa$

$B \to b$

derive aab







amp

Ambiguity is the main problem with grammars

Compiler A grammar is ambiguous when there are more than one left most derivation or left right most derivation

Ambiguity

A grammar G is said to be ambiguous if there exists some string of language generated by G which has 2 or more left most derivations or 2 or more right most derivation (hence consiquently having 2

or more distinct parse tree)

9. Check whether the grammar is ambiguous or not

H.W

1) $S \longrightarrow S+S \mid S*S \mid a \mid b$

2) $S \rightarrow aSbS \mid bSaS \mid \epsilon$

3) $R \rightarrow R+R \mid RR \mid R^* \mid a \mid b \mid \epsilon$

4) ID -- then -- else

Simplification of context free grammars

unit production

$S \rightarrow aS/A/c \Rightarrow S \rightarrow aS/a/c$

eg:

$A \rightarrow a$

$B \rightarrow aa$ — useless

$C \rightarrow aCb$ — useless

A CFG may not use odd symbols for deriving sentence. A CFG can be simplified in following steps

1. Eliminating useless symbols
2. Eliminating ϵ productions
3. Eliminating unit productions

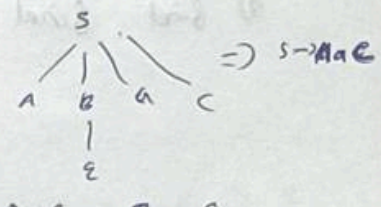
In the example B is useless

* 1. A variable in a grammar is said to be useless if there is no way of getting a terminal string from it and or it cannot be reached from starting symbol

2. Eliminating ε productions

$S \to ABaC, A \to BC, B \to b/\varepsilon, C \to B/\varepsilon \quad D \to d$

$B \to \varepsilon \quad C \to \varepsilon \quad A \to \varepsilon$



$S \to ABaC / AaC / ABa / Aa / a / BaC / Ba / aC$

$A \to ABC / B / C$

$B \to b$

$C \to D$

$D \to d$

3. Eliminating unit productions

In above eg $C \to D \quad D \to d$ it can be replaced by $C \to d$

General form of unit production is

$A \to B$ where $A, B \in V$

Q. $S \to Aa / B$

$B \to A / bb$

$A \to a / bc / B$

what are unit production here

ans

$S \to B \rightleftarrows A$

dependency

All three are 4

$S \to B$

$B \to A$

$A - B$

$S \to A$

am 1) identify non unit production divide into non unit and unit production

$S \rightarrow Aa$

$B \rightarrow bb$

$A \rightarrow a \mid bc$

2) identify dependencies

$S \rightarrow B \rightleftarrows A$

3) find final set of unit production

$S \rightarrow B$

$B \rightarrow A$

$A \rightarrow B$

$S \rightarrow A$

final am

$S \rightarrow Aa \mid bb \mid a \mid bc$ S is associated with B and A so add B's and A's non unit production

$B \rightarrow bb \mid a \mid bc$

$A \rightarrow a \mid bc \mid bb$

1) Divide the production into non unit and unit production.

2) Keep the non unit production

3) For each derivation relationship, add non unit production

4)

imp Simplify the grammer

$S \to AB/\varepsilon$

$A \to CD$

$B \to DC$

$D \to dB/\varepsilon$

$C \to cC/\varepsilon$

ans 1) eliminate useless symbols

2) eliminate ε production

$S \to \varepsilon$ $D \to \varepsilon$ $C \to \varepsilon$

as CD goes to ε A B goes to C

so,

$S \to AB/A/B$

$S \to AB$

$A \to CD$

$B \to DC/D$

$D \to dB/$ [illegible] → remove

$C \to cC/c$

$S \to AB/A/B$

$A \to CD/C/D$

$B \to DC/D/$ [illegible]

$D \to dB$

$C \to cC/c$

3) eliminate unit production

$S \to AB/A/B$

$A \to CD/C/D/c/dB$

$B \to DC/D/\varepsilon/c/dB$

$D \to dB$

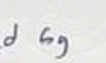
$C \to cC/c$

Push down automata (PDA)

