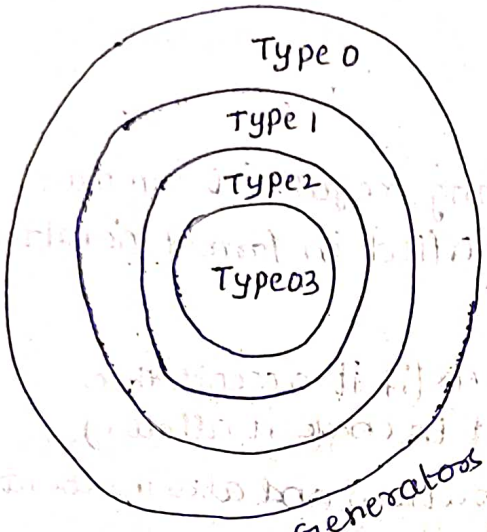
13/09/2023.

Formal Languages & Automata Theory.

Chomsky hierarchy



	Grammar	Automata	Language Type.
Type0:	Unrestricted Grammars	Turing Machine	Recursive / Recursive Enumerable languages.
Type1:	context Sensitive Grammars	Linear Bounded Automata	context sensitive languages.
Type2:	context Free Grammars	push Down Automata	context Free Languages
Type3:	Regular Grammars	Finite Automata	Regular languages.

⇒ All regular Grammars are is context free.

Grammar: set of Rules to form language.

Automata: checks the Grammar is to accept or reject according to the Grammar

→ strings pattern or formation is important than the meaning of it.

eg:- email id

soujana.cse@.rgukt.ac.in

Format: username @ Domain : .org, .in, .ac.in, .com → to specify this format something has to be there

→ pattern of the string is important not meaning.
↳ Formal Languages
Usage: In compiler design.
In Form filling.
mobile number
Date of Birth
Variable names patter → using regular Grammars
Grammar: Set of Rules specified in formal notation.
Syntax Representation
Automata: Recognize. Syntax (if it accepts then
⇒ Paranthesis Balance: need to check it allows)
the pattern and also the count.

Memorization past: For this context free
some only here ↓
context sensitive ↓
no restriction of memorization
in unrestricted.

phase:
1) lexical Analysis: variable names, function names,
2) syntax " : declaration mapping
3) semantic " :
4) Intermediate code Generation.
5) code " :

c+b=a X
a=c+b ✓
Type checking
↳ Semantics

Alphabet: (Σ): Set of symbols to form characters
eg: 0,1 strings.
14/09/2023 Thursday,
String: sequence of symbols set sequence
↓ order matters
Language(L): Set of strings formed
over given alphabet ABC → 3! → 6 strings
eg: All binary strings ending with 0 → then it become
a language.

substring (n)
sub ub bs st tr ri in ng (n-1)
sub ub bs st tr ri in ng (n-1)

prefix:

For 'n' length string how many prefixes, suffixes, substrings, subsequences possible?

prefix → n+1
ϵ → epsilon (no string or empty string)
p
pr
pre
pref
prefi
prefix

Suffix
Suffix
uffix
ffix
fix
ix
x

prefix: By removing 0 or any number of characters from end.

Suffix: By removing 0 or any number from start

substring: By removing characters from both the end

substring → 1 (n)
substring
su, ub, bs, st, tr, ri, in, ng, (n-1)

ABC
A B C → 3
AB BC → 2
ABC → 1

n + (n-1) + (n-2) + ... + (n-(n-1))
= n(n+1)/2 + 1 (because ϵ)

subsequence: we can even remove characters in the string.
ubeq
→ we can take or not
→ 2^n (it includes ϵ and string itself present in it)

power of an alphabet.

Σ^1 → all the strings of 1 length.
$\Sigma = \{0, 1\}$
$\Sigma^1 = \{0, 1\}$ ⇒ 2.
$\Sigma^2 = \{00, 01, 10, 11\}$
$\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$
Σ^n → all n length strings formed over sigma.
those symbols can be repeated.
$\Sigma = \{a, b, c\}$
$\Sigma^2 = \{aa, ab, ac, ba, bb, bc, ca, cb, cc\}$
with repetition = ? ∴ depending on sigma number
how many varies.

$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \ldots\ldots$

$= \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001 \ldots\ldots\}$

all possibilities of Σ.

Language → Rule
automata → Recognise this language.

Language
Alphabet: Σ
Language: set of strings
Automata:
Grammars: should generate only those (strings)

eg:- even length strings → one language
odd length strings → another language.
prime length strings → "

String end with 0 → "

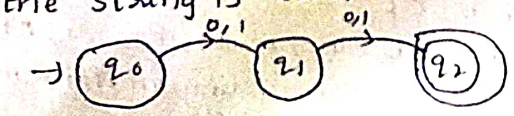
⇒ There exist n number of languages.

Finite Automata / Finite state Machine.

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

Q = finite set of states
Σ = input alphabet.
δ = Transition function.
$\delta: Q \times \Sigma \to Q$
q_0: Initial state
F: Non-empty set of final states.

→ one symbol at a time, it can go to any one of the state.
one after the other all the string is read,
coming to the end char of string, if at this time if the machine is in the final state then the string is accepted otherwise rejected.



$Q = \{q_0, q_1, q_2\}$
$\Sigma = \{0, 1\}$
$\delta = \{ \delta(q_0, 0) \to q_1$
$\delta(q_0, 1) \to q_1$
$\delta(q_0, 0) \to q_2$
$\delta(q_1, 1) \to q_2 \}$

q_0
$F = \{q_2\}$

$L = \{00, 01, 10, 11\}$

eg:- 00 ✓ $q_0\ q_1\ q_2$
0 X $q_0\ q_1$

Machine in Final state so, String (0,0) is accepted.

Machine not in Final state, so string is not accepted.

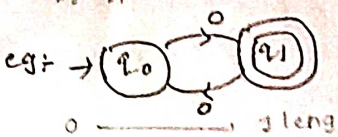
000 X

eg:-

$L = \{$ all binary strings of length 3$\}$
000, 001,

000
q_0 q_1 q_2 q_3

eg:-



$L = \{0, 000, 00000 \ldots\ldots\}$

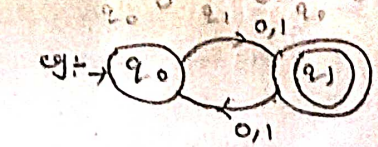
0 ——— 1 length
0 0 0 → 3 length
0 0 0 0 0

even length strings

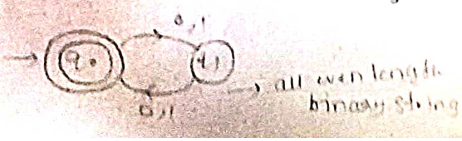
q_0 0 q_1 ✓

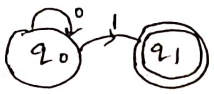
q_0 0 q_1 0,1 q_0 0 q_1 ✓

eg:-



$L = \{$ all odd length binary strings$\}$

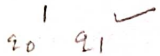




Any no of zeroes followed by 1

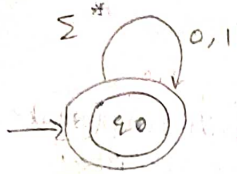
$L = \{1, 01, 001, 0001, \ldots\}$

Infinite number of times we can take

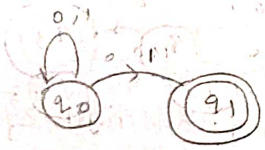


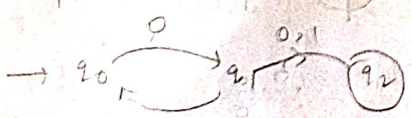
finite
↳ because no of states

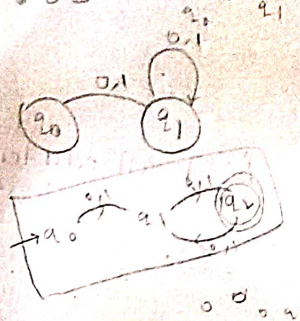
① construct a finite automata that recognizes all the strings over sigma.



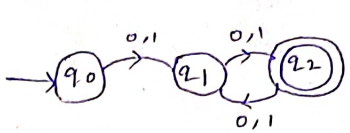
② construct an automata which recognizes even length binary strings.



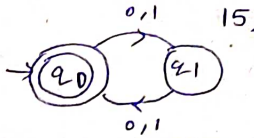




15/09/2023.

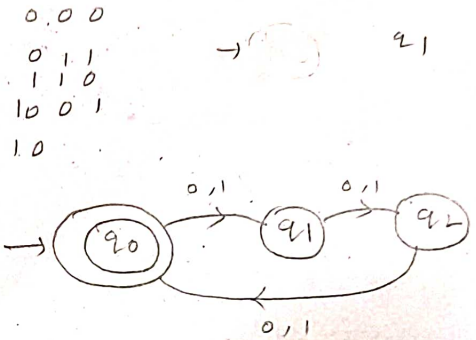


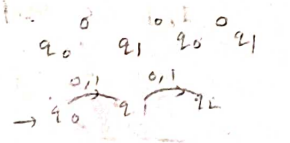
Excluding ϵ

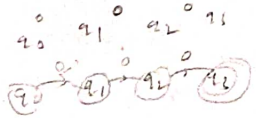


Including ϵ (epsilon)

③ construct the automata that accepts all the binary strings whose length is divisible by 3.

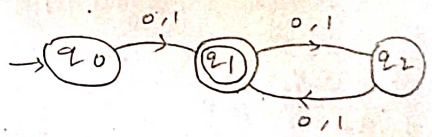




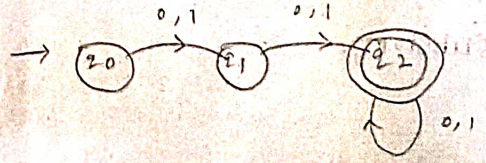


4) whose length is 1%3

1, 4, 7, 10

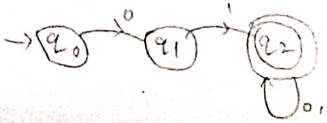


5) 2%3

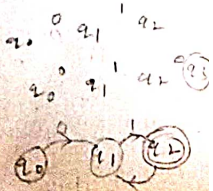


when divide by 3

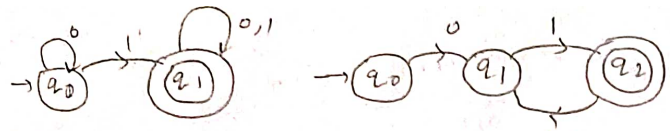
6) construct the automata whose prefix is 0,1



L = {01
010
011
0100
0111



which is part of language not part of automata

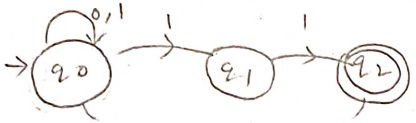


If 1
accepted
but wrong

0 1 0 x not accepting

7

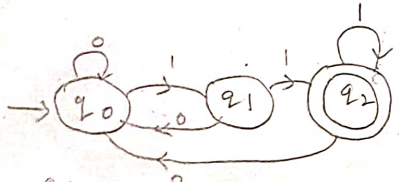
1, 1 as suffix.



NFA Not deterministic
If atleast one path is leading to the acceptance state we can take

0 1 1
1 1 1
0 0 1 1
0 1 1 1

	0	1
q_0	q_1	q_2, —
q_0	q_1	q_2, q_0



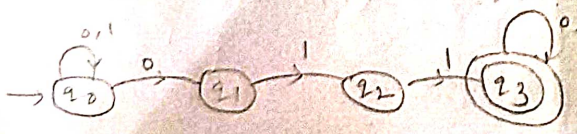
Implementation of DFA than NFA

It is Deterministic one

Answer වෙන්න String final state & Reach වෙන්නම ඕනේ

8)

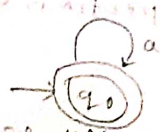
0 1 1 as substring.



9)

$L = \{ a^m \mid m \geq 0 \}$

$a^0 = \epsilon$ $a^1 = a$ $a^2 = a\,a$ $a^3 = a\,a\,a$



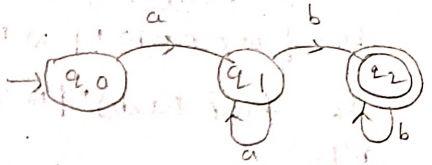
If different patterns show many states

10)

$L_2 = \{ a^m \mid m > 0 \}$

$L_3 = \{ a^m b^n \mid$

eg:-
First datatype
second variable.
any no of a's and
any no of b's



$L_3 = \{ ab, aab, abb, aabb \ldots \}$

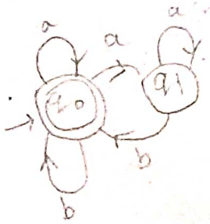
n+1 = states

divisible by 3 ⇒ n states
In other cases = n+1

$L_4 = \{ a^m b^n \mid m, n \geq 0 \}$

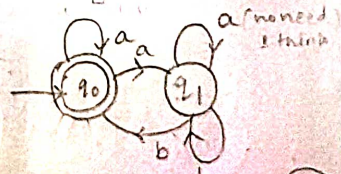
$L_5 = \{ a^m b^n \mid m \geq 0, n > 0 \}$

$L_6 = \{ a^m b^n \mid m > 0, n \geq 0 \}$.

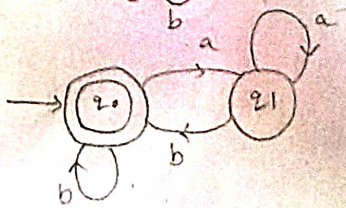


$L_5 = \{ \epsilon, a, ab, aab, abb \ldots \}$

$a^0 = \epsilon$ $\epsilon, ab, aabb, abb$



L_6



21/09/2023

$L_1 = \{ a^m \mid m \geq 0 \}$

$L_2 = \{ a^m \mid m \geq 1 \}$

$L_3 = \{ a^m \mid m$ is divisible by $2 \}$

$L_4 = \{ w \in (a,b)^* \mid w$ has a's divisible by $2 \}$

$L_5 = \{ w \in (a,b)^* \mid w$ has b's divisible by $3 \}$

$L_6 = \{ w \in (a,b)^* \mid w$ starts with 'a' and ends with 'b' $\}$

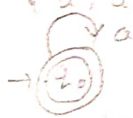
$L_7 = \{ a^m b^n \mid m \geq 1, n \geq 1 \}$

$L_8 = \{ w \in (a,b)^* \mid w$ starts with 'a' and ends with 'a' $\}$

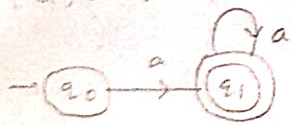
$L_9 = \{ w \in (a,b)^* \mid w$ starting and ending symbols are same $\}$

$L_{10} = \{ w \in (a,b)^* \mid w$ starting and ending symbols are different $\}$

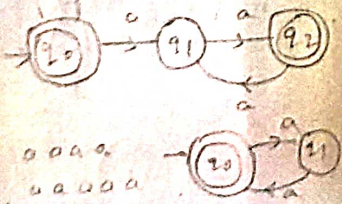
$L_1 = \{ \epsilon, a, aa, aaa \cdots \}$



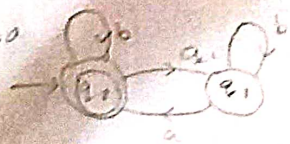
$L_2 = \{ a, aa, aaa \cdots \}$



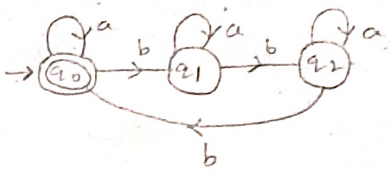
$L_3 = \{ \epsilon, aa, aaaa \cdots \}$



$L_4 = \{ \epsilon, aab, baa, bbaa, aaaaa \cdots \}$

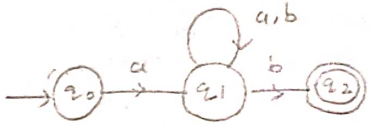


$L_5 = \{ \epsilon, bbb, abbba, abbbaa \cdots \}$



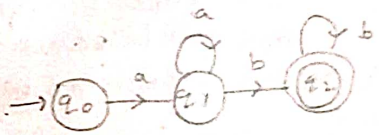
a
bbbbbb
bb
aabbb
bbaab

$L_6 = \{$



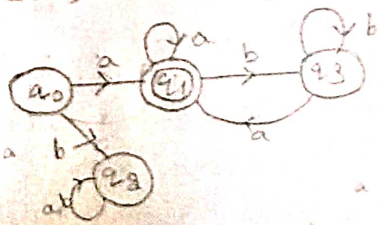
ab
aabb
aaab
abbb
abab

$L_7 = \{ ab, aab, abb, aaab, abbb \cdots \}$



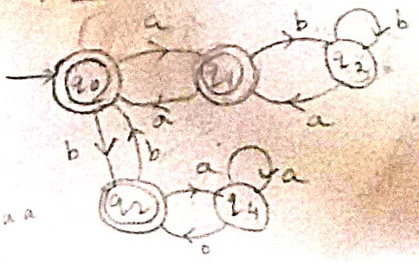
ab
aa

$L_8 = \{ aa, aaa, aaaa \cdots \}$



aa
aba
abaa
abba
abb
abbaba

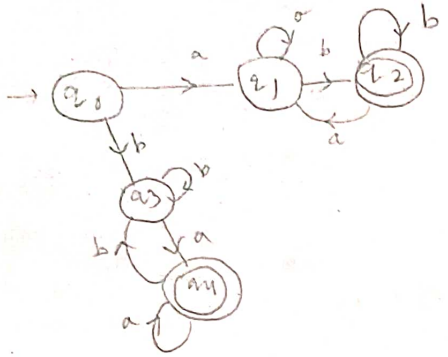
$L_9 = \{ a, b, aa, bb, aba, bab, \cdots \}$



bab
aabb
abba
baab
baba

L(G): { ab, abb, abbb
aab, aaab
ba, baa, baba, baaabaa

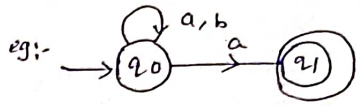
baba
bababab
ababa



Non-deterministic Finite Automata (NFA): 21/09/2023 Thursday.