Transition depends on current symbol, symbol in stack

Push down automata have absolute memory called stack

Q. Design a push down automata for accepting a language $a^n b^n$

An empty stack is represented by Z_0 

last symbol is ε

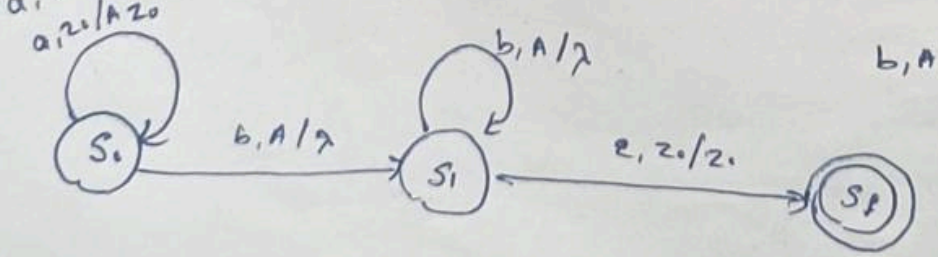
When an a come A is pushed into stack

When b come A is poped out of the stack

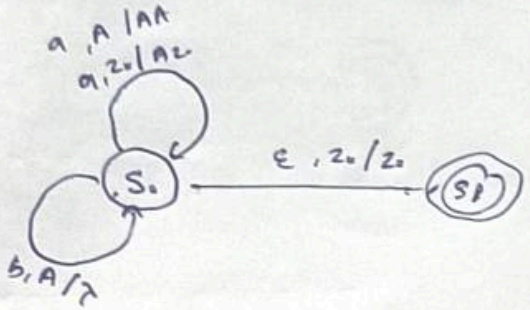
When ε come : if the stack contains Z_0 then equal no of a and b's

$a, Z_0/AZ_0$ => top Z_0, a comes / top is AZ_0

$b, A/\lambda$ => when b comes top is A and λ => poping



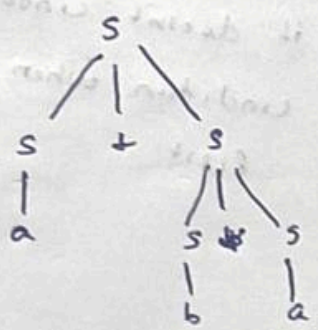
When we give middle pushing and poping as a single state our pattern is changed (equal no. of a's followed by equal no. of b's or same no of a's and same number of b's

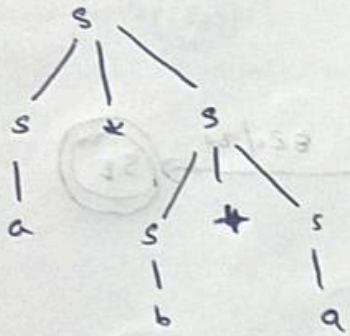


Now $L = \{ w \mid |w|_a = |w|_b \}$

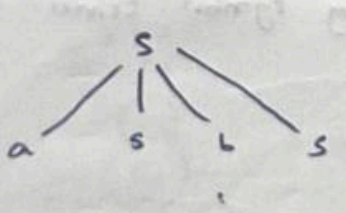
Check whether the grammer is ambiguous or not

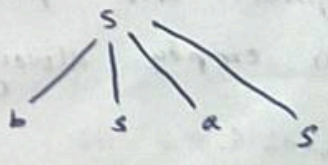
i) $S \rightarrow S+S \mid S*S \mid a \mid b$



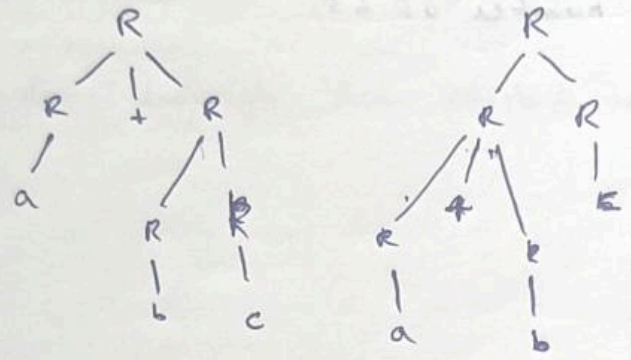


ii) $S \rightarrow aSbS \mid bSaS \mid \varepsilon$





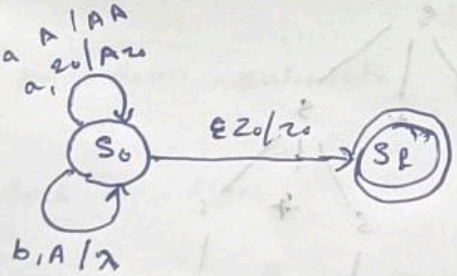
3) $R \rightarrow R+R \mid RR \mid R^* \mid a \mid b \mid c$



4a)

4) if then else

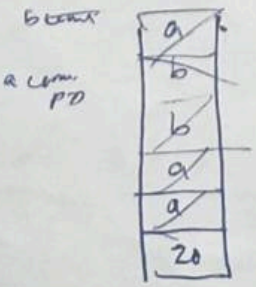
$L = \{ w \mid |w|_a = |w|_b \}$



→ it doesn't contains the condition when b comes first

Whatever the symbol comes push into stack. if the same symbol comes push. else pop. If any symbol comes when stack is empty always push