A finite automata in which from every state on any input symbol there exists zero or any number of transitions is said to be NFA.

eg:-



next
a q0 or q1

which path to take: Non-determinism.

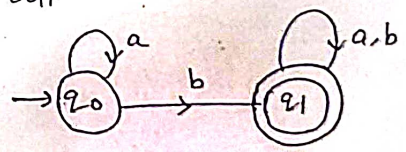
Transition table

	a	b
initial → q0	q0, q1	q0
* q1 final	–	–

If in table more than one state or (–) are there then it is Non-deterministic Finite Automata (NFA):

(DFA)
Deterministic Finite Automata: A finite automata in which from every state on every input symbol there exist exactly one transition is said to be deterministic finite automata or DFA.

⇒ The table will exactly contain one state in each cell.

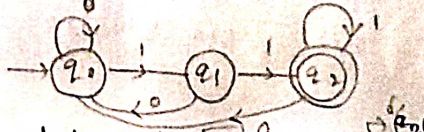


	a	b
→ q0	q0	q1
* q1	q1	q1

⇒ Both NFA and DFA are equivalent.

1) construct a DFA that accepts all the binary strings having suffix 11?

L = { 11, 011, 111, 0011 ... }

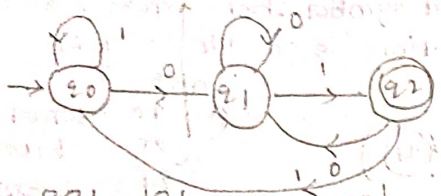


δ(q0, 0) = q0
δ(q0, 1) = q1
δ(q1, 0) = q0
δ(q1, 1) = q2
δ(q2, 0) = q0
δ(q2, 1) = q2

	0	1
→ q0		q1
q1		q2
* q2		

2)

Suffix 01



L = {01, 001, 101, 0001, 1100}

	0	1
→ q0	q1	
q1		q2
q2		q2

δ(q0, 01)

01

δ(q1, 0)

001

δ(q2, 0)

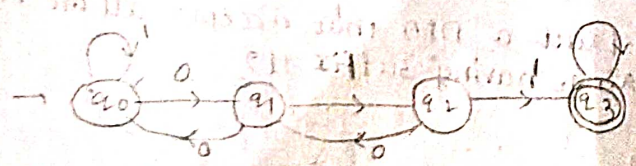
0101

δ(q2, 1)

0111

3)

011 substring.



L = {011, 1011, 0110, 0111, 0001101, 1001101}

δ(q0, 0) 011

δ(q0, 1) 1011

	0	1
→ q0	q1	
q1		q2
q2	q3	q3
q3		

construct a DFA that accepts all the binary strings that have 01 as prefix.

δ(q0, 1) reject

δ(q1, 0) reject

1

00×

δ(q2, 0)

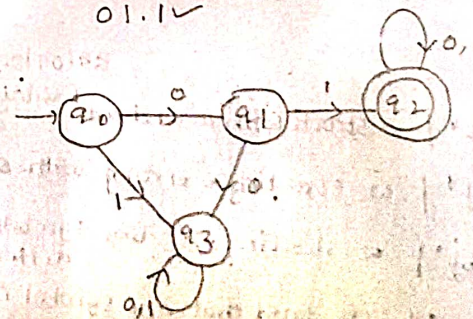
010✓

δ(q2, 1)

011✓

	0	1
→ q0	q1	q3
q1	q3	q2
q2	q2	q2
q3	q3	q3

q3 → Dead state



001 suffix

	0	0	1
→ q_0	q_1		
q_1	q_2		
q_2	q_3		
q_3			

$\delta(q_0, 0)$: $\delta(q_0, 0)$ $\delta(q_1, 0)$ $\delta(q_2, 1)$

$\delta(q_0, 1)$: 0 0 0 1

$\delta(q_0, 0)$: 0 0 (0 1)

$\delta(q_1, 0)$: 0 0 1

$\delta(q_0, 1)$: $\delta(q_1, 1)$ 0 1

$\delta(q_0, 0)$: 0 1 (0 0 1)

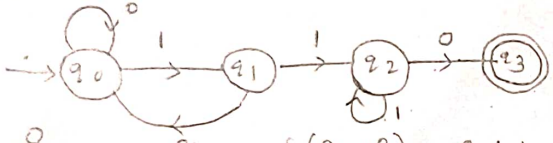
$\delta(q_0, 0)$ 1 $\delta(q_0, 0)$ $\delta(q_1, 1)$ $\delta(q_0, 1)$ 1 0 0 1

$\delta(q_0, 1)$ 0 0 0 1 0 0 1 1 $\delta(q_1, 1)$

0 1 0 0 1

1 0 0 1 1 0 0 1

110 suffix



	1	1	0
→ q_0	q_1		
q_1	q_2		
q_2	q_3		
q_3			

$\delta(q_0, 0)$ 0 1 1

$\delta(q_1, 0)$ 1 1 0 0 1 1 0

1 0 1 1 0

$\delta(q_2, 1)$

1 1 1 0

22/09/2023 Friday

$L_1 = \{ w \in \{a,b\}^* \mid w$ starts with 'a' and ends with 'a' $\}$

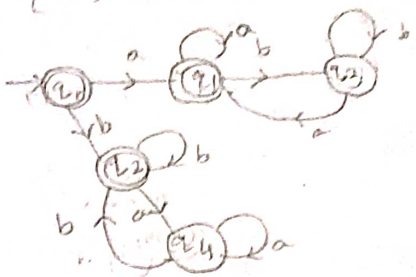
$L_2 = \{ w \in \{a,b\}^* \mid w$ starting & ending with same symbol $\}$

$L_3 = \{ w \in \{a,b\}^* \mid w$ starting & ending symbols are different $\}$

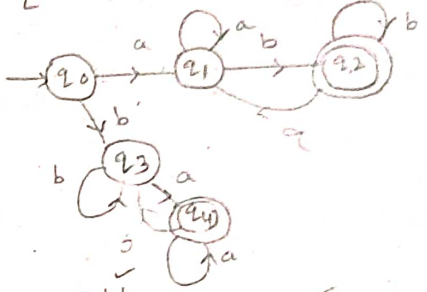
$L_4 = \{ w \in \{a,b\}^* \mid w$'s last but one symbol is 'b' (second symbol form right side is 'b') $\}$

$L_1 = \{ a, aa, aba, abba, aaa, aabaa \ldots \}$

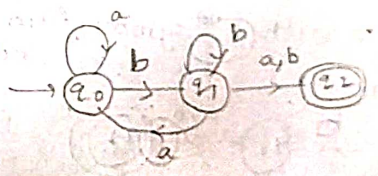
$L_2 = \{ \epsilon, aa, bb, aba, bab$



$L_3 = \{ ab, ba, abb, bba, abab, baba \ldots \}$



$L_4 = \{ ba, aba, aabb, aaabb$ bbbbb



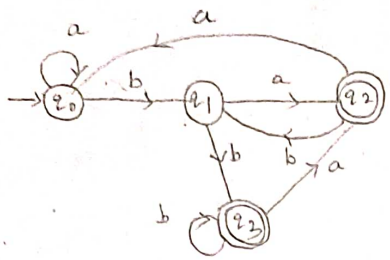
bab ✗
aba ✓
abba ✓

aaabb

baaaba

$L = \{bb, ba, bba, aba, abb, bbb \ldots\}$

23/09/2023
Monday.



b a a
b b a

	a	b
→q_0	q_0	q_1
q_1	q_2	q_3
q_2	q_0	q_1
*q_3	q_2	q_3

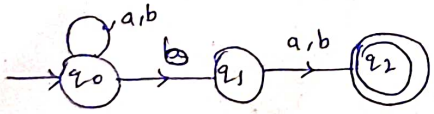
$\delta(q_0, a)$ — a ba
$\delta(q_0, b)$ — $\delta(q_2, a)$
$\delta(q_1, a)$ — b a a (b) ×
$\delta(q_1, b)$ — b a a (b) (a/b) ✓
$\delta(q_2, b)$ — b a b (a/b)
$\delta(q_3, a)$ — b b a (b) (a/b) so q_2
$\delta(q_3, b)$

last 3 → (8) 2³
last 4 → $2^4 = 16$

NFA to DFA conversion:

$L = \{w \in \{a,b\}^* \mid w$ with 2nd symbol from right hand side is 'b'$\}$

NFA



DFA.

NFA Table

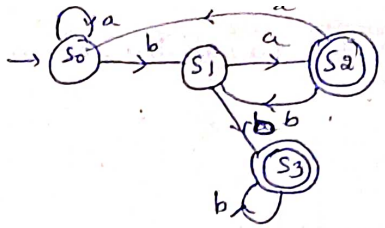
	a	b
→q_0	q_0	$q_0 q_1$
q_1	q_2	q_2
*q_2	–	–

DFA Table

	a	b
q_0	q_0	$[q_0 q_1]$
$[q_0 q_1]$	$[q_0 q_2]$	$[q_0 q_1 q_2]$
$[q_0 q_2]$	q_0	$[q_0 q_1]$
$[q_0 q_1 q_2]$	$[q_0 q_2]$	$[q_0 q_1 q_2]$

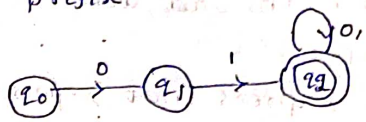
→ if here q_2 comes (q_3 q_2 dead) q_2 dead dead
Dead Dead Dead

Renaming $S_0 = [q_0]$
$S_1 = [q_0 q_1]$
$S_2 = [q_0 q_2]$
$S_3 = [q_0 q_1 q_2]$



	a	b
S_0	S_0	S_1
S_1	S_2	S_3
S_2	S_0	S_1
S_3	S_2	S_3

eg:- 0 1 prefix.



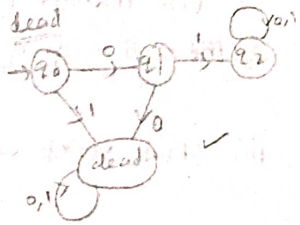
writing is the best Meditation practice.

NFA:

Table

	0	1
→q_0	q_1	dead
q_1	dead	q_2
*q_2	q_2	q_2
dead	dead	dead

	0	1
q_0	q_1	dead
q_1	dead	q_2
dead	dead	dead
q_2	q_2	q_2



⇒ Expressive power of NFA = Expressive power of DFA.
⇒ Every DFA is NFA
⇒ for a language if NFA is constructed then there exist DFA also
⇒ No of Languages are expressed by NFA can be expressed using DFA also.
⇒ Languages that are constructed by finite automata are called Regular languages.
⇒ Languages for which finite automata can construct are Regular languages.

Regular expression: A formal declarative representation (formal) is regular expression

Operators: * Kleene closure
• Concatenation
+ union

Σ
Input symbols & operators

$(a+b)^* = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab \ldots\}$

all the strings over a,b

$(ab)^2 = \{abab\}$

$(a+b)^2 = \{aa, ab, ba, bb\}$

$a(a+b)^* : \{a, aa, ab, aaa, abb, \ldots\}$

↓ all strings with prefix a.

① write a regular express that represents all the strings over a,b

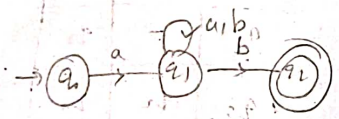
(if there exists a)

(i) that end with b

$(a+b)^* b$

(ii) that start with a and b.

$a(a+b)^* b$ $\quad a \in b$ $\quad a\ ab$ $\quad a\ bb$



(iii) Length is divisible by 2.

$((a+b)^2)^*$

or $(aa+ab+ba+bb)^*$

exactly 2

$(a+b)^2$

(iv) atmost 2

$(a+b+\epsilon)^2$ $\quad (a+b+\epsilon)(a+b+\epsilon)$

or

$= aa, ab, a, ba, bb, b$

binary strings.

① 01 prefix

$01(0+1)^*$

② suffix 11

$(0+1)^*11$

③ substring 011

$(0+1)^* 011 (0+1)^*$

④ 2nd last symbol is '0'

$(0+1)^* 0(0+1)$

⑤ 2nd symbol from starting is '0'

$(0+1)\,0\,(0+1)^*$

Regular language

Finite Automata ⟺ Regular expression

⑥ the strings over a,b that do not contain consecutive b's.

ε, a, b, ab, ba, aba

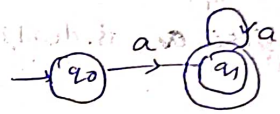
$(ab+ba)^*$ — I think

Conversion of finite automata to regular expression

1) Arden's Method
2) State elimination Method.

$R = Q + RP$

then solution is $R = QP^*$



Characteristic equation for every state..

here (edges)

$q_0 = \epsilon$

$q_1 = q_0 a + q_1 a + q_1 b$

$q_1 = q_0 a + q_1 (a+b)$

Substitute $q_0 = \epsilon$

$q_1 = \epsilon(a) + q_1(a+b)$

$q_1 = a + q_1(a+b)$

is of the form $R = Q + RP$.

$\boxed{q_1 = a(a+b)^*}$

Regular expression is not commutative

$ab \neq ba$

$q_0 = \epsilon + q_1 a$

$q_1 = q_0 a + q_1 b$ —— (2)

Substitute q_0 in (1)

$q_1 = (q_1 a + \epsilon)a + q_1 b$

$= \epsilon + q_1 aa + q_1 b$

$q_1 = a + q_1(aa+b)$

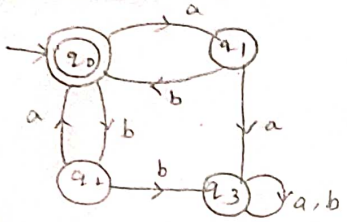
$q_1 = a(aa+b)^*$

$\boxed{q_0 = \epsilon + a(aa+b)^* a}$

1) Using Arden's method, find the equivalent expression.

07/04/2023 Wednesday.

i)



$q_0 = \epsilon + q_1 b + q_2 a$ — ①

$q_1 = q_0 a$ — ②

$q_2 = q_0 b$ — ③

$q_3 = q_2 b + q_1 a + q_3 (a+b)$ — ④

Here q_3 is dead state.