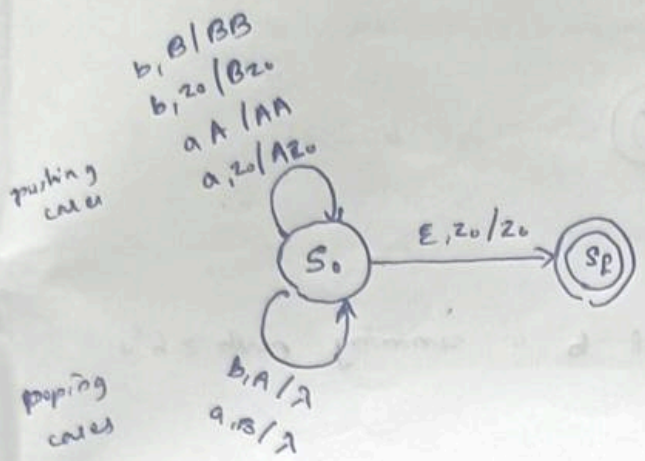
aabbbabaab ε



b comes pop

when the ε comes if the stack is empty then it contains equal no. of a and b else unequal

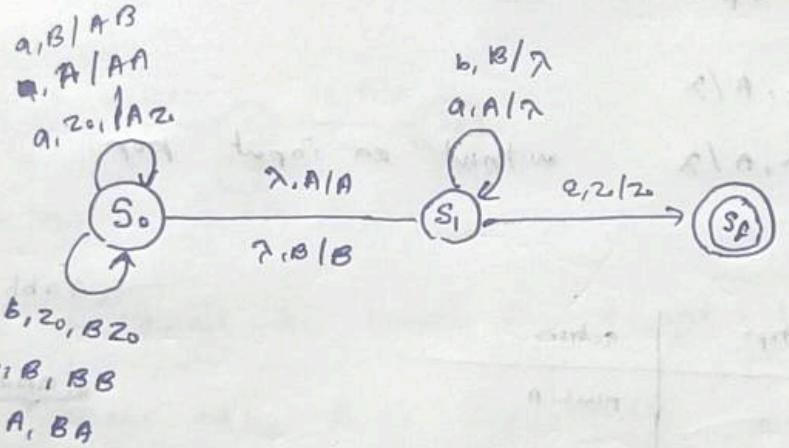
$L = \{ w \mid |w|_a = |w|_b \}$



ε, Z0 => when ε come and stack is empty

Non deterministic pushdown automata

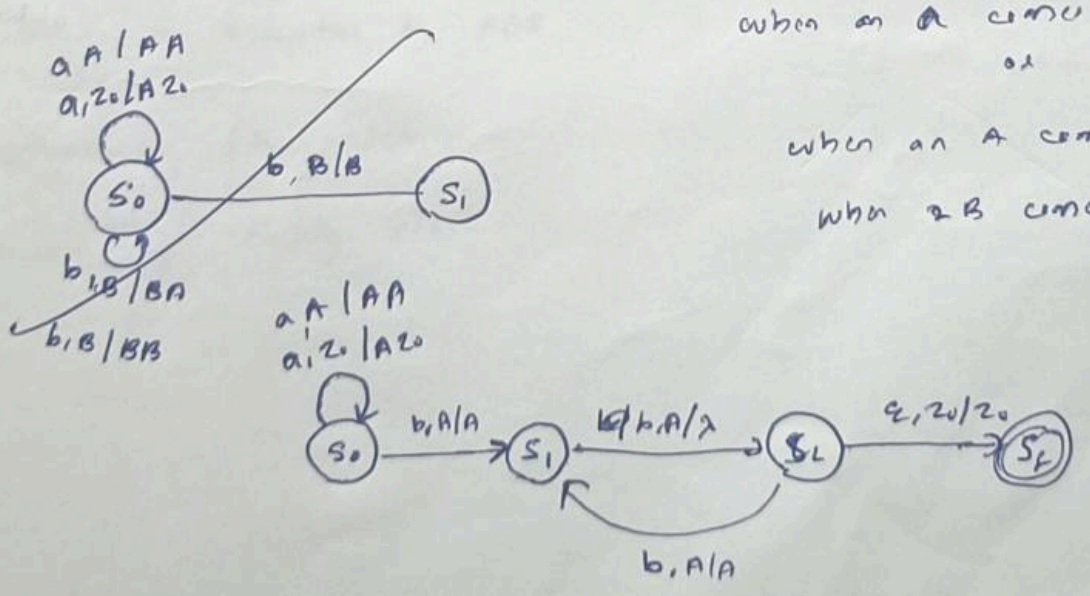
e) Construct an NPDA for ww^R (2nd half is reverse of first)



λ, A/A } NE I/P
λ, B/B } transition

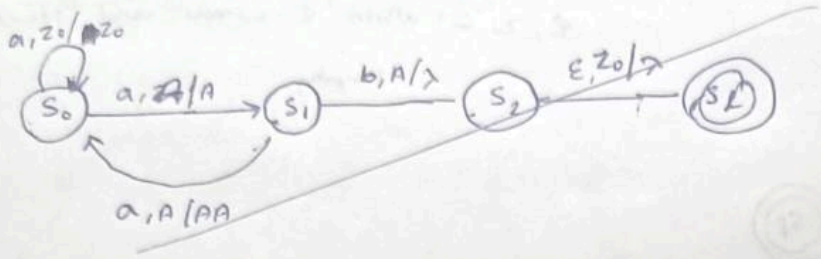
wcw^R - DPDA

e) Construct an NPDA $a^n b^{2n}$



when an A come push 2 A
or
when an A come push 1 A
when 2 B come pop one A

Q. Number of a's is twice no of b's



Initially when a comes push into stack. If b is coming push 2 b's into stack.

If any symbol comes and the same symbol is top of the stack then push.

When b comes and top of stack is A we have to pop 2 A's

Transition :- b, A/λ

λ, A/λ without an input pop

I/p	stack top	action
a	A	push A
b	A	

baa

bb

aab

aabbaa

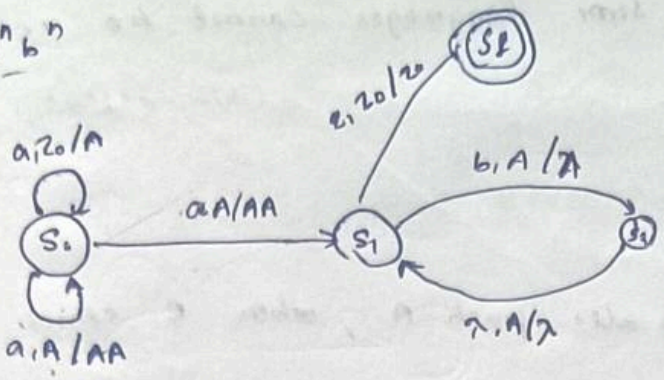
aba

aAZ0

b came do nothing

abbaaa

a) $a^{2n}b^n$

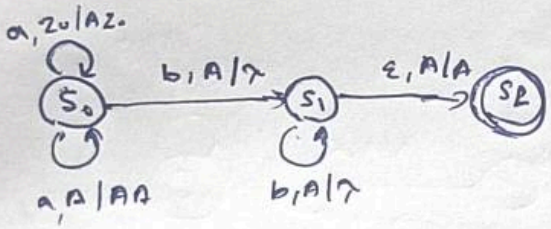


b, A/A :- when B comes and stack of top is A do nothing (same remains at top of stack)

b, Z0/BBZ0 :- when a b comes push 2B)

a) $a^m b^n$ where $m > n$, $a^n b^m$ $n > m$

whether the stack should be empty always to accept



stack should not be empty to accept. It is accept only when there are A in the stack

Two methods of accepting by PDA

i) acceptance by final state

ii) acceptance by empty stack

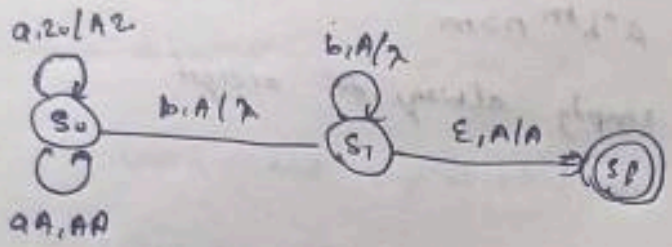
Due to strictly LIFO nature some languages cannot be accepted by PDA (Ex. $a^n b^n c^n$)

* $a^n b^m c^{n+m}$

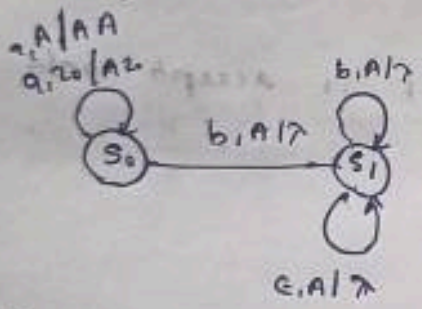
When a comes push A, for b also push A, when c comes pop A

Q) $a^n b^m \quad n > m$

accepted by final state



accepted by empty stack



ε,A/λ

when no input is there and top of stack is A then pop

NPDA

PDA is the powerful device accepting CFG

Definition

$\langle S, \Sigma, \Gamma, \delta, s, z_0 \rangle$

S - state

Σ - input

Γ - stack top

δ - transition fn

s - starting state

z_0 - empty stack

Γ - the

$\cup$ - union

transition function

$$\delta : S \times \{\Sigma \cup \{\epsilon\}\} \times \Gamma \longrightarrow S \times \Gamma^*$$

- Suppose $\delta(\varepsilon, q, X) \longrightarrow (p, \alpha)$ $\quad X$ - current top