Substituting ② and ③ in ①

$q_0 = \epsilon + q_0 ab + q_0 ba$

$q_0 = \epsilon + q_0 (ab + ba)$

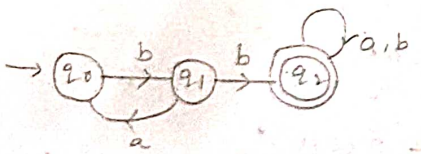
$R = Q + RP$

$R = QP^*$

$q_0 = \epsilon (ab+ba)^*$

$\therefore$ $q_0 = (ab+ba)^*$

ii)



$q_0 = \epsilon + q_1 a$ — ①

$q_1 = q_0 b$ — ②

$q_2 = q_1 b + q_2 (a+b)$ — ③

Sub

$q_1 = (ba)^* b$

$q_2 = (ba)^* bb + q_2 (a+b)$

$R = Q + RP$

$q_2 = (ba)^* bb (a+b)^*$

② in ①

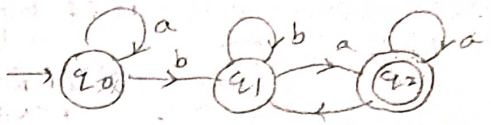
$q_0 = \epsilon + q_0 ba$

$q_0 = \epsilon + q_0 ba$

$q_0 = \epsilon (ba)^*$

$q_0 = (ba)^*$ — ④

iii)



$q_0 = \epsilon + q_0 a$ — ①

$q_1 = q_0 b + q_1 b + q_2 b$ — ②

$q_2 = q_1 a + q_2 a$ — ③

$\frac{q_0}{R} = \frac{\epsilon}{Q} + \frac{q_0}{R}\frac{a}{P}$

$q_0 = \epsilon (a)^*$

$q_0 = (a)^*$

Sub in ②

$q_1 = (a)^* b + q_1 b + q_2 b$

$\frac{q_1}{R} = \frac{(a)^* b + q_2 b}{Q} + \frac{q_1 b}{R}\frac{}{P}$

$q_1 = (a^* b + q_2 b)(b)^*$

Sub in ③

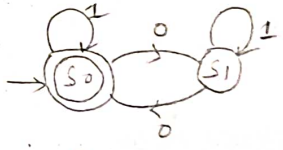
$q_2 = (a^* b + q_2 b)(b)^* a + q_2 a$

$= a^* b b^* a + q_2 b b^* a + q_2 a$

$= a^* b b^* a + q_2 (bb^* a + a)$

$q_2 = a^* b b^* a (bb^* a + \ldots$

iv)



$S_0 = \epsilon + S_1 0 + S_0 1$ — ①

$\frac{S_1}{R} = \frac{S_1 1 + S_0 0}{RP \quad Q}$ — ②

$R = QP^*$

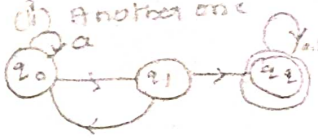
$S_1 = S_0 1 0^*$ Sub in ① we get

$S_0 = \epsilon + S_0 10^* 0 + S_0 1$

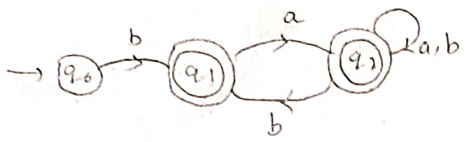
$= \epsilon + S_0 (10^* 0 + 1)$

$S_0 = \epsilon (10^* 0 + 1)^*$

(i) Another one



(iv)



$q_0 = \epsilon$ ——— ①

$q_1 = q_0 b + q_2 b$ ——— ②

$q_2 = q_1 a + q_2 (a+b)$ ——— ③

① in ②

$q_1 = \epsilon b + q_2 b$

$q_1 = b + q_2 b$

q_1 in q_2

$q_2 = (b + q_2 b) a + q_2 (a+b)$

$q_2 = ba + q_2 ba + q_2 (a+b)$

$\underset{R}{q_2} = \underset{Q}{ba} + \underset{R}{q_2}\underset{P}{(ba + (a+b))}$

$q_2 = ba\,(ba + (a+b))^*$

$q_1 = b + (ba\,(ba + (a+b))^*)\,b$

$\therefore q_1 + q_2 = b + (ba\,(ba + (a+b))^*)\,b + ba\,(ba + (a+b))^*$

1) Regular expression satisfies commutative property over union.

(i) a) $r_1 + r_2 = r_2 + r_1$

b) $r_1 r_2 \neq r_2 r_1$

$ab \neq ba$

(ii) a) $(r_1 + r_2) + r_3 = r_1 + (r_2 + r_3)$

b) $(r_1 r_2) r_3 = r_1 (r_2 r_3)$

1) If one variable is there

2) If no substitution possible then apply lemma

(iii) distribution of concatenation over union

(a) $r_1 \cdot (r_2 + r_3) = r_1 r_2 + r_1 r_3$ (right distribution)

(b) $r_1 + (r_2 r_3) \neq (r_1 + r_2)(r_1 + r_3)$

Left distribution $(r_1 + r_2) r_3 = r_1 r_3 + r_2 r_3$

(iv) Identity:

$r_1 + X = r_1 \quad \forall\ r_1 \in$ Regular expression

X is constant.

annihilator $\overset{a}{X} = \phi$

$r_1 + \phi = r_1$

$r_1 \cdot X = r_1$

$r_1 \cdot \epsilon = r_1, \quad \epsilon \cdot r_1 = r_1$

(v) Annihilator: $r_1 + X = X$

no annihilator over union

$r_1 \cdot X = X$

$r_1 \cdot \phi = \phi$

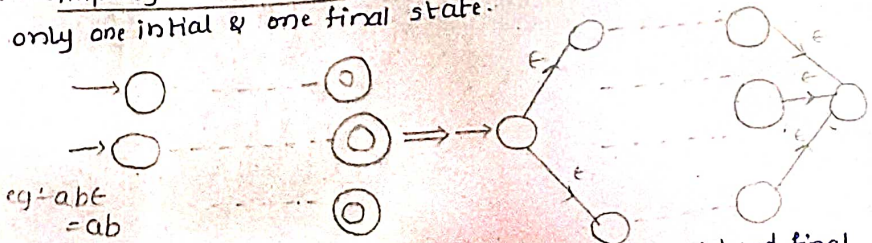
$X = \phi$

04/10/2023 Wednesday

F.A to R.E.

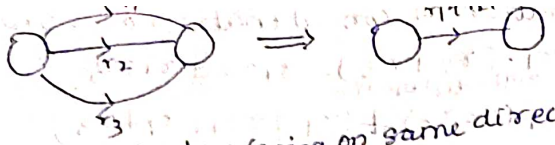
Algorithm steps. → multiple initial & final states are there

(i) simplify transition graph in such a way that it has only one initial & one final state.



(ii) simplify in such a way. It has different initial and final states.

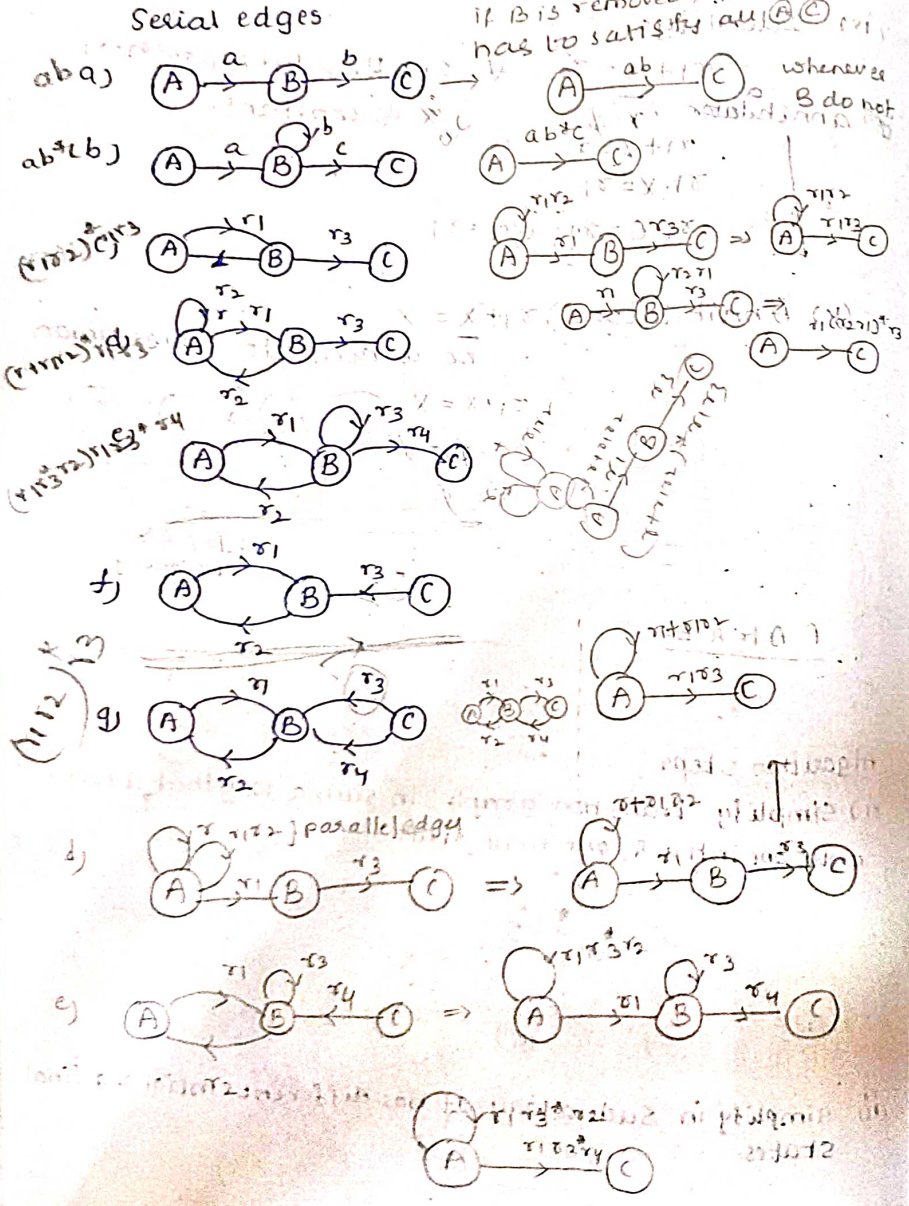
(iii)

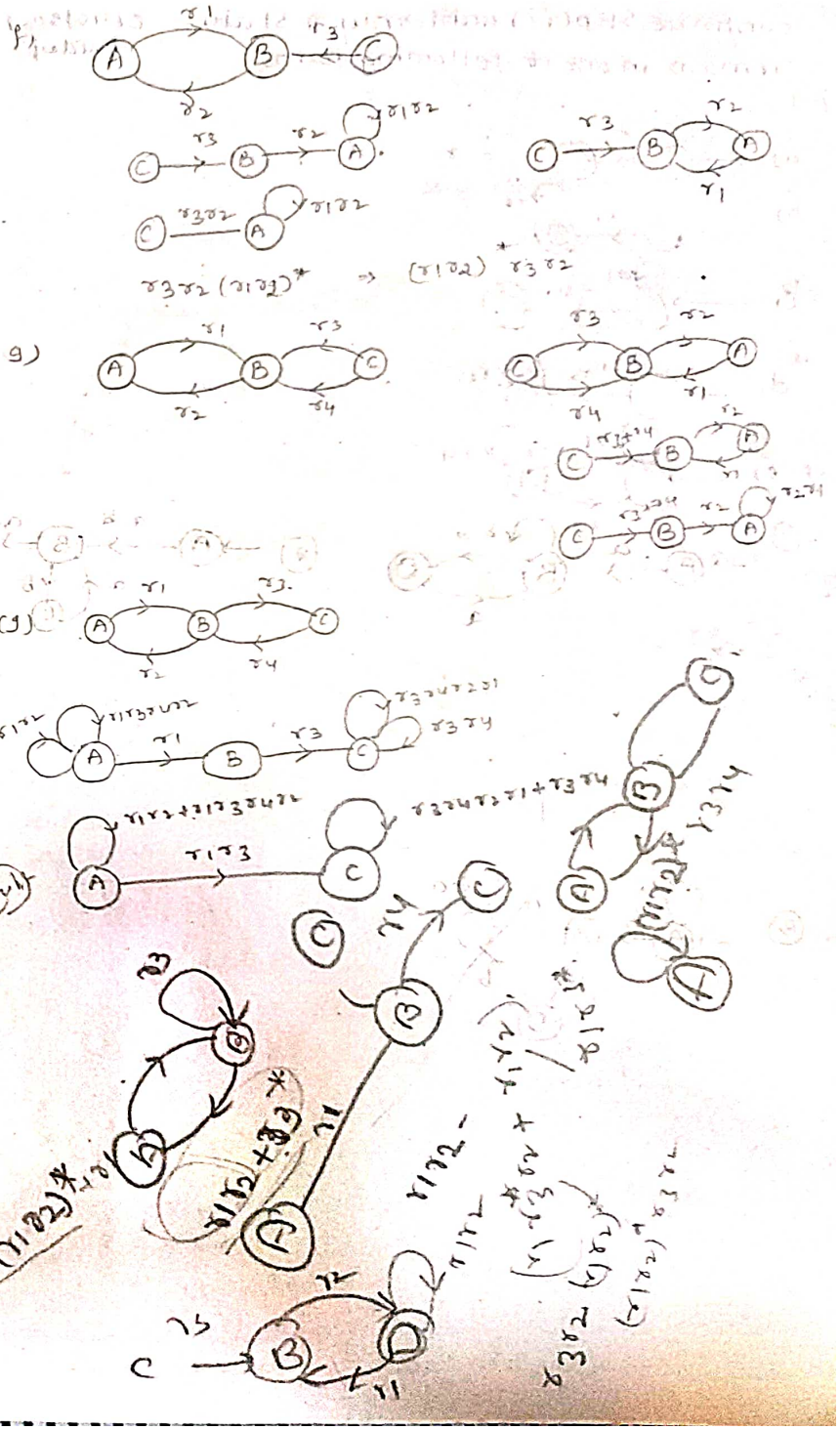


parallel edges (going on same direction)

(iv) State elimination:

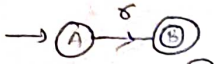
Serial edges

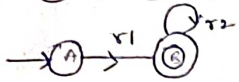


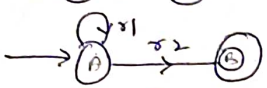


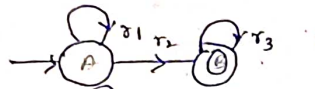
continue Step(iv) until only 2 states remain in one of following forms.

(v)

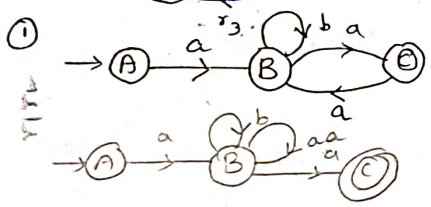
a)  = r

b)  = $r_1 r_2^*$

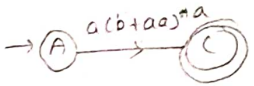
c)  = $r_1^* r_2$

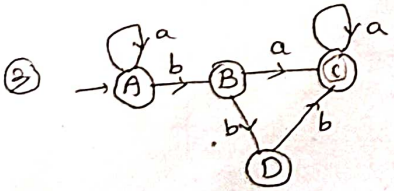
d)  = $r_1^* r_2 r_3^*$

e)  = $(r_1 + r_2 r_4^* r_3)^* r_2 r_4^*$

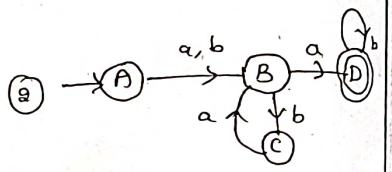
1) 

$a(b+aa)^* a$



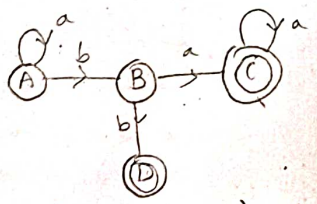
2) 

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

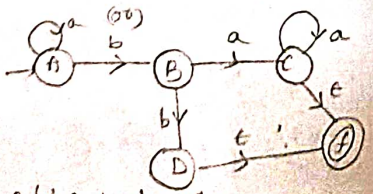
2) 

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

$(a+b)^* = \{ a^* \cup b^* \cup a^* b^* \cup b^* a^* \}$



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



$abb\epsilon + aba a^* \epsilon$

$ab(bf + aa^* \epsilon)$

$ab(b+aa^*)$

13/10/2023 Friday

R·E → F·A

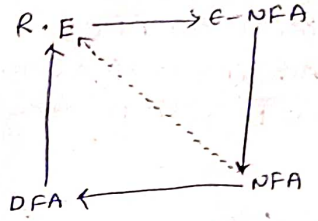
regular expression → finite automata

State Creation Method:

→ If epsilon is there → epsilon NFA.

If not → NFA.

→ A language for which Regular expression and finite automata exists.



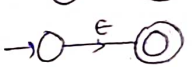
Theoritically there exists a finite automata for every R·E.

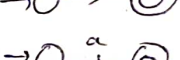
Equivalent blw R·E & F·A.

basic R·E

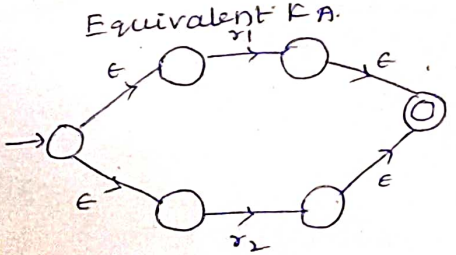
F·A

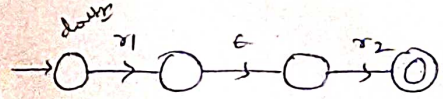
$r = \phi$ 

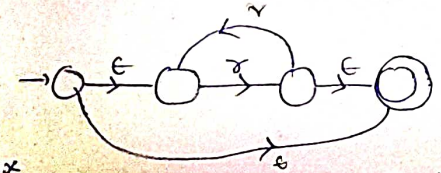
$r = \epsilon$ 

$r = a$ 

R·E with operators

$r_1 + r_2$ 

$r_1 r_2$ 

r^* 

For instance complex expression can be divided and can be write in simple $(a^* b + ba)^*$.

complexity

ε-NFA : more states, more paths } than NFA.

Simulation ε-NFA is difficult.

Q: why not ε-NFA to DFA?

⇒ Transitions, final states different.

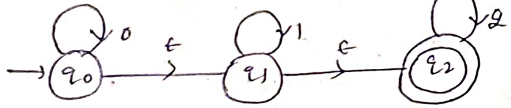
⇒ we from a state by reading ε if reach final state then that state is also final.