  Show this as instantaneous description or move ($\vdash$)

ans

$$(q, aw, X_B) \vdash (p, w, \alpha B)$$

// a - is consumed

aw { a - current symbol
     w - remaining

X_B - X - current top
        B - remaining symbol in stack

~~Language of~~ PD

Language of a PDA

A string can be accepted by 2 method

i) acceptance by final state ii) acceptance by empty stack

Acceptance by final state

$$L(P) = \{ w \mid (q_0, w, z_0) \vdash_P^* (q_f, \varepsilon, \alpha) \}$$

w is string belong to L, starting q_0 and top is z_0 after apply all inputs w ~~after~~

$\vdash_P^*$ after many production

we reach final state when ε comes and stop if stack is α (contains some symbol)

Acceptance by empty stack

$$N(P) = \{ w \mid (q_0, w, z_0) \vdash_P^* (q_i, \varepsilon, z_0) \}$$

q_i - it not need to be final state

Equivalence NPDA and CFG

S → aA A → aABC / bB / a B → b C → c

aaabc

pw s on top of stack

replace

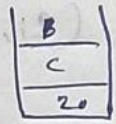
Then check productions of S. Since i/p is a check production of S starting with a.

Input a is consumed. Replace S → A

Now look the production of A, starting with 'a' aABC

a is consumed. Replace A with ABC in reverse order

Now look the production of A. A → a a is consumed. No variable is present in that production to replace with A. so replace with nothing (remove A)



B → b aaab is consumed. remove B

c has only 1 production. C → c so c is consumed and c is removed.

Now aaabc is produced.

$\delta(q_0, \varepsilon, Z_0) \Rightarrow \{(q_1, SZ_0)\}$ at q_0 without any i/p, Z_0 stack top pushed S ont the top of stack

$\delta(q_1, a, S) = \{(q_1, A)\}$ // if stack top is S and i/p is a a is consumed replace S with A

$\delta(q_1, a, A) = \{(q_1, ABC), (q_1, \lambda)\}$ // $(q_1, \lambda) \Rightarrow A$ how a production starting with a. So when a comes and stack top is a you can take either of this

$\delta(q_1, b, A) = \{(q_1, B)\}$

$\delta(q_1, b, B) = \{(q_1, \lambda)\}$

$\delta(q_1, c, C) = \{(q_1, \lambda)\}$ There are set of moves expected when a particular i/p comes

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

Here all the P variable gives with a single terminal and some variables so terminal can be consumed and variable can be added to stack

eg: $S \rightarrow aBdd$

$S \rightarrow aBC$
$C \rightarrow dD$
$D \rightarrow d$

Greibach normal form

A In a production A variable gives single terminal and some variable

application

directly converting CFG to PDA

It is making some kind of systematic derivation

For a five length string 5 derivation are required

$S \to aA$

$S \to aaABC$

$S \to aaaBC$

$S \to aaabC$

$S \to aaabc$

Q)

$S \to aSBb/a$ Construct a NPDA

$S \to aSB/a$

$B \to bb/bC$

$C \to b$

$\delta(q_1, \varepsilon, z_0) = \{q_1, Sz_0\}$

$\delta(q_1, a, S) = \{(q_1, SB), (q_1, \lambda)\}$

$\delta(q_1, b, B) = \{(q_1, C)\}$

$\delta(q_1, b, C) = \{(q_1, \lambda)\}$

$\delta(q_1, \varepsilon, z_0) = \{(q_1, z_0)\}$

imp

Pumping Lemma for context free languages

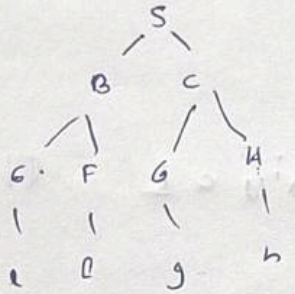
Chomsky normal form (CNF)

$A \rightarrow BC$

$A \rightarrow a$

A variable gives either 2 variable or single terminal

- A grammar with m variable (m is the no of variable in the path)



2^{m-1}

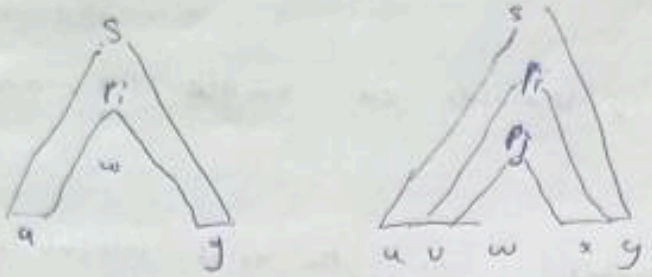
terminal = 2^{m-1}

- If m variables involved in a derivation then leaf (derived sentence) contains 2^{m-1} symbols
- If we take n length sentence where $n > 2^m$, $m+1$ variable are required to derive $n = 2^m$ length sentence

- Assume 'm' unique variable, ~~require~~ take $n = 2^m$ strings

Then derivation requires $m+1$ variables. But we have only m variables, so atleast one variable is repeating

$z = uvwxy$ & v and x are pumping path



* $z = uvwxy$

$|vx| \geq 1$

$|vwx| \leq n$ then $uv^i wx^i y \in L$ (i can be any value from 0)

(3) Prove the following language is not context free

$$L = \{ a^n b^n c^j \mid n \leq j \leq 2n \}$$

or

Assume there exist a CFG accepting this with m variable

take a string whose length is greater than 2^m

let $z = a^n b^n c^{2n}$ such that $|z| \geq 2^m$

we divide z into 5 parts $z = uvwxy$ &

such that (i) $|vx| \geq 1$

$|vwx| \leq n$

u = a vwx may be consisting a's _ y can be c^j

the key pumping lemma

Suppose L is context free then there exist a +ve integer n s.t and for any z element of L with length of $z >= n$ can be decomposed as uvwxy satisfying the following conditions

1. length of $|vx| \geq 1$ $vx \geq 1$
2. $|vwx| \leq n$
   $uv^i w x^i y \in L$ for all $i \geq 0$

we decompose z satisfying condition 1 and 2, then

vwx consists of one symbol (only a's / only b's / only c's) or

strea of 2 symbols (a's and b's or b's or c's)

then we have 5 possibilities

1) if vwx consists of b's only then a's will be exactly n. the word consist of exactly n a's 2n c's and less than n b's (vx is not) now word cannot be in L

ii) a's only

iii) c's only

iv) if wx contain strings of 2 symbols say a's and b's then we can choose i such that uv^iwx^iy has more than $2n$ occurences of a or b and exactly 2^n occurence of c. Thus uv^iwx^iy cannot be in n

v)

a) $L = a^p \mid p$ is prime number

$z = uvwxy$

take string uv^iwx^iy with $i = 0$

$|uwy| = p$, prime

now take a string.

$uv^pwx^py := p + p(|vx|)^1$

$2p$:. not prime number

Closure properties of CFLs

✓ Union
Concatenation
Kleen closure

× Intersection
Complimentation

Union

Let L_1 and L_2 be 2 CFL's

$L_1 = a^n b^n$ $\quad L_2 = c^m d^m$ $\quad n, m \geq 0$

$S_1 \rightarrow aS_1b \mid \varepsilon$ $\quad S_2 \rightarrow cS_2d \mid \varepsilon$

$L = L_1 \cup L_2$

$= a^n b^n \cup c^m d^m$

$S \rightarrow S_1 \mid S_2$

Union of CFL is again a CFL

Concatenation

Let There are 2 CFL $L_1 = a^n b^n$ $\quad L_2 = c^m d^n$ $\quad n, m, n \geq 0$

$S_1 \rightarrow aS_1b \mid \varepsilon$ $\quad S_2 \rightarrow cS_2d \mid \varepsilon$

$L = L_1 L_2$

$= a^n b^n c^m d^m$

$S \rightarrow S_1 S_2$

Klean closure

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

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

$S_1 \rightarrow S_1 S$

$S \rightarrow aSb \mid \varepsilon$

Intersection

Let there are 2 CFL's

$L_1 = \{a^n b^n c^m \mid n, m \geq 0\}$

$L_2 = \{a^m b^n c^n \mid n, m \geq 0\}$

$L = L_1 \cap L_2$

$= \{a^n b^n c^n \mid n \geq 0\}$ it is not CFL

(if a comes we put in stack, if b comes we pop each a, when c comes we can't there is nothing in the stack. so we cannot check the match of c

Complimentation

Let L_1 and L_2 are CFL

Assume $\bar{L}_1$ and $\bar{L}_2$ are CFL

then

$L_1 \cap L_2 = \overline{\bar{L}_1 \cup \bar{L}_2}$ demorgan's law

∴ we know union of 2 CFL's are context free

∴ $\bar{L}_1 \cup \bar{L}_2$ is CFL

we know $L_1 \cap L_2$ is not context free

∴ the complement of the 2 context free languages are not context free

Intersection with regular language

Every regular language is context free

a^5b^5 is a regular language because there are finite number of states. ie 10+1 states. ∴ it is regular

$L_1 = \{a^n b^n \mid n \geq 0\}$

$r_1 = \{a^5 b^5\}$

$L_1 \cap r_1 = a^5 b^5$

$L_1 \cap r_1 \rightarrow = r'$

Deterministic PDA

A DPDA has no more computing power than an NPDA

There exists certain languages that are accepted by NPDA but not by DPDA

eg:- $L = wcw^R$

Here in DPDA we don't know when to start popping.