E-NFA:

NFA having ε-transitions.

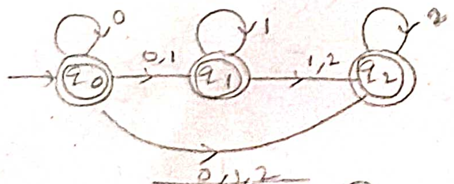
conversion of ε-NFA to NFA.

ε-closure



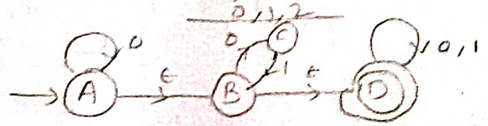
	0	1	2
→ q_0	$q_0 q_1 q_2$	$q_1 q_2$	q_2
q_1	–	$q_1 q_2$	q_2
q_2	–	–	q_2

q_0 0 చదివాంటే ε use చేసినా వెళ్ళ states వెళ్తుంటే ఆ states కూడా

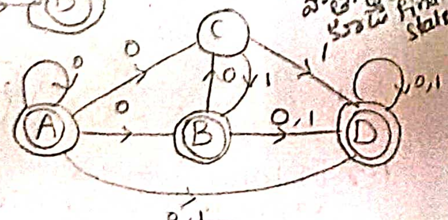


doubt {final states how? we decide?

eg:



	0	1
→ A	ABCD	D
B	CD	D
C	–	BD
D	D	D



Epsilon read చేసినాం ఉంటే final states వెళ్తే ఆ state కూడా final state

Final states A, B, D.

ε-closure(q) = set of all states that are reachable from q on reading ε.

First 2 eg: ε-closure(A) = {A, B, D}

1 eg: ε-closure(q_0) = {q_0, q_1, q_2}.

eg: $\delta'(A,0) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(A), 0))$

$\delta(A,0)$

ε-closure(δ((A, B, D), 0))

ε-closure(δ(A,0), δ(B,0), δ(D,0))

A on 0 = A, B on 0 = C, D on 0 = D

ε-closure (A ∪ C ∪ D)

ε-closure (A, C, D)

ε-closure(A) ∪ ε-closure(C) ∪ ε-closure(D)

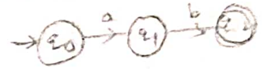
= {A, B, D} ∪ {C} ∪ {D}

= {A, B, C, D}.

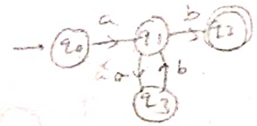
18/10/2023
wednesday.

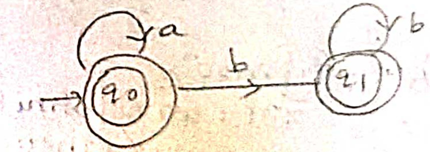
Non-regular Language: A language for which finite automata does not exist.

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



$L_1 = \{ \epsilon, a, b, ab, aab, abb$

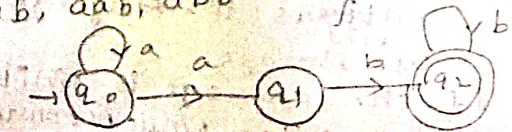




aaabbb.

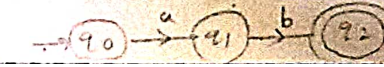
$L_2 = \{ a^m b^n \mid m, n \geq 1 \}$

= { ab, aab, abb … }



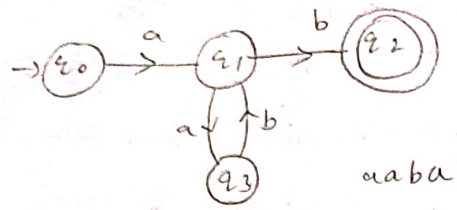
$L_3 = \{ a^m b^m \mid m > 0 \}$

{ ab, aabb, aaabbb …



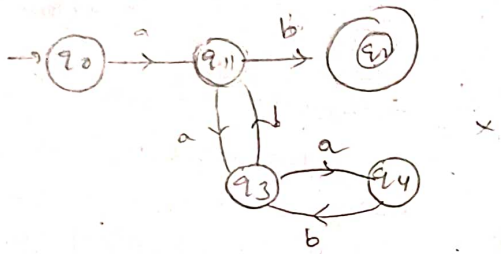
$L_3 = \{ a^m b^m \mid 0 < m \leq 3 \}$

$m \leq 2$



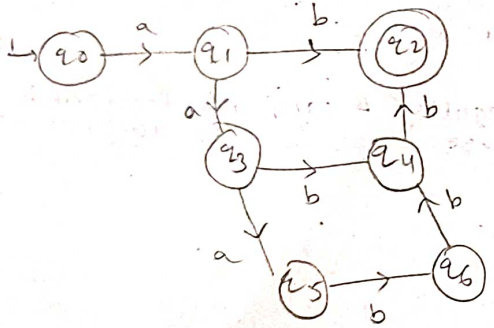
+

aaba ✗



×

2n+1 states.



eg:-

$L_4 = \{ a^m b^m \mid m > 0 \}$

here no of states are not finite.

∴ non-regular language

⇒ we cannot construct finite automata for patterns (if strings are infinite) and use need storage.

⇒ If the language is finite (finite no of strings) then finite Automata exists.

Note: For every finite language there will be a FA exist ∴ then Regular

converse: If infinite language is there, it may or may not be regular.

⇒ It is becoming Regular because of pattern repetition.

⇒ If memory requirement is there then it cannot be regular.

$L_5 = \{ a^m b^n \mid m < n \}$ → Non-Regular

So, here we need store m value to compare with n.

push-down Automata:- context free language.

Stack ∴ 1 of one stack.

$L_6 = \{ a^m b^n \mid m > n \}$ ⇒ N.R

memory needed

$L_7 = \{ a^p \mid p \text{ is prime} \}$ ⇒ N.R.

$L_{7'} = \{ a^{2n} \mid n > 0 \}$ → Regular

$L_8 = \{ a^{3n} \mid n > 0 \}$ → Regular

No of a's divisible by 3.

$L_{8'} = \{ a^{3n+1} \mid n > 0 \}$ → 1%3

1 modulo 3

$L_9 = \{ a^{3n+2} \mid n > 0 \}$ → 2 modulo 3

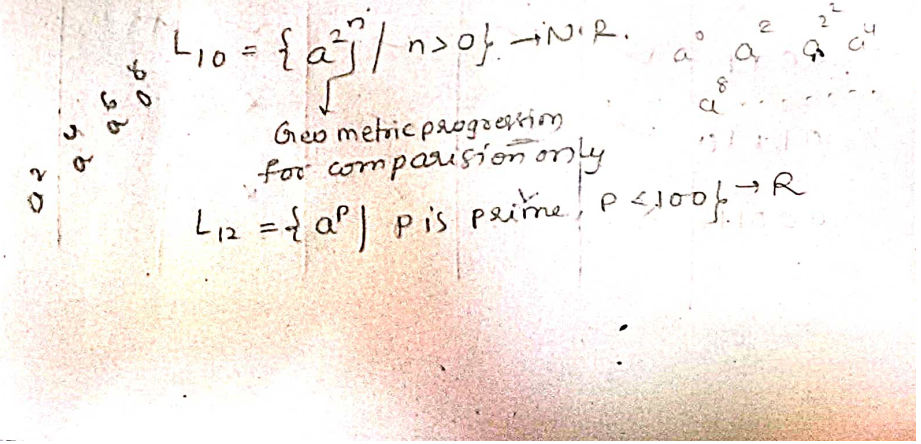
⇒ R

$L_{10} = \{ a^{2^n} \mid n > 0 \}$ → N.R.

Geometric progression for comparision only

$L_{12} = \{ a^p \mid p \text{ is prime}, p < 100 \}$ → R

$L_7 = \{a, aaa, aaaaa, \ldots\}$



Imp | on regular languages are used. are guaranteed to produce RL.

closure properties of R.L.s.

19/10/2023 Thursday.

closure refers to some operation on a language } all are closed

that is of same "type" as originally operated on i.e. regular. Regular languages are closed under following operations.

1) Union
2) Intersection
3) Concatenation
4) Kleene closure
5) +ve closure.
6) Difference.
7) Symmetric difference.
8) complement.
9) Reverse.
10) prefix.
11) Quotient.
12) Substitution.
13) Homomorphism.
14) Inverse Homomorphism.

NOTE: Regular languages are closed under all operators beside.

1) Regular languages are closed under union.

Good!!

L_1 L_2
R_1 $R_2 \Rightarrow R_1 + R_2$ ✓

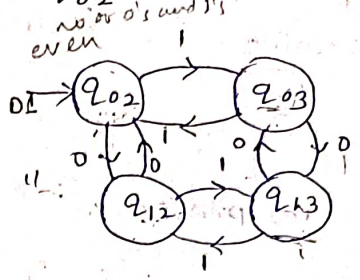
L_1: binary strings having even no of 0's

L_2: " " " " " " 1's

	L_1	L_2	$L_1 \cup L_2$	$L_1 \cap L_2$	$L_1 - L_2$	$L_2 - L_1$
q_{12} 0 0 0 1 1	✗	✓	✓	✗	✗	✓
q_{02} 0 0 1 1	✓	✓	✓	✓	✓	✗
q_{03} 0 1 1 1 0	✓	✗	✓	✗	✓	✗
q_{13} 0 0 0 1 1 1	✗	✗	✗	✗	✗	✗

combinations.

$q_{02} \Rightarrow q_0 q_2$ q_{03} q_{12} q_{13}



Final states: $q_0 \rightarrow L_1$

$q_2 \rightarrow L_2$

My final states are which are having either (q_0 q_2)

Final states: $L_1 \cup L_2$: q_{02}, q_{03}, q_{12} (final state)

$L_1 \cap L_2$: q_{02} (both have to be there, final state)

⇒ The state is final if both the final states included.

$L_1 - L_2$: L_1 but not of L_2.

q_{02} ✗ because q_2 also is there.

q_{03}

⇒ only final state of machine 01 must be there not L_2.

$L_2 - L_1$: q_{12}.

$(L_1 \cup L_2) - (L_1 \cap L_2) = q_{12}, q_{03}$.

q_{02} on 0 does q_0 on 0 and q_2 on 0.

q_0 on $0 \Rightarrow q_1$

q_2 on $0 \Rightarrow q_2$

q_{12}

q_{02} on 1

q_0 on $1 = q_0$

q_2 on $1 = q_3$

q_{03}.

q_{03} on 1

q_0 on $1 = q_0$

q_3 on $1 = q_2$

q_{03} on 0

q_0 on $0 = q_1$

q_3 on $0 = q_3$

q_{12} on 0.

q_1 on $0 \Rightarrow q_0$

q_2 on $0 \Rightarrow q_2$

q_{12} on 1

q_1 on $1 \Rightarrow q_1$

q_2 on $1 \Rightarrow q_3$

q_{13} on $0 \Rightarrow$

q_1 on $0 \Rightarrow q_0$

q_3 on $0 \Rightarrow q_2$

q_{13} on $1 \Rightarrow$

q_1 on $1 \Rightarrow q_1$

q_3 on $1 \Rightarrow q_2$

3) concatenation

$L_1 L_2$ & $\rightarrow$ R·E

L_1 followed by L_2 & $R_1 R_2$

$R_1 R_2$.

L^* & $\rightarrow R^*$

4) +ve closure

$L^+ \rightarrow R^+$

excluding ϵ

$L_1, L_1 \ldots L_\infty$

closure including ϵ.

$L_0, L_1 \ldots L_\infty$

8) complement:

$L^C = \Sigma^* - L$

universal set

Final $\longleftrightarrow$ Non-final

Including dead state

DFA & వాడుకలో applicable.

Reversing the Roles of F and NF

10) prefix:

11 0

1 11 110

accept all the prefixes of the string.

excluding dead state.

except dead state all other are accepted as final.

9) Revese: directions should be revessed.

Initial $\longleftrightarrow$ final

$L \rightarrow L^R$

Minimization of DFA:

02/11/2023
Thursday.
mi Bed
class.

	0	1
→ A	F	B
B	C	G
* C	C	A
D	G	C
E	F	H
F	G	C
G	E	G
H	C	G

① $S_0 = \{A, B, C, D, E, F, G, H\}$

Differentiate between final and non-final states.

level 1: {C} {A, B, D, E, F, G, H}

on '0'

level 2: {C}, {B, H} {A, D, E, F, G}

on '1'

level 3: {C} {B, H} {D, F} {A, E, G}

A E G on '0'

level 4: {C} {B, H} {D, F} {A, E} {G}

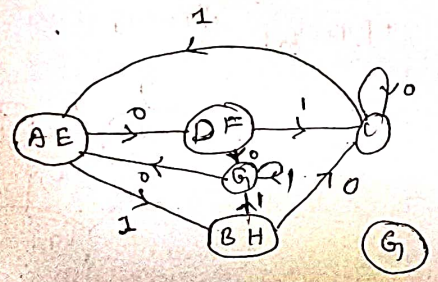
final, Initial

① ② ③ ④ ⑤

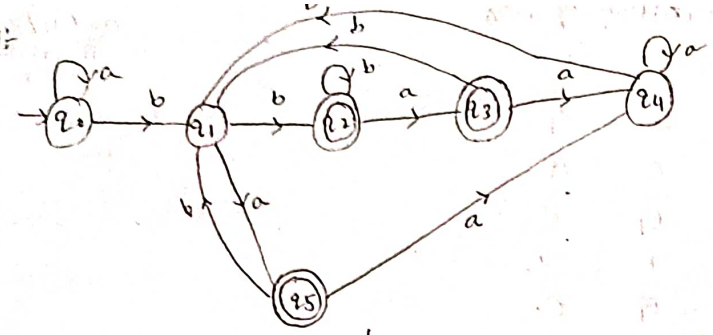
Which state value is initial that group is initial state

which state value is final state that group final.

	0	1
→[A E]	[D, F]	[B H]
[B H]	[C]	[G]
*[C]	[C]	[A, E]
[D F]	[G]	[C]
[G]	[A E]	[G]



eg:-



	a	b
→ q_0	q_0	q_1
q_1	q_5	q_2
* q_2	q_3	q_2
* q_3	q_4	q_1
q_4	q_4	q_1
* q_5	q_4	q_1

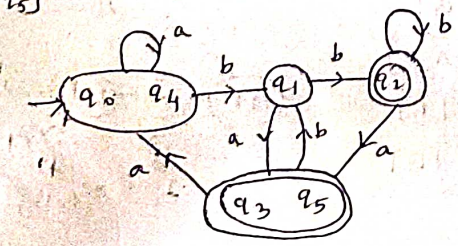
$S_0 = \{ q_0, q_1, q_2, q_3, q_4, q_5 \}$

level 1: $\langle q_2, q_3, q_5 \rangle \{ q_0, q_1, q_4 \}$

level 2: $\{ q_2 \} \{ q_3\ q_5 \} \{ q_0\ q_4 \} \{ q_1 \}$

Minimized DFA for the given example.

	a	b
→ $[q_0\ q_4]$	$[q_0\ q_4]$	$[q_1]$
$[q_1]$	$[q_3\ q_5]$	$[q_2]$
* $[q_2]$	$[q_3\ q_5]$	$[q_2]$
* $[q_3\ q_5]$	$[q_0\ q_4]$	$[q_1]$



Context Free Grammars (CFG). 03/11/2022 Friday.

Remember
Recall

$G(N, T, P, S)$ — starting Non-Terminal.

Non-Terminal, Terminals, productions (Rules).

(In other text book)

(V, Σ, P, S)

Non-terminal, Terminal

P: $\alpha \longrightarrow \beta$ → production formal

CFG: α is a single N.T.

β is combination of Ts & NTs

$(T + NT)^*$

here capital letter

Grammar: generator
Automata: Recognizer

Terminals: which have termination, do not have further substitution → left side ever ಆಗಲ್ಲ.

Non-Terminals: have further substitution.

Grammar: → set of productions
starting Non-Terminal → then substitution
↓
if all the symbols are terminals
↓
then done.

eg:-

$A \longrightarrow aAb \mid ab$

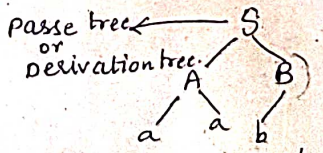
$S \longrightarrow AB$

$A \longrightarrow aa / bb$ =: $\{ A \longrightarrow aa \quad A \longrightarrow bb$

or

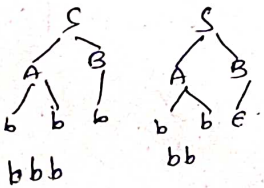
$B \longrightarrow b / \epsilon$

Passe tree or Derivation tree:



String is: aab

aa



bbb

bb

{ aab, aa, bbb, bb }

Derivation:

$S \longrightarrow AB$
aaB
aab
LMD

LMD: Left Most Derivation ↓ replacing leftmost non-terminal symbol by RHS of corresponding production rule.