Conditions

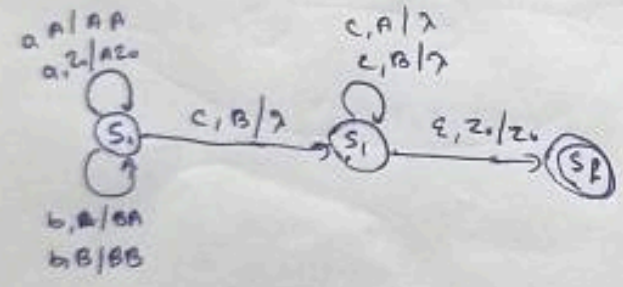
1) For a given I/P symbol a DPDA can't have 2 different choices of states / 2 different choices of strings to be pushed into stack

2) A DPDA can't have ε moves

3) A DPDA states that have ε moves (ie input is over / last symbol) can have only that move

Design an NPDA and DPDA for the language

ie $a^i b^j c^k$ given $i, j, k \geq 1$ and $i+j = k$



Context sensitive Languages

Instead of stack we use read and write tape (table ~~is~~ called input table whose length is infinite)

eg: $a^n b^n c^n$

$	a	a	b	b	c	c	#	#	#

when a comes put X there and check for b if b is not put Y there and check for c if c is not found then put Z.

X	a	Y	b	Z	c

when the stack contains '#' keep it as such and move to ~~right~~ right

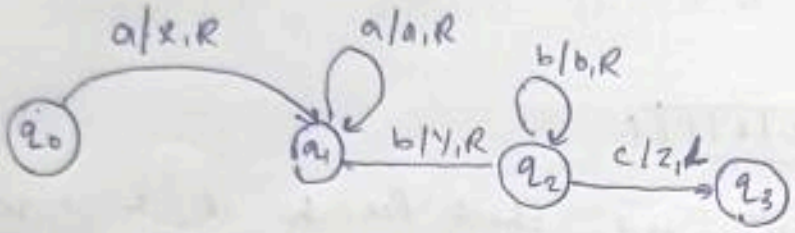
Again check from

When we saw 1st saw, we rewrite it with X, then move to right until it saw b, then rewrite b with Y, then move to right until it saw 1st c the rewrite it as Z. The head will come back to left (at first it then move to right)

Repeat this process.

When Y comes after X mean all 'a's are exhausted then check Z comes after Y. If after Z is blank symbol (B) then it moves to halting state (a

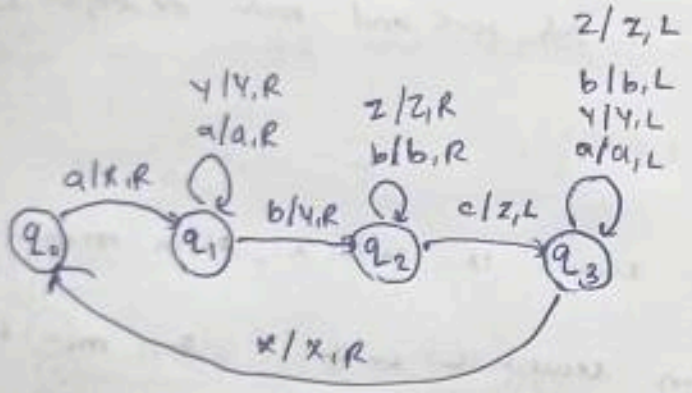
B | B B | a | a | a | b | b | b | c | c | c | B | B B



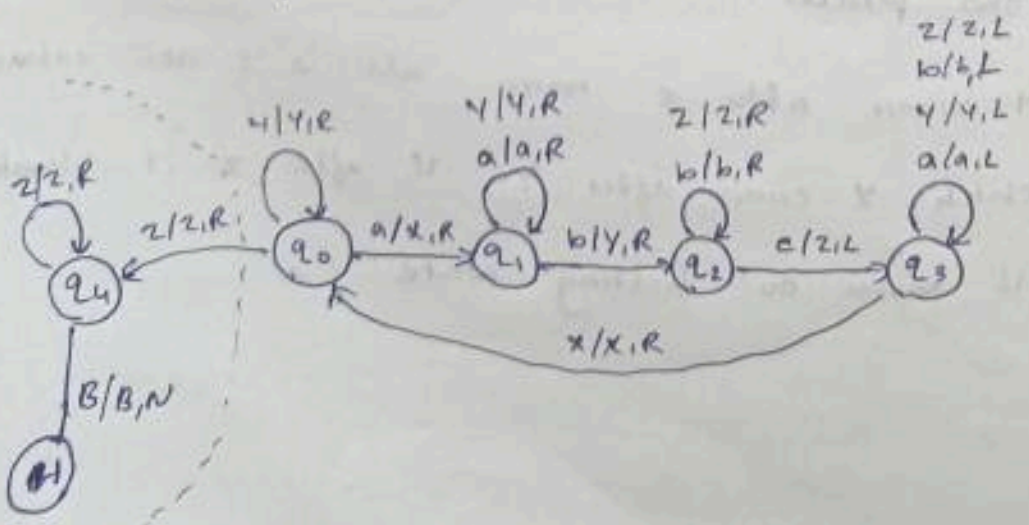
a/x,R : when a comes write x and move head to right

a/a,R : when a comes write to right

on seeing 1st c we have to go back until x is reached on reaching x head moves to right. In the 2nd stat B/a/Y b, we have to skip Y also



Here when we marked all a then, after x there is no 'a'. Here we are not mentioning what happens when Y comes after Y:



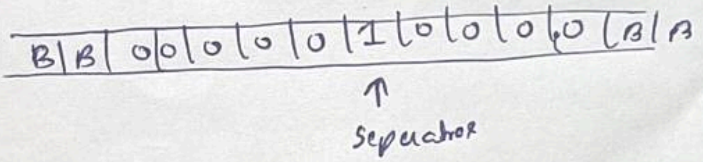
final state

- Design a turing machine to accept $a^n b^n$

- Design a turing machine to add 2 numbers

  here we use unary

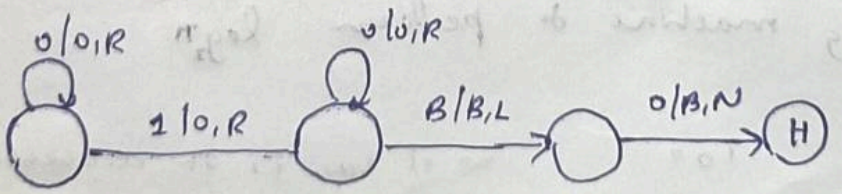
  5 - 00000 4 - 0000



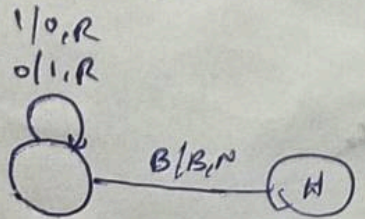
FC

while adding when 1 comes make it zero and move to right

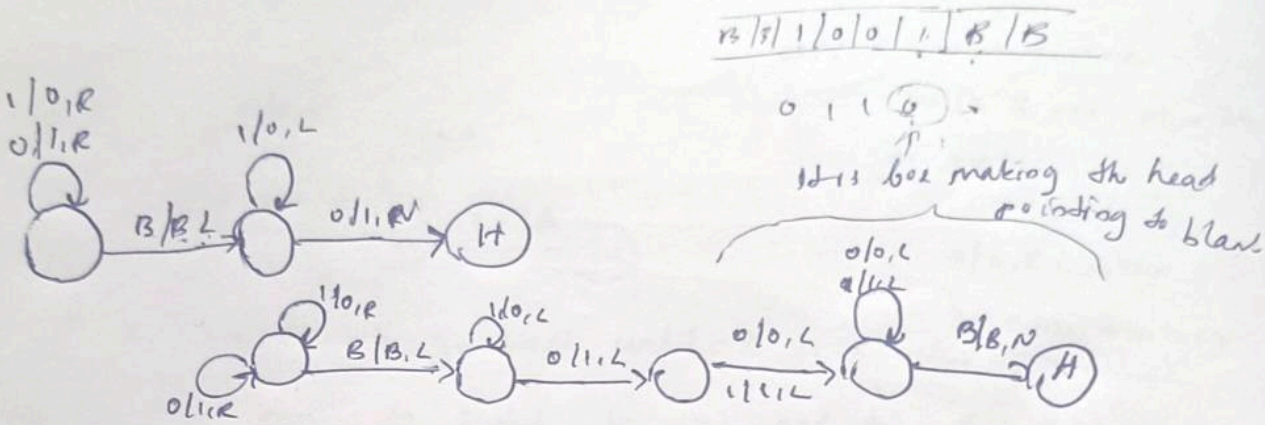
on reaching blank symbol move left and make the zero as B.



- Design a turing machine to find 1's complement of a numbers



- Design turing machine to find 2's complement of a binary number



- Design a turing machine to accept a language with equal no of zero and ones

- Design a turing machine to perform $\log_2 n$

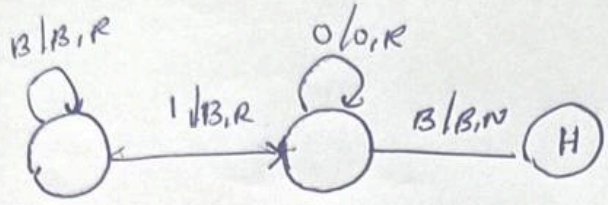
$\log_2 4 = 2$ 100 no of zeros in the answer

$\log_2 8 = 3$ 1000

$\log_2 16 = 4$ 10000

B	1	0	0	0	0	B

make that one as blank



imp. Design a turing machine to multiply 2 numbers

am numbers will be represented using unary form

2 x 3

B|0|0|1|0|0|0|B

when a zero comes, repeat the no