RMD: Right Most Derivation.

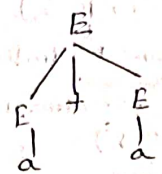
$S \longrightarrow AB$
Ab
aab
RMD

eg:- $S \longrightarrow aSb$

G1: ({E}, {a, +, *}, P, E)

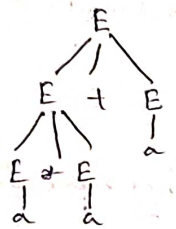
P: E → E+E | E*E | a

binary arithmetic expression

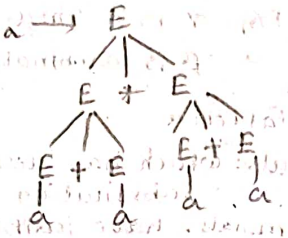


aa+ ✗
+aa ✗

a+a

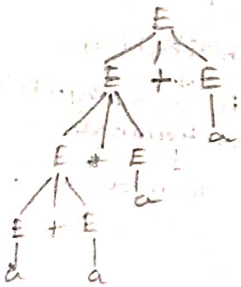


a+a*a+a → a*a+a



More than one Tree: Ambiguous grammar

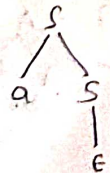
04/11/2023 Saturday



→ any number of a's including ϵ

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

$S \rightarrow aS \mid \epsilon$



2) $L_2 = \{a^n \mid n > 0\}$

$S \rightarrow aS \mid a$

3) $L_3 = \{a^n \mid n > 1\}$

$S \rightarrow aS \mid aa$

$S \rightarrow aS$

aaa

2 substitutions for aaa.

4) $L_4 = \{a^{2n} \mid n \geq 0\}$

$S \rightarrow aaS \mid \epsilon$

5) $L_5 = \{a^{2n} \mid n > 0\}$

$S \rightarrow aaS \mid aa$

6) $L_6 = \{a^m b^n \mid m, n \geq 0\}$

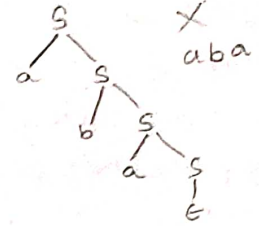
$a^m b^n$

$S \rightarrow AB \mid \epsilon$

$A \rightarrow aA \mid \epsilon$

$B \rightarrow bB \mid \epsilon$

① $S \rightarrow aS \mid bS \mid \epsilon$ → $(a+b)^*$ (Σ^*)



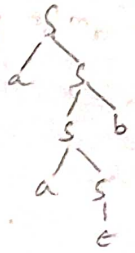
✗ aba

① $S \rightarrow aSbS \mid bSaS \mid \epsilon$

② $S \rightarrow aSb \mid a \mid \epsilon$

equal number of a's and b's + 1

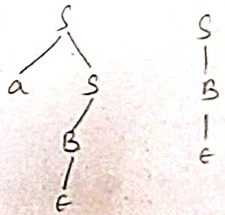
⑤ $S \rightarrow aS \mid Sb \mid \epsilon$ ✓



order also important.

④ $S \rightarrow as \mid B$ ✓

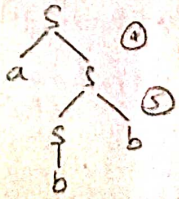
$B \rightarrow bB \mid \epsilon$



⑥ $S \rightarrow aS \mid Sb \mid aSb \mid \epsilon$

not needed.

$L_7 = \{a^m b^n \mid m, n \geq 1\}$.



④ $S \rightarrow aS \mid Sb \mid ab$

⑤ $S \rightarrow aS \mid aB$

$B \rightarrow bB \mid b$

⑦ $S \rightarrow AB$

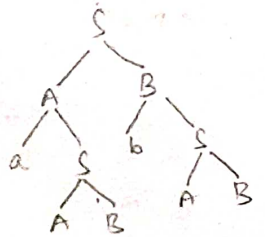
$A \rightarrow aA \mid a$

$B \rightarrow bB \mid b$

④ $S \rightarrow AB$

$A \rightarrow aAB \mid \epsilon$

$B \rightarrow b \mid \epsilon$



a తరువాత b వస్తుంది

✗

$S \rightarrow AB$

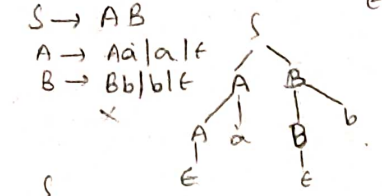
$A \rightarrow aA \mid \epsilon$

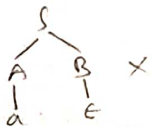
$B \rightarrow bB \mid \epsilon$ ✓

$L_8 = \{ a^m b^n \mid m = n,\ m, n \geq 0 \}$

ϵ; ab, aabb

$S \to AB$
$A \to Aa \mid a \mid \epsilon$
$B \to Bb \mid b \mid \epsilon$



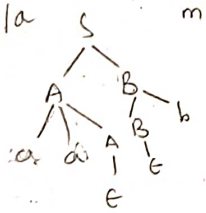


$L_9 = \{ a^m b^n \mid m > n,\ m, n > 0 \}$

$S \to AB$
$A \to aaAB \mid \epsilon \mid a$
$B \to b$

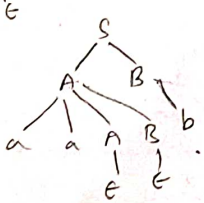
m=2 n=1
m=3 n=1
m=4 n=1

{ aab
aaabb
aaaab

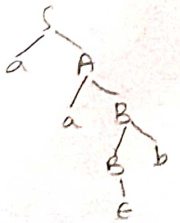


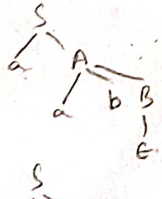
$A \to aaAB \mid \epsilon$
$B \to Bb \mid \epsilon$

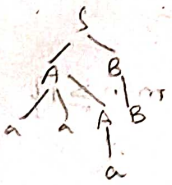
$S \to aA$
$A \to aBb \mid \epsilon$
$B \to b \mid \epsilon$



aab.







$S \to aSb \mid \epsilon$

$S \to abS \mid \epsilon$
$(ab)^n$
$a^m b^{2m}$
$S \to aSbb \mid \epsilon$
$a^{2m} b^m$
$S \to aaSb \mid \epsilon$.

④ ① $S \to aSb \mid aS \mid aab$

② $S \to AB$
$A \to aA \mid a$
$B \to aBb \mid ab$

Try aaab? doubt

L10 ② $S \to aSb \mid Sb \mid ab$

① $S \to AB$

L11:-
$S \to aSbS \mid bSaS \mid \epsilon$

L_{12}: $S \to aSa \mid bSb \mid c$ → palindrome

$L_{13} = \{ ww^R \mid \}$

$L_{10} = \{ a^m b^n \mid m < n,\ m, n > 0 \}$

$L_{11} = \{ w \mid w \in \{a, b\}^*$, w has equal no of a's & b's$\}$

$L_{12} = \{ wcw^R \mid w \in \{a, b\}^*$, c is constant character, w^R; reverse of w$\}$

$L_{13} = \{ ww^R \mid w \in \{a, b\}^* \}$

$S \to aSa \mid bSb \mid \epsilon$ abaaba → even length palindromes only
babbab

$L_{14} = \{ w \mid w$ is a palindrome over $\{a, b\} \}$.

$S \to aSa \mid bSb \mid a \mid b \mid \epsilon$ → odd ones.

$L_{15} = \{ a^m b^m c^n d^n \mid m, n \geq 0 \}$

$S \to AB$
$A \to aAb \mid \epsilon$
$B \to cBd \mid \epsilon$

for b a pop
c push for d c pop

CFL → context free language

$L_{16} = \{ a^m b^n c^n d^m \mid m, n \geq 0 \}$

CFL

$S \to aSd \mid \epsilon$
$X \to bXc \mid \epsilon$

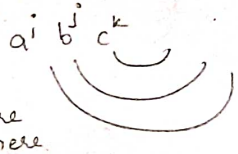
$S \to aXd$
$X \to bXc \mid Y$
$Y \to bYd \mid \epsilon$

a push b push
for c pop b
for d pop a

L_{17}: $\{ a^m b^n c^m d^n \mid m, n \geq 0 \}$

not CFL

Because Two stacks are needed here



Assignment: 1) simplification of CFG → procedure

2) CNF
chomsky normal form Grammar
$\alpha \to \beta$ format

3) GNF
Greibach Normal form Grammar

definitions

01/11/2023 Wednesday.

$S \rightarrow aB \mid bA$

$A \rightarrow a \mid aS \mid bAA$

$B \rightarrow b \mid bS \mid aBB$

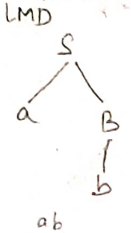
String: aaabbabbba

parse tree

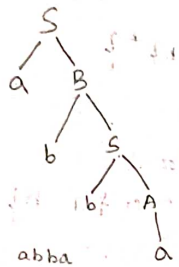
LMD

RMD

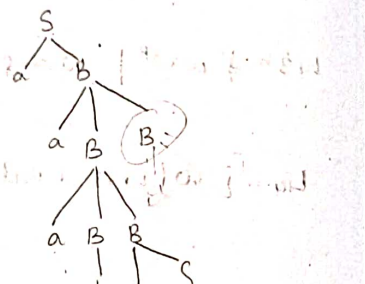
LMD

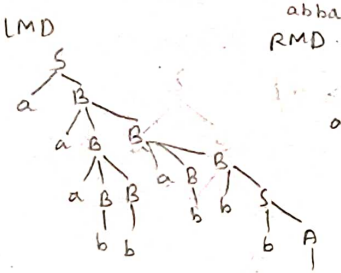


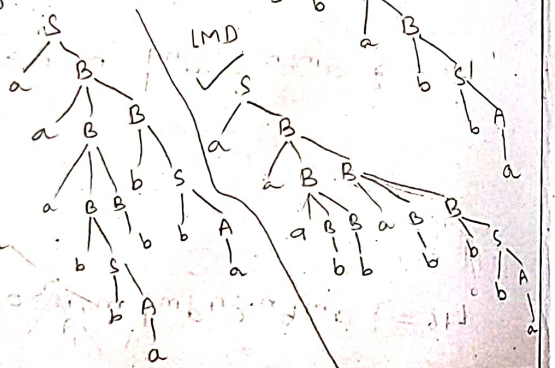
ab



abba







LMD

$S = aB$

$S = aaBB$

$S = aaaBBB$

$S = aaabBB$

$S = aaabbB$

$S = aaabbaBB$

$S = aaabbabB$

$S = aaabbabbbA$

$S = aaabbabbbba$

RMD.

$S = aB$

$= aaBB$

$= aaBbS$

$= aaBbbA$

$= aaBbba$

$= aaaBBbba$

$= aaaBbSbba$

$= aaaBbabbba$

$= aaabbabbba$

abbabba

Push Down Automata

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

Q = finite set of states.

Σ = Input Alphabet

T = stack alphabet.

δ = transition function

$\delta : Q \times \Sigma \times T \rightarrow (Q, T^*)$

q_0 = Initial state.

Z_0 : Initial stack symbol / bottom of stack Symbol.

F : final state.

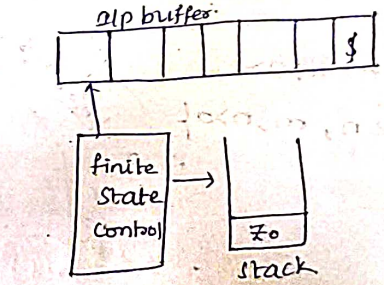
⇒ Every RL is context free → True.

eg: $\delta(q_0, a, z_0) \rightarrow (q_1, AZ_0)$ ↑ in stack.

stack emptyness recognition symbol: Z_0.

Relevant ఉన్నవి మాత్రమే రాస్తాము.

⇒ If transition not defined: then reject



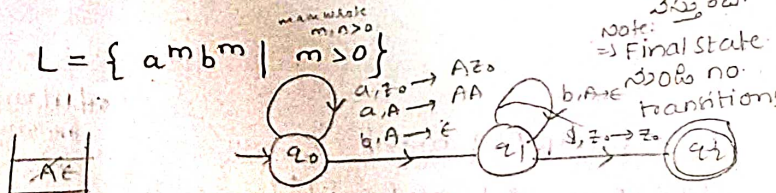
May be less no. of states in pushDown Automata

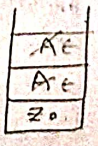
Why state change అవుతుంది?

b, A → check చేసారు అని assurance వస్తుంది.

Note:
⇒ Final state నుండి no transitions

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





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

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

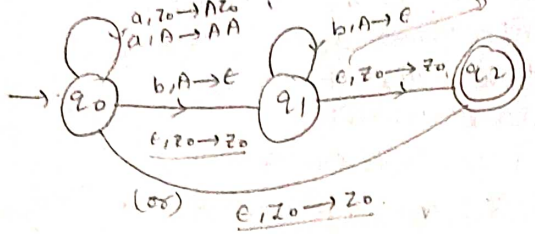
$\delta(q_0, b, A) \rightarrow \delta(q_1, \epsilon)$

$\delta(q_1, b, A) \rightarrow \delta(q_1, \epsilon)$

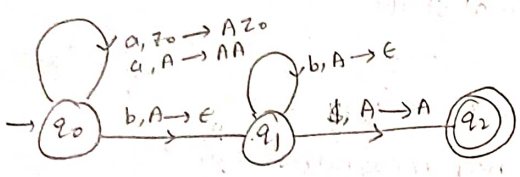
$\delta(q_1, \$, Z_0) \rightarrow \delta(q_2, Z_0)$

no action Z_0 → to చేయకూడదు

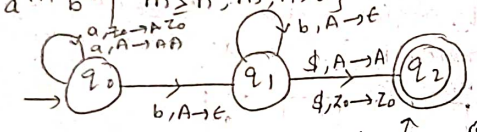
$L_2 = \{a^m b^m \mid m \geq 0\}$



$L_3 = \{a^m b^n \mid m > n,\ m, n > 0\}$ $l_3 = \{a, aab, aaaabb, aaaaab \ldots\}$

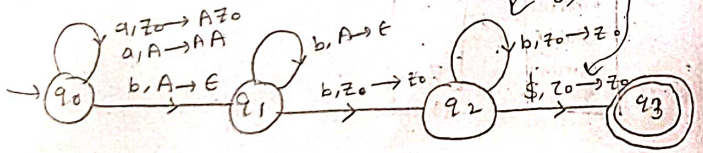


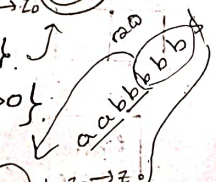
if $m \geq n$ $a^m b^n \mid m \geq n, m, n > 0\}$



$L_4 = \{a^m b^n \mid m \geq n,\ m, n > 0\}$

$L_5 = \{a^m b^n \mid m < n,\ m, n > 0\}$





09/11/2023.
Thursday.

$L_6 = \{a^m b^{2m} \mid m > 0\}$

$L_7 = \{wcw^R \mid w \in \{a,b\}^*\}$

NPDA $L_8 = \{ww^R \mid w \in \{a,b\}^*\}$

$L_9 = \{a^m b^{m+m} c^n \mid m, n > 0\}$ $\{a^m b^m b^n c^n \mid m, n > 0\}$

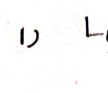
$L_{10} = \{a^m b^n c^n d^m \mid m > 0\}$

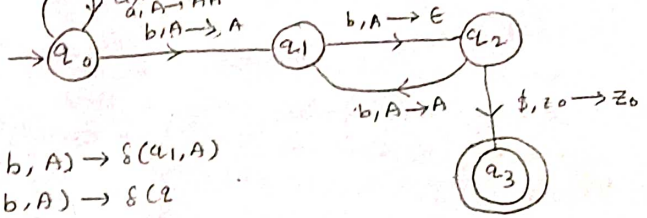
$L_{11} = \{a^m b^n c^{n+m} \mid m, n > 0\}$

$L_{12} = \{a^m b^n a^m \mid m > 0\}$

1) $L_6 = \{a^m b^{2m} \mid m > 0\}$.

$L_6 = \{abb,\ aabbbb$



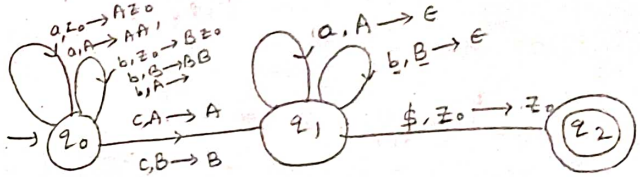


$\delta(q_0, b, A) \rightarrow \delta(q_1, A)$

$\delta(q_1, b, A) \rightarrow \delta(q$

DPPDS

2) $L_7 = \{wcw^R \mid w \in \{a,b\}^*\}$. $\{aca, abcba \ldots$ $bcb, bacab \ldots$



$\delta(q_0, a, z_0) \rightarrow \delta(q_0, A z_0)$

$\delta(q_0, b, z_0) \rightarrow \delta(q_0, B z_0)$

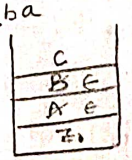
$\delta(q_0, a, A) \rightarrow \delta(q_0, AA)$

$\delta(q_0, a, B) \rightarrow \delta(q_0, AB)$

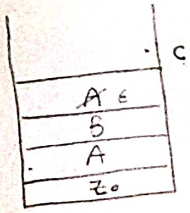
$\delta(q_0, b, A) \rightarrow \delta(q_0, BA)$

$\delta(q_0, b, B) \rightarrow \delta(q_0, BB)$

abcba



abacaba



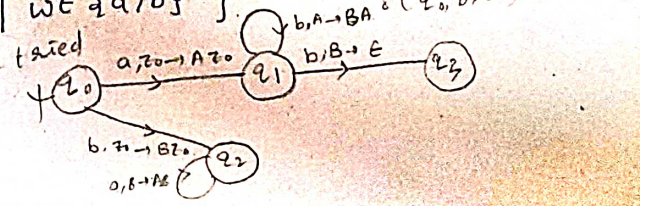
a వచ్చి A ఉన్నప్పుడే pop.

b వచ్చి B ఉన్నప్పుడే pop.

Non-deterministic PDA

3) $L_8 = \{wcw^R \mid w \in \{a,b\}^*\}$.

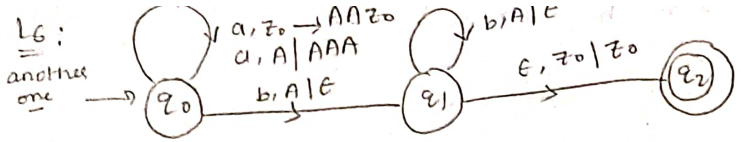
I tried



$\delta(q_0, a, z_0) \rightarrow \delta(q_0, Az_0)$

$\delta(q_0, b, z_0) \rightarrow \delta(q_0, Bz_0)$

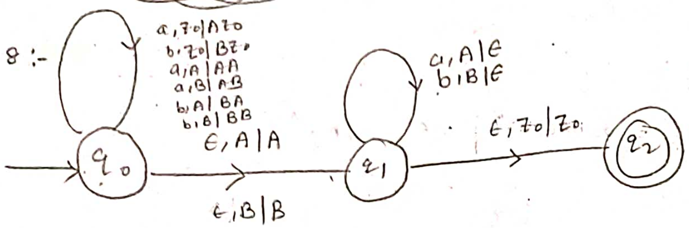
L6 :
another one



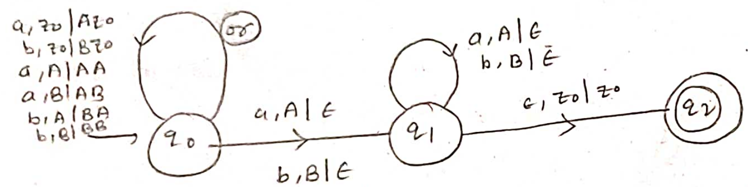
wcw

a b b c a b b overlapping is there
we can't use stack here

L8 :-



(or)



Note: NPDA ఉంటే DPDA ఉండాలని లేదు

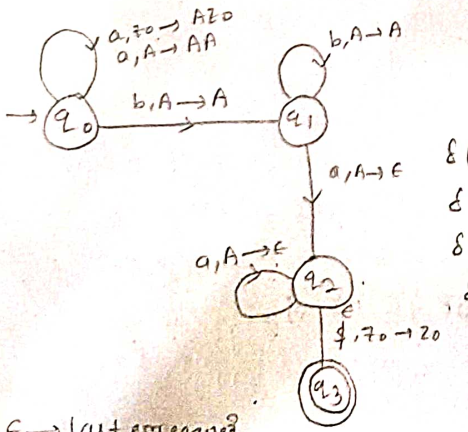
H.W
PDA's
CFG's

10/11/2023
Friday.

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

a → push
b
a → pop

$L = \{a^i b^i\}$



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

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

$\delta(q_0, b, A) \rightarrow \delta(q_1, A)$

$\delta(q_1, a, A) \rightarrow \delta(q_1, a, \epsilon)$.

ε → last
No strings or symbols to be read
ε → ... symbols ... ε ...

① Acceptance by final state:

Theory of computation
Aho Ulma

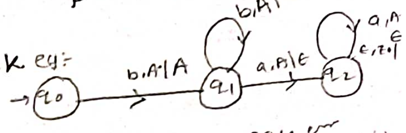
$(Q, \Sigma, T, \delta, q_0, Z_0, F)$

$(q_0, w, Z_0) \vdash^* (q_f, \epsilon, \lambda)$

Initial state
w: given string.

$q_f \in F, \lambda \in T^*$

② Acceptance by empty stack eg:-



$(Q, \Sigma, T, \delta, q_0, Z_0)$

for m > n case or
ε, A | ε add it
ε, Z0 | ε

$(q_0, w, Z_0) \vdash^* (q, \epsilon, \epsilon)$

for any $q \in Q$

⇒ Information needed to start from where the machine has been halted

ID (Instantaneous Description)

$(q, aw, Z\lambda_1) \vdash (q_1, w, \beta\lambda_2)$ ⇒ with each transition ID changes

present state.

remaining string to be read.

Symbol on stack

Next state.

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

for b a's pop

if Z0 ...
$a^m b^m$?
stack empty
...
so ...
string ...
... Z0 in stack.
At last Z0 is also popped and then string is accepted.

CFL's are closed under union, concatenation, kleene closure

Closure properties of context free languages.

1) union ✓
2) concatenation ✓
3) Intersection.
4) kleene closure ✓
5) complement.

If L1 is CFL
then $L1^*$ is CFL

① union:

L_1 L_2

G_1 G_2

(V_1, Σ, P_1, S_1) (V_2, Σ, P_2, S_2)

$L_1 \cup L_2 : G : (\{V_1 \cup V_2 \cup S\}, \Sigma, P_1 \cup P_2 \cup \{S \rightarrow S_1 \mid S_2\}, S)$

② concatenation:

L_1 L_2

G_1 G_2

(V_1, Σ, P_1, S_1) (V_2, Σ, P_2, S_2)

$L_1 L_2 : G : \left(\{V_1 \cup V_2 \cup S\}, \Sigma, P_1 \cup P_2 \cup \{S \to S_1 S_2\}, S \right)$

eg: N not closed under subtraction atleast one not satisfy.

4) Kleene closure:

$L_1^* : G_1 : \left(\{V_1 \cup S\}, \Sigma, P_1 \cup \{S \to S_1 S \mid \epsilon\}, S \right)$

eg:- $A \to aA \mid \epsilon$

a^*

Intersection

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

$L_2 = \{a^p b^q c^q \mid p, q > 0\}$

$L_1 \cap L_2 = \{a^k b^k c^k \mid k > 0\}$

not a CFL

Intersection of two CFLs may or may not be a CFL.

closure properties of Regular Languages.

1) Kleene closure: Rs is a RE whose language is $L \cdot R^*$ is a regular expression whose language is L^*.

Regular expression: A RE can be defined as a language or string accepted by a finite automata. We know that a finite automata consists of five touples $\{Q, \Sigma, \delta, q_0, F\}$. Among them a RE is a string on Σ, i.e it will consist only with input alphabets.

2) positive closure: RS is a regular expression whose language is L, R^+ is a regular expression(RE) whose language is L^+.

3) complement: The complement of a Language L (w.r.t an alphabet E such that E^* contains L) is $E^* - L$. since E^* is surely regular, the complement of a RL is always regular.

4) Reverse operator: Given language L, L^R is the set of strings whose reversal is in L.

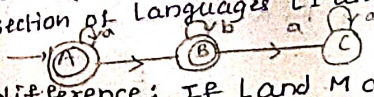
eg:- $L = \{0, 01, 100\}$, $L^R = \{0, 10, 001\}$.

Proof:- Let E be a RE for L. we show how to reverse E, to provide a regular expression E^R for L^R.

5) Union: Let L and M be the languages of RE R and S respectively. Then R+S is RE whose language is $(L \cup M)$.

6) Intersection: If L and M are RLs, then $L \cap M$ is also RL.

Proof: Assume L_1 and L_2 are regular languages, and we wish to demonstrate that the intersection of L_1 and L_2 is likewise a regular language. DeMorgen's law may be used to calculate the intersection of Languages L_1 and L_2.



7) Set difference: If L and M are regular languages then so is L - M = strings in L but not M.

Proof: Let A and B be DFA's whose languages are L and M respectively. Construct C, the product automation of A and B make the final states of C be the pairs, where A-state is final but B-state is not.

Homomorphism: A homomorphism is with domain Σ^* for some alphabet Σ which preserves concatenation: $h(v.w) = h(v).h(w)$.

eg: String MAMATHA can be converted to corresponding to ASCII language by replacing each letter with corresponding ASCII value as 83727286565.

need?: It helps to translate one language to other easily by keeping all its properties same.

How are closed?: Let E be a RE for L.
apply h to each symbol in E
Then h(L) is RE

eg: $h(0) = ab$; $h(1) = c$.

Let L be the language of RE $01^* + 10^*$

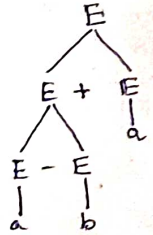
Then h(L) is the language of RE $ab\,c^* + c(ab)^*$
(so is Regular language.

Inverse homomorphism: Let h be a homomorphism and L a language whose alphabet is the o/p language of h. $h^{-1}(L)$
$= \{w \mid h(w) \text{ is in } L\}$.

1) Leftmost derivation: read input string from left to right

eg: $E \Rightarrow E+E$
$E \to E-E$ Input: a-b+a
$E \to a \mid b$

LMD



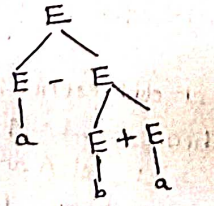
a-b+a ✓

By substitution

$E = \underline{E} + E$
$E = E-E+E$
$E = a-E+E$
$E = a-b+E$
$E = a-b+a$

RMD.

$E = E-E$
$= E-E+E$
$= E-E+a$
$= E-b+a$
$= a-b+a$



a-b+a ✓

Simplification of context free Grammar (CFG): It is possible in a CFG that the derivation string does not require the use of all the production rules and symbols. Additionally, there might be some unit and null productions. The term "simplification of CFGs" refers to the removal of certain productions and symbols. Essentially, simplification consists of the following steps:

- CFG Reduction
- Unit productions Removal
- Null productions Removal

Simplification: Context-free grammar can effectively represent a variety of languages. Because not all grammar is streamlined, some grammars may contain superflous symbols (non-terminal). Adding extra pointless symbols lengthens the grammar. The grammar that has been reduced by the removal of symbols is said to be simpler. Below are some reduced characteristics of grammar:

1) A word in L is derived from each variable (i.e., non-terminal) and every terminal in G.
2) Where x and Y are not terminal, there should be no production as $X \longrightarrow Y$
3) There need not to be a production of $X \to \epsilon$ if it is not in the L language.

Removal of Useless symbols.

If a symbol does not exist on the RHS of the production rule and does not participate in the derivation of any string, it may be considered to be useless. Similar to this, if a variable is not used to create any strings, it may be worthless.

eg:

$T \longrightarrow xxY \mid xbX \mid xzT$
$X \longrightarrow xx$
$Y \longrightarrow xY \mid y$
$Z \longrightarrow zz$

In the example given above, the variable 'z' will never occur in any string's derivation. Thus, the production $Z \rightarrow xz$ is useless. Therefore, let's eliminate it. This way, all the other productions would be written in a way that the variable z can never reach from the 'T' starting variable.

Production $X \rightarrow xx$ would also be useless because it cannot terminate it in anyway. But if it never terminates, it can never ever produce a string. Thus, such a production can not participate in any derivation ever.

For the removal of this useless production $X \rightarrow xX$, we will find all the variables first that never lead to a terminal string, like the variable 'x'. we will then remove all the productions where the variable 'B' occurs.

Elimination of ϵ production:

All the productions that are of type $S \rightarrow \epsilon$ are known as ϵ productions. These types of productions can be removed only from the grammar that do not happen to generate ϵ.

Step 1: Find all nullable non-terminal variables which derive ϵ first.

Step 2: For all the productions of $X \rightarrow a$, construct all production $X \rightarrow a'$, where, a is obtained from X by removing non-terminals from the first step.

Step 3: Combine the result of the second step with the original production. Remove all the ϵ productions.

eg:- $S \rightarrow ABA$

$A \rightarrow 0A \mid \epsilon$

$B \rightarrow 1B \mid \epsilon$

while removing ϵ production, the rule $A \rightarrow \epsilon$ and $B \rightarrow \epsilon$ would be deleted. To preserve CFG's meaning, we are placing ϵ actually at the RHS whenever A B have appeared.

Let us take

$S \rightarrow ABA$

If the first A at RHS is ϵ, Then $S \rightarrow BA$

Similarly, if the last A in RHS = ϵ. Then, $S \rightarrow AB$

If $B = \epsilon$ then $S \rightarrow AA$

If $A = \epsilon$ then $S \rightarrow B$

If B and A are ϵ then $S \rightarrow A$

Now

$S \rightarrow AB \mid BA \mid AA \mid A \mid B$

Now, consider

$A \rightarrow 0A$

If $A = \epsilon$ then $A \rightarrow 0 \Rightarrow A \rightarrow 0A \mid 0$

If $B = \epsilon$ then $B \rightarrow 1 \Rightarrow B \rightarrow 1B \mid 1$

Removing unit productions

The productions that result in giving of one non-terminal to another are known as unit productions. To get rid of unit productions, take the following actions.

Step 1: To remove $A \rightarrow B$, add production $A \rightarrow x$ to the grammar rule whenever $B \rightarrow x$ occurs in the grammar.

Step 2: Delete $A \rightarrow B$ from the grammar.

Step 3: Repeat the first and second steps until all the unit productions are removed.

$S \rightarrow 0x \mid 1Y \mid z$

$x \rightarrow 0S \mid 00$

$Y \rightarrow 1 \mid x$

$z \rightarrow 01$

$S \rightarrow z$ is a unit production. However while removing $S \rightarrow z$, we have to consider what z gives. So, we can add a rule to S.

$S \rightarrow 0x \mid 1Y \mid 01$

Similarly, $Y \rightarrow x$ is also a unit production, so we can modify it as

$B \rightarrow 1 \mid 0S \mid 00$

Finally,

$S \rightarrow 0X \mid 1Y \mid 01$

$X \rightarrow 0S \mid 00$

$Y \rightarrow 1 \mid 0S \mid 00$

$Z \rightarrow 01$

2) Chomsky's Normal Form (CNF):

CNF stands for chomsky normal form. A CFG (context free grammar) is in CNF (chomsky normal form) if all production rules satis

CNF stands for chompsky normal form. A CFG (context free grammar) is in CNF if all production rules satisfy one of the following conditions.

- start symbol generating ϵ. For example, $A \rightarrow \epsilon$
- A non-terminal generating two non-terminals. For example, $S \rightarrow AB$
- A non-terminal generating a terminal. For example $S \rightarrow a$.

Greibach Normal Form (GNF): GNF stands for Greibach normal form. A CFG (context free Grammar) is in GNF if all the production rules satisfy one of the following conditions.

- A start symbol generating ϵ. For example, $S \rightarrow \epsilon$
- A non-terminal generating a terminal. For example $A \rightarrow a$.
- A non-terminal generating a terminal which is followed by any number of non-terminals. For example, $S \rightarrow aASB$.

Definitions:

Regular Grammar: Regular Grammar generates regular language. They have a single non-terminal on the left-hand side consisting of a single terminal or single terminal followed by a non-terminal The production must be in the form:

$A \rightarrow xB$

$A \rightarrow x$

$A \rightarrow Bx$

where $A, B \in$ variable (V) and $x \in T^*$ i.e string of terminals.

Context-free Grammar: context free Grammar is formal grammar the syntax or structure of a formal language can be described using context-free grammar (CFG), a type of formal grammar. The grammar has four tuples: (V, T, P, S).

V -> It is the collection of variables or nonterminal symbols.

T → It is a set of terminals.

P → It is the production rules that consist of both terminals and non-terminals.

S → It is the starting symbol.

A Grammar is said to be context-free grammar if every production is in the form of:

$G \rightarrow (V \cup T)^*$, where $G \in V$.

Context-Sensitive Grammar: A context-sensitive grammar is an unrestricted grammar in which all productions are of form-

$\alpha \rightarrow \beta$

where $\alpha, \beta \in (V \cup T)^*$ and $|\alpha| \leq |\beta|$

where α and β are strings of non-terminals and terminals.

context-sensitive grammars are more powerful than context-free grammar because there are some languages that can be described by CSG but not by CFG and CSL are less powerful than unrestricted grammar

$G = \{N, \Sigma, P, S\}$

Where N = set of non-terminal Symbols.

Σ = set of terminal Symbols

S = start symbol of the production.

P = finite set of productions

All rules in P are of the form $\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$.

Unrestricted Grammar: Unrestricted Grammar or Phrase Structure Grammar is the most general in the chomsky Hierarchy of classification. This is type 0 Grammar, generally used to generate Recursively Enumerable languages. It is called unrestricted because no other restriction is made on this except each of their Left hand side being non-empty.

$G = \{N, \Sigma, P, S\}$

N = A finite set of non-terminal symbols or variables

Σ = It is a set of terminal symbols or the alphabet of the language being described, where $N \cap \Sigma = \phi$,

P = It is a finite set of "productions" or "rules".

S = It is a start variable or non-terminal symbol.

If α and β are two strings over the alphabet $N \cup \Sigma$ Then, the rules or productions are of the form $\alpha \rightarrow \beta$. The start variable S appears on the left side of the rule.

① Input : abb

productions

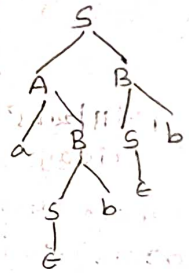
$S \rightarrow AB \mid \epsilon$

$AB \rightarrow aB$

$Bb \rightarrow Sb$

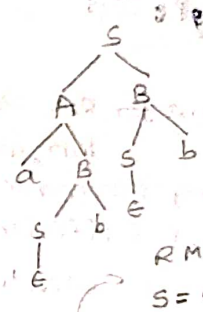
LMD

$S = \underline{A}B$
$= a\underline{B}B$
$= asbB$
$= a\epsilon bB$
$= abB$
$= absb$
$= ab\epsilon b$
$= abb$



RMD

$S = A\underline{B}$
$= ASb$
$= A\epsilon b$
$= Ab$
$= aBb$
$= asbb$
$= a\epsilon bb$
$= abb$



② Input = aabbabba

Productions.

$S \rightarrow aB \mid bA$

$AS \rightarrow a \mid aS \mid bAA$

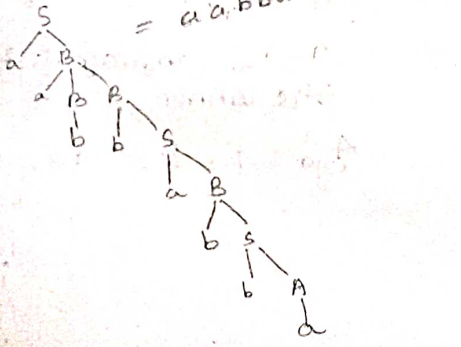
$BS \rightarrow b \mid aS \mid aBB$

LMD

$S = a\underline{B}$
$= aa\underline{B}B$
$= aab\underline{S}$
$= aabbA$
$= aabbaB$
$= aabbabS$
$= aabbabbA$
$= aabbabba$

RMD

$S = aB$
$= aa\underline{B}B$
$= aaBbS$
$= aaBbbA$
$= aaBbba$
$= aabSbba$
$= aabbAbba$
$= aabbabba$



Pumping Lemma (For Regular languages)

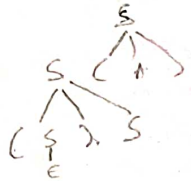
⇒ Pumping lemma is used to prove that a language IS NOT REGULAR.

If A is RL, then A has pumping length p such that any string s where $|s| \geq p$ may be divided into 3 parts $s = xyz$ such that the following conditions must be true.

$xy^i z \in A$ for $\forall\ i \geq 0$

$|y| > 0$

$|xy| \leq p$



$S \rightarrow (S)S \mid \epsilon$

$A \rightarrow ()A \mid \epsilon$

Pumping Lemma for Regular languages.

17/11/2023 MTech Friday.

⇒ NOTE: pumping lemma is necessary but not sufficient.

Let 'L' be a regular language. There exists a natural no. 'n' (pumping lemma constant) for every $z \in L$, $|z| \geq n$, for a division of $z = uvw$ satisfying

1) $|v| \geq 1$ → if $v = \epsilon$ then uw only one string ಬರುತ್ತೆ.
2) $|uv| \leq n$ → w can be anything
3) $uv^i w \in L$, $\forall\ i \geq 0$

⇒ If a Language is Regular then it has to satisfy this lemma.

eg: $L = \{a, aa, aaa \ldots\}$

$v = a$.

$i = 0$ aa
$i = 1$ aaa
$i = 2$ aaaa
$i = 3$ aaaaa

$a^m b^n$

b any one can be v here.

$v = a$ or $v = b$.

n = ?

because

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

↓

which will not fit into the pattern

So here $n = 2$ ಅಂದರೆ ಯಾವುದೇ pattern ಬಂದ್ರೂ ಸಾಧ್ಯ. (n is required).

pumping lemma for CFLs

Let 'L' be a CFL. Then, we can find a natural no 'n' (P.L.C) such that every $z \in L$, $|z| \geq n$ can be written as $z = uvwxy$ satisfying

both v, w cannot be ε.

ಅಂದರೆ ಎರಡರಲ್ಲಿ ಒಂದು ಸಾಧ್ಯ, ಎರಡೂ ಸಾಧ್ಯ ಇರಬಾರದು

1) $|vx| \geq 1$
2) $|vwx| \leq n$
3) $uv^i wx^i y \in L$, $\forall\ i \geq 0$.

To prove non-regularity or non-context free things we use pumping lemma.

$S \rightarrow aSxd \mid \epsilon$

$x \rightarrow bxd \mid \epsilon$

$S \rightarrow aSd \mid A$

$A \rightarrow bAc \mid \epsilon$

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

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

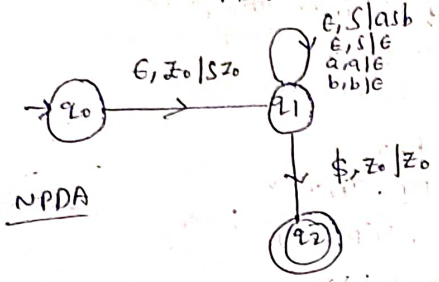
× $L_3 = \{wcw \mid w \in \{a, b\}^*\}$

abcab.

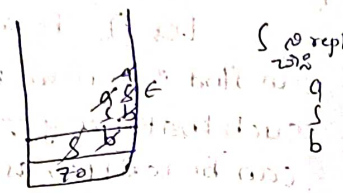
equivalence b/w CFG & PDA.

CFG ⇒ PDA

PDA ⇒ CFG.



push: non-terminal ಮೇಲೆ ಯಾವ production ಆ push ಮಾಡಬೇಕು
terminal ಮೇಲೆ pop match



S ಅಲ್ಲಿ replace ಮಾಡಿ
q
S
b

2/3/11/2023.
Thursday.

CFG to PDA

1) push 'S' on to stack initially

$\delta(q_0, \epsilon, S) = (q_1, S)$ non-terminal

2) when top of stack (TOS) is NT: NT's production is

$NT \rightarrow \beta$

$\delta(q_1, \epsilon, NT) = (q_1, \beta)$

write the above kind of transitions for every NT.

3) when TOS is T: & I/P symbol is the same T.

$\delta(q_1, T, T) = (q_1, \epsilon)$

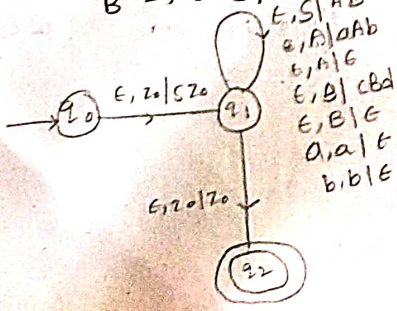
4) $\delta(q_1, \epsilon, Z_0) = (q_2, \epsilon)$ or ϵ_{Z_0}

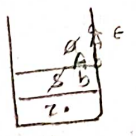
q_2 is F.S (final state).

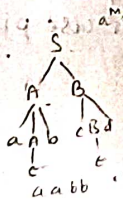
eg:- $S \rightarrow AB$

$A \rightarrow aAb \mid \epsilon$

$B \rightarrow cBd \mid \epsilon$

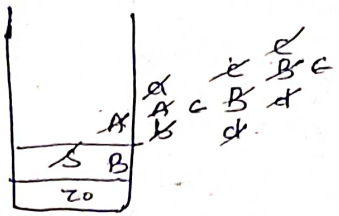


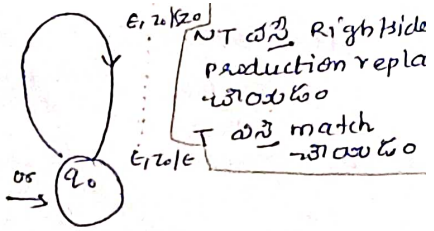




For all the productions → push
For all the input terminal symbols & pop action.

eg:- a a b b c c d d





State sequence: $q_0 q_1 q_2 q_2 q_2 q_2 q_2 q_1 q_2 q_2 q_2 q_2 q_2 q_2$

⇒ Any CFG can be converted into PDA with single state with ~~final state~~ empty stack acceptance.

$\epsilon, A \mid aAb$

$a, A \mid Ab$

See once in textbook while reading.

Home work:

Grammars $\frac{\alpha}{?} \rightarrow \frac{\beta}{?}$ Rules. $\alpha \rightarrow$

R

CF

context sensitive $aB \rightarrow aBd$, $cA \rightarrow cAc$

context defined ಆಗಿರುವಂತಹವು

1) What is context sensitive Grammar.

A context sensitive grammar whose productions are of the form $\alpha A \beta \rightarrow \alpha \gamma \beta$ where $\alpha, \beta \in (N \cup T)^*$, $A \in N$; $\gamma \in (N \cup T)^+$ and a rule of the form $S \rightarrow \lambda$ is allowed if the start symbol S do not appear on the right hand side of any rule. The language generated by such a grammar is called a context-sensitive language.

eg: $a^n b^n c^n$, $n \geq 1$

$S \rightarrow dSBC \mid aBC$

$CB \rightarrow HB$

$HB \rightarrow HC$

$HC \rightarrow BC$

$aB \rightarrow ab$

$bB \rightarrow bb$

$bC \rightarrow bc$

$cC \rightarrow cc$

String aabbcc

$S \rightarrow aSBC \mid aBC$

$CB \rightarrow HB$
$HB \rightarrow HC$
$HC \rightarrow BC$
$aB \rightarrow ab$
$bB \rightarrow bb$
$bC \rightarrow bc$
$cC \rightarrow cc$

$S = aSBC$
$= aaBCBC$
$= aabCBC$
$= aabHBC$
$= aabHCC$
$= aabBCC$
$= aabbCC$
$= aabbcC$
$= aabbcc$

a) context free Grammar

Given a production rule in the form of $A \rightarrow \alpha B \beta$ where A is a non terminal symbol, and α and β are sequences of terminal and/or nonterminal symbols, then production can be applied by replacing the nonterminal B with the sequence α.

$A \rightarrow \alpha B \beta$

- A is nonterminal symbol that is being replaced
- α is the sequence of symbols that can appear before the nonterminal B.
- B is the nonterminal symbol that is being replaced by the sequence α.
- β is the sequence of symbols that can appear after the nonterminal B.

aquaitance of language
logic → PS
use [illegible]

PDA to CFG.

Dec 6 2023
wednesday.

$(Q, \Sigma, T, \delta, q_0, z_0)$

1) $S \longrightarrow [q_0\ z_0\ q]$ $q \in Q$

$2^n \rightarrow$ positive possiblities

Ex: $q_0, q_1 \in Q$

$S \rightarrow [q_0\ z_0\ q_0] / [q_0\ z_0\ q_1]$

2) $\delta(q, a, z) \rightarrow (p, \epsilon)$

$[q, z, p] \rightarrow a$

$3^2 = 3 \times 3$

$z_0\ q_1$

$\_\ \_\ \_ = 2^3 = 8$

⇒ how many $z_1 z_2 z_3$ leaves

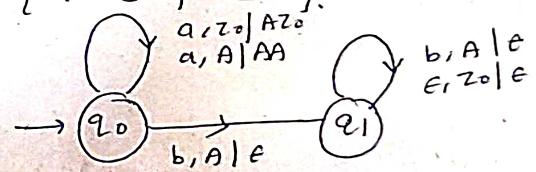
3) $\delta(q, a, z) \longrightarrow (p, z_1 z_2)$

$[q\ z\ ^{①}] \longrightarrow a[p\ z_1\ ^{②}][\ ^{②}\ z_2\ ^{①}]$

if $z_1 z_2 z_3$ is there

$[q\ z\ \_] \longrightarrow a[p\ z_1\ \_][\_\ z_2\ \_][\_\ z_3\ \_]$

$\{a^m b^m \mid m \geq 1\}$



① $S \rightarrow [q_0\ z_0\ q_0]$

$S \rightarrow [q_0\ z_0\ q_1]$

② pop actions. (3)

$\delta(q_0, b, A) \longrightarrow (q_1, \epsilon)$

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

$\delta(q_1, b, A) \longrightarrow (q_1, \epsilon)$

$[q_1\ A\ q_1] \rightarrow b$

$\delta(q_0, \epsilon, z_0) \rightarrow (q_1, \epsilon)$

$[q_0\ z_0\ q_1] \rightarrow \epsilon$

3) i) $\delta(q_0, a, z_0) \rightarrow (q_0, Az_0)$

ii) $\delta(q_0, a, A) \rightarrow (q_0, AA)$

i) $[q_0\ z_0\ \_] \longrightarrow a[q_0\ A\ \_][\_\ z_0\ \_]$

✗ $[q_0\ z_0\ q_0] \longrightarrow a[q_0\ A\ q_0][q_0\ z_0\ q_0]$

✗ $[q_0\ z_0\ q_0] \longrightarrow a[q_0\ A\ q_1][q_1\ z_0\ q_0]$

✗ $[q_0\ z_0\ q_1] \longrightarrow a[q_0\ A\ z_0][q_0\ z_0\ q_1]$

$[q_0\ z_0\ q_1] \longrightarrow a[q_0\ A\ q_1][q_1\ z_0\ q_1]$

ii) ✗ $[q_0\ A\ q_0] \longrightarrow a[q_0\ A\ q_0][q_0\ A\ q_0]$

✗ $[q_0\ A\ q_0] \longrightarrow a[q_0\ A\ q_1][q_1\ A\ q_0]$

✗ $[q_0\ A\ q_1] \longrightarrow a[q_0\ A\ q_0][q_0\ A\ q_1]$

$[q_0\ A\ q_1] \longrightarrow a[q_0\ A\ q_1][q_1\ A\ q_1]$

$[q_0\ A\ q_0] \longrightarrow a[q_0\ A\ q_0][q_0\ A\ z_0]$ → deriving itself ($q_0\ A\ z_0$) but not defined later

Useless productions

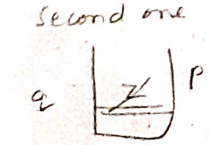
=> i) Non Reachable ii) Non-Generating

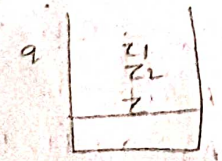
Non-Reachable from starting non-terminal.

$S \rightarrow AB$

$A \rightarrow aA$ $A \rightarrow aA \mid \epsilon$ (accepted)

Second one

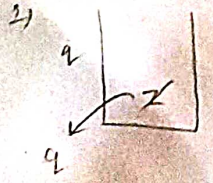




( )

ಕೊನೆಯಲ್ಲಿ ಸ್ಟಾಕ್ empty

2)



$S \rightarrow A$
$A \rightarrow aBC$
$B \rightarrow aBD \mid b$
$D \rightarrow b$
$C \rightarrow \epsilon$

$S \rightarrow A$
$A \rightarrow aB$
$B \rightarrow aBb \mid b$

07/12/2023
Thursday.

work:
+ Grammar definitions

$(Q, \Sigma, T, \delta, q_0, B, q_{accept}, q_{reject})$

Q : Finite set of states

Σ : Input Alphabet.

T : Tape Symbols.

δ : Transition function

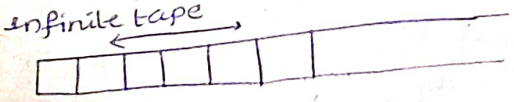
$\delta : Q \times \{\Sigma \cup T\} \rightarrow Q \times T \times [L, R]$

q_0 : Initial state.

B : Blank symbol.

q_{accept} : Acceptance state.

q_{reject} : rejection state.

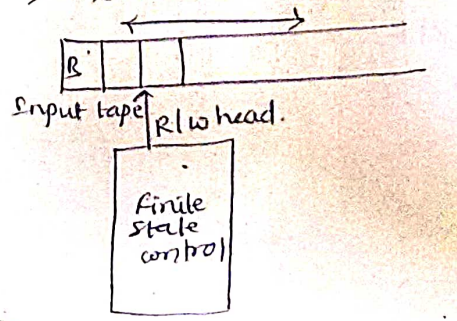


Turing Machine :

=> Advantages

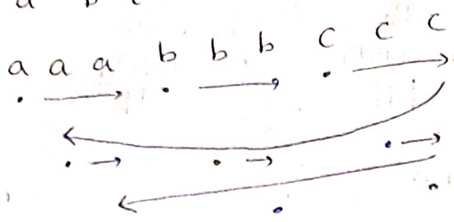
1) It is infinite (huge memory)
2) The head can both read and write
3) Both directions => forward and backwood.

=> The ends of the strings filled with blank spaces



L → Left
R → Right
one symbol at a time
Q → present state.
$\Sigma \cup T$ → input symbol
Q → next state.
T → movement of the head
X[L/R]

eg:- $a^m b^m c^m$



eg:- $a^m b^n c^m d^n$ not CFG but TML ✓

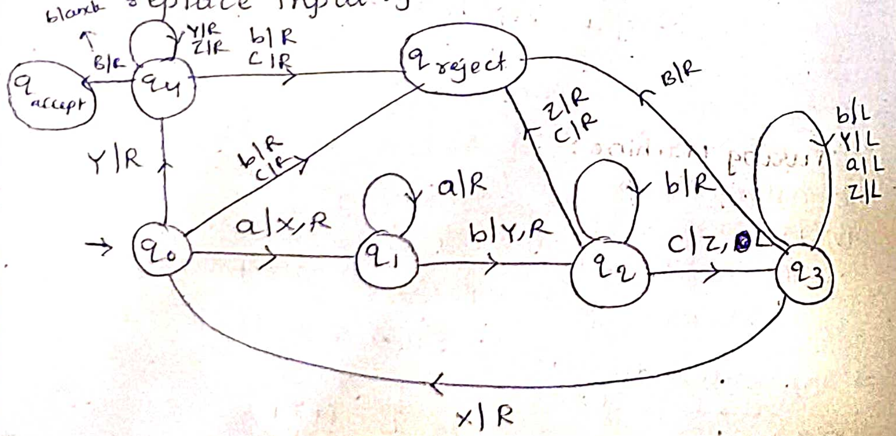
eg:- wcw

abbcabb not CFG but TML ✓

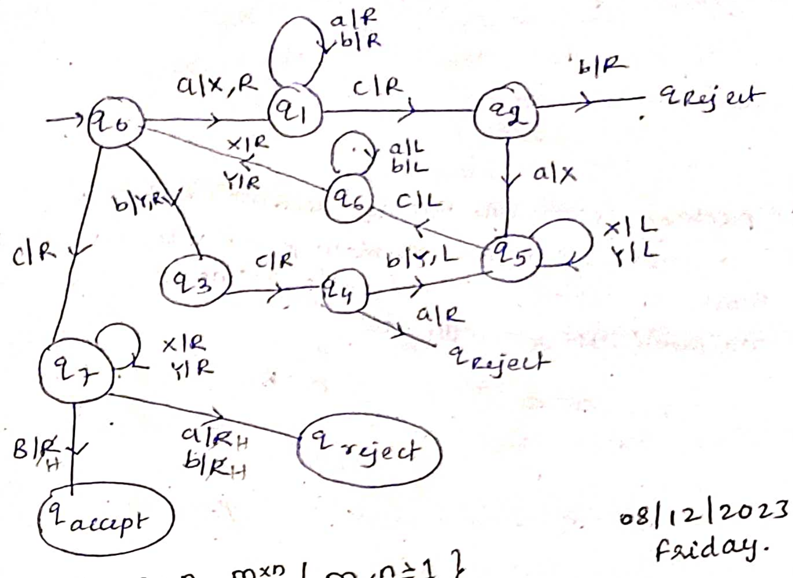
replacing →

X Y Z
a a a b b b c c c

⇒ According to the requirement we can replace input symbols in Tape.



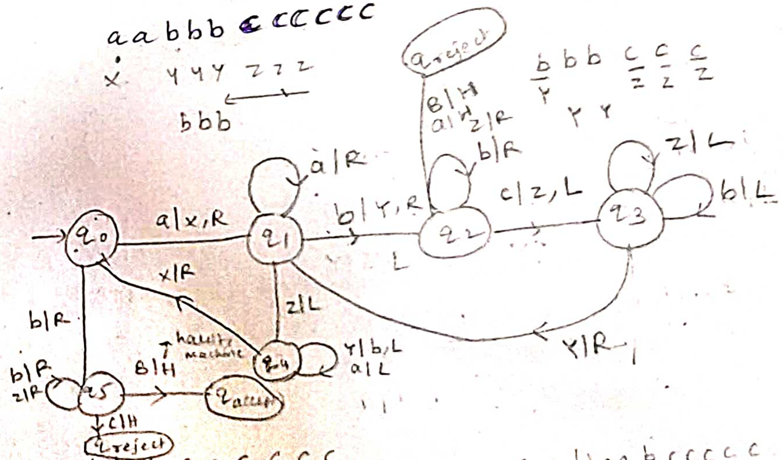
eg:- wcw



08|12|2023
Friday.

eg:- $\{ a^m b^n c^{m \times n} \mid m, n \geq 1 \}$

a a b b b c c c c c c
X Y Y Y Z Z Z
b b b

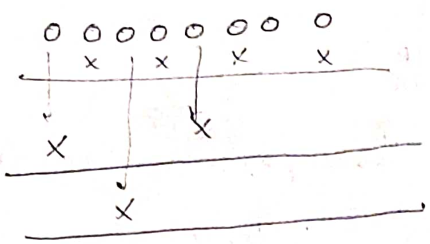


a a b b b c c c c c c
↑ ↑ ↑ ↑ ↑ ↑ ↑
X Y Y Y Z Z Z

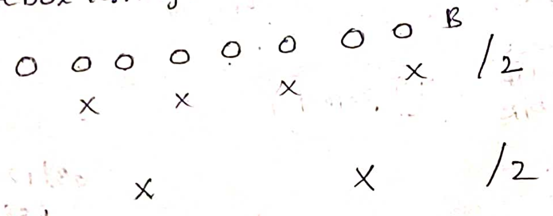
a a bbaa b c c c c c
b తర్వాత a's వస్తే reject

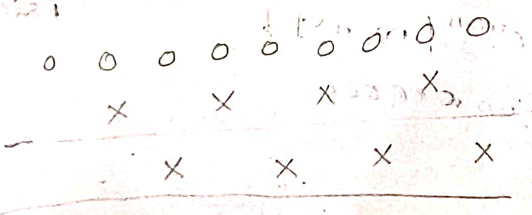
0^{2n} 0^{2n} → even 0^{2n+1} → odd

0 0 0 0 0 0 B ✓ even
X X X
0 0 0
X



⇒ If problem is solvable Turing machine exist.
⇒ If Turing machine exist then problem is solvable.
⇒ Blackbox testing: without code, testing o/p
⇒ Whitebox testing: with code



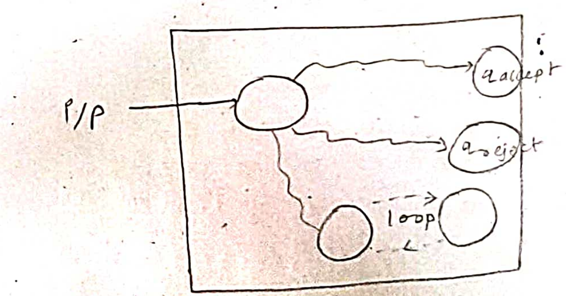


Turing machine Languages. 09/12/2023.
Saturday.

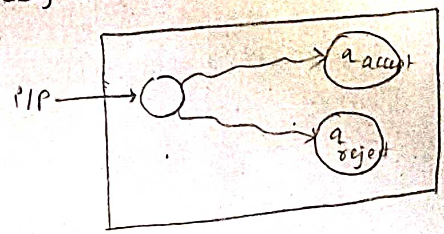
1) Turing Acceptable / Recognizable Language. → also called as Recursive enumerable languages.
2) Turing Decidable Languages. → Recursive Languages.

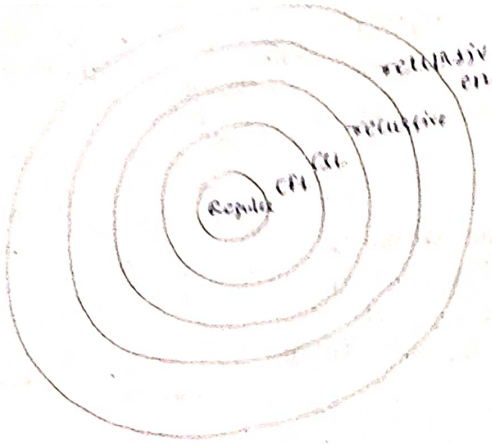
Turing Acceptable languages, A Language is turing acceptable or turing recognisable if there exists a turing machine that haults and accepts L or haults and rejects or loop for an infinite time.

⇒ It may not hault it may loop forever in for some strings.
⇒ loop forever identify చెప్పలేము.
⇒ accept అయ్యే strings & accept అవుతుంది. వాటికి తెలుసు.
⇒ In decidable → both వాటికీ accept ✓ reject ✓



Turing decidable Languages: A language is said to be turing decidable or recursive if there exists a turing machine m such that on input x m accepts if x ∈ L and rejects otherwise.

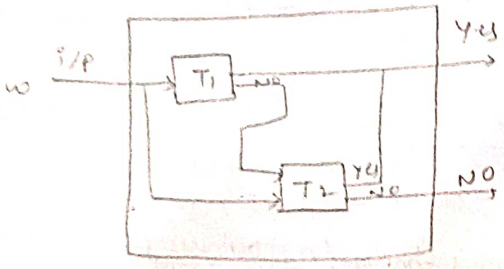




Closure properties

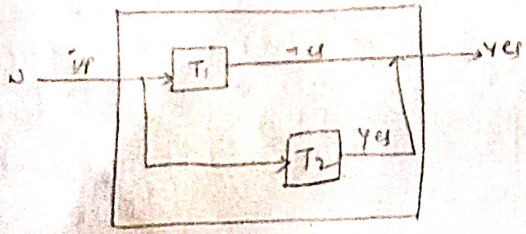
⇒ Union: union of two recursive languages is recursive.

$L_1 \cup L_2$ → is recursive



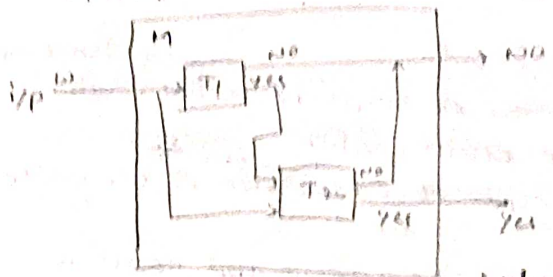
⇒ Union of two recursive enumerable language is recursive enumerable.

Yes తప్పిస్తేనే చెప్పగలము. NO తప్పుకు తెలియదు.



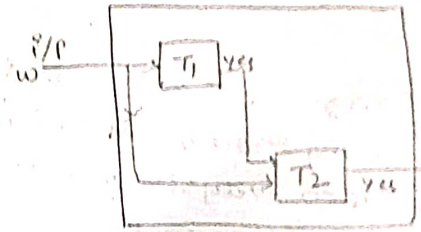
⇒ Intersection of two recursive languages is recursive.

$L_1 \cap L_2$

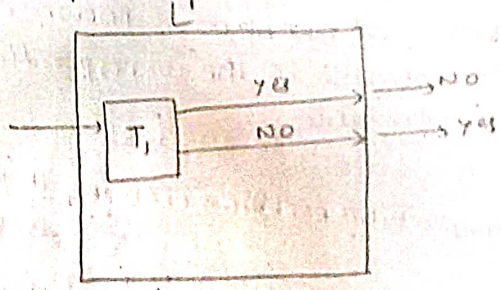


Let T_1, T_2 be the turing machines belong to L_1 and L_2 and M be the machine belongs to $L_1 \cap L_2$

⇒ Intersection of two recursively enumerable languages is recursive enumerable.



⇒ complement of recursive language is recursive.



⇒ complement of recursive enumerable language is neither recursive nor recursive enumerable.

Turing machine does not exist.

⇒ It is undecidable actually.

- every recursive is always recursive enumerable
- If language L and L' are recursive enumerable then we say recursive.

⇒ concatenation of two ... language is recursive enumerable

⇒ concatenation of two recursive languages is recursive

⇒ Kleene closure of recursive enumerable language is recursive enumerable.

⇒ Kleene closure of recursive language is recursive.

⇒ Intersection of recursive and recursive enumerable language is recursive enumerable

Undecidability:

Decidable langu-age: A language L is decidable if there exists a turing machine M such that M accepts and halts on every string in L: rejects and halts on every string in L'.

⇒ A computational problem is known as decidable or solvable if the corresponding language is decidable.

church and Turing: scientists.

NP complete: using another problem → reducing it to known problem.

Recognizable Language: A Language L is called recognizable if there exists a turing machine 'M' such that 'M' accepts either rejects or fails to halts on every string in L'.

Undecidable language: An undecidable language has no decider, there does not exist any turing machine which accepts the language and makes a decision by halting for every input string.

eg: 1. Hamiltonian cycle or not → verifiable
↳ but not decidable

or step by step procedure లేదు decide చేయలేము

2. Unambiguous లేదో చెప్పగల step by step procedure లేదు whether a CFG is ambiguous or not → is undecidable

3. halting problem: Given a turing machine whether it halts or not.

UTM: Universal Turing machine is a machine that can simulate any given turing machine.

also undecidable.

church Turing thesis) halting problem definitions.

Post's correspondance Problem: An instance of PCP (PCP)

consists of two lists of strings over some alphabet which are of equal length let the two lists be A and B written as $A = w_1, w_2, w_3 \ldots w_k$ and $B = x_1, x_2, x_3 \ldots x_k$ for some integer K. For each i the pair (w_i, x_i) is said to be corresponding pair.

we say this instance of pcp has a solution if there is a sequence of one or more integers $i_1, i_2 \ldots i_m$ that when interpreted as indexes for strings in A and B lists yield the same string i.e $w_{i_1} w_{i_2} \ldots w_{i_m} = x_{i_1} x_{i_2} \ldots x_{i_m}$

is
we say the sequence the $i_1\ i_2 \cdots i_m$ is the solution this instance of PCP.

1)

	A	B
i	w_i	x_i
1	110	110110
2	0011	00
3	0110	110

110 0110 0011

110 110 110 00

w_i x_i

2 3 1 $\Rightarrow$ 0011 0110 110 = 0011 0110 110

$\frac{A}{B}$ [0011 / 00] [0110 / 110] [110 / 110]

$\frac{A}{B}$ [0011 / 00] [0110 / 110] [110 / 110] ...

3 2 1

2)

A = {10, 11, 110}

B = {10, 011, 11}.

Dominos method:

[110 / 11] [11 / 011] → 32 ✓ → 1 ✓ → 321 ✓

correct sequence: accept

If not correct sequence: either it ends → rejected or loops → forever.

$\Rightarrow$ PCP is also undecidable.

$\Rightarrow$ PCP is useful for showing undecidability of many other problems by means of reducability.

Reducability:

$\Rightarrow$ Reducing one problem into another problem.

$\Rightarrow$ def: A reduction is a process of converting one problem into another solved problem in such a way that the solution of the second problem can be used to solve the first problem.

$\Rightarrow$ Let the two problems be A and B where B is a solved problem.

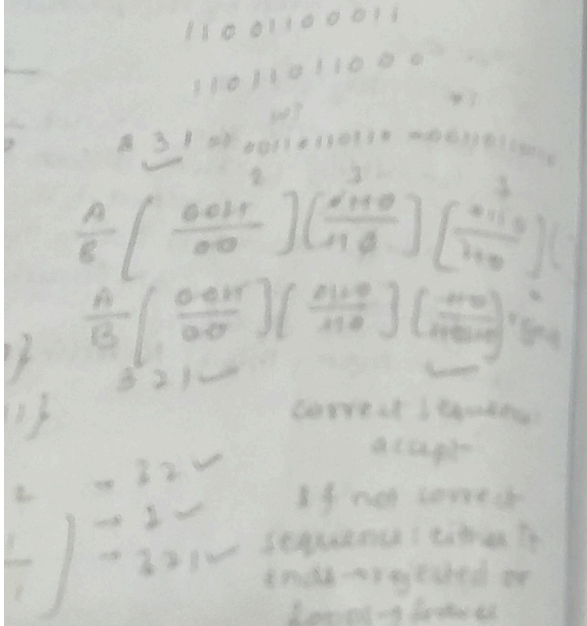
(i) When A is reduced to B, solving A cannot be harder than solving B.

eg: B is polynomial.
A is also cannot be more than polynomial.

(ii) If A is reducable to B, if B is decidable problem then A is also decidable

NOTE: If A is undecidable problem and reducible to B then B is also undecidable.

equence the i_1 i_2 ... i_m is the instance of PCP



correct sequence accept

If not correct sequence either it ends up getting rejected or keeps on looping

able.
owing undecidability of
by means of reducability

nto another problem

on of converting one

NOTE: If A is undecidable problem and reducible to B then B is also undecidable.